IBM WebSphere eXtreme Scale Version 7.1.1
Version 7.1

Présentation du produit

21 novembre 2011

décembre 2011

LE PRESENT DOCUMENT EST LIVRE EN L'ETAT SANS AUCUNE GARANTIE EXPLICITE OU IMPLICITE. IBM DECLINE NOTAMMENT TOUTE RESPONSABILITE RELATIVE A CES INFORMATIONS EN CAS DE CONTREFACON AINSI QU'EN CAS DE DEFAUT D'APTITUDE A L'EXECUTION D'UN TRAVAIL DONNE.

Ce document est mis à jour périodiquement. Chaque nouvelle édition inclut les mises à jour. Les informations qui y sont fournies sont susceptibles d'être modifiées avant que les produits décrits ne deviennent eux-mêmes disponibles. En outre, il peut contenir des informations ou des références concernant certains produits, logiciels ou services non annoncés dans ce pays. Cela ne signifie cependant pas qu'ils y seront annoncés.

Pour plus de détails, pour toute demande d'ordre technique, ou pour obtenir des exemplaires de documents IBM, référez-vous aux documents d'annonce disponibles dans votre pays, ou adressez-vous à votre partenaire commercial.

Vous pouvez également consulter les serveurs Internet suivants :

- http://www.fr.ibm.com (serveur IBM en France)
- http://www.can.ibm.com (serveur IBM au Canada)
- http://www.ibm.com (serveur IBM aux Etats-Unis)

Compagnie IBM France
Direction Qualité
17, avenue de l'Europe
92275 Bois-Colombes Cedex

Table des matières

Figures

Tableaux

Avis aux lecteurs canadiens

Le présent document a été traduit en France. Voici les principales différences et particularités dont vous devez tenir compte.

Illustrations

Les illustrations sont fournies à titre d'exemple. Certaines peuvent contenir des données propres à la France.

Terminologie

La terminologie des titres IBM peut différer d'un pays à l'autre. Reportez-vous au tableau ci-dessous, au besoin.

IBM France	IBM Canada
ingénieur commercial	représentant
agence commerciale	succursale
ingénieur technico-commercial	informaticien
inspecteur	technicien du matériel

Claviers

Les lettres sont disposées différemment : le clavier français est de type AZERTY, et le clavier français-canadien de type QWERTY.

OS/2 et Windows - Paramètres canadiens

Au Canada, on utilise :

- les pages de codes 850 (multilingue) et 863 (français-canadien),
- le code pays 002,
- le code clavier CF.

Nomenclature

Les touches présentées dans le tableau d'équivalence suivant sont libellées différemment selon qu'il s'agit du clavier de la France, du clavier du Canada ou du clavier des États-Unis. Reportez-vous à ce tableau pour faire correspondre les touches françaises figurant dans le présent document aux touches de votre clavier.

France	Canada	Etats-Unis
(Pos1)		Home
Fin	Fin	End
(PgAr)		PgUp
(PgAv)		PgDn
Inser	Inser	Ins
Suppr	Suppr	Del
Echap	Echap	Esc
Attn	Intrp	Break
Impr écran	ImpEc	PrtSc
Verr num	Num	Num Lock
Arrêt défil	Défil	Scroll Lock
(Verr maj)	FixMaj	Caps Lock
AltGr	AltCar	Alt (à droite)

Brevets

Il est possible qu'IBM détienne des brevets ou qu'elle ait déposé des demandes de brevets portant sur certains sujets abordés dans ce document. Le fait qu'IBM vous fournisse le présent document ne signifie pas qu'elle vous accorde un permis d'utilisation de ces brevets. Vous pouvez envoyer, par écrit, vos demandes de renseignements relatives aux permis d'utilisation au directeur général des relations commerciales d'IBM, 3600 Steeles Avenue East, Markham, Ontario, L3R 9Z7.

Assistance téléphonique

Si vous avez besoin d'assistance ou si vous voulez commander du matériel, des logiciels et des publications IBM, contactez IBM direct au 1 800 465-1234.

A propos de la *Présentation du produit*

La documentation de WebSphere eXtreme Scale inclut trois volumes qui fournissent les informations nécessaires pour utiliser, programmer et administrer le produit WebSphere eXtreme Scale.

Bibliothèque WebSphere eXtreme Scale

La bibliothèque WebSphere eXtreme Scale contient les documents suivants :

- La *Présentation du produit* contient une vue de haut niveau des concepts de WebSphere eXtreme Scale, avec des scénarios d'utilisation et des tutoriels.
- Le *Guide d'installation* explique comment installer les topologies communes de WebSphere eXtreme Scale.
- Le *Guide d'administration* contient les informations nécessaires pour les administrateurs système et explique notamment comment planifier les déploiements d'application, planifier la capacité, installer et configurer le produit, démarrer et arrêter des serveurs, surveiller l'environnement et le sécuriser.
- Le *Guide de programmation* contient des informations destinées aux développeurs d'applications, sur la manière de développer des applications pour WebSphere eXtreme Scale à l'aide des informations d'API incluses.

Pour télécharger les documents, accédez à la page de la bibliothèque de WebSphere eXtreme Scale.

Vous pouvez également accéder aux mêmes informations dans cette bibliothèque dans le centre de documentation de WebSphere eXtreme Scale version 7.1.1.

Utilisation hors ligne des manuels

Tous les manuels de la bibliothèque WebSphere eXtreme Scale contiennent des liens vers le centre de documentation, avec l'URL racine `http://publib.boulder.ibm.com/infocenter/wxsinfo/v7r1m1`. Ces liens vous permettent d'accéder directement aux informations associées. Toutefois, si vous travaillez hors ligne et rencontrez l'une de ces liens, vous pouvez rechercher le titre du lien dans les autres manuels dans la bibliothèque. La documentation d'API, le glossaire et les références des messages ne sont pas disponibles dans les manuels PDF.

A qui s'adresse ce document

Ce document est destiné à quiconque souhaite en savoir plus sur WebSphere eXtreme Scale.

Obtention des mises à jour de ce document

Vous pouvez obtenir les mises à jour de ce document en téléchargeant la version la plus récente à partir de la page de la bibliothèque de WebSphere eXtreme Scale.

Comment envoyer vos commentaires

Contactez l'équipe chargée de la documentation. Avez-vous trouvé ce que vous recherchiez ? Ces informations étaient-elles précises et complètes ? Envoyez vos commentaires sur cette documentation par courrier électronique, à l'adresse wasdoc@us.ibm.com.

Chapitre 1. Présentation du produit

WebSphere eXtreme Scale est une grille de données élastique et évolutive, entièrement présente en mémoire. La grille de données met en cache, partitionne, réplique et gère les données et la logique métier sur plusieurs serveurs de manière dynamique. WebSphere eXtreme Scale traite avec une extrême efficacité des transactions en quantités massives tout en pouvant monter en puissance de manière linéaire. WebSphere eXtreme Scale permet également de bénéficier de qualités de services comme l'intégrité transactionnelle, la haute disponibilité et la prévisibilité des temps de réponse.

Présentation générale de WebSphere eXtreme Scale

WebSphere eXtreme Scale est une grille de données élastique et évolutive, entièrement présente en mémoire. La grille de données met en cache, partitionne, réplique et gère les données et la logique métier sur plusieurs serveurs. WebSphere eXtreme Scale traite avec une extrême efficacité des transactions en quantités massives tout en pouvant monter en puissance de manière linéaire. WebSphere eXtreme Scale permet également de bénéficier de qualités de services comme l'intégrité transactionnelle, la haute disponibilité et la prévisibilité des temps de réponse.

WebSphere eXtreme Scale est utilisable de plusieurs manières différentes. Vous pouvez utiliser le produit comme un cache extrêmement puissant, comme espace de traitement de base de données interne pour gérer l'état des applications ou pour créer applications Extreme Transaction Processing (XTP). Ces fonctionnalités XTP incluent une infrastructure d'application pour prendre en charge vos applications stratégiques les plus exigeantes.

Elasticité de l'évolutivité

L'élasticité de l'évolutivité est possible grâce à l'utilisation de la mise en cache répartie des objets. Son extensibilité élastique permet à la grille de se surveiller et de se gérer elle-même. La grille de données peut ajouter ou supprimer des serveurs de la topologie, ce qui augmente ou diminue, la mémoire, le débit réseau et la capacité de traitement en fonction des besoins. Lorsqu'un processus scale-out est lancé, de la capacité est ajoutée à la grille de données lorsqu'elle est en cours d'exécution sans avoir à la redémarrer. Inversement, un scale-in supprimera de la capacité, tout aussi immédiatement. La grille de données se répare elle-même automatiquement en reprenant son activité après des pannes.

WebSphere eXtreme Scale ou base de données interne

WebSphere eXtreme Scale ne peut pas être considéré réellement comme une base de données interne. Une base de données interne est trop simple pour pouvoir gérer certaines des complexités que WebSphere eXtreme Scale peut gérer. Si un serveur d'une base de données interne est défaillant, le serveur ne peut pas résoudre le problème. Un incident peut être désastreux si l'ensemble de votre environnement se trouve sur ce serveur.

Pour résoudre ce problème, eXtreme Scale fractionne le jeu de données concerné en partitions qui sont équivalentes à des schémas arborescents soumis à des contraintes. Ces schémas décrivent les relations entre les entités. Lorsque vous

utilisez des partitions, les relations des entités doivent modéliser une structure de données arborescente. Dans cette structure, la tête de l'arborescence est l'entité racine et la seule entité partitionnée. Toutes les entités enfants de cette entité racine sont stockées dans la même partition que leur entité racine. Chaque partition existe sous la forme d'une copie primaire (fragment). Les partitions contiennent également des fragments de réplique destinés à sauvegarder les données. Une base de données interne ne peut pas fournir cette fonction, car elle n'est ni structurée ni dynamique de cette manière. Avec une base de données interne, vous devez implémenter les opérations que WebSphere eXtreme Scale n'implémente pas automatiquement. Vous pouvez exécuter des opérations SQL sur des bases de données en mémoire pour améliorer la vitesse de traitement par rapport aux bases de données qui ne sont pas en mémoire. WebSphere eXtreme Scale possède son propre langage de requêtes à la place de SQL. Ce langage de requête plus élastique, permet de partitionner les données et fournit une fonction de récupération fiable après incident.

WebSphere eXtreme Scale avec des bases de données

Sa fonctionnalité de cache en écriture différée permet à WebSphere eXtreme Scale de servir de cache frontal à une base de données. L'utilisation de ce cache frontal augmente les débits tout en déchargeant et en libérant la base de données. WebSphere eXtreme Scale fournit une évolutivité croissante et décroissante à des coûts de traitement prévisibles.

L'illustration suivante montre que dans un environnement de cache cohérent et réparti, les clients eXtreme Scale envoient des données à la grille de données et en reçoivent. La grille de données peut être automatiquement synchronisée avec un magasin de données dorsal. Le cache est dit cohérent car les clients y voient tous les mêmes données. Chaque élément de donnée est stocké exactement sur un seul serveur inscriptible dans le cache. L'existence d'une copie de chaque élément de données permet d'éviter la prolifération des copies d'enregistrements qui peuvent contenir des versions différentes des données. Un cache cohérent contient de plus en plus de données au fur et à mesure que l'on ajoute des serveurs à la grille et le cache évolue de manière linéaire au fur et à mesure que la grille croît en taille. Les données peuvent également être répliquées, si l'on souhaite bénéficier de la tolérance aux pannes.

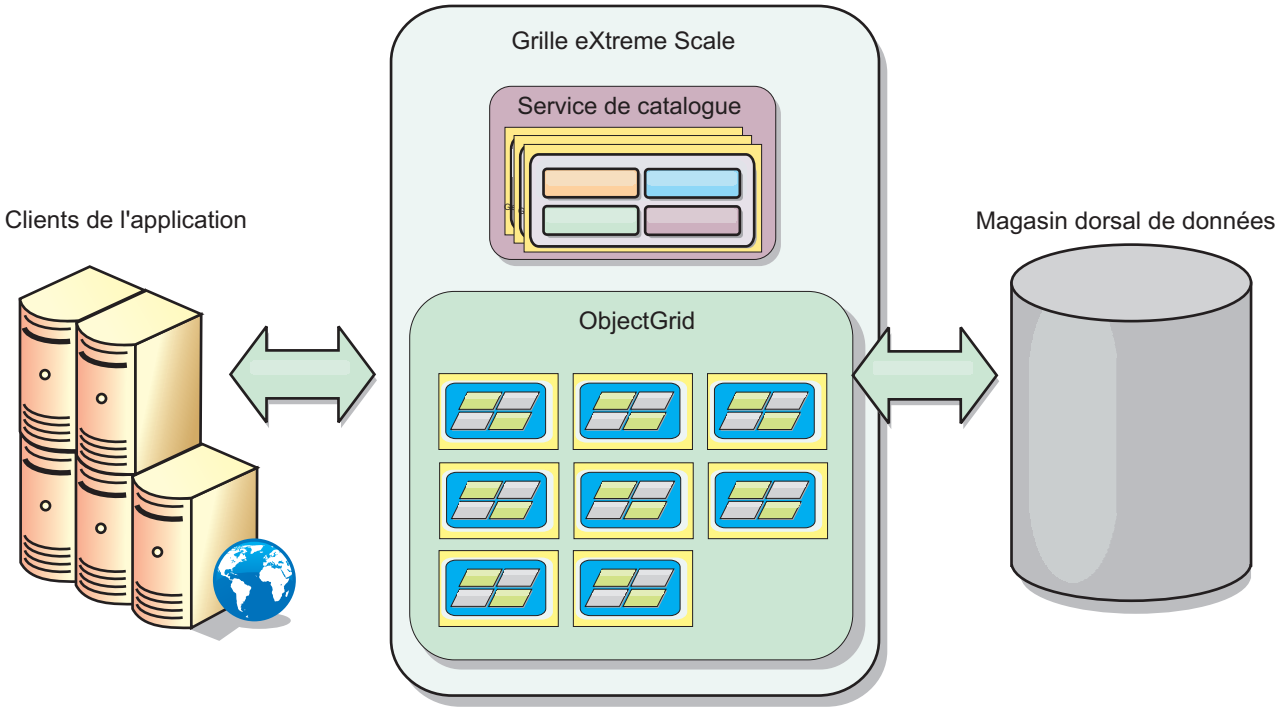


Figure 1. Topologie globale

WebSphere eXtreme Scale utilise des serveurs, appelés *serveurs de conteneur*, qui fournissent sa grille de données en mémoire. Ces serveurs peuvent s'exécuter dans WebSphere Application Server ou sur de simples machines JVM (Java Standard Edition (J2SE) Java). Plusieurs serveurs de conteneur peut s'exécuter sur un seul serveur physique. En conséquence, la grille de données en mémoire peut être volumineuse. La grille n'est pas limitée par la mémoire ou l'espace adresse de l'application ou du serveur d'applications et elle n'a aucun impact sur cette mémoire ou cet espace adresse. La mémoire peut correspondre à la somme de la mémoire de centaines, voire de milliers de machines virtuelles Java exécutées sur de nombreux serveurs physiques différents.

En tant qu'espace de traitement de base de données en mémoire, WebSphere eXtreme Scale peut s'appuyer sur un disque, dans une base de données ou les deux.

Bien que eXtreme Scale fournisse plusieurs API Java, la plupart des cas d'utilisation ne nécessite aucune programmation de la part de l'utilisateur, mais simplement d'exécuter des opérations de configuration et de déploiement dans l'infrastructure WebSphere.

Présentation de la grille de données

L'interface de programmation simple eXtreme Scale est l'interface ObjectMap qui est une simple interface de mappage comprenant une méthode map.put(key,value) pour placer une valeur dans le cache et une méthode map.get(key) pour extraire la valeur.

Le paradigme fondamental d'une grille de données est la paire clé-valeur où la grille stocke des valeurs (des objets Java) en leur associant une clé (un autre objet Java). La clé permet ultérieurement de récupérer la valeur. Dans eXtreme Scale, les mappes sont constituées d'entrées qui ne sont autres que ces paires clé-valeur.

WebSphere eXtreme Scale offre un grand nombre de configurations de grille, allant d'un simple cache local unique à un énorme cache réparti utilisant une multiplicité de machines virtuelles Java ou de serveurs.

Outre les objets Java, il est possible de stocker des objets avec des relations. Il est possible d'utiliser un langage de requêtes semblable à SQL avec des instructions `SELECT... FROM ... WHERE` pour extraire ces objets. Ainsi, un objet Commande sera associé à un objet Client et à plusieurs objets Articles. Dans WebSphere eXtreme Scale, les relations peuvent être de type un à un, un à plusieurs, plusieurs à un ou plusieurs à plusieurs.

WebSphere eXtreme Scale prend également en charge une interface de programmation EntityManager pour le stockage des entités dans le cache. Cette interface de programmation s'apparente aux entités dans Java Enterprise Edition. Les relations entre entités peuvent être détectées automatiquement à partir d'un fichier XML de descripteur d'entité ou d'annotations dans les classes Java. Vous pouvez extraire une entité à partir de la mémoire cache en fonction de la clé primaire à l'aide de la méthode find de l'interface EntityManager. Les entités peuvent être conservées dans la grille de données ou être retirées de cette dernière, le tout au sein d'une limite de transaction.

Prenons un exemple de cache réparti où la clé est un simple nom alphabétique. Le cache pourra être fractionné en quatre partitions en fonction des clés : partition 1 pour les clés qui commencent par A-E, partition 2 pour celles qui commencent par F-L, etc. Pour garantir la disponibilité, une partition dispose d'un fragment principal et d'un fragment de réplique. Les modifications apportées aux données du cache sont reportées dans le fragment primaire et répliquées dans le fragment de réplique. Vous configurez le nombre de serveurs qui contiennent les données de grille de données et eXtreme Scale répartit les données dans des fragments sur ces instances de serveurs. Pour assurer la disponibilité, les fragments de réplique sont placés sur des serveurs physiques différents des fragments primaires.

WebSphere eXtreme Scale utilise un service de catalogue pour repérer le fragment primaire de chaque clé. Il gère le transfert des fragments entre les serveurs eXtreme Scale lorsque les serveurs physiques sont défaillants et redeviennent fonctionnels. Si, par exemple, le serveur contenant un fragment réplique tombe en panne, eXtreme Scale allouera un nouveau fragment réplique. Si un serveur contenant un fragment primaire est défaillant, le fragment de réplique devient le fragment primaire. Comme précédemment, un nouveau fragment de réplique est alors construit.

Nouveautés de la version 7.1.1

WebSphere eXtreme Scale 7.1.1 contient de nombreuses nouvelles fonctions. Consultez cette rubrique pour en savoir plus sur les dernières mises à jour du produit.

Plug-ins DataSerializer

Lorsque les clients et les serveurs échangent des informations ou que les serveurs répliquent des données d'un serveur vers un autre, les données doivent être converties, ou sérialisés, de sorte qu'elles puissent être transmises sur le réseau. Dans les versions précédentes, vous pouviez utiliser la sérialisation Java par défaut ou le plug-in ObjectTransformer pour sérialiser les données. Dans cette version, vous pouvez utiliser des plug-ins DataSerializer pour décrire précisément le format de sérialisation, ou tableau d'octets, pour WebSphere eXtreme Scale pour que le

produit puisse interagir avec le tableau d'octets sans utiliser un format d'objet spécifique. Pour en savoir plus...

Infrastructure OSGi

En utilisant l'infrastructure OSGi, vous pouvez exposer les plug-ins comme services OSGi pour qu'ils puissent être utilisés par l'environnement d'exécution eXtreme Scale. En outre, vous pouvez démarrer les serveurs et les clients eXtreme Scale dans un conteneur OSGi, ce qui permet d'ajouter ou de mettre à jour dynamiquement les plug-ins eXtreme Scale dans l'environnement d'exécution. Pour en savoir plus...

Amélioration des performances du fournisseur de cache dynamique

Le traitement de l'invalidation dans le fournisseur de cache dynamique WebSphere eXtreme Scale a été amélioré. Les demandes d'invalidation sont traitées de manière asynchrone et par lots lorsque le paramètre **wait** de la méthode invalidate(key, wait) a la valeur `false`. Cette extension améliore considérablement les performances. Pour en savoir plus...

Modification du comportement du placement par défaut

Dans les éditions précédentes, lorsqu'un nouveau serveur de conteneur démarrait dans la grille de données, le placement des fragments sur ce serveur de conteneur commençait immédiatement. Ce placement immédiat sollicite davantage le processeur des serveurs qui contiennent les nouveaux serveurs de conteneur. Le comportement par défaut a été changé pour définir un délai de 15 000 ms, soit 15 secondes, avant l'exécution du placement. Vous pouvez modifier la fréquence de placement avec la propriété **placementDeferralInterval** du serveur. Pour en savoir plus...

Topologie intra-domaine pour les configurations de plug-in de cache niveau 2 (L2) JPA (Java Persistence API)

En configurant une topologie intra-domaine dans le cache L2 JPA, un fragment primaire est placé sur chaque serveur de conteneur dans la configuration. Chaque fragment primaire contient la totalité du contenu de la partition. Avec cette configuration, vous pouvez augmenter les performances, car les clients peuvent accéder aux données en local et tous les fragments primaires peuvent écrire dans la grille de données. Pour en savoir plus...

Utilitaire `xscmd`

L'utilitaire **xscmd** est la nouvelle version prise en charge de l'utilitaire **xsadmin** L'utilitaire **xsadmin** a été inclus comme exemple non pris en charge dans les versions précédentes. Pour en savoir plus...

Outil de génération de rapports d'analyse de journal

Avec le nouvel outil **xsloganalyzer**, vous pouvez générer des rapports à partir de vos fichiers journaux qui peuvent vous aider à analyser les performances de votre environnement et résoudre les problèmes. Pour en savoir plus...

IBM eXtremeIO et IBM eXtremeMemory

En configurant eXtremeMemory, vous pouvez stocker des objets dans la mémoire native plutôt que dans le segment de mémoire Java. La configuration eXtremeMemory active eXtremeIO, un nouveau mécanisme de transport. En retirant des objets du segment de mémoire Java, vous pouvez éviter les pauses de récupération d'espace, ce qui permet de bénéficier de performances plus constantes et de temps de réponse plus prévisibles. Pour en savoir plus...

Support WebSphere Application Server Version 8

Désormais, WebSphere eXtreme Scale Version 7.1.1 peut être installé sur et WebSphere Application Server et WebSphere Application Server Network Deployment Version 8. Pour en savoir plus...

Notes sur l'édition

Des liens permettent d'accéder au site Web de support technique, à la documentation, aux dernières mises à jour, aux restrictions et aux incidents recensés du produit.

- «Dernières mises à jour, limitations et problèmes connus»
- «Accès à la configuration système et logicielle requise»
- «Accès à la documentation du produit»
- «Accès au site de support technique du produit», à la page 7
- «Contacter le service de support logiciel IBM», à la page 7

Dernières mises à jour, limitations et problèmes connus

Les notes sur l'édition sont disponibles sur le site de support technique du produit sous forme de notes techniques. Pour afficher une liste de toutes les notes techniques de WebSphere eXtreme Scale, accédez à la page Web du support technique. Cliquer sur les liens indiqués ici générera une recherche dans la page Web du support des notes sur l'édition correspondantes, lesquelles seront retournées sous forme de liste.

- 7.1.1+ Pour consulter la liste des notes sur l'édition de la version 7.1.1, allez à la page Web du support.
- Pour voir la liste des notes sur l'édition de la version 7.1, allez à la page Web du support.
- Pour voir la liste des notes sur l'édition de la version 7.0, allez à la page Web du support.
- Pour afficher la liste des notes sur l'édition de la version 6.1, accédez à la page du wiki des notes sur l'édition.

Accès à la configuration système et logicielle requise

La configuration matérielle et logicielle requise est détaillée dans les pages suivantes :

- Configuration requise détaillée

Accès à la documentation du produit

Pour accéder à l'ensemble des informations, accédez à la page Bibliothèque.

Accès au site de support technique du produit

Pour rechercher les informations de support technique et notamment les dernières notes techniques, les fichiers à télécharger et les correctifs, accédez au portail du support.

Contacter le service de support logiciel IBM®

Si un incident survient lors de l'utilisation du produit, essayez tout d'abord d'effectuer les opérations suivantes :

- Suivez les étapes décrites dans la documentation du produit
- Recherchez la documentation connexe dans l'aide en ligne
- Recherchez les messages d'erreur dans le document de référence des messages

Si vous ne parvenez pas à résoudre l'erreur à l'aide d'une des méthodes précédentes, prenez contact avec le support technique IBM.

Configurations matérielle et logicielle requises

Vue d'ensemble des conditions requises en termes de matériels et de systèmes d'exploitation. Bien que vous ne soyez pas tenu d'utiliser un niveau spécifique de matériel ou de système d'exploitation pour WebSphere eXtreme Scale, nous n'en fournissons pas moins sur le site de support du produit (page Configuration requise) une liste détaillée des matériels et logiciels officiellement pris en charge. En cas de conflit entre les informations présentées par le Centre de documentation et celles figurant sur cette page, les informations fournies par le site Web prévalent. Les conditions préalables répertoriées par le Centre de documentation sont fournies à titre informatif uniquement.

Voir la page System Configurations requises pour connaître les configurations matérielles et logicielles officielles.

Vous n'êtes pas obligé d'installer et de déployer eXtreme Scale sur un niveau de système d'exploitation spécifique. Chaque installation Java Platform, Standard Edition (Java SE) et Java Platform, Enterprise Edition (Java EE) requiert des niveaux de système d'exploitation ou des correctifs différents.

Vous pouvez installer et déployer le produit dans les environnements Java EE et Java SE. Vous pouvez également regrouper le composant client avec les applications Java EE directement sans les intégrer à WebSphere Application Server. WebSphere eXtreme Scale prend en charge Java SE 5 et les versions suivantes et WebSphere Application Server Version 6.1 et les versions suivantes.

Configuration matérielle

WebSphere eXtreme Scale ne requiert pas la présence d'un niveau spécifique de matériel. La configuration matérielle requise dépend du matériel pris en charge pour l'installation de Java Platform, Standard Edition que vous utilisez pour exécuter WebSphere eXtreme Scale. Si vous utilisez eXtreme Scale avec WebSphere Application Server ou une autre implémentation Java Platform, Enterprise Edition, la configuration matérielle requise par ces plateformes est suffisante pour WebSphere eXtreme Scale.

Configuration requise en matière de système d'exploitation

- **Sans la console Web**

eXtreme Scale ne requiert pas la présence d'un système d'exploitation d'un niveau donné. Chaque implémentation Java SE et Java EE requiert un niveau différent du système d'exploitation ou des correctifs pour les problèmes identifiés lors du test de l'implémentation Java. Les niveaux nécessaires à ces implémentations sont suffisants pour eXtreme Scale.

- **Avec la console Web**

  Les conditions suivantes s'appliquent pour chaque système d'exploitation si vous utilisez la console :

  – Linux : JMV 32 bits ou 64 bits
  – Linux PPC : JVM 32 bits uniquement
  – Windows : JVM 32 bits uniquement
  – AIX : JVM 32 bits uniquement

Navigateurs Web requis

La console Web prend en charge les navigateurs Web suivants :

- Mozilla Firefox, version 3.5.x et versions ultérieures
- Mozilla Firefox, version 3.6.x et versions ultérieures
- Microsoft Internet Explorer version 7 ou 8

Configuration requise pour WebSphere Application Server

- WebSphere Application Server Version 6.1.0.39 ou version suivante
- WebSphere Application Server Version 7.0.0.19 ou version suivante
- WebSphere Application Server Version 8.0.0.1 ou version suivante

Pour plus d'informations, consultez la section Recommended fixes for WebSphere Application Server.

Autres conditions requises par le serveur d'applications

Les autres implémentations Java EE peuvent utiliser la phase d'exécution d'eXtreme Scale en tant qu'instance locale ou client pour les serveurs eXtreme Scale. Pour implémenter Java SE, vous devez utiliser la version 5 ou une version suivante.

Conventions relatives aux répertoires

Les conventions de répertoire suivantes sont utilisées dans toute la documentation pour faire référence à des répertoires spéciaux, tels que *wxs_install_root* et *wxs_home*. Vous pouvez accéder à ces répertoires pendant plusieurs scénarios différents, y compris lors de l'installation et de l'utilisation des outils de ligne de commande.

racine_install_wxs
Le répertoire *wxs_install_root* est le répertoire racine où sont installés les fichiers du produit WebSphere eXtreme Scale. Le répertoire *wxs_install_root* peut être le répertoire dans lequel l'archive d'évaluation est extraite ou depuis lequel le produit est installé WebSphere eXtreme Scale.

- Exemple où la version d'essai a été extraite :

  Exemple : `/opt/IBM/WebSphere/eXtremeScale`

- Exemple où WebSphere eXtreme Scale est installé dans un répertoire autonome :

  Exemple : `/opt/IBM/eXtremeScale`

- Exemple lorsque WebSphere eXtreme Scale est intégré à WebSphere Application Server :

  Exemple : `/opt/IBM/WebSphere/AppServer`

wxs_home
Le répertoire *wxs_home* est le répertoire racine du produit, des bibliothèques, des exemples et des composants WebSphere eXtreme Scale. Ce répertoire est identique au répertoire *wxs_install_root* lorsque l'archive d'évaluation est extraite. Pour les installations autonomes, le répertoire *wxs_home* est le sous-répertoire `ObjectGrid` du répertoire *wxs_install_root*. Pour les installations qui sont intégrées à WebSphere Application Server, ce répertoire est le répertoire `optionalLibraries/ObjectGrid` du répertoire *wxs_install_root*.

- Exemple lorsque la version d'essai a été extraite :

  Exemple : `/opt/IBM/WebSphere/eXtremeScale`
- Exemple lorsque WebSphere eXtreme Scale est installé dans un répertoire autonome :

  Exemple : `/opt/IBM/eXtremeScale/ObjectGrid`
- Exemple lorsque WebSphere eXtreme Scale est intégré à WebSphere Application Server :

  Exemple : `/opt/IBM/WebSphere/AppServer/optionalLibraries/ObjectGrid`

was_root
Le répertoire *was_root* est le répertoire racine d'une installation WebSphere Application Server :

Exemple : `/opt/IBM/WebSphere/AppServer`

restservice_home
Le répertoire *restservice_home* est le répertoire dans lequel se trouvent les bibliothèques et les exemples du service de données REST d'WebSphere eXtreme Scale. Ce répertoire s'appelle `restservice` et il est le sous-répertoire de *wxs_home*.

- Exemple pour les déploiements autonomes :

  Exemple : `/opt/IBM/WebSphere/eXtremeScale/ObjectGrid/restservice`
- Exemple pour les déploiements intégrés à WebSphere Application Server :

  Exemple : `/opt/IBM/WebSphere/AppServer/optionalLibraries/ObjectGrid/restservice`

tomcat_root
Le répertoire *home_tomcat* est le répertoire racine de l'installation d'Apache Tomcat.

Exemple : `/opt/tomcat5.5`

wasce_root
wasce_root est le répertoire racine de l'installation WebSphere Application Server Community Edition.

Exemple : `/opt/IBM/WebSphere/AppServerCE`

java_home
Le répertoire *java_home* est le répertoire racine d'une installation de Java Runtime Environment Kit (JRE).

Exemple : `/opt/IBM/WebSphere/eXtremeScale/java`

samples_home
samples_home est le répertoire dans lequel vous extrayez les exemples de fichiers qui sont utilisés pour les tutoriels.

Exemple : `/wxs-samples/`

dvd_root
dvd_root est le répertoire racine du DVD qui contient le produit.

Exemple : `dvd_root/docs/`

equinox_root
Le répertoire *equinox_root* est le répertoire racine de l'installation de l'infrastructure OSGi Eclipse Equinox.

Exemple :`/opt/equinox`

user_home
Le répertoire *user_home* est l'emplacement de stockage des fichiers utilisateur, tels que les profils de sécurité.

Windows `c:\Documents and Settings\user_name`

UNIX `/home/user_name`

Présentation technique de WebSphere eXtreme Scale

WebSphere eXtreme Scale est une grille de données élastique et évolutive, entièrement présente en mémoire. De manière dynamique, cette grille de données met en cache, partitionne, réplique et gère les données et les logiques applicatives sur une multiplicité de serveurs.

Etant donné que WebSphere eXtreme Scale n'est pas une base de données interne, vous devez tenir compte d'exigences de configuration spécifiques. La première phase du déploiement d'une grille de données consiste à démarrer un groupe central et un service de catalogue qui agit en tant que coordinateur de toutes les autres machines virtuelles Java qui font partie de la grille de données et à gérer les informations de configuration. Le démarrage des processus WebSphere eXtreme Scale s'effectue à l'aide de simples scripts de commandes appelés depuis la ligne de commande.

L'étape suivante consiste à démarrer les processus serveur WebSphere eXtreme Scale pour la grille de données permettant de stocker et d'extraire les données. Lorsque les serveur démarrent, ils s'enregistrent automatiquement auprès du groupe central et du service de catalogue pour leur permettre de coopérer en fournissant des services de grille de données. Un plus grand nombre de serveurs augmente la capacité et la fiabilité de la grille de données.

Une grille de données locale est une simple, une grille à instance unique dans laquelle toutes les données se trouvent dans la grille. Pour utiliser efficacement WebSphere eXtreme Scale comme espace de traitement de la base de données interne, vous pouvez configurer et déployer une grille de données répartie. Dans une grille répartie, les données sont réparties entre les divers serveurs eXtreme Scale qui les contiennent de manière à ce que chaque serveur n'en contienne qu'une partie, appelée précisément partition.

Un paramètre de configuration de grille de données répartie par clé correspond au nombre de paramètres dans la grille. Les données de la grille sont partitionnées dans ce nombre de sous-ensembles, chaque sous-ensemble étant appelé partition.

Le service de catalogue se charge de repérer en fonction de sa clé la partition correspondant à une donnée particulière. Le nombre de partitions affecte directement la capacité et l'extensibilité de la grille de données. Un serveur peut contenir une ou plusieurs partitions de données de la grille. De ce fait, la taille des partitions est limitée par l'espace mémoire du serveur. Inversement, augmenter le nombre de partitions augmente la capacité de la grille de données. La capacité maximale d'une grille de données correspond au nombre de partitions multiplié par la taille de la mémoire utilisable de chaque serveur. Un serveur peut être une machine virtuelle Java, mais vous pouvez définir le serveur eXtreme Scale pour l'adapter à votre environnement de déploiement.

Les données d'une partition sont stockées dans un fragment. Pour la disponibilité, une grille de données peut être configurée avec des répliques qui peuvent être synchrones ou asynchrones. Les modifications apportées aux données de la grille sont effectuées dans le fragment primaire et répliquées vers les fragments secondaires. La mémoire totale qui est utilisée ou requise par une grille de données est donc égale à la taille de la grille multipliée par (1 (pour le fragment primaire) + le nombre de répliques).

WebSphere eXtreme Scale distribue les fragments d'une grille de données entre le nombre de serveurs de la grille. Ces serveurs peuvent se trouver sur le même serveur ou des serveurs différents. Pour que la disponibilité, les fragments de réplique sont placés sur des serveurs physiques distincts à partir de fragments primaires.

WebSphere eXtreme Scale surveille le statut de ses serveurs et déplace les fragments en cas de défaillance ou de reprise des serveurs de fragments ou physiques. Par exemple, si le serveur contenant un fragment de réplique est défaillant, WebSphere eXtreme Scale alloue un nouveau fragment de réplique et réplique les données du fragment primaire vers le nouveau fragment de réplique. Si un serveur qui contient un fragment primaire est défaillant, le fragment de réplique devient le fragment primaire et le nouveau fragment de réplique est construit. Si vous démarrez un serveur supplémentaire pour la grille de données, les fragments sont équilibrés sur tous les serveurs. Ce rééquilibrage est appelé scale-out. De même, pour le scale-in, vous pouvez arrêter l'un des serveurs pour réduire les ressources utilisées par une grille de données. Par conséquent, les fragments sont équilibrés sur les serveurs restants.

Mise en cache : présentation d'ensemble

WebSphere eXtreme Scale peut fonctionner comme un espace de traitement de la base de données en mémoire offrant une fonction de mise en cache en ligne pour une base de données dorsale ou servant de cache secondaire. La mise en cache en ligne utilise eXtreme Scale comme moyen principal d'interaction avec les données. Lorsqu'eXtreme Scale est utilisé en tant que cache secondaire, le système dorsal est utilisé conjointement avec la grille de données. Cette section décrit divers concepts et scénarios de mise en cache et compare les différentes topologies utilisables pour le déploiement d'une grille de données.

Architecture de la mise en cache : mappes, conteneurs, clients et catalogues

Avec WebSphere eXtreme Scale, l'architecture de votre système peut utiliser la mise en cache des données locales en mémoire ou la mise en cache des données client-serveur réparties.

WebSphere eXtreme Scale requiert une infrastructure supplémentaire minimale pour pouvoir fonctionner. Cette infrastructure consiste en des scripts permettant d'installer, de démarrer et d'arrêter une application Java Platform, Enterprise Edition sur un serveur. Les données mises en cache sont stockées dans le serveur eXtreme Scale et les clients se connectent à distance à ce serveur.

Les caches répartis permettent d'améliorer les performances, la disponibilité et l'évolutivité du système. Les topologies dynamiques utilisées pour les configurer permettent d'équilibrer automatiquement les serveurs. Vous pouvez également ajouter d'autres serveurs sans redémarrer les serveurs eXtreme Scale existants. Il est possible de créer des déploiements simples ou des déploiements plus vastes se chiffrant en téraoctets et comptant plusieurs milliers de serveurs.

Service de catalogue

Le service de catalogue héberge une logique qui doit être inactive lorsque l'état est stabilisé et qui a peu d'influence sur l'évolutivité. Le service de catalogue est généré pour gérer plusieurs centaines de conteneurs devenant disponibles simultanément. Il exécute des services en vue de leur gestion.

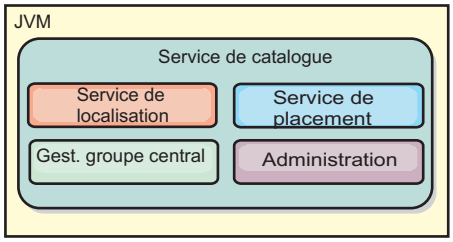


Figure 2. Service de catalogue

Les responsabilités du catalogue de service sont les suivantes :

Service de localisation
Le service de localisation fournit une localité aux clients qui recherchent des conteneurs hébergeant des applications et aux conteneurs cherchant à enregistrer des applications hébergées à l'aide du service de positionnement. Il s'exécute dans tous les membres de la grille et permet de supprimer cette fonction.

Service de positionnement
Le service de positionnement constitue le système nerveux central de la grille. Il est responsable de l'allocation des fragments à leur conteneur hôte. Il s'exécute en tant que service Un sur N choisi dans le cluster. Comme c'est la règle Un sur N qui est utilisée, il y a toujours une et une seule instance de ce service en cours d'exécution. Si cette instance doit s'arrêter, un autre processus prend la relève. A des fins de redondance, tous les états du service de catalogue sont répliqués sur tous les serveurs hébergeant le service de catalogue.

Gestionnaire du groupe central
Le gestionnaire du groupe central gère le regroupement homologue en vue de la surveillance de la santé, organise les conteneurs en petits groupes de serveurs et fédère automatiquement les groupes de serveurs. Lorsqu'un conteneur contacte le service de catalogue pour la première fois, le conteneur attend d'être affecté à un nouveau groupe ou à un groupe existant de plusieurs machines virtuelles Java. Chaque groupe de machines virtuelles Java surveille la disponibilité de chacun de ses membres à l'aide

du signal de présence. L'un des membres du groupe relaie les informations sur la disponibilité au service de catalogue afin de lui permettre de réagir aux défaillances en procédant à des réallocations et à des réacheminements.

Administration
L'administration de l'environnement WebSphere eXtreme Scale se compose de quatre phases : la planification, le déploiement, la gestion et la surveillance.

Pour la disponibilité, configurez un domaine de services de catalogue. Un domaine de services de catalogue consiste en plusieurs machines virtuelles Java, dont une machine maîtresse et plusieurs machines virtuelles Java de sauvegarde.

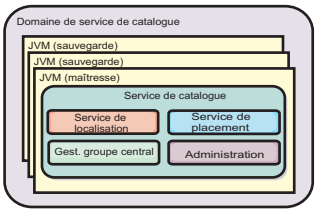


Figure 3. Domaine de services de catalogue

Serveurs de conteneur, partitions et fragments

Le serveur de conteneur stocke les données d'application pour la grille de données. Ces données sont généralement divisées en fractions appelées partitions qui sont hébergés sur plusieurs serveurs de conteneur. Chaque serveur de conteneur à son tour héberge un sous-ensemble de l'ensemble des données. Une machine virtuelle Java peut héberger un ou plusieurs serveurs de conteneur et chaque serveur de conteneur peut héberger plusieurs fragments.

A faire : Planifiez la taille des segments de mémoire des serveurs de conteneur qui hébergent l'ensemble de vos données. Configurez en conséquence les paramètres du segment de mémoire.

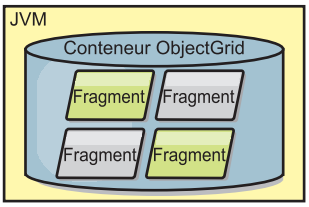


Figure 4. Serveur de conteneur

Les partitions hébergent un sous-ensemble des données dans la grille. WebSphere eXtreme Scale place automatiquement plusieurs partitions dans un serveur de conteneur unique et répartit les partitions à mesure que le nombre de serveurs de conteneur disponible augmente.

Important : Choisissez soigneusement le nombre de partitions avant le déploiement final, car ce nombre ne peut pas être modifié dynamiquement. Un mécanisme de hachage permet de localiser les partitions dans le réseau et eXtreme Scale ne peut pas hacher à nouveau l'ensemble des données après leur déploiement. En règle générale, vous pouvez surestimer le nombre de partitions

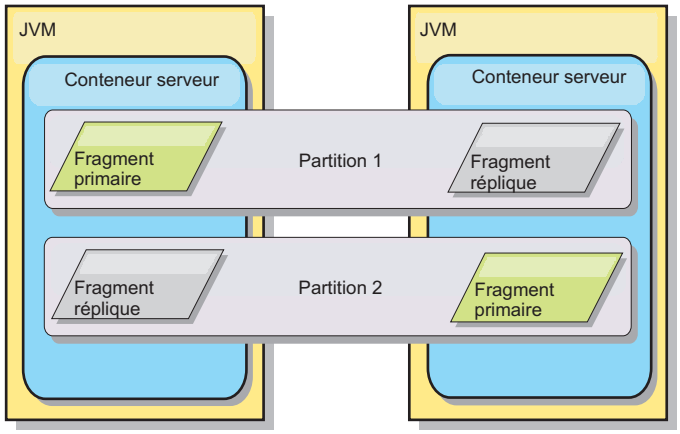


Figure 5. Partition

Les fragments sont des instances de partitions. Ils peuvent avoir un rôle primaire ou un rôle de réplique. Le fragment primaire et ses fragments réplique constituent la manifestation physique de la partition. Chaque partition contient plusieurs fragments, dont chacun héberge toutes les données contenues dans celle-ci. L'un des fragments est le fragment primaire, les autres sont les fragments réplique, c'est-à-dire des copies redondantes des données du fragment primaire. Le fragment primaire est la seule instance de partition permettant à des transactions d'écrire dans le cache. Un fragment réplique est une instance "miroir" de la partition. Il reçoit des mises à jour du fragment primaire de manière synchrone ou asynchrone. Le fragment réplique autorise uniquement les transactions à lire à partir du cache. Les répliques ne sont jamais hébergées dans le même serveur de conteneur que le fragment primaire et ne sont normalement pas hébergées sur la même machine que le fragment primaire.

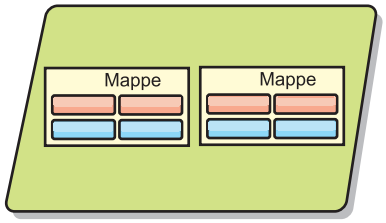


Figure 6. Fragment

Pour améliorer la disponibilité des données ou garantir la persistance, répliquez les données. La réplication augmente toutefois le coût des transactions et améliore les performances au détriment de la disponibilité. Avec eXtreme Scale, vous pouvez contrôler les coûts car les réplications synchrones et asynchrones sont prises en charge, ainsi que les modèles de réplication hybrides utilisant les deux modes de réplication. Un fragment réplique synchrone reçoit des mises à jour lors de la transaction du fragment primaire visant à garantir la cohérence des données. Un fragment réplique synchrone peut doubler le temps de réponse car la transaction doit valider le fragment primaire et le fragment réplique synchrone avant que la transaction se termine. Un fragment réplique asynchrone reçoit des mises à jour après que la transaction a validé la limitation de l'impact sur les performances, mais introduit la possibilité de perte de données car les fragments réplique asynchrones peuvent impliquer le traitement de plusieurs transactions après le fragment primaire.

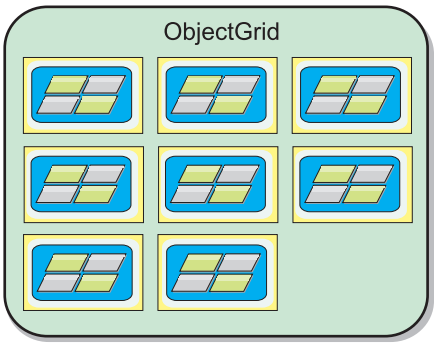


Figure 7. ObjectGrid

Mappes

Une mappe est un conteneur de paires clé-valeur permettant à une application de stocker une valeur indexée par une clé. Les mappes prennent en charge les index pouvant être ajoutés pour indexer les attributs sur la clé ou sur la valeur. Ces index sont automatiquement utilisés par l'environnement d'exécution des requêtes pour déterminer le mode d'exécution des requêtes le plus efficace.

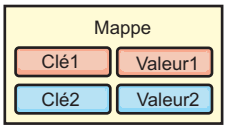


Figure 8. Mappe

Un groupe de mappes est une collection de mappes partageant un même algorithme de partitionnement. Les données contenues dans les mappes sont répliquées en fonction des règles définies par le groupe de mappes. Les groupes de mappes sont uniquement utilisés pour les topologies réparties. Pour les topologies locales, ils ne sont pas nécessaires.

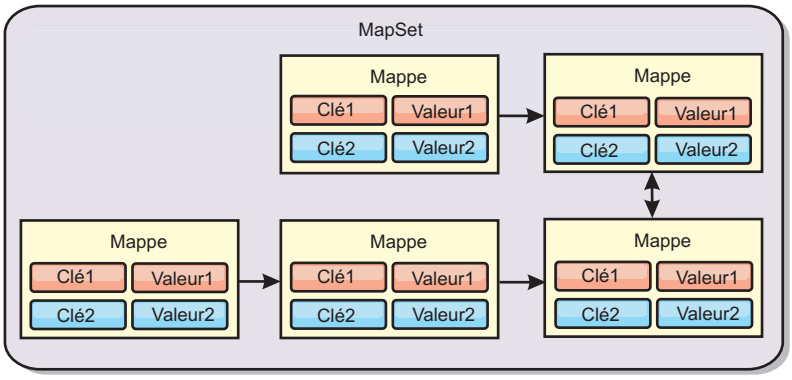


Figure 9. Groupes de mappes

Un groupe de mappes peut être associé à un schéma. Un schéma est l'ensemble des métadonnées décrivant les relations entre différentes mappes lorsque des types d'objet ou des entités hétérogènes sont utilisés.

WebSphere eXtreme Scale peut stocker des objets Java sérialisables dans chacune des mappes utilisant l'API ObjectMap. Il est possible de définir un schéma sur les mappes pour identifier la relation entre les objets dans les mappes contenant des objets d'un seul type. Il est nécessaire de définir un schéma pour pouvoir interroger le contenu des objets de la mappe. Plusieurs schémas de mappes peuvent être définis pour WebSphere eXtreme Scale.

WebSphere eXtreme Scale peut également stocker des entités à l'aide de l'API EntityManager. Chaque entité est associée à une mappe. Le schéma d'un groupe de mappes d'entités est automatiquement reconnu à l'aide d'un fichier XML descripteur d'entité ou de classes Java annotées. Chaque entité est associée à un ensemble d'attributs clés et à un ensemble d'attributs non clés. Des relations peuvent aussi exister entre une entité et d'autres entités. WebSphere eXtreme Scale prend en charge les relations one-to-one, one-to-many, many-to-one et many-to-many. Chaque entité est physiquement mappée vers une seule mappe du groupe de mappes. Grâce aux entités, la présence de graphes d'objets complexes s'étendant sur plusieurs mappes est possible dans les applications. Une topologie répartie permet la coexistence de plusieurs schémas d'entités.

Pour plus d'informations, voir Mise en cache d'objets et de leurs relations (API EntityManager).

Clients

Les clients se connectent à un service de catalogue, extraient une description de la topologie du serveur et communiquent directement avec chaque serveur. Lorsque la topologie du serveur est modifiée en raison de l'ajout de nouveaux serveurs ou de la défaillance de certains serveurs existants, le service de catalogue dynamique achemine le client vers le serveur hébergeant les données approprié. Les clients doivent examiner les clés des données d'application pour déterminer vers quelle partition la demande doit être acheminée. Les clients peuvent lire les données de plusieurs partitions dans une même transaction. Ils peuvent cependant mettre à jour une seule partition dans une transaction. Une fois que le client a mis à jour certaines entrées, la transaction client doit utiliser cette partition pour les mises à jour.

La liste suivante répertorie les combinaisons de déploiement possibles :

- Un service de catalogue existe dans sa propre grille de machines virtuelles Java. Le même service de catalogue peut être utilisé pour gérer plusieurs clients ou serveurs eXtreme Scale.
- Un conteneur peut être démarré dans une JVM ou chargé dans une JVM arbitraire avec d'autres conteneurs destinés à des instances ObjectGrid différentes.
- Un client peut exister dans une JVM et communiquer avec une ou plusieurs instances ObjectGrid. Un client peut également exister dans la même JVM que le conteneur.

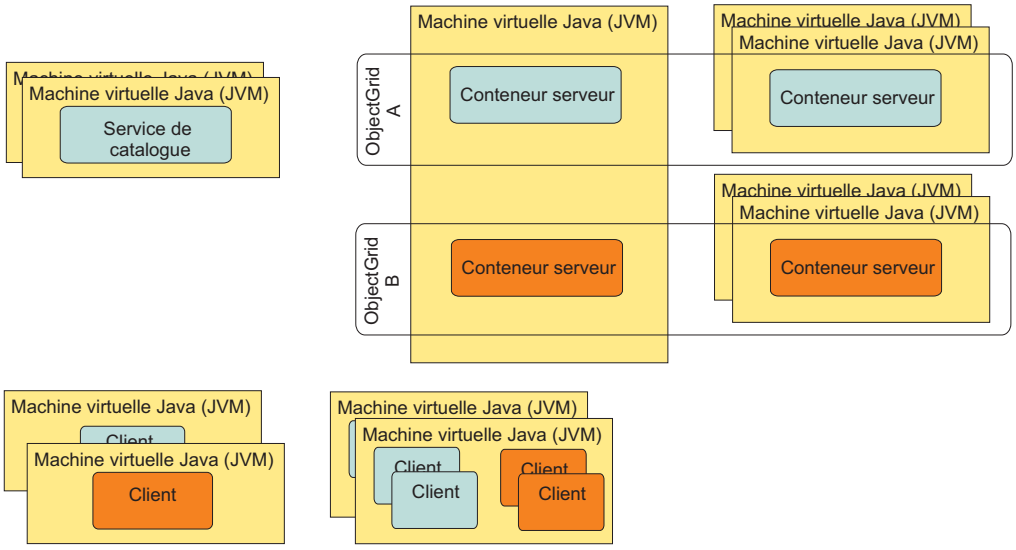


Figure 10. Topologies possibles

Zones

Les zones vous permettent de contrôler le placement des fragments. Les zones sont des regroupements de serveurs physiques, définis par l'utilisateur. Exemples des différents types de zones : serveurs lames différents, châssis de serveurs lames, étages d'un bâtiment, bâtiments ou différents emplacements géographiques dans un environnement à plusieurs centres de données. Un environnement virtualisé dans lequel un grand nombre d'instances de serveur, ayant une adresse IP unique, s'exécutent sur un même serveur physique est un autre cas d'utilisation.

Zones définies entre les centres de données

L'exemple de cas d'utilisation classique des zones implique aux moins deux centres de données qui se trouvent dans des lieux géographiques différents. Les centres de données dispersés étendent la grille de données à différents emplacements pour la reprise d'un centre de données défaillant. Par exemple, vous souhaitez vous assurer que vous disposez d'un ensemble complet de fragments de réplique asynchrones pour votre grille de données dans un centre de données distant. Avec cette stratégie, vous pouvez restaurer le centre de données local détaillant en toute transparence, sans perte de données. Les centres de données eux-mêmes utilisent des réseaux haut débit à faible latence. Toutefois, la communication entre un centre de données et un autre a une latence plus élevée. Les répliques synchrones sont utilisées dans chaque centre de données où la faible latence minimise l'impact de la réplication sur les temps de réponse. L'utilisation de la réplication asynchrone réduit l'impact sur les temps de réponse. La distance géographique maintient la disponibilité en cas de défaillance du centre de données local.

Dans l'exemple suivant, les fragments primaires pour la zone de Chicago ont des répliques dans la zone London. Les fragments primaires de la zone London ont des répliques dans la zone de Chicago.

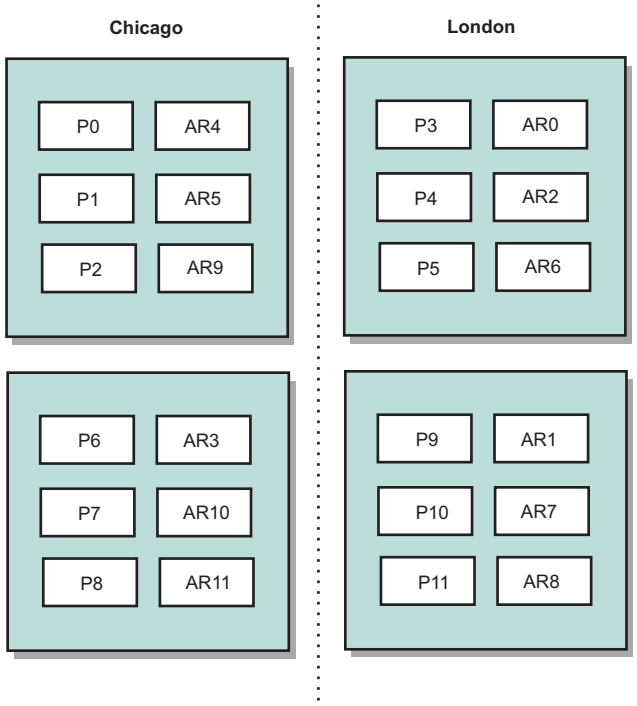


Figure 11. Segments principaux et répliques dans les zones

Trois paramètres de configuration dans eXtreme Scale contrôlent le placement des fragments :

- Définition du fichier de déploiement
- Regroupement de conteneurs
- Définition de règles

Les sections suivantes expliquent les différentes options, de la moins complexe à la plus complexe.

Désactivation du mode de développement

Dans le fichier XML de déploiement définissez `developmentMode="false"`.

Cette étape simple active la règle de placement de fragment eXtreme Scale pour la première fois.

Pour plus d'informations sur les fichiers XML, voir Fichier XML du descripteur de la règle de déploiement.

Règle 1 : les fragments de la même partition sont placés dans des serveurs physiques distincts.

Prenons un exemple simple d'une grille de données avec un fragment de réplique. Avec cette stratégie, les fragments primaires et de réplique de chaque partition se trouvent sur différents serveurs physiques. Si un serveur physique unique tombe en panne, aucune donnée n'est perdue. Le fragment primaire ou le fragment de réplique de chaque partition se trouve sur différents serveurs physiques qui n'ont pas connu d'incident ou les deux se trouvent sur un autre serveur physique qui n'a pas connu d'incident.

La haute disponibilité et la grande simplicité de cette règle constituent la configuration la plus efficace pour tous les environnements de production. Dans la plupart des cas, l'application de cette règle est la seule étape requise pour contrôler efficacement le placement des fragments dans votre environnement.

En appliquant cette stratégie, un serveur physique est défini par une adresse IP. Les fragments sont placés dans des serveurs de conteneur. Les serveurs de conteneur ont une adresse IP, par exemple, le paramètre **-listenerHost** dans le script **startOgServer**. Plusieurs serveurs de conteneur peuvent avoir la même adresse IP.

Etant donné qu'un serveur physique possède plusieurs adresses IP, envisagez l'étape suivante pour renforcer la surveillance de votre environnement.

Définition de zones pour regrouper les serveurs de conteneur

Les serveurs de conteneur sont affectés à des zones avec le paramètre **-zone** sur le script startOgServer. Dans un environnement WebSphere Application Server, les zones sont définies via des groupes de noeuds avec un format de nom spécifique : `ReplicationZone<Zone>`. De cette façon, vous choisissez le nom et les membres de vos zones. Pour plus d'informations, voir Définition des zones des serveurs de conteneur.

Règle 2 : les fragments de la même partition sont placés dans des zones distinctes

Envisagez d'étendre l'exemple d'une grille de données avec un fragment de réplique en le déployant dans deux centres de données. Définissez chaque centre de données sous la forme d'une zone indépendante. Utilisez un nom de zone DC1 pour les serveurs de conteneur dans le premier centre de données pour la première fois, et DC2 pour les serveurs de conteneur dans le second centre de données. Avec cette stratégie, les fragments primaires et de réplique de chaque partition se trouvent dans des centres de données différents. Si un centre de données est défaillant les données sont perdues. Pour chaque partition, son fragment primaire ou de réplique se trouve dans l'autres centre de données.

Dans cette stratégie, vous pouvez contrôler le placement des fragments en définissant des zones. Vous choisissez votre limite logique ou physique ou regroupement d'intérêt. Ensuite, sélectionnez un nom de zone unique pour chaque groupe et démarrez les serveurs de conteneur dans chacune des zones avec le nom de la zone approprié. Ainsi, eXtreme Scale place les fragments pour que les fragments d'une même partition soient placés dans des zones distinctes.

Définition de règles de zone

Le meilleur niveau de contrôle sur le placement de fragment est obtenu à l'aide des règles de zone. Les règles de zone sont définies dans l'élément zoneMetadata du code XML du descripteur de stratégie de déploiement eXtreme Scale. Une règle de zone définit un ensemble de zones dans lesquels les fragments sont placés. Un élément shardMapping affecte un fragment à une règle de zone. L'attribut de fragment de l'élément shardMapping définit le type de fragment :

- P spécifie le fragment primaire
- S spécifie des fragments de réplique synchrones
- A spécifie des fragments de réplique asynchrones.

Si plusieurs répliques synchrones ou asynchrones existent, vous devez fournir des éléments shardMapping correspondant au type de fragment approprié. L'attribut

exclusivePlacement de l'élément zoneRule détermine le placement des fragments dans la même partition dans les zones. Les valeurs de l'attribut exclusivePlacement sont :

- `true` (un fragment ne peut pas être placé dans la même zone qu'un autre fragment de la même partition).

  A faire : Dans le cas " true ", vous devez disposer au minimum d'autant de zones dans la règle que de fragments qui l'utilisent. Dans ce cas, chaque fragment se trouve dans sa propre zone.
- `false` (les fragments d'une même partition peuvent être placés dans la même zone).

Le paramètre par défaut est true.

Pour plus d'informations, voir Exemple : Définitions de zone dans le fichier XML de descripteur de stratégie de déploiement.

Cas d'utilisation étendu

Voici différents cas d'utilisation pour les stratégies de placement des fragments :

Mise à niveau cycliques

Supposons que vous voulez appliquer des mises à niveau cycliques aux serveurs physiques, y compris la maintenance qui implique de redémarrer le déploiement. Dans cet exemple, nous supposons que vous disposez d'une grille de données répartie sur 20 serveurs physiques, définie avec une réplique synchrone. Vous voulez arrêtez 10 des serveurs physiques simultanément pour la maintenance.

Lorsque vous arrêtez des groupes de 10 serveurs physiques, aucune partition n'a ses fragments primaire et de réplique sur les serveurs que vous arrêtez. Autrement, vous perdez les données de la partition.

La solution la plus simple consiste à définir une troisième zone. Au lieu d'utiliser deux zones contenant chacune 10 serveurs physiques, utilisez-en trois, deux avec sept serveurs physiques et une avec six serveurs. En répartissant les données dans plusieurs zones vous améliorez la reprise en ligne pour la disponibilité.

Au lieu de définir une autre zone, l'autre approche consiste à ajouter une réplique.

Mise à niveau d'eXtreme Scale

Lorsque vous mettez à niveau le logiciel eXtreme Scale de manière cyclique avec des grilles de données qui contiennent des données actives, tenez comptes des points suivants. La version du logiciel de service de catalogue doit être supérieure ou égale à la version du logiciel serveur de conteneur. Mettez à niveau tous les serveurs de catalogue pour la première fois à l'aide d'une stratégie cyclique. Voir la rubrique Mise à jour des serveurs eXtreme Scale pour en savoir plus sur la mise à niveau du déploiement.

Modification du modèle de données

Une question connexe est la manière de modifier le modèle de données ou le schéma des objets qui sont stockés dans la grille de données sans entraîner d'indisponibilité. La modification du modèle de données en arrêtant la grille de données et en redémarrant avec les classes de modèles de données mises à jour

dans le chemin d'accès aux classes du serveur de conteneur et en rechargeant la grille de données constituerait une gêne. Une autre solution consiste à lancer une nouvelle grille de données avec le nouveau schéma, copier les données de l'ancienne grille de données vers la nouvelle et fermer l'ancienne grille de données.

Chacun de ces processus génère des perturbations et entraîne un arrêt. Pour modifier le modèle de données sans interruption, stockez l'objet dans l'un des formats suivants :

- Utilisation de XML comme valeur.
- Utilisation d'un objet BLOB créé avec Google protobuf
- Utilisation de JSON (JavaScript Object Notation)

Ecrivez des sérialiseurs pour remplacer un objet Java (POJO) par l'un des ces formats aisément sur le client. Les modifications de schéma deviennent plus simples.

Virtualisation

L'informatique en nuage et la virtualisation sont des technologies émergentes qui connaissent un succès croissant. Par défaut, eXtreme Scale garantit que deux fragments de la même partition ne sont jamais placés à la même adresse IP comme indiqué dans la règle 1. Lorsque vous déployez sur des images virtuelles, telles que VMware, la plupart des instances de serveur, ayant chacune une adresse IP unique, peuvent être exécutées sur le même serveur physique. Pour que les répliques puissent être placées uniquement sur des serveurs physiques distincts, vous pouvez utiliser des zones pour résoudre le problème. Regroupez les serveurs physiques dans des zones et utilisez des règles de placement de zone pour maintenir les fragments primaire et de réplique dans des zones distinctes.

Zones pour les réseaux WAN (wide-area networks)

Vous voudrez peut-être déployer une grille de données eXtreme Scale unique dans des bâtiments ou centres de données avec des connexions réseau lentes. La lenteur accrue des connexions réseau entraîne la réduction de la bande passante et l'augmentation des temps d'attente pour les connexions. Dans ce mode, des partitions réseau sont plus susceptibles de se produire en raison de la congestion du réseau et d'autres facteurs.

Pour faire face à ces risques, le service de catalogue eXtreme Scale organise les serveurs de conteneurs dans des groupes centraux qui échangent des signaux de présence pour détecter leur défaillance éventuelle. Ces principaux groupes ne couvrent pas les zones. Un responsable dans chaque groupe principal envoie les informations d'appartenance au service de catalogue. Le service de catalogue vérifie les défaillances signalées avant de répondre à des informations d'appartenance en envoyant un signal de présence au serveur de conteneur concerné. Si le service de catalogue identifie une fausse détection de défaillance, le service de catalogue n'effectue aucune action. La partition du groupe principal se rétablit rapidement. Le service de catalogue envoie également un signal de présence aux responsables de groupe principal régulièrement à une fréquence lente pour traiter le cas de l'isolement de groupe principal.

Les expulseurs

Les expulseurs retirent des données de la grille de données. Vous pouvez définir un expulseur Time ou un expulseur par défaut. Comme des expulseurs sont associés à des mappes de sauvegarde, utilisez l'interface BackingMap pour spécifier l'expulseur enfichable.

Expulseur par défaut

Un expulseur basé sur la durée de vie est créé par défaut avec chaque mappe de sauvegarde. L'expulseur par défaut supprime les entrées en se basant sur un concept de durée de vie. Ce comportement est défini par l'attribut ttlType, dont les types sont les suivants :

Aucun

Spécifie que les entrées n'expirent jamais et par conséquent ne sont jamais supprimées de la mappe.

Heure de création

Spécifie que les entrées sont expulsées en fonction de la date et de l'heure auxquelles elles ont été créées.

Si vous utilisez l'attribut ttlType CREATION_TIME, l'expulseur expulsera une entrée lorsque elle aura été créée à la date et à l'heure spécifiées par la valeur de l'attribut, laquelle est définie en millisecondes dans la configuration de l'application. Si vous définissez à 10 secondes la valeur de l'attribut TimeToLive, l'entrée est automatiquement expulsée dix secondes après son insertion.

Il est important de faire attention lors de la définition de cette valeur. Cet expulseur s'avère utile dans les cas de quantités raisonnablement élevées d'ajouts au cache utilisés pendant un temps donné. Avec cette stratégie, tout ce qui est créé est supprimé à la fin de la durée définie.

Le type TTL CREATION_TIME est utile dans des scénarios où l'on doit actualiser des cours boursiers toutes les 20 minutes, voire moins. Supposons qu'une application Web récupère les cours de la bourse, sans que l'obtention des cours les plus récents soit cruciale. Dans ce cas, les cours sont mis en cache dans un ObjectGrid pendant 20 minutes. Après 20 minutes, les entrées de la mappe ObjectGrid expirent et sont expulsées. Toutes les vingt minutes environ, la mappe ObjectGrid utilise le plug-in Loadere pour actualiser ses données avec des données neuves provenant de la base. Celle-ci est actualisée toutes les 20 minutes avec les cours les plus récents.

Heure du dernier accès

Spécifie que les entrées sont expulsées en fonction de l'heure de leur dernier accès, qu'elles aient été lues ou actualisées.

Heure de la dernière modification

Spécifie que les entrées sont expulsées en fonction de l'heure de leur dernière modification.

Si vous utilisez l'attribut ttlType LAST_ACCESS_TIME ou LAST_UPDATE_TIME, donnez une valeur plus faible à TimeToLive que si vous utilisiez l'attribut CREATION_TIME, car les entrées d'attribut TimeToLive sont réinitialisées lors de chaque accès. En d'autres termes, si l'attribut TimeToLive est égal à 15 et qu'une entrée a existé 14 secondes

mais qu'on y accède, elle n'expire pas pendant les 15 prochaines secondes. Si vous définissez TimeToLive sur une valeur relativement élevée, il est possible que de nombreuses entrées ne soient pas expulsées. Cependant, si vous définissez la valeur sur quelque chose comme 15 secondes, les entrées avec peu d'accès risquent d'être supprimées.

Le type TTL LAST_ACCESS_TIME ou LAST_UPDATE_TIME sont utiles dans des scénarios où l'on doit conserver les données de session d'un client dans une mappe ObjectGrid. Les données de session doivent être détruites si le client ne les utilise pas pendant un certain laps de temps. Par exemple, après 30 minutes d'inactivité du client. Dans ce cas, l'utilisation d'un type TTL LAST_ACCESS_TIME ou LAST_UPDATE_TIME avec un attribut TimeToLive défini à 30 minutes convient tout à fait à cette application.

Vous pouvez également écrire vos propres expulseurs : pour plus d'informations, voir Ecriture d'un expulseur personnalisé.

Expulseur enfichable

L'expulseur TTL par défaut utilise une stratégie d'expulsion basée sur le temps, et le nombre d'entrées dans la mappe de sauvegarde n'a pas d'effet sur le délai d'expiration d'une entrée. Vous pouvez utiliser un expulseur connectable facultatif pour expulser des entrées en fonction du nombre d'entrées existantes au lieu du temps.

Les expulseurs optionnels fournissent des algorithmes communément utilisés pour décider quelles entrées expulser dès qu'une mappe de sauvegarde dépasse une certaine taille.

- L'expulseur LRUEvictor utilise un algorithme déterminant les entrées les moins récemment utilisées (LRU).
- L'expulseur LFUEvictor utilise un algorithme déterminant les entrées les moins fréquemment utilisées(LFU).

La mappe de sauvegarde informe un expulseur quand des entrées sont créées, modifiées ou supprimées d'une transaction. La mappe de sauvegarde suit ces entrées et choisit quand expulser de l'instance BackingMap une ou plusieurs de ces entrées.

Une instance BackingMap ne dispose pas d'informations de configuration pour une taille maximale. A la place, les propriétés d'expulseur sont définies pour contrôler le comportement de ce dernier. Le LRUEvictor et le LFUEvictor ont une taille maximale utilisée pour déclencher l'expulsion des entrées quand une certaine taille est dépassée. Comme l'expulseur TTL, il est possible que les expulseurs LRU et LFU n'expulsent pas immédiatement une entrée quand le nombre maximal d'entrées est atteint, pour minimiser l'impact sur les performances.

Si l'algorithme d'expulsion LRU ou LFU se révèle inadéquat pour une application particulière, vous pouvez écrire vos propres expulseurs pour créer votre stratégie d'expulsion.

Expulsion basée sur la mémoire

Important : L'expulsion basée sur la mémoire est prise en charge uniquement par Java Platform, Enterprise Edition Version 5 ou ultérieure.

Tous les expulseurs pré-intégrés prennent en charge l'expulsion basée sur la mémoire, qui peut être activée sur l'interface BackingMap en définissant l'attribut evictionTriggers sur `MEMORY_USAGE_THRESHOLD`. Pour plus d'informations sur la définition de l'attribut evictionTriggers dans BackingMap, voir Interface BackingMap et Fichier XML du descripteur d'ObjectGrid.

L'expulsion basée sur la mémoire repose sur un seuil d'utilisation d'un segment de mémoire. Quand cette expulsion est activée dans la mappe de sauvegarde et que cette dernière dispose d'un expulseur pré-intégré, le seuil d'utilisation est défini en un pourcentage par défaut de la mémoire totale, si le seuil n'a pas été défini auparavant.

Lors de l'utilisation de l'expulsion basée sur la mémoire, vous devez configurer le seuil de récupération de place sur la même valeur que l'utilisation du segment de mémoire cible. Par exemple, si le seuil de l'expulsion basée sur la mémoire est à 50 % et que le seuil de récupération de place est à la valeur par défaut de 70 %, alors l'utilisation du segment de mémoire peut aller jusqu'à 70 %. Cette augmentation de l'utilisation du segment de mémoire se produit car l'expulsion basée sur la mémoire n'est déclenchée qu'après un cycle de récupération de place.

Pour modifier le pourcentage du seuil d'utilisation, définissez la propriété memoryThresholdPercentage dans les fichiers de propriétés de conteneur et de serveur pour les processus de serveur eXtreme Scale. Pour définir le seuil d'utilisation cible dans un processus client, vous pouvez utiliser MemoryPoolMXBean.

L'expulsion basée sur la mémoire utilisée par WebSphere eXtreme Scale est sensible au comportement de l'algorithme en cours d'utilisation. Le meilleur algorithme pour l'expulsion basée sur la mémoire est le collecteur de débit par défaut d'IBM. Les algorithmes de génération de récupération de place peuvent avoir des comportements indésirables et il est déconseillé de les utiliser avec l'expulsion basée sur la mémoire.

Pour changer le pourcentage du seuil d'utilisation, définissez la propriété memoryThresholdPercentage sur les fichiers de propriété de conteneur et de serveur pour les processus serveur eXtreme Scale.

Si pendant l'exécution, l'utilisation de la mémoire dépasse le seuil cible, les expulseurs basés sur la mémoire commencent à expulser des entrées et essaient de garder l'utilisation de la mémoire sous le seuil d'utilisation cible. Cependant, il n'existe aucune garantie que la vitesse d'expulsion soit assez grande pour éviter une erreur de dépassement de mémoire si l'exécution du système continue à consommer rapidement de la mémoire.

Présentation de l'infrastructure OSGi

OSGi définit un système de module dynamique pour Java. La plateforme de service OSGi présente une architecture à couches et elle est conçue pour s'exécuter dans plusieurs profils standard Java. Vous pouvez démarrer les serveurs et les clients WebSphere eXtreme Scale dans un conteneur OSGi.

Avantages de l'exécution des applications dans le conteneur OSGi

Le support WebSphere eXtreme Scale OSGi permet de déployer le produit dans l'infrastructure OSGi Eclipse Equinox. Auparavant, si vous souhaitiez mettre à

niveau les plug-ins utilisés par eXtreme Scale, vous deviez redémarrer la machine virtuelle Java (JVM) pour appliquer les nouvelles versions des plug-ins. Avec le support de plug-ins dynamiques fourni par l'infrastructure OSGi, vous pouvez désormais mettre à jour les classes de plug-in sans redémarrer la machine virtuelle Java. Ces plug-ins sont exportés par ensembles d'utilisateur comme services. WebSphere eXtreme Scale accède au(x) service(s) en les recherchant dans le registre OSGi.

Les conteneurs eXtreme Scale peuvent être configurés pour démarrer plus aisément et dynamiquement le service d'administration de configuration OSGi ou avec OSGi Blueprint. Si vous voulez déployer une nouvelle grille avec sa stratégie de placement, vous pouvez le faire en créant une configuration OSGi ou en déployant un ensemble avec des fichiers descripteurs XML eXtreme Scale. Avec le support OSGi, les ensembles d'applications contenant des données de configuration eXtreme Scale peuvent être installés, démarrés, mis à jour et désinstallés sans redémarrer tout le système. Avec cette fonction, vous pouvez mettre à niveau l'application sans perturber la grille de données.

Les beans de plug-in et les services peuvent être configurés avec des portées de fragment personnalisées pour permettre une intégration précise d'options aux autres services exécutés dans la grille. Chaque plug-in peut utiliser des classements OSGi Blueprint pour vérifier que chaque instance activée du plug-in correspond au niveau de version correct. Un bean géré OSGi (MBean) et l'utilitaire **xscmd** sont fournis pour vous permettre d'exécuter des requêtes sur les services OSGi de plug-in eXtreme Scale et leurs classements.

Avec cette fonction, les administrateurs peuvent identifier rapidement les erreurs potentielles de configuration et d'administration et mettre à jour les classements de service de plug-in utilisés par eXtreme Scale.

Ensembles OSGi

Pour interagir avec les plug-ins et déployer des plug-ins dans l'infrastructure OSGi, vous devez utiliser des *ensembles*. Dans la plateforme de service OSGi, un ensemble est un fichier d'archive JAR (Java archive) qui contient du code Java, des ressources et un manifeste qui décrit l'ensemble et ses dépendances. L'ensemble représente l'unité de déploiement d'une application. Le produit eXtreme Scale prend en charge les types d'ensembles suivants :

Ensemble de serveur
L'ensemble de serveur est le fichier `objectgrid.jar`. Il est installé avec le serveur autonome eXtreme Scale et il est nécessaire pour exécuter les serveurs eXtreme Scale. Vous pouvez également l'utiliser pour exécuter les clients eXtreme Scale ou les mémoires caches internes locales. L'ID d'ensemble pour le fichier `objectgrid.jar` est com.ibm.websphere.xs.server_`<version>`, où la version a le format `<Version>.<Release>.<Modification>`. Par exemple, l'ensemble de serveur pour eXtreme Scale version 7.1.1 est com.ibm.websphere.xs.server_7.1.1.

Ensemble de client
L'ensemble de client est le fichier `ogclient.jar`. Il est installé en même temps que les installations client et autonome eXtreme Scale et il est utilisé pour exécuter les clients eXtreme Scale ou les mémoires caches internes locales. L'ID d'ensemble du fichier `ogclient.jar` est com.ibm.websphere.xs.client_`version`, où la version a le format `<Version>.<Release>.<Modification>`. Par exemple, l'ensemble de client pour eXtreme Scale version 7.1.1 est com.ibm.websphere.xs.client_7.1.1.

Limitations

Vous ne pouvez pas redémarrer l'ensemble eXtreme Scale, car vous ne pouvez pas redémarrer l'ORB (object request broker). Pour redémarrer le serveur eXtreme Scale, vous devez redémarrer l'infrastructure OSGi.

Présentation de l'intégration du cache

L'élément crucial qui permet à WebSphere eXtreme Scale de fonctionner avec cette souplesse et fiabilité est son application des concepts de mise en cache pour optimiser la persistance et la recollecte des données dans pratiquement dans n'importe quel déploiement.

Plug-in de cache niveau 2 (L2) JPA

WebSphere eXtreme Scale inclut des plug-in de mémoire cache de niveau 2 pour les fournisseurs OpenJPA et Hibernate Java Persistence API (JPA). Lorsque vous utilisez l'un de ces plug-ins, l'application utilise l'API JPA. Une grille de données est introduite ente l'application et la base de données pour améliorer les temps de réponse.

L'utilisation d'eXtreme Scale en tant que fournisseur de cache de niveau 2 améliore les performances lors de la lecture et de l'interrogation des données et réduit la charge pesant sur la base de données. WebSphere eXtreme Scale présente plusieurs avantages par rapport aux implémentations de cache pré-intégrées car le cache est automatiquement répliqué entre tous les processus. Lorsqu'un client met une valeur en cache, tous les autres clients peuvent utiliser la valeur mise en cache en local.

Vous pouvez configurer la topologie et les propriétés pour le fournisseur de cache L2 dans le fichier `persistence.xml`. Pour plus d'informations sur la configuration de ces propriétés, voir Propriétés de configuration du cache JPA.

Conseil : Le plug-in de cache L2 JPA requiert une application qui utilise les API JPA. Si vous souhaitez utiliser les API WebSphere eXtreme Scale pour accéder à une source de données JPA, utilisez le chargeur JPA. Pour plus d'informations, voir «Chargeurs JPA», à la page 67.

Remarques relatives à la topologie cache L2 JPA

Les facteurs suivants affectent le type de topologie à configurer :

1. **Quelle quantité de données voulez-vous placer en mémoire cache ?**
   - Si les données peuvent tenir dans un seul segment de mémoire JVM, utilisez la «Topologie imbriquée», à la page 28 ou «Topologie intra-domaine», à la page 27.
   - Dans le cas contraire, utilisez la «Topologie imbriquée et partitionnée», à la page 29 ou «Topologie distante», à la page 31
2. **Quel est le taux de lecture/écriture prévu ?**

   Ce taux affecte les performances du cache L2. Chaque topologie gères différemment les opérations de lecture et d'écriture.
   - «Topologie imbriquée», à la page 28 : lecture locale, écriture distante
   - «Topologie intra-domaine», à la page 27 : lecture locale, écriture locale
   - «Topologie imbriquée et partitionnée», à la page 29 : partitionnée : lecture distante, écriture distante

- «Topologie distante», à la page 31 : lecture distante, écriture distante.

Les applications qui fonctionnent principalement en lecture seule doivent utiliser des topologies intra-domaines lorsque cela est possible. Les applications qui exécutent des opération d'écriture principalement doivent utiliser des topologies intra-domaines.

3. **Quel est le pourcentage de données recherchées par rapport au pourcentage de données trouvées par une clé ?**

   Lorsque le cache des requêtes JPA est activé, les opérations d'interrogation l'utilisent. Activez ce cache pour les applications avec des taux de lecture/écriture élevés uniquement, par exemple, lorsque vous approchez de 99 % d'opérations de lecture. Si vous utilisez le cache des requêtes JPA, vous devez utiliser «Topologie imbriquée», à la page 28 ou «Topologie intra-domaine».

   L'opération de recherche par clé recherche une entité cible si l'entité cible n'a pas de relation. Si l'entité cible a des relations avec le type de recherche EAGER, ces relations sont recherchées avec l'entité cible. Dans le cache de données JPA, un petit nombre de réussites en mémoire obtient toutes les données de relation lors de la recherche de ces relations.

4. **Quel est niveau d'obsolescence toléré des données ?**

   Dans un système comportant un petit nombre de machines JVM, il existe une latence de réplication des données pour les opérations d'écriture. Le cache a pour fonction de gérer une vue de données synchronisée dans toutes les machines JVM. Lorsque vous utilisez la topologie intra-domaine, il existe un délai de réplication de données pour les opérations d'écriture. Les applications qui utilisent cette topologie doivent pouvoir tolérer les lectures obsolètes et les écritures simultanées qui peuvent remplacer les données.

7.1.1+ Topologie intra-domaine

Avec une topologie intra-domaine, les fragments primaires sont placés sur chaque serveur de conteneur dans la topologie. Ces fragments primaires contiennent l'ensemble des données de la partition. N'importe lequel de ces fragments primaires peut également exécuter des opérations d'écriture dans la mémoire cache. Cette configuration élimine le goulot d'étranglement dans la topologie intégrée dans lequel toutes les opérations d'écriture de la mémoire cache doivent passer par un fragment primaire unique.

Dans une topologie intra-domaine, aucun fragment de réplique n'est créé, même si vous avez défini des répliques dans vos fichiers de configuration. Chaque fragment primaire redondant contient une copie complète des données, de sorte qu'il peut également être considéré comme un fragment de réplique. Cette configuration utilise une partition unique, similaire à la topologie intégrée.

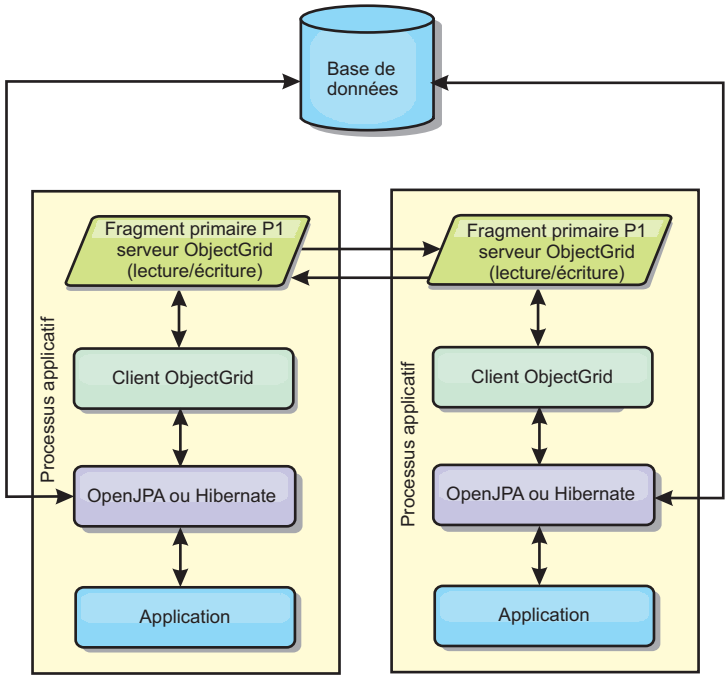


Figure 12. Topologie intra-domaine JPA

Propriétés de configuration du cache JPA associées pour la topologie intra-domaine :

```
ObjectGridName=objectgrid_name,ObjectGridType=EMBEDDED,PlacementScope=CONTAINER_SCOPE,PlacementScopeTopology=HUB | RING
```

Avantages :

- Lectures de cache et des mises à jour localement
- Simple à configurer.

Limitations :

- Cette topologie est la mieux adaptée lorsque les serveurs de conteneur peuvent contenir l'ensemble des données de la partition.
- Les fragments de réplique, même s'ils sont configurés, ne sont jamais placés, car chaque serveur de conteneur héberge un fragment primaire. Toutefois, tous les fragments primaires sont répliqués avec les autres fragments primaires, de sorte que ces fragments primaires deviennent des répliques les uns des autres.

Topologie imbriquée

Conseil : Envisagez d'utiliser une topologie intra-domaine pour obtenir de meilleures performances.

Une topologie imbriquée crée un serveur de conteneur dans l'espace de traitement de chaque application. Les plug-in OpenJPA et Hibernate lisent directement la copie en mémoire du cache et écrivent dans toutes les autres copies. Vous pouvez améliorer les performances d'écriture à l'aide de la réplication asynchrone. Cette topologie par défaut produit un résultat optimal lorsque la quantité de données mises en cache est suffisamment réduite pour être traitée par un seul processus. Avec une topologie intégrée, créez une seule partition pour les données.

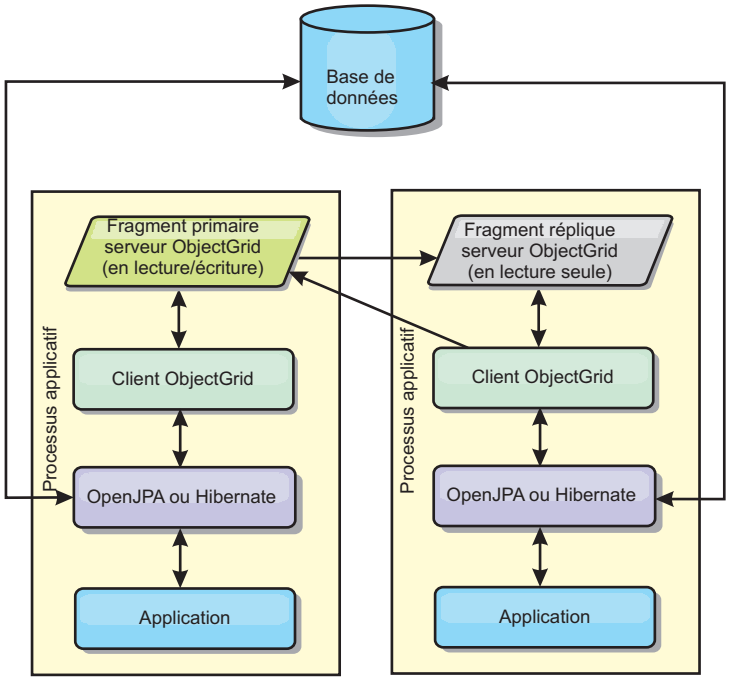


Figure 13. Topologie imbriquée JPA

Propriétés de configuration du cache JPA de la topologie intégrée :

```
ObjectGridName=objectgrid_name,ObjectGridType=EMBEDDED,MaxNumberOfReplicas=num_replicas,ReplicaMode=SYNC | ASYNC | NONE
```

Avantages :

- Toutes les lectures de cache sont des accès locaux rapides.
- Simple à configurer.

Limitations :

- La quantité de données est limitée à la taille du processus.
- Toutes les mises à jour de cache sont envoyées via un fragment primaire, ce qui crée un goulot d'étranglement.

Topologie imbriquée et partitionnée

Conseil : Envisagez d'utiliser une topologie intra-domaine pour obtenir de meilleures performances.

ATTENTION :
N'utilisez pas le cache des requêtes JPA avec un topologie partitionnée. Le cache de requêtes stocke les résultats des requêtes qui sont une collection de clés d'entité. Le cache des requêtes recherche les données d'entité dans l'antémémoire données. Comme l'antémémoire données est divisée entre plusieurs processus, ces appels supplémentaires peuvent faire perdre les avantages du cache L2.

Lorsque les données en mémoire cache sont trop volumineuses pour tenir dans un seul processus, vous pouvez utiliser la topologie partitionnée intégrée. Cette topologie divise le données dans plusieurs processus. Les données sont divisées entre les fragments primaires de sortie que chaque fragment primaire contient un sous-ensemble des données. Vous pouvez toujours utiliser cette option lorsque la latence de la base de données est élevée.

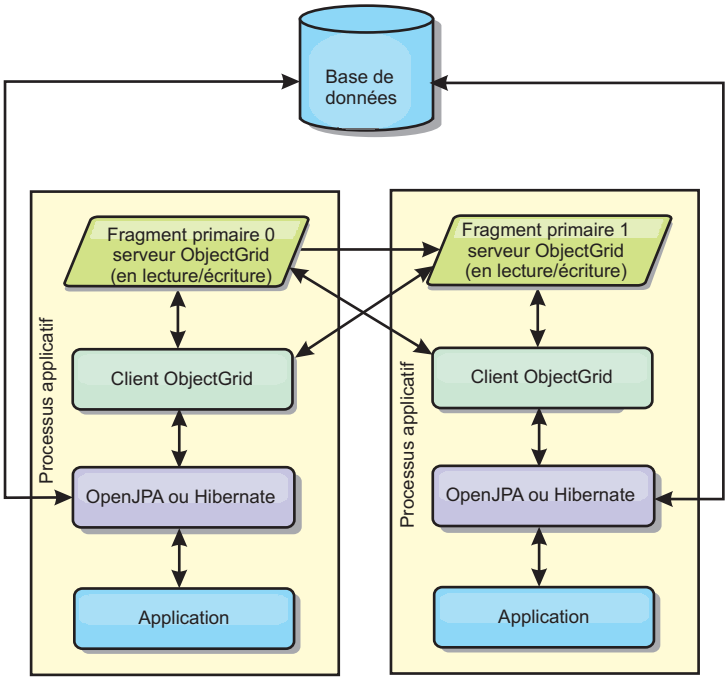


Figure 14. Topologie imbriquée et partitionnée JPA

Propriétés de configuration du cache JPA de la topologie partitionnée intégrée :

```
ObjectGridName=objectgrid_name,ObjectGridType=EMBEDDED_PARTITION,ReplicaMode=SYNC | ASYNC | NONE,
NumberOfPartitions=num_partitions,ReplicaReadEnabled=TRUE | FALSE
```

Avantages :

- Stocke de grandes quantités de données.
- Simple à configurer.
- Les mises à jour de cache sont réparties sur plusieurs processus.

Limitation :

- La plupart des lectures et des mises à jour de cache sont distantes.

Par exemple, pour mettre en cache 10 Go de données avec un maximum de 1 Go par machine JVM, 10 machines virtuelles Java sont nécessaires. Le nombre de partitions doit par conséquent être défini sur 10 ou plus. Idéalement, le nombre de partitions doit être un nombre premier où chaque fragment stocke une quantité raisonnable de mémoire. Le paramètre numberOfPartitions est généralement égal au nombre de machines virtuelles Java. Chaque machine virtuelle Java stocke une partition à l'aide de ce paramètre. Si vous activez la réplication, vous devez augmenter le nombre de machines virtuelles Java dans le système. Dans le cas contraire, chaque machine virtuelle Java stocke également une réplique de partition qui consomme autant de mémoire que la partition principale.

Consultez la rubrique relative à la définition de la taille de la mémoire et au calcul du nombre de partitions dans le *Guide d'administration* pour optimiser les performances de la configuration choisie.

Par exemple, dans un système avec quatre machines virtuelles Java et avec la valeur de paramètre numberOfPartitions 4, chaque machine virtuelle Java héberge une partition principale. Une opération de lecture a 25 pourcents de chances d'extraire des données d'une partition disponible en local, ce qui est sensiblement

plus rapide qu'à partir d'une machine virtuelle Java distante. Si une opération de lecture, telle que l'exécution d'une requête, doit extraire une collection de données impliquant une répartition égale de quatre partitions, 75 pourcents des appels sont distants et 25 pourcents sont locaux. Si le paramètre ReplicaMode est défini sur `SYNC` ou `ASYNC` et si le paramètre ReplicaReadEnabled est défini sur `true`, quatre répliques de partitions sont créées et réparties entre quatre machines virtuelles Java. Chaque machine virtuelle Java héberge une partition principale et une réplique. L'opération de lecture a désormais à 50 pourcents de chances de s'exécuter en local. L'opération de lecture qui extrait une collection de données impliquant une répartition égale de quatre partitions comporte 50 pourcents d'appels distants et 50 pourcents d'appels locaux. Les appels locaux sont considérablement plus rapides que les appels distants. Dès que des appels distants sont effectués, les performances chutent.

Topologie distante

ATTENTION :
N'utilisez pas le cache des requêtes JPA avec une topologie distante. Le cache des requêtes stocke les résultats des requêtes qui sont une collection de clés d'entité. Le cache des requêtes utilise l'antémémoire données pour rechercher toutes les données d'entité. Comme l'antémémoire données est distante, ces appels supplémentaires peuvent faire perdre les avantages du cache L2.

Conseil : Envisagez d'utiliser une topologie intra-domaine pour obtenir de meilleures performances.

Une topologie distante stocke toutes les donnés mises en cache dans un ou plusieurs processus, ce qui réduit la sollicitation de la mémoire par les processus applicatifs. Vous pouvez tirer parti de la répartition de vos données dans des processus distincts en déployant une grille de données eXtreme Scale partitionnée répliquée. Contrairement aux configurations intégrées et intégrées et partitionnées décrites dans les sections précédentes, si vous souhaitez gérer la grille de données distante, vous devez le faire indépendamment de l'application et du fournisseur JPA.

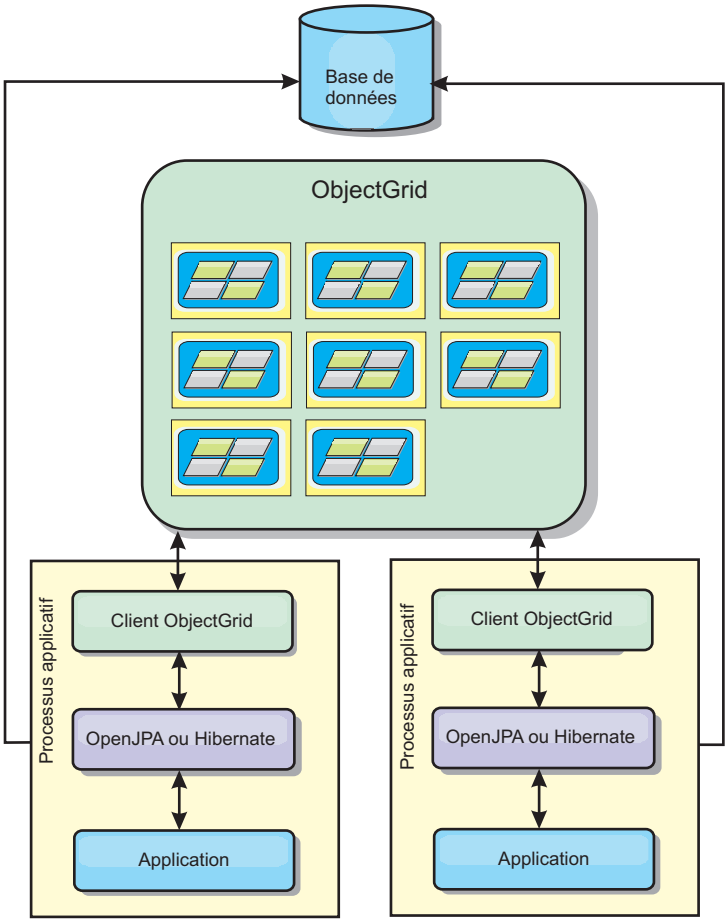


Figure 15. Topologie distante JPA

Propriétés de configuration du cache JPA de la topologie distante :

`ObjectGridName=`*`objectgrid_name`*`,ObjectGridType=REMOTE`

Le type d'ObjectGrid REMOTE ne nécessite pas de paramètres de propriété car l'ObjectGrid et la règle de déploiement sont définis distinctement de l'application JPA. Le plug-in de cache JPA se connecte à distance à un ObjectGrid éloigné existant.

Toute interaction avec la grille d'objets étant éloignée, cette topologie offre les moins bonnes performances parmi tous les types de grille d'objets.

Avantages :

- Stocke de grandes quantités de données.
- Le processus applicatif est exempt de données en cache.
- Les mises à jour de cache sont réparties sur plusieurs processus.
- Options de configuration souples.

Limitation :

- Toutes lectures et mises à jour de cache sont distantes.

Gestion des sessions HTTP

Le gestionnaire de réplication de session fourni avec WebSphere eXtreme Scale peut fonctionner avec le gestionnaire de session par défaut dans le serveur d'applications. Les données de session sont répliquées d'un processus vers l'autre pour prendre en charge la haute disponibilité des données de session utilisateur.

Caractéristiques

Le gestionnaire de session est conçu pour fonctionner dans n'importe quelle version Java Platform, Enterprise Edition et les versions suivantes. Le gestionnaire de sessions ne dépendant en aucune façon des API WebSphere, il peut prendre en charge les diverses versions de WebSphere Application Server ainsi que les environnement de serveurs d'applications du commerce.

Le gestionnaire de sessions HTTP offre des fonctions de réplication de sessions pour une application associée. Le gestionnaire de réplication de session fonctionne avec le gestionnaire de session du conteneur Web. Conjointement, le gestionnaire de session et le conteneur Web créent des sessions HTTP et gèrent les cycles de vie des sessions HTTP associés à l'application. La gestion du cycle de vie comprend : l'invalidation des sessions en fonction d'un délai d'attente, d'un servlet explicite ou d'un appel à des JSP (JavaServer Pages). Autre activité de gestion du cycle de vie : l'invocation de programmes d'écoute des sessions associées à la session ou à l'application Web. Le gestionnaire de sessions conserve ses sessions dans une grille de données complètement partitionnée, clusterisée et répliquée. L'utilisation du gestionnaire de sessions WebSphere eXtreme Scale permet aux gestionnaires de session de fournir le support de basculement de session HTTP lorsque les serveurs d'applications sont arrêtés ou s'arrêtent inopinément. Le gestionnaire de sessions peut également être exécuté dans des environnements qui ne prennent pas en charge l'affinité lorsque cette dernière n'est pas appliquée par un groupe de serveurs d'équilibrage de charge qui diffusent les demandes au groupe de serveurs d'applications.

Scénarios d'utilisation

Le gestionnaire de sessions est particulièrement utile dans les cas suivants :

- Dans les environnements qui utilisent des serveurs d'applications correspondant à des versions différentes de WebSphere Application Server, comme dans le cas d'un scénario de migration.
- Dans les déploiements qui utilisent des serveurs d'application provenant de différentes fournisseurs. Par exemple, application en cours de développement sur des serveurs d'applications source ouverte et hébergée sur WebSphere Application Server. Une application promue de la phase de transfert à la phase de produit en est un autre exemple. Une migration transparente de ces versions de serveur d'applications est possible alors que toutes les sessions HTTP sont opérationnelles et en service.
- Dans les environnements qui nécessitent que l'utilisateur conserve les sessions avec des niveaux de qualité of service (QoS) plus élevés. La disponibilité des sessions est mieux garantie lors du basculement de serveur que les niveaux de qualité de service par défaut WebSphere Application Server.
- Dans un environnement dans lequel l'affinité de session ne peut pas être garantie ou dans les environnements dans lesquels l'affinité est gérée par un équilibreur de charge tiers. Avec un équilibreur de charge de fournisseur, le mécanisme d'affinité doit être personnalisé pour cet équilibreur de charge.

- Dans un environnement pour décharger le traitement requis pour la gestion des sessions et le stockage dans un processus Java externe.
- Dans plusieurs cellules pour activer le basculement de session entre les cellules.
- Dans plusieurs centres de données ou plusieurs zones.

Fonctionnement du gestionnaire de sessions

Le gestionnaire de réplication de session utilise un programme d'écoute de sessions pour écouter les modifications des données de session. Le gestionnaire de réplication de session conserve les données de session dans une instance ObjectGrid localement ou à distance. Des outils livrés avec WebSphere eXtreme Scale vous permettent d'ajouter l'écouteur de session et le filtre de servlet à chacun des modules Web de votre application. Vous pouvez également ajouter manuellement ces écouteurs et ces filtres au descripteur de déploiement Web de votre application.

Ce gestionnaire de réplication de session fonctionne avec chaque gestionnaire de session de conteneur Web de fournisseur pour répliquer les données de session sur les machines virtuelles Java. Lorsque le serveur d'origine expire, les utilisateurs peuvent extraire des données de session d'autres serveurs.

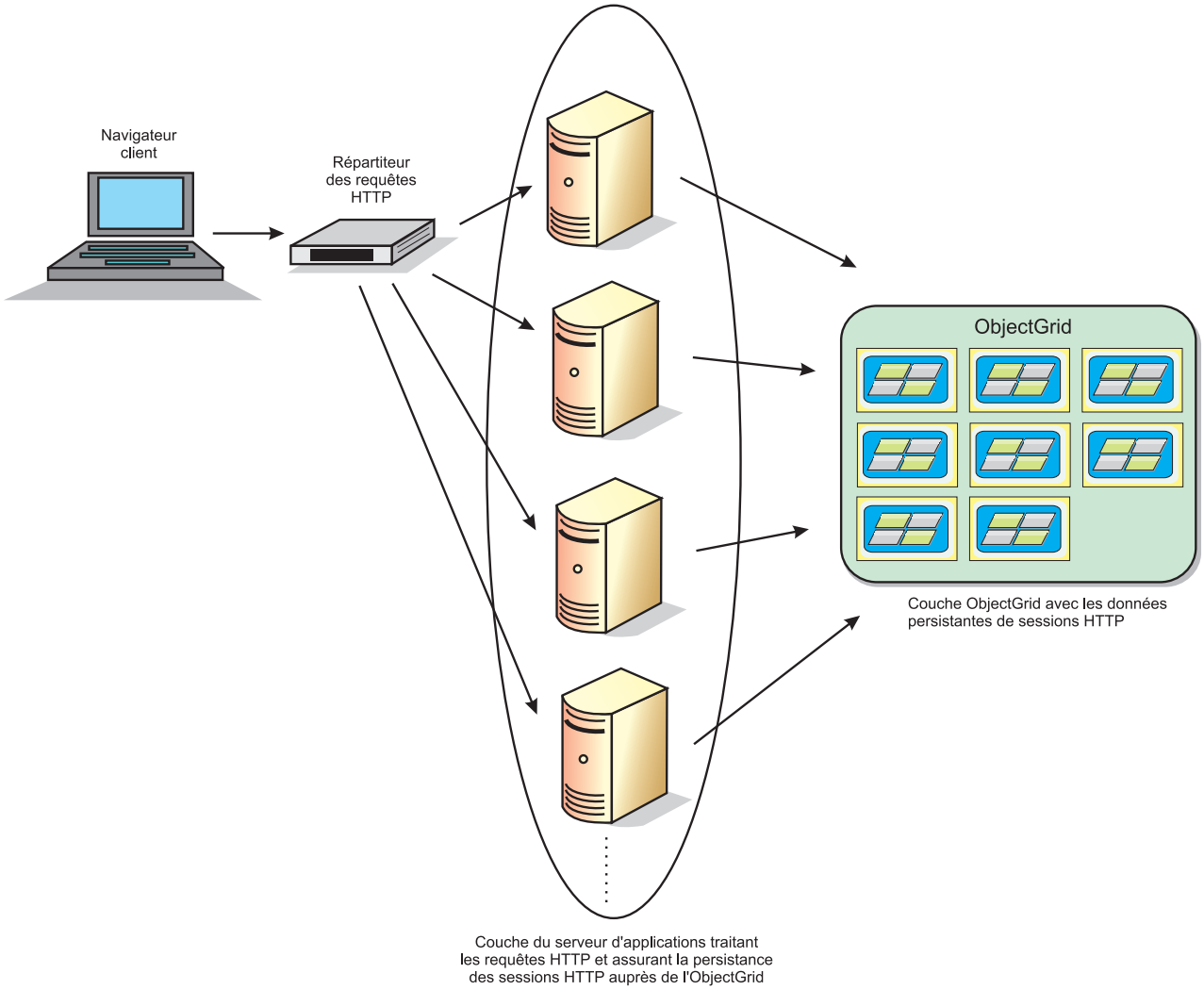


Figure 16. Topologie de gestion de session HTTP avec configuration de conteneur à distance

Topologies de déploiement

Le gestionnaire de sessions peut être configuré à l'aide de deux scénarios de déploiement dynamiques :

Serveurs de conteneur eXtreme Scale connectés au réseau intégrés
Dans ce scénario, les serveurs eXtreme Scale sont regroupés dans les mêmes processus que les servlets. Le gestionnaire de sessions peut communiquer directement avec l'instance ObjectGrid locale, pour éviter les retards coûteux du réseau. Ce scénario est préférable dans une exécution avec affinité où les performances sont vitales.

Serveurs de conteneur eXtreme Scale connectés au réseau distants
Dans ce scénario, les serveurs eXtreme Scale s'exécutent dans des processus externes au processus dans lequel les servlets sont exécutés. Le gestionnaire de sessions communique avec une grille du serveur eXtreme Scale distant. Ce scénario est préférable lorsque le groupe de serveurs conteneurs Web ne dispose pas de la mémoire pour stocker les données de session. Les données de session sont déchargées vers un groupe distinct, ce

qui réduit la consommation de la mémoire sur le groupe de serveurs conteneurs Web. La latence augmente, car les données se trouvent dans un emplacement distant.

Démarrage du conteneur intégré générique

eXtreme Scale démarre automatiquement un conteneur ObjectGrid intégré dans un processus serveur d'applications lorsque le conteneur Web initialise le programme d'écoute de session ou le filtre de servlet si la propriété objectGridType a la valeur `EMBEDDED`. Pour plus de détails, reportez-vous à la rubrique Paramètres d'initialisation du contexte de servlet.

Vous n'êtes pas obligé de regrouper un fichier `ObjectGrid.xml` et un fichier `objectGridDeployment.xml` dans votre fichier WAR ou EAR d'application. Les fichiers par défaut `ObjectGrid.xml` et `objectGridDeployment.xml` sont regroupés dans le fichier JAR du produit. Des mappes dynamiques sont créées par défaut pour les différents contextes de l'application Web. Les mappes eXtreme Scale statiques n'en sont pas moins toujours prises en charge.

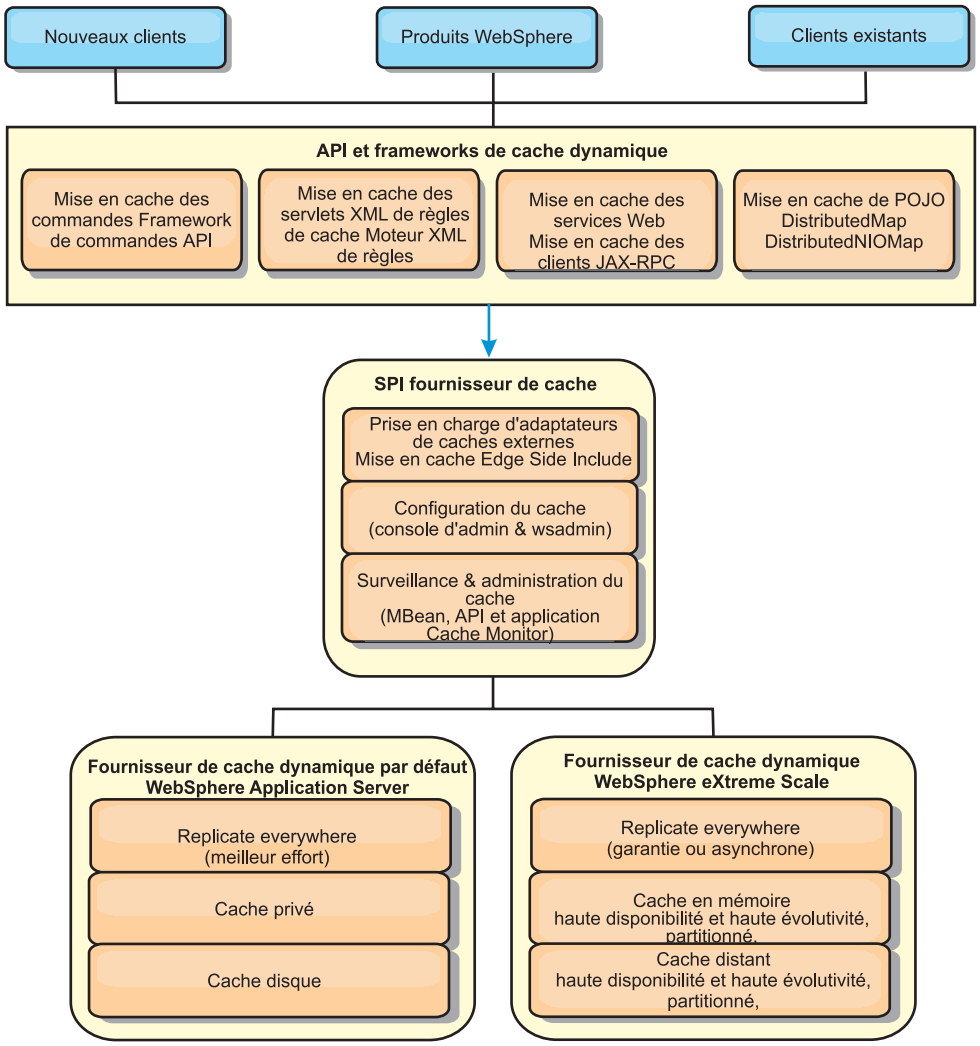
Ce démarrage des conteneurs ObjectGrid intégrés s'applique à tous les types de serveurs d'applications. L'utilisation de composant WebSphere Application Server ou d'un GBean WebSphere Application Server Community Edition GBean est abandonnée.

Fournisseur de cache dynamique

L'API de cache dynamique est disponible pour les applications Java EE qui sont déployées dans WebSphere Application Server. Le fournisseur de cache dynamique peut être optimisé pour mettre en cache les données métier et les fichiers HTML générés ou pour synchroniser les données en cache de la cellule à l'aide du service de réplication des données.

Présentation

Le seul fournisseur de services auparavant disponible était le moteur de cache dynamique par défaut intégré à WebSphere Application Server. Les clients peuvent utiliser l'interface du fournisseur de cache dynamique de WebSphere Application Server pour connecter eXtreme Scale au cache dynamique. En configurant cette fonction, vous permettez aux applications écrites avec l'API de cache dynamique ou aux applications utilisant la mise en cache au niveau des conteneurs (par exemple les servlets) d'optimiser les fonctions et les performances de WebSphere eXtreme Scale.



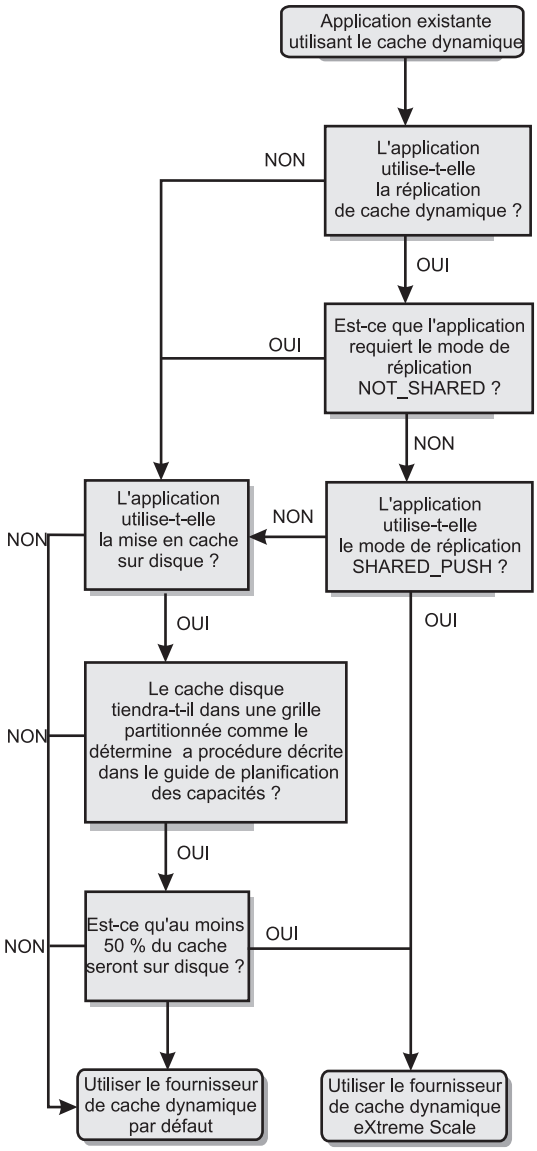
Vous pouvez installer et configurer le fournisseur de cache dynamique comme décrit dans Configuration du fournisseur de cache dynamique pour WebSphere eXtreme Scale.

Différents modes d'optimisation de WebSphere eXtreme Scale

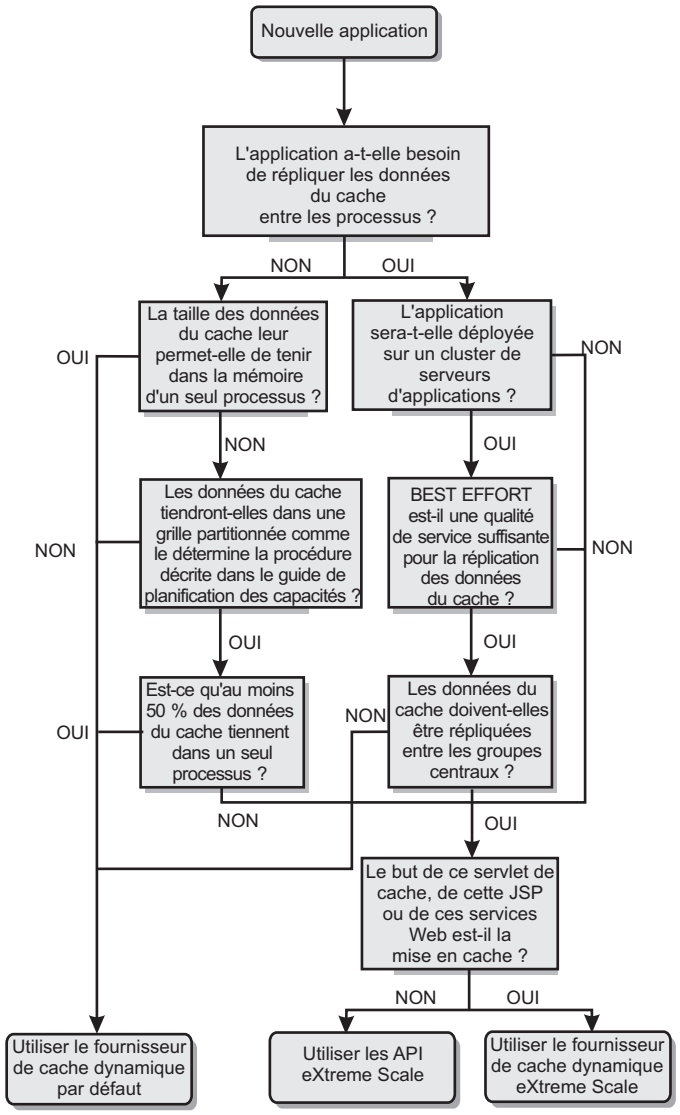
Les différentes fonctions de WebSphere eXtreme Scale permettent d'augmenter de manière significative les fonctions réparties de l'API de cache dynamique au-delà des possibilités offertes par le moteur de cache dynamique et par le service de réplication des données par défaut. Avec eXtreme Scale, vous pouvez créer des mémoires cache véritablement réparties entre plusieurs serveurs et non simplement répliquées et synchronisées d'un serveur à l'autre. Par ailleurs, les mémoires cache eXtreme Scale sont transactionnelles et hautement disponibles : chaque serveur voit ainsi le même contenu pour le service de cache dynamique. WebSphere eXtreme Scale offre une qualité de service en matière de réplication supérieure à celle proposée par DRS.

Tous ces avantages ne signifient cependant pas que le fournisseur de cache dynamique eXtreme Scale constitue la meilleure solution pour toutes les applications. Pour identifier la technologie la mieux adaptée à votre application, utilisez l'arbre de décision et la matrice de comparaison des fonctions.

Arbre de décision permettant de faire migrer des applications existantes de cache dynamique



Arbre de décision permettant de choisir un fournisseur de cache pour les nouvelles applications.



Comparaison des fonctionnalités

Tableau 1. Comparaison des fonctionnalités

Fonctionnalités de cache	Fournisseur par défaut	Fournisseur eXtreme Scale	API eXtreme Scale
Mise en cache local en mémoire	x	x	x
Mise en cache réparti	Imbriqué	Imbriqué, partitionné imbriqué et partitionné distant	Multiple
Evolutivité linéaire		x	x
Réplication fiable (synchrone)		ORB	ORB
Dépassement de capacité des disques	x		

Tableau 1. Comparaison des fonctionnalités (suite)

Fonctionnalités de cache	**Fournisseur par défaut**	**Fournisseur eXtreme Scale**	**API eXtreme Scale**
Expulsion	LRU/TTL/en fonction des segments	LRU/TTL (par partition)	Multiple
Invalidation	x	x	x
Relations	ID de dépendance, modèles	ID de dépendance, modèles	x
Recherches non-clés			Requête et index
Intégration dorsale			Loaders
Transactionnel		Implicite	x
Stockage à base de clés	x	x	x
Evénements et programmes d'écoute	x	x	x
Intégration à WebSphere Application Server	Une seule cellule	Plusieurs cellules	Indépendant des cellules
Prise en charge de Java Standard Edition		x	x
Surveillance et statistiques	x	x	x
Sécurité	x	x	x

Tableau 2. Intégration transparente à la technologie

Fonctionnalités de cache	**Fournisseur par défaut**	Fournisseur **eXtreme Scale**	**API eXtreme Scale**
Mise en cache des résultats servlets/JSP WebSphere Application Server	V5.1+	V6.1.0.25+	
Mise en cache des résultats (JAX-RPC) WebSphere Application Server Web Services	V5.1+	V6.1.0.25+	
Mise en cache des sessions HTTP			x
Fournisseur de cache pour OpenJPA et Hibernate			x
Synchronisation de la base de données à l'aide d'OpenJPA et Hibernate			x

Tableau 3. Interfaces de programmation

Fonctionnalités de cache	Fournisseur par défaut	Fournisseur eXtreme Scale	API eXtreme Scale
API à base de commandes	API de structure des commandes	API de structure des commandes	API DataGrid
API à base de mappes	API DistributedMap	API DistributedMap	API ObjectMap
API EntityManager			x

Pour plus d'informations sur le fonctionnement des mémoires caches réparties eXtreme Scale, voir les informations de configuration du déploiement dans *Guide d'administration*.

Remarque : Le cache eXtreme Scale réparti peut uniquement stocker des entrées dont la clé et la valeur implémentent toutes les deux l'interface java.io.Serializable.

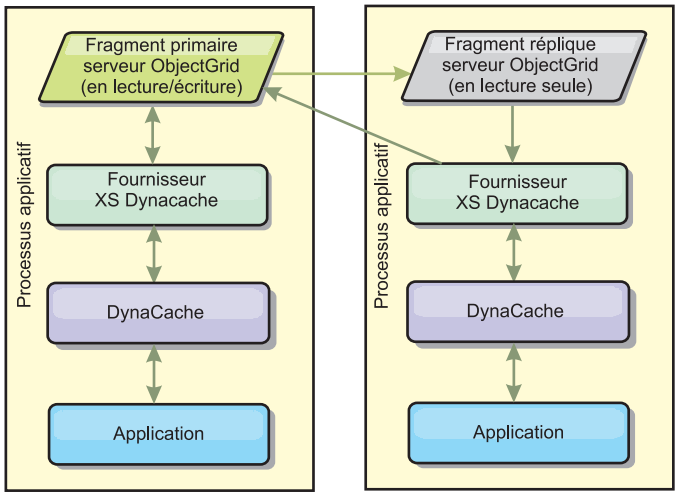
Types de topologie

Un service de cache dynamique créé à l'aide du fournisseur eXtreme Scale peut être déployé dans l'une des trois topologies disponibles. Vous pouvez ainsi personnaliser cette mémoire en fonction de vos besoins en matière de performances, de ressources et d'administration. Ces topologies sont les suivantes : topologie imbriquée, topologie partitionnée imbriquée et topologie distante.

Topologie imbriquée

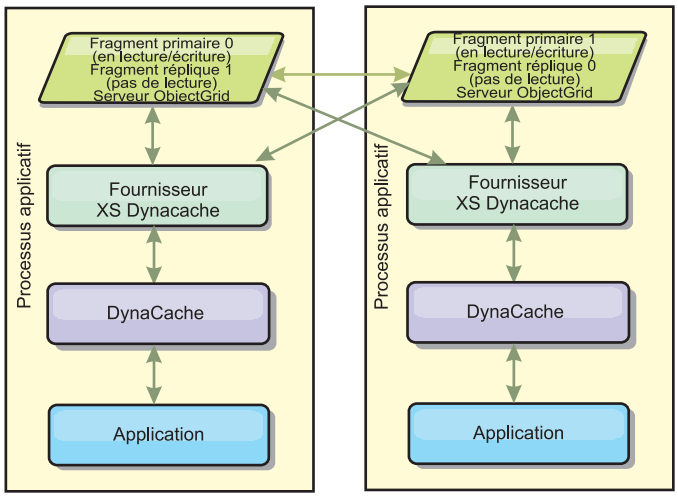
La topologie imbriquée est semblable au cache dynamique et au fournisseur de DRS par défaut. Les instances du cache réparti créées à l'aide de la topologie imbriquée conservent une copie complète de ce cache dans chaque processus eXtreme Scale qui accède au service de cache dynamique, ce qui permet à toutes les opérations de lecture de se faire localement. Toutes les opérations d'écriture passent par un processus serveur unique, au cours duquel les verrous transactionnels sont gérés. Elles sont ensuite répliquées sur les serveurs restants. Cette topologie convient par conséquent mieux aux charges de travail comptant davantage d'opérations de lecture que d'opérations d'écriture.

Avec la topologie imbriquée, les entrées nouvelles ou mises à jour ne sont pas immédiatement visibles sur tous les processus serveur unique. Une entrée de cache n'est pas visible, même pour le serveur l'ayant générée, tant qu'elle ne s'est pas propagée via les services de réplication asynchrones de WebSphere eXtreme Scale. Ces services sont aussi rapides que le matériel le permet, mais il existe toujours un léger décalage. Le graphique suivant présente une topologie imbriquée :



Topologie partitionnée imbriquée

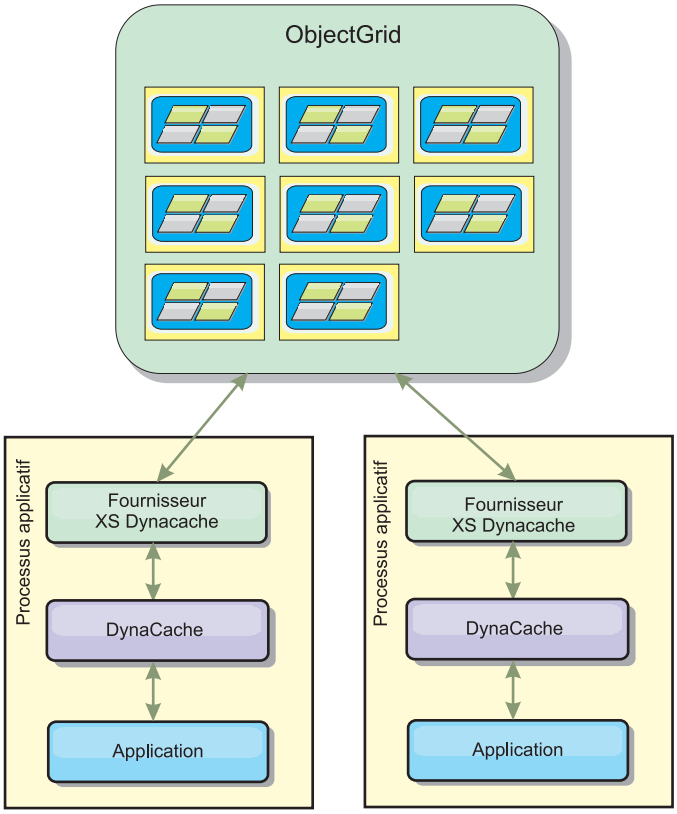
Pour les charges de travail dans lesquelles les opérations d'écriture en cache sont plus fréquentes que les opérations de lecture, une topologie partitionnée imbriquée ou une topologie distante est recommandée. La première conserve toutes les données du cache dans les processus WebSphere Application Server qui accèdent à ce cache. Toutefois, chaque processus ne stocke qu'une partie de ces données. Toutes les opérations de lecture et d'écriture relatives aux données situées sur cette "partition" passent par ce processus, ce qui signifie que la plupart des demandes adressées au cache sont traitées à l'aide d'un appel de procédure distant. Il en découle un temps d'attente plus élevé pour les opérations de lecture qu'avec la topologie imbriquée, mais la capacité du cache réparti à gérer les opérations de lecture et d'écriture évolue de manière linéaire avec le nombre de processus WebSphere Application Server qui accèdent au cache. Par ailleurs, dans cette topologie, la taille maximale du cache n'est pas liée à la taille complète d'un processus WebSphere. Chaque processus en effet ne détenant qu'une partie du cache, la taille maximale de ce dernier est égale à la somme de la taille de tous les processus, moins la taille du processus lui-même. Le graphique suivant représente une topologie partitionnée imbriquée :



Supposons par exemple que vous disposiez d'une grille de processus serveur dont chacun est associé à 256 mégaoctets de segments de mémoire disponibles pour héberger un service de cache dynamique. S'ils utilisaient une topologie imbriquée, le fournisseur de cache dynamique par défaut et le fournisseur eXtreme Scale seraient limités à une taille de cache interne égale à 256 mégaoctets moins la mémoire utilisée par eux-mêmes. Reportez-vous à la section relative à la planification de la capacité et à la haute disponibilité du présent manuel. S'il utilisait une topologie partitionnée imbriquée, le fournisseur eXtreme Scale serait limité à une taille de cache égale à un gigaoctet moins la mémoire utilisée par lui-même. Le fournisseur WebSphere eXtreme Scale permet ainsi de disposer d'un cache dynamique en mémoire plus grand qu'un simple processus serveur. Le fournisseur de cache dynamique par défaut utilise le cache-disque pour permettre aux instances du cache de croître au-delà de la taille d'un simple processus. Dans de nombreux cas, le fournisseur WebSphere eXtreme Scale rend superflue l'utilisation du cache-disque et des systèmes coûteux de stockage sur disque pour exécuter ces instances.

Topologie distante

La topologie distante rend également superflue l'utilisation du cache-disque. La seule différence entre la topologie distante et la topologie partitionnée imbriquée est qu'avec la première, toutes les données du cache sont stockées hors des processus WebSphere Application Server. WebSphere eXtreme Scale prend en charge les processus conteneur autonomes pour les données en cache. Ces processus utilisent moins de mémoire qu'un processus WebSphere Application Server et ils ne sont pas limités à l'utilisation d'une machine virtuelle Java donnée. Les données associées à un service de cache dynamique auquel un processus WebSphere Application Server 32 bits accède peuvent par exemple se trouver sur un processus conteneur s'exécutant sur une machine virtuelle Java eXtreme Scale. Les utilisateurs peuvent ainsi exploiter la capacité mémoire accrue des processus 64 bits pour la mise en cache, sans supporter la mémoire supplémentaire nécessaire aux processus serveur de l'application. Le graphique suivant représente une topologie distante :



Compression des données

La compression est une autre fonction du fournisseur de cache dynamique WebSphere eXtreme Scale pouvant aider les utilisateurs à gérer l'excédent de mémoire généré par le cache. Le fournisseur de cache dynamique ne permet pas la compression en mémoire des données du cache. Avec le fournisseur eXtreme Scale, cette opération est possible. La compression du cache à l'aide de l'algorithme deflate peut être activée dans les trois topologies. Elle génère une charge supplémentaire pour les opérations de lecture et d'écriture, mais permet d'améliorer considérablement la densité de la mémoire cache pour les applications telles que les applications de mise en cache des servlets et de JSP.

Cache interne local

Le fournisseur de cache dynamique de WebSphere eXtreme Scale est utilisable également pour les instances de cache dynamique dans lesquelles la **réplication est désactivée**. Tout comme avec le fournisseur par défaut, ces caches peuvent stocker des données non sérialisables. Ils offrent également de meilleures performances sur les serveurs d'entreprise contenant multiprocesseurs car le codepath eXtreme Scale est conçu pour optimiser les accès simultanés en mémoire cache interne.

Moteur de cache dynamique et différences fonctionnelles avec eXtreme Scale

Lorsque la réplication est désactivée pour les caches locaux en mémoire, il n'existe aucune différence fonctionnelle réelle entre les caches du fournisseur de cache dynamique par défaut et WebSphere eXtreme Scale. Les utilisateurs ne constatent

aucune différence, sinon que les caches WebSphere eXtreme Scale ne supportent pas le déchargement sur disque ni les statistiques ou opérations en rapport avec la taille du cache en mémoire.

Lorsque la réplication est activée, aucune différence notable ne pourra être observée dans les résultats renvoyés par la plupart des appels de l'API de cache dynamique lorsque le client utilise le fournisseur de cache dynamique ou lorsqu'il utilise le fournisseur eXtreme Scale. Pour certaines opérations, il est impossible d'émuler le comportement du moteur de cache dynamique à l'aide d'eXtreme Scale.

Statistiques du cache dynamique

Les statistiques de la mémoire cache dynamique sont générées via l'application CacheMonitor ou via le bean géré de cache dynamique. Lorsque le fournisseur de cache dynamique eXtreme Scale est utilisé, les statistiques sont générées via ces interfaces, mais le contexte des valeurs statistiques est différent.

Si une instance de cache dynamique est partagée entre trois serveurs nommés A, B et C, l'objet des statistiques renvoie des statistiques uniquement pour la copie du cache se trouvant sur le serveur à partir duquel l'appel a été lancé. Si les statistiques sont extraites à partir du serveur A, elles reflètent uniquement l'activité du serveur A.

Avec eXtreme Scale, il n'existe qu'un seul cache réparti partagé entre tous les serveurs : il est donc impossible d'effectuer un suivi de la plupart des statistiques serveur par serveur, contrairement à ce qui est le cas avec le fournisseur de cache dynamique par défaut. Vous trouverez ci-dessous une liste des statistiques générées par l'API Cache Statistics, ainsi que les explications correspondantes lorsque vous utilisez le fournisseur WebSphere eXtreme Scale de cache dynamique. Tout comme le fournisseur par défaut, ces statistiques ne sont pas synchronisées. Elles peuvent donc varier jusqu'à 10 % pour des charges de travail simultanées.

- **Réussites en cache** : les réussites en cache font l'objet d'un suivi par serveur. Si le trafic enregistré sur le serveur A génère 10 réussites en cache et que le trafic enregistré sur le serveur B génère 20 réussites en cache, cette statistique fait état de 10 réussites en cache sur le serveur A et de 20 réussites en cache sur le serveur B.
- **Echecs en cache** : comme les réussites, les échecs en cache font l'objet d'un suivi par serveur.
- **Entrées en cache** : cette statistique renvoie le nombre d'entrées présentes dans le cache réparti. Chaque serveur qui accède au cache renvoie la même valeur, qui correspond au nombre total d'entrées en cache en mémoire pour tous les serveurs.
- **Taille du cache en Mo** : cette mesure n'est actuellement prise en charge que pour les caches utilisant les topologies distantes, imbriquées ou partitionnées imbriquées. Elle indique combien de mégaoctets du segment de mémoire Java sont utilisés par le cache dans l'ensemble de la grille. Cette statistique ne porte que sur l'utilisation des segments de mémoire par les partitions primaires. C'est à vous de prendre en compte les fragments réplique. Le paramètre par défaut des topologies distantes et partitionnées imbriquées étant le fragment réplique asynchrone, il convient en effet de doubler ce chiffre pour connaître l'utilisation effective de la mémoire par le cache.
- **Suppressions dans le cache** : nombre total d'entrées supprimées du cache par quelque méthode que ce soit. Agrégat des suppressions de l'ensemble du cache

réparti. Si le trafic enregistré sur le serveur A génère 10 invalidations et que le trafic enregistré sur le serveur B génère 20 invalidations, la valeur correspondant à la somme des deux serveurs est 30.

- **Suppression LRU (Least Recently Used) du cache** : tout comme les suppressions dans le cache, il s'agit d'un agrégat. Il représente le nombre d'entrées ayant été supprimées pour maintenir la taille du cache en deçà de sa taille maximale.
- **Invalidation pour dépassement du délai d'attente** : agrégat du nombre des entrées qui ont été supprimées en raison d'un dépassement du délai d'attente.
- **Invalidations explicites** : agrégat du nombre des entrées qui ont été supprimées du fait d'une invalidation directe par clé, ID dépendance ou modèle.
- **Statistiques étendues** : le fournisseur de cache dynamique eXtreme Scale exporte les chaînes de clé de statistiques étendues suivantes.
  - **com.ibm.websphere.xs.dynacache.remote_hits** : nombre total de réussites en mémoire faisant l'objet d'un suivi dans le conteneur eXtreme Scale. Il s'agit d'une statistique agrégée. Sa valeur dans la mappe des statistiques étendues est une valeur `longue`.
  - **com.ibm.websphere.xs.dynacache.remote_misses** : nombre total d'échecs en mémoire faisant l'objet d'un suivi dans le conteneur eXtreme Scale. Il s'agit d'une statistique agrégée. Sa valeur dans la mappe des statistiques étendues est une valeur `de type long`.

Rapport sur la réinitialisation des statistiques

Le fournisseur de cache dynamique vous permet de réinitialiser les statistiques du cache. Avec le fournisseur par défaut, l'opération de réinitialisation efface uniquement les statistiques du serveur concerné. Le fournisseur de cache dynamique eXtreme Scale suit la plupart de ses données statistiques sur les conteneurs de cache distant. Ces données ne sont ni effacées ni modifiées lors de la réinitialisation des statistiques. Au contraire, le comportement du cache dynamique par défaut est simulé sur le client en signalant la différence entre la valeur actuelle d'une statistique donnée et sa valeur au moment du dernier appel de réinitialisation de ce serveur.

Si, par exemple, le trafic enregistré sur le serveur A génère 10 suppressions dans le cache, les statistiques concernant le serveur A et le serveur B indiquent 10 suppressions. Si les statistiques du serveur B sont réinitialisées et que 10 suppressions supplémentaires sont enregistrées sur le serveur A, les statistiques du serveur A indiqueront 20 suppressions et celles du serveur B indiqueront 10 suppressions.

Evénements du cache dynamique

L'API de cache dynamique permet aux utilisateurs d'enregistrer des programmes d'écoute d'événements. Lorsque vous utilisez eXtreme Scale en tant que fournisseur de cache, ces programmes fonctionnent comme prévu pour les mémoires cache internes.

Pour les mémoires cache réparties, le comportement des événements dépend de la topologie utilisée. Si la topologie est imbriquée, les événements sont générés sur le serveur qui gère les opérations d'écriture, également appelé fragment primaire. Ceci signifie qu'un seul serveur reçoit les notifications d'événements, mais qu'il recevra toutes les notifications normalement attendues par le fournisseur de cache

dynamique. Comme WebSphere eXtreme Scale choisit le fragment primaire au moment de l'exécution, il est impossible de s'assurer qu'un processus serveur donné reçoit toujours ces événements.

Les mémoires cache partitionnées imbriquées génèrent des événements sur les serveurs hébergeant une partition du cache. Ainsi, si un cache contient 11 partitions et que chaque serveur d'une grille WebSphere Network Deployment constituée de 11 serveurs héberge l'une des partitions, chaque serveur reçoit les événements du cache dynamique relatifs aux entrées de cache qu'il héberge. Aucun processus serveur ne peut voir à lui tout seul tous les événements, à moins que les 11 partitions ne soient hébergées dans ce processus serveur. Comme dans le cas de la topologie imbriquée, il est impossible de s'assurer qu'un processus serveur donné va recevoir un ensemble donné d'événements ou aucun événement.

Les mémoires cache utilisant une topologie distante ne prennent pas en charge les événements du cache dynamique.

Appels des beans gérés

Le fournisseur de cache dynamique WebSphere eXtreme Scale ne prend pas en charge la mise en cache sur un disque. Les appels de beans gérés relatifs à une mise en cache sur un disque ne fonctionnent pas.

Mappage des règles de réplication du cache dynamique

Le fournisseur de cache dynamique pré-intégré de WebSphere Application Server prend en charge plusieurs règles de réplication du cache. Vous pouvez configurer ces règles de manière globale ou pour chaque entrée du cache. Pour obtenir une description de ces règles, consultez la documentation relative au cache dynamique.

Le fournisseur de cache dynamique eXtreme Scale n'honore pas directement ces règles. Les caractéristiques de réplication du cache sont déterminées par le type de topologie répartie eXtreme Scale configuré. Elles s'appliquent à toutes les valeurs placées dans cette mémoire, quelles que soient les règles de réplication définies par le service de cache dynamique pour l'entrée. Vous trouverez ci-dessous une liste de toutes les règles de réplication prises en charge par le service de cache dynamique et une description des topologies eXtreme Scale offrant des caractéristiques de réplication semblables.

Remarquez que le fournisseur de cache dynamique eXtreme Scale ignore les règles de réplication DRS sur un cache ou sur une entrée de cache. Les utilisateurs doivent choisir la topologie la mieux adaptée à leurs besoins en réplication.

- NOT_SHARED – aucune des topologies actuellement proposées par le fournisseur de cache dynamique eXtreme Scale ne se rapproche de ces règles. Cela signifie que toutes les données stockées en cache doivent être associées à des clés et à des valeurs qui implémentent java.io.Serializable.
- SHARED_PUSH : la topologie imbriquée se rapproche de ces règles de réplication. Lorsqu'une entrée de cache est créée, elle est répliquée vers l'ensemble des serveurs. Ceux-ci recherchent les entrées de cache localement uniquement. Si une entrée n'est trouvée localement, elle est supposée ne pas exister et les autres serveurs ne sont pas interrogés à son sujet.
- SHARED_PULL et SHARED_PUSH_PULL : les topologies partitionnées imbriquées et les topologies distantes se rapprochent de ces règles de réplication. L'état réparti du cache est parfaitement cohérent sur tous les serveurs.

Ces informations sont destinées à vous aider à vérifier que la topologie choisie correspond à vos besoins en matière de cohérence répartie. Si, par exemple, la topologie imbriquée correspond parfaitement à vos besoins en matière de déploiement et de performances, mais que vous souhaitiez bénéficier du niveau de cohérence du cache apporté par SHARED_PUSH_PULL, vous pouvez envisager d'utiliser la topologie partitionnée imbriquée en dépit de ses performances légèrement inférieures.

Sécurité

Vous pouvez sécuriser les instances de cache dynamique qui s'exécutent dans des topologies imbriquées ou dans des topologies partitionnées imbriquées grâce à la fonction de sécurité intégrée à WebSphere Application Server. Consultez la documentation relative à Sécurisation des applications dans leur environnement dans le centre de documentation de WebSphere Application Server.

Lorsqu'un cache s'exécute dans une topologie distante, un client eXtreme Scale autonome peut se connecter au cache et affecter le contenu de l'instance de cache dynamique. Le fournisseur de cache dynamique eXtreme Scale est doté d'une fonction légère de chiffrement pouvant empêcher la lecture ou la modification des données du cache par des clients autres que les clients WebSphere Application Server. Pour l'activer, associez le paramètre facultatif **com.ibm.websphere.xs.dynacache.encryption_password** à la même valeur pour chaque instance WebSphere Application Server qui accède au fournisseur de cache dynamique. La valeur et les métadonnées utilisateur de l'entrée de cache sont ainsi chiffrés à l'aide de la fonction de chiffrement AES 128 bits. Il est très important de définir la même valeur sur tous les serveurs, car ils ne parviendront pas à lire les données mises en cache par d'autres serveurs avec une valeur différente.

Si le fournisseur eXtreme Scale détecte différentes valeurs pour cette variable dans le même cache, il génère un avertissement dans le journal du processus conteneur eXtreme Scale.

Reportez-vous à la documentation eXtreme Scale sur «Sécurité», à la page 136 si SSL ou l'authentification SSL ou du client est nécessaire.

Informations supplémentaires

- Redbook relatif au cache dynamique
- Documentation relative au cache dynamique
  - WebSphere Application Server 7.0
  - WebSphere Application Server 6.1
- Documentation relative à DRS
  - WebSphere Application Server 7.0
  - WebSphere Application Server 6.1

Intégration de la base de données : caches avec écriture différée, caches en ligne et caches secondaires

WebSphere eXtreme Scale est utilisé pour servir de frontal à une base de données classiques et ainsi éliminer l'activité de lecture qui est normalement envoyée vers la base de données. Un cache cohérent peut être utilisé avec une application soit directement, soit indirectement en passant alors par un associateur relationnel d'objets (ORM). Le cache cohérent peut décharger des tâches de lecture la base de données ou le dorsal. Dans un scénario un tout petit peu plus complexe, comme

celui d'un accès transactionnel à un dataset dans lequel seules certaines données requièrent des garanties de persistance classique, il est possible d'utiliser le filtrage pour décharger même les transactions d'écriture.

Vous pouvez configurer WebSphere eXtreme Scale pour qu'il fonctionne en tant qu'espace extrêmement flexible de traitement de base de données interne. Cela dit, WebSphere eXtreme Scale n'est pas un associateur relationnel d'objets. Il ne sait pas d'où les données de la grille de données proviennent. Une application ou un associateur relationnel d'objets peuvent placer des données sur un serveur eXtreme Scale. C'est à la source de données qu'il incombe de vérifier la cohérence des données avec leur base de données d'origine. En d'autres termes, eXtreme Scale ne peut pas invalider les données qu'il a extraites automatiquement d'une base de données. C'est à l'application ou à l'associateur de fournir cette fonction et de gérer les données stockées dans eXtreme Scale.

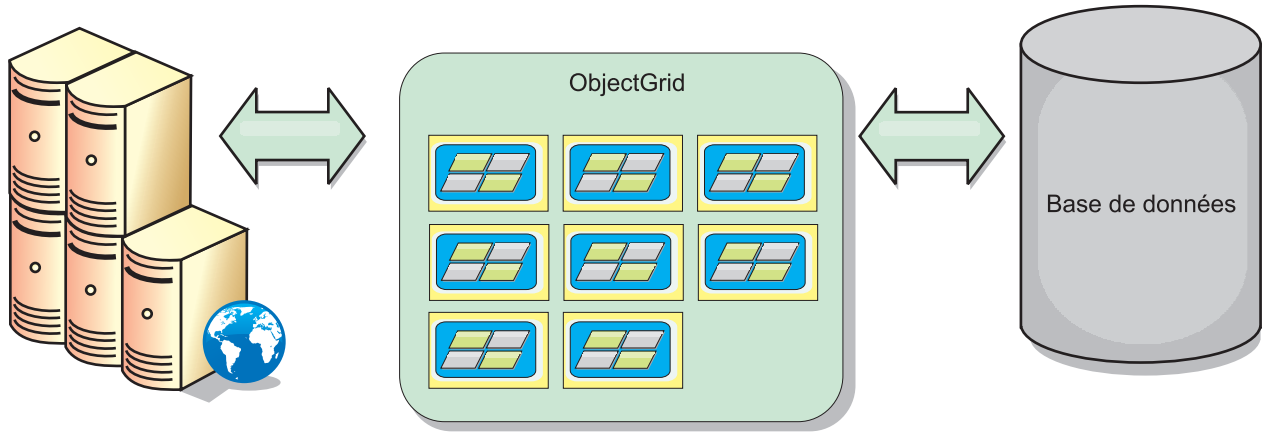


Figure 17. ObjectGrid en tant que mémoire tampon de base de données

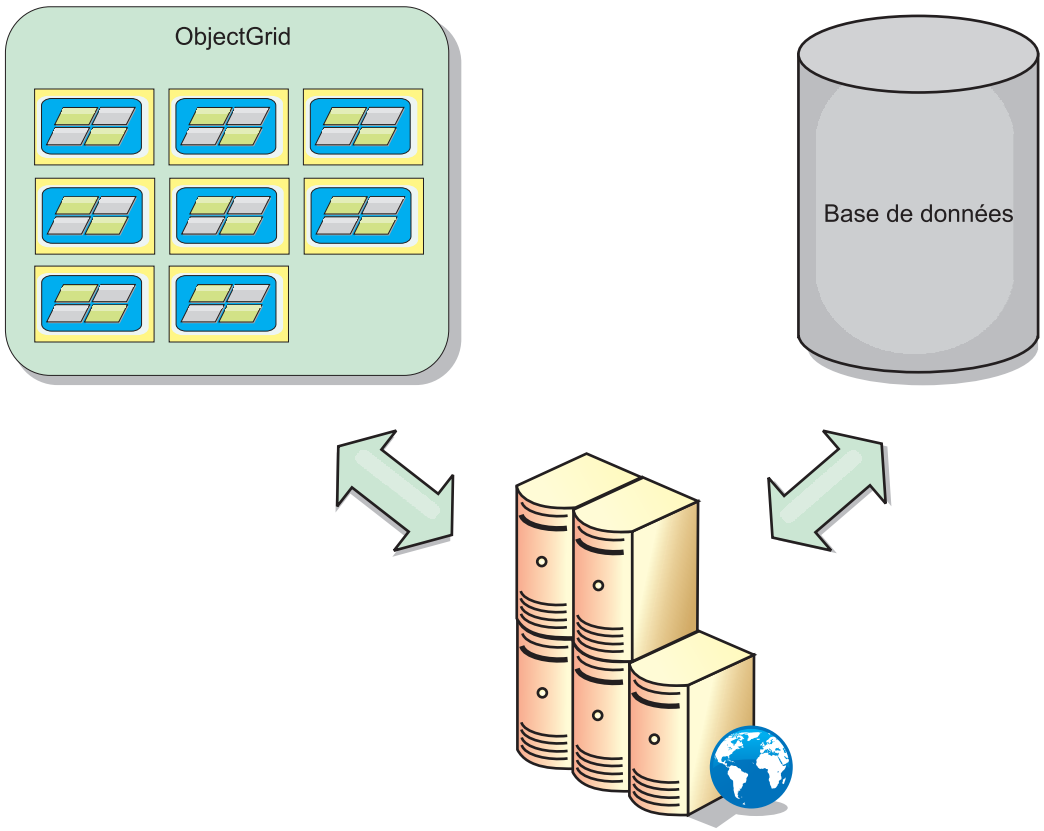


Figure 18. ObjectGrid en tant que cache secondaire

Cache partiel et cache complet

WebSphere eXtreme Scale peut s'utiliser en tant que cache partiel ou que cache complet. Un cache partiel ne conserve qu'un sous-ensemble des données totales, alors qu'un cache complet conserve toutes les données et peut être rempli en différé en fonction des besoins en données. Les caches partiels sont normalement accessibles à l'aide de clés (et non pas d'index ou de requêtes), car les données sont partiellement disponibles uniquement.

Cache partiel

Si une clé est absente dans un cache partiel ou que les données ne sont pas disponibles et qu'un échec de cache se produit, le niveau suivant est appelé. Les données sont extraites d'une base de données, par exemple, et elles sont insérées au groupe de caches de grille de données. Si vous utilisez une requête ou un index, seules les valeurs actuellement chargées sont accessibles et les requêtes ne sont pas transférées aux autres groupes.

Cache complet

Un cache complet comporte toutes les données requises et il est possible d'y accéder à l'aide d'attributs non-clés avec des index ou des requêtes. Un cache complet est préchargé avec des données de la base de données avant que l'application tente d'accéder aux données. Un cache complet peut fonctionner sous la forme d'un remplacement de base de données une fois que les données sont chargées. Etant donné que toutes les données sont disponibles, les requêtes et les

index peuvent être utilisés pour rechercher et agréger les données.

Cache secondaire

Lorsque WebSphere eXtreme Scale est utilisé en tant que cache secondaire, le système dorsal est utilisé avec la grille de données.

Cache secondaire

Vous pouvez configurer le produit en tant que cache secondaire pour la couche d'accès aux données d'une application. Dans ce scénario, WebSphere eXtreme Scale permet de stocker temporairement des objets qui seraient normalement extraits d'une base de données dorsale. Les applications vérifient si la grille de données contient les données. Si les données se trouvent dans la grille de données, ces données sont renvoyées à l'appelant. Si elles n'existent pas, elles sont extraites de la base de données dorsale. Elles sont ensuite insérées dans la grille de données afin que la demande suivante puisse utiliser la copie mise en cache. Le diagramme suivant montre comment WebSphere eXtreme Scale peut être utilisé en tant que cache secondaire à l'aide d'une couche d'accès aux données arbitraire, telle qu'OpenJPA ou Hibernate.

Plug-in de cache secondaire pour Hibernate et OpenJPA

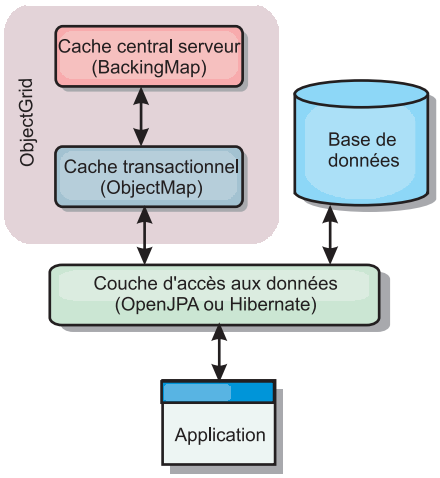


Figure 19. Cache secondaire

Les plug-in de cache pour OpenJPA et Hibernate sont inclus dans WebSphere eXtreme Scale pour que vous puissiez utiliser le produit comme cache secondaire automatique. L'utilisation d'WebSphere eXtreme Scale en tant que fournisseur de cache améliore les performances lors de la lecture et de l'interrogation des données et réduit la charge pesant sur la base de données. WebSphere eXtreme Scale présente plusieurs avantages par rapport à des implémentations de cache pré-intégrées car le cache est automatiquement répliqué entre tous les processus. Lorsqu'un client met une valeur en mémoire cache, tous les autres clients peuvent l'utiliser.

Cache en ligne

Vous pouvez configurer la mise en cache en ligne pour un système dorsal de base de données ou en tant que cache secondaire pour une base de données. La mise en cache en ligne utilise eXtreme Scale comme moyen principal pour interagir avec les

données. Lorsque eXtreme Scale est utilisé en tant que cache en ligne, l'application interagit avec le système dorsal à l'aide d'un plug-in Loader.

Cache en ligne

Lorsque WebSphere eXtreme Scale est utilisé en tant que cache en ligne, il interagit avec le système dorsal à l'aide d'un plug-in Loader. Ce scénario permet de simplifier l'accès aux données car les applications peuvent accéder aux API eXtreme Scale directement. Plusieurs scénarios de cache sont pris en charge dans eXtreme Scale pour assurer la synchronisation des données dans le cache et des données dans le système dorsal. Le diagramme suivant illustre l'interaction entre le cache en ligne, l'application et le système dorsal.

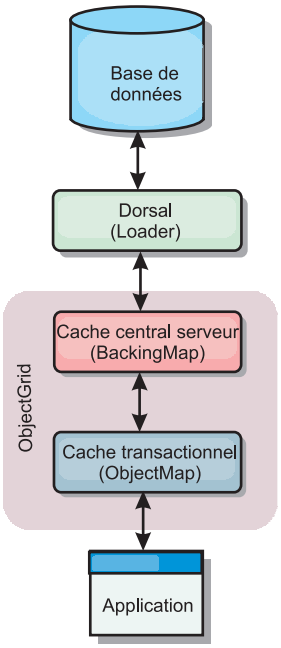


Figure 20. Cache en ligne

L'option de mise en cache en ligne simplifie l'accès aux données en permettant aux applications d'accéder directement aux API eXtreme Scale. WebSphere eXtreme Scale prend en charge plusieurs scénarios de mise en cache en ligne, comme suit.

- Sans interruption
- Ecriture immédiate
- Post-écriture

Scénario de mise en cache sans interruption

Un cache sans interruption est un cache partiel chargeant en lazy loading à partir d'une clé les entrées de données au fur et à mesure que ces entrées sont demandées. Cette opération peut se dérouler sans que l'appelant sache comment sont renseignées les entrées. Si les données sont introuvables dans le cache eXtreme Scale, eXtreme Scale récupère les données manquantes auprès du plug-in Loader qui charge les données provenant de la base de données d'arrière plan et les insère dans le cache. Les requêtes suivantes pour la même clé de données se trouveront dans le cache, jusqu'à ce qu'elles soient supprimées, invalidées ou expulsées.

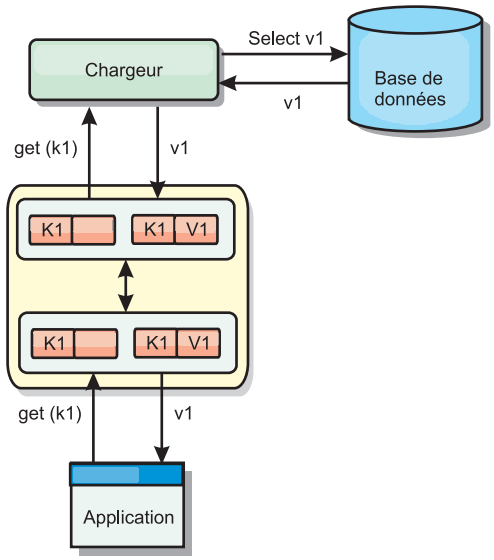


Figure 21. Mise en cache sans interruption

Scénario de mise en cache à écriture immédiate

Dans un cache à écriture immédiate, chaque écriture dans le cache est inscrite de manière synchrone dans la base de données à l'aide du chargeur. Cette méthode permet la cohérence avec le système dorsal, mais réduit les performances d'écriture étant donné que l'opération de base de données est synchrone. Le cache et la base de données étant tous deux mis à jour, les lectures suivantes à la recherche des mêmes données auront lieu dans le cache, évitant ainsi de faire appel à la base de données. Un cache à écriture immédiate est souvent utilisé conjointement à un cache sans interruption.

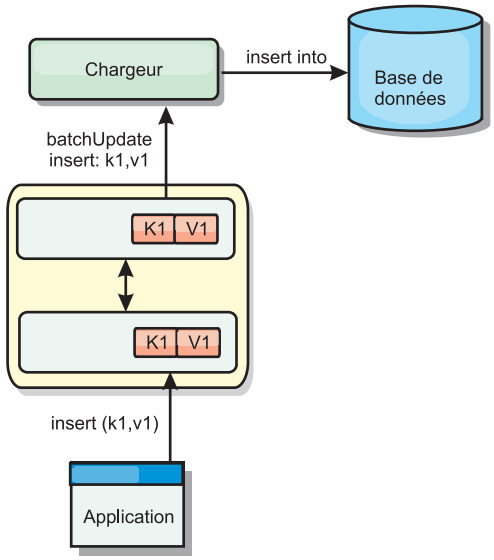


Figure 22. Mise en cache à écriture immédiate

Scénario de mise en cache en écriture différée

La synchronisation de la base de données peut être améliorée en écrivant les modifications de manière asynchrone. Cette opération est appelée mise en cache en

écriture différée. Les modifications, normalement écrites de manière synchrone dans le chargeur, sont mises en mémoire tampon dans eXtreme Scale et écrites dans la base de données à l'aide d'une unité d'exécution en arrière-plan. Les performances d'écriture sont considérablement améliorées, car l'opération de base de données est supprimée de la transaction client et les écritures de la base de données peuvent être comprimées.

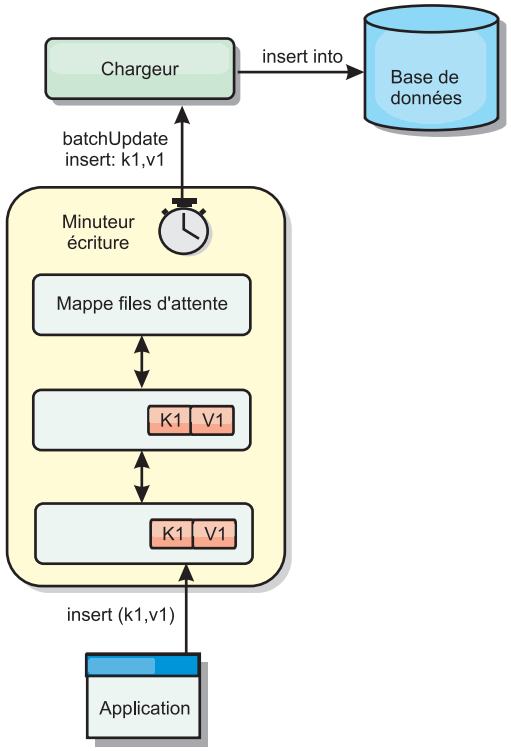


Figure 23. Mise en cache en écriture différée

Mise en cache en écriture différée

Vous pouvez utiliser la mise en cache en écriture différée pour réduire le temps système supplémentaire nécessaire lors de la mise à jour d'une base de données utilisée en tant que base de données dorsale.

Présentation de la mise en cache en écriture différée

La mise en cache en écriture différée met en file d'attente de manière asynchrone les mises à jour du plug-in Loader. Vous pouvez améliorer les performances en déconnectant les mises à jour, les insertions et les suppressions au sein d'une mappe, le temps système pour la mise à jour de la base de données dorsale. La mise à jour asynchrone est effectuée après un retard (de cinq minutes, par exemple) ou après un certain nombre d'entrées (1 000 entrées).

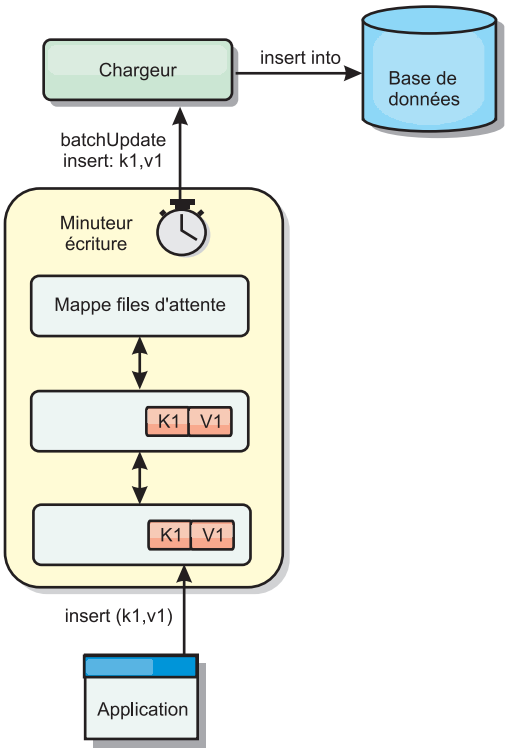


Figure 24. Mise en cache en écriture différée

La configuration à écriture différée sur une mappe de sauvegarde crée une unité d'exécution entre le Loader et la mappe. Le Loader délègue alors les demandes de données via l'unité d'exécution en fonction des paramètres de configuration de la méthode BackingMap.setWriteBehind. Lorsqu'une transaction eXtreme Scale insère, met à jour ou supprime une entrée dans une mappe, un objet LogElement est créé pour chacun de ces enregistrements. Ces éléments sont envoyés au Loader à écriture différée et mis en file d'attente dans une ObjectMap spéciale appelée mappe de files d'attente. Chaque mappe de sauvegarde pour laquelle le paramètre d'écriture différée est activé a ses propres mappes de files d'attente. L'unité d'exécution à écriture différée supprime périodiquement les données mises en file d'attente des mappes correspondantes et les insère dans le Loader dorsal.

Le chargeur à écriture différée envoie uniquement les types insertion, mise à jour et suppression des objets LogElement au chargeur réel. Tous les autres types, par exemple le type EVICT, sont ignorés.

La prise en charge de l'écriture différée *est* une extension du plug-in Loader, qui vous permet d'intégrer eXtreme Scale à la base de données. A ce sujet, vous pouvez consulter avec profit les explications Configuration des chargeurs JPA sur la configuration d 'un chargeur JPA.

Avantages

L'activation de l'écriture différée présente les avantages suivants :

- **Isolement en cas d'arrêt anormal de la base de données dorsale :** la mise en cache à écriture différée propose une couche d'isolement en cas d'arrêt anormal de la base de données dorsale. Les mises à jour sont alors placées dans la mappe de files d'attente. Les applications peuvent continuer à envoyer des transactions

vers eXtreme Scale. Lors de la reprise du système dorsal, les données contenues dans la mappe de files d'attente sont insérées dans celui-ci.

- **Réduction de la charge du système dorsal :** le chargeur à écriture différée fusionne les mises à jour en fonction des clés de façon qu'une seule mise à jour fusionnée par clé existe dans la mappe de files d'attente. Cette fusion diminue le nombre de mises à jour dans la base de données dorsale.
- **Amélioration des performances de la transaction :** la durée de chaque transaction eXtreme Scale est réduite car la transaction n'a plus à attendre que les données soient synchronisées avec le système dorsal.

Considérations liées à la conception d'applications

L'activation de l'écriture différée est une opération simple, mais la création d'une application devant utiliser l'écriture différée requiert une attention particulière. Sans écriture différée, la transaction ObjectGrid encadre la transaction dorsale. La transaction ObjectGrid démarre avant la transaction dorsale et se termine après celle-ci.

Lorsque la prise en charge de l'écriture différée est activée, la transaction ObjectGrid se termine avant le début de la transaction dorsale. La transaction ObjectGrid et la transaction dorsale sont dissociées.

Contraintes d'intégrité référentielle

Chaque mappe de sauvegarde configurée avec écriture différée dispose de sa propre unité d'exécution d'écriture différée pour envoyer les données vers le système dorsal. Les données mises à jour dans les différentes mappes d'une transaction ObjectGrid sont mises à jour dans le système dorsal via différentes transactions dorsales. Par exemple, la transaction T1 met à jour la clé key1 dans la mappe Map1 et la clé key2 dans la mappe Map2. La clé key1 mise à jour dans la mappe Map1 est actualisée vers le système dorsal dans une transaction expéditrice et la clé key2 actualisée dans la mappe Map2 l'est dans une autre transaction expéditrice ; cette mise à jour vers le dorsal est effectuée dans deux unités d'exécution différentes, toutes deux en écriture différée. Si les données stockées dans Map1 et Map2 ont des liens, tels que des contraintes de clé externe dans le système dorsal, les mises à jour sont susceptibles d'échouer.

Lors de la conception de contraintes d'intégrité référentielle dans votre base de données dorsale, vérifiez que les mises à jour désordonnées sont autorisées.

Verrouillage d'une mappe de files d'attente

Le comportement des transactions en matière de verrouillage constitue une autre différence notable. La grille d'objets prend en charge trois stratégies de verrouillage différentes : PESSIMISTIC, OPTIMISITIC et NONE. Les mappes de files d'attente à écriture différée utilisent la stratégie de verrouillage pessimiste, quelle que soit la stratégie de verrouillage configurée pour leur mappe de sauvegarde. Deux types différents d'opérations permettant d'acquérir un verrou sur la mappe de files d'attente existent :

- Lorsqu'une transaction ObjectGrid est validée ou qu'un vidage (de mappe ou de session) se produit, la transaction lit la clé de la mappe de files d'attente et place un verrou S sur la clé.
- Lorsqu'une transaction ObjectGrid est validée, elle tente de mettre à niveau le verrou S vers un verrou X sur la clé.

Ce comportement de mappe de files d'attente supplémentaire vous permet de voir quelques différences dans le comportement de verrouillage.

- Si la mappe de l'utilisateur est configurée comme stratégie de verrouillage pessimiste, la différence dans le comportement de verrouillage n'est pas grande. Chaque fois qu'un vidage ou qu'une validation est appelée, un verrou S est placé sur la même clé dans la mappe de files d'attente. Au moment de la validation, un verrou X est acquis pour la clé dans la mappe de l'utilisateur, mais également pour la clé dans la mappe de files d'attente.
- Si la mappe de l'utilisateur est configurée comme stratégie de verrouillage optimiste ou inexistante, la transaction utilisateur suit le modèle de la stratégie pessimiste. Chaque fois qu'un vidage ou qu'une validation est appelée, un verrou S est acquis sur la même clé dans la mappe de files d'attente. Au moment de la validation, un verrou X est acquis pour la clé dans la mappe de files d'attente utilisant la même transaction.

Nouvelles tentatives de transaction du chargeur

ObjectGrid ne prend pas en charge les transactions à 2 phases ou transactions XA. L'unité d'exécution à écriture différée supprime les enregistrements de la mappe de files d'attente et met à jour les enregistrements dans le système dorsal. En cas d'échec du serveur au milieu de la transaction, certaines mises à jour dorsales risquent d'être perdues.

Le chargeur à écriture différée tente automatiquement d'écrire à nouveau les transactions ayant échoué et envoie une LogSequence en attente de validation au système dorsal pour éviter toute perte de données. Pour que l'exécution de cette action soit possible, le chargeur doit être idempotent, ce qui signifie que lorsque Loader.batchUpdate(TxId, LogSequence) est appelé deux fois avec la même valeur, le résultat est le même que s'il était appelé une fois. Les implémentations du chargeur doivent implémenter l'interface RetryableLoader pour activer cette fonction. Pour plus de détails, consultez la documentation relative à l'API.

Echecs du chargeur

Le plug-in du chargeur (Loader) risque d'échouer lorsqu'il ne parvient pas à communiquer avec le dorsal de base de données. Cela peut se produire si le serveur de base de données ou la connexion réseau est arrêté(e). Le chargeur en écriture différée met en file d'attente les mises à jour et tente d'envoyer régulièrement les données modifiées au chargeur. Ce dernier doit signaler le problème de connectivité à l'environnement d'exécution ObjectGrid en générant une exception LoaderNotAvailableException.

L'implémentation du chargeur doit donc pouvoir distinguer un échec lié aux données d'une défaillance physique du chargeur. En cas d'échec lié aux données, une exception LoaderException ou OptimisticCollisionException doit être générée, alors qu'en cas de défaillance physique du chargeur, une exception LoaderNotAvailableException doit être générée. ObjectGrid gère ces deux exceptions de manière différente :

- Si une exception LoaderException est interceptée par le chargeur en écriture différée, celui-ci considère que l'échec est dû à une défaillance de données, telle qu'une erreur de clé en double. Le chargeur dégroupe la mise à jour et tente de ne mettre à jour qu'un enregistrement à la fois, afin d'isoler la défaillance de données. Si une exception {{LoaderException}} est à nouveau détectée lors de la mise à jour de l'enregistrement concerné, un enregistrement d'échec de la mise à jour est créé et consigné dans la mappe des mises à jour ayant échoué.

- Si une exception LoaderNotAvailableException est interceptée par le chargeur en écriture différée, celui-ci considère que l'échec est dû à l'impossibilité de se connecter à la base de données, par exemple, lorsque la base de données dorsale est inactive, lorsque la connexion à une base de données est indisponible ou lorsque le réseau est inactif. Le chargeur attend 15 secondes, puis tente à nouveau la mise à jour par lots de la base de données.

L'erreur courante est d'émettre une exception LoaderException à la place d'une exception LoaderNotAvailableException. Tous les enregistrements du chargeur à écriture différée deviennent alors des enregistrements d'échec de la mise à jour, ce qui réduit à néant l'objectif de l'isolement en cas d'arrêt anormal du système dorsal.

Remarques sur les performances

En supprimant de la transaction la mise à jour du chargeur, la mise en cache en écriture différée augmente le temps de réponse. Elle augmente également la capacité de traitement de la base de données, car les mises à jour de base de données sont combinées. Il est important de comprendre le temps système supplémentaire généré par l'unité d'exécution à écriture différée, qui permet de retirer les données de la mappe de files d'attente et de les insérer dans le chargeur.

Le temps de mise à jour maximal et le nombre de mises à jour maximal doivent être ajustés en fonction de l'environnement et des types d'utilisation prévus. Si la valeur du temps ou du nombre de mises à jour maximal est trop petite, le temps système de l'unité d'exécution d'écriture différée peut dépasser les avantages tirés. Définir une valeur élevée pour ces deux paramètres peut également augmenter l'utilisation de la mémoire pour la mise en file d'attente des données et retarder le moment de péremption des enregistrements de bases de données.

Pour des performances optimales, réglez les paramètres d'écriture différée en fonction des facteurs suivants :

- ratio des transactions de lecture et d'écriture
- fréquence de mise à jour d'enregistrements identiques
- temps d'attente pour la mise à jour de la base de données

Chargeurs

Avec un plug-in Loader, une mappe de grille de données peut se comporter comme un cache pour les données généralement conservées dans un magasin persistant sur le même système ou un autre système. Généralement, une base de données ou un système de fichiers est utilisé comme stockage de persistance. Une machine virtuelle Java (JVM) peut également être utilisée comme source des données, ce qui permet de créer des caches basés sur un concentrateur à l'aide d'eXtreme Scale. Un chargeur peut lire et écrire des données vers un stockage persistant ou à partir de celui-ci.

Présentation

Les chargeurs sont des plug-in de mappe de sauvegarde appelés lorsque des modifications sont apportées à la mappe de sauvegarde ou lorsque cette dernière est dans l'impossibilité de répondre à une demande de données (absence dans le cache). Le chargeur est appelé lorsque le cache ne peut pas satisfaire une demande de clé, offrant ainsi une fonction de lecture et un remplissage laborieux du cache. Un chargeur permet également les mises à jour de la base de données lorsque les valeurs du cache viennent à changer. Toutes les modifications dans une transaction

sont regroupées pour réduire le nombre d'interactions de base de données. Un plug-in TransactionCallback est utilisé conjointement avec le chargeur pour déclencher la démarcation de la transaction principale. L'utilisation de ce plug-in est importante lorsque plusieurs mappes sont incluses dans une seule transaction ou lorsque les données de transaction sont vidées dans le cache sans validation.

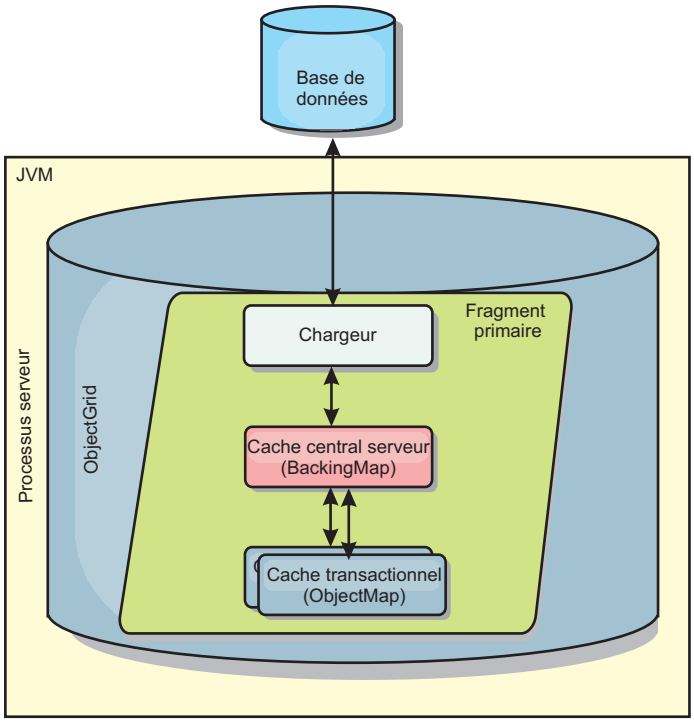


Figure 25. Chargeur

Le chargeur peut donc utiliser les mises à jour sur-qualifiées pour éviter le verrouillage intempestif de la base de données. En stockant un attribut de version dans la valeur du cache, le chargeur peut distinguer l'image de la valeur avant et après la mise à jour dans le cache. Cette valeur peut ensuite être utilisée lors de la mise à jour de la base de données ou du programme d'arrière plan pour vérifier que les données n'ont pas été mises à jour. Un chargeur peut également être configuré pour précharger la grille de données lorsqu'elle démarre. Lorsqu'elle est partitionnée, une instance de chargeur est associée à chaque partition. Si la mappe de la société comporte dix partitions, il existe dix instances de chargeur, une pour chaque partition principale. Lorsque le fragment primaire de la mappe est activé, la méthode preloadMap du chargeur est appelée de manière synchrone ou asynchrone, ce qui déclenche le chargement automatique de la partition de la mappe avec les données du programme d'arrière plan. Lorsqu'il est appelé de manière synchrone, toutes les transactions client sont bloquées, ce qui empêche tout accès incohérent à la grille de données. Sinon, un préchargeur client peut être utilisé pour charger l'intégralité de la grille de données.

Deux chargeurs pré-intégrés peuvent simplifier considérablement l'intégration aux dorsaux de bases de données relationnelles. Les chargeurs JPA utilisent les fonctions du mappage objet-relationnel(ORM) des implémentations OpenJPA et Hibernate des spécifications JPA (Java Persistence API). Pour plus d'informations, voir «Chargeurs JPA», à la page 67.

Si vous utilisez des chargeurs dans une configuration à plusieurs centre de données, vous devez étudier la façon dont les données de révision et la cohérence de la mémoire cache est conservée entre les grilles de données. Pour plus d'informations, voir «Remarques sur les chargeurs dans une topologie multimaître» , à la page 172.

Configuration de chargeur

Pour ajouter un chargeur à la configuration BackingMap, vous pouvez utiliser la configuration à l'aide d'un programme ou la configuration XML. Un chargeur a la relation suivante avec une mappe de sauvegarde.

- Une mappe de sauvegarde peut avoir un seul chargeur.
- Une mappe de sauvegarde client (cache local) ne peut pas avoir de chargeur.
- Une définition de chargeur peut être appliquée à plusieurs mappes de sauvegarde, mais chaque mappe de sauvegarde dispose de sa propre instance de chargeur.

Préchargement et préremplissage des données

Dans la plupart des scénarios qui utilise un chargeur, vous pouvez préparer la grille de données en y préchargeant ses données.

Lorsque vous utilisez la grille de données comme un cache complet, elle doit contenir toutes les données et elle doit être chargée pour que les clients puissent s'y connecter. Lorsque vous utilisez un cache partiel, vous pouvez préparer le cache avec des données pour que les clients puissent avoir accès immédiatement à ces données dès qu'ils se connectent.

Il existe deux approches pour pré-charger des données dans la grille de données ; vous pouvez utiliser un plug-in Loader ou un chargeur client, comme décrit dans les sections suivantes.

Plug-in Loader

Le plug-in Loader est associé à chaque mappe et chargé de synchroniser un fragment primaire de partition avec la base de données. La méthode preloadMap du plug-in Loader est invoquée automatiquement lors de l'activation d'un fragment, Par exemple, vous disposez de 100 partitions, il existe 100 instances Loader, chacune chargeant les données de sa partition. En cas d'exécution synchrone, tous les clients sont bloqués jusqu'à la fin du préchargement.

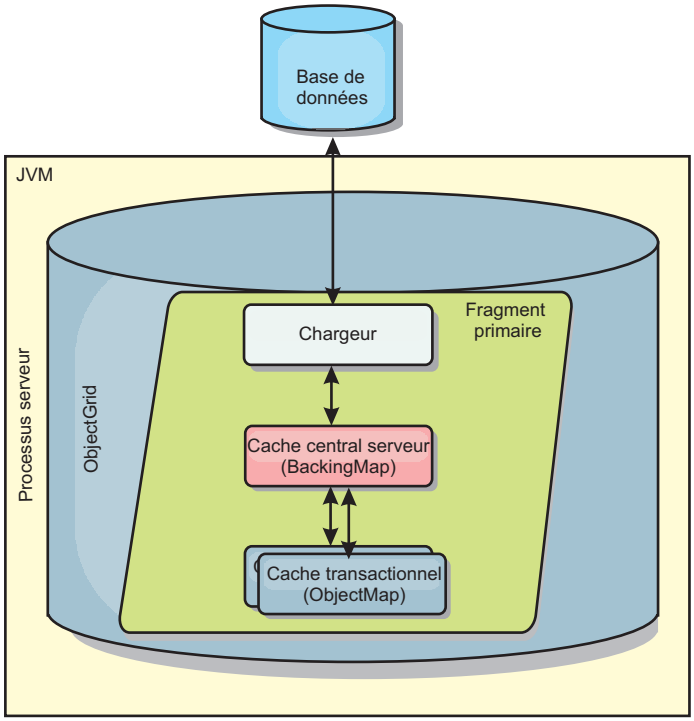


Figure 26. Plug-in Loader

Loader client

Un loader client est un pattern d'utilisation d'un ou plusieurs clients pour charger les données dans la grille. L'utilisation de plusieurs clients pour charger les données de la grille peut s'avérer efficace lorsque le schéma de partition n'est pas stocké dans la base de données. Vous pouvez appeler des chargeurs de client manuellement ou automatiquement lorsque la grille de données démarre. Ces chargeurs peuvent éventuellement utiliser StateManager pour faire passer la grille de données en mode de préchargement pour que les clients ne puissent pas accéder à la grille lorsqu'elle précharge les données. WebSphere eXtreme Scale contient un chargeur JPA (Java Persistence API) que vous pouvez utiliser pour charger automatiquement la grille de données avec le fournisseur JPA OpenJPA ou Hibernate. Pour plus d'informations sur les fournisseurs de cache, voir «Plug-in de cache niveau 2 (L2) JPA», à la page 26.

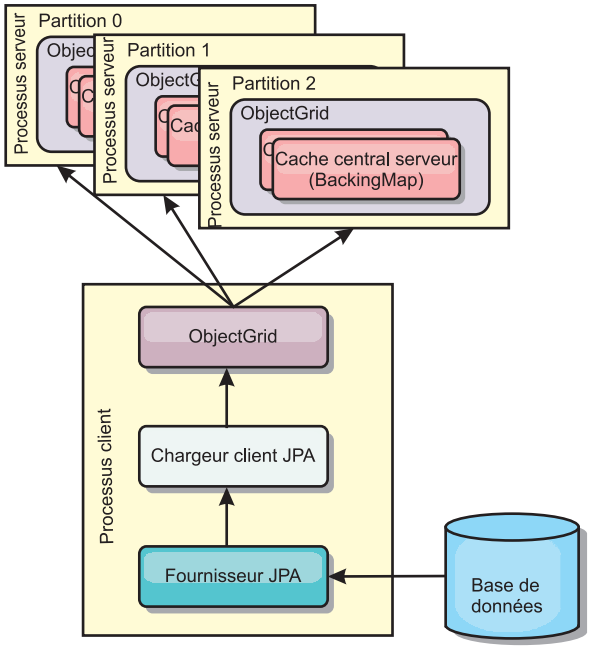


Figure 27. Loader client

Méthodes de synchronisation de base de données

Lorsque WebSphere eXtreme Scale est utilisé en tant que cache, les applications doivent être écrites de sorte qu'elles tolèrent les données périmées si la base de données peut être mise à jour de manière indépendante par rapport à une transaction eXtreme Scale. En tant qu'espace de traitement de base de données en mémoire synchronisé, eXtreme Scale permet d'assurer la mise à jour du cache de plusieurs manières.

Méthodes de synchronisation de base de données

Actualisation régulière

Le cache peut être régulièrement invalidé ou mis à jour de manière automatique à l'aide du programme de mise à jour temporelle de base de données JPA (Java Persistence API).Le programme de mise à jour interroge régulièrement la base de données à l'aide d'un fournisseur JPA, afin de rechercher des mises à jour ou des insertions survenues depuis la mise à jour précédente. Tous les changements détectés sont automatiquement invalidés ou mis à jour lorsqu'ils sont utilisés avec un cache incomplet. S'ils sont utilisés avec un cache complet, les entrées peuvent être détectées et insérées dans le cache. Les entrées ne sont jamais supprimées du cache.

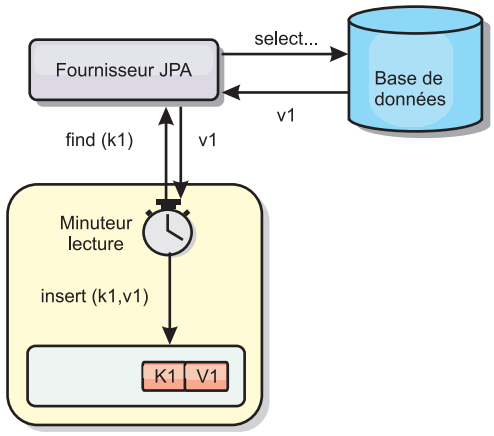


Figure 28. Actualisation régulière

Suppression

Les caches incomplets peuvent utiliser les stratégies de suppression pour supprimer automatiquement les données du cache sans que cela n'affecte la base de données. eXtreme Scale inclut trois stratégies : durée de vie, utilisation la moins récente et utilisation la moins fréquente. Si l'option de suppression en fonction de la mémoire est activée, ces trois stratégies suppriment les données de manière plus agressive à mesure que la mémoire est limitée.

Invalidation en fonction d'événements

Il est possible d'invalider les caches partiels et complets à l'aide d'un générateur d'événements comme JMS (Java Message Service). L'invalidation par le biais de JMS peut être associée de manière manuelle à tout processus qui met à jour le dorsal à l'aide d'un déclencheur de base de données. eXtreme Scale contient un plug-in JMS ObjectGridEventListener qui informe les clients des éventuelles modifications du cache du serveur. Cette procédure peut réduire la durée d'accès du client aux données périmées.

Invalidation par programme

Les API eXtreme Scale permettent l'interaction manuelle du cache local et du cache serveur à l'aide des méthodes des API Session.beginNoWriteThrough(), ObjectMap.invalidate() et EntityManager.invalidate(). Si un processus client ou serveur n'a plus besoin d'une partie des données, les méthodes d'invalidation peuvent être utilisées pour supprimer les données du serveur local ou du serveur cache. La méthode beginNoWriteThrough applique une opération ObjectMap ou EntityManager au cache local sans appeler le programme de chargement. Si l'opération est appelée à partir d'un client, elle s'applique uniquement au cache local (le programme de chargement distant n'est pas appelé). Si elle est appelée sur le serveur, l'opération s'applique uniquement au cache central du serveur sans appeler le programme de chargement.

L'invalidation des données

Pour supprimer les données de la mémoire cache d'évolution, vous pouvez utiliser un mécanisme d'invalidation basé sur les événements ou à l'aide d'un programme.

Invalidation basée sur les événements

Il est possible d'invalider les caches incomplets et complets à l'aide d'un générateur d'événements tel que Java Message Service (JMS). L'invalidation par le biais de JMS peut être associée de manière manuelle à tout processus qui met à jour le dorsal à l'aide d'un déclencheur de base de données. Un plug-in JMS ObjectGridEventListener est fourni dans eXtreme Scale pour permettre aux clients d'être informés des modifications dans le cache du serveur. Ce type de notification peut réduire la durée d'accès du client aux données obsolètes.

L'invalidation basée sur les événements est composée normalement des trois composants suivants.

- **File d'attente des événements :** une file d'attente d'événements stocke les événements de modification des données. Il peut s'agir d'une file d'attente JMS, d'une base de données, d'une file d'attente interne premier entré premier sorti ou de tout autre événement dans la mesure où elle peut gérer les événements de modification des données.
- **Publicateur d'événements :** un publicateur d'événements publie les événements de modification de données dans la file d'attente d'événements. Une publicateur d'événements est généralement une application que vous créez ou une implémentation de plug-in eXtreme Scale. Il sait quand les données ont été modifiées ou il modifie les données lui-même. Lorsqu'une transaction est validée, les événements sont générés pour les données modifiées et le publicateur d'événements publie ces événements dans la file d'attente d'événements.
- **Consommateur d'événements :** un consommateur d'événements consomme les événements de modification de données. Le consommateur d'événements est généralement une application permettant de vérifier la mise à jour des données de la grille cible avec les dernières modifications apportées aux autres grilles. Il interagit avec la file d'attente d'événements pour récupérer les dernières données et applique les modifications apportées aux données dans la grille cible. Les consommateurs d'événements peuvent utiliser les API eXtreme Scale pour invalider les données obsolètes ou mettre à jour la grille avec les dernières données.

Par exemple, JMSObjectGridEventListener comporte une option pour un modèle client-serveur dans lequel la file d'attente d'événements est une destination JMS désignée. Tous les processus serveur sont des publicateurs d'événements. Lorsqu'une transaction est validée, le serveur récupère les modifications apportées aux données et les publie à la destination JMS désignée. Tous les processus client sont des consommateurs d'événements. Ils reçoivent les modifications apportées aux données de la destination JMS désignée et appliquent les modifications au cache local du client.

Pour plus d'informations, consultez la rubrique relative au mécanisme d'invalidation de client dans le *Guide de l'administration* .

Invalidation par programme

Les API WebSphere eXtreme Scale autorise l'interaction manuelle du cache local et du cache serveur à l'aide des méthodes Session.beginNoWriteThrough(), ObjectMap.invalidate() et EntityManager.invalidate(). Si un processus client ou serveur n'a plus besoin d'une partie des données, les méthodes invalidate permettent de supprimer des données d'un cache local ou de serveur. La méthode beginNoWriteThrough applique toutes les opérations ObjectMap ou EntityManager

au cache local sans appeler le chargeur. Si l'opération est appelée à partir d'un client, elle s'applique uniquement au cache local (le programme de chargement distant n'est pas appelé). Si elle est appelée sur le serveur, l'opération s'applique uniquement au cache central du serveur sans appeler le programme de chargement.

Vous pouvez utiliser l'invalidation par programme à l'aide d'autres techniques pour déterminer quand il convient d'invalider les données. Par exemple cette méthode d'invalidation utilise des mécanismes d'invalidation basée sur les événements pour recevoir les événements de modification de données, puis utilise les API pour invalider les données obsolètes.

Indexation

Utilisez le plug-in MapIndexPlugin pour générer un ou plusieurs index dans une mappe BackingMap pour prendre en charge l'accès aux données ne correspondant pas à une clé.

Types d'indexation et configuration d'index

L'indexation est représentée par le plug-in MapIndexPlugin ou Index, en bref. Index est un plug-in BackingMap. Une mappe de sauvegarde peut avoir plusieurs index configurés, dès lors que chacun d'entre eux respecte les règles de configuration d'index.

Vous pouvez utiliser l'indexations pour générer une ou plusieurs index dans une mappe BackingMap. Un index se construit à partir d'un attribut ou d'une liste des attributs d'un objet de la mappe. L'indexation permet aux applications de trouver plus rapidement certains objets. Grâce à elle, en effet, les applications peuvent trouver les objets dont les attributs indexés ont une certaine valeur ou se situent dans une plage de valeurs.

Deux types d'indexation sont possibles : statiques et dynamiques. L'indexation statique oblige à configurer le plug-in d'indexation index dans la mappe de sauvegarde avant d'initialiser l'instance ObjectGrid. Comme pour la mappe de sauvegarde, cela peut se faire par programmation ou via XML. L'indexation statique commence à générer l'index pendant l'initialisation de la grille d'objets. L'index est synchrone en permanence avec la mappe de sauvegarde et il est prêt à être utilisé. Après que l'indexation statique a démarré, la maintenance de l'index fait partie de la gestion des transactions par eXtreme Scale. Lorsque les transactions valident leurs modifications, ces dernières actualisent également l'index statique et les modifications apportées à l'index sont annulées en cas d'annulation de la transaction.

L'indexation dynamique permet de créer un index dans une mappe de sauvegarde avant ou après l'initialisation de l'instance ObjectGrid qui contient cette mappe. Les applications contrôlent le cycle de vie de l'indexation dynamique, ce qui permet de supprimer un index dynamique devenu inutile. Lorsqu'une application crée un index dynamique, cet index n'est pas forcément utilisable immédiatement en raison du temps que met à s'effectuer la génération complète de l'index. Comme la durée dépend de la quantité de données indexées, l'interface DynamicIndexCallback est fournie pour les applications qui souhaitent recevoir des notifications lorsque se produisent certains événements l'indexation, à savoir les événements ready, error et destroy. Les applications peuvent implémenter cette interface de rappel et s'enregistrer auprès de l'indexation dynamique.

Si un plug-in d'indexation est configuré pour une mappe de sauvegarde, il est possible d'obtenir de la mappe d'objet correspondante l'objet proxy de l'index. L'appel de la méthode getIndex dans la mappe et la transmission du nom du plug-in Index renvoie l'objet proxy de l'index. L'objet proxy doit être transtypé vers l'interface d'indexation de l'application utilisée, MapIndex, MapRangeIndex ou une interface d'indexation personnalisée, par exemple. Une fois l'objet proxy obtenu, l'on peut utiliser les méthodes définies dans l'interface d'indexation de l'application afin de trouver des objets mis en cache.

La liste qui suit récapitule la procédure à appliquer pour procéder à l'indexation :

- ajout d'index statiques ou dynamiques dans la mappe de sauvegarde
- obtention d'un objet proxy d'index grâce à la méthode getIndex de la mappe d'objet
- transtypage de l'objet proxy vers l'interface d'indexation de l'application utilisée (MapIndex, MapRangeIndex ou une interface d'indexation personnalisée, par exemple)
- utilisation des méthodes qui sont définies dans l'interface d'indexation de l'application pour rechercher les objets mis en cache

La classe HashIndex est l'implémentation du plug-in d'indexation pré-intégré capable de prendre en charge les deux interfaces pré-intégrées d'API d'indexation : MapIndex et MapRangeIndex. Vous pouvez également créer vos propres index. Vous pouvez ajouter HashIndex à la BackingMap en tant qu'index statique ou dynamique, obtenir un objet proxy d'index MapIndex ou MapRangeIndex et utiliser cet objet proxy pour chercher des objets mis en cache.

Index par défaut

Si vous souhaitez effectuer une itération dans les clés d'une mappe locale, vous pouvez utiliser l'index par défaut. Cet index ne requiert pas de configuration, mais elle doit être utilisée sur le fragment en utilisant un agent ou une instance ObjectGrid extraite de la méthode ShardEvents.shardActivated(ObjectGrid shard).

Indexation et qualité des données obtenues par une requête d'index

Il faut bien avoir présent à l'esprit que les méthodes de requêtes sur les index ne représentent qu'un cliché des données à un instant t. Les entrées de données ne sont pas verrouillées après l'envoi à l'application des résultats de la requête. L'application doit être consciente que les données peuvent très bien être actualisées après lui avoir été retournées. Supposons, par exemple, que l'application obtienne la clé d'un objet mis en cache grâce à la méthode findAll de MapIndex. Cet objet key retourné est associé dans le cache à une entrée de données. L'application doit être capable d'exécuter la méthode get sur la mappe d'objet pour trouver un objet à partir de l'objet key. Si une autre transaction supprime du cache l'objet données juste avant l'appel à la méthode get, le résultat qui sera retourné sera null.

Points à prendre en considération à propos des performances de l'indexation

L'un des objectifs primordiaux de l'indexation est d'améliorer les performances globales de la mappe de sauvegarde. Une utilisation incorrecte de l'indexation peut compromettre les performances de l'application. Avant d'utiliser l'indexation, les facteurs suivants sont à prendre en considération :

- **Le nombre de transactions simultanées en écriture :** l'indexation peut se produire chaque fois qu'une transaction écrit des données dans une mappe de sauvegarde. Les performances se dégradent si un grand nombre de transactions écrivent en même temps des données dans la mappe au moment où une application lance des requêtes sur l'index.
- **La taille des résultats retournés par une requête :** les performances de la requête déclinent d'autant plus que la taille de ses résultats augmente. Les performances tendent à se dégrader lorsque la taille des résultats atteint 15 % ou plus de la mappe de sauvegarde.
- **Le nombre d'index générés sur la même mappe de sauvegarde :** chaque index consomme des ressources système. Les performances diminuent au fur et à mesure que le nombre d'index augmente sur la mappe de sauvegarde.

Cela dit, l'indexation peut augmenter considérablement les performances des mappes de sauvegarde. C'est particulièrement vrai lorsque la mappe de sauvegarde comporte surtout des opérations de lecture. Les résultats des requêtes représentent alors un faible pourcentage des entrées de la mappe et seul un petit nombre d'index sont générés sur la mappe.

Chargeurs JPA

La Java Persistence API (JPA) est une spécification de mappage des objets Java à des bases de données relationnelles. JPA contient une spécification ORM (Object-Relational Mapping) complète utilisant des annotations de métadonnées de langage Java, des descripteurs XML ou les deux pour définir le mappage entre les objets Java et une base de données relationnelle. Un certain nombre d'implémentations commerciales et de code source ouvert sont disponibles.

Vous pouvez utiliser une implémentation de plug-in de chargeur Java Persistence API (JPA) avec eXtreme Scale pour interagir avec les bases de données prises en charge par le chargeur choisi. Pour utiliser JPA, vous devez disposer d'un fournisseur JPA pris en charge, tel qu'OpenJPA ou Hibernate, des fichiers JAR et un fichier `META-INF/persistence.xml` dans votre chemin d'accès aux classes.

Les plug-in JPALoader com.ibm.websphere.objectgrid.jpa.JPALoader et JPAEntityLoader com.ibm.websphere.objectgrid.jpa.JPAEntityLoader sont deux plug-in pré-intégrés de chargeur JPA qui permettent de synchroniser les mappes ObjectGrid avec une base de données. Pour utiliser cette fonction, vous devez disposer d'une implémentation JPA, comme Hibernate ou OpenJPA. La base de données peut correspondre à tout programme d'arrière plan prise en charge par le fournisseur JPA choisi.

Vous pouvez utiliser le plug-in JPALoader lorsque vous stockez des données à l'aide de l'API ObjectMap. Utilisez le plug-in JPAEntityLoader lorsque vous stockez des données à l'aide de l'API EntityManager.

Architecture du chargeur JPA

Le chargeur JPA est utilisé pour les mappes eXtreme Scale qui stockent des objets Java simples (Plain Old Java Objects - POJO).

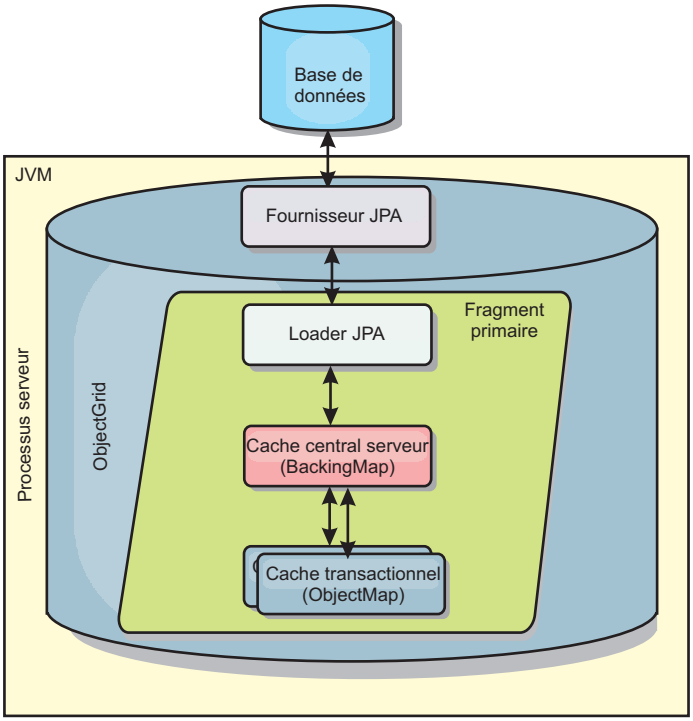


Figure 29. Architecture du chargeur JPA

Lorsqu'une méthode ObjectMap.get(Object key) est appelée, l'exécution eXtreme Scale vérifie tout d'abord si l'entrée est contenue dans la couche de l'ObjectMap. Si ce n'est pas le cas, l'exécution délègue la demande au chargeur JPA. Sur demande de chargement de la clé, JPALoader appelle la méthode JPA EntityManager.find(Object key) pour trouver les données à partir du chargeur JPA. Si les données sont contenues dans le gestionnaire d'entités JPA, elles sont renvoyées ; dans le cas contraire, le fournisseur JPA interagit avec la base de données pour obtenir la valeur.

Lors d'une mise à jour de l'ObjectMap, par exemple à l'aide de la méthode ObjectMap.update(Object key, Object value), l'exécution eXtreme Scale crée un élément LogElement pour cette mise à jour et l'envoie au JPALoader. Le JPALoader appelle la méthode JPA EntityManager.merge(Object value) pour mettre à jour la valeur dans la base de données.

Le plug-in JPAEntityLoader implique les quatre mêmes couches. Toutefois, étant donné que le plug-in JPAEntityLoader est utilisé pour les mappes qui stockent des entités eXtreme Scale, les relations entre les entités peuvent compliquer le scénario d'usage. Une entité eXtreme Scale se distingue d'une entité JPA. Pour plus d'informations, voir les informations relatives au plug-in JPAEntityLoader dans *Guide de programmation*.

Méthodes

Les chargeurs présentent trois méthodes principales :

1. get : renvoie une liste de valeurs qui correspond à l'ensemble des clés qui sont transmises par l'extraction des données à l'aide de JPA. La méthode utilise JPA pour trouver les entités dans la base de données. Pour le plug-in JPALoader, la liste renvoyée contient une liste des entités JPA extraite directement de

l'opération find. Pour le plug-in JPAEntityLoader, la liste renvoyée contient des tuples de valeur d'entité eXtreme Scale qui sont convertis à partir des entités JPA.

2. batchUpdate : écrit les données des mappes ObjectGrid dans la base de données. En fonction des différents types d'opérations (insert, update ou delete), le chargeur utilise les opérations JPA persist, merge et remove pour mettre à jour les données dans la base de données. Pour le JPALoader, les objets de la mappe sont directement utilisés en tant qu'entités JPA. Pour le JPAEntityLoader, les tuples d'entité de la mappe sont convertis en objets utilisés en tant qu'entités JPA.
3. preloadMap : précharge la mappe à l'aide de la méthode de chargeur client ClientLoader.load. Pour les mappes partitionnées, la méthode preloadMap est uniquement appelée dans une seule partition. La partition est spécifiée par la propriété preloadPartition de la classe JPALoader ou JPAEntityLoader. Si la valeur preloadPartition est inférieure à zéro ou supérieure à (*nombre_total_de_partitions* - 1), le préchargement est désactivé.

Les plug-in JPALoader et JPAEntityLoader s'associent à la classe JPATxCallback pour coordonner les transactions eXtreme Scale et les transactions JPA. La classe JPATxCallback doit être configurée dans l'instance ObjectGrid pour utiliser ces deux chargeurs.

Configuration et programmation

Si vous utilisez des chargeurs JPA dans un environnement multi-maître, voir «Remarques sur les chargeurs dans une topologie multimaître», à la page 172. Pour plus d'informations sur la configuration des chargeurs JPA, voir les informations sur les chargeurs JPA dans *Guide d'administration*. Pour plus d'informations sur la programmation des chargeurs JPA, reportez-vous au *Guide de programmation*.

Présentation de la sérialisation

Les données sont toujours exprimées, mais pas nécessairement stockées, comme les objets Java dans la grille de données. WebSphere eXtreme Scale utilise plusieurs processus Java pour sérialiser les données en convertissant les instances d'objet Java en octets, puis de nouveau en objets, si nécessaire, pour transférer les données entre les processus client et serveur.

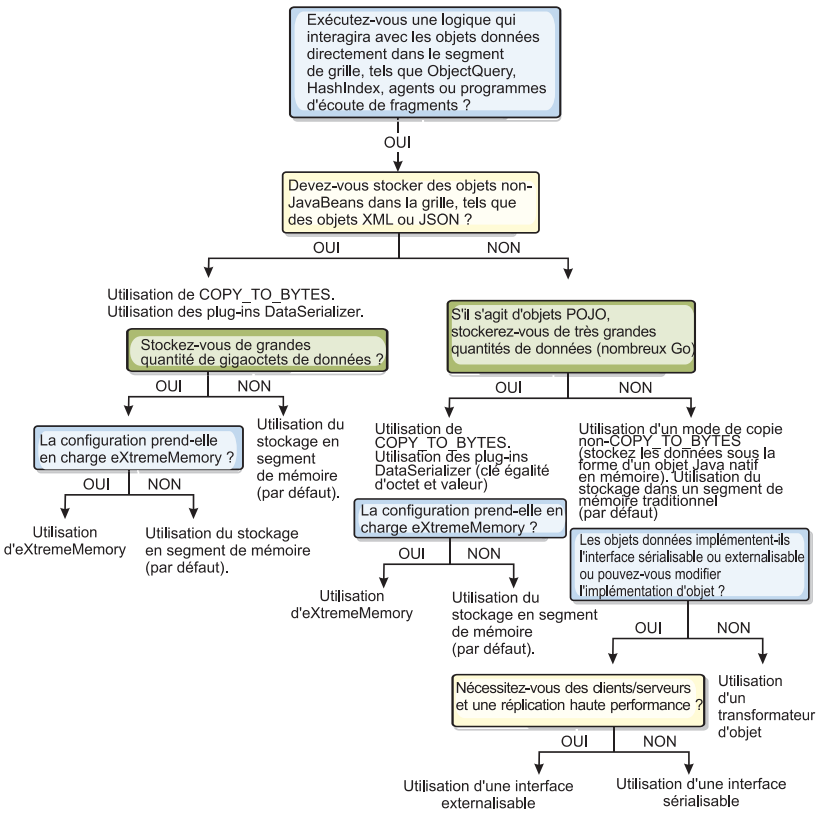
Les données sont sérialisées lorsqu'elles sont converties en flux de données pour la transmission dans un réseau dans les cas suivants :

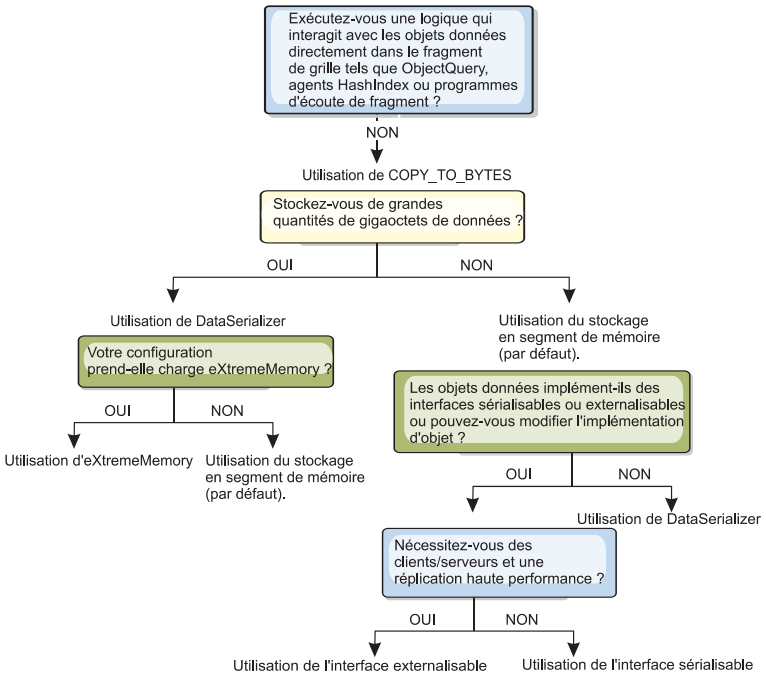
- Lorsque les clients communiquent avec les serveurs et que les serveurs renvoient les informations au client.
- Lorsque les serveurs répliquent d'un serveur vers un autre.

Vous pouvez également abandonner le processus de sérialisation via WebSphere eXtreme Scale et stocker les données sous forme de tableaux d'octets. Les tableaux d'octets sont beaucoup plus économiques à stocker en mémoire, car la machine JVM (Java Virtual Machine) doit rechercher moins d'objets pour récupérer de la place et ils peuvent être désérialisés lorsque cela est nécessaire. Utilisez des tableaux d'octets uniquement si vous ne devez pas accéder aux objets en utilisant des requêtes ou des index. Comme les données sont stockées sous forme d'octets, eXtreme Scale ne dispose pas de métadonnées pour décrire les attributs à interroger.

Pour sérialier les données dans eXtreme Scale, vous pouvez utiliser la sérialisation Java, le pug-in ObjectTransformer ou les plug-ins DataSerializer. Pour optimiser la sérialisation avec ces options, vous pouvez utiliser le mode COPY_TO_BYTES pour améliorer les performances jusqu'à 70 %, car les données sont sérialisées lorsque les transactions sont validées, ce qui implique que la sérialisation n'a lieu qu'une seule fois. Les données sérialisées sont envoyées du client au serveur ou du serveur au serveur répliqué. En utilisant le mode COPY_TO_BYTES, vous pouvez réduire la mémoire que peut occuper un grand tableau d'objets.

Utilisez les figures suivantes pour déterminer le type de méthode de sérialisation la mieux adaptée à vos besoins de développement. La première figure décrit les méthodes de sérialisation disponibles lorsque vous exécutez la logique qui interagit avec les objets de données directement dans le fragment de grille. La dernière figure montre les options disponibles lorsque vous n'interagissez pas directement avec le fragment de grille.





Pour plus d'informations sur les formes de sérialisation prises en charge dans le produit eXtreme Scale , voir les rubriques suivantes :

Sérialisation à l'aide de Java

La sérialisation Java fait référence à une sérialisation par défaut qui utilise l'interface Serializable ou à la sérialisation personnalisée qui utilise à la fois les interfaces Serializable et Externalizable.

Sérialisation par défaut

Pour utiliser la sérialisation par défaut, implémentez l'interface java.io.Serializable qui contient l'API qui convertit des objets en octets, qui sont ensuite désérialisés. Utilisez la classe java.io.ObjectOutputStream pour rendre l'objet persistant. Ensuite, appelez la méthode ObjectOutputStream.writeObject() pour initier la sérialisation et mettre à plat l'objet Java.

Sérialisation personnalisée

Dans certains cas, des objets doivent être modifiés pour utiliser la sérialisation personnalisée, telle que pour l'implémentation de l'interface java.io.Externalizable ou en implémentant des méthodes writeObject et readObject pour les classes qui mettent en oeuvre l'interface java.io.Serializable. Les techniques de sérialisation personnalisée doivent être employées lorsque les objets sont sérialisés à l'aide de mécanismes autres que les méthodes de l'API ObjectGrid ou EntityManager.

Par exemple, lorsque les objets ou entités sont stockés en tant que données d'instance dans un agent d'API DataGrid ou lorsque l'agent renvoie des objets ou des entités, ces objets ne sont transformés à l'aide d'un ObjectTransformer. Toutefois, l'agent fait automatiquement appel à l'ObjectTransformer lors de l'utilisation de l'interface `EntityMixin`. Pour plus de détails, voir Agents DataGrid et mappes basées sur les entités.

Plug-in ObjectTransformer

Le plug-in ObjectTransformer permet de sérialiser, désérialiser et copier des objets du cache afin d'améliorer les performances.

 L'interface ObjectTransformer a été remplacée par les plug-ins DataSerializer que vous pouvez utiliser pour stocker efficacement les données arbitraires dans WebSphere eXtreme Scale pour que les API de produit existantes puissent interagir efficacement avec vos données.

Si vous constatez des problèmes de performances dans l'utilisation des processeurs, ajoutez à chaque mappe un plug-in ObjectTransformer. Sans ce plug-in ObjectTransformer, jusqu'à 60-70 % du temps processeur sera consacré à la sérialisation et à la copie des entrées.

Utilité

Le plug-in ObjectTransformer permet à vos applications de fournir des méthodes personnalisées pour les opérations suivantes :

- sérialisation ou désérialisation de la clé d'une entrée
- sérialisation ou désérialisation de la valeur d'une entrée
- copie de la clé ou de la valeur d'une entrée

Si aucun plug-in ObjectTransformer n'est fourni, vous devrez savoir sérialiser vous-mêmes les clés et les valeurs car l'ObjectGrid utilise une séquence de sérialisation/désérialisation pour copier les objets. Cette méthode est onéreuse ; c'est pourquoi il convient d'utiliser un plug-in ObjectTransformer lorsque les performances sont en jeu. La copie ne se produit que lorsqu'une application recherche pour la première fois un objet dans une transaction. Vous pouvez éviter la copie en donnant au mode copy de la mappe la valeur NO_COPY ou réduisant la copie en donnant à ce mode la valeur COPY_ON_READ. Optimisez l'opération de copie lorsque l'application a besoin d'en effectuer une en fournissant une méthode personnalisée de copie dans ce plug-in. Ce plug-in peut faire tomber le temps système consacré à la copie de 65–70 % à 2-3 % du temps processeur total.

La première fois, les implémentations par défaut des méthodes copyKey et copyValue tentent d'utiliser la méthode clone si cette méthode est fournie. Si aucune implémentation de clone n'est fournie, par défaut, l'implémentation passe à la sérialisation.

La sérialisation des objets est également utilisée directement lorsque eXtreme Scale s'exécute en mode réparti. LogSequence utilise le plug-in ObjectTransformer pour sérialiser les clés et les valeurs avant de transmettre les modifications aux homologues présents dans l'ObjectGrid. Vous devez prendre un certain nombre de précautions lorsque vous fournissez une méthode personnalisée de sérialisation au lieu d'utiliser la sérialisation pré-intégrée du kit de développement Java. La vérification des versions d'objets est en effet un problème complexe et vous risquez de rencontrer des problèmes de compatibilité de versions si vos méthodes personnalisées ne sont pas conçues pour gérer cette vérification.

La liste qui suit décrit comment eXtreme Scale s'y prend pour sérialiser les clés et les valeurs :

- Si un plug-in ObjectTransformer personnalisé est écrit et connecté, eXtreme Scale appelle les méthodes présentes dans l'interface ObjectTransformer pour sérialiser les clés et les valeurs et pour obtenir des copies de ces clés et de ces valeurs.

- S'il n'est pas fait usage d'un plug-in ObjectTransformer personnalisé, eXtreme Scale sérialise et désérialise les valeurs conformément à la méthode par défaut. Si c'est le Plug-in par défaut qui est utilisé, chaque objet est implémenté comme externalisable ou comme sérialisable.
  - Si l'objet prend en charge l'interface Externalizable, c'est la méthode writeExternal qui est appelée. Les objets implémentés comme externalisables donnent de meilleures performances.
  - Si l'objet ne prend pas en charge l'interface Externalizable et qu'il implémente l'interface Serializable, il est enregistré à l'aide de la méthode ObjectOutputStream.

Utiliser l'interface ObjectTransformer

Un objet ObjectTransformer doit implémenter l'interface ObjectTransformer et se conformer aux conventions communes des plug-in ObjectGrid.

Comme toujours, deux approches sont possibles pour ajouter un objet ObjectTransformer à la configuration BackingMap : la configuration par programmation et la configuration XML.

Configuration par programmation du plug-in ObjectTransformer

Le fragment de code suivant crée l'objet ObjectTransformer personnalisé et l'ajoute à une BackingMap :

```
ObjectGridManager objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid myGrid = objectGridManager.createObjectGrid("myGrid", false);
BackingMap backingMap = myGrid.getMap("myMap");
MyObjectTransformer myObjectTransformer = new MyObjectTransformer();
backingMap.setObjectTransformer(myObjectTransformer);
```

Configuration par XML du plug-in ObjectTransformer

Supposons que le nom de la classe de l'implémentation d'ObjectTransformer soit com.company.org.MyObjectTransformer. Cette classe implémente l'interface ObjectTransformer. Le code XML suivant permet de configurer une implémentation d'ObjectTransformer :

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
    xmlns="http://ibm.com/ws/objectgrid/config">
    <objectGrids>
        <objectGrid name="myGrid">
     <backingMap name="myMap" pluginCollectionRef="myMap" />
        </objectGrid>
    </objectGrids>

    <backingMapPluginCollections>
 <backingMapPluginCollection id="myMap">
      <bean id="ObjectTransformer" className="com.company.org.MyObjectTransformer" />
 </backingMapPluginCollection>
    </backingMapPluginCollections>
</objectGridConfig>
```

Scénarios d'utilisation d'ObjectTransformer

Vous pouvez utiliser le plug-in ObjectTransformer dans les situations suivantes :

- objet non sérialisable
- objet sérialisable mais nécessité d'améliorer les performances de la sérialisation
- copie de clés ou de valeurs

Dans l'exemple qui suit, l'ObjectGrid sert à stocker la classe Stock :

```
/**
* Objet Stock pour la démo ObjectGrid
*
*
*/
public class Stock implements Cloneable {
    String ticket;
    double price;
    String company;
    String description;
    int serialNumber;
    long lastTransactionTime;
    /**
    * @return retourne la description.
    */
    public String getDescription() {
        return description;
    }
    /**
    * @param description La description à définir.
    */
    public void setDescription(String description) {
        this.description = description;
    }
    /**
    * @return Retourne le lastTransactionTime.
    */
    public long getLastTransactionTime() {
        return lastTransactionTime;
    }
    /**
    * @param lastTransactionTime Le lastTransactionTime à définir.
    */
    public void setLastTransactionTime(long lastTransactionTime) {
        this.lastTransactionTime = lastTransactionTime;
    }
    /**
    * @return Retourne le prix.
    */
    public double getPrice() {
        return price;
    }
    /**
    * @param price Le prix à définir.
    */
    public void setPrice(double price) {
        this.price = price;
    }
    /**
    * @return Retourne le serialNumber.
    */
    public int getSerialNumber() {
        return serialNumber;
    }
    /**
    * @param serialNumber Le serialNumber à définir.
    */
    public void setSerialNumber(int serialNumber) {
        this.serialNumber = serialNumber;
    }
    /**
    * @return Retourne le ticket.
    */
    public String getTicket() {
        return ticket;
    }
    /**
    * @param ticket Le ticket à définir.
    */
    public void setTicket(String ticket) {
        this.ticket = ticket;
    }
    /**
    * @return Retourne la Company.
    */
    public String getCompany() {
        return company;
    }
    /**
    * @param company La Company à définir.
```

```
    */
    public void setCompany(String company) {
        this.company = company;
    }
    //clone
    public Object clone() throws CloneNotSupportedException
    {
        return super.clone();
    }
}
```

Vous pouvez écrire une classe ObjectTransformer personnalisée pour la classe Stock :

```
/**
* Implémentation personnalisée d'ObjectGrid ObjectTransformer pour l'objet Stock
*
*/
public class MyStockObjectTransformer implements ObjectTransformer {
/* (non-Javadoc)
* @see
* com.ibm.websphere.objectgrid.plugins.ObjectTransformer#serializeKey
* (java.lang.Object,
* java.io.ObjectOutputStream)
*/
public void serializeKey(Object key, ObjectOutputStream stream) throws IOException {
    String ticket= (String) key;
    stream.writeUTF(ticket);
}

/* (non-Javadoc)
* @see com.ibm.websphere.objectgrid.plugins.
ObjectTransformer#serializeValue(java.lang.Object,
java.io.ObjectOutputStream)
*/
public void serializeValue(Object value, ObjectOutputStream stream) throws IOException {
    Stock stock= (Stock) value;
    stream.writeUTF(stock.getTicket());
    stream.writeUTF(stock.getCompany());
    stream.writeUTF(stock.getDescription());
    stream.writeDouble(stock.getPrice());
    stream.writeLong(stock.getLastTransactionTime());
    stream.writeInt(stock.getSerialNumber());
}

/* (non-Javadoc)
* @see com.ibm.websphere.objectgrid.plugins.
ObjectTransformer#inflateKey(java.io.ObjectInputStream)
*/
public Object inflateKey(ObjectInputStream stream) throws IOException, ClassNotFoundException {
    String ticket=stream.readUTF();
    return ticket;
}

/* (non-Javadoc)
* @see com.ibm.websphere.objectgrid.plugins.
ObjectTransformer#inflateValue(java.io.ObjectInputStream)
*/

public Object inflateValue(ObjectInputStream stream) throws IOException, ClassNotFoundException {
    Stock stock=new Stock();
    stock.setTicket(stream.readUTF());
    stock.setCompany(stream.readUTF());
    stock.setDescription(stream.readUTF());
    stock.setPrice(stream.readDouble());
    stock.setLastTransactionTime(stream.readLong());
    stock.setSerialNumber(stream.readInt());
    return stock;
}

/* (non-Javadoc)
* @see com.ibm.websphere.objectgrid.plugins.
ObjectTransformer#copyValue(java.lang.Object)
*/
public Object copyValue(Object value) {
    Stock stock = (Stock) value;
    try {
        return stock.clone();
    }
    catch (CloneNotSupportedException e)
    {
        // affichage du message d'exception     }
}

/* (non-Javadoc)
* @see com.ibm.websphere.objectgrid.plugins.
ObjectTransformer#copyKey(java.lang.Object)
*/
```

```
public Object copyKey(Object key) {
    String ticket=(String) key;
    String ticketCopy= new String (ticket);
    return ticketCopy;
}
}
```

Vous pouvez alors connecter cette classe personnalisée MyStockObjectTransformer dans la BackingMap :

```
ObjectGridManager ogf=ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid og = ogf.getObjectGrid("NYSE");
BackingMap bm = og.defineMap("NYSEStocks");
MyStockObjectTransformer ot = new MyStockObjectTransformer();
bm.setObjectTransformer(ot);
```

Sérialisation à l'aide des plug-ins DataSerializer

Utilisez le plug-in DataSerializer pour stocker efficacement des données arbitraires dans WebSphere eXtreme Scale pour que les API existantes du produit puissent interagir efficacement avec vos données.

La sérialisation des méthodes, telles que la sérialisation Java et le plug-in ObjectTransformer, permettent de convertir les données dans le réseau. En outre, lorsque vous utilisez ces options de sérialisation avec le mode de copie COPY_TO_BYTES, le transfert de données entre les clients et les serveurs devient moins coûteux et les performances s'améliorent. Toutefois, ces options ne résolvent pas les problèmes suivants qui peuvent exister :

- Les clés ne sont pas stockées sous forme d'octets ; ce sont toujours des objets Java.
- Le code côté serveur doit toujours augmenter l'objet. Par exemple, une requête et un index utilisent toujours la réflexion et doit étendre l'objet. De plus, les agents, les programmes d'écoute et les plug-ins sont toujours des objets.
- Les classes doivent toujours se trouver dans le chemin d'accès aux classes.
- Les données sont toujours dans le format de sérialisation Java (ObjectOutputStream).

Les plug-ins DataSerializer fournissent un moyen efficace de résoudre ces problèmes. Notamment, il vous permettent de décrire le format de sérialisation, ou tableau d'octets, dans WebSphere eXtreme Scalefin que le produit puisse interroger le tableau d'octets sans un format d'objet spécifique. Les classes et interface du plug-in DataSerializer public se trouvent dans le package com.ibm.websphere.objectgrid.plugins.io Pour plus d'informations, voir la .

Important : Les objets Java entité ne sont pas stockés directement dans les mappes de sauvegarde (BackingMaps) lorsque vous utilisez l'API EntityManager. L'API EntityManager convertit l'objet d'entité en objets de bloc de données. Les mappes d'entité sont automatiquement associées à un ObjectTransformer hautement optimisé. Lorsque l'API ObjectMap ou EntityManager est utilisée pour interagir avec les mappes d'entité, l'entité ObjectTransformer est appelée. Par conséquent, lorsque vous utilisez des entités, aucun travail n'est requis pour la sérialisation, car le produit exécute automatiquement le processus automatiquement.

Présentation de l'évolutivité

WebSphere eXtreme Scale est évolutif grâce aux données partitionnées. Il peut s'adapter à des milliers de conteneurs si besoin est, car les conteneurs sont indépendants les uns des autres.

WebSphere eXtreme Scale divise les ensembles de données en partitions distinctes pouvant être déplacées entre les processus ou même entre les serveurs physiques

au moment de l'exécution. Vous pouvez, par exemple, démarrer avec un déploiement de quatre serveurs, puis passer à un déploiement de 10 serveurs lorsque les demandes sur le cache augmentent. De la même façon que vous pouvez ajouter d'autres serveurs physiques et unités de traitement pour l'évolutivité verticale, vous pouvez étendre la capacité d'évolutivité élastique horizontalement grâce au partitionnement. L'évolutivité horizontale est un avantage majeur de l'utilisation de WebSphere eXtreme Scale par rapport à une base de données interne. Les bases de données interne peuvent évoluer verticalement.

Avec WebSphere eXtreme Scale, vous pouvez également utiliser un groupe d'API pour obtenir un accès transactionnel à ces données partitionnées et réparties. Les choix que vous faites pour interagir avec le cache sont tout aussi importants que les fonctions permettant de gérer le cache pour assurer la disponibilité du point de vue des performances.

Remarque : L'évolutivité n'est pas disponible lorsque les conteneurs communiquent entre eux. Le protocole de gestion de la disponibilité, ou groupement central, est un algorithme de maintenance de vue et de pulsation $O(N^2)$, mais il est atténué en réduisant le nombre de membres du groupe principal à moins de 20. Seule la réplication d'égal à égal entre les fragments existe.

Clients répartis

Le protocole client WebSphere eXtreme Scale prend en charge un grand nombre de clients. La stratégie de partitionnement suppose que tous les clients ne sont pas forcément intéressés par toutes les partitions, car les connexions peuvent s'étendre à plusieurs conteneurs. Les clients sont connectés directement aux partitions, de sorte que la latence est limitée à une connexion transférée.

Grilles de données, partitions et fragments

Une grille de données est divisée en partitions. Une partition contient un sous-ensemble unique de données. Une partition contient un ou plusieurs fragments : un fragment primaire et des fragments de réplique. Les fragments de réplique ne sont pas nécessaires dans une partition, mais vous pouvez en utiliser pour fournir la haute disponibilité. Que votre déploiement soit une grille de données indépendante stockée en mémoire ou un espace de traitement de base de données interne, l'accès aux données dans eXtreme Scale repose en grande partie sur les fragments.

Les données d'une partition sont stockées dans un ensemble de fragments au moment de l'exécution. Cet ensemble de fragments inclut un fragment primaire et une ou plusieurs éventuels fragments réplique. Un fragment est la plus petite unité que eXtreme Scale puisse ajouter ou supprimer d'une machine virtuelle Java.

Il existe deux stratégies de placement : placement de partition fixe (par défaut) et placement par conteneur. La section ci-dessous porte sur l'utilisation de la stratégie de placement de partition fixe.

Nombre total de fragments

Si votre environnement comprend 10 partitions contenant un million d'objets sans réplique, il existe 10 fragments contenant chacun 100 000 objets. Si vous ajoutez une réplique à ce scénario, un fragment supplémentaire existe dans chaque partition. Dans ce cas, il existe 20 fragments : 10 fragments primaires et 10

fragments de réplique. Chacun de ces fragments contient 100 000 objets. Chaque partition est composée d'un fragment primaire et d'une ou plusieurs répliques (N). La définition du nombre optimal de fragments s'avère déterminante. Si vous configurez quelques fragments, les données ne sont pas réparties de façon homogène entre les fragments, ce qui entraîne des erreurs de mémoire insuffisante et des problèmes de surcharge du processeur. Vous devez disposer d'au moins 10 fragments pour chaque JVM lorsque vous effectuez des modifications. Lorsque vous déployez initialement la grille de données, vous utilisez potentiellement un grand nombre de partitions.

Scénarios de nombre de fragments par JVM

Scénario : petite quantité de fragments pour chaque JVM

Les données sont ajoutées et supprimées d'une JVM à partir d'unités de fragments. Les fragments ne sont jamais divisés en plusieurs éléments. Pour 10 Go de données et 20 fragments qui les contiennent, chaque fragment contient en moyenne 500 Mo de données. Si neuf machines virtuelles Java hébergent la grille de données, chaque JVM possède deux fragments en moyenne. Etant donné que le nombre 20 n'est pas divisible par 9, quelques machines virtuelles Java comportent trois fragments selon la distribution suivante :

- Sept machines virtuelles Java avec deux fragments
- Deux machines virtuelles Java avec trois fragments

Etant donné que chaque fragment contient 500 Mo de données, les données ne sont pas réparties de façon égale. Les sept machines virtuelles Java dotées de deux fragments hébergent chacune 1 Go de données. Les deux machines virtuelles Java dotées de trois fragments contiennent 50 % de données en plus, soit 1,5 Go, ce qui représente une charge nettement supérieure pour la mémoire. Comme les deux machines virtuelles Java hébergent trois fragments, elles reçoivent aussi moitié plus de demande pour leurs données. En conséquence, un faible nombre de fragments pour chaque JVM génère un déséquilibre. Pour optimiser les performances, vous pouvez augmenter le nombre de fragments pour chaque JVM.

Scénario : quantité accrue de fragments pour chaque JVM

Dans ce scénario, augmentez nettement le nombre de fragments. Dans ce scénarios, il existe 101 fragments avec neuf machines virtuelles Java hébergeant 10 Go de données. Dans ce cas, chaque fragment contient 99 Mo de données. Les machines virtuelles Java présentent le distribution de fragments suivante :

- Sept machines virtuelles Java avec 11 fragments
- Deux machines virtuelles Java avec 12 fragments

Les deux machines virtuelles Java dotées de 12 fragments contiennent 99 Mo de données en plus que les autres fragments, ce qui constitue une différence de 9 %. Ce scénario est beaucoup plus régulièrement réparti que la différence de 50 % dans le scénario avec quelques fragments. Du point de vue de l'utilisation du processeur, seulement 9 % de plus de travail existe pour les deux machines virtuelles Java dotées de 12 fragments par rapport aux sept machines virtuelles Java dotées de 11 fragments. En augmentant le nombre de fragments dans chaque JVM, l'utilisation des données et du processeur est répartie de façon égale et juste.

Lorsque vous créez votre système, utilisez 10 fragments pour chaque JVM dans le scénario de taille maximale correspondant ou lorsque le système exécute le nombre maximal de machines virtuelles Java dans le cadre de votre planification.

Facteurs supplémentaires de positionnement

Le nombre de partitions, la stratégie de placement et nombre et le type des répliques sont définis dans la stratégie de déploiement. Le nombre de fragments placés dépend de la stratégie de déploiement que vous définissez. Les attributs minSyncReplicas, developmentMode, maxSyncReplicas et maxAsyncReplicas affectent le placement des partitions et des répliques.

Les facteurs suivants affectent le moment où les fragments sont placés :

- Les commandes **xscmd -c suspendBalancing** et **xscmd -c resumeBalancing**.
- 7.1.1+ Le fichier des propriétés du serveur, qui contient la propriété **placementDeferralInterval** qui définit le délai en millisecondes précédant le placement des fragments sur les serveurs de conteneur.
- L'attribut **numInitialContainers** dans la stratégie de déploiement.

Si le nombre maximal de répliques n'est pas placé au cours du démarrage initial, des répliques supplémentaires peuvent être placées si vous démarrez des serveurs supplémentaires ultérieurement. Lorsque vous planifiez le nombre de fragments par machine virtuelle Java, le nombre maximal de fragments primaires et de réplique est indépendant du fait de disposer d'un nombre suffisant de machines virtuelles Java pour prendre en charge le nombre maximal de répliques. Un fragment de réplique n'est jamais placé dans le même processus que le fragment primaire. Si un processus est perdu, le fragment principal et le fragment de réplique sont perdus. Lorsque l'attribut developmentMode a la valeur `false`, le fragment primaire et les fragments de réplique ne sont pas placés sur le même serveur physique.

Partitionnement

Utilisation du partitionnement pour étendre une application. Vous pouvez définir le nombre de partitions dans votre stratégie de déploiement.

A propos du partitionnement

Le partitionnement n'est pas similaire à la technologie RAID, qui divise chaque instance de chaque segment. Chaque partition héberge les données complètes d'entrées individuelles. Le partitionnement est un moyen très efficace d'extension, mais il n'est pas valable pour toutes les applications. Les applications qui nécessitent des garanties transactionnelles dans des ensembles de données volumineux ne sont pas évolutives et ne peuvent pas être partitionnées de manière efficace. WebSphere eXtreme Scale ne prend pas en charge actuellement la validation en deux phases dans les partitions.

Important : Sélectionnez le nombre de partitions avec précaution. Le nombre de partitions défini dans la règle de déploiement affecte directement le nombre de serveurs de conteneur sur lesquels une application peut évoluer. Chaque partition est composée d'un fragment primaire et du nombre configuré de fragments de réplique. La formule `(Nombre_Partitions*(1 + Nombre_Répliques))` correspond au nombre de conteneurs qui peuvent être utilisés pour étendre une application.

Utiliser des partitions

Une grille de données peut comporter des milliers de partitions. Une grille de données peut évoluer jusqu'au produit du nombre de partitions multiplié par le nombre de fragments par partition. Si, par exemple, vous avez 16 partitions dont chacune comporte un fragment primaire et un fragment réplique, soit deux

fragments, vous pouvez monter jusqu'à 32 machines virtuelles Java. Dans ce cas, il est défini un seul fragment pour chaque machine virtuelle Java. Vous devez choisir un nombre raisonnable de partitions en fonction du nombre de machines virtuelles Java que vous êtes censé utiliser. Pour le système, chaque fragment augmente l'utilisation de processeurs et de mémoire. Le système est conçu pour monter en puissance afin de gérer cette charge en fonction du nombre de machines virtuelles Java de serveurs qui sont disponibles.

Les applications ne doivent pas utiliser des milliers de partitions si l'application s'exécute sur une grille de données de quatre serveurs de conteneur machines virtuelles Java. L'application doit être configurée afin de disposer d'un nombre raisonnable de fragments pour chaque machine virtuelle Java de serveur de conteneur. Exemple de configuration déraisonnable : 2 000 partitions de deux fragments chacune, qui s'exécutent sur quatre machines virtuelles Java conteneurs. Résultat d'une telle configuration : 4 000 fragments placés dans quatre machines virtuelles Java conteneurs, soit 1 000 fragments par machine virtuelle Java conteneur.

Il serait plus intelligent d'avoir 10 fragments pour chacune des machines virtuelles Java attendues. Cette configuration n'empêche pas une évolutivité élastique de dix fois la configuration initiale tout en conservant une quantité raisonnable de fragments par machine virtuelle Java conteneur.

Supposons que vous disposiez de six serveurs physiques avec deux machines virtuelles Java de conteneur par serveur physique et que vous pensiez atteindre 20 serveurs physiques au cours des trois prochaines années. Avec 20 serveurs physiques, vous disposez de 40 machines virtuelles Java de serveur de conteneur et vous en choisissez 60 pour le mode pessimiste. Vous voulez utiliser quatre fragments par machine virtuelle Java de conteneur. Cela vous fait soixante conteneurs potentiels, soit un total de deux cent quarante fragments. Si vous avez un fragment primaire et un fragment réplique par partition, vous voudrez cent vingt partitions. Cet exemple vous donne 240 divisé par 12 machines virtuelles Java conteneurs, soit vingt fragments par machine virtuelle Java conteneur pour le déploiement initial avec le potentiel pour monter ultérieurement à vingt ordinateurs.

ObjectMap et partitionnement

Avec la stratégie FIXED_PARTITION de positionnement par défaut, les mappes sont fractionnées entre des partitions et des clés hachent vers les différentes partitions. Le client n'a pas besoin de savoir à quelles partitions appartiennent les clés. Si un groupe de mappes comporte plusieurs mappes, ces mappes doivent être validées dans des transactions distinctes.

Entités et partitionnement

Les entités EntityManager sont dotées d'une optimisation qui aide les clients utilisant des entités sur un serveur. Le schéma d'entités sur le serveur pour le groupe de mappes peut spécifier une seule entité racine. Le client doit accéder à toutes les entités via cette entité racine. Le gestionnaire d'entités peut alors trouver dans la même partition les entités liées à partir de cette racine sans avoir besoin que les mappes liées aient une clé commune. C'est l'entité racine qui établit l'affinité avec la partition. Cette partition sert à toutes les recherches d'entités au sein de la transaction une fois que l'affinité a été établie. Cette affinité peut

économiser de la mémoire car les mappes liées ne nécessitent pas de clé commune. L'entité racine doit être spécifiée avec une annotation d'entité modifiée, comme dans l'exemple suivant :

```
@Entity(schemaRoot=true)
```

Utilisez l'entité pour trouver la racine du graphe d'objets. Le graphe d'objets définit les relations entre une ou plusieurs entités. Chaque entité liée par une même relation doit se résoudre vers la même partition. Toutes les entités enfants sont censées se trouver dans la même partition que la racine. Les entités enfants présentes dans le graphe d'objets ne sont accessibles à un client qu'à partir de leur entité racine. Les entités racines sont toujours requises dans les environnements partitionnés lorsqu'on utilise un client eXtreme Scale pour communiquer avec le serveur. Il n'est possible de définir qu'une seule entité racine par clients. Les entités racines ne sont pas requises lorsque vous utilisez des ObjectGrids de type XTP (Extreme Transaction Processing), car toute la communication avec la partition s'effectue par accès direct en local et non via le mécanisme client et serveur.

Placement et partitions

Vous disposez de deux stratégies de placement pour WebSphere eXtreme Scale : partition fixe et par conteneur. Le choix de la stratégie de placement affecte la manière dont la configuration de votre déploiement place les partitions dans la grille de données distante.

Placement sur partition fixe

Vous pouvez définir dans le fichier XML de règles de déploiement la stratégie de positionnement. La stratégie par défaut est la stratégie de positionnement sur partition fixe, qui est activée avec le paramètre `FIXED_PARTITION`. Le nombre de fragments primaires qui sont placés dans les conteneurs disponibles est égal au nombre de partitions que vous avez configurées avec l'attribut numberOfPartitions. Si vous avez configuré des fragments réplique, le nombre total minimum de fragments placés est défini par la formule suivante : `((1 fragment primaire + minimum de fragments synchrones)*partitions définies)`. Le nombre total maximum de fragments placés est défini par la formule suivante : `((1 fragment primaire + maximum de fragments synchrones + maximum de fragments asynchrones)*partitions définies)`. Votre déploiement WebSphere eXtreme Scale dissémine ces fragments dans les conteneurs disponibles. Les clés de chaque mappe sont hachées en partitions attribuées en fonction du nombre total de partitions que vous avez définies. Ces clés hachent vers la même partition même si celle-ci est déplacée pour cause de basculement ou de changement de serveur.

Si, par exemple, la valeur de numberPartitions est de 6 et celle de minSync est de 1 pour MapSet1, le nombre total de fragments pour ce groupe de mappes sera de 12 car chacune des 6 partitions requiert un fragment réplique synchrone. Si trois conteneurs sont démarrés, WebSphere eXtreme Scale placera quatre fragments par conteneur pour MapSet1.

Placement par conteneur

L'autre stratégie de placement est le placement par conteneur qui est activé avec le paramètre `PER_CONTAINER` de l'attribut placementStrategy dans l'élément mapset du fichier XML de déploiement. Avec cette stratégie, le nombre de fragments primaires qui sont placés dans chaque nouveau conteneur est égal au nombre *P* de partitions que vous avez configuré. L'environnement de déploiement de WebSphere eXtreme Scale place *P* répliques de chaque partition pour chaque conteneur restant. Le

paramètre numInitialContainers est ignoré lorsqu'on utilise le positionnement par conteneur. Les partitions grandissent en même temps que les conteneurs. Dans cette stratégie, les clés des mappes ne sont pas fixées à une partition données. C'est le client qui route vers une partition en utilisant une partition primaire aléatoire. Pour pouvoir se reconnecter à une session qu'il a déjà utilisée pour trouver une clé, le client doit utiliser un descripteur de session.

Pour plus d'informations, voir la rubrique sur l'utilisation du descripteur SessionHandle pour le routage dans le *guide de programmation*.

En cas de basculement ou d'arrêt de serveur, dans la stratégie de positionnement par conteneur, l'environnement WebSphere eXtreme Scale déplace les fragments primaires si ces fragments contiennent encore des données. S'ils sont vides, ils sont éliminés. Dans la stratégie par conteneur, les anciens fragments primaires ne sont pas conservés car de nouveaux fragments primaires sont placés pour chaque conteneur.

WebSphere eXtreme Scale permet d'utiliser le placement par conteneur comme une alternative à ce qu'on peut appeler une stratégie de placement "type", une approche reposant sur le partitionnement fixe avec la clé d'une mappe hachée dans l'une de ces partitions. Dans le cas du placement par conteneur (que vous pouvez définir à l'aide de PER_CONTAINER), votre déploiement place les partitions dans l'ensemble des serveurs de conteneur en ligne et les étend ou les réduit à mesure que des conteneurs sont ajoutés ou supprimés de la grille de données des serveur. Une grille de données avec l'approche partition fixe fonctionne bien pour les grilles à base de clés, si l'application utilise un objet key pour localiser es données dans la grille. Ici, nous allons voir l'autre approche.

Exemple de grille de données par conteneur

Les grilles de données PER_CONTAINER sont différentes. Vous indiquez que la grille utilise la stratégie de placement PER_CONTAINER avec l'attribut placementStrategy dans le fichier XML de déploiement. Au lieu de définir le nombre total de partitions dans la grille de données, vous pouvez définir le nombre de partitions nécessaires pour chaque conteneur que vous démarrez.

Par exemple, si vous définissez cinq partitions par conteneur, cinq nouvelles partitions primaires anonymes sont créées lorsque vous démarrez ce serveur de conteneur, et les répliques nécessaires sont créées sur les autres serveurs de conteneur déployés.

Voici une séquence possible dans un environnement par conteneur à mesure que la taille de la grille augmente.

1. L'on démarre le conteneur C0 qui héberge 5 partitions primaires (P0-P4).
   - C0 héberge P0, P1, P2, P3, P4.
2. L'on démarre le conteneur C1 qui héberge 5 autres partitions primaires (P5-P9). Des fragments réplique sont répartis de manière équilibrée sur les conteneurs.
   - C0 héberge P0, P1, P2, P3, P4, R5, R6, R7, R8, R9.
   - C1 héberge P5, P6, P7, P8, P9, R0, R1, R2, R3, R4.
3. L'on démarre le conteneur C2 qui héberge 5 autres partitions primaires (P10-P14). Là aussi, des fragments réplique sont répartis de manière équilibrée sur les conteneurs.
   - C0 héberge P0, P1, P2, P3, P4, R7, R8, R9, R10, R11, R12.
   - C1 héberge P5, P6, P7, P8, P9, R2, R3, R4, R13, R14.

- C2 héberge P10, P11, P12, P13, P14, R5, R6, R0, R1.

Le schéma continue au fur et à mesure que d'autres conteneurs sont ajoutés en créant cinq nouvelles partitions primaires chaque fois et en rééquilibrant les fragments réplique sur les conteneurs disponibles dans la grille de données.

Remarque : WebSphere eXtreme Scale ne déplace pas les fragments primaires lors de l'utilisation de la stratégie PER_CONTAINER, mais uniquement les fragments de réplique.

Ne perdez pas de vue que, le nombre des partitions étant arbitraire et n'ayant rien à voir avec des clés, il est impossible d'utiliser un routage à base de clés. Si un conteneur s'arrête, les ID de partitions créés pour ce conteneur ne sont plus utilisés, ce qui fait qu'il y a un trou dans les ID de partitions. Dans notre exemple, en cas de défaillance du conteneur C2, il n'y aurait plus de partitions P5-P9 et il ne resterait plus que les partitions P0-P4 et P10-P14, ce qui rend impossible tout hachage à base de clés.

L'utilisation de valeurs, telles que cinq ou plus probablement 10 pour le nombre de partitions par conteneur, fonctionne mieux si vous tenez compte des conséquences de la défaillance d'un conteneur. Pour répartir la charge d'hébergement des fragments dans la grille de données, vous avez besoin de plusieurs partitions pour chaque conteneur. Si l'on n'a qu'une seule partition par conteneur, en cas de défaillance de l'un des conteneurs, c'est un seul conteneur (celui qui héberge le fragment réplique correspondant) qui devra porter toute la charge du fragment primaire perdu. Dans ce cas, la charge est immédiatement doublée pour le conteneur. Toutefois, si vous avez cinq partitions par conteneur, alors cinq conteneurs recueillent la charge du conteneur perdu en réduisant l'impact sur chacun d'entre eux de 80 %. L'utilisation de plusieurs partitions par conteneur diminue de manière substantielle l'impact potentiel sur chacun des conteneurs. Plus directement, l'on peut envisager le cas où un conteneur connaît de manière inopinée des pics d'utilisation. La charge de la réplication sur ce conteneur sera alors répartie sur 5 autres conteneurs et non uniquement sur un seul.

Utiliser la stratégie par conteneur

Plusieurs scénarios font de la stratégie par conteneur la configuration idéale, pour la réplication de sessions HTTP, par exemple, ou pour l'état de session d'application. Dans un pareil cas, un routeur HTTP affecte une session à un conteneur de servlet. Ce dernier a besoin de créer une session HTTP et il choisit l'une des 5 partitions primaires locales. L'"ID" de la partition choisie est alors stocké dans un cookie. Le conteneur de servlet peut désormais accéder en local à l'état de la session, ce qui signifie un temps d'attente nul pour l'accès aux données dans le cadre de cette demande aussi longtemps que l'on maintient l'affinité de session. Une grille eXtreme Scale réplique toutes les modifications apportées à la partition.

Dans la pratique, rappelez-vous ce que nous disions plus haut des conséquences avantageuses entraînées par le fait d'avoir plusieurs partitions (nous disions 5) par conteneur. Naturellement, chaque nouveau conteneur qui démarre ajoute 5 nouvelles partitions et 5 autres fragments réplique. Au fil du temps, d'autres partitions doivent normalement être créées et elles ne doivent ni être déplacées ni être détruites. Mais ce n'est pas ainsi que se comportent en fait les conteneurs. Lorsqu'un conteneur démarre, il héberge 5 fragments primaires, que l'on peut appeler des fragments primaires "home", et qui existent dans les conteneurs respectifs qui les ont créés. En cas de défaillance du conteneur, les fragments

réplique deviennent des fragments primaires et eXtreme Scale crée 5 autres fragments réplique afin de conserver la haute disponibilité (à moins que l'on n'ait désactivé la réparation automatique). Les nouveaux fragments primaires se trouvant sur un conteneur autre que celui qui les a créés, on peut les appeler des fragments primaires "externes". L'application ne doit en aucun cas placer un nouvel état ou de nouvelles sessions dans un fragment primaire externe. Au final, le fragment primaire externe ne comporte aucune entrée et eXtreme Scale le supprime automatiquement ainsi que les fragments réplique qui lui sont associés. Les fragments primaires externes n'ont de raison d'être que de permettre aux sessions existantes d'être toujours disponibles (aux sessions existantes, mais non à de nouvelles sessions).

Un client peut toujours interagir avec une grille de données qui ne s'appuie pas sur des clés. Le client commence simplement une transaction et il stocke les données dans la grille de données indépendamment des clés. Il réclame à la session un objet SessionHandle, qui est un descripteur sérialisable lui permettant d'interagir avec la même partition lorsque c'est nécessaire. Pour plus d'informations, voir dans le *Guide de programmation* la rubrique consacrée à l'utilisation d'un SessionHandle pour le routage. WebSphere eXtreme Scale choisit une partition pour le client dans la liste des partitions primaires home. Il ne retourne jamais de partition primaire externe. SessionHandle peut être sérialisé en cookie HTTP, par exemple, et le cookie peut ultérieurement être reconverti en SessionHandle. Ensuite, les API WebSphere eXtreme Scale peuvent obtenir une liaison Session à la même partition de nouveau, à l'aide de SessionHandle.

Remarque : Vous ne pouvez pas utiliser d'agents pour interagir avec une grille de données PER_CONTAINER.

Avantages

La description antérieure est différente d'une grille FIXED_PARTITION normale ou des données de hachage, parce que le client par conteneur stocke les données dans un endroit de la grille, obtient un descripteur pour ces données et utilise ce descripteur pour accéder à nouveau aux données. Il n'existe aucune clé fournie par application comme c'est le cas dans la partition fixe.

Le déploiement ne crée pas de nouvelle partition pour chaque session. De ce fait, dans un déploiement par conteneur, les clés qui servent à stocker les données dans la partition doivent exister en un seul exemplaire au sein de cette partition. L'on peut, par exemple, faire générer par le client un SessionID unique que l'on utilisera comme clé afin de trouver les informations dans les mappes de cette partition. De multiples sessions clients interagissent avec la même partition ; c'est pourquoi l'application a besoin d'utiliser des clés uniques pour stocker les données de session dans chacune des partitions concernées.

Les exemples donnés plus haut utilisaient 5 partitions, mais le paramètre numberOfPartitions du fichier XML objectgrid permet de spécifier autant de partitions que l'on veut. Le paramètre n'est pas par grille de données, mais par conteneur. (Le nombre de fragments réplique est spécifié de la même manière qu'avec la stratégie de partition fixe).

La stratégie par conteneur est également utilisable avec des zones multiples. Dans la mesure du possible, eXtreme Scale retourne un SessionHandle à une partition dont le fragment primaire est situé dans la même zone que le client. Celui-ci peut

spécifier la zone au conteneur comme paramètre ou à 'aide d'une API. L'ID de zone du client peut être défini à l'aide de `serverproperties` ou de `clientproperties`.

La stratégie PER_CONTAINER est adaptée aux applications qui stockent un état de type transactionnel et non pas des données orientées base de données. La clé permettant d'accéder aux données sera un ID de conversation et non un enregistrement spécifique de base de données. Cette stratégie permet des performances plus élevées (car les partitions primaires peuvent cohabiter, avec des servlets, par exemple) et une configuration plus facile (pas besoin de calculer les partitions et les conteneurs).

Transactions à partition unique et cross-data-grid

La différence majeure entre WebSphere eXtreme Scale et les solutions classiques de stockage de données (bases de données relationnelles ou bases de données en mémoire) consiste en l'utilisation du partitionnement, qui permet au cache d'évoluer de manière linéaire. Les principaux types de transactions à prendre en compte sont les transactions à une seule partition et les transactions every-partition (inter-data-grid).

En règle générale, les interactions avec le cache peuvent être classées comme des transactions à partition unique ou des transactions cross-data-grid, comme décrit dans la section suivante.

Transactions dans une partition unique

Les transactions dans une partition unique constituent le mode préférentiel d'interaction avec les mémoires cache hébergées par WebSphere eXtreme Scale. Lorsqu'une transaction est limitée à une partition, elle se limite par défaut à une seule machine virtuelle Java et donc à un seul serveur. Un serveur peut exécuter un nombre *M* de transactions par seconde. Si votre système contient *N* ordinateurs, vous pouvez exécuter `M*N` transactions par seconde. Si votre activité augmente et que vous ayez besoin de doubler le nombre de transactions par seconde, vous pouvez doubler *N* en installant davantage d'ordinateurs. Vous pouvez alors répondre à vos besoins en capacité sans modifier l'application, mettre à niveau le matériel ni utiliser l'application hors ligne.

Outre qu'elles permettent au cache d'évoluer de manière significative, les transactions s'exécutant dans une seule partition permettent aussi d'optimiser la disponibilité de ce dernier. Chaque transaction est liée à un seul ordinateur. Une défaillance peut se produire sur l'un des autres ordinateurs `(N-1)` sans que la réussite ou le temps de réponse de la transaction en soit affecté. Si 100 ordinateurs sont en cours d'exécution et que l'un d'eux échoue, 1 % des transactions en cours au moment de l'échec sont annulées. Après cet échec, WebSphere eXtreme Scale relocalise les partitions qui sont hébergées par le serveur défaillant vers les 99 autres ordinateurs. Pendant cette courte période, ces ordinateurs continuent à exécuter des transactions. Seules les transactions impliquant les partitions qui sont en cours de relocalisation sont bloquées. Une fois le basculement terminé, le cache peut continuer à s'exécuter en étant pleinement opérationnel, c'est-à-dire à 99 % de sa capacité de traitement d'origine. Une fois que le serveur défaillant est remplacé et revenu dans la grille de données, le cache retrouve 100 % de sa capacité.

Transactions Cross-data-grid

En termes de performances, de disponibilité et d'évolutivité, les transactions cross-data-grid sont l'opposé des transactions à partition unique. Elles ont accès à toutes les partitions et donc à chaque ordinateur de la configuration. Chaque ordinateur de la grille de données est sollicité pour rechercher les données, puis renvoyer le résultat. La transaction n'est pas terminée tant que tous les ordinateurs n'ont pas répondu, et donc le traitement de l'ensemble de la grille de données est limité par l'ordinateur le plus lent. L'ajout d'ordinateurs ne permet pas d'améliorer la vitesse de l'ordinateur le plus lent ni d'augmenter la capacité de traitement du cache.

Les transactions cross-data-grid ont des effets semblables sur la disponibilité. Reprenons l'exemple précédent : si 100 serveurs sont en cours d'exécution et que l'un d'eux échoue, 100 % des transactions en cours sont annulées. Après cet échec, WebSphere eXtreme Scale commence à relocaliser les partitions qui sont hébergées par ce serveur vers les 99 autres ordinateurs. Pendant ce temps, avant la fin du basculement, la grille de données ne peut traiter aucune de ces transactions. Une fois le basculement terminé, le cache continue à s'exécuter, mais à un niveau de capacité réduit. Si chaque ordinateur de la grille de données gérait 10 partitions, 10 des 99 ordinateurs restants reçoivent au moins une partition supplémentaire dans le cadre du processus de basculement. L'ajout d'une partition supplémentaire augmente la charge de travail de cet ordinateur d'au moins 10 %. Etant donné que la capacité de la grille de données est limitée à la capacité de traitement de l'ordinateur le plus lent dans une transaction cross-data-grid, en moyenne, le traitement est réduit de 10.

Il est préférable d'utiliser des transactions à une seule partition que des transactions cross-data-grid pour l'évolutivité avec un cache d'objet haute disponibilité réparti tel que WebSphere eXtreme Scale. L'optimisation des performances de ces systèmes requiert l'utilisation de techniques qui diffèrent des méthodologies relationnelles traditionnelles, mais vous pouvez transformer les transactions cross-data-grid en transactions évolutives à une seule partition.

Méthodes recommandées en matière de génération de modèles de données évolutifs

Les méthodes recommandées pour générer des applications évolutives avec des produits tels que WebSphere eXtreme Scale sont au nombre de deux : les principes fondamentaux et les conseils relatifs à l'implémentation. Les principes fondamentaux sont les grandes idées que vous devez capturer lors de la conception des données. Une application n'observant pas ces principes aura peu de chance de pouvoir évoluer de manière satisfaisante, même pour les transactions principales. Les conseils d'implémentation s'appliquent aux transactions posant problème dans une application par ailleurs bien conçue et observant les principes généraux relatifs aux modèles de données évolutifs.

Principes fondamentaux

Certains concepts ou principes de base à ne pas oublier constituent les éléments clés pour optimiser l'évolutivité.

Duplication plutôt que normalisation

> Il est essentiel de bien comprendre que les produits tels que WebSphere eXtreme Scale sont conçus pour répartir des données dans un grand nombre d'ordinateurs. Si votre objectif est d'exécuter la plupart ou

l'ensemble des transactions sur un même ordinateur, le modèle de données doit s'assurer que toutes les données nécessaires à la transaction sont situées dans cette partition. Dans la plupart des cas, la seule manière d'y parvenir consiste à dupliquer les données.

Prenons l'exemple d'un forum électronique. Deux transactions très importantes pour ce forum présentent tous les messages publiés par un utilisateur donné et tous les messages publiés sur un sujet donné. Imaginez tout d'abord comment ces transactions se comporteraient dans le cadre d'un modèle de données normalisé contenant un enregistrement utilisateur, un enregistrement relatif au sujet et un enregistrement relatif au message et contenant le texte réel. Si les messages publiés sont partitionnés avec les enregistrements utilisateur, l'affichage du sujet devient une transaction impliquant l'ensemble de la grille, et inversement. Il est impossible de partitionner les sujets et les utilisateurs car leur relation est de type many-to-many.

La meilleure méthode pour permettre à ce forum électronique d'évoluer est de dupliquer les messages publiés, c'est-à-dire de copier chaque message avec l'enregistrement relatif au sujet et avec l'enregistrement utilisateur. L'affichage des messages publiés à partir d'un utilisateur constitue alors une transaction dans une partition unique, de même que l'affichage des messages publiés dans un sujet, tandis que la mise à jour ou la suppression d'un message publié est une transaction impliquant deux partitions. Ces trois transactions évolueront de manière linéaire à mesure que des ordinateurs sont ajoutés à la grille de données.

L'évolutivité plutôt que les ressources

Le principal obstacle à surmonter en matière de modèles de données dénormalisés est l'impact de ces modèles sur les ressources. La conservation de deux ou trois copies, voire plus, de certaines données risque d'entraîner la consommation d'une trop grande quantité de ressources pour être pratique. Lorsque vous êtes confronté à un tel scénario, gardez ceci en mémoire : les coûts liés à l'achat de matériel diminuent d'année en année. Autre considération, et non des moindres : WebSphere eXtreme Scale permet de supprimer la plupart des coûts associés au déploiement de ressources supplémentaires.

Mesurez les ressources en termes de coût plutôt qu'en termes purement informatiques comme les mégaoctets et les processeurs. Les magasins de données utilisant des données relationnelles normalisées doivent généralement être situés sur le même ordinateur. Cela signifie que vous devez acheter un seul ordinateur d'entreprise puissant, plutôt que plusieurs machines modestes. Il n'est pas rare qu'un ordinateur d'entreprise pouvant exécuter un million de transactions par seconde soit bien plus onéreux que dix ordinateurs pouvant traiter 100 000 transactions par seconde.

L'ajout de ressources a également un coût. Une entreprise en pleine croissance finit par manquer de capacité. Lorsque vous vous trouvez dans cette situation, vous devez soit arrêter votre système lors du passage à un ordinateur plus rapide et plus puissant, soit créer un second environnement de production. Quelle que soit l'option choisie, vous devrez prendre en charge des coûts supplémentaires liés à la réduction du volume de traitement ou au maintien de la capacité de traitement, pratiquement doublée pendant la durée de la transaction.

Avec WebSphere eXtreme Scale, il n'est pas nécessaire de fermer l'application lors de l'accroissement de la capacité. Si votre entreprise prévoit que vous avez besoins de 10 % de capacité de plus pour l'année prochaine, augmentez le nombre d'ordinateurs de 10 % dans la grille de données. Ce pourcentage peut augmenter sans entraîner d'indisponibilité de l'application et sans que vous ayez à accroître la capacité.

Des transformations de données inutiles

Lorsque vous utilisez WebSphere eXtreme Scale, vous devez stocker les données dans un format directement utilisable par la logique métier. La répartition des données dans un format plus primitif consomme davantage de ressources. La transformation doit se faire lors de l'écriture et lors de la lecture des données. Dans le cas d'une base de données relationnelle, cette transformation est nécessaire car les données sont enregistrées fréquemment sur le disque, mais avec WebSphere eXtreme Scale, elle n'est pas obligatoire. La plupart des données sont stockées en mémoire et peuvent donc être stockées au format requis par l'application.

L'observation de cette règle simple vous aide à dénormaliser les données conformément au premier principe. Le type de transformation des données métier le plus commun est l'opération JOIN nécessaire pour transformer des données normalisées en un ensemble de résultats correspondant aux besoins de l'application. Le stockage des données au format correct permet implicitement d'éviter d'avoir à exécuter ces opérations de type JOIN et génère un modèle de données dénormalisé.

Abandon des requêtes illimitées

Quelle que soit la structure de vos données, les requêtes illimitées évoluent mal. Nous ne vous recommandons par exemple pas d'exécuter une transaction visant à obtenir une liste de tous les articles triés par valeur. Elle peut renvoyer un résultat correct au début, lorsque le nombre d'articles est égal à 1 000, mais lorsque celui-ci atteint 10 millions, la transaction renvoie ces 10 millions d'articles. L'exécution d'une telle transaction risque d'entraîner un délai d'inactivité de celle-ci ou une erreur liée à une insuffisance de mémoire du client.

La meilleure solution consiste à modifier la logique métier de façon que seuls les 10 ou 20 vingt premiers articles soient renvoyés. Cette modification permet de faire en sorte que la transaction soit gérable quel que soit le nombre d'articles présents en cache.

Définition d'un schéma

Le principal avantage lié à la normalisation des données est que la base de données peut prendre en charge la cohérence des données en arrière-plan. Lorsque les données sont dénormalisées à des fins d'évolutivité, la gestion automatique de la cohérence des données disparaît. Vous devez implémenter un modèle de données pouvant fonctionner dans la couche d'applications ou en tant que plug-in de la grille de données répartie pour garantir la cohérence des données.

Prenons l'exemple du forum électronique. Si une transaction supprime un message publié d'un sujet, sa copie dans l'enregistrement utilisateur doit également être supprimée. Sans modèle de données, il est possible que le développeur prévoie la suppression du message publié dans le code de l'application, mais qu'il oublie de le supprimer de l'enregistrement utilisateur. Toutefois, s'il avait utilisé un modèle de données au lieu d'interagir directement avec le cache, la méthode removePost aurait permis

d'extraire l'ID utilisateur du message, de rechercher l'enregistrement utilisateur et de supprimer la copie du message en arrière-plan.

Vous pouvez également implémenter un programme d'écoute s'exécutant sur la partition et capable de détecter la modification apportée au sujet et de modifier automatiquement l'enregistrement utilisateur. Un tel programme peut se révéler avantageux car la modification de l'enregistrement utilisateur est effectuée localement si celui-ci se trouve sur la partition ou, s'il se trouve sur une autre partition, la transaction est exécutée entre deux serveurs et non entre le client et le serveur. La connexion réseau entre des serveurs est probablement plus rapide que la connexion existant entre le client et le serveur.

Disparition des conflits

Evitez les scénarios contenant par exemple un compteur global. La grille de données n'évoluera pas si un enregistrement est utilisé un nombre de fois disproportionné par rapport aux autres enregistrements. Les performances de la grille de données seront limitées par celle de l'ordinateur qui contient cet enregistrement.

Dans une telle situation, essayez de fractionner l'enregistrement afin qu'il soit géré au niveau de la partition. Prenons le cas d'une transaction renvoyant le nombre total d'entrées présentes dans le cache réparti. Plutôt que d'exécuter une opération d'insertion et de suppression accédant à un enregistrement unique qui s'incrémente, installez un programme d'écoute sur chaque partition pour suivre ces opérations. Grâce à ce suivi, les opérations d'insertion et de suppression deviennent des transactions s'exécutant dans une partition unique.

La lecture du compteur devient une opération cross-data-grid, mais dans l'ensemble elle était déjà aussi inefficace qu'une opération cross-data-grid, car ses performances étaient liées à celles de l'ordinateur contenant l'enregistrement.

Conseils relatifs à l'implémentation

Pour optimiser l'évolutivité, prenez en compte les conseils suivants.

Utilisation des index de recherche inversée

Prenons le cas d'un modèle de données dénormalisé dans lequel les enregistrements client sont partitionnés en fonction du numéro d'ID du client. Cette méthode de partitionnement est un choix logique car pratiquement toutes les opérations métier exécutées avec l'enregistrement client utilisent cet ID. Toutefois, la transaction de connexion, qui est essentielle, ne l'utilise pas. Les données utilisées pour la connexion sont plus fréquemment le nom d'utilisateur ou l'adresse électronique.

L'approche la plus simple pour le scénario de connexion consiste à utiliser une transaction cross-data-grid pour rechercher l'enregistrement client. Comme expliqué précédemment, cette approche n'est pas évolutive.

Autre option : procéder à un partitionnement en fonction du nom d'utilisateur ou de l'adresse électronique. Cette option n'est pas pratique, car toutes les opérations basées sur l'ID du client deviennent des transactions cross-data-grid. De plus, les clients de votre site pourraient souhaiter modifier leur nom d'utilisateur ou leur adresse électronique. Dans les produits tels que WebSphere eXtreme Scale, la valeur utilisée pour partitionner les données doit rester constante.

La bonne solution consiste alors à utiliser un index de recherche inversée. Avec WebSphere eXtreme Scale, il est possible de créer un cache dans la même grille répartie que le cache contenant tous les enregistrements utilisateur. Ce cache est à haute disponibilité, partitionnée et évolutif. Il permet de mapper un nom d'utilisateur ou une adresse électronique vers un ID client. Plutôt que d'avoir une opération impliquant l'ensemble de la grille, il transforme la procédure de connexion en transaction s'exécutant sur deux partitions. Ce scénario n'est pas aussi optimal qu'une transaction impliquant une seule partition, mais la capacité de traitement augmente cependant de manière linéaire par rapport au nombre d'ordinateurs.

Calcul des valeurs lors de l'écriture

Les calculs les plus fréquents, tels que les moyennes ou les totaux, peuvent consommer une grande quantité de ressources car ils supposent la lecture d'un grand nombre d'entrées. Les lectures étant plus fréquentes que les écritures dans la plupart des applications, il est plus efficace de calculer ces valeurs lors de l'écriture, puis de stocker le résultat dans le cache. Les opérations gagnent ainsi en rapidité et en évolutivité.

Zones facultatives

Prenons l'exemple d'un enregistrement utilisateur contenant un numéro de téléphone professionnel, un numéro de téléphone personnel et un numéro de téléphone portable. Tous ces numéros ou une combinaison d'entre eux (ou aucun) peuvent être définis pour un utilisateur. Si les données ont été normalisées, une table utilisateur et une table contenant les numéros de téléphone existent. Vous pouvez alors identifier les numéros de téléphone d'un utilisateur donné à l'aide d'une opération JOIN entre les deux tables.

Pour dénormaliser cet enregistrement, aucune duplication des données n'est nécessaire, car la plupart des utilisateurs ne partagent pas leurs numéros de téléphone. Au contraire, les emplacements vides doivent être autorisés dans l'enregistrement utilisateur. Au lieu de constituer une table contenant les numéros de téléphone, ajoutez trois attributs à l'enregistrement utilisateur, chacun correspondant à un type de numéro de téléphone. L'ajout de ces attributs rend superflue l'opération JOIN et permet d'effectuer la recherche des numéros de téléphone d'un utilisateur en tant qu'opération impliquant une seule partition.

Positionnement des relations many-to-many

Prenons l'exemple d'une application qui assure le suivi de certains produits et des magasins dans lesquels ceux-ci sont commercialisés. Un même produit est vendu dans plusieurs magasins et un même magasin vend plusieurs produits. Supposons que cette application assure le suivi de 50 revendeurs importants. Chaque produit est vendu dans 50 magasins au maximum, chaque magasin commercialisant des milliers de produits.

Constituez une liste des magasins dans l'entité produit (organisation A) plutôt qu'une liste des produits dans chaque entité magasin (organisation B). Une simple observation des transactions que cette application devrait exécuter permet de comprendre pourquoi l'organisation A est la plus évolutive.

Considérons d'abord les mises à jour. Dans l'organisation A, la suppression d'un produit du stock d'un magasin verrouille l'entité produit. Si la grille de données contient 10 000 produits, seul 1/1O 000 de la grille doit être verrouillé lors de la mise à jour. Dans l'organisation B, la grille de données contient seulement 50 magasins, si bien qu'1/50 de la grille doit être

verrouillé pour terminer la mise à jour. Même si les deux opérations peuvent être considérées comme des opérations impliquant une seule partition, l'organisation A permet une meilleure évolutivité.

Considérons maintenant les opérations de lecture avec l'organisation A : la recherche des magasins dans lesquels un produit est commercialisé est une transaction impliquant une seule partition, pouvant évoluer et rapide car elle transmet une faible quantité de données. Avec l'organisation B, cette transaction devient une transaction cross-data-grid, car chaque entité de magasin doit être accessible afin de déterminer si le produit est vendu dans ce magasin, ce qui donne un avantage de performance remarquable pour l'organisation A.

Evolutivité avec les données normalisées

L'évolution du traitement des données est l'une des utilisations légitimes des transactions cross-data-grid. Si une grille de données comporte 5 ordinateurs et qu'une transaction cross-data-grid est distribuée pour trier environ 100 000 enregistrements sur chaque ordinateur, cette transaction trie 500 000 enregistrements. Si l'ordinateur le plus lent dans la grille de données peut traiter 10 de ces transactions par seconde, la grille de données est capable de trier 5 000 000 d'enregistrements par seconde. Si les données de la grille doublent, chaque ordinateur doit trier 200 000 enregistrements et chaque transaction trie 1 million d'enregistrements. Cette progression des données ramène la capacité de l'ordinateur le plus lent à 5 transactions par seconde, ce qui réduit le traitement de la grille de données à 5 transactions par seconde. Cependant, la grille trie toujours les données de 5 000 000 d'enregistrements par seconde.

Dans un tel scénario, le fait de doubler le nombre d'ordinateurs permet à chaque ordinateur de revenir à sa charge précédente et à l'ordinateur le plus lent de traiter 10 de ces transactions par seconde. La capacité de la grille de données reste la même à 10 demandes par seconde, mais chaque transaction traite désormais 1 000 000 d'enregistrements, si bien que la grille a doublé sa capacité en terme de traitement des enregistrements pour atteindre 10 000 000 par seconde.

Pour les applications, telles qu'un moteur de recherche, qui ont besoin d'évoluer à la fois en termes de traitement des données pour s'adapter à la taille croissante de l'Internet et de capacité pour s'adapter à l'augmentation du nombre d'utilisateurs, vous devez créer plusieurs grilles de données avec un traitement circulaire des demandes entre les grilles. Si vous devez augmenter le débit, ajoutez des ordinateurs et ajoutez une grille de données aux demandes de service. Si vous devez augmenter le traitement des données, ajoutez des ordinateurs et ne modifiez pas le nombre de grilles.

Mise à l'échelle en unités ou capsules

Bien que vous puissiez déployer une grille de données sur des milliers de machines virtuelles Java, vous pouvez envisager de scinder la grille de données en unités ou capsules pour améliorer la fiabilité et faciliter les tests de votre configuration. Une capsule est un groupe de serveurs qui exécute le même ensemble d'applications.

Déploiement d'une grille de données volumineuses unique

Les tests ont montré que eXtreme Scale peut s'étendre à plus de 1 000 machines virtuelles Java. Ces tests encouragent la création d'applications permettant de

déployer des grilles de données uniques sur un grand nombre de machines. Bien qu'il soit possible d'effectuer cette opération, elle n'est pas recommandée, pour plusieurs raisons :

1. **Budget :** votre environnement ne peut pas tester, à l'évidence, une grille de 1 000 serveurs. Cependant, il peut tester une grille de données beaucoup plus petite en tenant compte des raisons budgétaires, de sorte que vous n'avez pas besoin d'acheter deux fois le matériel, en particulier pour un si grand nombre de serveurs.
2. **Différentes versions d'application :** l'utilisation d'un grand nombre de sous-systèmes de stockage pour chaque unité d'exécution de test n'est pas pratique. Le risque est de ne pas tester les mêmes facteurs que si vous étiez dans un environnement de production.
3. **Perte de données :** l'exécution d'une base de données sur un lecteur de disque dur n'est pas fiable. Tout problème avec le disque dur peut entraîner la perte de données. L'exécution d'une application dont la taille augmente sur une grille de données unique est similaire. Il est probable que vous rencontriez des bogues dans votre environnement et vos applications. Le positionnement toutes les données sur un seul grand système mène souvent à la perte d'un grand nombre de données.

Fractionnement de la grille de données

Le fractionnement de la grille de données de l'application en capsules (unités) est une option plus fiable. Une capsule est un groupe de serveurs qui exécutent une pile d'applications homogènes. Les capsules peuvent avoir n'importe quelle taille, mais elles doivent idéalement se composer de 20 serveurs physiques. Au lieu d'avoir 500 serveurs physiques dans une grille de données unique, vous pouvez avoir 25 capsules de 20 serveurs physiques. Une seule version d'une pile d'applications doit s'exécuter dans une capsule donnée, mais les différentes capsules peuvent avoir leur propre version d'une pile d'applications.

Généralement, une pile d'applications prend en compte les niveaux des composants suivants.

- système d'exploitation
- matériel
- machines virtuelles Java
- version d'WebSphere eXtreme Scale
- application
- autres composants nécessaires

Une capsule est une unité de déploiement de taille adaptée aux tests. Dans le cadre des tests, il est plus pratique d'avoir 20 serveurs plutôt que plusieurs centaines. Vous continuez néanmoins de tester la même configuration que vous auriez en production. La production utilise des grilles de 20 serveurs maximum, chacune constituant une capsule. Vous pouvez effectuer des tests de contrainte sur une capsule, puis déterminer sa capacité, le nombre d'utilisateurs, la quantité de données et la capacité de traitement. Cela facilite la planification et permet une évolutivité et des coûts prévisibles.

Configuration d'un environnement basé sur capsule

Selon les cas, la capsule ne contient pas forcément 20 serveurs. La taille de la capsule doit être adaptée aux tests. La taille d'une capsule doit être telle que, si la capsule rencontre un problème en production, la quantité des transactions affectées est tolérable.

Dans l'absolu, un bogue affecte une seule capsule. Un bogue n'aurait un impact que sur quatre pourcent des transactions de l'application, au lieu de 100 %. De plus, les mises à niveau sont facilitées, car elles peuvent être appliquées à une seule capsule à la fois. De ce fait, si une mise à niveau vers une capsule crée des problèmes, l'utilisateur peut ramener la capsule au niveau antérieur. Les mises à niveau incluent toutes les modifications apportées à l'application, la pile d'applications ou les mises à jour système. Dans la mesure du possible, les mises à niveau doivent modifier un seul élément de la pile à la fois, afin de permettre un diagnostic plus précis des problèmes.

Pour implémenter un environnement, vous avez besoin d'une couche de routage au-dessus des capsules qui soit compatible avec les versions antérieures et ultérieures si les capsules reçoivent des mises à niveau de logiciel. Vous devez également créer un répertoire contenant les informations sur le contenu de chaque capsule. Vous pouvez utiliser une autre grille de données eXtreme Scale pour cela avec une base de données derrière, de préférence dans un scénario d'écriture différée. Cela génère une solution à deux niveaux. Le niveau 1 est le répertoire, et est utilisé pour déterminer quelle capsule gère quelle transaction. Le niveau 2 se compose des capsules. Lorsque le niveau 1 identifie une capsule, la configuration achemine chaque transaction vers le serveur approprié dans la capsule, qui est généralement le serveur contenant la partition pour les données utilisées par la transaction. Le cas échéant, vous pouvez également utiliser un cache proche sur le niveau 1 afin de minimiser l'impact généré par la recherche de la capsule adéquate.

L'utilisation de capsules est légèrement plus complexe que l'utilisation d'une grille de données unique, mais le fonctionnement, les tests et l'amélioration de la fiabilité en font un élément essentiel du test d'évolutivité.

Disponibilité : présentation générale

Haute disponibilité

Grâce à sa haute disponibilité, WebSphere eXtreme Scale garantit une grande fiabilité dans la redondance des données et la détection des échec.

WebSphere eXtreme Scale organise les grilles de données machines virtuelles Java dans une arborescence fédérée souple. Le service de catalogue à la racine et dans les groupes centraux contenant les conteneurs sont les feuilles de l'arbre. Pour plus d'informations, voir «Architecture de la mise en cache : mappes, conteneurs, clients et catalogues», à la page 11.

Chaque groupe central est automatiquement créé par le service de catalogue en groupes d'environ 20 serveurs. Les membres de ce groupe assurent la surveillance de la santé des autres membres. Chaque groupe central choisit un membre qui aura la responsabilité de communiquer les informations relatives au groupe au service de catalogue. Lorsque la taille du groupe central est limitée, la surveillance de la santé et l'évolutivité de l'environnement s'améliorent.

Remarque : Dans un environnement WebSphere Application Server, dans lequel la taille du groupe central peut être modifiée, eXtreme Scale ne prend pas en charge plus de 50 membres par groupe central.

Signaux de présence

1. Les sockets restent ouverts entre les machines virtuelles Java et si un socket se ferme de manière inattendue, cette fermeture est détectée comme un incident de la machine virtuelle Java homologue. Cette détection intercepte les incidents tels que l'arrêt très rapide d'une machine virtuelle Java. Une telle détection permet généralement une reprise en ligne de moins d'une seconde après ces types d'incident.
2. Les autres types d'incident incluent : un blocage du système d'exploitation, un problème de serveur physique ou une panne réseau. Ces incidents sont détectés par l'intermédiaire des signaux de présence.

Des signaux de présence sont envoyés de manière périodique entre des paires de processus : Si un nombre donné de signaux de présence sont manquants, un incident est supposé. Cette méthode détecte les erreurs en N*M secondes. N est le nombre de pulsations manquées et M correspond à la fréquence des pulsations. La définition directe de M et N n'est pas prise en charge. Un mécanisme mobile est utilisé pour permettre d'utiliser une plage de combinaisons M et N.

Echecs

Il existe plusieurs types d'échecs des processus. Un processus peut échouer car la limite des ressources (par exemple la taille maximale des segments de mémoire) a été atteinte ou parce qu'une logique de contrôle de processus a mis fin à un processus. Ceci risque alors d'entraîner un échec du système d'exploitation et la perte des processus en cours d'exécution. Moins fréquemment, une défaillance du matériel, par exemple de la carte d'interface réseau, peut se produire : le système d'exploitation est alors déconnecté du réseau. D'autres échecs peuvent survenir et entraîner l'indisponibilité du processus. Dans ce contexte, les échecs peuvent être classés en deux types : échec de processus et perte de connectivité.

Echec de processus

WebSphere eXtreme Scale réagit rapidement aux erreurs de processus. Lorsqu'un processus échoue, le système d'exploitation est responsable du nettoyage des ressources utilisées par celui-ci. Ce nettoyage comprend l'allocation de port et la connectivité. Lorsqu'un processus échoue, un signal est envoyé via les connexions utilisées par ce processus pour fermer celles-ci. Ce signal permet aux autres processus connectés au processus ayant échoué de détecter instantanément cet échec.

Perte de connectivité

Une perte de connectivité se produit lorsque le système d'exploitation est déconnecté. Il ne parvient alors plus à envoyer des signaux à d'autres processus. Plusieurs raisons peuvent expliquer la perte de connectivité. Elles peuvent être classées en deux catégories : échec de l'hôte ou îlotage.

Echec de l'hôte

Si la prise d'alimentation est débranchée, la machine cesse immédiatement de fonctionner.

Ilotage

Ce scénario constitue l'échec le plus complexe à gérer par le logiciel car le processus est censé être indisponible, alors qu'il ne l'est pas. Plus simplement, un serveur ou un autre processus semble avoir échoué alors qu'il s'exécute correctement.

Défaillances de conteneur

L'échec d'un conteneur est généralement identifié par des conteneurs homologues via le mécanisme du groupe central. Lorsqu'un conteneur ou un jeu de conteneurs échoue, le service de catalogue migre les fragments hébergés par ce ou ces conteneurs. Le service de catalogue commence par rechercher un fragment réplique synchrone avant de procéder à la migration vers un fragment réplique asynchrone. Une fois les fragments primaires migrés vers les nouveaux conteneurs, le service de catalogue recherche de nouveaux conteneurs hôtes pour les fragments réplique manquants.

Remarque : Isolement de conteneur : le service de catalogue migre les fragments d'un conteneur lorsque le conteneur est réputé être indisponible. S'il redevient disponible, le service de catalogue considèrent que des fragments peuvent à nouveau y être positionnés, comme dans le flux de démarrage normal.

Latence de détection des défaillances de conteneur

Les échecs peuvent être classés en deux catégories : les échecs liés aux logiciels et les échecs liés au matériel. Les échecs liés aux logiciels se produisent généralement lorsqu'un processus échoue. De tels incidents sont détectés par le système d'exploitation qui peut récupérer les ressources utilisées, par exemple les sockets réseau, rapidement. Cette détection se fait en général en moins d'une seconde. Les défaillances matérielles peuvent nécessiter un délai de détection de 200 avec l'optimisation par défaut des pulsations. Ces problèmes incluent les blocages des machines physiques, les déconnexions de câbles réseau ou les défaillances de système d'exploitation. Ce délai repose sur des signaux pour détecter les problèmes matériels qui peuvent être configurés.

Echec du service de catalogue

La grille du service de catalogue étant aussi une grille eXtreme Scale, elle utilise le mécanisme de regroupement central de la même manière que le processus d'échec des conteneurs, à cette différence près : le domaine de services de catalogue utilise une sélection d'homologue pour définir le fragment primaire plutôt que l'algorithme de service de catalogue utilisé pour les conteneurs.

Le service de placement et le service de regroupement central sont des services Un sur N. Un service Un sur N ne s'exécute que sur un seul membre du groupe haute disponibilité. Le service de localisation et l'administration s'exécutent, quant à eux, sur tous les membres de ce groupe. Le service de positionnement et le service de regroupement central sont des singletons car ils sont responsables de l'agencement du système. Le service de localisation et d'administration sont des services en lecture seule qui existent partout à des fins d'évolutivité.

Le service de catalogue utilise la réplication pour garantir sa tolérance aux erreurs. Si un processus de service de catalogue échoue, le service redémarre pour restaurer le système au niveau de disponibilité souhaité. Si tous les processus qui hébergent le service de catalogue échouent, la grille de données a une perte de données

critique. Cet échec entraîne un redémarrage systématique de tous les serveurs de conteneur. Comme le service de catalogue peut s'exécuter sur plusieurs processus, cet échec est improbable. Toutefois, si vous exécutez tous les processus sur un même sous-système de stockage, sur un même châssis lame ou à partir d'un même commutateur réseau, un échec est très probable. Essayez de supprimer les modes d'échecs les plus communs des sous-systèmes de stockage qui hébergent le service de catalogue pour limiter les risques d'échec.

Echec de plusieurs conteneurs

Un fragment réplique n'est jamais placé dans le même processus que le fragment primaire, car en cas de perte du processus, les deux fragments seraient perdus. Dans un environnement de développement ne comprenant qu'une machine, vous pouvez prévoir deux conteneurs afin de procéder à des fragments réplique. Vous pouvez définir l'attribut du mode de développement de la stratégie de déploiement pour configurer une réplique à placer sur la même machine qu'un fragment primaire. Cependant dans un environnement de production, une seule machine n'est pas suffisante, car la perte de cet hôte entraînerait la perte des deux serveurs de conteneur. Pour passer d'un environnement de développement comptant une seule machine à un environnement de production comprenant plusieurs machines, désactivez le mode de développement dans le fichier de configuration de la règle de déploiement.

Tableau 4. Récapitulatif de la reconnaissance d'échec et de la reprise en ligne

Type de perte	Mécanisme de reconnaissance (détection)	Méthode de récupération
Perte de processus	E-S	Redémarrage
Perte de serveur	Signal de présence	Redémarrage
Indisponibilité du réseau	Signal de présence	Rétablissement du réseau et de la connexion
Blocage côté serveur	Signal de présence	Arrêt et redémarrage du serveur
Serveur occupé	Signal de présence	Attendre que le serveur soit disponible

Réplication à des fins de disponibilité

La réplication fournit la tolérance aux pannes et augmente les performances d'une topologie eXtreme Scale répartie. La réplication est activée en associant des mappes de sauvegarde à un groupe de mappes.

A propos des groupes de mappes

Un groupe de mappes est un ensemble de mappes catégorisées par une clé de partition. Cette clé de partition provient de la clé de la mappe en prenant son hachage modulo le nombre de partitions. Si un groupe de mappes au sein du groupe de mappes a une clé de partition X, ces mappes sont stockées dans la partition X correspondant dans la grille de données. Si un autre groupe a une clé de partition Y, toutes les mappes sont stockées dans la partition Y, et ainsi de suite. Les données dans les mappes sont répliquées en fonction de la stratégie définie dans le groupe de mappes. La réplication a lieu dans des topologies distribuées.

Les groupes de mappes sont affectés du nombre de partition et d'une règle de réplication. La configuration de la réplication du groupe de mappes identifie le

nombre de fragments de réplique synchrone et asynchrone que doit avoir le groupe de mappes en plus du fragment primaire. Par exemple, s'il existe une réplique synchrone et une réplique asynchrone, chacune des mappes de sauvegarde BackingMaps affectée au groupe de mappes ont un fragment de réplique distribué automatiquement dans le groupe de serveurs de conteneur disponibles pour la grille de données. La configuration de la réplication peut également permettre aux clients de lire les données depuis les serveurs répliqués de manière synchrone. Cela peut étaler la charge des demandes de lecture sur d'autres serveurs de la grille eXtreme Scale. La réplication a un impact sur le modèle de programmation lorsque vous préchargez les mappes de sauvegarde.

Préchargement des mappes

Les mappes peuvent être associées à des loaders. Un loader sert à aller chercher des objets quand ils sont introuvables dans la mappe (absence du cache) et il sert également à écrire les modifications à un dorsal lors de la validation des transactions. Les loaders peuvent également servir à précharger des données dans une mappe. La méthode preloadMap de l'interface Loader est appelée sur chaque mappe lorsque sa partition correspondante dans le groupe de mappes devient la partition principale. La méthode preloadMap n'est pas appelée sur les répliques. Cette méthode tente de charger dans la mappe à partir du dorsal toutes les données de référence concernées à l'aide de la session fournie. La mappe pertinente est identifiée par l'argument BackingMap qui est passé à la méthode preloadMap.

```
void preloadMap(Session session, BackingMap backingMap) throws LoaderException;
```

Préchargement dans un groupe de mappes partitionné

Les mappes peuvent être partitionnées en N partitions. Elles peuvent donc s'étendre sur plusieurs serveurs, chaque entrée étant identifiée par une clé qui n'est stockée que sur l'un de ces serveurs. Les mappes de très grande taille peuvent être détenues sur un eXtreme Scale car l'application n'est plus limitée par la taille du segment d'une seule JVM pour contenir toutes les entrées d'une mappe. Les applications qui veulent effectuer un préchargement avec la méthode preloadMap de l'interface Loader doivent identifier le sous-ensemble des données à précharger. Les partitions existent toujours en nombre fixe. Il est possible de déterminer ce nombre à l'aide de cet exemple de code :

```
int numPartitions = backingMap.getPartitionManager().getNumOfPartitions();
int myPartition = backingMap.getPartitionId();
```

Cet exemple de code montre qu'une application peut identifier le sous-ensemble des données préchargées depuis la base de données. Les applications doivent toujours utiliser ces méthodes même si la mappe n'est pas initialement partitionnée. Ces méthodes permettent la flexibilité : si la mappe est ultérieurement partitionnée par les administrateurs, le loader continuera à opérer correctement.

L'application doit émettre des requêtes pour extraire du dorsal le sous-ensemble *myPartition*. Si une base de données est utilisée, il peut être plus facile d'avoir une colonne avec l'identificateur de partition d'un enregistrement donné à moins qu'il n'y ait une requête naturelle permettant aux données de la table de se partitionner facilement.

Performances

L'implémentation du préchargement copie dans la mappe les données à partir du dorsal en stockant plusieurs objets dans la mappe en une seule transaction. Le nombre optimal d'enregistrements à stocker par transaction dépend de plusieurs facteurs, notamment la complexité et la taille. Ainsi, après que la transaction comprend des blocs de plus de 100 entrées, les avantages en termes de performances diminuent au fur et à mesure que l'on augmente le nombre des entrées. Pour déterminer le nombre optimal, commencez par 100 entrées, puis augmentez le nombre jusqu'à ce que les performances deviennent nulles. Les transactions de grande taille donnent de meilleures performances de réplication. N'oubliez pas que seul le fragment primaire exécute le code de préchargement. Les données préchargées sont répliquées depuis le fragment primaire vers les fragments réplique qui sont en ligne.

Préchargement des groupes de mappes

Si l'application utilise un groupe de mappes avec plusieurs mappes, chaque mappe a son propre chargeur. Chaque chargeur a une méthode preload. Chaque mappe est chargée en série par eXtreme Scale. Ce sera plus efficace de précharger toutes les mappes en désignant une mappe comme la mappe de préchargement. Ce processus est une convention d'application. On pourrait, par exemple, avoir deux mappes, Department et Employee, qui utilisent le loader Department pour précharger les deux mappes. Cette procédure garantit que, de manière transactionnelle, si une application veut un département, les salariés de ce département seront dans le cache. Lorsque le loader Department précharge un département depuis le dorsal, il récupère également les salariés de ce département. L'objet Department et les objets Employee qui lui sont associés sont ajoutés à la mappe à l'aide d'une seule transaction.

Préchargement récupérable

Il arrive que les clients aient des ensembles de données de très grosse taille et qui nécessitent d'être mis en cache. Précharger ces données peut prendre énormément de temps. Parfois, le préchargement doit être terminé pour que l'application puisse aller en ligne. C'est là que rendre récupérable le préchargement devient intéressant. Supposons qu'il y ait un million d'enregistrements à précharger. Le fragment primaire les télécharge et échoue au 800 000e enregistrement. En principe, le fragment réplique choisi pour être le nouveau fragment primaire efface tout état répliqué et repart du début. eXtreme Scale peut utiliser une interface ReplicaPreloadController. Le loader de l'application aura également besoin d'implémenter cette interface. Notre exemple ajoute une seule méthode au loader : `Status checkPreloadStatus(Session session, BackingMap bmap);`. Cette méthode est appelée par l'environnement d'exécution eXtreme Scale avant la méthode preload de l'interface Loader. eXtreme Scale teste le résultat de cette méthode (Status) pour déterminer son comportement au cas où un fragment réplique passe au statut de fragment primaire.

Tableau 5. Valeur du statut et réponse

Valeur de statut retournée	Réponse d'eXtreme Scale
Status.PRELOADED_ALREADY	eXtreme Scale n'appelle pas du tout la méthode preload car ce statut indique que la mappe est complètement préchargée.
Status.FULL_PRELOAD_NEEDED	eXtreme Scale efface la mappe et appelle de manière normale la méthode preload.

Tableau 5. Valeur du statut et réponse (suite)

Valeur de statut retournée	Réponse d'eXtreme Scale
Status.PARTIAL_PRELOAD_NEEDED	eXtreme Scale laisse la mappe comme elle est et appelle la méthode preload. Cette stratégie permet au loader de l'application de continuer le préchargement à partir du point où il en était resté.

A l'évidence, lorsqu'un fragment primaire précharge la mappe, il doit laisser un état dans une mappe du groupe de mappes à répliquer pour que la réplique détermine l'état à retourner. Vous pouvez utiliser une mappe supplémentaire appelée, par exemple, RecoveryMap. Cette mappe doit faire partie du groupe de mappes à précharger pour que la mappe soit répliquée de manière cohérente avec les données à précharger. Nous allons suggérer une implémentation.

Lorsque le préchargement valide chaque bloc d'enregistrements, dans le cadre de cette transaction, le processus actualise également un compteur ou une valeur dans la RecoveryMap. Les données préchargées et celles de la RecoveryMap sont répliquées de manière atomique vers les fragments réplique. Lorsque le fragment réplique passe au statut de fragment primaire, il est à présent en mesure de vérifier la RecoveryMap pour voir ce qui s'est passé.

La RecoveryMap peut contenir une seule entrée avec la clé d'état. Si aucun objet n'existe pour cette clé, vous avez besoin d'une méthode complète preload (checkPreloadStatus returns FULL_PRELOAD_NEEDED). Si un objet existe pour cette clé d'état et que la valeur est COMPLETE, le préchargement prend fin et la méthode checkPreloadStatus retourne PRELOADED_ALREADY. Autrement, l'objet value indique le point de redémarrage du préchargement et la méthode checkPreloadStatus retourne `PARTIAL_PRELOAD_NEEDED`. Le loader peut stocker le point de récupération dans une variable d'instance du loader de manière à ce que, lorsque le préchargement est appelé, le loader sache d'où partir. La RecoveryMap peut également détenir une entrée par mappe si chacune des mappes est préchargée de manière indépendante.

Gestion de la récupération en mode de réplication synchrone avec un chargeur

L'environnement d'exécution d'eXtreme Scale est conçu pour ne pas perdre de données validées en cas de défaillance du fragment primaire. Nous allons voir quels algorithmes sont utilisés à cet effet. Ces algorithmes ne s'appliquent que lorsqu'un groupe de réplication utilise la réplication synchrone. L'usage d'un loader n'est pas obligatoire.

Il est possible de configurer l'environnement d'exécution d'eXtreme Scale pour répliquer de manière synchrone vers les fragments réplique toutes les modifications d'un fragment primaire. Lorsqu'il est placé, un fragment réplique synchrone reçoit une copie des données existant dans le fragment primaire. Pendant ce temps, le fragment primaire continue de recevoir des transactions qu'il copie de manière asynchrone vers le fragment de réplique. A ce moment-là, le fragment réplique n'est pas encore considéré comme étant en ligne.

Une fois que le fragment réplique a rattrapé le fragment primaire, il passe en mode homologue et la réplication synchrone peut commencer. Toute transaction validée sur le fragment primaire est envoyée aux fragments réplique synchrones et le fragment primaire attend la réponse de chacun de ces fragments. Une séquence de validation synchrone avec un loader dans le fragment primaire ressemble à la procédure suivante :

Tableau 6. Séquence de validation dans le fragment primaire

Avec loader	Sans loader
Obtention des verrous pour les entrées	Identique
Vidage des modifications vers le loader	"No operation"
Enregistrement des modifications dans le cache	Identique
Envoi des modifications aux fragments de réplique et attente d'accusé de réception	Identique
Validation vers le loader via le plug-in TransactionCallback	Appel de validation de plug-in, mais sans effet
Libération des verrous pour les entrées	Identique

Vous remarquerez que les modifications sont envoyées aux fragments réplique avant d'être validées vers le loader. Pour déterminer lorsque les modifications sont validées dans le fragment réplique, modifiez cette séquence : lors de l'initialisation, initialisez de la manière suivante les listes tx dans le fragment primaire.

```
CommitedTx = {}, RolledBackTx = {}
```

Pendant le traitement synchrone des validations, utilisez la séquence suivante :

Tableau 7. Traitement synchrone des validations

Avec loader	Sans loader
Obtention des verrous pour les entrées	Identique
Vidage des modifications vers le loader	"No operation"
Enregistrement des modifications dans le cache	Identique
Envoi des modifications avec une transaction validée, annulation de la transaction dans le fragment de réplique et attente d'accusé de réception	Identique
Effacement de la liste des transactions validées et des transactions annulées	Identique
Validation vers le loader via le plug-in TransactionCallBack	La validation via le plug-in TransactionCallBack est toujours appelée, mais en principe sans effet
Si la validation réussit, ajout de la transaction aux transactions validées, sinon, ajout aux transactions annulées	"No operation"
Libération des verrous pour les entrées	Identique

Pour le traitement des fragments réplique, utilisez la séquence suivante :

1. Réception des modifications
2. Validation de toutes les transactions reçues dans la liste des transactions validées
3. Annulation de toutes les transactions reçues dans la liste des transactions annulées
4. Démarrage d'une transaction ou d'une session
5. Application des modifications à la transaction ou à la session
6. Enregistrement de la transaction ou de la session dans la liste en attente
7. Renvoi de la réponse

Vous remarquerez que, dans le fragment réplique, il ne se passe aucune interaction avec le loader tant que le fragment réplique est en mode réplique. C'est au fragment primaire d'impulser toutes les modifications via le loader. La réplique n'envoie pas de modifications. Un effet collatéral de cet algorithme est que le fragment réplique dispose toujours des transactions, mais celles-ci ne sont validées qu'après que la transaction primaire suivante a envoyé le statut de validation de ces transactions. Les transactions sont alors validées ou annulées dans le fragment réplique. Jusque-là, les transactions ne sont pas validées. Il est possible d'ajouter dans le fragment primaire un minuteur qui envoie le résultat de la transaction après un bref délai (quelques secondes). Ce minuteur limite, sans l'éliminer tout à fait, le décalage de ce créneau. Ce décalage n'est un problème qu'en mode de lecture de réplique. Sinon, il n'a aucun impact sur l'application.

Lorsque le fragment primaire échoue, c'est vraisemblablement que quelques transactions ont été validées ou annulées dans ce fragment primaire, mais que le message n'a jamais été transmis au fragment réplique avec ces résultats. Lorsqu'un fragment réplique passe au rang de nouveau fragment primaire, l'une de ses premières actions est de gérer cette situation. Chaque transaction en attente est traitée à nouveau par rapport à l'ensemble de mappes du nouveau fragment primaire. S'il y a un loader, chaque transaction est remise à ce dernier. Ces transactions sont appliquées dans un ordre FIFO strict. Si une transaction échoue, elle est ignorée. Si trois transactions A, B et C sont en attente, A peut être validée, B peut être annulée et C peut être aussi validée. Aucune de ces trois transactions n'a d'impact sur les autres. Supposons qu'elles soient indépendantes.

Lorsqu'il se trouve en mode de reprise par basculement, un loader pourra vouloir utiliser une logique légèrement différente de celle utilisée en mode normal. L'implémentation de l'interface ReplicaPreloadController permet au loader de savoir facilement lorsqu'il est en mode de reprise par basculement. La méthode checkPreloadStatus n'est appelée que lorsque la reprise par basculement est terminée. Par conséquent, si la méthode apply de l'interface Loader est appelée avant la méthode checkPreloadStatus, il y a une transaction de récupération. Après l'appel de la méthode checkPreloadStatus, la reprise par basculement est terminée.

Equilibrage de la charge entre les fragments réplique

eXtreme Scale, sauf configuration différente, envoie au serveur primaire toutes les demandes de lecture et d'écriture concernant un groupe de réplication donné. Ce serveur primaire doit servir toutes les demandes émanant des clients. Vous voudrez peut-être autoriser l'envoi des demandes de lecture aux répliques du fragment primaire. L'envoi des demandes de lecture aux fragments réplique permet à la charge de ces demandes d'être partagées par plusieurs machines virtuelles Java. Cela dit, il faut savoir que l'utilisation des fragments réplique pour les demandes de lecture peut donner des réponses incohérentes.

Equilibrer la charge entre les fragments réplique s'utilise en général uniquement lorsque les clients mettent en cache des données qui changent en permanence ou lorsque les clients utilisent le verrouillage pessimiste.

Si les données changent en permanence et qu'elles sont ensuite invalidées dans les caches locaux du client, le fragment primaire devrait constater en résultat un taux relativement élevé de demandes get provenant des clients. De même, en mode de verrouillage pessimiste, il n'existe aucun cache local, c'est pourquoi toutes les demandes sont envoyées au fragment primaire.

Si les données sont relativement statiques ou que le mode pessimiste n'est pas utilisé, l'envoi des demandes de lecture au fragment de réplique n'a pas un énorme impact sur les performances. La fréquence des demandes get émanant des clients avec des caches pleins de données n'est pas élevée.

Lors du premier démarrage d'un client, son cache local est vide. Les demandes adressées au cache vide sont transmises au fragment primaire. Au fil du temps, le cache client obtient des données, ce qui fait tomber la charge des demandes. Si un grand nombre de clients démarrent simultanément, la charge peut être importante et la lecture des fragments de réplique peut être un choix approprié.

Réplication côté client

Avec eXtreme Scale, vous pouvez répliquer une mappe serveur vers un ou plusieurs clients à l'aide de la réplication asynchrone. Un client peut demander une copie locale en lecture seule d'une mappe côté serveur à l'aide de la méthode ClientReplicableMap.enableClientReplication.

```
void enableClientReplication(Mode mode, int[] partitions, ReplicationMapListener
listener) throws ObjectGridException;
```

Le premier paramètre est le mode de réplication. Il peut s'agir d'une réplication continue ou d'une réplication instantanée. Le deuxième paramètre est une matrice de partitions représentant les partitions à partir desquelles la réplication doit se faire. Si la valeur est nulle ou si la matrice est vide, les données sont répliquées à partir de toutes les partitions. Le dernier paramètre est programme d'écoute permettant de recevoir les événements de réplication du client. Pour plus d'informations, voir les sections sur ClientReplicableMap et ReplicationMapListener dans la documentation relative aux API.

Une fois la réplication activée, le serveur démarre le processus de réplication de la mappe vers le client. A tout moment, le client est en retard de quelques transactions seulement par rapport au serveur.

Service de catalogue à haute disponibilité

Un domaine de services de catalogue est la grille de données des serveurs de catalogues que vous utilisez, qui conservent les informations relatives à la topologie de tous les conteneurs de votre environnement eXtreme Scale. Le service de catalogue contrôle les fonctions d'équilibrage et de routage pour tous les clients. Pour déployer eXtreme Scale en tant qu'espace de traitement de la base de données en mémoire, vous devez regrouper le service de catalogue en un cluster de domaine de services de catalogue afin de garantir la haute disponibilité.

Composants du domaine de services de catalogue

Lorsque plusieurs serveurs de catalogues démarrent, l'un d'eux est choisi comme serveur maître acceptant les signaux de présence IIOP (Internet Inter-ORB Protocol) et gérant les modifications des données du système consécutives aux modifications apportées aux services de catalogue ou aux conteneurs.

Lorsque des clients contactent l'un des serveurs de catalogues, la table de routage du domaine de services de catalogue est propagée vers ces clients via le contexte de service CORBA (Common Object Request Broker Architecture).

Configurez au moins trois serveurs de catalogues. Les serveurs de catalogue doivent être installés sur des noeuds distincts ou dans des images d'installation distinctes de vos serveurs de conteneur pour pouvoir mettre à niveau de manière

transparente vos serveurs ultérieurement. Si votre configuration contient des zones, vous pouvez configurer un serveur de catalogues par zone.

Lorsqu'un serveur conteneur eXtreme Scale contacte l'un des serveurs de catalogues, la table de routage du domaine de services de catalogue est également propagée au serveur conteneur eXtreme Scale via le contexte de service CORBA. En outre, si le serveur contacté n'est pas le serveur maître, la demande est automatiquement redirigée vers le serveur maître actuel et la table de routage du serveur de catalogues est mise à jour.

Remarque : Un domaine de services de catalogue et la grille de données des serveurs de conteneur sont très différents. La première, le domaine de services de catalogue, permet de garantir la haute disponibilité de vos données système. La grille de données du serveur de conteneur est destinée à la haute disponibilité des données, l'évolutivité et la gestion de la charge de travail. Par conséquent, il existe deux tables différentes de routage : la table de routage du domaine de services de catalogue et la table de routage pour les fragments de la grille de données du serveur de conteneur.

Les responsabilités du domaine de services de catalogue sont divisées en une série de services :

Gestionnaire du groupe central

Le service de catalogue utilise le gestionnaire de haute disponibilité (gestionnaire HA) pour regrouper des processus en vue de la surveillance de la disponibilité. Chaque regroupement est un groupe central. Avec eXtreme Scale, le gestionnaire du groupe central regroupe les processus de manière dynamique. Ces processus doivent être de petite taille afin de permettre l'évolutivité. Chaque groupe central choisit un responsable qui sera chargé d'envoyer un statut au gestionnaire du groupe central en cas de défaillance de certains membre. Le même mécanisme de statut est utilisé pour identifier une éventuelle défaillance de tous les membres d'un groupe, qui pourrait entraîner un échec de la communication avec le responsable.

Le gestionnaire du groupe central est un service entièrement automatique responsable de l'organisation des conteneurs en petits groupes de serveurs qui sont alors automatiquement fédérés de manière souple pour constituer une grille d'objets. Lorsqu'un conteneur contacte le service de catalogue pour la première fois, il attend d'être affecté à un nouveau groupe ou à un groupe existant. Un déploiement eXtreme Scale consiste en plusieurs groupes de ce type et ce regroupement est une tâche d'activation essentielle pour l'évolutivité. Chaque groupe est un groupe de machines virtuelles Java utilisant les signaux de présence pour surveiller la disponibilité des autres groupes. L'un des membres de ce groupe est choisi comme responsable. Il est chargé de relayer les informations relatives à la disponibilité au service de catalogue afin de lui permettre de réagir aux défaillances en procédant à des réallocations et à des réacheminements.

Service de positionnement

Le service de catalogue gère le placement des fragments dans l'ensemble des serveurs de conteneur disponibles. Le service de placement est responsable du maintien de l'équilibre entre les différentes ressources physiques. Il prend également en charge l'allocation des fragments à leur conteneur hôte. Il s'exécute en tant que service Un sur N choisi dans la grille de données afin qu'il ne s'exécute qu'une instance et une seule de ce service. Si cette instance échoue, un autre processus est choisi et il prend le

relais. Pour garantir la redondance, l'état du service de catalogue est répliqué sur tous les serveurs qui hébergent le service de catalogue.

Administration
Le service de catalogue constitue le point d'entrée logique de l'administration du système. Le service de catalogue héberge un bean géré (MBean) et fournit des URL Java Management Extensions (JMX) pour tous les serveurs que gère le service de catalogue.

Service de localisation
Le service de localisation fait office de point de contact pour les clients qui recherchent les conteneurs hébergeant l'application voulue, ainsi que pour les serveurs de conteneurs qui enregistrent les applications hébergées auprès du service de placement. Le service de localisation s'exécute sur tous les membres de la grille de données pour étendre cette fonction.

Déploiement du domaine de services de catalogue

Le service de catalogue héberge une logique qui est généralement inactive lorsque l'état est stabilisé. Il a donc très peu d'incidence sur l'évolutivité. Il permet de gérer des centaines de conteneurs devenant disponibles simultanément. Pour la disponibilité, configurez le service de catalogue dans une grille de données.

Planification

Une fois qu'un domaine de services de catalogue a démarré, les membres de la grille de données se lient. La topologie du domaine de services de catalogue doit faire l'objet d'une planification prudente et méticuleuse, car il sera impossible de modifier la configuration du domaine au moment de l'exécution. Etendez la grille de données de manière aussi diversifiée que possible pour éviter d'éventuelles erreurs.

Démarrage d'un domaine de services de catalogue

Pour plus d'informations sur la création d'un domaine de services de catalogue, voir les informations relatives au démarrage d'un service de domaine de catalogue dans *Guide d'administration*.

Connexion à un domaine autonome de services de catalogue des conteneurs eXtreme Scale intégrés à WebSphere Application Server

Vous pouvez configurer des conteneurs eXtreme Scale qui sont imbriqués dans un environnement WebSphere Application Server pour qu'ils se connectent à un domaine autonome de services de catalogue.

- Vous pouvez créer des domaine de services de catalogue dans la console d'administration de WebSphere Application Server. Voir Création de domaines de services de catalogue dans WebSphere Application Serverles informations relatives à la création de domaines de services de catalogue dans *Guide d'administration* pour plus d'informations.
-  (déprécié) Dans les précédentes versions, la connexion de services de catalogue à un domaine de services de catalogue s'effectuait par la création d'une propriété personnalisée. Cette propriété est toujours utilisable, mais son utilisation n'est pas encouragée car elle est considérée comme déprécié. Pour plus d'informations sur cette propriété personnalisée, voir Démarrage et arrêt des serveurs dans un environnement WebSphere Application Server les

informations relatives au processus de service de catalogue dans un WebSphere Application Server dans *Guide d'administration*..

Remarque : Collision des noms de serveur : cette propriété étant utilisée pour démarrer le serveur de catalogues eXtreme Scale et pour s'y connecter, un serveur de catalogues ne doit pas porter le même nom qu'un serveur WebSphere Application Server.

Voir «Quorums de serveurs de catalogue» pour plus d'informations

Quorums de serveurs de catalogue

Lorsque le mécanisme du quorum est activé, tous les serveurs de catalogue dans le quorum doivent être disponibles pour que les opérations de placement puissent être exécutées dans la grille de données.

Termes importants à connaître

- **Pulsation** : signal qui est envoyé entre les serveurs pour indiquer qu'ils sont en cours d'exécution.
- **Quorum** : groupe de serveurs de catalogue qui peuvent communiquer et effectuer des opérations de placement dans la grille de données. Ce groupe se compose de tous les serveurs de catalogue dans la grille de données, sauf si vous remplacez manuellement le mécanisme du quorum par des actions d'administration.
- **Microcoupure** : perte provisoire de connectivité entre un ou plusieurs serveurs.
- **Interruption totale** : perte définitive de la connectivité entre un ou plusieurs serveurs.
- **Centre de données** : groupe de serveurs dans une zone géographique connectés avec un réseau LAN (local area network).
- **Zone** : option de configuration qui est utilisée pour regrouper des serveurs ayant en commun une caractéristique physique. Un centre de données, un réseau de zone, un bâtiment ou un étage de bâtiment sont des exemples de zones.

Pulsations et détection des incidents

Serveurs de conteneur et groupes centraux

Le service de catalogue place les serveurs de conteneur dans des groupes centraux dont la taille est limitée. Un groupe central essaie de détecter la défaillance de ses membres. Un membre du groupe central est déclaré responsable du groupe. Il indique régulièrement au service de catalogue que le groupe central est actif et lui signale les modifications des membres. Une modification de membre peut être une machine JVM défaillante ou une machine JVM ajoutée au groupe central.

Si un socket JVM est fermé, la machine JVM est considérée ne plus être disponible. Chaque membre du groupe central envoie également des pulsations sur ces sockets à une fréquence définie par la configuration. Si une machine JVM ne répond pas à

ces pulsations dans un délai maximum défini, la machine JVM est considérée ne plus être disponible, ce qui déclenche un échec de détection.

Si le service de catalogue marque une machine virtuelle Java de conteneur comme étant défaillante et que le serveur de conteneur est ultérieurement signalé comme étant disponible, il est demandé à la machine virtuelle d'arrêter le serveur de conteneur WebSphere eXtreme Scale. Une machine virtuelle Java dans cet état n'est pas visible dans les requêtes de commande de l'utilitaire **xscmd**. Des messages dans les journaux de la machine virtuelle Java conteneur indiqueront que cette machine est tombée en panne. Vous devrez la redémarrer manuellement.

Si le responsable du groupe central ne parvient pas à contacter un membre, il continue d'essayer de contacter le membre.

La défaillance complète de la totalité des membres d'un groupe central est une éventualité qui peut arriver. Si la totalité du groupe central est défaillant, c'est au service de catalogue de détecter cette perte.

Pulsations du domaine de services de catalogue

Le domaine de services de catalogue ressemble à un groupe central privé dont les membres sont statiques avec un mécanisme de quorum. La détection des défaillances s'effectue de la même manière que pour un groupe central normal. La différence tient à la modification de son comportement puisqu'il inclut une logique de quorum. Le service de catalogue utilise aussi une configuration de pulsation moins agressive.

Détection des défaillances

WebSphere eXtreme Scale détecte la fin des processus via des événements de fermeture de socket anormale. Le service de catalogue est immédiatement notifié lorsqu'un processus prend fin.

Pour plus d'informations sur la configuration des pulsations, voir Optimisation de la valeur de l'intervalle des pulsations pour la détection des basculements les informations relatives à la configuration de la détection du basculement dans *Guide d'administration*.

Comportement de quorum

Normalement, les membres du service de catalogue disposent d'une connectivité pleine et entière. Le domaine de services de catalogue est un ensemble statique de machines virtuelles Java. WebSphere eXtreme Scale s'attend à ce que tous les membres du service de catalogue soient en ligne. Lorsque tous les membres sont en ligne, le service de catalogue a le quorum. Le service de catalogue répond aux événements de conteneur uniquement lorsque le service de catalogue a le quorum.

Causes de la perte du quorum

WebSphere eXtreme Scale s'attend à perdre le quorum dans les cas suivants :

- Un membre JVM de service de catalogue est défaillant.
- Une microcoupure réseau se produit.
- Une perte de centre de données se produit.

WebSphere eXtreme Scale ne perd pas le quorum dans les cas suivants :

- Arrêt d'une instance de serveur de catalogue avec la commande **stopOgServer** ou toute autre opération d'administration. Le système sait que l'instance de serveur a été arrêtée, ce qui est différent d'une défaillance de la machine JVM ou d'une microcoupure.

Si le service de catalogue perd un quorum, il s'attend à ce que le quorum soit rétabli. Lorsque le service de catalogue ne dispose pas d'un quorum, il ignore les événements provenant des serveurs de conteneur. Les serveurs conteneurs continuent d'essayer toutes les demandes qui sont rejetées par le serveur de catalogue pendant ce temps. Les pulsations sont suspendues jusqu'à ce qu'un quorum soit rétabli.

Perte de quorum due à une défaillance de machine virtuelle Java

Un serveur de catalogue défaillant provoque la perte du quorum. Si une machine JVM échoue, vous devez remplacer le quorum aussi rapidement que possible. Le service de catalogue défaillant ne peut rejoindre la grille de données tant que le quorum n'a pas été redéfini.

Perte de quorum due à une microcoupure réseau

WebSphere eXtreme Scale est conçu pour prendre en charge les microcoupures éventuelles. Une microcoupure est une perte provisoire de connectivité entre des centres de données. Elles sont généralement transitoires et disparaissent en quelques secondes ou minutes. Bien que WebSphere eXtreme Scale essaie de maintenir le fonctionnement pendant la microcoupure, une microcoupure est considérée comme une simple défaillance. La défaillance est censée être résolue et le fonctionnement normal reprend sans intervention.

Une microcoupure qui s'éternise ne pourra être requalifiée en interruption totale que par une intervention d'utilisateur. Le remplacement du quorum sur un côté de la microcoupure est nécessaire pour que l'événement soit classé comme une interruption.

Cycle entre les services de catalogue

Si un serveur de catalogue est arrêté à l'aide de la commande **stopOgServer**, le quorum diminue d'un serveur. Les serveurs restants ont toujours le quorum. Le redémarrage du serveur de catalogue rétablit le nombre précédent du quorum.

Conséquences de la perte de quorum

Si une machine virtuelle Java conteneur est défaillante pendant que la perte du quorum, la reprise n'a lieu qu'après la récupération du quorum. Dans un scénario d'interruption, la récupération ne se produit que lorsque vous exécutez la commande de remplacement de quorum. La perte de quorum et la défaillance d'un conteneur sont considérées être une double défaillance, ce qui est un événement rare. En raison de la double défaillance, les applications peuvent perdre l'accès en écriture aux données qui étaient stockées sur la machine JVM défaillante. Lorsque le quorum est restauré, la récupération normale est exécutée.

De même, si vous tentez de démarrer un conteneur au cours d'un événement de perte de quorum, le conteneur ne démarre pas.

La connectivité clients complète est autorisée pendant la perte de quorum. S'il ne se produit aucune défaillance de conteneur ou de problèmes de connectivité pendant la perte du quorum, les clients pourront toujours interagir complètement avec les serveurs conteneurs.

Si une microcoupure se produit, certains clients peuvent ne pas avoir accès aux copies primaires ou aux copies de réplique des données jusqu'à la fin de microcoupures.

Les nouveaux clients peuvent être démarrés, car une machine virtuelle Java de service de catalogue doit exister dans chaque centre de données. Par conséquent, au moins un serveur de catalogue est accessible au client, même pendant une microcoupure.

Rétablissement du quorum

Si le quorum est perdu pour une quelconque raison, lorsque le quorum est rétabli, un protocole de récupération est exécuté. Lorsque la perte du quorum se produit, toute vérification d'activité des groupes centraux est suspendue et les rapports signalant des défaillances sont ignorés. Une fois le quorum est rétabli, le service de catalogue contrôle tous les groupes centraux pour déterminer immédiatement leurs membres. Tous les fragments précédemment hébergés sur des machines virtuelles Java signalées comme étant défaillantes sont récupérés. En cas de perte de fragments primaires, les répliques survivantes sont déclarées fragments primaires. Si les fragments de réplique ont été perdus, des fragments de réplique supplémentaires sont créés à partir des survivants.

Redéfinir le quorum

Remplacez le quorum uniquement en cas de défaillance d'un centre de données. Une perte de quorum suite à la défaillance d'une machine virtuelle Java de service de catalogue ou à une microcoupure du réseau est résolue automatiquement une fois la machine virtuelle Java de service de catalogue redémarrée ou la microcoupure du réseau terminée.

Seuls les administrateurs sont informés d'une défaillance de centre de données. WebSphere eXtreme Scale traite une microcoupure et une interruption de la même manière. Vous devez informer l'environnement WebSphere eXtreme Scale de ces échecs avec la commande **xscmd -c overrideQuorum**. Cette commande indique au service de catalogue de supposer que le quorum est atteint avec les membres actuels, et la reprise complète est effectuée. Lorsque vous émettez une commande de remplacement de quorum, vous garantissez que les machines virtuelles dans le centre de données défaillant sont réellement en panne et ne peuvent pas être récupérées.

La liste qui suit envisage quelques scénarios de redéfinition de quorum. Dans ce scénario, vous disposez des trois serveurs A, B et C.

- **Microcoupure :** le serveur de catalogue, C soit provisoirement isolé. Le service de catalogue perd le quorum et attend la fin de la microcoupure. Une fois la microcoupure terminée, le serveur de catalogue C rejoint le domaine de services de catalogue et le quorum est rétabli. Votre application ne percevra aucun problème pendant ce temps.
- **Echec temporaire :** au cours d'une erreur temporaire, le serveur de catalogue C est défaillant et le service de catalogue perd le quorum. Vous devez remplacer le quorum. Une fois que le quorum est rétabli, vous pouvez redémarrer le serveur

de catalogue C. Le serveur de catalogue C rejoint le domaine de services de catalogue de nouveau lorsqu'il redémarre. Votre application ne percevra aucun problème pendant ce temps.

- **Echec de centre de données :** vous vérifiez que le centre de données est défaillant et qu'il est bien isolé sur le réseau. Ensuite, vous exécutez la commande **xscmd -c overrideQuorum**. Les deux centres de données survivants exécutent une reprise complète en remplaçant les fragments qui étaient hébergés dans le centre de données défaillant. Le service de catalogue s'exécute à présent avec un quorum complet des serveurs de catalogue A et B. L'application peut être affectée par des retards ou des exceptions entre le début de l'interruption et le remplacement du quorum. Une fois le quorum remplacé, la grille de données et le fonctionnement normale reprennent.
- **Reprise du centre de données :** les centres de données survivants fonctionnent déjà avec un quorum redéfini. Lorsque le centre de données qui contient le serveur de catalogue C est redémarré, toutes les machines virtuelles Java dans le centre de données doivent être redémarrées. Ensuite, le serveur de catalogue C rejoint le domaine de services de catalogue existant à nouveau et le paramètre de quorum revient à la situation normale sans intervention de l'utilisateur.
- **Echec de centre de données et microcoupure :** le centre de données qui contient le serveur de catalogue C est défaillant. Le quorum est redéfini et rétabli sur les centres de données restants. Si une microcoupure se produit entre les serveurs de catalogue A et B, les règles de récupération normale liés aux microcoupures s'appliquent. Une fois la microcoupure terminée, le quorum est rétabli et la récupération nécessaire après la perte du quorum se produit.

Comportement du conteneur pendant la perte du quorum

Les conteneurs hébergent un ou plusieurs fragments. Les fragments sont soit des fragments primaires, soit des fragments réplique pour une partition spécifique. Le service de catalogue affecte des fragments à un conteneur et le serveur de conteneur utilise cette affectation jusqu'à ce que de nouvelles instructions arrivent du service de catalogue. Par exemple, un fragment primaire continue d'essayer de communiquer avec ses fragments de réplique pendant les microcoupures jusqu'à ce que le service de catalogue fournisse des instructions supplémentaires pour le fragment primaire.

Comportement des fragments de réplique synchrones

Le fragment primaire peut accepter de nouvelles transactions alors que la connexion est interrompue si le nombre de répliques en ligne correspond au moins à la valeur de la propriété **minsync** du groupe de mappes. Si de nouvelles transactions sont traitées sur le fragment primaire pendant que la liaison avec la réplique synchrone est interrompue, le fragment de réplique est resynchronisé avec l'état actuel du fragment primaire lorsque la liaison est rétablie.

Ne configurez pas une réplication synchrone entre les centres de données ou sur une liaison WAN.

Comportement de la réplique asynchrone

Pendant que la connexion est interrompue, le fragment primaire peut accepter de nouvelles transactions. Le fragment primaire place en mémoire tampon les modifications jusqu'à une certaine limite. Si la connexion au fragment réplique est rétablie avant que cette limite ne soit atteinte, le fragment réplique sera actualisé avec les modifications mises en mémoire tampon. Si la limite est atteinte, le

fragment primaire détruit la liste en tampon et, lorsqu'il se reconnecte, le fragment réplique est effacé et resynchronisé.

Comportement du client pendant la perte du quorum

Les clients sont toujours en mesure de se connecter au serveur de catalogue pour s'amorcer sur la grille de données, que le domaine de services de catalogue ait ou non le quorum. Le client tente de se connecter à n'importe quelle instance de serveur de catalogue pour obtenir une table de routage et il interagit ensuite avec la grille de données La connectivité réseau peut empêcher le client d'interagir avec certaines partitions en raison de la configuration du réseau. Le client peut se connecter aux répliques locales des données distantes s'il a été configuré en conséquence. Les clients ne peuvent pas mettre à jour les données si la partition principale de ces données n'est pas disponible.

Répliques et fragments

Avec eXtreme Scale, il est possible de répliquer une base de données en mémoire ou un fragment d'une machine virtuelle Java (JVM) vers une autre. Un fragment représente une partition qui est placée dans un conteneur. Plusieurs fragments représentant différentes partitions peuvent coexister dans un même conteneur. Chaque partition a une instance qui est un fragment primaire et un nombre configurable de fragments réplique. Les fragments réplique sont synchrones ou asynchrones. Les types et le positionnement des fragments réplique est déterminé par eXtreme Scale à l'aide d'une règle de déploiement qui spécifie les nombres minimum et maximum de fragments synchrones et asynchrones.

Types de fragment

La réplication utilise trois types de fragments :

- primaire
- fragment réplique synchrone
- fragment réplique asynchrone

Le fragment primaire reçoit toutes les opérations d'insertion, d'actualisation et de suppression. Le fragment primaire ajoute et supprime des fragments réplique, réplique les données vers les fragments réplique et gère les validations et les annulations des transactions.

Les fragments réplique synchrones maintiennent le même état que le fragment primaire. Lorsqu'un fragment primaire réplique des données vers un fragment réplique synchrone, la transaction n'est validée qu'après avoir été validée dans le fragment réplique synchrone.

Les fragments réplique asynchrones ne sont pas forcément dans le même état que le fragment primaire. Lorsqu'un fragment primaire réplique des données vers un fragment réplique asynchrone, il n'attend pas la validation de ce dernier.

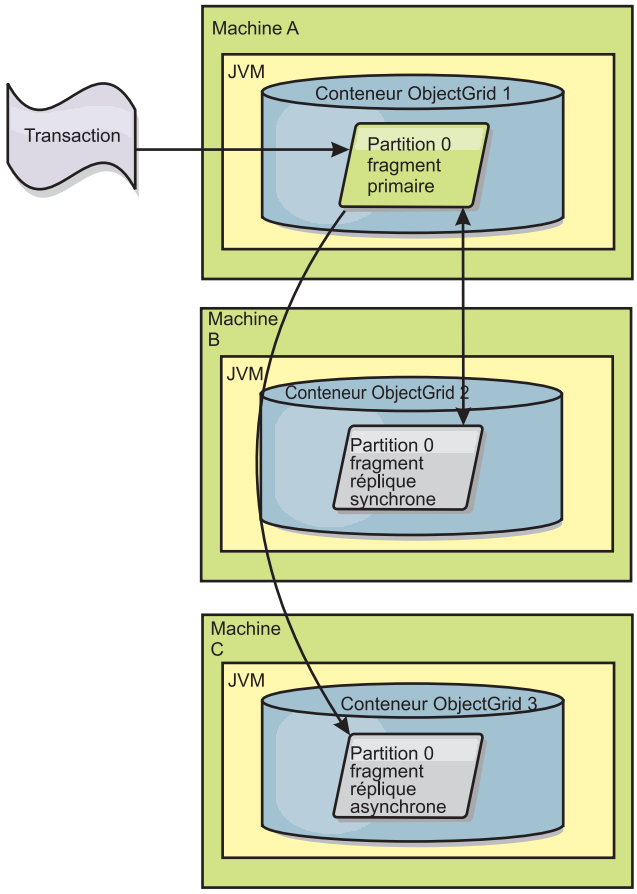


Figure 30. Chemin de la communication entre un fragment primaire et un fragment réplique

Fragments réplique synchrones minimum

Lorsqu'un fragment primaire se prépare à valider des données, il vérifie combien de fragments réplique synchrones ont élu de valider la transaction. Le fragment réplique élit de valider la transaction lorsqu'il traite normalement cette dernière. Si quelque chose ne va pas dans le fragment réplique synchrone, celui-ci décide de ne pas valider. Pour qu'un fragment primaire valide, le nombre de fragments réplique synchrones qui élisent de valider doit correspondre au paramètre minSyncReplica de la règle de déploiement. Si ce nombre est trop bas, le fragment primaire ne valide pas la transaction et une erreur se produit. Cette action garantit la disponibilité du nombre de fragments réplique synchrones avec les bonnes données. Les fragments réplique synchrones qui ont rencontré des erreurs se réenregistrent pour corriger leur état. Pour plus d'informations sur le réenregistrement, voir Récupération des fragments réplique.

Le fragment primaire lève une erreur ReplicationVotedToRollbackTransactionException si trop peu de fragments réplique synchrones ont élu de valider.

Réplication et loaders

En principe, le fragment primaire écrit de manière synchrone les modifications dans la base de données via le loader. Le loader et la base de données sont toujours synchrones. Lorsque le fragment primaire bascule vers un fragment réplique, la base de données et le loader risquent de ne plus l'être. Exemple :

- Le fragment primaire peut envoyer la transaction au fragment réplique, puis tombe en panne avant de valider vers la base de données.
- Le fragment primaire peut valider vers la base de données, puis tomber en panne avant d'envoyer au fragment réplique.

Dans les deux cas, le fragment réplique est décalé par rapport à la base de données : en avance ou en retard d'une transaction. Cette situation est inacceptable. eXtreme Scale utilise un protocole spécial et un contrat avec l'implémentation du loader afin de résoudre ce problème sans validation en deux phases. Le protocole fonctionne de la manière suivante :

Côté fragment primaire

- Envoi de la transaction avec les résultats de la transaction précédente.
- Ecriture dans la base de données et tentative de validation de la transaction.
- Si la base de données valide, validation dans eXtreme Scale. Si elle ne valide pas, annulation de la transaction.
- Enregistrement du résultat.

Côté fragment réplique

- Réception et mise en mémoire tampon d'une transaction.
- Pour tous les résultats, envoi avec la transaction, validation de toutes les transactions mises en mémoire tampon et suppression des transactions annulées.

Côté fragment réplique lors d'un basculement

- Pour toutes les transactions mises en mémoire tampon, fourniture des transactions au loader et tentative par ce dernier de valider les transactions.
- Le loader doit avoir été écrit de manière à rendre chaque transaction idempotente.
- Si la transaction est déjà dans la base de données, le Loader n'effectue aucune opération.
- Si la transaction n'est pas dans la base de données, le Loader applique la transaction.
- Une fois que toutes les transactions ont été traitées, le nouveau fragment primaire peut commencer à servir les demandes.

Ce protocole garantit que la base de données est au même niveau que l'état du nouveau fragment primaire.

Placement de fragment

Le service de catalogue est responsable de l'organisation des fragments. Chaque grille d'objets contient un certain nombre de partitions et chaque partition contient un fragment primaire et un ensemble facultatif de fragments réplique. Le service de catalogue alloue les fragments en les équilibrant pour qu'ils soient répartis uniformément dans les serveurs de conteneur disponibles. Les fragments de réplique et primaires d'une même partition ne sont jamais placés sur le même serveur de conteneur ou à la même adresse IP, à moins que la configuration soit en mode développement.

Si un nouveau serveur de conteneur démarre, eXtreme Scale extrait les fragments les serveurs de conteneur relativement surchargés et les place dans le nouveau serveur de conteneur vide. Ce mouvement des fragments permet une évolutivité horizontale.

Ajouts

Evolution signifie que lorsque des serveurs de conteneur supplémentaires sont ajoutés à une grille de données, eXtreme Scale tente de transférer les fragments primaires ou de réplique de l'ancien ensemble de serveurs de conteneur vers le nouvel ensemble. Ce mouvement étend la grille de données pour tirer parti du processeur, du réseau et de la mémoire des serveurs de conteneur nouvellement ajoutés. Le mouvement équilibre également la grille de données et tente de garantir que chaque JVM dans la grille de données héberge la même quantité de données. Comme la grille de données augmente, chaque serveur héberge un sous-ensemble réduit de la totalité de la grille. eXtreme Scale suppose que les données sont réparties de façon égale entre les différentes partitions. Cet agrandissement correspond à un ajout.

Suppressions

On parle de suppression lorsqu'en cas de défaillance d'une JVM, eXtreme Scale essaie de procéder à une réparation. Si la JVM concernée a une machine réplique, eXtreme Scale remplace la machine réplique victime de la défaillance par une nouvelle machine réplique créée sur une JVM n'ayant subi aucun dommage. Si la JVM ayant échoué a une machine primaire, eXtreme Scale recherche la meilleure machine réplique parmi les machines n'ayant subi aucun dommage et promeut celle-ci au rang de nouvelle machine primaire. eXtreme Scale remplace alors la machine réplique promue par une nouvelle machine réplique créée sur les serveurs restants. Afin de garantir l'évolutivité, eXtreme Scale conserve le même nombre de machines réplique des partitions en cas de défaillance des serveurs.

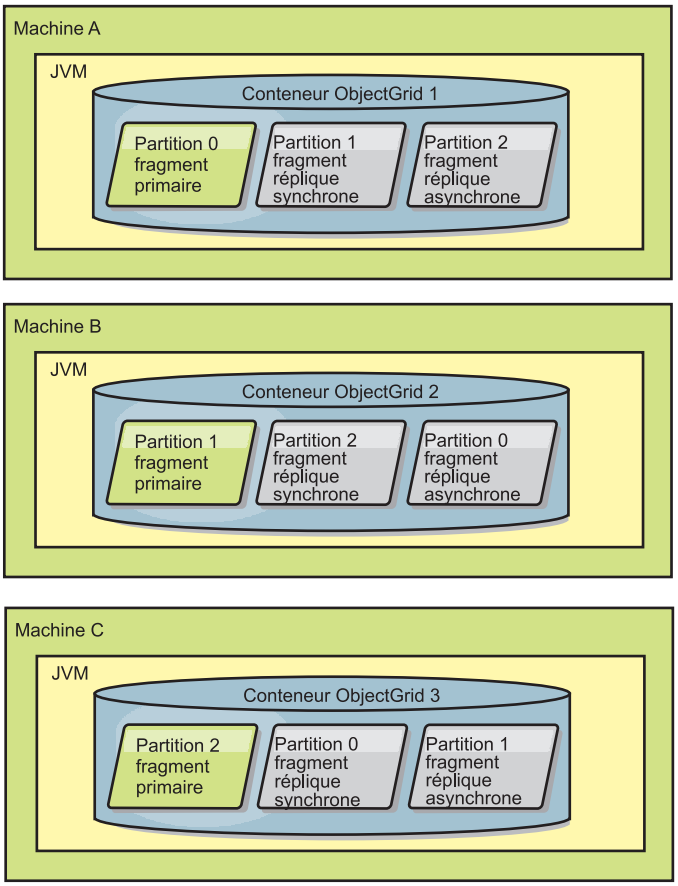


Figure 31. Placement d'un groupe de mappes ObjectGrid avec une stratégie de déploiement de 3 partitions avec la valeur minSyncReplicas1, la valeur maxSyncReplicas 1 et la valeur maxAsyncReplicas 1

Lecture à partir de répliques

Il est possible de configurer des groupes de mappes pour autoriser un client à lire les fragments répliqués au lieu d'être restreint aux fragments primaires.

Il peut souvent être intéressant de permettre aux fragments réplique de sortir de leur simple rôle de simples fragments primaires en puissance en cas de défaillances. Il est, par exemple, possible de configurer des groupes de mappes pour autoriser le routage des opérations de lecture vers les fragments réplique. Pour ce faire, il suffit de donner la valeur true à l'option replicaReadEnabled de MapSet. Par défaut, le paramètre a la valeur false.

Pour plus d'informations sur l'élément MapSet, voir dans le *Guide d'administration* la rubrique consacrée au fichier XML du descripteur de la règle de déploiement.

Activer la lecture des fragments réplique peut améliorer les performances en disséminant les demandes de lecture sur davantage de machines virtuelles Java. Si l'option n'est pas activée, toutes les demandes de lecture, telles les méthodes ObjectMap.get ou Query.getResultIterator, seront routées vers les fragments primaires. Lorsque replicaReadEnabled a la valeur true, certaines demandes get risquent de retourner des données périmées ; l'application utilisant cette option doit donc pouvoir tolérer cette éventualité. Mais il ne se produira pas d'échec en cache. Si les données ne sont pas dans le fragment réplique, la demande get est redirigée vers le fragment primaire et une nouvelle tentative est effectuée.

L'option replicaReadEnabled peut être utilisée dans les deux modes de réplication, synchrone et asynchrone.

Equilibrage de la charge entre les fragments réplique

Equilibrer la charge entre les fragments réplique s'utilise en général uniquement lorsque les clients mettent en cache des données qui changent en permanence ou lorsque les clients utilisent le verrouillage pessimiste.

eXtreme Scale, sauf configuration différente, envoie au serveur primaire toutes les demandes de lecture et d'écriture concernant un groupe de réplication donné. Ce serveur primaire doit servir toutes les demandes émanant des clients. Vous voudrez peut-être autoriser l'envoi des demandes de lecture aux répliques du fragment primaire. L'envoi des demandes de lecture aux fragments réplique permet à la charge de ces demandes d'être partagées par plusieurs machines virtuelles Java. Cela dit, il faut savoir que l'utilisation des fragments réplique pour les demandes de lecture peut donner des réponses incohérentes.

Equilibrer la charge entre les fragments réplique s'utilise en général uniquement lorsque les clients mettent en cache des données qui changent en permanence ou lorsque les clients utilisent le verrouillage pessimiste.

Si les données changent en permanence et qu'elles sont ensuite invalidées dans les caches locaux du client, le fragment primaire devrait constater en résultat un taux relativement élevé de demandes get provenant des clients. De même, en mode de verrouillage pessimiste, il n'existe aucun cache local, c'est pourquoi toutes les demandes sont envoyées au fragment primaire.

Si les données sont relativement statiques ou si le mode pessimiste n'est pas utilisé, l'envoi des demandes au fragment réplique n'a pas un énorme impact sur les performances. La fréquence des demandes get émanant des clients avec des caches pleins de données n'est pas élevée.

Lors du premier démarrage d'un client, son cache local est vide. Les demandes adressées au cache vide sont transmises au fragment primaire. Au fil du temps, le cache client obtient des données, ce qui fait tomber la charge des demandes. Si un grand nombre de clients démarrent simultanément, la charge peut être importante et la lecture par les fragments réplique peut être un choix approprié pour les performances.

Cycles de vie des fragments

Les fragments passent par différents états et événements pour prendre en charge la réplication. Le cycle de vie d'un fragment inclut la connexion, l'exécution, l'arrêt, le basculement et la gestion des erreurs. Les fragments peuvent passer de l'état de fragment de réplique à celui de fragment primaire afin de gérer les modifications d'état du serveur.

Evénements de cycle de vie

Lorsque les fragments primaires et répliques sont placés et démarrés, ils passent par une série d'événements qui leur permettent d'être en ligne et en mode écoute.

Fragment primaire

Le service de catalogue place un fragment primaire pour une partition. Le service de catalogue s'attache également à équilibrer les emplacements de fragment primaire et à lancer le basculement des fragments primaires.

Lorsqu'un fragment devient un fragment primaire, il reçoit du service de catalogue une liste de fragments réplique. Le nouveau fragment primaire crée un groupe de fragments réplique et enregistre tous les fragments réplique.

Lorsque le fragment primaire est prêt, un message signalant qu'il est prêt s'affiche dans le fichier `SystemOut.log` pour le conteneur d'exécution. Pour ouvrir le message ou le message CWOBJ1511I, répertorie le nom de mappe, le nom du groupe de mappes et le numéro de partition du fragment primaire démarré.

```
CWOBJ1511I: mapName:mapSetName:partitionNumber (primary) is open for business.
```

Pour plus d'informations sur le positionnement de fragments par le service de catalogue, reportez-vous à la rubrique «Placement de fragment», à la page 112.

Fragment réplique

Les fragments réplique sont principalement contrôlés par le fragment primaire à moins que le fragment réplique détecte un problème. Pendant un cycle de vie normal, le fragment primaire place, enregistre et annule l'enregistrement d'un fragment réplique.

Lorsque le fragment primaire initialise un fragment réplique, un message affiche le journal décrivant où s'exécute le fragment réplique pour indiquer que ce dernier est disponible. Pour ouvrir le message ou le message CWOBJ1511I, répertorie le nom de mappe, le nom du groupe de mappes et le numéro de partition du fragment réplique. Le message suivant s'affiche :

```
CWOBJ1511I: mapName:mapSetName:partitionNumber (synchronous replica) is open for business.
```

ou

```
CWOBJ1511I: mapName:mapSetName:partitionNumber (asynchronous replica) is open for business.
```

Fragment réplique asynchrone : Un fragment réplique scrute les données dans son fragment primaire. Le fragment réplique ajustera automatiquement l'intervalle auquel il scrute ces données s'il n'en reçoit pas du fragment primaire, ce qui est l'indice qu'il est à niveau avec ce dernier. Il ajustera également cet intervalle s'il reçoit une erreur pouvant indiquer que le fragment primaire est en panne ou qu'il y a un problème réseau.

Lorsqu'un fragment réplique passe en mode homologue, il imprime le message suivant dans son fichier `SystemOut.log`. Ce message peut apparaître plusieurs fois par message CWOBJ1511. Il s'imprimera à nouveau si le fragment réplique se connecte à un autre fragment primaire ou en cas d'ajout de mappes modèles.

```
CWOBJ1543I: Le fragment réplique asynchrone objectGridName:mapsetName:partitionNumber a démarré ou a
continué à se répliquer à partir du fragment primaire.
Réplication en cours pour les mappes : [nomMappe]
```

Fragment réplique synchrone : Lorsque le fragment réplique démarre pour la première fois, il n'est pas encore en mode homologue. Lorsqu'un fragment réplique est en mode homologue, il reçoit les données du fragment primaire au fur et à mesure de leur arrivée dans ce dernier. Avant de passer en mode homologue, le fragment réplique a besoin d'une copie de toutes les données existant dans le fragment primaire.

Le fragment réplique synchrone copie les données depuis le fragment primaire comme cela se passe vers un fragment réplique asynchrone, en scrutant le fragment primaire. Lorsqu'il copie les données existantes à partir du fragment

primaire, il passe en mode homologue et commence à recevoir les données en même temps que celles-ci parviennent au fragment primaire.

Lorsqu'un fragment réplique atteint le mode homologue, il imprime un message dans son fichier SystemOut.log. La valeur de temps se réfère à la durée qu'il a fallu au fragment réplique pour obtenir toutes ses données initiales du fragment primaire. La valeur de temps s'affiche comme étant zéro ou étant très faible si le fragment primaire ne comporte aucune donnée existante à répliquer. Ce message peut apparaître plusieurs fois par message CWOBJ1511. Il s'imprimera à nouveau si le fragment réplique se connecte à un autre fragment primaire ou en cas d'ajout de mappes modèles.

```
CWOBJ1526I: Replica objectGridName:mapsetName:partitionNumber:mapName entering peer
mode after X seconds.
```

Lorsque le fragment réplique synchrone est en mode homologue, le fragment primaire doit répliquer les transactions vers la totalité des fragments réplique synchrones qui sont en mode homologue. Le fragment réplique synchrone demeure au même niveau que les données du fragment primaire. Si, dans la règle de déploiement, il est défini un nombre minimum de fragments réplique ou si minSync est défini dans cette règle, ce nombre de fragments réplique synchrones doit voter pour valider avant que la validation de la transaction puisse s'effectuer dans le fragment primaire.

Evénements de reprise

La finalité de la réplication est de permettre la reprise après des échecs et des événements d'erreur. Si un fragment primaire tombe en panne, un autre fragment réplique prend le relais. Si des erreurs sont détectées sur les fragments réplique, le fragment réplique tente de se rétablir. Le service de catalogue contrôle le positionnement et les transactions sur les nouveaux fragments primaires ou sur les nouveaux fragments réplique.

Les fragments réplique deviennent un fragment primaire

Un fragment réplique devient un fragment primaire pour deux raisons. Le fragment primaire s'est arrêté ou a échoué ou une décision d'équilibre a été prise pour déplacer le fragment précédent vers un nouvel emplacement.

Le service de catalogue sélectionne un nouveau fragment primaire des fragments réplique synchrones existants. Si un déplacement de fragment primaire est nécessaire et qu'il n'existe aucun fragment réplique, un fragment réplique temporaire sera positionné pour effectuer la transition. Le nouveau fragment primaire enregistre tous les fragments réplique existants et accepte les transactions en tant que nouveau fragment primaire. Si les fragments réplique existants comportent le niveau de données adéquat, les données en cours sont conservées lorsque le fragment réplique s'enregistre auprès du nouveau fragment primaire. Les fragments réplique asynchrones scruteront le nouveau fragment primaire.

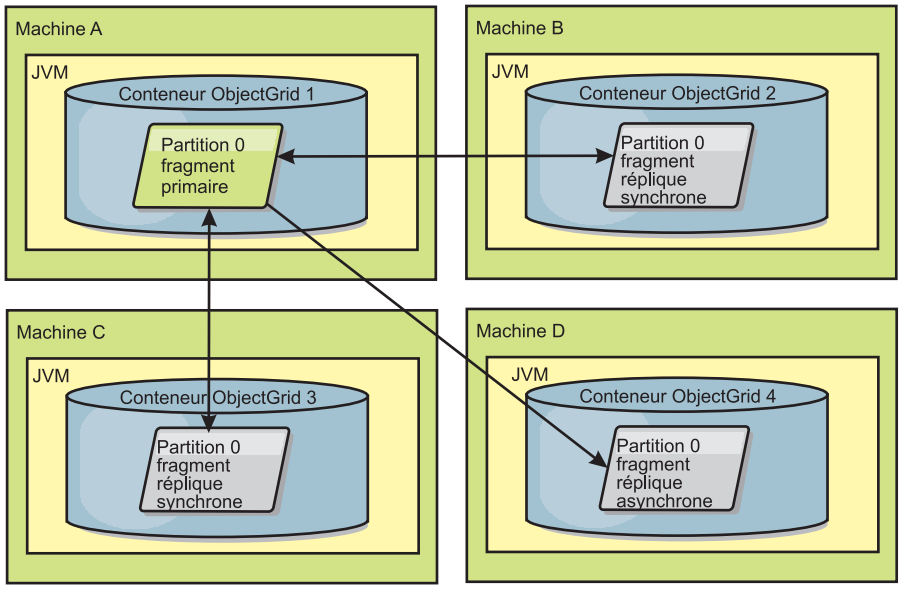


Figure 32. Exemple de positionnement d'une mappe ObjectGrid définie pour la partition partition0. La règle de déploiement comprend une valeur minSyncReplicas de 1, une valeur maxSyncReplicas de 2 et une valeur maxAsyncReplicas de 1.

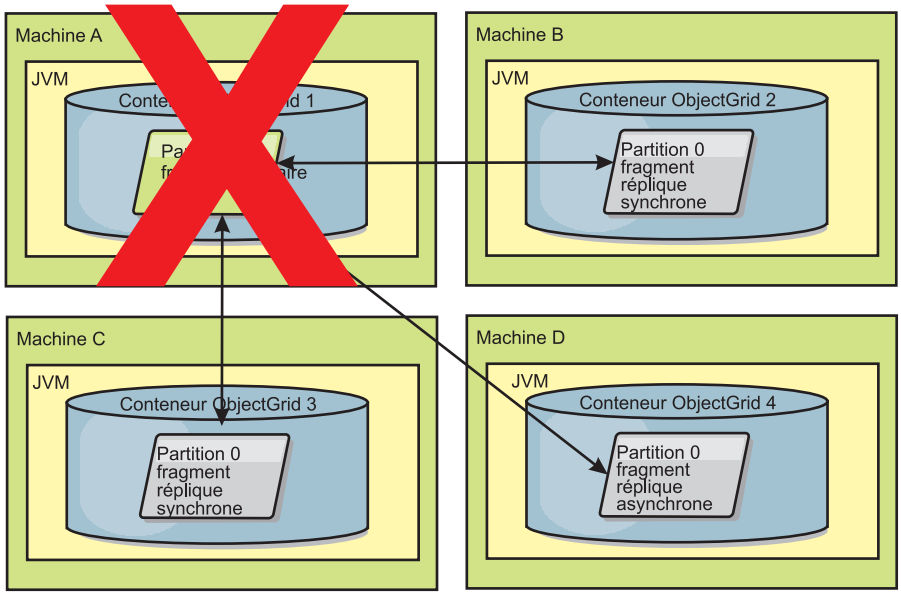


Figure 33. Le conteneur pour le fragment primaire échoue.

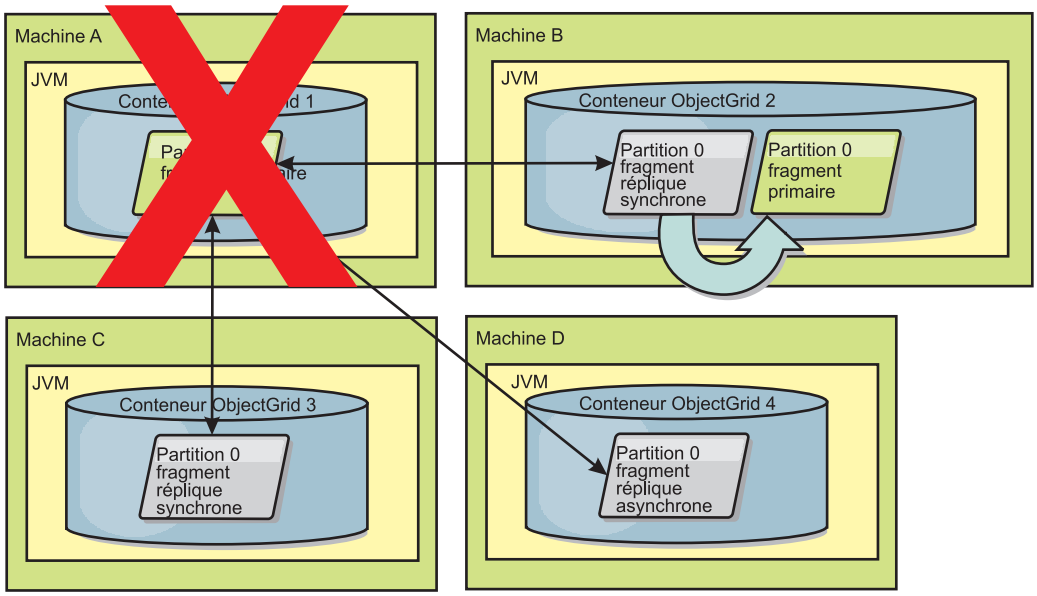


Figure 34. Le fragment de réplique synchrone sur le conteneur ObjectGrid 2 devient un fragment primaire.

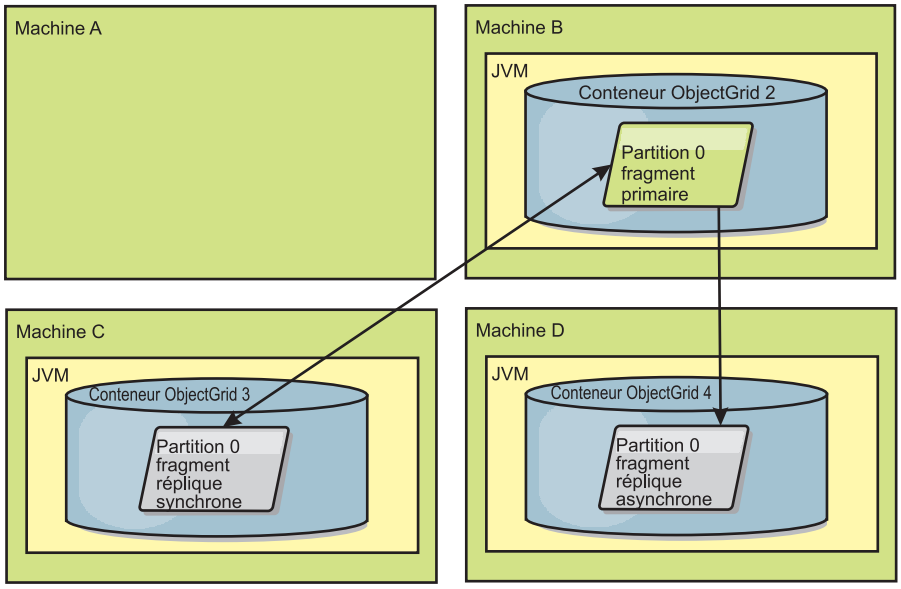


Figure 35. La machine B contient le fragment primaire. En fonction de la définition du mode de réparation automatique et de la disponibilité des conteneurs, un nouveau fragment réplique synchrone peut ou peut ne pas être placé sur une machine.

Reprise des fragments réplique

Un fragment réplique synchrone est contrôlé par le fragment primaire. Toutefois, si un fragment réplique détecte un problème, il peut déclencher un événement de réenregistrement pour corriger l'état des données. La réplique supprime les données en cours et obtient une nouvelle copie du fragment primaire.

Lorsqu'un fragment réplique lance un événement de réenregistrement, il imprime un message de journal.

```
CWOBJ1524I: Replica listener
objectGridName:mapSetName:partition must re-register with the primary.
Reason: Exception listed
```

Si une transaction cause une erreur sur un fragment réplique pendant le traitement, le fragment réplique est à l'état inconnu. La transaction a correctement eu lieu sur le fragment primaire mais un incident s'est produit sur le fragment réplique. Pour remédier à cette situation, le fragment réplique lance un événement de réenregistrement. Avec une nouvelle copie des données du fragment primaire, le fragment réplique peut se poursuivre. Si le même problème se reproduit, le fragment réplique n'effectue pas le réenregistrement en continu. Voir «Evénements d'arrêt anormal» pour plus de détails.

Evénements d'arrêt anormal

Un fragment réplique peut arrêter de répliquer les données s'il rencontre des situations d'erreur dans lesquelles il n'a aucun moyen de reprendre son activité.

Trop de tentatives d'enregistrement

Si un fragment réplique déclenche un réenregistrement plusieurs fois sans parvenir à valider les données, il s'arrête. L'arrêt empêche une réplique d'entrer dans une boucle de réenregistrement infinie. Par défaut, un fragment réplique tente de réenregistrer trois fois dans une ligne avant de s'interrompre.

Si une réplique réenregistre trop de fois, elle imprime le message suivant dans le journal.

```
CWOBJ1537E: objectGridName:mapSetName:partition exceeded the maximum number
of times to reregister (timesAllowed) without successful transactions.
```

Si le fragment réplique est incapable d'être restauré en se réenregistrant, un problème pervasif peut exister avec les transactions relatives à ce fragment. Il pourrait en résulter un manque de ressources sur le chemin d'accès aux classes si une erreur survient lors de l'inflation des clés ou des valeurs de la transaction.

Echec lors de l'activation du mode homologue

Si un fragment réplique tente de passer en mode homologue et rencontre une erreur lors du traitement des données en bloc existantes à partir du fragment primaire (données de point de contrôle), le fragment réplique s'arrête. L'arrêt empêche le fragment réplique démarrer avec des données initiales erronées. Etant donné qu'il reçoit les mêmes données du fragment primaire s'il se iréenregistre, le fragment réplique ne fait aucune nouvelle tentative.

Si une réplique ne parvient pas à entrer en mode homologue, elle imprime le message suivant dans le journal :

```
CWOBJ1527W Replica objectGridName:mapSetName:partition:mapName failed to enter peer mode after numSeconds seconds.
```

Un message supplémentaire s'affiche dans le journal et explique pourquoi le fragment réplique n'a pu passer en mode homologue.

Reprise après réenregistrement ou échec du mode homologue

Si une réplique ne peut pas se réenregistrer ou entrer en mode homologue, elle est à l'état inactif jusqu'à ce qu'un nouvel événement de positionnement ait lieu. Un nouvel événement de positionnement peut être le démarrage ou l'arrêt d'un nouveau serveur. Vous pouvez également démarrer un événement de

positionnement à l'aide de la méthode triggerPlacement sur le bean géré PlacementServiceMBean.

Groupes de mappes pour la réplication

La réplication est activée par l'association de BackingMaps à un groupe de mappes.

Un groupe de mappes est un ensemble de mappes catégorisées par partition-key. Cette partition-key est dérivée de la clé des mappes individuelles par une opération consistant à prendre son hachage modulo le nombre de partitions. Si un groupe de mappes au sein du groupe de mappes a une partition-key X, ces mappes seront stockées dans une partition X correspondante dans la grille de données. Si un autre groupe possède une partition-key Y, toutes les mappes seront stockées dans la partition Y, et ainsi de suite. En outre, les données contenues dans les mappes sont répliquées en fonction des règles définies dans le groupe de mappes qui est utilisé uniquement pour les topologies réparties eXtreme Scale (inutile pour les instances locales).

Voir «Partitionnement», à la page 79 pour plus de détails.

Les groupes de mappes sont affectés du nombre de partitions qu'ils auront et possèdent une stratégie de réplication. La configuration de la réplication de groupes de mappes identifie simplement le nombre de fragments de réplique synchrone et asynchrone qu'un groupe de mappes doit avoir en plus du fragment primaire. Par exemple, s'il doit exister 1 réplique synchrone et 1 réplique asynchrone, tous les BackingMaps affectés au groupe de mappes auront chacun un fragment de réplique distribué automatiquement dans le groupe des conteneurs disponibles pour eXtreme Scale. La configuration de la réplication peut également permettre aux clients de lire les données depuis les serveurs répliqués de manière synchrone. Cela peut étaler la charge des demandes de lecture sur d'autres serveurs de la grille eXtreme Scale. La réplication a un impact sur le modèle de modèle de programmation lorsqu'on précharge les mappes de sauvegarde.

Traitement des transactions

WebSphere eXtreme Scale utilise les transactions comme mécanisme d'interaction avec les données.

Pour interagir avec les données, l'unité d'exécution de votre application requiert sa propre session. Lorsque l'application souhaite utiliser ObjectGrid sur une unité d'exécution, appelez l'une des méthodes ObjectGrid.getSession pour obtenir une unité d'exécution. Avec la session, l'application peut travailler avec les données stockées dans les mappes ObjectGrid.

Lorsqu'une application utilise un objet Session, la session doit être dans le contexte d'une transaction. Une transaction commence et se valide ou commence et s'annule en utilisant les méthodes begin, commit et rollback sur l'objet Session. Les applications fonctionnent également en mode d'auto-validation : la session commence et valide automatiquement une transaction chaque fois qu'une opération est effectuée sur la mappe. Le mode d'auto-validation ne permet pas de regrouper des opérations multiples en une seule transaction. Il s'agit donc de l'option la plus lente si vous créez un lot d'opérations multiples dans une seule transaction. Cependant, pour les transactions contenant une seule opération, l'auto-validation est l'option la plus rapide.

Transactions

Les transactions disposent de nombreux avantages pour le stockage et la manipulation de données. Vous pouvez utiliser des transactions pour protéger la grille de données contre les changements simultanés, appliquer plusieurs changements comme unité simultanée, répliquer des données et implémenter un cycle de vie aux verrous appliqués aux changements.

Quand une transaction démarre, WebSphere eXtreme Scale alloue une mappe spéciale des différences pour contenir les changements en cours ou des copies des paires de valeur-clé que la transaction utilise. En règle générale, quand un accès à une paire valeur-clé se produit, la valeur est copiée avant que l'application ne reçoive la valeur. La mappe des différences contrôle tous les changements pour les opérations telles qu'insérer, mettre à jour, obtenir, supprimer, etc. Les clés ne sont pas copiées, car elles sont considérées comme non modifiables. Si un objet ObjectTransformer est spécifié, il est utilisé pour copier la valeur. Si la transaction utilise un verrouillage optimiste, les images précédentes des valeurs sont également suivies afin d'être comparées quand la transaction est validée.

Si une transaction est annulée, les informations de la mappe des différences sont supprimées et les verrous sur les entrées sont retirés. Quand une transaction est validée, les changements sont appliqués à la mappe et les verrous retirés. Si un verrouillage optimiste est utilisé, eXtreme Scale compare les versions des images précédentes des valeurs avec les valeurs qui se trouvent dans la mappe. Ces valeurs doivent correspondre pour que la transaction soit validée. Cette comparaison autorise un plan de verrouillage à version multiple, mais au prix de deux copies créées quand la transaction accède à l'entrée. Toute les valeurs sont à nouveau copiées et la nouvelle copie est stockée dans la mappe. WebSphere eXtreme Scale crée cette copie pour se protéger contre le fait que l'application change sa référence à la valeur après validation.

Il est possible de ne pas utiliser plusieurs copies des informations. L'application peut enregistrer une copie en utilisant un verrouillage pessimiste au lieu d'un verrouillage optimiste au prix d'une limitation des accès simultanés. La copie de la valeur au moment de la validation peut également être évitée si l'application accepte de ne pas changer la valeur après une validation.

Avantages des transactions

Utilisez les transactions pour les raisons suivantes :

Par le biais des transactions, vous pouvez :

- Annuler les modifications si une exception se produit ou si la logique application requiert l'annulation des changements d'état.
- Pour appliquer plusieurs changements en tant qu'unité atomique au moment de la validation
- Verrouiller et déverrouiller les données afin d'appliquer des changements multiples en tant qu'unité atomique au moment de la validation.
- Protéger une unité d'exécution de modifications simultanées.
- Implémenter un cycle de vie pour les verrous appliqués aux modifications.
- Produire une unité atomique de réplication.

Taille des transactions

Les transactions volumineuses sont plus efficaces, en particulier pour la réplication. Cependant, les grandes transactions peuvent avoir un impact sur les accès simultanés car les verrous sur entrées sont maintenus plus longtemps. Si vous utilisez de grandes instructions, vous pouvez accroître les performances de réplication. Cette augmentation des performances est important lors du pré-chargement d'une mappe. Essayez différentes tailles de lots pour déterminer ce qui fonctionne le mieux pour votre scénario.

Les grandes transactions aident également avec les programmes de chargement. Si un chargeur utilisé peut effectuer du traitement par lots SQL, alors des gains de performances significatifs peuvent être réalisés sur la transaction, ainsi que des réductions de charges significatives du côté de la base de données. Ces gains de performances dépendent de l'implémentation du chargeur.

Mode de validation automatique

Si aucune transaction n'a démarré activement, quand une application interagit avec un objet ObjectMap, une opération automatique de démarrage et de validation est effectuée pour l'application. Cette opération fonctionne, mais elle empêche l'annulation et le verrouillage de fonctionner efficacement. La vitesse de réplication synchrone est affectée à cause de la très petite taille des transactions. Si vous utilisez une application de gestion des entités, n'utilisez pas le mode de validation automatique car les objets recherchés avec la méthode EntityManager.find deviennent immédiatement non gérés au retour de la méthode et deviennent inutilisables.

Coordinateurs de transactions externes

En règle générale, les transactions commencent avec la méthode session.begin et se termine par la méthode session.commit. Cependant, quand eXtreme Scale est imbriqué, les transactions peuvent être démarrées et terminées par un coordinateur de transactions externes. Si vous en utilisez un, vous n'avez pas besoin d'appeler la méthode session.begin et de terminer avec la méthode session.commit. Si vous utilisez WebSphere Application Server, vous pouvez utiliser le plug-in WebSphereTransactionCallback.

Attribut CopyMode

Vous pouvez ajuster le nombre de copies en définissant l'attribut CopyMode des objets BackingMap et ObjectMap dans le fichier descripteur XML d'ObjectGrid.

Vous pouvez ajuster le nombre de copies en définissant l'attribut CopyMode des objets BackingMap et ObjectMap. Cet attribut a les valeurs suivantes :

- COPY_ON_READ_AND_COMMIT
- COPY_ON_READ
- NO_COPY
- COPY_ON_WRITE
- COPY_TO_BYTES
- COPY_TO_BYTES_RAW

COPY_ON_READ_AND_COMMIT est la valeur par défaut. La valeur COPY_ON_READ copie les données initiales récupérées, mais ne copie pas au moment de la validation. Ce mode est sûr si l'application ne modifie pas une valeur après la validation d'une transaction. La valeur NO_COPY ne copie pas de

données, ce qui n'est sûr que pour les données en lecture seule. Si les données ne changent jamais, vous n'avez alors pas besoin de les copier pour des raisons d'isolement.

Lorsque vous utilisez la valeur d'attribut NO_COPY, faites attention aux mappes pouvant être mises à jour. WebSphere eXtreme Scale utilise la copie en premier appui pour autoriser l'annulation de la transaction. L'application a changé uniquement la copie, et par conséquent, eXtreme Scale supprime la copie. Si la valeur d'attribut NO_COPY est utilisée et que l'application modifie la valeur validée, il est impossible de procéder à une annulation. La modification de la valeur validée génère des problèmes d'index, de réplication, etc. car les index et les fragments réplique se mettent à jour à la validation de la transaction. Si vous modifiez des données validées puis annulez la transaction, qui n'est pas vraiment annulée, alors les index ne sont pas mis à jour et les réplications ne se produisent pas. D'autres unités d'exécution peuvent voir les changements non validés immédiatement, même s'ils sont verrouillés. Utilisez la valeur d'attribut NO_COPY pour les mappes en lecture seule pour les applications qui effectuent la copie appropriée avant la modification de la valeur. Si vous utilisez la valeur d'attribut NO_COPY et que vous contactez le support technique IBM pour un problème d'intégrité de données, il vous est demandé de reproduire le problème avec le mode de copie défini sur COPY_ON_READ_AND_COMMIT.

La valeur COPY_TO_BYTES stocke les valeurs dans la mappe dans un formulaire sérialisé. Au moment de la lecture, eXtreme Scale gonfle la valeur depuis un formulaire sérialisé et la stocke dans un autre au moment de la validation. Avec cette méthode, une copie est créée au moment de la lecture et au moment de la validation.

Le mode de copie par défaut pour une mappe est configurable sur l'objet BackingMap. Vous pouvez également changer le mode de copie sur les mappes avant de commencer une transaction en utilisant la méthode ObjectMap.setCopyMode.

Un exemple de fragment de mappe de sauvegarde provenant d'un fichier `objectgrid.xml` qui montre comment définir le mode de copie pour une mappe de sauvegarde donnée suit. Cet exemple part du principe que vous utilisez `cc` comme espace de noms `objectgrid/config`.

```
<cc:backingMap name="RuntimeLifespan" copyMode="NO_COPY"/>
```

Gestionnaire de verrous

Lorsque vous configurez une stratégie de verrouillage, un gestionnaire de verrous est créé pour la mappe de sauvegarde pour conserver la cohérence des entrées de cache.

Configuration du gestionnaire de verrou

Lors de l'utilisation d'une stratégie de verrouillage PESSIMISTIC ou OPTIMISTIC, un gestionnaire de verrou est créé pour la mappe de sauvegarde. Le gestionnaire de verrou utilise une mappe de hachage pour rechercher les entrées verrouillées par une ou plusieurs transactions. S'il existe plusieurs entrées de mappe dans la mappe hash, plus les compartiments de verrouillage sont nombreux, plus les performances sont meilleures. Le risque de collision de synchronisation Java diminue lorsque le nombre de compartiments augmente. Plus les compartiments de verrouillage sont nombreux, plus les accès simultanés le sont également. Les exemples précédents montrent comment une application peut définir le nombre de compartiments de verrouillage à utiliser pour une instance BackingMap donnée.

Pour éviter une exception java.lang.IllegalStateException, la méthode setNumberOfLockBuckets doit être appelée avant d'appeler les méthodes initialize ou getSession sur une instance ObjectGrid. Le paramètre de la méthode setNumberOfLockBuckets est un entier primitif Java spécifiant le nombre de compartiments de verrouillage à utiliser. L'utilisation d'un nombre primitif peut permettre une distribution uniforme des entrées de mappe entre les compartiments de verrouillage. Pour optimiser les performances, commencez par définir le nombre de compartiments de verrouillage sur environ 10 % du nombre attendu d'entrées BackingMap.

Stratégies de verrouillage

Les stratégies de verrouillage sont de type pessimiste, optimiste ou aucune. Pour choisir une stratégie de verrouillage, vous devez prendre en compte les aspects tels que le pourcentage des types d'opérations, l'utilisation ou non d'un chargeur, etc.

Les verrous sont liés aux transactions. Vous pouvez spécifier les paramètres de verrouillage suivants :

- **Aucun verrouillage** : l'exécution sans verrouillage est la plus rapide. Si vous utilisez des données en lecture seule, vous n'avez peut-être pas besoin de verrouillage.
- **Verrouillage pessimiste** : place des verrous sur les entrées, puis les maintient jusqu'au moment de la validation. Cette stratégie offre une bonne cohérence au prix de la capacité de traitement.
- **Verrouillage optimiste** : prend une image précédente de chaque enregistrement sur lequel la transaction appuie et compare l'image avec les valeurs d'entrées en cours quand la transaction est validée. Si les valeurs d'entrées changent, la transaction est annulée. Aucun verrou n'est maintenu jusqu'au moment de la validation. Cette stratégie de verrouillage offre un meilleur accès simultané que les stratégies pessimistes, au risque que la transaction soit annulée et au prix de la mémoire nécessaire pour une copie supplémentaire de l'entrée.

Définissez la stratégie de verrouillage sur la mappe de sauvegarde. Il n'est pas possible de changer de stratégie pour chaque transaction. Un fragment de code XML suit, montrant comment définir le mode de verrouillage sur une mappe à l'aide du fichier XML, en partant du principe que `cc` est le nom d'espace pour `objectgrid/config` :

```
<cc:backingMap name="RuntimeLifespan" lockStrategy="PESSIMISTIC" />
```

Verrouillage pessimiste

La stratégie de verrouillage pessimiste est à utiliser pour les opérations de mappe en lecture et en écriture lorsqu'aucune autre stratégie de verrouillage n'est possible. Lorsqu'une mappe ObjectGrid est configurée en mode de stratégie de verrouillage pessimiste, un verrou de transaction pessimiste est obtenu pour une entrée de mappe à la première transmission de l'entrée à la mappe de sauvegarde. Le verrou pessimiste est maintenu jusqu'à la fin de la transaction. La stratégie de verrouillage pessimiste est généralement utilisée dans les cas suivants :

- Lorsque la mappe de sauvegarde est configurée avec ou sans chargeur et que les informations sur les versions ne sont pas disponibles.
- Lorsque la mappe de sauvegarde est utilisée directement par une application qui nécessite l'assistance de eXtreme Scale pour le contrôle des accès simultanés.
- Lorsque les informations sur les versions sont disponibles mais que les transactions de mise à jour entrent régulièrement en conflit avec les entrées de sauvegarde, ce qui entraîne des échecs de mise à jour optimiste.

Etant donné que la stratégie de verrouillage pessimiste a un impact majeur sur les performances et l'évolutivité, elle doit uniquement être utilisée pour les mappes en lecture et écriture lorsque les autres stratégies de verrouillage ne sont pas adaptées. Par exemple, ces situations peuvent être : échecs réguliers des mises à jour optimistes ou reprise après échec optimiste difficile à gérer pour une application.

Verrouillage optimiste

La stratégie de verrouillage optimiste présuppose qu'il ne peut se faire que deux transactions tentent d'actualiser la même entrée de mappe au même moment. A partir de ce principe, il n'est pas nécessaire de maintenir le mode de verrouillage pendant tout le cycle de vie de la transaction car il est peu probable que plusieurs transactions puissent actualiser la même entrée de mappe exactement au même moment. La stratégie de verrouillage optimiste est généralement utilisée dans les situations suivantes :

- Lorsqu'une mappe de sauvegarde est configurée avec ou sans chargeur et que les informations sur les versions sont disponibles.
- Lorsqu'une mappe de sauvegarde contient essentiellement des transactions qui exécutent des opérations de lecture. Les opérations insert, update ou remove sur les entrées de mappe ne sont pas fréquentes pour une mappe de sauvegarde.
- Lorsqu'une mappe de sauvegarde est insérée, mise à jour ou supprimée plus fréquemment qu'elle n'est lue mais que les transactions entrent rarement en conflit sur la même entrée de mappe.

A l'instar de la stratégie de verrouillage pessimiste, les méthodes dans l'interface ObjectMap déterminent comment eXtreme Scale tente automatiquement d'obtenir un mode de verrouillage pour l'entrée de mappe à laquelle l'accès est octroyé. Toutefois, les stratégies pessimiste et optimiste diffèrent sur les points suivants :

- Comme la stratégie de verrouillage pessimiste, un mode de verrou S est obtenu par les méthodes get et getAll lorsque la méthode est appelée. En revanche, avec le verrouillage optimiste, le mode de verrou S n'est maintenu que lorsque la transaction s'achève. Le mode de verrou S est libéré avant que la méthode soit renvoyée à l'application. L'objectif d'un mode de verrou consiste en ce que eXtreme Scale vérifie que seules les données validées provenant d'autres transactions soient visibles pour la transaction en cours. Une fois que eXtreme Scale a confirmé la validation des données, le mode de verrou S est libéré. Au moment de la validation, une vérification optimiste des versions est effectuée pour faire en sorte qu'aucune autre transaction n'a modifié l'entrée de mappe après la libération du mode de verrou S par la transaction en cours. Si une entrée n'est pas extraite d'une mappe avant d'être mise à jour, invalidée ou supprimée, l'exécution eXtreme Scale extrait implicitement l'entrée de la mappe. Cette opération get implicite est effectuée pour obtenir la valeur actuelle au moment de la demande de modification de l'entrée.
- Contrairement à la stratégie de verrouillage pessimiste, les méthodes getForUpdate et getAllForUpdate sont traitées exactement comme les méthodes get et getAll de la stratégie de verrouillage optimiste. Un mode de verrou S est obtenu au début de la méthode et est libéré avant d'être renvoyé à l'application.

Toutes les autres méthodes ObjectMap sont traitées exactement comme elles le sont dans la stratégie de verrouillage pessimiste. Lorsque la méthode commit est appelée, un mode de verrou X est obtenu pour toutes les entrées de mappe insérées, mises à jour, supprimées, corrigées ou invalidées et le mode de verrou X est maintenu jusqu'à ce que la transaction effectue le processus de validation.

La stratégie de verrouillage optimiste considère qu'aucune transaction simultanée ne tente de mettre à jour la même entrée de mappe. A partir de ce principe, le mode de verrouillage ne doit pas être maintenu pour le cycle de vie de la transaction car il est peu probable qu'une transaction puisse mettre à jour simultanément la même entrée de mappe. Toutefois, étant donné qu'aucun mode de verrouillage n'est maintenu, une autre transaction simultanée peut potentiellement mettre à jour l'entrée de mappe après la libération du mode de verrou S par la transaction en cours.

Pour gérer cette possibilité, eXtreme Scale obtient un verrou X au moment de la validation et effectue une vérification optimiste des versions pour faire en sorte qu'aucune autre transaction n'a modifié l'entrée de mappe après la lecture de l'entrée de mappe de la mappe de sauvegarde par la transaction en cours. Si une autre transaction modifie l'entrée de mappe, la vérification de versions échoue et une exception OptimisticCollisionException se produit. Cette exception force l'annulation de la transaction en cours et l'application doit réessayer la transaction entière. La stratégie de verrouillage optimiste s'avère très utile lors qu'une mappe est principalement lue et que les mises à jour de la même entrée de mappe sont peu probables.

Aucun verrouillage

Lorsqu'une mappe de sauvegarde est configurée pour n'utiliser aucune stratégie de verrouillage, la valeur retournée est aucun verrou de transaction pour une entrée de mappe.

La non-utilisation d'une stratégie de verrouillage est utile lorsqu'une application est un gestionnaire de persistance tel qu'un conteneur EJB (Enterprise JavaBeans) ou qu'une application utilise Hibernate pour obtenir les données persistantes. Dans ce scénario, la mappe de sauvegarde est configurée sans chargeur et le gestionnaire de persistance utilise la mappe de sauvegarde comme cache de données. Dans ce scénario, le gestionnaire de persistance fournit le contrôle des accès simultanés entre les transactions qui accèdent aux mêmes entrées de mappe.

WebSphere eXtreme Scale n'a pas besoin d'obtenir de verrous de transaction à des fins de contrôle des accès simultanés. Cette situation considère que le gestionnaire de persistance ne libère pas ses verrous de transaction avant de mettre à jour la mappe ObjectGrid avec les modifications validées. Si le gestionnaire de persistance libère ses verrous, une stratégie de verrouillage pessimiste ou optimiste doit être utilisée. Par exemple, supposons que le gestionnaire de persistance d'un conteneur d'EJB met à jour une mappe ObjectGrid avec les données validées dans la transaction EJB gérée par conteneur. Si la mise à jour de la mappe ObjectGrid a lieu avant la libération des verrous du gestionnaire de persistance, vous pouvez utiliser la stratégie sans verrou. Si la mise à jour de la mappe ObjectGrid a lieu après la libération des verrous du gestionnaire de persistance, vous devez utiliser la stratégie optimiste ou pessimiste.

La stratégie sans verrou peut être utilisée également lorsque l'application utilise directement une mappe de sauvegarde et qu'un chargeur est configuré pour la mappe. Dans ce scénario, le chargeur utilise le fonction de contrôle des accès simultanés fournies par un système de gestion de base de données relationnelle (SGBDR) à l'aide de la connectivité JDBC (Java Database Connectivity) ou le plug-in Hibernate pour accéder aux données dans une base de données relationnelle. L'implémentation du chargeur peut utiliser une approche optimiste ou pessimiste. Un chargeur qui utilise une approche optimiste de verrouillage ou de gestion des versions contribue à optimiser les performances et l'accès simultané.

Pour plus d'informations sur l'implémentation d'une approche de verrouillage optimiste, reportez-vous à la section OptimisticCallback dans les considérations relatives aux chargeurs dans le *Guide d'administration*. Si vous utilisez un chargeur qui utilise le verrouillage pessimiste d'un programme d'arrière plan, il est conseillé d'utiliser le paramètre forUpdate transmis à la méthode get de l'interface Loader. Définissez ce paramètre sur true si la méthode getForUpdate de l'interface ObjectMap a été utilisée par l'application pour obtenir les données. Le chargeur peut se servir de ce paramètre pour déterminer s'il convient de demander un verrou pouvant être mis à niveau sur la ligne en cours de lecture. Par exemple, DB2 obtient un verrou pouvant être mis à niveau lorsqu'une instruction SQL select contient une clause `FOR UPDATE`. Cette approche offre la même protection contre les interblocages que celle décrite à la rubrique «Verrouillage pessimiste», à la page 125.

Pour plus d'informations, reportez-vous à la rubrique relative à la gestion des verrous dans le *Guide de programmation* ou au verrouillage des entrées de mappe dans le *Guide d'administration*.

Répartition des transactions

Utilisez JMS (Java Message Service) pour les modifications de transaction répartie entre différents groupes de serveurs ou dans des environnements mixtes.

JMS est un protocole idéal pour les modifications réparties entre différents groupes de serveurs ou dans des environnements mixtes. Par exemple, certaines applications qui utilisent eXtreme Scale peuvent être déployées sur IBM WebSphere Application Server Community Edition, Apache Geronimo ou Apache Tomcat alors que d'autres applications peuvent être exécutées sur WebSphere Application Server version 6.x. JMS se prête parfaitement aux modifications réparties entre des homologues eXtreme Scale dans ces environnements différents. Les messages du gestionnaire de haute disponibilité sont transférés très rapidement mais peuvent uniquement répartir les modifications vers les machines virtuelles Java rassemblées dans un groupe central unique. JMS est plus lent mais il autorise le partage d'un ObjectGrid par des ensembles de clients d'applications plus importants et plus variés. JMS est adapté au partage de données dans un ObjectGrid entre un client Swing lourd et une application déployée sur WebSphere Extended Deployment.

Deux fonctionnalités pré-intégrées, le mécanisme d'invalidation de client et la réplication entre homologues, sont des exemples de répartition de modifications transactionnelles JMS. Voir informations relatives à la configuration de la réplication entre homologues avec JMS dans *Guide d'administration* pour plus d'informations.

Implémentation de JMS

JMS est implémenté pour la répartition de modifications transactionnelles à l'aide d'un objet Java qui se comporte comme un ObjectGridEventListener. Cet objet peut propager l'état à l'aide de l'une des quatre méthodes suivantes :

1. Invalidate : toute entrée expulsée, mise à jour ou supprimée est retirée de toutes les machines virtuelles Java homologues à la réception du message.
2. Invalidate conditional : l'entrée est expulsée uniquement si la version locale est identique ou ultérieure à celle disponible sur le diffuseur de publications.
3. Push : toute entrée expulsée, mise à jour, supprimée ou insérée est ajoutée ou écrasée sur toutes les machines virtuelles Java homologues à la réception du message JMS.

4. Push conditional : l'entrée est uniquement mise à jour ou ajoutée côté récepteur si l'entrée locale est moins récente que la version en cours de publication.

Mode écoute pour les modifications de publication

Le plug-in implémente l'interface ObjectGridEventListener pour intercepter l'événement transactionEnd. Lorsque eXtreme Scale appelle cette méthode, le plug-in tente de convertir la liste LogSequence pour toutes les mappes concernées par la transaction en un message JMS et ensuite de la publier. Le plug-in peut être configuré de façon à publier les modifications pour toutes mappes ou un sous-ensemble de mappes. Les objets LogSequence sont traités pour les mappes pour lesquelles la publication est activée. La classe LogSequenceTransformer de l'ObjectGrid sérialise vers un flux une liste LogSequence filtrée pour chaque mappe. Après sérialisation de toutes les listes LogSequence vers le flux, un message ObjectMessage JMS est créé et publié dans une rubrique connue.

Mode écoute pour les messages JMS et application à l'ObjectGrid locale

Le même plug-in lance une unité d'exécution qui tourne en boucle, recevant tous les messages publiés dans la rubrique connue. A l'arrivée d'un message, il transmet le contenu de ce dernier à la classe LogSequenceTransformer au niveau de laquelle il est converti en un ensemble d'objets LogSequence. Une transaction no-write-through est ensuite démarrée. Chaque objet LogSequence est fourni à la méthode Session.processLogSequence qui met à jour les mappes locales pour refléter les modifications. La méthode processLogSequence comprend le mode de répartition. La transaction est validée et le cache local reflète désormais les modifications. Pour plus d'informations sur l'utilisation de JMS pour la répartition de modifications de transaction, voir les informations relatives à la répartition des modifications entre des machines JVM (Java Virtual Machines) homologues dans *Guide d'administration*.

Transactions à partition unique et cross-data-grid

La différence majeure entre WebSphere eXtreme Scale et les solutions classiques de stockage de données (bases de données relationnelles ou bases de données en mémoire) consiste en l'utilisation du partitionnement, qui permet au cache d'évoluer de manière linéaire. Les principaux types de transactions à prendre en compte sont les transactions à une seule partition et les transactions every-partition (inter-data-grid).

En règle générale, les interactions avec le cache peuvent être classées comme des transactions à partition unique ou des transactions cross-data-grid, comme décrit dans la section suivante.

Transactions dans une partition unique

Les transactions dans une partition unique constituent le mode préférentiel d'interaction avec les mémoires cache hébergées par WebSphere eXtreme Scale. Lorsqu'une transaction est limitée à une partition, elle se limite par défaut à une seule machine virtuelle Java et donc à un seul serveur. Un serveur peut exécuter un nombre *M* de transactions par seconde. Si votre système contient *N* ordinateurs, vous pouvez exécuter `M*N` transactions par seconde. Si votre activité augmente et que vous ayez besoin de doubler le nombre de transactions par seconde, vous pouvez doubler *N* en installant davantage d'ordinateurs. Vous pouvez alors répondre à vos besoins en capacité sans modifier l'application, mettre à niveau le matériel ni utiliser l'application hors ligne.

Outre qu'elles permettent au cache d'évoluer de manière significative, les transactions s'exécutant dans une seule partition permettent aussi d'optimiser la disponibilité de ce dernier. Chaque transaction est liée à un seul ordinateur. Une défaillance peut se produire sur l'un des autres ordinateurs (N-1) sans que la réussite ou le temps de réponse de la transaction en soit affecté. Si 100 ordinateurs sont en cours d'exécution et que l'un d'eux échoue, 1 % des transactions en cours au moment de l'échec sont annulées. Après cet échec, WebSphere eXtreme Scale relocalise les partitions qui sont hébergées par le serveur défaillant vers les 99 autres ordinateurs. Pendant cette courte période, ces ordinateurs continuent à exécuter des transactions. Seules les transactions impliquant les partitions qui sont en cours de relocalisation sont bloquées. Une fois le basculement terminé, le cache peut continuer à s'exécuter en étant pleinement opérationnel, c'est-à-dire à 99 % de sa capacité de traitement d'origine. Une fois que le serveur défaillant est remplacé et revenu dans la grille de données, le cache retrouve 100 % de sa capacité.

Transactions Cross-data-grid

En termes de performances, de disponibilité et d'évolutivité, les transactions cross-data-grid sont l'opposé des transactions à partition unique. Elles ont accès à toutes les partitions et donc à chaque ordinateur de la configuration. Chaque ordinateur de la grille de données est sollicité pour rechercher les données, puis renvoyer le résultat. La transaction n'est pas terminée tant que tous les ordinateurs n'ont pas répondu, et donc le traitement de l'ensemble de la grille de données est limité par l'ordinateur le plus lent. L'ajout d'ordinateurs ne permet pas d'améliorer la vitesse de l'ordinateur le plus lent ni d'augmenter la capacité de traitement du cache.

Les transactions cross-data-grid ont des effets semblables sur la disponibilité. Reprenons l'exemple précédent : si 100 serveurs sont en cours d'exécution et que l'un d'eux échoue, 100 % des transactions en cours sont annulées. Après cet échec, WebSphere eXtreme Scale commence à relocaliser les partitions qui sont hébergées par ce serveur vers les 99 autres ordinateurs. Pendant ce temps, avant la fin du basculement, la grille de données ne peut traiter aucune de ces transactions. Une fois le basculement terminé, le cache continue à s'exécuter, mais à un niveau de capacité réduit. Si chaque ordinateur de la grille de données gérait 10 partitions, 10 des 99 ordinateurs restants reçoivent au moins une partition supplémentaire dans le cadre du processus de basculement. L'ajout d'une partition supplémentaire augmente la charge de travail de cet ordinateur d'au moins 10 %. Etant donné que la capacité de la grille de données est limitée à la capacité de traitement de l'ordinateur le plus lent dans une transaction cross-data-grid, en moyenne, le traitement est réduit de 10.

Il est préférable d'utiliser des transactions à une seule partition que des transactions cross-data-grid pour l'évolutivité avec un cache d'objet haute disponibilité réparti tel que WebSphere eXtreme Scale. L'optimisation des performances de ces systèmes requiert l'utilisation de techniques qui diffèrent des méthodologies relationnelles traditionnelles, mais vous pouvez transformer les transactions cross-data-grid en transactions évolutives à une seule partition.

Méthodes recommandées en matière de génération de modèles de données évolutifs

Les méthodes recommandées pour générer des applications évolutives avec des produits tels que WebSphere eXtreme Scale sont au nombre de deux : les principes fondamentaux et les conseils relatifs à l'implémentation. Les principes fondamentaux sont les grandes idées que vous devez capturer lors de la

conception des données. Une application n'observant pas ces principes aura peu de chance de pouvoir évoluer de manière satisfaisante, même pour les transactions principales. Les conseils d'implémentation s'appliquent aux transactions posant problème dans une application par ailleurs bien conçue et observant les principes généraux relatifs aux modèles de données évolutifs.

Principes fondamentaux

Certains concepts ou principes de base à ne pas oublier constituent les éléments clés pour optimiser l'évolutivité.

Duplication plutôt que normalisation

Il est essentiel de bien comprendre que les produits tels que WebSphere eXtreme Scale sont conçus pour répartir des données dans un grand nombre d'ordinateurs. Si votre objectif est d'exécuter la plupart ou l'ensemble des transactions sur un même ordinateur, le modèle de données doit s'assurer que toutes les données nécessaires à la transaction sont situées dans cette partition. Dans la plupart des cas, la seule manière d'y parvenir consiste à dupliquer les données.

Prenons l'exemple d'un forum électronique. Deux transactions très importantes pour ce forum présentent tous les messages publiés par un utilisateur donné et tous les messages publiés sur un sujet donné. Imaginez tout d'abord comment ces transactions se comporteraient dans le cadre d'un modèle de données normalisé contenant un enregistrement utilisateur, un enregistrement relatif au sujet et un enregistrement relatif au message et contenant le texte réel. Si les messages publiés sont partitionnés avec les enregistrements utilisateur, l'affichage du sujet devient une transaction impliquant l'ensemble de la grille, et inversement. Il est impossible de partitionner les sujets et les utilisateurs car leur relation est de type many-to-many.

La meilleure méthode pour permettre à ce forum électronique d'évoluer est de dupliquer les messages publiés, c'est-à-dire de copier chaque message avec l'enregistrement relatif au sujet et avec l'enregistrement utilisateur. L'affichage des messages publiés à partir d'un utilisateur constitue alors une transaction dans une partition unique, de même que l'affichage des messages publiés dans un sujet, tandis que la mise à jour ou la suppression d'un message publié est une transaction impliquant deux partitions. Ces trois transactions évolueront de manière linéaire à mesure que des ordinateurs sont ajoutés à la grille de données.

L'évolutivité plutôt que les ressources

Le principal obstacle à surmonter en matière de modèles de données dénormalisés est l'impact de ces modèles sur les ressources. La conservation de deux ou trois copies, voire plus, de certaines données risque d'entraîner la consommation d'une trop grande quantité de ressources pour être pratique. Lorsque vous êtes confronté à un tel scénario, gardez ceci en mémoire : les coûts liés à l'achat de matériel diminuent d'année en année. Autre considération, et non des moindres : WebSphere eXtreme Scale permet de supprimer la plupart des coûts associés au déploiement de ressources supplémentaires.

Mesurez les ressources en termes de coût plutôt qu'en termes purement informatiques comme les mégaoctets et les processeurs. Les magasins de données utilisant des données relationnelles normalisées doivent généralement être situés sur le même ordinateur. Cela signifie que vous

devez acheter un seul ordinateur d'entreprise puissant, plutôt que plusieurs machines modestes. Il n'est pas rare qu'un ordinateur d'entreprise pouvant exécuter un million de transactions par seconde soit bien plus onéreux que dix ordinateurs pouvant traiter 100 000 transactions par seconde.

L'ajout de ressources a également un coût. Une entreprise en pleine croissance finit par manquer de capacité. Lorsque vous vous trouvez dans cette situation, vous devez soit arrêter votre système lors du passage à un ordinateur plus rapide et plus puissant, soit créer un second environnement de production. Quelle que soit l'option choisie, vous devrez prendre en charge des coûts supplémentaires liés à la réduction du volume de traitement ou au maintien de la capacité de traitement, pratiquement doublée pendant la durée de la transaction.

Avec WebSphere eXtreme Scale, il n'est pas nécessaire de fermer l'application lors de l'accroissement de la capacité. Si votre entreprise prévoit que vous avez besoins de 10 % de capacité de plus pour l'année prochaine, augmentez le nombre d'ordinateurs de 10 % dans la grille de données. Ce pourcentage peut augmenter sans entraîner d'indisponibilité de l'application et sans que vous ayez à accroître la capacité.

Des transformations de données inutiles

Lorsque vous utilisez WebSphere eXtreme Scale, vous devez stocker les données dans un format directement utilisable par la logique métier. La répartition des données dans un format plus primitif consomme davantage de ressources. La transformation doit se faire lors de l'écriture et lors de la lecture des données. Dans le cas d'une base de données relationnelle, cette transformation est nécessaire car les données sont enregistrées fréquemment sur le disque, mais avec WebSphere eXtreme Scale, elle n'est pas obligatoire. La plupart des données sont stockées en mémoire et peuvent donc être stockées au format requis par l'application.

L'observation de cette règle simple vous aide à dénormaliser les données conformément au premier principe. Le type de transformation des données métier le plus commun est l'opération JOIN nécessaire pour transformer des données normalisées en un ensemble de résultats correspondant aux besoins de l'application. Le stockage des données au format correct permet implicitement d'éviter d'avoir à exécuter ces opérations de type JOIN et génère un modèle de données dénormalisé.

Abandon des requêtes illimitées

Quelle que soit la structure de vos données, les requêtes illimitées évoluent mal. Nous ne vous recommandons par exemple pas d'exécuter une transaction visant à obtenir une liste de tous les articles triés par valeur. Elle peut renvoyer un résultat correct au début, lorsque le nombre d'articles est égal à 1 000, mais lorsque celui-ci atteint 10 millions, la transaction renvoie ces 10 millions d'articles. L'exécution d'une telle transaction risque d'entraîner un délai d'inactivité de celle-ci ou une erreur liée à une insuffisance de mémoire du client.

La meilleure solution consiste à modifier la logique métier de façon que seuls les 10 ou 20 vingt premiers articles soient renvoyés. Cette modification permet de faire en sorte que la transaction soit gérable quel que soit le nombre d'articles présents en cache.

Définition d'un schéma

Le principal avantage lié à la normalisation des données est que la base de données peut prendre en charge la cohérence des données en arrière-plan. Lorsque les données sont dénormalisées à des fins d'évolutivité, la gestion automatique de la cohérence des données disparaît. Vous devez implémenter un modèle de données pouvant fonctionner dans la couche d'applications ou en tant que plug-in de la grille de données répartie pour garantir la cohérence des données.

Prenons l'exemple du forum électronique. Si une transaction supprime un message publié d'un sujet, sa copie dans l'enregistrement utilisateur doit également être supprimée. Sans modèle de données, il est possible que le développeur prévoie la suppression du message publié dans le code de l'application, mais qu'il oublie de le supprimer de l'enregistrement utilisateur. Toutefois, s'il avait utilisé un modèle de données au lieu d'interagir directement avec le cache, la méthode removePost aurait permis d'extraire l'ID utilisateur du message, de rechercher l'enregistrement utilisateur et de supprimer la copie du message en arrière-plan.

Vous pouvez également implémenter un programme d'écoute s'exécutant sur la partition et capable de détecter la modification apportée au sujet et de modifier automatiquement l'enregistrement utilisateur. Un tel programme peut se révéler avantageux car la modification de l'enregistrement utilisateur est effectuée localement si celui-ci se trouve sur la partition ou, s'il se trouve sur une autre partition, la transaction est exécutée entre deux serveurs et non entre le client et le serveur. La connexion réseau entre des serveurs est probablement plus rapide que la connexion existant entre le client et le serveur.

Disparition des conflits

Evitez les scénarios contenant par exemple un compteur global. La grille de données n'évoluera pas si un enregistrement est utilisé un nombre de fois disproportionné par rapport aux autres enregistrements. Les performances de la grille de données seront limitées par celle de l'ordinateur qui contient cet enregistrement.

Dans une telle situation, essayez de fractionner l'enregistrement afin qu'il soit géré au niveau de la partition. Prenons le cas d'une transaction renvoyant le nombre total d'entrées présentes dans le cache réparti. Plutôt que d'exécuter une opération d'insertion et de suppression accédant à un enregistrement unique qui s'incrémente, installez un programme d'écoute sur chaque partition pour suivre ces opérations. Grâce à ce suivi, les opérations d'insertion et de suppression deviennent des transactions s'exécutant dans une partition unique.

La lecture du compteur devient une opération cross-data-grid, mais dans l'ensemble elle était déjà aussi inefficace qu'une opération cross-data-grid, car ses performances étaient liées à celles de l'ordinateur contenant l'enregistrement.

Conseils relatifs à l'implémentation

Pour optimiser l'évolutivité, prenez en compte les conseils suivants.

Utilisation des index de recherche inversée

Prenons le cas d'un modèle de données dénormalisé dans lequel les enregistrements client sont partitionnés en fonction du numéro d'ID du client. Cette méthode de partitionnement est un choix logique car pratiquement toutes les opérations métier exécutées avec l'enregistrement

client utilisent cet ID. Toutefois, la transaction de connexion, qui est essentielle, ne l'utilise pas. Les données utilisées pour la connexion sont plus fréquemment le nom d'utilisateur ou l'adresse électronique.

L'approche la plus simple pour le scénario de connexion consiste à utiliser une transaction cross-data-grid pour rechercher l'enregistrement client. Comme expliqué précédemment, cette approche n'est pas évolutive.

Autre option : procéder à un partitionnement en fonction du nom d'utilisateur ou de l'adresse électronique. Cette option n'est pas pratique, car toutes les opérations basées sur l'ID du client deviennent des transactions cross-data-grid. De plus, les clients de votre site pourraient souhaiter modifier leur nom d'utilisateur ou leur adresse électronique. Dans les produits tels que WebSphere eXtreme Scale, la valeur utilisée pour partitionner les données doit rester constante.

La bonne solution consiste alors à utiliser un index de recherche inversée. Avec WebSphere eXtreme Scale, il est possible de créer un cache dans la même grille répartie que le cache contenant tous les enregistrements utilisateur. Ce cache est à haute disponibilité, partitionnée et évolutif. Il permet de mapper un nom d'utilisateur ou une adresse électronique vers un ID client. Plutôt que d'avoir une opération impliquant l'ensemble de la grille, il transforme la procédure de connexion en transaction s'exécutant sur deux partitions. Ce scénario n'est pas aussi optimal qu'une transaction impliquant une seule partition, mais la capacité de traitement augmente cependant de manière linéaire par rapport au nombre d'ordinateurs.

Calcul des valeurs lors de l'écriture

Les calculs les plus fréquents, tels que les moyennes ou les totaux, peuvent consommer une grande quantité de ressources car ils supposent la lecture d'un grand nombre d'entrées. Les lectures étant plus fréquentes que les écritures dans la plupart des applications, il est plus efficace de calculer ces valeurs lors de l'écriture, puis de stocker le résultat dans le cache. Les opérations gagnent ainsi en rapidité et en évolutivité.

Zones facultatives

Prenons l'exemple d'un enregistrement utilisateur contenant un numéro de téléphone professionnel, un numéro de téléphone personnel et un numéro de téléphone portable. Tous ces numéros ou une combinaison d'entre eux (ou aucun) peuvent être définis pour un utilisateur. Si les données ont été normalisées, une table utilisateur et une table contenant les numéros de téléphone existent. Vous pouvez alors identifier les numéros de téléphone d'un utilisateur donné à l'aide d'une opération JOIN entre les deux tables.

Pour dénormaliser cet enregistrement, aucune duplication des données n'est nécessaire, car la plupart des utilisateurs ne partagent pas leurs numéros de téléphone. Au contraire, les emplacements vides doivent être autorisés dans l'enregistrement utilisateur. Au lieu de constituer une table contenant les numéros de téléphone, ajoutez trois attributs à l'enregistrement utilisateur, chacun correspondant à un type de numéro de téléphone. L'ajout de ces attributs rend superflue l'opération JOIN et permet d'effectuer la recherche des numéros de téléphone d'un utilisateur en tant qu'opération impliquant une seule partition.

Positionnement des relations many-to-many

Prenons l'exemple d'une application qui assure le suivi de certains produits et des magasins dans lesquels ceux-ci sont commercialisés. Un même produit est vendu dans plusieurs magasins et un même magasin vend

plusieurs produits. Supposons que cette application assure le suivi de 50 revendeurs importants. Chaque produit est vendu dans 50 magasins au maximum, chaque magasin commercialisant des milliers de produits.

Constituez une liste des magasins dans l'entité produit (organisation A) plutôt qu'une liste des produits dans chaque entité magasin (organisation B). Une simple observation des transactions que cette application devrait exécuter permet de comprendre pourquoi l'organisation A est la plus évolutive.

Considérons d'abord les mises à jour. Dans l'organisation A, la suppression d'un produit du stock d'un magasin verrouille l'entité produit. Si la grille de données contient 10 000 produits, seul 1/1O 000 de la grille doit être verrouillé lors de la mise à jour. Dans l'organisation B, la grille de données contient seulement 50 magasins, si bien qu'1/50 de la grille doit être verrouillé pour terminer la mise à jour. Même si les deux opérations peuvent être considérées comme des opérations impliquant une seule partition, l'organisation A permet une meilleure évolutivité.

Considérons maintenant les opérations de lecture avec l'organisation A : la recherche des magasins dans lesquels un produit est commercialisé est une transaction impliquant une seule partition, pouvant évoluer et rapide car elle transmet une faible quantité de données. Avec l'organisation B, cette transaction devient une transaction cross-data-grid, car chaque entité de magasin doit être accessible afin de déterminer si le produit est vendu dans ce magasin, ce qui donne un avantage de performance remarquable pour l'organisation A.

Evolutivité avec les données normalisées

L'évolution du traitement des données est l'une des utilisations légitimes des transactions cross-data-grid. Si une grille de données comporte 5 ordinateurs et qu'une transaction cross-data-grid est distribuée pour trier environ 100 000 enregistrements sur chaque ordinateur, cette transaction trie 500 000 enregistrements. Si l'ordinateur le plus lent dans la grille de données peut traiter 10 de ces transactions par seconde, la grille de données est capable de trier 5 000 000 d'enregistrements par seconde. Si les données de la grille doublent, chaque ordinateur doit trier 200 000 enregistrements et chaque transaction trie 1 million d'enregistrements. Cette progression des données ramène la capacité de l'ordinateur le plus lent à 5 transactions par seconde, ce qui réduit le traitement de la grille de données à 5 transactions par seconde. Cependant, la grille trie toujours les données de 5 000 000 d'enregistrements par seconde.

Dans un tel scénario, le fait de doubler le nombre d'ordinateurs permet à chaque ordinateur de revenir à sa charge précédente et à l'ordinateur le plus lent de traiter 10 de ces transactions par seconde. La capacité de la grille de données reste la même à 10 demandes par seconde, mais chaque transaction traite désormais 1 000 000 d'enregistrements, si bien que la grille a doublé sa capacité en terme de traitement des enregistrements pour atteindre 10 000 000 par seconde.

Pour les applications, telles qu'un moteur de recherche, qui ont besoin d'évoluer à la fois en termes de traitement des données pour s'adapter à la taille croissante de l'Internet et de capacité pour s'adapter à l'augmentation du nombre d'utilisateurs, vous devez créer plusieurs grilles de données avec un traitement circulaire des demandes entre les grilles. Si vous devez augmenter le débit, ajoutez des ordinateurs et ajoutez une grille de

données aux demandes de service. Si vous devez augmenter le traitement des données, ajoutez des ordinateurs et ne modifiez pas le nombre de grilles.

Sécurité

WebSphere eXtreme Scale permet de sécuriser l'accès aux données et l'intégration de fournisseurs de sécurité externes.

Remarque : Dans un magasin de données non mis en cache, une base de données, par exemple, il est probable que certaines fonctions pré-intégrées de sécurité ne vous serviront à rien pour la configuration ou l'activation. Cependant, une fois vos données mises en cache avec eXtreme Scale, vous devez prendre en compte le fait que vos fonctions de sécurité du dorsal ne sont plus actives. Vous pouvez configurer la sécurité de eXtreme Scale aux niveaux nécessaires, de sorte que votre nouvelle architecture mise en cache soit également sécurisée.
Vous trouverez ci-dessous un bref récapitulatif des fonctions de sécurité de eXtreme Scale. Pour des informations plus détaillées sur la configuration de la sécurité, voir *Guide d'administration* et *Guide de programmation*.

Notions de base sur la sécurité répartie

La sécurité répartie eXtreme Scale se base sur trois concepts :

Authentification approuvée
: Possibilité de déterminer l'identité du demandeur. WebSphere eXtreme Scale prend en charge l'authentification client-serveur et serveur-serveur.

Autorisation
: Possibilité d'octroyer des droits d'accès au demandeur. WebSphere eXtreme Scale prend en charge différentes autorisations pour des opérations diverses.

Transfert sécurisé
: Transmission sécurisé des données sur le réseau. WebSphere eXtreme Scale prend en charge les protocoles Transport Layer Security/Secure Sockets Layer (TLS/SSL).

Authentification

WebSphere eXtreme Scale prend en charge les structures de serveurs clients répartis. Une infrastructure de sécurité du serveur client est en place pour sécuriser l'accès aux serveurs eXtreme Scale. Par exemple, lorsque l'authentification est requise par le serveur eXtreme Scale, un client eXtreme Scale doit fournir ses informations d'identification pour s'authentifier sur le serveur. Ces informations peuvent être un nom d'utilisateur et un mot de passe, un certificat client, un ticket Kerberos ou des données présentées dans un format choisi par le client et le serveur.

Autorisation

Les autorisations WebSphere eXtreme Scale sont basées sur des objets et des permissions. Vous pouvez utiliser le service JAAS (Java Authentication and Authorization Services) pour autoriser l'accès, ou vous pouvez choisir une approche personnalisée, telle que Tivoli Access Manager (TAM), pour gérer les autorisations. Les autorisations suivantes peuvent être octroyées à un client ou un groupe :

Autorisation de mappes
Effectuez des opérations d'insertion, de lecture, de mise à jour, d'expulsion ou de suppression sur les mappes.

Autorisation ObjectGrid
Lancez des requêtes sur un objet ou une entité et des requêtes de flux sur les objets ObjectGrid.

Autorisation de l'agent DataGrid
Permet aux agents DataGrid d'être déployés en une base de données ObjectGrid.

Autorisation de mappes côté serveur
Répliquez une mappe de serveur côté client ou créez un index dynamique pour la mappe de serveur.

Autorisation d'administration
Effectuez des tâches d'administration.

Sécurité du transfert

Pour sécuriser la communication du serveur client, WebSphere eXtreme Scale prend en charge les protocoles TLS/SSL. Ces protocoles fournissent une sécurité de couche de transport, avec des fonctions d'authentification, d'intégrité et de confidentialité pour une connexion sécurisée entre le client eXtreme Scale et le serveur.

Sécurité de grille

Dans un environnement sécurisé, un serveur doit être capable de vérifier l'authenticité d'un autre serveur. WebSphere eXtreme Scale utilise un mécanisme de clé secrète partagée dans ce but. Ce mécanisme est similaire à un mot de passe partagé. Tous les serveurs eXtreme Scale s'accordent sur une clé secrète partagée. Lorsqu'un serveur rejoint la grille de données, il est invité à présenter la chaîne secrète. Si la clé secrète du serveur tentant de se joindre correspond à la clé sur serveur principal, le serveur peut se joindre à la grille. Dans le cas contraire, la requête de jointure est rejetée.

L'envoi d'une clé secrète en texte clair n'est pas sécurisé. L'infrastructure de sécurité eXtreme Scale fournit un plug-in SecureTokenManager pour permettre au serveur de sécuriser cette clé secrète avant l'envoi. Vous pouvez choisir la façon dont vous souhaitez implémenter l'opération sécurisée. Avec WebSphere eXtreme Scale, une opération sécurisée est implémentée pour chiffrer et signer la clé secrète.

Fonctions de sécurité Java Management Extensions (JMX) dans une topologie de déploiement dynamique

Les fonctions de sécurité JMX MBeans sont prises en charge dans toutes les versions de eXtreme Scale. Les clients des beans gérés de serveur de catalogue et de serveur de conteneur peuvent être authentifiés, et l'accès aux opérations MBean peut être forcé.

Sécurité eXtreme Scale locale

La sécurité eXtreme Scale locale est différente du modèle eXtreme Scale réparti car l'application s'instancie directement et utilise une instance ObjectGrid. Votre application et les instances eXtreme Scale se trouvent dans la même machine virtuelle Java (JVM). Etant donné qu'aucun concept client-serveur n'existe dans ce

modèle, l'authentification n'est pas prise en charge. Vos applications doivent gérer leur propre authentification, puis transmettre l'objet authentifié à eXtreme Scale. Cependant, le mécanisme d'autorisation utilisé pour le modèle de programmation eXtreme Scale est le même que celui utilisé pour le modèle client-serveur.

Configuration et programmation

Pour plus d'informations sur la configuration et la programmation de la sécurité, voir Intégration de la sécurité à des fournisseurs externes et API de sécurité.

Présentation des services de données REST

Le service de données REST de WebSphere eXtreme Scale est un service HTTP Java qui est compatible avec Microsoft WCF Data Services (ex-ADO.NET Data Services) et qui implémente Open Data Protocol (OData). Microsoft WCF Data Services est compatible avec cette spécification lorsqu'on utilise Visual Studio 2008 SP1 et .NET Framework 3.5 SP1.

Compatibilités requises

Le service de données REST autorise n'importe quel client HTTP à accéder à une grille de données. Il est compatible avec la prise en charge de WCF Data Services, qui est fournie avec Microsoft .NET Framework 3.5 SP1. Il est possible de développer des applications compatibles REST avec les outils fournis par Microsoft Visual Studio 2008 SP1. L'illustration montre l'interaction de WCF Data Services avec les clients et les bases de données.

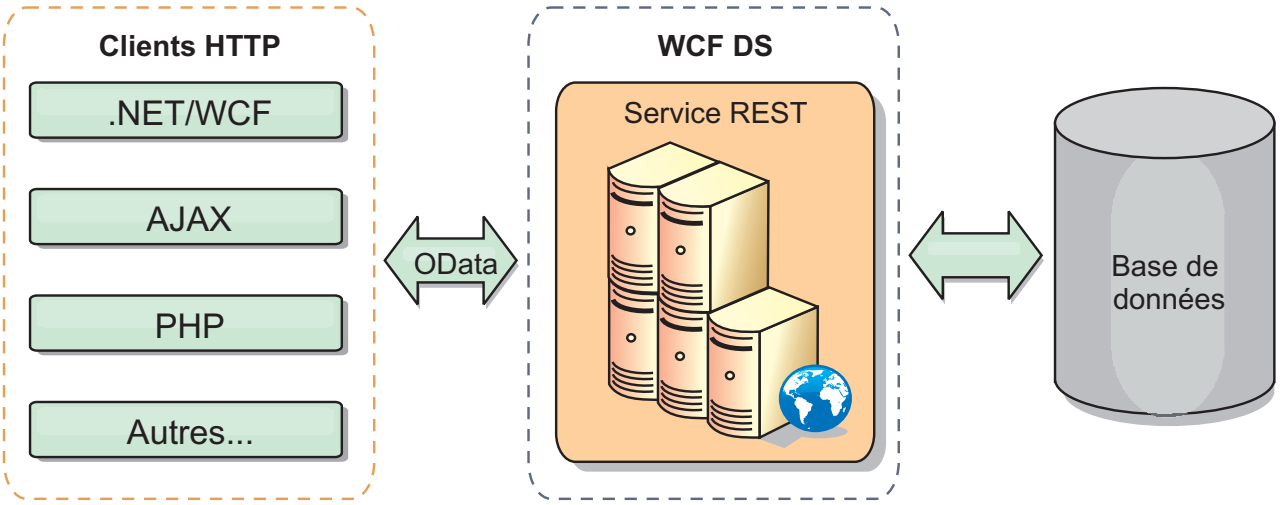


Figure 36. Microsoft WCF Data Services

WebSphere eXtreme Scale inclut un ensemble d'API riche en fonctionnalités pour les clients Java. Comme le montre l'illustration ci-dessous, le service de données REST est une passerelle entre les clients HTTP et la grille de données WebSphere eXtreme Scale, la communication avec la grille s'effectuant via un client WebSphere eXtreme Scale. Le service de données REST est un servlet Java qui permet des déploiements flexibles pour des plateformes Java Platform, Enterprise Edition (JEE) comme WebSphere Application Server. Le service de données REST communique avec la grille de données WebSphere eXtreme Scale en utilisant les API WebSphere eXtreme Scale Java. Il autorise le recours aux clients WCF Data Services ou à tout autre client capable de communiquer avec HTTP et XML.

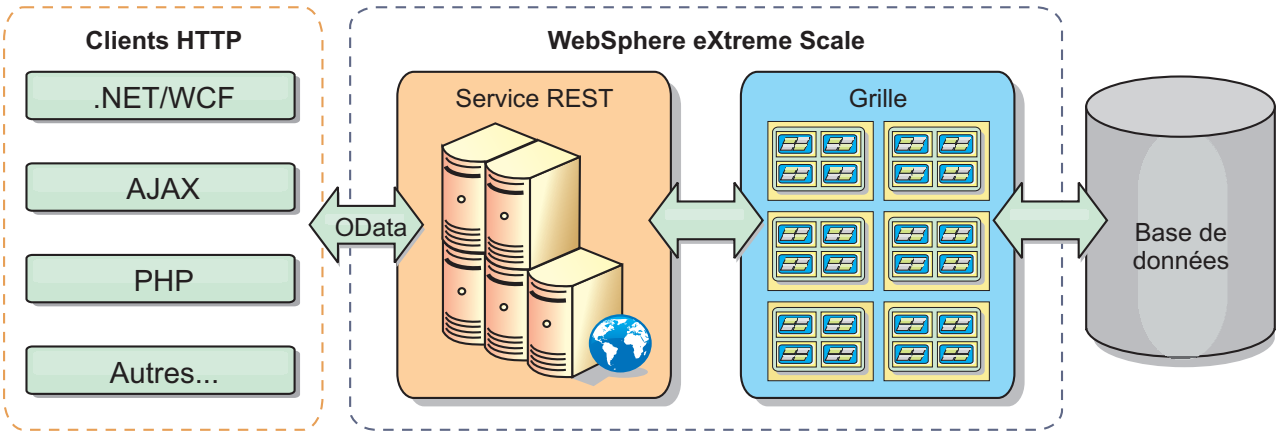


Figure 37. Service de données REST de WebSphere eXtreme Scale

Pour en savoir davantage sur WCF Data Services, reportez-vous à Configuration des services de données REST ou suivez les liens ci-dessous.

- Microsoft WCF Data Services Developer Center
- Présentation d'ADO.NET Data Services sur MSDN
- Livre blanc : Using ADO.NET Data Services
- Atom Publish Protocol : Data Services URI and Payload Extensions
- Conceptual Schema Definition File Format
- Entity Data Model for Data Services Packaging Format
- Open Data Protocol
- Open Data Protocol FAQ

Fonctionnalités

Cette version du service de données REST eXtreme Scale prend en charge les fonctionnalités suivantes :

- modélisation automatique des entités de l'API EntityManager d'eXtreme Scale en tant qu'entités WCF Data Services, ce qui inclut les prises en charge suivantes :
  - conversion du type de données Java en type Entity Data Model
  - prise en charge de l'association d'entités
  - prise en charge de la racine de schéma et de l'association de clés, obligatoire pour les grilles de données partitionnées

  Pour plus d'informations, voir Modèle d'entité.
- format XML Atom Publish Protocol (AtomPub ou APP) et format JavaScript Object Notation (JSON) de contenu des données
- opérations CRUD (Create, Read, Update et Delete) à l'aide des méthodes respectives de demandes HTTP : POST, GET, PUT et DELETE. En plus, l'extension Microsoft MERGE est prise en charge
- requêtes simples à l'aide de filtres
- extraction par lot et demandes d'ensembles de modifications
- support de grille de données partitionnées pour la haute disponibilité
- interopérabilité avec les clients API EntityManager d'eXtreme Scale
- prise en charge des serveurs Web JEE standard
- contrôle d'accès simultanés

- autorisation et authentification des utilisateurs entre le service de données REST et la grille de données eXtreme Scale

Problèmes connus et limitations

- Les demandes placées en tunnel ne sont pas prises en charge.

Chapitre 2. Planification

Avant d'installer WebSphere eXtreme Scale et de déployer vos applications de grille de données, vous devez choisir votre topologie de mise en cache, planifier la capacité, vérifier les configurations matérielle et logicielle requises et les paramètres de réseau et d'optimisation, etc. Vous pouvez également utiliser la liste de contrôle opérationnelle pour vérifier que votre environnement est prêt pour le déploiement d'applications.

Vous trouverez une discussion des pratiques recommandées pour la conception d'applications WebSphere eXtreme Scale dans l'article suivant de developerWorks : Principles and best practices for building high performing and highly resilient WebSphere eXtreme Scale applications.

Planification de la topologie

Avec WebSphere eXtreme Scale, l'architecture de votre système peut utiliser la mise en cache des données locales en mémoire ou la mise en cache des données client-serveur réparties. L'architecture peut avoir des relations différentes avec vos bases de données. Vous pouvez également configurer la topologie pour l'étendre à plusieurs centres de données.

WebSphere eXtreme Scale requiert une infrastructure supplémentaire minimale pour pouvoir fonctionner. Cette infrastructure consiste en des scripts permettant d'installer, de démarrer et d'arrêter une application Java Platform, Enterprise Edition sur un serveur. Les données mises en cache sont stockées dans les serveurs de conteneur et les clients se connectent à distance au serveur.

Environnements internes

Lors du déploiement dans un environnement interne, WebSphere eXtreme Scale s'exécute dans une seule machine virtuelle Java et il n'est pas répliqué. Pour configurer un environnement local, vous pouvez utiliser un fichier XML ObjectGrid ou les API ObjectGrid.

Environnement réparti

Lorsque effectuez le déploiement dans un environnement réparti, WebSphere eXtreme Scale s'exécute dans un ensemble de machines virtuelles Java, ce qui améliore les performances, la disponibilité et l'évolutivité. Dans cette configuration, vous pouvez utiliser les fonctions de réplication et de partitionnement des données. Vous pouvez également ajouter d'autres serveurs sans redémarrer les serveurs eXtreme Scale existants. Comme dans le cas d'un environnement local, un fichier XML ObjectGrid ou une configuration par programmation équivalente est nécessaire dans un environnement réparti. Vous devez également fournir un fichier XML de stratégie de déploiement contenant les détails de la configuration.

Il est possible de créer des déploiements simples ou des déploiements plus vastes se chiffrant en téraoctets et comptant plusieurs milliers de serveurs.

Cache interne local

Dans le cas le plus simple, WebSphere eXtreme Scale peut être utilisé comme cache de grille de données locale (non répartie) en mémoire. Cette mise en cache locale peut s'avérer particulièrement utile pour les applications au nombre d'accès simultanés élevé où plusieurs unités d'exécution doivent accéder aux données temporaires et les modifier. Les données conservées dans une grille de données locale peuvent être indexées et extraites à l'aide de requêtes. Les requêtes permettent d'utiliser des jeux de données volumineux en mémoire. Le support fourni avec machine virtuelle Java (JVM), qui est prêt à être utilisé, dispose d'une structure de données limitées.

La topologie de cache local en mémoire de WebSphere eXtreme Scale permet d'octroyer un accès cohérent et transactionnel aux données temporaires dans une machine virtuelle Java unique.

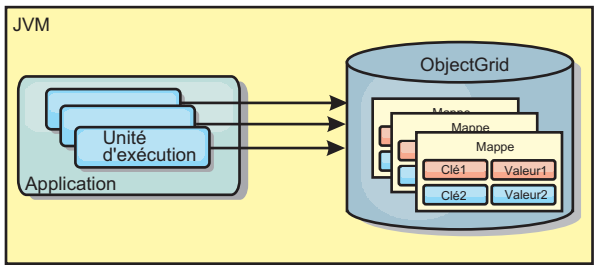


Figure 38. Scénario de cache local en mémoire

Avantages

- Configuration simple : une ObjectGrid peut être créée à l'aide d'un programme ou de manière déclarative avec le fichier XML du descripteur de déploiement ObjectGrid ou à l'aide d'une autre structure telle que Spring.
- Rapide : chaque mappe de sauvegarde peut être ajustée de façon indépendante pour optimiser l'utilisation de la mémoire et des accès simultanés.
- Configuration idéale pour les topologies de machine virtuelle Java dotées de petits jeux de données ou pour la mise en cache de données fréquemment consultées.
- Transactionnelle. Les mises à jour de mappe de sauvegarde peuvent être regroupées dans la même unité d'oeuvre et peuvent être intégrées en dernier lieu aux transactions constituées de deux phases telles que les transactions JTA (Java Transaction Architecture).

Inconvénients

- Aucune tolérance de panne.
- Les données ne sont pas répliquées. Les mémoires cache internes se prêtent aux données de référence en lecture seule.
- Non évolutive. La quantité de mémoire requise par la base de données peut dépasser la capacité de la machine virtuelle Java.
- Problèmes survenant lors de l'ajout de machines virtuelles Java :
  - Les données ne peuvent pas être facilement partitionnées ;
  - Nécessité de répliquer manuellement l'état entre les machines virtuelles Java ou chaque instance de cache peut présenter différentes versions des mêmes données.

- L'invalidation est coûteuse.
- Chaque cache doit être préchauffé indépendamment. Le préchauffage est la période de chargement d'un jeu de données permettant de remplir le cache avec des données valides.

Utilisation

La topologie de déploiement de la mémoire cache interne locale ne doit être utilisée que lorsque la quantité de données à mettre en cache est limitée (peut être abritée par une seule machine virtuelle Java) et est relativement stable. Cette approche doit tolérer les données obsolètes. L'utilisation d'expulseurs pour conserver les données les plus fréquemment ou récemment utilisées dans le cache peut contribuer à réduire la taille du cache et à accroître la pertinence des données.

Cache local répliqué sur des homologues

Vous devez vous assurer que le cache est synchronisé si plusieurs processus avec des instances indépendantes de cache existent. Pour vérifier que les instances de cache sont synchronisées, activez un cache répliqué sur des homologues avec JMS (Java Message Service).

WebSphere eXtreme Scale comprend deux plug-in qui propagent automatiquement les modifications de transactions entre les instances ObjectGrid homologues. Le plug-in JMSObjectGridEventListener propage automatiquement les modifications eXtreme Scale à l'aide de JMS.

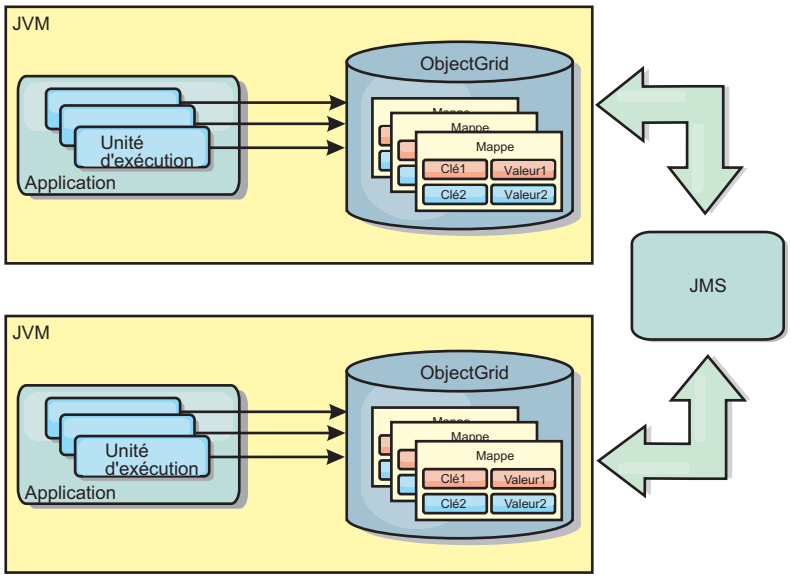


Figure 39. Cache répliqué sur des homologues avec des modifications qui sont propagées à l'aide de JMS

Si vous exécutez un environnement WebSphere Application Server, le plug-in TranPropListener est aussi disponible. Il utilise le gestion HA (high availability) pour propager les modifications à chaque instance de cache homologue.

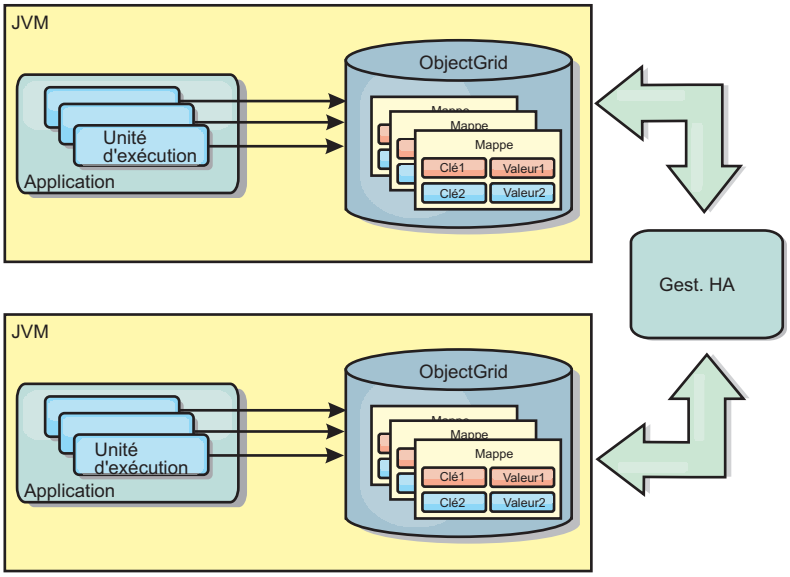


Figure 40. Cache répliqué sur des homologues avec des modifications qui sont propagées à l'aide du gestionnaire de haute disponibilité

Avantages

- Plus grande validité des données car celles-ci sont actualisées plus souvent.
- Avec le plug-in TranPropListener, tout comme avec l'environnement local, il est possible de créer la grille de données eXtreme Scale par programmation ou de manière déclarative avec le fichier XML du descripteur de déploiement d'eXtreme Scale ou avec d'autres structures de travail comme Spring. L'intégration au gestionnaire de haute disponibilité s'effectue automatiquement.
- Chaque mappe de sauvegarde peut être optimisée indépendamment en termes d'utilisation de la mémoire et de simultanéité des accès.
- Il est possible de regrouper en une seule unité d'oeuvre les mises à jour des mappes de sauvegarde qui peuvent être intégrées comme derniers participants de transactions en deux phases comme le sont les transactions Java Transaction Architecture (JTA).
- Idéal pour les topologies comprenant un nombre restreint de machines virtuelles Java avec un dataset de taille raisonnablement réduite ou pour la mise en cache des données à accès fréquent.
- Les modifications de la grille de données eXtreme Scale sont répliquées à toutes les instances eXtreme Scale homologues. Les modifications sont cohérentes tant qu'un abonnement durable est utilisé.

Inconvénients

- La configuration et la maintenance du plug-in JMSObjectGridEventListener peut s'avérer une tâche complexe. Il est possible de créer la grille de données eXtreme Scale par programmation ou de manière déclarative avec le fichier XML du descripteur de déploiement d'eXtreme Scale ou avec d'autres structures de travail comme Spring.
- Pas d'extensibilité : la quantité de mémoire requise par la base de données risque de submerger la machine virtuelle Java.
- Fonctionne de manière incorrecte lorsqu'on ajoute des machines virtuelles Java :
  - les données ne sont pas facilement partitionnées
  - l'invalidation est onéreuse

– chaque cache doit être prérempli de manière indépendante

Quand l'utiliser

Utilisez la topologie de déploiement uniquement lorsque la quantité de données à mettre en cache est faible, peut tenir sur une seule machine virtuelle Java, et relativement stable.

Cache imbriqué

Les grilles WebSphere eXtreme Scale peuvent s'exécuter dans des processus existants, tels que des serveurs eXtreme Scale intégrés ou vous pouvez les gérer comme des processus externes.

Les grilles imbriquées sont utiles lorsque l'exécution se fait dans un serveur d'applications tel que WebSphere Application Server. Vous pouvez démarrer les serveurs eXtreme Scale non imbriqués à l'aide de scripts de ligne de commande et les exécuter dans un processus Java.

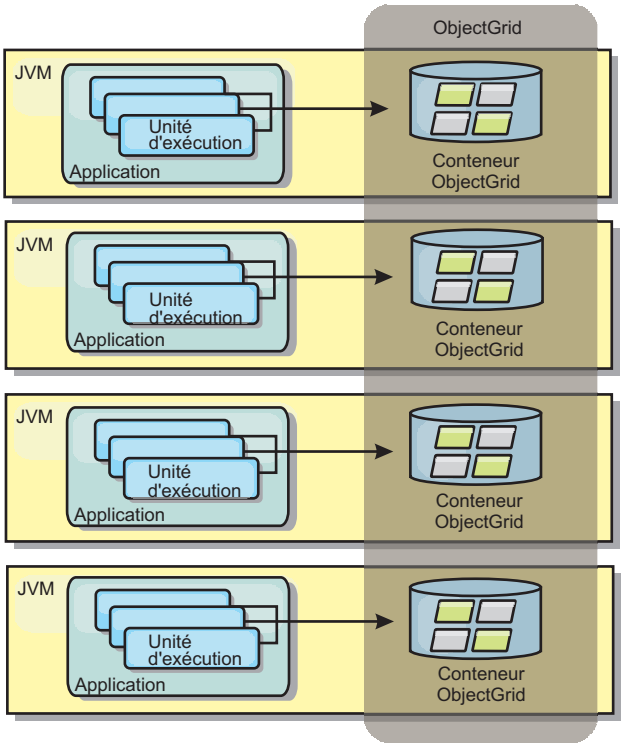


Figure 41. Cache imbriqué

Avantages

- simplification de l'administration en raison du nombre inférieur de processus à gérer
- simplification du déploiement d'application car la grille utilise le chargeur de classe de l'application client
- prise en charge du partitionnement et de la haute disponibilité.

Inconvénients

- augmentation de l'encombrement mémoire dans le processus client car toutes les données sont regroupées dans le processus
- augmentation de l'utilisation de l'unité centrale en vue de la gestion des demandes des clients
- plus grande difficulté à gérer les mises à niveau des applications car les clients utilisent les mêmes fichiers d'archive Java que les serveurs
- moindre flexibilité. Les clients et les serveurs de grille ne peuvent évoluer au même rythme. Lorsque des serveurs sont définis en externe, la gestion du nombre de processus devient plus flexible

Utilisation

Utilisez les grilles imbriquées lorsqu'une grande quantité de mémoire est disponible dans le processus client pour les données de la grille et pour les données de basculement.

Plus d'informations, voir la rubrique relative à l'activation du mécanisme d'invalidation de client dans *Guide d'administration*.

Cache réparti

La plupart du temps, WebSphere eXtreme Scale est utilisé en tant que cache partagé permettant un accès transactionnel aux données de plusieurs composants là où une base de données classique aurait été nécessaire. Avec le cache partagé, il n'est plus nécessaire de configurer une base de données.

Cohérence de la mémoire cache

Le cache est cohérent car tous les clients y voient les mêmes données. Chaque donnée est stockée dans le cache sur un seul serveur ce qui permet d'éviter la coexistence de plusieurs copies d'enregistrements risquant de contenir des versions différentes des données. Un cache cohérent contient un nombre croissant de données au fur et à mesure que l'on ajoute des serveurs à la grille et le cache évolue de manière linéaire au fur et à mesure que la taille de la grille augmente. Comme les clients accèdent aux données de cette grille de données avec des appels de procédure distante, cette mémoire est également appelée cache distant ou éloigné. Grâce au partitionnement des données, chaque processus contient un sous-ensemble unique de données. Les grandes grilles peuvent contenir davantage de données et traiter plus de demandes pour ces données. Par ailleurs la cohérence évite d'avoir à envoyer les données d'invalidation autour de la grille de données, car aucune donnée périmée n'existe. Le cache cohérent contient uniquement la copie la plus récente de chaque donnée.

Si vous exécutez un environnement WebSphere Application Server, le plug-in TranPropListener est aussi disponible. Il utilise le composant de haute disponibilité (gestionnaire HA) de WebSphere Application Server pour propager les modifications à chaque instance de cache ObjectGrid homologue.

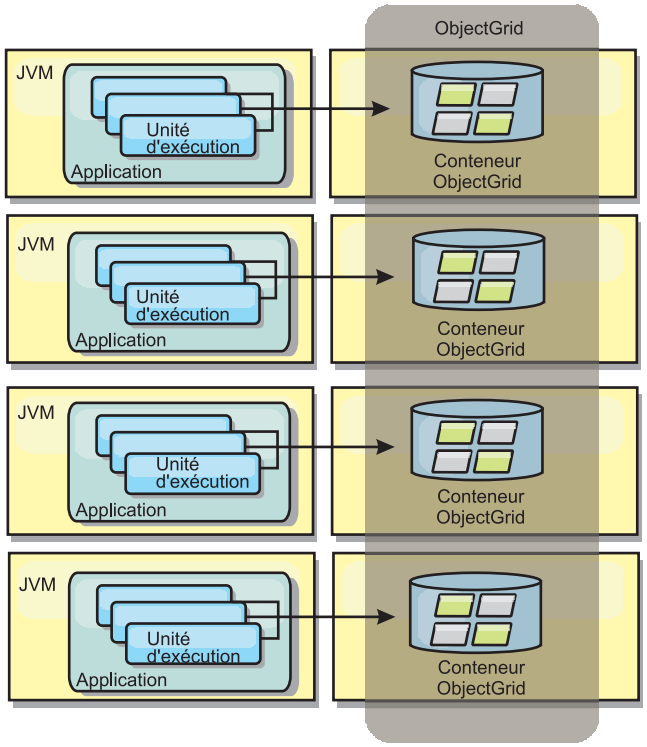


Figure 42. Cache réparti

Cache local

Lorsqu'eXtreme Scale est utilisé dans le cadre d'une topologie répartie, les clients peuvent éventuellement disposer d'un cache local en ligne. L'on appelle cache local ce cache facultatif. Il s'agit d'un ObjectGrid indépendant, présent sur chaque client et faisant office de cache du cache distant côté serveur. Il est activé par défaut lorsque le verrouillage est configuré sur OPTIMISTIC ou sur NONE. Son utilisation est impossible lorsque le verrouillage est configuré sur PESSIMISTIC.

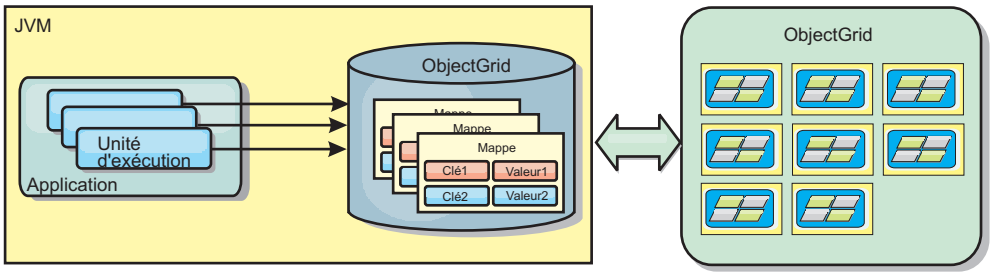


Figure 43. Cache local

Le cache local est très rapide car il offre un accès en mémoire à un sous-ensemble des données stockées à distance sur les serveurs eXtreme Scale. Il n'est pas partitionné et contient des données provenant de n'importe quelle partition eXtreme Scale distante. Jusqu'à trois groupes de caches peuvent exister dans WebSphere eXtreme Scale :

1. Le cache du groupe des transactions contient toutes les modifications apportées à une même transaction. Il contient une copie de travail des données jusqu'à ce que la transaction soit validée. Lorsqu'une transaction client demande des données à une ObjectMap, la transaction est vérifiée en priorité.

2. Le cache local du groupe des clients contient un sous-ensemble des données du groupe des serveurs. Lorsque le groupe des transactions ne contient pas les données, les données sont extraites du niveau client, si elles sont disponibles et insérées dans le cache des transactions.
3. La grille de données dans le groupe des serveurs contient la majorité des données et elle est partagée entre tous les clients. Le groupe des serveurs peut être partitionné, ce qui permet la mise en cache d'un grand nombre de données. Lorsque le cache local ne contient pas de données, celles-ci sont extraites du groupe des serveurs et insérées dans le cache du client. Le groupe des serveurs peut aussi avoir un plug-in Loader. Lorsque la grille ne contient pas les données demandées, le chargeur est appelé et les données résultantes sont insérées dans la grille à partir du magasin de données dorsal.

Pour désactiver le cache local, donnez la valeur 0 à l'attribut numberOfBuckets dans la configuration du descripteur eXtreme Scale des remplacements par le client. Pour plus d'informations sur les stratégies de verrouillage dans eXtreme Scale, consultez la rubrique relative au verrouillage des entrées de mappe. Le cache local peut également être configuré de façon à utiliser d'autres règles d'expulsion et des plug-in différents qui utilisent une configuration de descripteur eXtreme Scale des remplacements par les clients.

Avantage

- Rapidité du temps de réponse, car tous les accès aux données se font localement La recherche de données dans le cache local évite de consulter la grille des serveurs et rend les données distantes accessibles localement.

Inconvénients

- Augmentation de la durée des données obsolètes, car le cache local à chaque niveau est peut-être désynchronisé avec les données en cours dans la grille de données.
- Basé sur un expulseur pour invalider les données afin d'éviter de manquer de mémoire.

Utilisation

A utiliser lorsque le temps de réponse est élevé et que la présence de données périmées est tolérée.

Intégration de la base de données : caches avec écriture différée, caches en ligne et caches secondaires

WebSphere eXtreme Scale est utilisé pour servir de frontal à une base de données classiques et ainsi éliminer l'activité de lecture qui est normalement envoyée vers la base de données. Un cache cohérent peut être utilisé avec une application soit directement, soit indirectement en passant alors par un associateur relationnel d'objets (ORM). Le cache cohérent peut décharger des tâches de lecture la base de données ou le dorsal. Dans un scénario un tout petit peu plus complexe, comme celui d'un accès transactionnel à un dataset dans lequel seules certaines données requièrent des garanties de persistance classique, il est possible d'utiliser le filtrage pour décharger même les transactions d'écriture.

Vous pouvez configurer WebSphere eXtreme Scale pour qu'il fonctionne en tant qu'espace extrêmement flexible de traitement de base de données interne. Cela dit, WebSphere eXtreme Scale n'est pas un associateur relationnel d'objets. Il ne sait pas d'où les données de la grille de données proviennent. Une application ou un

associateur relationnel d'objets peuvent placer des données sur un serveur eXtreme Scale. C'est à la source de données qu'il incombe de vérifier la cohérence des données avec leur base de données d'origine. En d'autres termes, eXtreme Scale ne peut pas invalider les données qu'il a extraites automatiquement d'une base de données. C'est à l'application ou à l'associateur de fournir cette fonction et de gérer les données stockées dans eXtreme Scale.

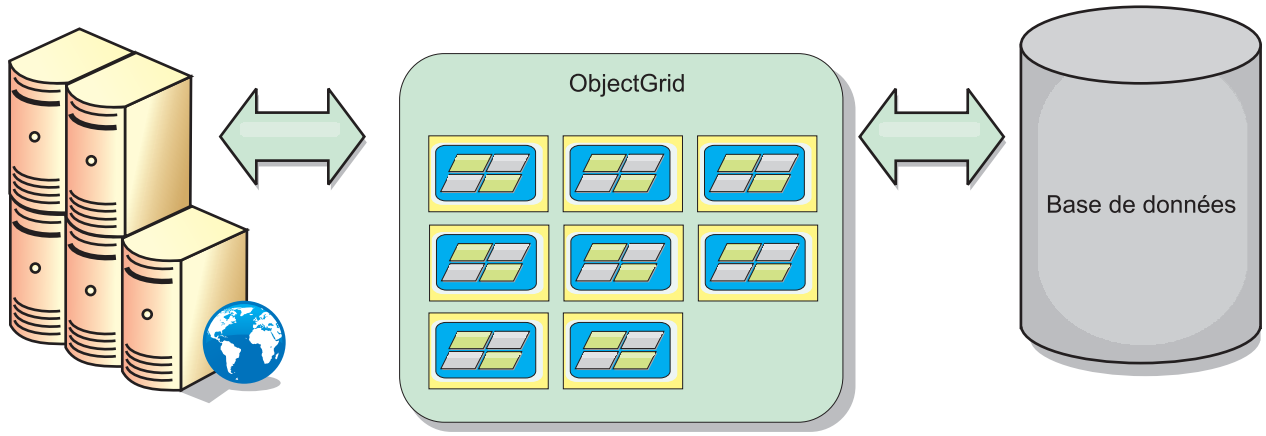


Figure 44. ObjectGrid en tant que mémoire tampon de base de données

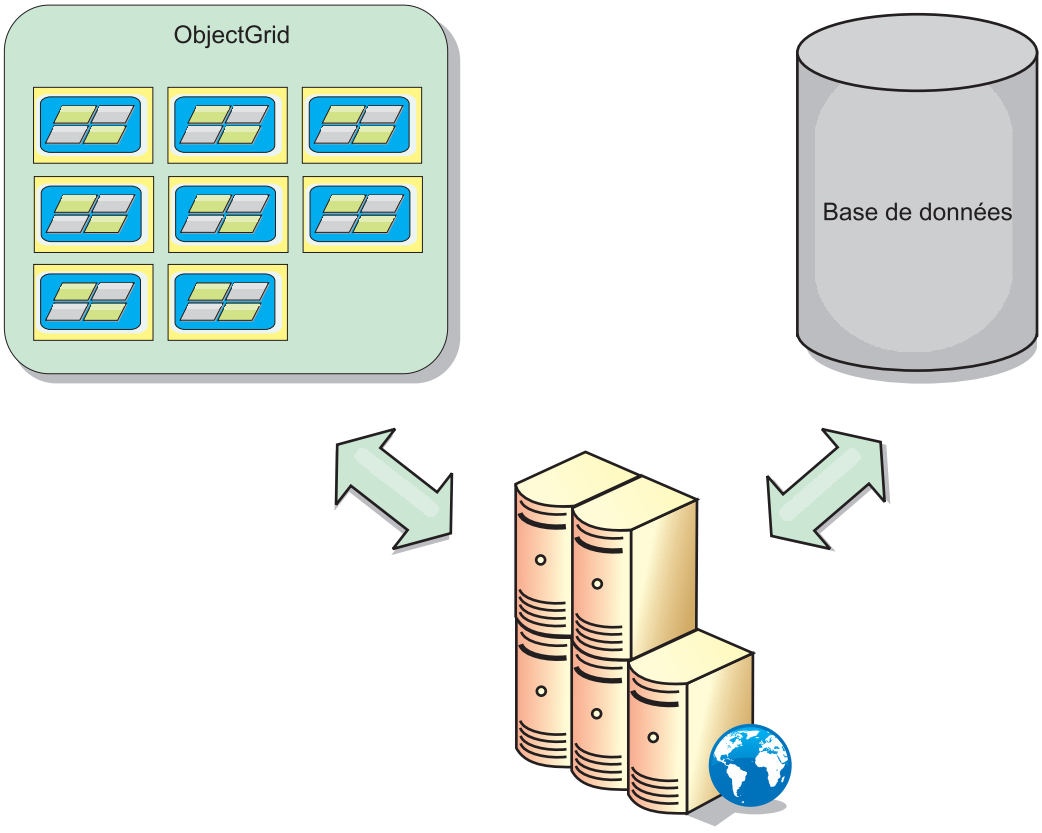


Figure 45. ObjectGrid en tant que cache secondaire

Cache partiel et cache complet

WebSphere eXtreme Scale peut s'utiliser en tant que cache partiel ou que cache complet. Un cache partiel ne conserve qu'un sous-ensemble des données totales,

alors qu'un cache complet conserve toutes les données et peut être rempli en différé en fonction des besoins en données. Les caches partiels sont normalement accessibles à l'aide de clés (et non pas d'index ou de requêtes), car les données sont partiellement disponibles uniquement.

Cache partiel

Si une clé est absente dans un cache partiel ou que les données ne sont pas disponibles et qu'un échec de cache se produit, le niveau suivant est appelé. Les données sont extraites d'une base de données, par exemple, et elles sont insérées au groupe de caches de grille de données. Si vous utilisez une requête ou un index, seules les valeurs actuellement chargées sont accessibles et les requêtes ne sont pas transférées aux autres groupes.

Cache complet

Un cache complet comporte toutes les données requises et il est possible d'y accéder à l'aide d'attributs non-clés avec des index ou des requêtes. Un cache complet est préchargé avec des données de la base de données avant que l'application tente d'accéder aux données. Un cache complet peut fonctionner sous la forme d'un remplacement de base de données une fois que les données sont chargées. Etant donné que toutes les données sont disponibles, les requêtes et les index peuvent être utilisés pour rechercher et agréger les données.

Cache secondaire

Lorsque WebSphere eXtreme Scale est utilisé en tant que cache secondaire, le système dorsal est utilisé avec la grille de données.

Cache secondaire

Vous pouvez configurer le produit en tant que cache secondaire pour la couche d'accès aux données d'une application. Dans ce scénario, WebSphere eXtreme Scale permet de stocker temporairement des objets qui seraient normalement extraits d'une base de données dorsale. Les applications vérifient si la grille de données contient les données. Si les données se trouvent dans la grille de données, ces données sont renvoyées à l'appelant. Si elles n'existent pas, elles sont extraites de la base de données dorsale. Elles sont ensuite insérées dans la grille de données afin que la demande suivante puisse utiliser la copie mise en cache. Le diagramme suivant montre comment WebSphere eXtreme Scale peut être utilisé en tant que cache secondaire à l'aide d'une couche d'accès aux données arbitraire, telle qu'OpenJPA ou Hibernate.

Plug-in de cache secondaire pour Hibernate et OpenJPA

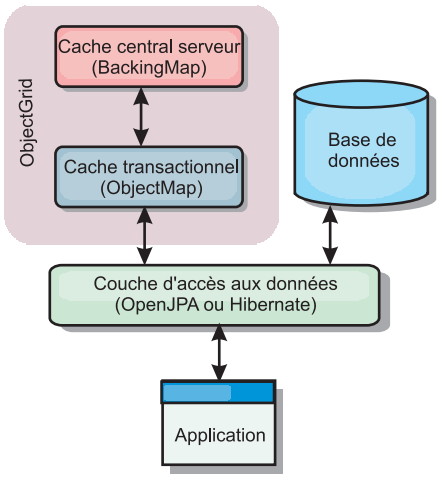


Figure 46. Cache secondaire

Les plug-in de cache pour OpenJPA et Hibernate sont inclus dans WebSphere eXtreme Scale pour que vous puissiez utiliser le produit comme cache secondaire automatique. L'utilisation d'WebSphere eXtreme Scale en tant que fournisseur de cache améliore les performances lors de la lecture et de l'interrogation des données et réduit la charge pesant sur la base de données. WebSphere eXtreme Scale présente plusieurs avantages par rapport à des implémentations de cache pré-intégrées car le cache est automatiquement répliqué entre tous les processus. Lorsqu'un client met une valeur en mémoire cache, tous les autres clients peuvent l'utiliser.

Cache en ligne

Vous pouvez configurer la mise en cache en ligne pour un système dorsal de base de données ou en tant que cache secondaire pour une base de données. La mise en cache en ligne utilise eXtreme Scale comme moyen principal pour interagir avec les données. Lorsque eXtreme Scale est utilisé en tant que cache en ligne, l'application interagit avec le système dorsal à l'aide d'un plug-in Loader.

Cache en ligne

Lorsque WebSphere eXtreme Scale est utilisé en tant que cache en ligne, il interagit avec le système dorsal à l'aide d'un plug-in Loader. Ce scénario permet de simplifier l'accès aux données car les applications peuvent accéder aux API eXtreme Scale directement. Plusieurs scénarios de cache sont pris en charge dans eXtreme Scale pour assurer la synchronisation des données dans le cache et des données dans le système dorsal. Le diagramme suivant illustre l'interaction entre le cache en ligne, l'application et le système dorsal.

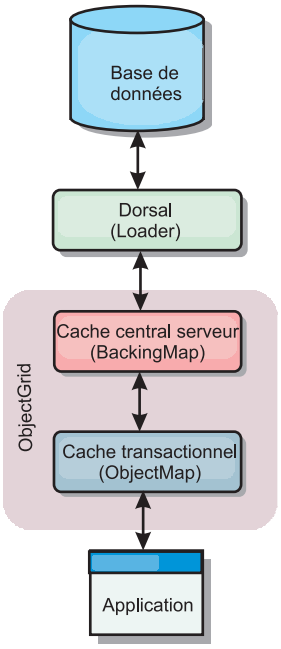


Figure 47. Cache en ligne

L'option de mise en cache en ligne simplifie l'accès aux données en permettant aux applications d'accéder directement aux API eXtreme Scale. WebSphere eXtreme Scale prend en charge plusieurs scénarios de mise en cache en ligne, comme suit.

- Sans interruption
- Ecriture immédiate
- Post-écriture

Scénario de mise en cache sans interruption

Un cache sans interruption est un cache partiel chargeant en lazy loading à partir d'une clé les entrées de données au fur et à mesure que ces entrées sont demandées. Cette opération peut se dérouler sans que l'appelant sache comment sont renseignées les entrées. Si les données sont introuvables dans le cache eXtreme Scale, eXtreme Scale récupère les données manquantes auprès du plug-in Loader qui charge les données provenant de la base de données d'arrière plan et les insère dans le cache. Les requêtes suivantes pour la même clé de données se trouveront dans le cache, jusqu'à ce qu'elles soient supprimées, invalidées ou expulsées.

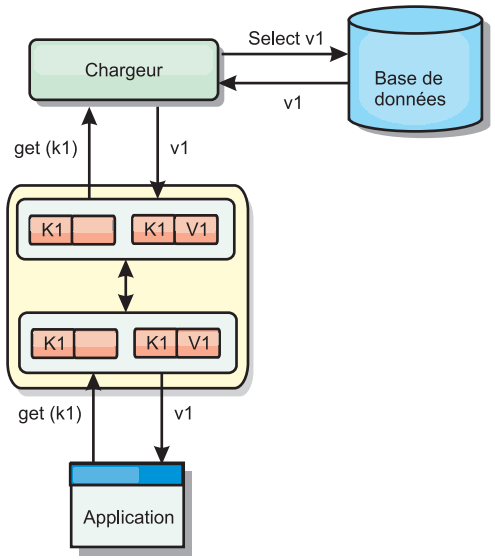


Figure 48. Mise en cache sans interruption

Scénario de mise en cache à écriture immédiate

Dans un cache à écriture immédiate, chaque écriture dans le cache est inscrite de manière synchrone dans la base de données à l'aide du chargeur. Cette méthode permet la cohérence avec le système dorsal, mais réduit les performances d'écriture étant donné que l'opération de base de données est synchrone. Le cache et la base de données étant tous deux mis à jour, les lectures suivantes à la recherche des mêmes données auront lieu dans le cache, évitant ainsi de faire appel à la base de données. Un cache à écriture immédiate est souvent utilisé conjointement à un cache sans interruption.

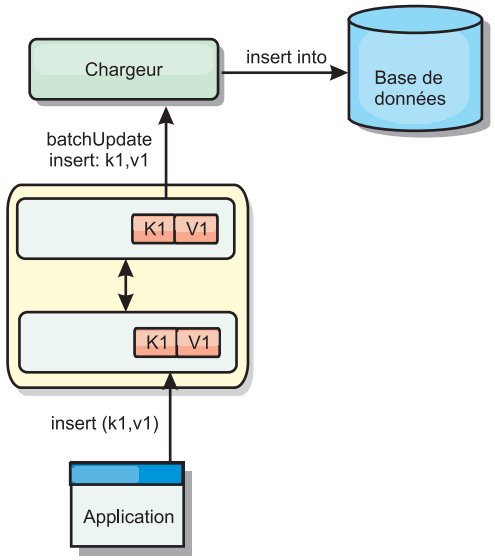


Figure 49. Mise en cache à écriture immédiate

Scénario de mise en cache en écriture différée

La synchronisation de la base de données peut être améliorée en écrivant les modifications de manière asynchrone. Cette opération est appelée mise en cache en

écriture différée. Les modifications, normalement écrites de manière synchrone dans le chargeur, sont mises en mémoire tampon dans eXtreme Scale et écrites dans la base de données à l'aide d'une unité d'exécution en arrière-plan. Les performances d'écriture sont considérablement améliorées, car l'opération de base de données est supprimée de la transaction client et les écritures de la base de données peuvent être comprimées.

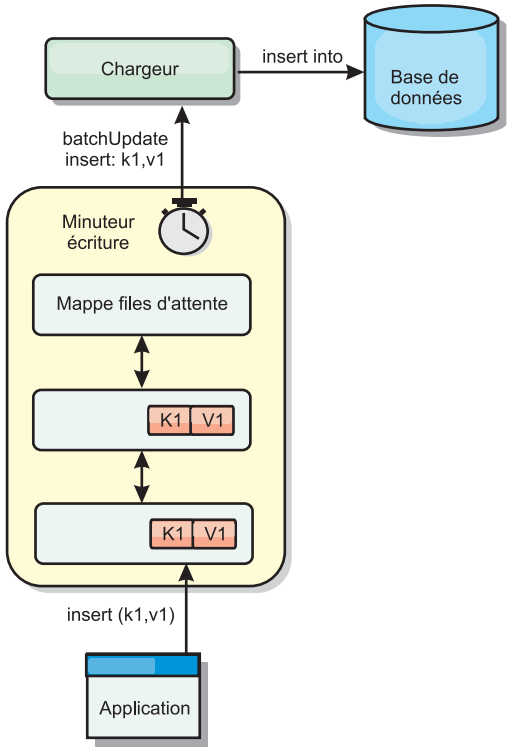


Figure 50. Mise en cache en écriture différée

Mise en cache en écriture différée

Vous pouvez utiliser la mise en cache en écriture différée pour réduire le temps système supplémentaire nécessaire lors de la mise à jour d'une base de données utilisée en tant que base de données dorsale.

Présentation de la mise en cache en écriture différée

La mise en cache en écriture différée met en file d'attente de manière asynchrone les mises à jour du plug-in Loader. Vous pouvez améliorer les performances en déconnectant les mises à jour, les insertions et les suppressions au sein d'une mappe, le temps système pour la mise à jour de la base de données dorsale. La mise à jour asynchrone est effectuée après un retard (de cinq minutes, par exemple) ou après un certain nombre d'entrées (1 000 entrées).

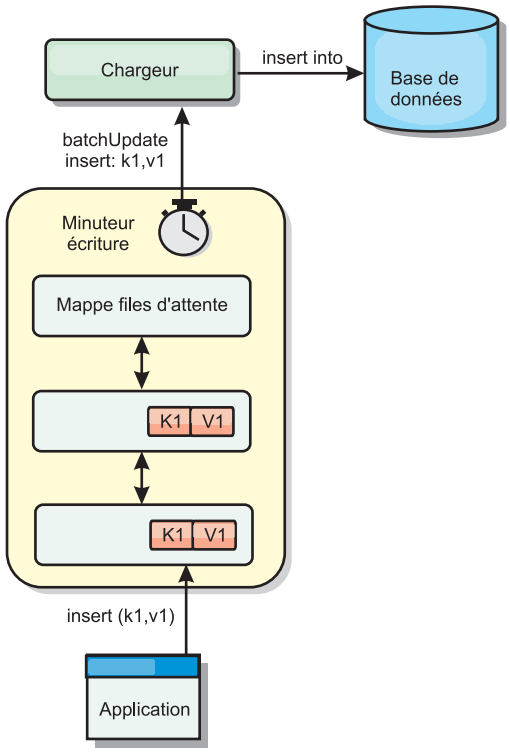


Figure 51. Mise en cache en écriture différée

La configuration à écriture différée sur une mappe de sauvegarde crée une unité d'exécution entre le Loader et la mappe. Le Loader délègue alors les demandes de données via l'unité d'exécution en fonction des paramètres de configuration de la méthode BackingMap.setWriteBehind. Lorsqu'une transaction eXtreme Scale insère, met à jour ou supprime une entrée dans une mappe, un objet LogElement est créé pour chacun de ces enregistrements. Ces éléments sont envoyés au Loader à écriture différée et mis en file d'attente dans une ObjectMap spéciale appelée mappe de files d'attente. Chaque mappe de sauvegarde pour laquelle le paramètre d'écriture différée est activé a ses propres mappes de files d'attente. L'unité d'exécution à écriture différée supprime périodiquement les données mises en file d'attente des mappes correspondantes et les insère dans le Loader dorsal.

Le chargeur à écriture différée envoie uniquement les types insertion, mise à jour et suppression des objets LogElement au chargeur réel. Tous les autres types, par exemple le type EVICT, sont ignorés.

La prise en charge de l'écriture différée *est* une extension du plug-in Loader, qui vous permet d'intégrer eXtreme Scale à la base de données. A ce sujet, vous pouvez consulter avec profit les explications Configuration des chargeurs JPA sur la configuration d 'un chargeur JPA.

Avantages

L'activation de l'écriture différée présente les avantages suivants :

- **Isolement en cas d'arrêt anormal de la base de données dorsale :** la mise en cache à écriture différée propose une couche d'isolement en cas d'arrêt anormal de la base de données dorsale. Les mises à jour sont alors placées dans la mappe de files d'attente. Les applications peuvent continuer à envoyer des transactions

vers eXtreme Scale. Lors de la reprise du système dorsal, les données contenues dans la mappe de files d'attente sont insérées dans celui-ci.

- **Réduction de la charge du système dorsal :** le chargeur à écriture différée fusionne les mises à jour en fonction des clés de façon qu'une seule mise à jour fusionnée par clé existe dans la mappe de files d'attente. Cette fusion diminue le nombre de mises à jour dans la base de données dorsale.
- **Amélioration des performances de la transaction :** la durée de chaque transaction eXtreme Scale est réduite car la transaction n'a plus à attendre que les données soient synchronisées avec le système dorsal.

Considérations liées à la conception d'applications

L'activation de l'écriture différée est une opération simple, mais la création d'une application devant utiliser l'écriture différée requiert une attention particulière. Sans écriture différée, la transaction ObjectGrid encadre la transaction dorsale. La transaction ObjectGrid démarre avant la transaction dorsale et se termine après celle-ci.

Lorsque la prise en charge de l'écriture différée est activée, la transaction ObjectGrid se termine avant le début de la transaction dorsale. La transaction ObjectGrid et la transaction dorsale sont dissociées.

Contraintes d'intégrité référentielle

Chaque mappe de sauvegarde configurée avec écriture différée dispose de sa propre unité d'exécution d'écriture différée pour envoyer les données vers le système dorsal. Les données mises à jour dans les différentes mappes d'une transaction ObjectGrid sont mises à jour dans le système dorsal via différentes transactions dorsales. Par exemple, la transaction T1 met à jour la clé key1 dans la mappe Map1 et la clé key2 dans la mappe Map2. La clé key1 mise à jour dans la mappe Map1 est actualisée vers le système dorsal dans une transaction expéditrice et la clé key2 actualisée dans la mappe Map2 l'est dans une autre transaction expéditrice ; cette mise à jour vers le dorsal est effectuée dans deux unités d'exécution différentes, toutes deux en écriture différée. Si les données stockées dans Map1 et Map2 ont des liens, tels que des contraintes de clé externe dans le système dorsal, les mises à jour sont susceptibles d'échouer.

Lors de la conception de contraintes d'intégrité référentielle dans votre base de données dorsale, vérifiez que les mises à jour désordonnées sont autorisées.

Verrouillage d'une mappe de files d'attente

Le comportement des transactions en matière de verrouillage constitue une autre différence notable. La grille d'objets prend en charge trois stratégies de verrouillage différentes : PESSIMISTIC, OPTIMISITIC et NONE. Les mappes de files d'attente à écriture différée utilisent la stratégie de verrouillage pessimiste, quelle que soit la stratégie de verrouillage configurée pour leur mappe de sauvegarde. Deux types différents d'opérations permettant d'acquérir un verrou sur la mappe de files d'attente existent :

- Lorsqu'une transaction ObjectGrid est validée ou qu'un vidage (de mappe ou de session) se produit, la transaction lit la clé de la mappe de files d'attente et place un verrou S sur la clé.
- Lorsqu'une transaction ObjectGrid est validée, elle tente de mettre à niveau le verrou S vers un verrou X sur la clé.

Ce comportement de mappe de files d'attente supplémentaire vous permet de voir quelques différences dans le comportement de verrouillage.

- Si la mappe de l'utilisateur est configurée comme stratégie de verrouillage pessimiste, la différence dans le comportement de verrouillage n'est pas grande. Chaque fois qu'un vidage ou qu'une validation est appelée, un verrou S est placé sur la même clé dans la mappe de files d'attente. Au moment de la validation, un verrou X est acquis pour la clé dans la mappe de l'utilisateur, mais également pour la clé dans la mappe de files d'attente.
- Si la mappe de l'utilisateur est configurée comme stratégie de verrouillage optimiste ou inexistante, la transaction utilisateur suit le modèle de la stratégie pessimiste. Chaque fois qu'un vidage ou qu'une validation est appelée, un verrou S est acquis sur la même clé dans la mappe de files d'attente. Au moment de la validation, un verrou X est acquis pour la clé dans la mappe de files d'attente utilisant la même transaction.

Nouvelles tentatives de transaction du chargeur

ObjectGrid ne prend pas en charge les transactions à 2 phases ou transactions XA. L'unité d'exécution à écriture différée supprime les enregistrements de la mappe de files d'attente et met à jour les enregistrements dans le système dorsal. En cas d'échec du serveur au milieu de la transaction, certaines mises à jour dorsales risquent d'être perdues.

Le chargeur à écriture différée tente automatiquement d'écrire à nouveau les transactions ayant échoué et envoie une LogSequence en attente de validation au système dorsal pour éviter toute perte de données. Pour que l'exécution de cette action soit possible, le chargeur doit être idempotent, ce qui signifie que lorsque Loader.batchUpdate(TxId, LogSequence) est appelé deux fois avec la même valeur, le résultat est le même que s'il était appelé une fois. Les implémentations du chargeur doivent implémenter l'interface RetryableLoader pour activer cette fonction. Pour plus de détails, consultez la documentation relative à l'API.

Echecs du chargeur

Le plug-in du chargeur (Loader) risque d'échouer lorsqu'il ne parvient pas à communiquer avec le dorsal de base de données. Cela peut se produire si le serveur de base de données ou la connexion réseau est arrêté(e). Le chargeur en écriture différée met en file d'attente les mises à jour et tente d'envoyer régulièrement les données modifiées au chargeur. Ce dernier doit signaler le problème de connectivité à l'environnement d'exécution ObjectGrid en générant une exception LoaderNotAvailableException.

L'implémentation du chargeur doit donc pouvoir distinguer un échec lié aux données d'une défaillance physique du chargeur. En cas d'échec lié aux données, une exception LoaderException ou OptimisticCollisionException doit être générée, alors qu'en cas de défaillance physique du chargeur, une exception LoaderNotAvailableException doit être générée. ObjectGrid gère ces deux exceptions de manière différente :

- Si une exception LoaderException est interceptée par le chargeur en écriture différée, celui-ci considère que l'échec est dû à une défaillance de données, telle qu'une erreur de clé en double. Le chargeur dégroupe la mise à jour et tente de ne mettre à jour qu'un enregistrement à la fois, afin d'isoler la défaillance de données. Si une exception {{LoaderException}} est à nouveau détectée lors de la mise à jour de l'enregistrement concerné, un enregistrement d'échec de la mise à jour est créé et consigné dans la mappe des mises à jour ayant échoué.

- Si une exception LoaderNotAvailableException est interceptée par le chargeur en écriture différée, celui-ci considère que l'échec est dû à l'impossibilité de se connecter à la base de données, par exemple, lorsque la base de données dorsale est inactive, lorsque la connexion à une base de données est indisponible ou lorsque le réseau est inactif. Le chargeur attend 15 secondes, puis tente à nouveau la mise à jour par lots de la base de données.

L'erreur courante est d'émettre une exception LoaderException à la place d'une exception LoaderNotAvailableException. Tous les enregistrements du chargeur à écriture différée deviennent alors des enregistrements d'échec de la mise à jour, ce qui réduit à néant l'objectif de l'isolement en cas d'arrêt anormal du système dorsal.

Remarques sur les performances

En supprimant de la transaction la mise à jour du chargeur, la mise en cache en écriture différée augmente le temps de réponse. Elle augmente également la capacité de traitement de la base de données, car les mises à jour de base de données sont combinées. Il est important de comprendre le temps système supplémentaire généré par l'unité d'exécution à écriture différée, qui permet de retirer les données de la mappe de files d'attente et de les insérer dans le chargeur.

Le temps de mise à jour maximal et le nombre de mises à jour maximal doivent être ajustés en fonction de l'environnement et des types d'utilisation prévus. Si la valeur du temps ou du nombre de mises à jour maximal est trop petite, le temps système de l'unité d'exécution d'écriture différée peut dépasser les avantages tirés. Définir une valeur élevée pour ces deux paramètres peut également augmenter l'utilisation de la mémoire pour la mise en file d'attente des données et retarder le moment de péremption des enregistrements de bases de données.

Pour des performances optimales, réglez les paramètres d'écriture différée en fonction des facteurs suivants :

- ratio des transactions de lecture et d'écriture
- fréquence de mise à jour d'enregistrements identiques
- temps d'attente pour la mise à jour de la base de données

Chargeurs

Avec un plug-in Loader, une mappe de grille de données peut se comporter comme un cache pour les données généralement conservées dans un magasin persistant sur le même système ou un autre système. Généralement, une base de données ou un système de fichiers est utilisé comme stockage de persistance. Une machine virtuelle Java (JVM) peut également être utilisée comme source des données, ce qui permet de créer des caches basés sur un concentrateur à l'aide d'eXtreme Scale. Un chargeur peut lire et écrire des données vers un stockage persistant ou à partir de celui-ci.

Présentation

Les chargeurs sont des plug-in de mappe de sauvegarde appelés lorsque des modifications sont apportées à la mappe de sauvegarde ou lorsque cette dernière est dans l'impossibilité de répondre à une demande de données (absence dans le cache). Le chargeur est appelé lorsque le cache ne peut pas satisfaire une demande de clé, offrant ainsi une fonction de lecture et un remplissage laborieux du cache. Un chargeur permet également les mises à jour de la base de données lorsque les valeurs du cache viennent à changer. Toutes les modifications dans une transaction

sont regroupées pour réduire le nombre d'interactions de base de données. Un plug-in TransactionCallback est utilisé conjointement avec le chargeur pour déclencher la démarcation de la transaction principale. L'utilisation de ce plug-in est importante lorsque plusieurs mappes sont incluses dans une seule transaction ou lorsque les données de transaction sont vidées dans le cache sans validation.

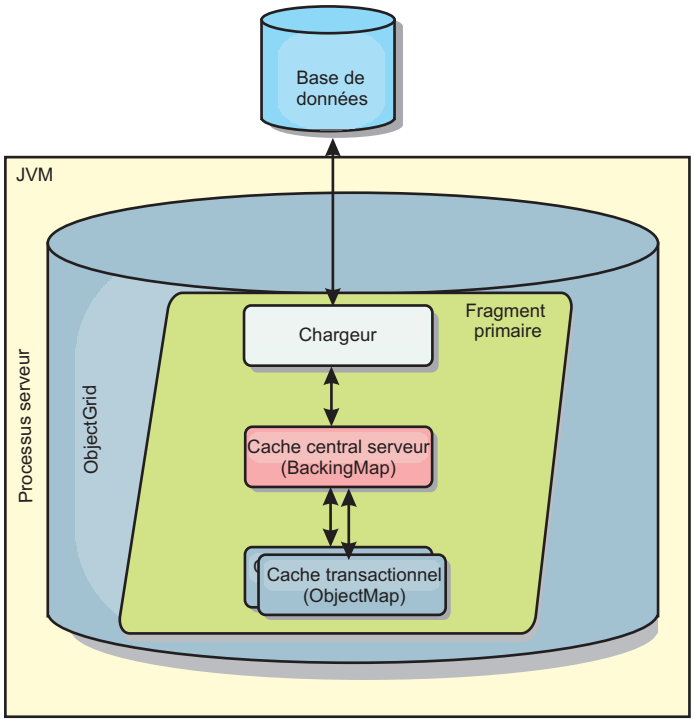


Figure 52. Chargeur

Le chargeur peut donc utiliser les mises à jour sur-qualifiées pour éviter le verrouillage intempestif de la base de données. En stockant un attribut de version dans la valeur du cache, le chargeur peut distinguer l'image de la valeur avant et après la mise à jour dans le cache. Cette valeur peut ensuite être utilisée lors de la mise à jour de la base de données ou du programme d'arrière plan pour vérifier que les données n'ont pas été mises à jour. Un chargeur peut également être configuré pour précharger la grille de données lorsqu'elle démarre. Lorsqu'elle est partitionnée, une instance de chargeur est associée à chaque partition. Si la mappe de la société comporte dix partitions, il existe dix instances de chargeur, une pour chaque partition principale. Lorsque le fragment primaire de la mappe est activé, la méthode preloadMap du chargeur est appelée de manière synchrone ou asynchrone, ce qui déclenche le chargement automatique de la partition de la mappe avec les données du programme d'arrière plan. Lorsqu'il est appelé de manière synchrone, toutes les transactions client sont bloquées, ce qui empêche tout accès incohérent à la grille de données. Sinon, un préchargeur client peut être utilisé pour charger l'intégralité de la grille de données.

Deux chargeurs pré-intégrés peuvent simplifier considérablement l'intégration aux dorsaux de bases de données relationnelles. Les chargeurs JPA utilisent les fonctions du mappage objet-relationnel(ORM) des implémentations OpenJPA et Hibernate des spécifications JPA (Java Persistence API). Pour plus d'informations, voir «Chargeurs JPA», à la page 67.

Si vous utilisez des chargeurs dans une configuration à plusieurs centre de données, vous devez étudier la façon dont les données de révision et la cohérence de la mémoire cache est conservée entre les grilles de données. Pour plus d'informations, voir «Remarques sur les chargeurs dans une topologie multimaître» , à la page 172.

Configuration de chargeur

Pour ajouter un chargeur à la configuration BackingMap, vous pouvez utiliser la configuration à l'aide d'un programme ou la configuration XML. Un chargeur a la relation suivante avec une mappe de sauvegarde.

- Une mappe de sauvegarde peut avoir un seul chargeur.
- Une mappe de sauvegarde client (cache local) ne peut pas avoir de chargeur.
- Une définition de chargeur peut être appliquée à plusieurs mappes de sauvegarde, mais chaque mappe de sauvegarde dispose de sa propre instance de chargeur.

Préchargement et préremplissage des données

Dans la plupart des scénarios qui utilise un chargeur, vous pouvez préparer la grille de données en y préchargeant ses données.

Lorsque vous utilisez la grille de données comme un cache complet, elle doit contenir toutes les données et elle doit être chargée pour que les clients puissent s'y connecter. Lorsque vous utilisez un cache partiel, vous pouvez préparer le cache avec des données pour que les clients puissent avoir accès immédiatement à ces données dès qu'ils se connectent.

Il existe deux approches pour pré-charger des données dans la grille de données ; vous pouvez utiliser un plug-in Loader ou un chargeur client, comme décrit dans les sections suivantes.

Plug-in Loader

Le plug-in Loader est associé à chaque mappe et chargé de synchroniser un fragment primaire de partition avec la base de données. La méthode preloadMap du plug-in Loader est invoquée automatiquement lors de l'activation d'un fragment, Par exemple, vous disposez de 100 partitions, il existe 100 instances Loader, chacune chargeant les données de sa partition. En cas d'exécution synchrone, tous les clients sont bloqués jusqu'à la fin du préchargement.

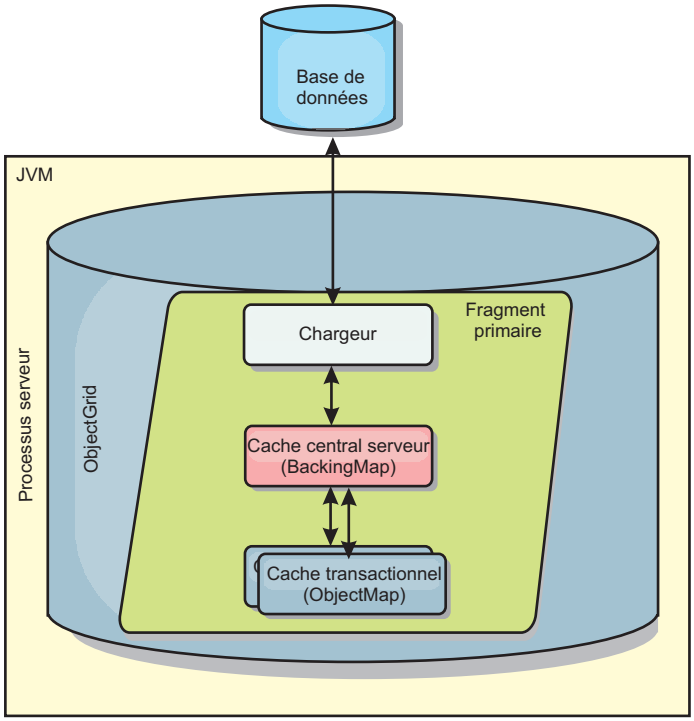


Figure 53. Plug-in Loader

Loader client

Un loader client est un pattern d'utilisation d'un ou plusieurs clients pour charger les données dans la grille. L'utilisation de plusieurs clients pour charger les données de la grille peut s'avérer efficace lorsque le schéma de partition n'est pas stocké dans la base de données. Vous pouvez appeler des chargeurs de client manuellement ou automatiquement lorsque la grille de données démarre. Ces chargeurs peuvent éventuellement utiliser StateManager pour faire passer la grille de données en mode de préchargement pour que les clients ne puissent pas accéder à la grille lorsqu'elle précharge les données. WebSphere eXtreme Scale contient un chargeur JPA (Java Persistence API) que vous pouvez utiliser pour charger automatiquement la grille de données avec le fournisseur JPA OpenJPA ou Hibernate. Pour plus d'informations sur les fournisseurs de cache, voir «Plug-in de cache niveau 2 (L2) JPA», à la page 26.

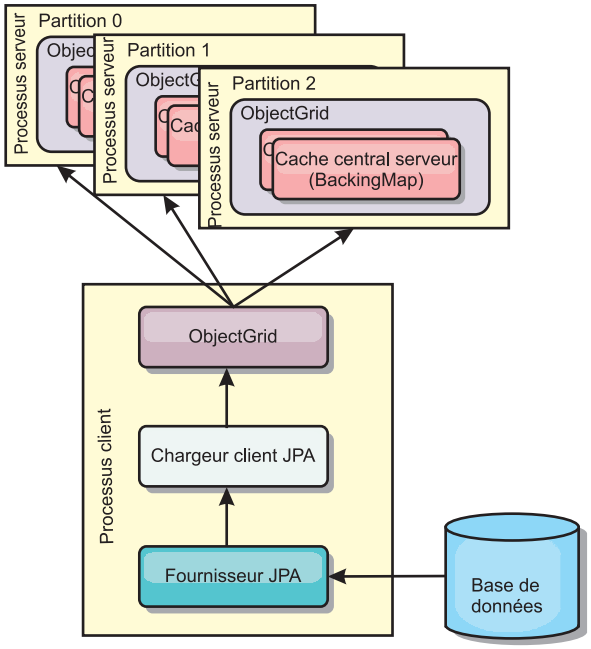


Figure 54. Loader client

Méthodes de synchronisation de base de données

Lorsque WebSphere eXtreme Scale est utilisé en tant que cache, les applications doivent être écrites de sorte qu'elles tolèrent les données périmées si la base de données peut être mise à jour de manière indépendante par rapport à une transaction eXtreme Scale. En tant qu'espace de traitement de base de données en mémoire synchronisé, eXtreme Scale permet d'assurer la mise à jour du cache de plusieurs manières.

Méthodes de synchronisation de base de données

Actualisation régulière

Le cache peut être régulièrement invalidé ou mis à jour de manière automatique à l'aide du programme de mise à jour temporelle de base de données JPA (Java Persistence API).Le programme de mise à jour interroge régulièrement la base de données à l'aide d'un fournisseur JPA, afin de rechercher des mises à jour ou des insertions survenues depuis la mise à jour précédente. Tous les changements détectés sont automatiquement invalidés ou mis à jour lorsqu'ils sont utilisés avec un cache incomplet. S'ils sont utilisés avec un cache complet, les entrées peuvent être détectées et insérées dans le cache. Les entrées ne sont jamais supprimées du cache.

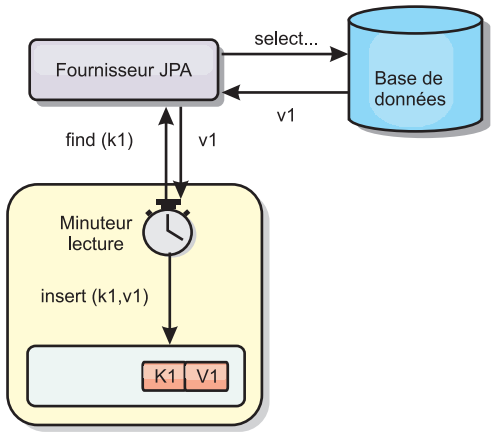


Figure 55. Actualisation régulière

Suppression

Les caches incomplets peuvent utiliser les stratégies de suppression pour supprimer automatiquement les données du cache sans que cela n'affecte la base de données. eXtreme Scale inclut trois stratégies : durée de vie, utilisation la moins récente et utilisation la moins fréquente. Si l'option de suppression en fonction de la mémoire est activée, ces trois stratégies suppriment les données de manière plus agressive à mesure que la mémoire est limitée.

Invalidation en fonction d'événements

Il est possible d'invalider les caches partiels et complets à l'aide d'un générateur d'événements comme JMS (Java Message Service). L'invalidation par le biais de JMS peut être associée de manière manuelle à tout processus qui met à jour le dorsal à l'aide d'un déclencheur de base de données. eXtreme Scale contient un plug-in JMS ObjectGridEventListener qui informe les clients des éventuelles modifications du cache du serveur. Cette procédure peut réduire la durée d'accès du client aux données périmées.

Invalidation par programme

Les API eXtreme Scale permettent l'interaction manuelle du cache local et du cache serveur à l'aide des méthodes des API Session.beginNoWriteThrough(), ObjectMap.invalidate() et EntityManager.invalidate(). Si un processus client ou serveur n'a plus besoin d'une partie des données, les méthodes d'invalidation peuvent être utilisées pour supprimer les données du serveur local ou du serveur cache. La méthode beginNoWriteThrough applique une opération ObjectMap ou EntityManager au cache local sans appeler le programme de chargement. Si l'opération est appelée à partir d'un client, elle s'applique uniquement au cache local (le programme de chargement distant n'est pas appelé). Si elle est appelée sur le serveur, l'opération s'applique uniquement au cache central du serveur sans appeler le programme de chargement.

L'invalidation des données

Pour supprimer les données de la mémoire cache d'évolution, vous pouvez utiliser un mécanisme d'invalidation basé sur les événements ou à l'aide d'un programme.

Invalidation basée sur les événements

Il est possible d'invalider les caches incomplets et complets à l'aide d'un générateur d'événements tel que Java Message Service (JMS). L'invalidation par le biais de JMS peut être associée de manière manuelle à tout processus qui met à jour le dorsal à l'aide d'un déclencheur de base de données. Un plug-in JMS ObjectGridEventListener est fourni dans eXtreme Scale pour permettre aux clients d'être informés des modifications dans le cache du serveur. Ce type de notification peut réduire la durée d'accès du client aux données obsolètes.

L'invalidation basée sur les événements est composée normalement des trois composants suivants.

- **File d'attente des événements :** une file d'attente d'événements stocke les événements de modification des données. Il peut s'agir d'une file d'attente JMS, d'une base de données, d'une file d'attente interne premier entré premier sorti ou de tout autre événement dans la mesure où elle peut gérer les événements de modification des données.
- **Publicateur d'événements :** un publicateur d'événements publie les événements de modification de données dans la file d'attente d'événements. Une publicateur d'événements est généralement une application que vous créez ou une implémentation de plug-in eXtreme Scale. Il sait quand les données ont été modifiées ou il modifie les données lui-même. Lorsqu'une transaction est validée, les événements sont générés pour les données modifiées et le publicateur d'événements publie ces événements dans la file d'attente d'événements.
- **Consommateur d'événements :** un consommateur d'événements consomme les événements de modification de données. Le consommateur d'événements est généralement une application permettant de vérifier la mise à jour des données de la grille cible avec les dernières modifications apportées aux autres grilles. Il interagit avec la file d'attente d'événements pour récupérer les dernières données et applique les modifications apportées aux données dans la grille cible. Les consommateurs d'événements peuvent utiliser les API eXtreme Scale pour invalider les données obsolètes ou mettre à jour la grille avec les dernières données.

Par exemple, JMSObjectGridEventListener comporte une option pour un modèle client-serveur dans lequel la file d'attente d'événements est une destination JMS désignée. Tous les processus serveur sont des publicateurs d'événements. Lorsqu'une transaction est validée, le serveur récupère les modifications apportées aux données et les publie à la destination JMS désignée. Tous les processus client sont des consommateurs d'événements. Ils reçoivent les modifications apportées aux données de la destination JMS désignée et appliquent les modifications au cache local du client.

Pour plus d'informations, consultez la rubrique relative au mécanisme d'invalidation de client dans le *Guide de l'administration* .

Invalidation par programme

Les API WebSphere eXtreme Scale autorise l'interaction manuelle du cache local et du cache serveur à l'aide des méthodes Session.beginNoWriteThrough(), ObjectMap.invalidate() et EntityManager.invalidate(). Si un processus client ou serveur n'a plus besoin d'une partie des données, les méthodes invalidate permettent de supprimer des données d'un cache local ou de serveur. La méthode beginNoWriteThrough applique toutes les opérations ObjectMap ou EntityManager

au cache local sans appeler le chargeur. Si l'opération est appelée à partir d'un client, elle s'applique uniquement au cache local (le programme de chargement distant n'est pas appelé). Si elle est appelée sur le serveur, l'opération s'applique uniquement au cache central du serveur sans appeler le programme de chargement.

Vous pouvez utiliser l'invalidation par programme à l'aide d'autres techniques pour déterminer quand il convient d'invalider les données. Par exemple cette méthode d'invalidation utilise des mécanismes d'invalidation basée sur les événements pour recevoir les événements de modification de données, puis utilise les API pour invalider les données obsolètes.

Indexation

Utilisez le plug-in MapIndexPlugin pour générer un ou plusieurs index dans une mappe BackingMap pour prendre en charge l'accès aux données ne correspondant pas à une clé.

Types d'indexation et configuration d'index

L'indexation est représentée par le plug-in MapIndexPlugin ou Index, en bref. Index est un plug-in BackingMap. Une mappe de sauvegarde peut avoir plusieurs index configurés, dès lors que chacun d'entre eux respecte les règles de configuration d'index.

Vous pouvez utiliser l'indexations pour générer une ou plusieurs index dans une mappe BackingMap. Un index se construit à partir d'un attribut ou d'une liste des attributs d'un objet de la mappe. L'indexation permet aux applications de trouver plus rapidement certains objets. Grâce à elle, en effet, les applications peuvent trouver les objets dont les attributs indexés ont une certaine valeur ou se situent dans une plage de valeurs.

Deux types d'indexation sont possibles : statiques et dynamiques. L'indexation statique oblige à configurer le plug-in d'indexation index dans la mappe de sauvegarde avant d'initialiser l'instance ObjectGrid. Comme pour la mappe de sauvegarde, cela peut se faire par programmation ou via XML. L'indexation statique commence à générer l'index pendant l'initialisation de la grille d'objets. L'index est synchrone en permanence avec la mappe de sauvegarde et il est prêt à être utilisé. Après que l'indexation statique a démarré, la maintenance de l'index fait partie de la gestion des transactions par eXtreme Scale. Lorsque les transactions valident leurs modifications, ces dernières actualisent également l'index statique et les modifications apportées à l'index sont annulées en cas d'annulation de la transaction.

L'indexation dynamique permet de créer un index dans une mappe de sauvegarde avant ou après l'initialisation de l'instance ObjectGrid qui contient cette mappe. Les applications contrôlent le cycle de vie de l'indexation dynamique, ce qui permet de supprimer un index dynamique devenu inutile. Lorsqu'une application crée un index dynamique, cet index n'est pas forcément utilisable immédiatement en raison du temps que met à s'effectuer la génération complète de l'index. Comme la durée dépend de la quantité de données indexées, l'interface DynamicIndexCallback est fournie pour les applications qui souhaitent recevoir des notifications lorsque se produisent certains événements l'indexation, à savoir les événements ready, error et destroy. Les applications peuvent implémenter cette interface de rappel et s'enregistrer auprès de l'indexation dynamique.

Si un plug-in d'indexation est configuré pour une mappe de sauvegarde, il est possible d'obtenir de la mappe d'objet correspondante l'objet proxy de l'index. L'appel de la méthode getIndex dans la mappe et la transmission du nom du plug-in Index renvoie l'objet proxy de l'index. L'objet proxy doit être transtypé vers l'interface d'indexation de l'application utilisée, MapIndex, MapRangeIndex ou une interface d'indexation personnalisée, par exemple. Une fois l'objet proxy obtenu, l'on peut utiliser les méthodes définies dans l'interface d'indexation de l'application afin de trouver des objets mis en cache.

La liste qui suit récapitule la procédure à appliquer pour procéder à l'indexation :

- ajout d'index statiques ou dynamiques dans la mappe de sauvegarde
- obtention d'un objet proxy d'index grâce à la méthode getIndex de la mappe d'objet
- transtypage de l'objet proxy vers l'interface d'indexation de l'application utilisée (MapIndex, MapRangeIndex ou une interface d'indexation personnalisée, par exemple)
- utilisation des méthodes qui sont définies dans l'interface d'indexation de l'application pour rechercher les objets mis en cache

La classe HashIndex est l'implémentation du plug-in d'indexation pré-intégré capable de prendre en charge les deux interfaces pré-intégrées d'API d'indexation : MapIndex et MapRangeIndex. Vous pouvez également créer vos propres index. Vous pouvez ajouter HashIndex à la BackingMap en tant qu'index statique ou dynamique, obtenir un objet proxy d'index MapIndex ou MapRangeIndex et utiliser cet objet proxy pour chercher des objets mis en cache.

Index par défaut

Si vous souhaitez effectuer une itération dans les clés d'une mappe locale, vous pouvez utiliser l'index par défaut. Cet index ne requiert pas de configuration, mais elle doit être utilisée sur le fragment en utilisant un agent ou une instance ObjectGrid extraite de la méthode ShardEvents.shardActivated(ObjectGrid shard).

Indexation et qualité des données obtenues par une requête d'index

Il faut bien avoir présent à l'esprit que les méthodes de requêtes sur les index ne représentent qu'un cliché des données à un instant t. Les entrées de données ne sont pas verrouillées après l'envoi à l'application des résultats de la requête. L'application doit être consciente que les données peuvent très bien être actualisées après lui avoir été retournées. Supposons, par exemple, que l'application obtienne la clé d'un objet mis en cache grâce à la méthode findAll de MapIndex. Cet objet key retourné est associé dans le cache à une entrée de données. L'application doit être capable d'exécuter la méthode get sur la mappe d'objet pour trouver un objet à partir de l'objet key. Si une autre transaction supprime du cache l'objet données juste avant l'appel à la méthode get, le résultat qui sera retourné sera null.

Points à prendre en considération à propos des performances de l'indexation

L'un des objectifs primordiaux de l'indexation est d'améliorer les performances globales de la mappe de sauvegarde. Une utilisation incorrecte de l'indexation peut compromettre les performances de l'application. Avant d'utiliser l'indexation, les facteurs suivants sont à prendre en considération :

- **Le nombre de transactions simultanées en écriture :** l'indexation peut se produire chaque fois qu'une transaction écrit des données dans une mappe de

sauvegarde. Les performances se dégradent si un grand nombre de transactions écrivent en même temps des données dans la mappe au moment où une application lance des requêtes sur l'index.

- **La taille des résultats retournés par une requête :** les performances de la requête déclinent d'autant plus que la taille de ses résultats augmente. Les performances tendent à se dégrader lorsque la taille des résultats atteint 15 % ou plus de la mappe de sauvegarde.
- **Le nombre d'index générés sur la même mappe de sauvegarde :** chaque index consomme des ressources système. Les performances diminuent au fur et à mesure que le nombre d'index augmente sur la mappe de sauvegarde.

Cela dit, l'indexation peut augmenter considérablement les performances des mappes de sauvegarde. C'est particulièrement vrai lorsque la mappe de sauvegarde comporte surtout des opérations de lecture. Les résultats des requêtes représentent alors un faible pourcentage des entrées de la mappe et seul un petit nombre d'index sont générés sur la mappe.

Planification de plusieurs topologies de centre de données

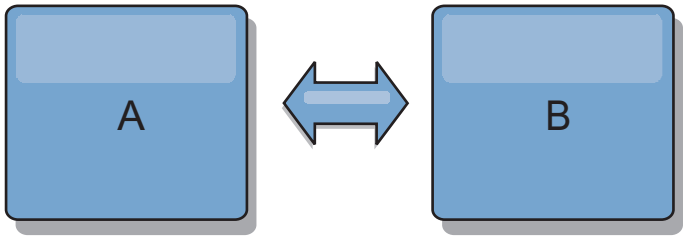
En utilisant la réplication asynchrone multimaître, au moins deux grilles de données peuvent devenir des copies exactes de l'une de l'autre. Chaque grille de données est hébergée dans un domaine de services de catalogue indépendant, avec ses propres de service de catalogue, serveurs de conteneur et un nom unique. Avec la réplication asynchrone multimaître, vous pouvez utiliser des liaisons pour connecter un ensemble de domaine de services de catalogue. Les domaine de services de catalogue sont ensuite synchronisés en utilisant la réplication via ces liaisons. Vous pouvez construire quasiment n'importe quelle topologie via la définition de liaisons entre les domaine de services de catalogue.

Topologies pour la réplication multimaître

Vous disposez de plusieurs options pour choisir la topologie de votre déploiement qui intègre la réplication multimaître.

Liaisons connectant des domaine de services de catalogue

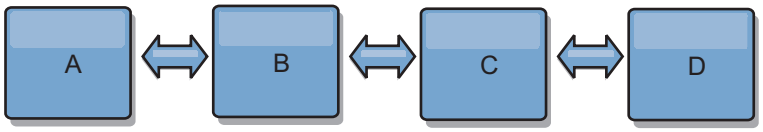
Une infrastructure de grilles de données de réplication est un graphique de domaine de services de catalogue interconnectés avec des liaisons bidirectionnelles. Avec une liaison, deux domaine de services de catalogue peuvent communiquer les modifications de données. Par exemple, la topologie la plus simple est une paire de domaine de services de catalogue avec une liaison unique entre eux. Les domaine de services de catalogue sont nommés par ordre alphabétique: A, B, C, etc., à partir de la gauche. Une liaison peut traverser un réseau WAN (wide area network) pour couvrir une grande distance. Même si la liaison est interrompue, vous pouvez toujours modifier les données dans l'un des domaine de services de catalogue. La topologie rapproche les modifications quand la liaison reconnecte les domaine de services de catalogue. Les liens tentent automatiquement de se reconnecter si la connexion réseau est interrompue.



Après avoir établi les liaisons, eXtreme Scale tente d'abord de rendre chaque domaine de services de catalogue identique. Ensuite, eXtreme Scale tente de maintenir identiques les conditions à mesure que des modifications se produisent dans un domaine de services de catalogue. L'objectif vise à faire de chaque domaine de services de catalogue le miroir exact d'un autre domaine de services de catalogue connecté par les liaisons. Les liaisons de réplication entre les domaine de services de catalogue permettent copier une modification effectuée dans un domaine vers les autres domaines.

Topologies linéaires

Même s'il s'agit d'un déploiement simple, une topologie linéaire montre certaines qualités des liaisons. Tout d'abord, il n'est pas nécessaire qu'un domaine de services de catalogue soit connecté directement à tous les autres domaine de services de catalogue pour pouvoir recevoir les modifications. Le domaine B extrait les modifications du domaine A. Le domaine C reçoit les modifications du domaine A via le domaine B, lequel connecte les domaines A et C. De même, le domaine D reçoit les modifications des autres domaines via le domaine C. De ce fait, la charge de la répartition des modifications est distribuée et elle n'incombe plus à la seule source de ces modifications.



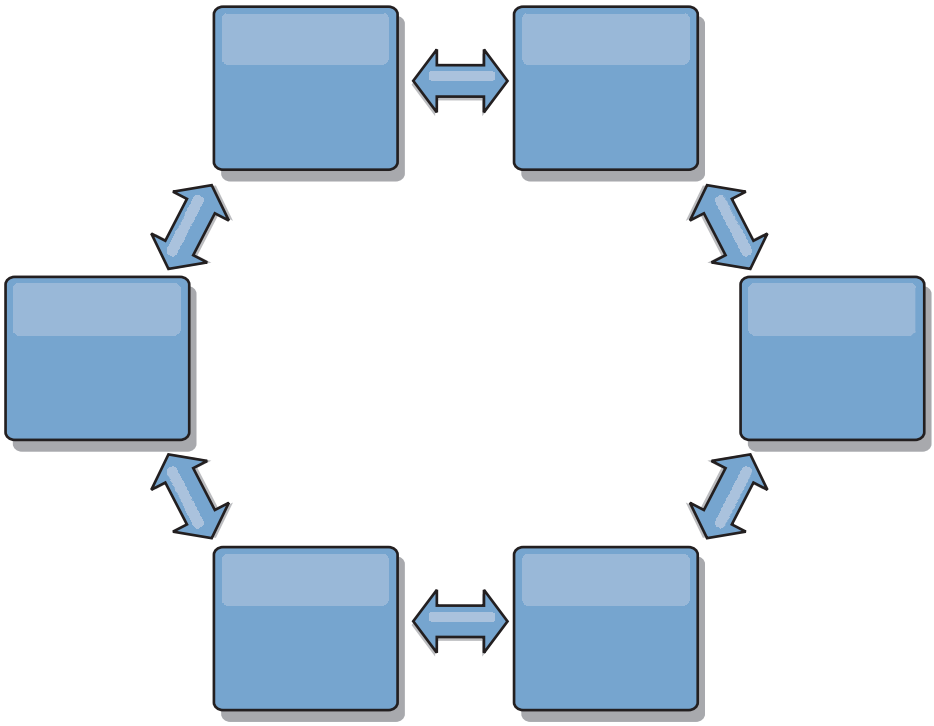
Notez que si le domaine C est défaillant, les actions suivantes se produisent :

1. Le domaine D serait orphelin jusqu'au redémarrage du domaine C.
2. Le domaine C doit se synchroniser avec le domaine B, lequel est une copie du domaine A.
3. Le domaine D utilise le domaine C pour se synchroniser avec les modifications des domaines A et B. Ces modifications se sont produites initialement lorsque le domaine D étaient orphelin (lorsque le domaine C était arrêté).

Enfin, les domaines A, B, C et D, sont de nouveau tous identiques.

Topologies en anneau

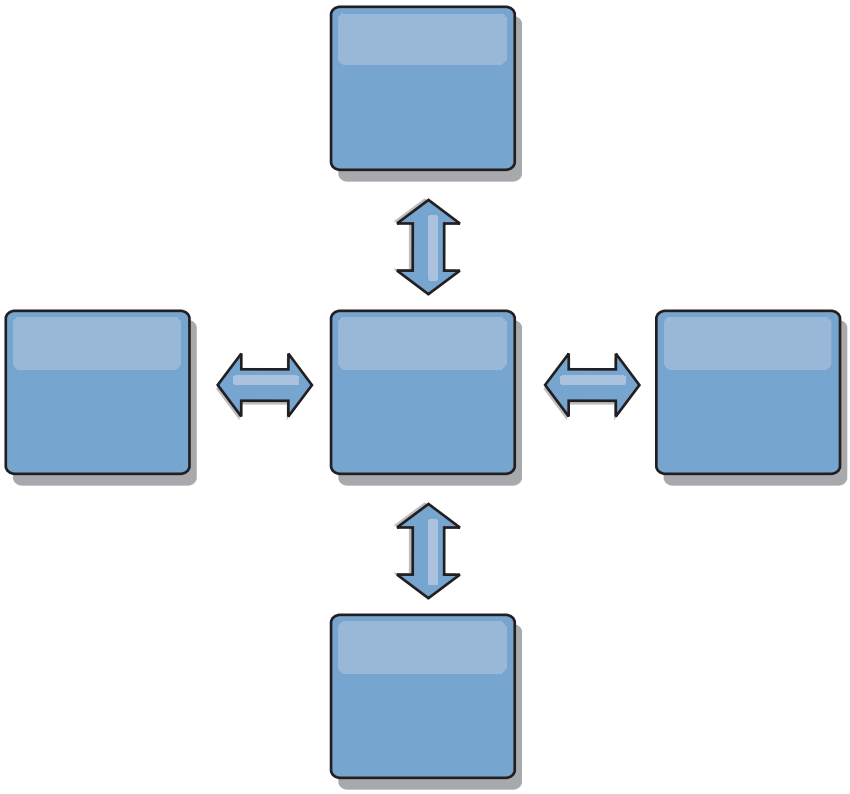
Les topologies en anneau sont un exemple de topologie encore plus résilientes. Lorsqu'un domaine de services de catalogue ou une liaison unique tombe en panne, les domaine de services de catalogue restants peuvent encore obtenir des modifications. Les domaine de services de catalogue parcourent l'anneau en s'éloignant de la défaillance. Chaque domaine de serveur de catalogue a au maximum deux liens vers d'autres domaine de services de catalogue, quelle que soit la taille de la topologie en anneau. Le délai de propagation des modifications peut être important. Les modifications d'un domaine de services de catalogue peuvent devoir traverser plusieurs liaisons pour que tous les domaine de services de catalogue aient les modifications. Une topologie linéaire a la même caractéristique.



Vous pouvez également déployer une topologie en anneau plus sophistiquée, avec un domaine de services de catalogue racine au centre de l'anneau. Le domaine de services de catalogue racine fait office de point central de réconciliation. Les autres domaine de services de catalogue font office de points distants de réconciliation pour les modifications se produisant dans le domaine de services de catalogue racine. Le domaine de services de catalogue racine peut arbitrer les modifications entre les domaine de services de catalogue. Si une topologie en anneau contient plusieurs anneaux autour d'un domaine de services de catalogue racine, le domaine ne peut pas arbitrer les modifications dans la partie interne de l'anneau. Toutefois, les résultats de l'arbitrage sont propagés dans les domaine de services de catalogue des autres anneaux.

Topologies en étoile

Avec une topologie en étoile, les modifications parcourent un domaine de services de catalogue en étoile. Etant donné que le concentrateur est le seul domaine de services de catalogue intermédiaire spécifié, les topologies en étoile ont une latence inférieure. Le domaine du concentrateur est connecté à chaque branche de domaine via une liaison. Le concentrateur distribue les modifications entre les domaine de services de catalogue. Il fait office de point de rapprochement pour les collisions. Dans un environnement soumis à une fréquence élevée de modifications, le concentrateur peut avoir besoin de s'exécuter sur plus de matériels que les branches pour rester synchronisé. WebSphere eXtreme Scale est conçu pour évoluer de manière linéaire, ce qui signifie que l'on peut, si nécessaire, étoffer le concentrateur sans difficultés. Toutefois, si le concentrateur tombe en panne, les modifications ne sont pas distribuées jusqu'à ce qu'il redémarre. Toutes les modifications sur les branches du sous-domaine de services de catalogue seront réparties après la reconnexion du concentrateur.



Vous pouvez également utiliser une stratégie avec les clients intégralement répliqués, une variante de la topologie qui utilise une paire de serveurs eXtreme Scale s'exécutant comme concentrateur. Chaque client crée une grille de données à conteneur unique autonome avec un catalogue dans la machine virtuelle Java client. Un client utilise sa grille de données pour se connecter au catalogue du concentrateur. Cette connexion provoque la synchronisation du client avec le concentrateur dès que le client obtient une connexion au concentrateur.

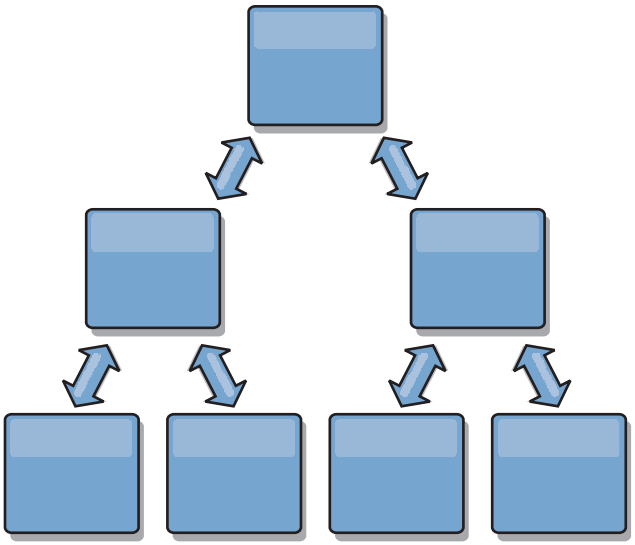
Toutes les modifications effectuées par le client sont locales pour le client et elles sont répliquées vers le concentrateur de manière asynchrone. Le concentrateur joue le rôle de domaine d'arbitrage, répartissant les modifications à tous les clients connectés. La topologie de clients intégralement répliqués fournit un cache L2 fiable pour un associateur relationnel d'objets comme OpenJPA. Les modifications sont réparties rapidement via le concentrateur entre les machines virtuelles client. Si la taille du cache peut être contenue dans le segment de mémoire disponible, la topologie est une architecture fiable pour ce style de cache L2.

Si nécessaire, utilisez plusieurs partitions pour échelonner le domaine concentrateur sur plusieurs machines virtuelles Java. Etant donné que toutes les données doivent toujours tenir sur une seule machine virtuelle Java client, plusieurs partitions augmentent la capacité du concentrateur à répartir et à arbitrer les modifications. Cependant, plusieurs partitions ne changent pas la capacité d'un domaine unique.

Topologies en arbre

Vous pouvez également utiliser un arbre dirigé acyclique. Un arbre acyclique n'a pas de cycles ou de boucles, et une configuration dirigée limite les liaisons aux parents et enfants existants uniquement. Cette configuration peut être utile pour les topologies disposant d'un grand nombre de domaine de services de catalogue, et il

n'est pas pratique d'avoir un concentrateur central connecté à chaque branche. Ce type de topologie peut également être utile lorsque vous devez ajouter des domaine de services de catalogue enfant sans mettre à jour le domaine de services de catalogue racine.



Une topologie en arbre peut toujours avoir un point central de rapprochement dans le domaine de services de catalogue racine. Le deuxième niveau peut toujours fonctionner en tant que point de rapprochement distant pour les modifications se produisant dans le domaine de services de catalogue en dessous. Le domaine de services de catalogue racine peut arbitrer les modifications entre les domaines de services de catalogue sur le deuxième niveau uniquement. Vous pouvez également utiliser des arbres n-aire ayant chacun n enfants à chaque niveau. Chaque domaine de services de catalogue se connecte à *n* liaisons.

Clients intégralement répliqués

Cette variante de la topologie implique une paire de serveurs eXtreme Scale s'exécutant comme concentrateur. Chaque client crée une grille de données à conteneur unique autonome avec un catalogue dans la machine virtuelle Java client. Un client utilise sa grille de données pour se connecter au catalogue du concentrateur, ce qui provoque la synchronisation du client avec le concentrateur dès que le client obtient une connexion au concentrateur.

Toutes les modifications effectuées par le client sont locales pour le client et elles sont répliquées vers le concentrateur de manière asynchrone. Le concentrateur joue le rôle de domaine d'arbitrage, répartissant les modifications à tous les clients connectés. La topologie de clients intégralement répliqués fournit un bon cache de niveau 2 pour un associateur relationnel d'objets comme OpenJPA. Les modifications sont réparties rapidement via le concentrateur entre les machines virtuelles client. Tant que la taille du cache peut être contenue dans l'espace de segment mémoire disponible des clients, cette topologie est une architecture tout à fait indiquée pour ce style de cache de niveau 2.

Si nécessaire, utilisez plusieurs partitions pour échelonner le domaine concentrateur sur plusieurs machines virtuelles Java. Toutes les données devant tenir sur une seule machine virtuelle Java, l'utilisation de partitions multiples augmente la capacité du concentrateur à répartir et à arbitrer les modifications, mais elle ne change pas la capacité d'un domaine unique.

Considérations de configuration pour les topologies multimaîtres

Tenez compte des points suivants lorsque vous déterminez l'opportunité et la manière d'utiliser des topologies de réplication multimaîtres.

- **Exigences de groupe de mappes**

  Les groupes de mappes doivent avoir les caractéristiques suivantes pour pouvoir répliquer les modifications dans les liaisons d'un domaine de services de catalogue :

  – Le nom ObjectGrid et le nom de groupe de mappes dans un domaine de services de catalogue doivent correspondre au nom ObjectGrid et au nom de groupe de mappes d'autres domaine de services de catalogue. Par exemple, ObjectGrid "og1" et le groupe de mappes "ms1" doivent être configurés dans les domaine de services de catalogue A et B pour pouvoir répliquer les données dans la mappe entre les domaine de services de catalogue.

  – Est une grille de données `FIXED_PARTITION`. Les grilles de données `PER_CONTAINER` ne peuvent pas être répliquées.

  – A le même nombre de partitions dans chaque domaine de services de catalogue. Le groupe de mappes peut ou peut ne pas avoir le même nombre et le même type de répliques.

  – A les mêmes types de données répliquées dans chaque domaine de services de catalogue.

  – Contient les mêmes mappes et modèles de mappes dynamiques dans chaque domaine de services de catalogue.

  – N'utilise pas le gestionnaire d'entités. Un groupe de mappes contenant une mappe d'entités n'est pas répliqué entre les domaine de services de catalogue.

  – N'utilise pas la mise en cache en écriture différée. Un groupe de mappes contenant une mappe qui est configurée avec la prise en charge de l'écriture différée n'est pas répliqué entre les domaine de services de catalogue.

  Tous les ensembles de mappes ayant les caractéristiques ci-dessus commencent à répliquer après que les domaine de services de catalogue dans la topologie ont été démarrés.

- **Chargeurs de classe avec plusieurs domaine de services de catalogue**

  Les domaine de services de catalogue doivent avoir accès à toutes les classes qui sont utilisées comme clés et valeurs Toutes les dépendances doivent être reflétées dans tous les chemins d'accès aux classes des machines virtuelles Java (JVM) de conteneur de la grille de données de tous les domaines. Si un plug-in CollisionArbiter extrait la valeur d'une entrée de cache, les classes correspondant aux valeurs doivent être présentes pour le domaine qui démarre l'arbitre.

Remarques sur les chargeurs dans une topologie multimaître

Lorsque vous utilisez des chargeurs dans une topologie multimaître, vous devez envisager les problèmes éventuels de collision et de maintenance des informations de révision. La grille de données conserve les informations de révision sur les éléments de façon à ce que les collisions puissent être détectées lorsque d'autres fragments primaires dans la configuration y écrivent des entrées. Lorsque des entrées sont ajoutées à partir d'un chargeur, ces informations de révision ne sont pas incluses et l'entrée prend une nouvelle révision. Etant donné que la révision de l'entrée semble être une nouvelle insertion, une fausse collision peut se produire si un autre fragment primaire modifie également cet état ou insère les mêmes informations à partir d'un chargeur.

Les modifications de réplication appellent la méthode get sur le chargeur avec la liste des clés qui ne sont pas déjà dans la grille de données, mais qui vont être modifiées lors de la transaction de réplication. Lorsque la réplication se produit,

ces entrées sont des entrées de collision. Lorsque les collisions sont arbitrées et que la révision est appliquée, une mise à jour par lots est appelée sur le chargeur pour appliquer les modifications à la base de données. Toutes les mappes qui ont été modifiées dans la fenêtre de révision sont mises à jour dans la même transaction.

L'énigme de préchargement

Supposons une topologie avec les deux centres de données A et B qui ont des bases de données indépendantes, mais seul le centre de données A a une grille active. Lorsque vous établissez une liaison entre les centres de données pour une configuration multimaître, les grilles de données dans le centre de données A commencent à envoyer les données aux nouvelles grilles dans le centre de données B, ce qui crée une collision avec chaque entrée. Un autre problème est l'existence de données dans la base de données du centre de données B, mais qui ne figurent pas dans la base de données du centre de données A. Ces lignes ne sont pas remplies et arbitrées, ce qui génère des incohérences qui ne sont pas résolues.

Solution de l'énigme de préchargement

Etant donné que les données qui se trouvent uniquement dans la base de données ne peuvent pas comporter des révisions, vous devez toujours précharger complètement la grille de données à partir de la base de données locale pour établir la liaison multimaître. Ensuite, les deux grilles de données peuvent réviser et arbitrer les données, pour atteindre finalement un état cohérent.

L'énigme du cache partiel

Avec un cache partiel, la première application tente de trouver des données dans la grille de données. Si les données ne sont pas dans la grille de données, elles sont recherchées dans la base de données à l'aide du chargeur. Les entrées sont supprimées de la grille de données régulièrement pour maintenir une mémoire cache de petite taille.

Ce type de mémoire cache peut être problématique dans un scénario de configuration multimaître, car les entrées dans la grille de données ont des métadonnées de révision qui permettent de détecter quand des collisions se produisent et de déterminer qui a effectué les modifications. Lorsque des liaisons entre les centres de données ne fonctionnent pas, un centre de données peut mettre à jour une entrée et ensuite éventuellement mettre à jour la base de données et invalider l'entrée dans la grille de données. Lorsque la liaison est rétablie, les centres de données tentent de synchroniser les révisions les unes par rapport aux autres. Toutefois, étant donné que la base de données a été mise à jour et que l'entrée de la grille de données a été invalidée, la modification est perdue du point de vue du centre de données qui s'est arrêté. En conséquence, les deux côtés de la grille de données sont désynchronisés et ne sont pas cohérents.

Solution de l'énigme de cache partiel

Topologie en étoile :

Vous pouvez exécuter le chargeur uniquement dans la topologie en étoile pour maintenir la cohérence des données lors de l'extension de la grille de données. Toutefois, si vous envisagez ce déploiement, notez que les chargeurs peuvent permettre à la grille de données d'être partiellement chargée, ce qui implique qu'un expulseur a été configuré. Si les rayons de la configuration sont des caches partiels, les échecs en mémoire cache n'ont aucun moyen d'extraire des données de la base

de données. En raison de cette restriction, vous devez utiliser une topologie de cache complètement remplie avec une configuration en étoile.

Invalidations et expulsion

L'invalidation crée des incohérences entre la grille de données et la base de données. Les données peuvent être supprimées de la grille de données, à l'aide d'un programme ou par l'expulsion. Lorsque vous développez votre application, sachez que le traitement des révisions ne réplique pas les modifications invalidées, ce qui provoque des incohérences entre les fragments primaires.

Les événements d'invalidation ne sont pas des modifications de l'état du cache et n'entraînent pas de réplication. Tous les expulseurs configurés s'exécutent indépendamment des autres expulseurs dans la configuration. Par exemple, vous pouvez avoir un expulseur configuré pour un seuil de mémoire dans un domaine de services de catalogue, mais un type d'expulseur différent moins agressif dans l'autre domaine de services de catalogue lié. Lorsque des entrées de grille de données sont supprimées en raison de la règle de seuil de mémoire, les entrées dans l'autre domaine de services de catalogue ne sont pas affectées.

Mises à jour de la base de données et invalidation de la grille de données

Des problèmes se produisent lorsque vous mettez à jour la base de données directement en arrière-plan lors de l'appel de l'invalidation dans la grille de données pour les entrées mises à jour dans une configuration multimaître. Ce problème se produit, car la grille de données ne peut pas répliquer la modifications dans les autres fragments primaires jusqu'à ce qu'un accès de cache transfère l'entrée vers la grille de données.

Plusieurs programmes d'écriture dans une seule base de données logique

Lorsque vous utilisez une seule base de données avec plusieurs fragments primaires qui sont connectés par l'intermédiaire d'un chargeur, des conflits transactionnels se produisent. Votre implémentation de chargeur doit gérer ces types de scénarios.

Mise en miroir des données à l'aide de la réplication multimaître

Vous pouvez configurer des bases de données indépendantes qui sont connectées à des domaine de services de catalogue indépendants. Dans cette configuration, le chargeur peut envoyer les modifications d'un centre de données vers un autre.

Considérations de conception pour la réplication multimaître

Lors de l'implémentation de la réplication multimaître, vous devez tenir compte de divers éléments dans votre conception, tels que l'arbitrage, les liaisons et les performances.

Points concernant l'arbitrage à prendre en considération dans la conception des topologies

Des collisions entre des modifications peuvent se produire s'il est possible à des enregistrements identiques d'être modifiés simultanément en deux endroits différents. Configurez chaque domaine de services de catalogue pour que les domaines aient le même nombre de processeurs, la même quantité de mémoire et le même nombre de ressources réseau. Vous remarquerez sans doute que des

domaine de services de catalogue d'exécution gérant les collisions de modifications (arbitrage) utilisent plus de ressources que d'autres domaine de services de catalogue. Les collisions sont détectées de manière automatique. Elles sont traitées avec l'un des deux mécanismes suivants :

- **Arbitre par défaut** : le protocole par défaut doit utiliser les modifications du domaine de services de catalogue occupant la position la moins basse alphabétiquement. Par exemple, si les domaine de services de catalogue A et B génèrent un conflit pour un enregistrement, la modification du domaine de services de catalogue B est ignorée. Le domaine de services de catalogue A conserve sa version et l'enregistrement dans le domaine de services de catalogue B est modifié pour qu'il corresponde à l'enregistrement du domaine de services de catalogue A. Ce comportement s'applique également aux applications où les utilisateurs ou les sessions sont normalement liés ou ont une affinité à l'une des grilles de données.
- **Arbitre personnalisé** : les applications peuvent fournir un arbitre personnalisé. Lorsqu'un domaine de services de catalogue détecte une collision, il démarre l'arbitre. Pour plus d'informations sur le développement d'un arbitre personnalisé utile, voir Développement d'arbitres personnalisés pour la réplication multi-maître.

Pour les topologies dans lesquelles les collisions sont possibles, songez à implémenter une topologie en étoile ou en arbre. Les deux topologies sont propices à éviter les collisions constantes, ce qui peut se produire dans les scénarios suivants :

1. Plusieurs domaine de services de catalogue sont affectés par une collision.
2. Chaque domaine de services de catalogue gère la collision en local, ce qui produit des révisions.
3. Les révisions entrent en collision, d'où des révisions de révisions.

Pour éviter les collisions, choisissez un domaine de services de catalogue spécifique, appelé *domaine de services de catalogue d'arbitrage* comme arbitre des collisions d'un sous-ensemble de domaine de services de catalogue. Par exemple, une topologie en étoile pourra utiliser le concentrateur comme gestionnaire de collisions. Le gestionnaire de collisions ignore toutes les collisions qui sont détectées par les sous-domaine de services de catalogue. Le domaine de services de catalogue du concentrateur crée des révisions, empêchant les révisions de collisions inattendues. Le domaine de services de catalogue qui est affecté à la gestion des collisions doit se lier à tous les domaines dont il est chargé de traiter les collisions. Dans une topologie en arbre, tous les domaines parent internes traitent les collisions pour leurs enfants immédiats. En revanche, si vous utilisez une topologie en anneau, vous ne pouvez pas désigner un domaine de services de catalogue dans le fichier comme arbitre.

Le tableau qui suit récapitule les approches en matière d'arbitrage qui sont les plus compatibles avec les diverses topologies.

Tableau 8. Approches en matière d'arbitrage. Ce tableau énonce si l'arbitrage entre applications est compatible avec les diverses topologies.

Topologie	Arbitrage d'application	Notes
Ligne de deux domaine de services de catalogue	Oui	Choisissez un domaine de services de catalogue comme arbitre.

Tableau 8. Approches en matière d'arbitrage (suite). Ce tableau énonce si l'arbitrage entre applications est compatible avec les diverses topologies.

Topologie	Arbitrage d'application	Notes
Ligne de trois domaine de services de catalogue	Oui	Le domaine de services de catalogue du milieu doit être l'arbitre. Assimilez ce domaine de services de catalogue au concentrateur dans une topologie en étoile simple.
Ligne de plus de trois domaine de services de catalogue	Non	L'arbitrage d'application n'est pas pris en charge.
Concentrateur avec n "rayons"	Oui	Le concentrateur avec des liens vers toutes les branches doit être le domaine de services de catalogue d'arbitrage.
Anneau de N domaine de services de catalogue	Non	L'arbitrage d'application n'est pas pris en charge.
Arbre dirigé acyclique (arbre n-aire)	Oui	Tous les noeuds racine doivent évaluer leurs descendants directs uniquement.

Points concernant les liens à prendre en considération dans la conception des topologies

Dans l'idéal, une topologie comprend le minimum de liens tout en optimisant les compromis entre les temps d'attente des modifications, la tolérance aux pannes et les caractéristiques de performances.

- **Temps d'attente des modifications**

  Le temps d'attente de modification est déterminé par le nombre de domaine de services de catalogue intermédiaires par lequel un changement doit passer avant d'arriver à un domaine de services de catalogue spécifique.

  Une topologie a le meilleur temps d'attente lorsqu'elle élimine les domaine de services de catalogue intermédiaires en liant chacun des domaine de services de catalogue à chacun des autres domaine de services de catalogue. Toutefois, un domaine de services de catalogue doit effectuer la réplication par rapport à son nombre de liens. Pour les topologies de grande taille, le nombre de liens à définir peut entraîner une charge administrative.

  La vitesse à laquelle une modification est copiée vers les autres domaine de services de catalogue dépend de facteurs supplémentaires, tels que :

  – Bande passante du processeur et du réseau dans le domaine de services de catalogue source
  – Nombre de domaine de services de catalogue intermédiaire et de liens entre la source et la cible du domaine de services de catalogue source et cible
  – Ressources en processeur et en réseau disponibles pour les domaine de services de catalogue source, cible et intermédiaires

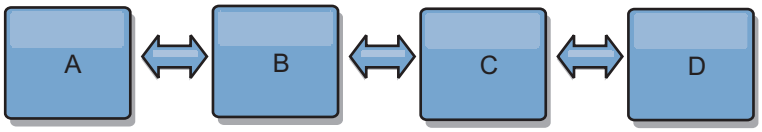
- **Tolérance aux pannes**

  La tolérance aux pannes est déterminée par le nombre de chemins existant entre deux domaine de services de catalogue pour la réplication des modifications.

  Si vous ne disposez que d'un seul lien entre une paire de domaine de services de catalogue, une défaillance de lien empêche la propagation des modifications. De même, les modifications ne sont pas propagées entre les domaine de services de catalogue si un incident de liaison se produit sur les domaines intermédiaires.

Votre topologie pourrait avoir un lien unique d'un domaine de services de catalogue vers un autre de sorte que le lien passe par des domaines intermédiaires. Dans ce cas, les modifications ne sont pas propagées si l'un des domaine de services de catalogue intermédiaires est défaillant.

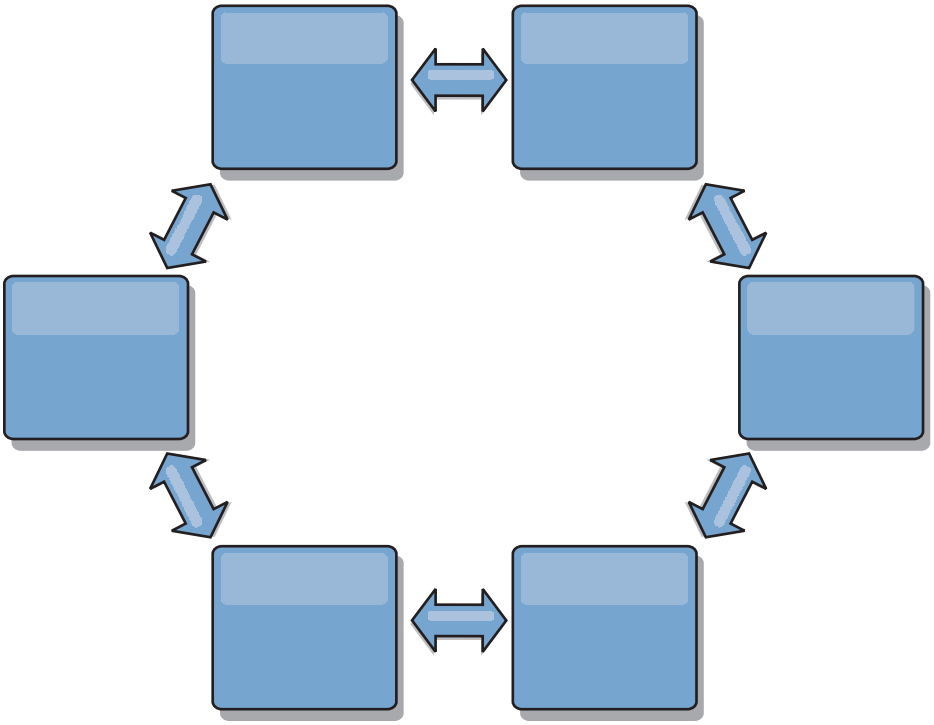
Supposons la topologie linéaire à quatre domaine de services de catalogue A, B, C et D :



Si l'une de ces conditions existe, le domaine D ne voit pas les modification de A :

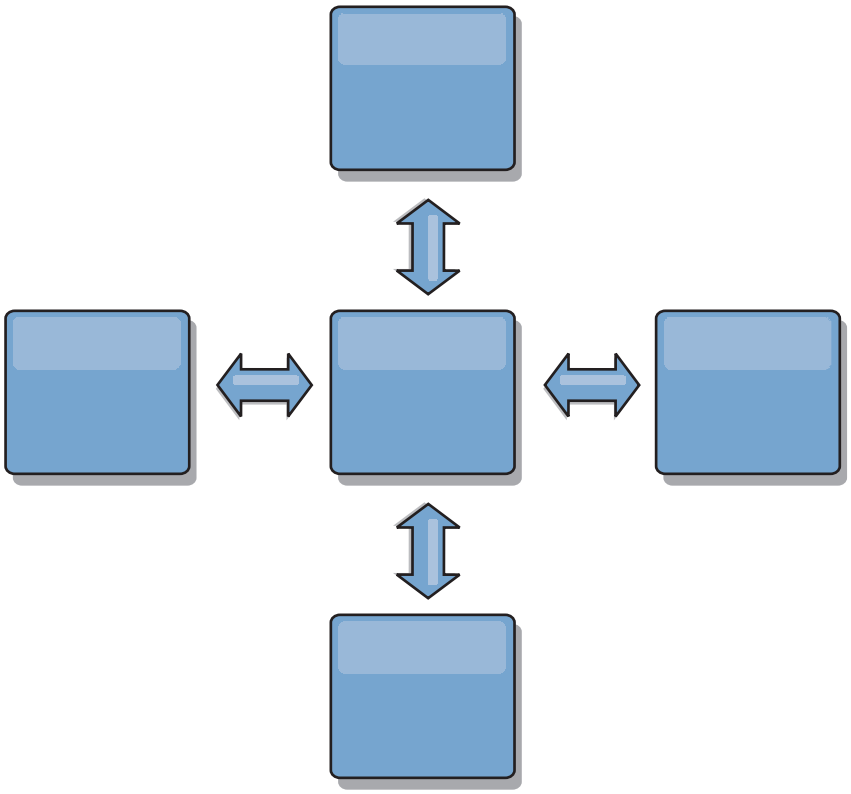
- Le domaine A est actif et B est arrêté.
- Les domaines A et B sont actifs et C est arrêté.
- Le lien entre A et B ne fonctionne pas.
- Le lien entre B et C ne fonctionne pas.
- Le lien entre C et D est arrêté.

En revanche, avec une topologie en anneau, chaque domaine de services de catalogue peut recevoir les modifications dans un sens ou dans l'autre.



Par exemple, si un service de catalogue donné de votre topologie en anneau est arrêté, les deux domaines contigus peuvent toujours extraire les modifications directement de l'autre.

Toutes les modifications sont propagées via le concentrateur. Par conséquent, contrairement aux topologies linéaires et en anneau, la conception en étoile peut tomber en panne si le concentrateur est défaillant.



Un domaine de services de catalogue unique est résilient à un certain degré de perte de service. Cependant, les incidents les plus importants, tels que les indisponibilités de réseau étendu ou les pertes de liaisons entre les centres de données physiques peuvent perturber les domaine de services de catalogue.

- **Liaison et performances**

  Le nombre de liaisons définies sur un domaine le service de catalogue affecte les performances. Un plus grand nombre de liaisons utilisent davantage de ressources et les performances de réplication peuvent baisser. La possibilité d'extraire les modifications pour un domaine A via d'autres domaines empêche le domaine A de répliquer ses transactions partout. La charge de la répartition des modifications dans un domaine est limitée par le nombre de liaisons qu'il utilise et non pas par le nombre de domaines dans la topologie. Cette propriété est synonyme d'évolutivité et les domaines de la topologie peuvent partager la charge de la répartition des modifications.

  Un domaine de services de catalogue peut extraire les modifications indirectement via d'autres domaine de services de catalogue. Supposons une topologie linéaire avec cinq domaine de services de catalogue.

  ```
  A <=> B <=> C <=> D <=> E
  ```

  – A extrait les modifications de B, C, D, et E via B
  – B extrait les modifications directement de A et de C et les modifications de D et de E via C
  – C extrait les modifications directement de B et de D et les modifications de A via B et de E via D
  – D extrait les modifications directement de C et de E et les modifications de A et de B via C
  – E extrait les modifications directement de D et les modifications de A, B et C via D

La charge de la répartition dans les domaine de services de catalogue A et E est la plus faible, car ils ont chacun une seule liaison à un domaine de services de catalogue unique. Les domaines B, C et D ont chacun une liaison avec deux domaines. Par conséquent, la charge de la répartition dans les domaines B, C, et D est le double de celle des domaines A et E. La charge de travail dépend du nombre de liaisons dans chaque domaine et non pas du nombre total de domaines dans la topologie. Par conséquent, la répartition de charge décrite demeurerait constante, même si la ligne contenait 1 000 domaines.

Considérations relatives aux performances de réplication multimaître

Tenez compte des limitations suivantes lorsque vous utilisez des topologies de réplication multimaître :

- **Optimisation de la répartition des modifications**, comme expliquée dans la section précédente.
- **Performances des liens de réplication** WebSphere eXtreme Scale crée un seul socket TCP/IP entre n'importe quelle paire de machines virtuelles Java. Tout le trafic entre les machines virtuelles Java passe par le socket unique, y compris le trafic de la réplication multimaître. Les domaine de services de catalogue sont hébergés dans au moins *n* machines virtuelles Java pour fournir au minimum *n* liaisons TCP aux domaine de services homologues. Ainsi, les domaine de services de catalogue avec un plus grand nombre de conteneurs offrent de meilleures performances de réplication. Un plus grand nombre de conteneurs requiert davantage de processeurs et de ressources réseau.
- **Le support de l'optimisation de la fenêtre dynamique TCP et RFC 1323** RFC 1323 à chaque extrémité d'une liaison renvoie plus de données pour un aller-retour. Ce support augmente le débit en développant la capacité de la fenêtre d'un facteur d'environ 16 000.

  Notez que les sockets TCP utilisent un mécanisme de fenêtre dynamique pour contrôler le flux des données en vrac. Ce mécanisme limite généralement le socket à 64 Ko pour un intervalle d'aller-retour. Si l'intervalle aller-retour est de 100 ms, la bande passante est limitée à 640 Ko/s sans optimisation supplémentaire. L'utilisation intégrale de la bande passante disponible sur un lien peut nécessiter une optimisation qui est spécifique au système d'exploitation. La plupart des systèmes d'exploitation comportent des paramètres d'optimisation, y compris des options RFC 1323, permettant d'améliorer le débit sur les liaisons à forte latence.

  Plusieurs facteurs peuvent affecter les performances de la réplication :

  – Vitesse d'extraction des modifications par eXtreme Scale.
  – Vitesse à laquelle eXtreme Scale peut traiter les demandes de réplication d'extraction.
  – Capacité de la fenêtre dynamique.
  – Avec l'optimisation de la mémoire tampon réseau aux deux extrémités d'une liaison, eXtreme Scale extrait les modification sur le socket de manière efficace.
- **Sérialisation des objets** Toutes les données doivent être sérialisables. Si un domaine de services de catalogue n'utilise pas `COPY_TO_BYTES`, il doit utiliser la sérialisation Java ou ObjectTransformers pour optimiser les performances de sérialisation.
- Par défaut **la compression** WebSphere eXtreme Scale compresse toutes les données envoyées entre les domaines de services de catalogue. Vous ne pouvez désactiver actuellement la compression.

- **Optimisation de la mémoire** L'utilisation de la mémoire pour une topologie de réplication multimaître est largement indépendante du nombre de domaine de services de catalogue dans la topologie.

  La réplication multimaître ajoute un temps de traitement fixe par entrée de mappe pour la gestion des versions. Chaque conteneur suit également une quantité fixe de données pour chaque domaine de services de catalogue dans la topologie. Une topologie à deux domaines de services de catalogue utilise approximativement la même quantité de mémoire qu'une topologie à 50 domaine de services de catalogue. WebSphere eXtreme Scale n'utilise pas de journaux de relecture ou de files d'attente similaires dans son implémentation. Ainsi, aucune structure de récupération n'est prête si la liaison de réplication n'est pas disponible pendant un certain temps et redémarre ensuite.

Interopérabilité avec d'autres produits WebSphere

WebSphere eXtreme Scale peut être intégré à d'autres produits de serveurs, comme WebSphere Application Server et WebSphere Application Server Community Edition.

WebSphere Application Server

Vous pouvez intégrer WebSphere Application Server à divers éléments de votre configuration WebSphere eXtreme Scale. Vous pouvez déployer des applications de grille de données et utiliser WebSphere Application Server pour héberger les serveurs de conteneur et de catalogue. Vous pouvez également utiliser la sécurité WebSphere Application Server dans votre environnement WebSphere eXtreme Scale.

WebSphere Portal

Vous pouvez rendre persistantes des sessions HTTP depuis WebSphere Portal dans une grille de données dans WebSphere eXtreme Scale.

WebSphere Application Server Community Edition

WebSphere Application Server Community Edition peut partager l'état des sessions, mais d'une manière peu efficace et non évolutive. WebSphere eXtreme Scale fournit une couche de persistance répartie à hautes performances qui peut servir à répliquer l'état mais sans s'intégrer facilement aux autres serveurs d'applications extérieurs à WebSphere Application Server. Vous pouvez intégrer ces deux produits pour offrir une solution de gestion de session évolutive.

WebSphere Real Time

Avec le support pour WebSphere Real Time, l'offre Java temps réel la plus efficace, WebSphere eXtreme Scale permet aux applications Extreme Transaction Processing (XTP) d'avoir des temps de réponse plus cohérents et plus prévisibles.

Chapitre 3. Scénarios

Un scénario présente des exemples pour aider l'utilisateur à comprendre un concept ou à accomplir une tâche. Le scénario utilise des informations du monde réel pour construire une image complète.

Utilisation d'un environnement OSGi pour développer et exécuter les plug-ins eXtreme Scale

Utilisez ces scénarios pour effectuer les tâches courantes dans un environnement OSGi. Par exemple, l'infrastructure OSGi est idéale pour le démarrage des serveurs et des clients dans un conteneur OSGi, ce qui vous permet d'ajouter et de mettre à jour dynamiquement les plug-ins WebSphere eXtreme Scale dans l'environnement d'exécution.

Les scénarios suivants concernent la création et l'exécution de plug-ins dynamiques, ce qui permet d'installer, de démarrer, d'arrêter, de modifier et de désinstaller dynamiquement des plug-ins dynamiques. Vous pouvez également suivre un autre scénario probable, ce qui permet d'utiliser la structure OSGi sans fonctions dynamiques. Vous pouvez toujours modulariser vos applications comme des ensembles définis par et communiqués via des services. Ces ensembles basés sur des services offrent plusieurs avantages, notamment des fonctions de développement et de déploiement plus efficaces.

Objectifs des scénarios

Après avoir suivi les leçons de ce module, vous saurez :

- Générer des plug-ins dynamiques eXtreme Scale utilisables dans un environnement OSGi.
- Exécuter de conteneurs eXtreme Scale dans un environnement OSGi sans fonctions dynamiques.

Conditions prérequises

Consultez la rubrique «Présentation de l'infrastructure OSGi», à la page 24 pour en savoir plus sur la prise en charge d'OSGi et ses avantages.

Présentation de l'infrastructure OSGi

OSGi définit un système de module dynamique pour Java. La plateforme de service OSGi présente une architecture à couches et elle est conçue pour s'exécuter dans plusieurs profils standard Java. Vous pouvez démarrer les serveurs et les clients WebSphere eXtreme Scale dans un conteneur OSGi.

Avantages de l'exécution des applications dans le conteneur OSGi

Le support WebSphere eXtreme Scale OSGi permet de déployer le produit dans l'infrastructure OSGi Eclipse Equinox. Auparavant, si vous souhaitiez mettre à niveau les plug-ins utilisés par eXtreme Scale, vous deviez redémarrer la machine virtuelle Java (JVM) pour appliquer les nouvelles versions des plug-ins. Avec le support de plug-ins dynamiques fourni par l'infrastructure OSGi, vous pouvez

désormais mettre à jour les classes de plug-in sans redémarrer la machine virtuelle Java. Ces plug-ins sont exportés par ensembles d'utilisateur comme services. WebSphere eXtreme Scale accède au(x) service(s) en les recherchant dans le registre OSGi.

Les conteneurs eXtreme Scale peuvent être configurés pour démarrer plus aisément et dynamiquement le service d'administration de configuration OSGi ou avec OSGi Blueprint. Si vous voulez déployer une nouvelle grille avec sa stratégie de placement, vous pouvez le faire en créant une configuration OSGi ou en déployant un ensemble avec des fichiers descripteurs XML eXtreme Scale. Avec le support OSGi, les ensembles d'applications contenant des données de configuration eXtreme Scale peuvent être installés, démarrés, mis à jour et désinstallés sans redémarrer tout le système. Avec cette fonction, vous pouvez mettre à niveau l'application sans perturber la grille de données.

Les beans de plug-in et les services peuvent être configurés avec des portées de fragment personnalisées pour permettre une intégration précise d'options aux autres services exécutés dans la grille. Chaque plug-in peut utiliser des classements OSGi Blueprint pour vérifier que chaque instance activée du plug-in correspond au niveau de version correct. Un bean géré OSGi (MBean) et l'utilitaire **xscmd** sont fournis pour vous permettre d'exécuter des requêtes sur les services OSGi de plug-in eXtreme Scale et leurs classements.

Avec cette fonction, les administrateurs peuvent identifier rapidement les erreurs potentielles de configuration et d'administration et mettre à jour les classements de service de plug-in utilisés par eXtreme Scale.

Ensembles OSGi

Pour interagir avec les plug-ins et déployer des plug-ins dans l'infrastructure OSGi, vous devez utiliser des *ensembles*. Dans la plateforme de service OSGi, un ensemble est un fichier d'archive JAR (Java archive) qui contient du code Java, des ressources et un manifeste qui décrit l'ensemble et ses dépendances. L'ensemble représente l'unité de déploiement d'une application. Le produit eXtreme Scale prend en charge les types d'ensembles suivants :

Ensemble de serveur
L'ensemble de serveur est le fichier `objectgrid.jar`. Il est installé avec le serveur autonome eXtreme Scale et il est nécessaire pour exécuter les serveurs eXtreme Scale. Vous pouvez également l'utiliser pour exécuter les clients eXtreme Scale ou les mémoires caches internes locales. L'ID d'ensemble pour le fichier `objectgrid.jar` est com.ibm.websphere.xs.server_`<version>`, où la version a le format `<Version>.<Release>.<Modification>`. Par exemple, l'ensemble de serveur pour eXtreme Scale version 7.1.1 est com.ibm.websphere.xs.server_7.1.1.

Ensemble de client
L'ensemble de client est le fichier `ogclient.jar`. Il est installé en même temps que les installations client et autonome eXtreme Scale et il est utilisé pour exécuter les clients eXtreme Scale ou les mémoires caches internes locales. L'ID d'ensemble du fichier `ogclient.jar` est com.ibm.websphere.xs.client_`version`, où la version a le format `<Version>.<Release>.<Modification>`. Par exemple, l'ensemble de client pour eXtreme Scale version 7.1.1 est com.ibm.websphere.xs.client_7.1.1.

Limitations

Vous ne pouvez pas redémarrer l'ensemble eXtreme Scale, car vous ne pouvez pas redémarrer l'ORB (object request broker). Pour redémarrer le serveur eXtreme Scale, vous devez redémarrer l'infrastructure OSGi.

Installation de l'infrastructure OSGi Eclipse Equinox avec Eclipse Gemini pour les clients et les serveurs

Si vous souhaitez déployer WebSphere eXtreme Scale dans la structure OSGi, vous devez configurer l'environnement Eclipse Equinox.

Pourquoi et quand exécuter cette tâche

La tâche nécessite que vous téléchargiez et installiez l'infrastructure Blueprint qui permet de configurer ensuite les JavaBeans et de les exposer en tant que services. L'utilisation de services est importante, car vous pouvez exposer des plug-ins en tant que services OSGi pour qu'ils puissent être utilisés par l'environnement d'exécution eXtreme Scale. Le produit prend en charge deux conteneurs Blueprint dans l'infrastructure OSGi principale Eclipse Gemini et Apache Aries. Utilisez cette procédure pour configurer le conteneur Gemini Eclipse.

Procédure

1. Téléchargez Eclipse Equinox SDK Version 3.6.1 ou la version suivante à partir du site Web Eclipse. Créez un répertoire pour l'infrastructure Equinox, par exemple, `/opt/equinox`. Ces instructions font référence à ce répertoire sous la forme `equinox_root`. Extrayez le fichier compressé dans le répertoire `equinox_root`.
2. Téléchargez le fichier compressé gemini-plan d'incubation 1.0.0 depuis le site Web Eclipse. Extrayez le contenu du fichier dans un répertoire temporaire et copiez les fichiers extraits suivants vers le répertoire `equinox_root/plugins` :

   ```
   dist/gemini-blueprint-core-1.0.0.jar
   dist/gemini-blueprint-extender-1.0.0.jar
   dist/gemini-blueprint-io-1.0.0.jar
   ```

3. Téléchargez Spring Framework Version 3.0.5 à partir de la page Web SpringSource http://www.springsource.com/download/community. Extrayez le contenu du fichier dans un répertoire temporaire et copiez les fichiers extraits suivants vers le répertoire `equinox_root/plugins` :

   ```
   org.springframework.aop-3.0.5.RELEASE.jar
   org.springframework.asm-3.0.5.RELEASE.jar
   org.springframework.beans-3.0.5.RELEASE.jar
   org.springframework.context-3.0.5.RELEASE.jar
   org.springframework.core-3.0.5.RELEASE.jar
   org.springframework.expression-3.0.5.RELEASE.jar
   ```

4. Téléchargez le fichier AOP Alliance Java archive (JAR) depuis la page Web SpringSource. Copiez com.springsource.org.aopalliance-1.0.0.jar vers le répertoire `equinox_root/plugins` .
5. Téléchargez le fichier JAR Apache commons logging 1.1.1 JAR depuis la page Web SpringSource. Copiez le fichier `com.springsource.org.apache.commons.logging-1.1.1.jar` vers le répertoire `equinox_root/plugins`.
6. Téléchargez le client de ligne de commande Luminis OSGi Configuration Admin. Utilisez cet ensemble pour gérer les configurations d'administration OSGi. Vous pouvez télécharger le fichier JAR depuis la page Web https://opensource.luminis.net/wiki/display/SITE/

OSGi+Configuration+Admin+command+line+client. Copiez le fichier net.luminis.cmc-0.2.5.jar vers le répertoire equinox_root/plugins.

7. Téléchargez l'ensemble Apache Felix file installation Version 3.0.2 depuis la page Web http://felix.apache.org/site/index.html. Copiez le fichier org.apache.felix.fileinstall-3.0.2.jar vers le répertoire equinox_root/plugins.
8. Créez un répertoire de configuration dans le répertoire equinox_root/plugins, par exemple :

   ```
   mkdir equinox_root/plugins/configuration
   ```

9. Créez le fichier config.ini suivant dans le répertoire equinox_root/plugins/configuration en remplaçant *equinox_root* par le chemin absolu dans le chemin du répertoire equinox_root en supprimant tous les espaces après la barre oblique inverse dans chaque ligne. Vous devez placer une ligne blanche à la fin du fichier, par exemple :

   ```
   osgi.noShutdown=true
   osgi.java.profile.bootdelegation=none
   org.osgi.framework.bootdelegation=none
   eclipse.ignoreApp=true
   osgi.bundles=\
   org.eclipse.osgi.services_3.2.100.v20100503.jar@1:start, \
   org.eclipse.osgi.util_3.2.100.v20100503.jar@1:start, \
   org.eclipse.equinox.cm_1.0.200.v20100520.jar@1:start, \
   com.springsource.org.apache.commons.logging-1.1.1.jar@1:start, \
   com.springsource.org.aopalliance-1.0.0.jar@1:start, \
   org.springframework.aop-3.0.5.RELEASE.jar@1:start, \
   org.springframework.asm-3.0.5.RELEASE.jar@1:start, \
   org.springframework.beans-3.0.5.RELEASE.jar@1:start, \
   org.springframework.context-3.0.5.RELEASE.jar@1:start, \
   org.springframework.core-3.0.5.RELEASE.jar@1:start, \
   org.springframework.expression-3.0.5.RELEASE.jar@1:start, \
   org.apache.felix.fileinstall-3.0.2.jar@1:start, \
   net.luminis.cmc-0.2.5.jar@1:start, \
   gemini-blueprint-core-1.0.0.jar@1:start, \
   gemini-blueprint-extender-1.0.0.jar@1:start, \
   gemini-blueprint-io-1.0.0.jar@1:start
   ```

   Si vous avez déjà configuré l'environnement, vous pouvez nettoyer le référentiel de plug-in Equinox en supprimant le répertoire equinox_root\plugins\configuration\org.eclipse.osgi.
10. Exécutez la commande suivante pour démarrer la console Equinox.

    Si vous exécutez une version différente d'Equinox, le nom du fichier JAR est différent de celui de l'exemple ci-dessous :

    ```
    java -jar plugins\org.eclipse.osgi_3.6.1.R36x_v20100806.jar -console
    ```

Installation des ensembles eXtreme Scale

WebSphere eXtreme Scale inclut des ensembles qui peuvent être installés dans une infrastructure OSGi Eclipse Equinox. Ces ensembles sont nécessaires pour démarrer les serveurs eXtreme Scale ou utiliser les clients eXtreme Scale dans OSGi.

Avant de commencer

Cette tâche suppose que les produits suivants ont été installés :

- Infrastructure OSGi Eclipse Equinox
- Client ou serveur autonome eXtreme Scale

Pourquoi et quand exécuter cette tâche

eXtreme Scale inclut deux ensembles. Un seul des ensembles suivants est nécessaire dans une infrastructure OSGi :

objectgrid.jar
: L'ensemble de serveur est le fichier objectgrid.jar. Il est installé avec l'installation de serveur autonome eXtreme Scale et il est nécessaire pour exécuter les serveurs eXtreme Scale servers. Il peut être aussi utilisé pour

exécuter les clients eXtreme Scale ou les mémoires caches internes locales. L'ID d'ensemble du fichier `objectgrid.jar` est com.ibm.websphere.xs.server_<version>, où la version a le format <Version>.<Release>.<Modification>. Par exemple, l'ensemble de serveur pour eXtreme Scale version 7.1.1 est com.ibm.websphere.xs.server_7.1.1.

ogclient.jar
L'ensemble `ogclient.jar` est installé avec les installations client et autonomes eXtreme Scale et il est utilisé pour exécuter les clients eXtreme Scale ou les mémoires caches internes locales. L'ID d'ensemble du fichier `ogclient.jar` est com.ibm.websphere.xs.client_<version>, où la version a le format <Version>_<Release>_<Modification. Par exemple, l'ensemble client pour eXtreme Scale Version 7.1.1 est com.ibm.websphere.xs.client_7.1.1.

Pour plus d'informations sur le développement de plug-ins eXtreme Scale, voir la rubrique API système et plug-ins.

Procédure

Pour installer l'ensemble client ou serveur eXtreme Scale dans l'infrastructure OSGi Eclipse Equinox en utilisant la console OSGi :

1. Démarrez l'infrastructure Eclipse Equinox avec la console activée. Par exemple :
   ```
   rép_base_java/bin/java -jar <equinox_root>/plugins/
   org.eclipse.osgi_3.6.1.R36x_v20100806.jar -console
   ```
2. Installez l'ensemble serveur ou client eXtreme Scale dans la console Equinox :
   ```
   osgi> install file:///<path to bundle>
   ```
3. Equinox affiche l'ID d'ensemble du nouvel ensemble installé :
   ```
   Bundle id is 25
   ```
4. Démarrez l'ensemble dans la console Equinox, où <id> est l'ID affecté à l'ensemble lors de son installation :
   ```
   osgi>  start <id>
   ```
5. Extrayez l'état du service dans la console Equinox pour vérifier que l'ensemble a démarré. Par exemple :
   ```
   osgi> ss
   ```

   Lorsque l'ensemble a démarré correctement, il affiche l'état ACTIVE, par exemple :
   ```
   25      ACTIVE      com.ibm.websphere.xs.server_7.1.1
   ```

Installez l'ensemble client ou serveur eXtreme Scale dans l'infrastructure OSGi Eclipse Equinox en utilisant le fichier `config.ini` :

6. Copiez l'ensemble client ou serveur eXtreme Scale (objectgrid.jar ou ogclient.jar) de <wxs_install_root>/ObjectGrid/lib vers le répertoire Eclipse Equinox, par exemple : <equinox_root>/plugins
7. Modifiez le fichier de configuration Eclipse Equinox `config.ini` et ajoutez l'ensemble à la propriété osgi.bundles, par exemple :
   ```
   osgi.bundles=\
   org.eclipse.osgi.services_3.2.100.v20100503.jar@1:start, \
   org.eclipse.osgi.util_3.2.100.v20100503.jar@1:start, \
   org.eclipse.equinox.cm_1.0.200.v20100520.jar@1:start, \
   objectgrid.jar@1:start
   ```

   Important : Vérifiez qu'une ligne blanche existe après le dernier nom d'ensemble. Chaque ensemble est séparé par une virgule.

8. Démarrez l'infrastructure Eclipse Equinox avec la console activée. Par exemple :
   ```
   rép_base_java/bin/java -jar <equinox_root>/plugins/
   org.eclipse.osgi_3.6.1.R36x_v20100806.jar -console
   ```
9. Extrayez l'état du service dans la console Equinox pour vérifier que l'ensemble a démarré :
   ```
   osgi> ss
   ```

   Lorsque l'ensemble a démarré correctement, il affiche l'état ACTIVE, par exemple :
   ```
   25      ACTIVE      com.ibm.websphere.xs.server_7.1.1
   ```

Résultats

L'ensemble client ou serveur eXtreme Scale est installé et démarré dans l'infrastructure OSGi Eclipse Equinox.

Génération et exécution de plug-ins dynamiques eXtreme Scale pour une utilisation dans un environnement OSGi

Tous les plug-ins eXtreme Scale peuvent être configurés pour un environnement OSGi. Les plug-ins dynamiques offrent pour principal avantages de pouvoir les mettre à niveau sans fermer la grille. Cela permet de faire évoluer une application sans redémarrer les processus conteneur de la grille.

Pourquoi et quand exécuter cette tâche

Le support OSGi WebSphere eXtreme Scale permet de déployer le produit dans une infrastructure OSGi, telle que Eclipse Equinox. Auparavant, si vous souhaitiez mettre à niveau les plug-ins utilisés par eXtreme Scale, vous deviez redémarrer la machine virtuelle Java (JVM) pour appliquer les nouvelles versions des plug-ins. Avec le support des plug-ins dynamiques fourni par eXtreme Scale et la possibilité de mettre à jour les ensembles que l'infrastructure OSGi fournit, vous pouvez désormais mettre à jour les classes de plug-in sans redémarrer la machine JVM. Ces plug-ins sont exportés par *ensembles* comme services. WebSphere eXtreme Scale accède au service en consultant le registre OSGi. Dans la plateforme de service OSGi, un ensemble est un fichier archive Java (JAR) qui contient du code Java, des ressources et un manifeste qui décrit le regroupement et ses dépendances. L'ensemble représente l'unité de déploiement d'une application.

Procédure

1. Créer des plug-ins dynamiques eXtreme Scale.
2. Configurer les plug-ins eXtreme Scale avec OSGi Blueprint.
3. Installer et démarrer les plug-ins OSGi.

Génération de plug-ins dynamiques eXtreme Scale

WebSphere eXtreme Scale inclut les plug-ins ObjectGrid et BackingMap. Ces plug-ins sont implémentés dans Java et configurés en utilisant le fichier XML descripteur ObjectGrid. Pour créer un plug-in dynamique qui peut être mis à niveau dynamiquement, il doit connaître les événements de cycle de vie ObjectGrid et BackingMap, car il peut être nécessaire qu'il exécute des actions lors d'une mise à jour. L'amélioration d'un module d'extension avec des méthodes de rappel de cycle de vie, des programmes d'écoute d'événements ou les deux permet aux plug-ins d'effectuer ces actions au moment opportun.

Avant de commencer

Cette rubrique suppose que vous avez créé le plug-in approprié. Pour plus d'informations sur le développement de plug-ins eXtreme Scale, voir la rubrique API système et plug-ins.

Pourquoi et quand exécuter cette tâche

Tous les plug-ins eXtreme Scale s'appliquent à une instance BackingMap ou ObjectGrid. De nombreux plug-ins interagissent également avec d'autres plug-ins. Par exemple, un chargeur et un plug-in TransactionCallback fonctionnent ensemble pour interagir correctement avec une transaction de base de données et les divers appels JDBC de base de données. Certains modules d'extension peut également s'avérer nécessaires pour mettre en mémoire cache les données de configuration des autres plug-ins pour améliorer les performances.

Les plug-ins BackingMapLifecycleListener et ObjectGridLifecycleListener fournissent des opérations de cycle de vie pour les instances BackingMap et ObjectGrid correspondantes. Ce processus permet aux plug-ins d'être avertis lorsque le parent BackingMap ou ObjectGrid et ses plug-ins correspondants peuvent être modifiés. Les plug-ins BackingMap implémentent l'interface BackingMapLifecyleListener et les plug-ins ObjectGrid implémentent l'interface ObjectGridLifecycleListener. Ces plug-ins sont appelés automatiquement lorsque le cycle de vie du parent BackingMap ou ObjectGrid change. Pour plus d'informations sur les plug-ins de cycle de vie, voir la rubrique Gestion des cycles de vie du plug-in.

Vous améliorerez les ensembles en utilisant les méthodes de cycle de vie ou les programmes d'écoute dans les tâches communes suivantes :

- Démarrage et arrêt des ressources, telles que les unités d'exécution ou les abonnés de messagerie.
- Indication qu'une notification se produit lorsque des plug-ins homologues ont été mis à jour, ce qui permet d'accéder directement au plug-in et de détecter les modifications.

Lorsque vous accédez directement à un autre plug-in, accédez-y via le conteneur OSGi pour que tous les composants du système fassent référence au plug-in correct. Si, par exemple, un composant dans l'application fait directement référence, ou met en mémoire cache, une instance d'un plug-in, il conserve sa référence à cette version du plug-in, même lorsque le plug-in est mis à jour dynamiquement. Ce comportement peut causer des problèmes au niveau de l'application ainsi que des fuites de mémoire. Par conséquent, écrivez le code qui dépend des plug-ins dynamiques qui obtient sa référence en utilisant OSGi, la sémantique getService(). Si l'application doit mettre en cache un ou plusieurs plug-ins, elle écoute les événements de cycle de vie en utilisant les interfaces ObjectGridLifecycleListener et BackingMapLifecycleListener. L'application doit pouvoir également régénérer sa mémoire cache lorsque cela est nécessaire en sécurisant les unités d'exécution.

Tous les plug-ins eXtreme Scale utilisés avec OSGi doivent également implémenter l'interface BackingMapPlugin ou ObjectGridPlugin correspondante. Les nouveaux plug-ins, tels que l'interface MapSerializerPlugin, appliquent cette pratique. Ces interfaces fournissent à l'environnement d'exécution eXtreme Scale et OSGi une interface cohérente pour injecter l'état dans le plug-in et contrôler son cycle de vie.

Utilisez cette tâche pour spécifier qu'une notification se produit lorsque des plug-ins homologues sont mis à jour. Vous pouvez créez une fabrique d'écoute qui produit une instance de programme d'écouter.

Procédure

- Mettez à jour la classe de plug-in ObjectGrid pour implémenter l'interface ObjectGridPlugin. Cette interface contient des méthodes qui permettent à eXtreme Scale d'initialiser et définir l'instance ObjectGrid et détruire le plug-in. Reportez-vous à l'exemple de code suivant :

```
package com.mycompany;
import com.ibm.websphere.objectgrid.plugins.ObjectGridPlugin;
...

public class MyTranCallback implements TransactionCallback, ObjectGridPlugin {

    private ObjectGrid og = null;

    private enum State {
        NEW, INITIALIZED, DESTROYED
    }

    private State state = State.NEW;

    public void setObjectGrid(ObjectGrid grid) {
        this.og = grid;
    }

    public ObjectGrid getObjectGrid() {
        return this.og;
    }
    void initialize() {
        // Traiter l'initialisation de plug-in ici.  Cela peut être appelé par
        // eXtreme Scale et non pas par le gestionnaire de bean OSGi.
        state = State.INITIALIZED;
    }
    boolean isInitialized() {
        return state == State.INITIALIZED;
    }

    public void destroy() {
        // Détruire le plug-in et libérer les ressources.  Cela
        peut être appelé par le gestionnaire de bean OSGi ou eXtreme Scale.
        state = State.DESTROYED;
    }

    public boolean isDestroyed() {
        return state == State.DESTROYED;
    }
}
```

- Mettez à jour la classe de plug-in ObjectGrid pour implémenter l'interface ObjectGridLifecycleListener. Reportez-vous à l'exemple de code suivant :

```
package com.mycompany;
import com.ibm.websphere.objectgrid.plugins.ObjectGridLifecycleListener;
      import com.ibm.websphere.objectgrid.plugins.ObjectGridLifecycleListener.LifecycleEvent;
      ...

      public class MyTranCallback implements TransactionCallback, ObjectGridPlugin, ObjectGridLifecycleListener{
          public void objectGridStateChanged(LifecycleEvent event) {
              switch(event.getState()) {
              case NEW:
              case DESTROYED:
              case DESTROYING:
              case INITIALIZING:
                  break;
              case INITIALIZED:
                  // Rechercher un chargeur ou un plug-in MapSerializerPlugin en utilisant
                  // OSGi ou directement depuis l'instance ObjectGrid.
                  lookupOtherPlugins()
                  break;
              case STARTING:
              case PRELOAD:
                  break;
              case ONLINE:
                  if (event.isWritable()) {
                      startupProcessingForPrimary();
                  } else {
                      startupProcessingForReplica();
                  }
                  break;
              case QUIESCE:
                  if (event.isWritable()) {
                      quiesceProcessingForPrimary();
                  } else {
                      quiesceProcessingForReplica();
                  }
```

```
                break;
            case OFFLINE:
                shutdownShardComponents();
                break;
            }
        }
        ...
    }
```

- Mettez à jour un plug-in BackingMap. Mettez à jour la classe de plug-in BackingMap pour implémenter l'interface de plug-in BackingMap. Cette interface inclut des méthodes qui permettent à eXtreme Scale d'initialiser, définir l'instance BackingMap et détruire le plug-in. Reportez-vous à l'exemple de code suivant :

```
package com.mycompany;
import com.ibm.websphere.objectgrid.plugins.BackingMapPlugin;
...

public class MyLoader implements Loader, BackingMapPlugin {

    private BackingMap bmap = null;

    private enum State {
        NEW, INITIALIZED, DESTROYED
    }

    private State state = State.NEW;

    public void setBackingMap(BackingMap map) {
        this.bmap = map;
    }

    public BackingMap getBackingMap() {
        return this.bmap;
    }
    void initialize() {
        // Traiter l'initialisation de plug-in ici.  Cela peut être appelé par
        // eXtreme Scale et non pas par le gestionnaire de bean OSGi.
        state = State.INITIALIZED;
    }
    boolean isInitialized() {
        return state == State.INITIALIZED;
    }

    public void destroy() {
        // Détruire le plug-in et libérer les ressources.  Cela
        peut être appelé par le gestionnaire de bean OSGi ou eXtreme Scale.
        state = State.DESTROYED;
    }

    public boolean isDestroyed() {
        return state == State.DESTROYED;
    }
}
```

- Mettez à jour la classe de plug-in BackingMap pour implémenter une interface BackingMapLifecycleListener. Reportez-vous à l'exemple de code suivant :

```
package com.mycompany;

      import com.ibm.websphere.objectgrid.plugins.BackingMapLifecycleListener;
      import com.ibm.websphere.objectgrid.plugins.BackingMapLifecycleListener.LifecycleEvent;
      ...

      public class MyLoader implements Loader, ObjectGridPlugin, ObjectGridLifecycleListener{
          ...
          public void backingMapStateChanged(LifecycleEvent event) {
              switch(event.getState()) {
              case NEW:
              case DESTROYED:
              case DESTROYING:
              case INITIALIZING:
                  break;
              case INITIALIZED:
                  // Rechercher un plug-in MapSerializerPlugin en utilisant
                  // OSGi ou directement depuis l'instance ObjectGrid.
                  lookupOtherPlugins()
                  break;
              case STARTING:
              case PRELOAD:
                  break;
              case ONLINE:
                  if (event.isWritable()) {
                      startupProcessingForPrimary();
                  } else {
                      startupProcessingForReplica();
                  }
                  break;
              case QUIESCE:
                  if (event.isWritable()) {
```

```
                    quiesceProcessingForPrimary();
                } else {
                    quiesceProcessingForReplica();
                }
                break;
            case OFFLINE:
                shutdownShardComponents();
                break;
            }
        }
        ...
    }
```

Résultats

En implémentant l'interface ObjectGridPlugin ou BackingMapPlugin, eXtreme Scale peut contrôler le cycle de vie du plug-in au moment opportun.

En implémentant l'interface ObjectGridLifecycleListener ou BackingMapLifecycleListener, le plug-in est enregistré automatiquement comme programme d'écoute des événements de cycle de vie ObjectGrid ou BackingMap associés. L'événement INITIALIZING permet de signaler que tous les plug-ins ObjectGrid et BackingMap ont été initialisés et peuvent être recherchés et utilisés. L'événement ONLINE est utilisé pour signaler que ObjectGrid est en ligne et prêt à commencer le traitement des événements.

Configuration des plug-ins eXtreme Scale avec OSGi Blueprint

Tous les plug-ins eXtreme Scale ObjectGrid et BackingMap peuvent être définis comme beans et services OSGi en utilisant le service OSGi Blueprint disponible avec Eclipse Gemini ou Apache Aries.

Avant de commencer

Pour pouvoir configurer vos plug-ins comme services OSGi, vous devez regrouper les plug-ins dans un ensemble OSGi et connaître les concepts de base des plug-ins requis. L'ensemble doit importer les modules client ou serveur WebSphere eXtreme Scale et d'autres packages dépendants nécessaires aux plug-ins ou créer une dépendance d'ensemble dans les ensembles de serveur ou de client eXtreme Scale. Cette rubrique explique comment configurer le fichier XML Blueprint XML pour créer des beans de plug-in et les exposer comme services OSGi pour que eXtreme Scale les utilise.

Pourquoi et quand exécuter cette tâche

Les beans et services sont définis dans un fichier XML Blueprint et le conteneur Blueprint découvre, crée et interconnecte les beans et les expose comme services. Le processus rend les beans accessibles aux autres ensembles OSGi, y compris les ensembles de serveur et de client eXtreme Scale.

Lors de la création de services de plug-in personnalisés pour les utiliser avec eXtreme Scale, l'ensemble qui doit héberger les plug-ins doit être configuré pour utiliser Blueprint. En outre, un fichier XML Blueprint doit être créé et stocké dans l'ensemble. Lisez la rubrique relative à la création d'applications OSGi avec la spécification Blueprint Container qui décrit de manière générale la spécification.

Procédure

1. Créez un fichier XML Blueprint. Attribuez-lui un nom de votre choix. Toutefois, vous devez inclure l'espace de nom Blueprint :

```
<?xml version="1.0" encoding="UTF-8"?>
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">
...
</blueprint>
```

2. Créez des définitions de bean dans le fichier XML Blueprint pour chaque plug-in eXtreme Scale.

   Les beans sont définis en utilisant l'élément `<bean>`, ils peuvent être connectés à d'autres références de bean et ils peuvent inclure des paramètres d'initialisation.

   Important : Lors de la définition d'un bean, vous devez utiliser la portée correcte. Blueprint prend en charge les portées singleton et prototype. eXtreme Scale prend également en charge une portée de fragment personnalisée.

   Définissez la plupart des plug-ins eXtreme Scale comme prototype ou beans à portée de fragment, car tous les beans doivent être uniques pour chaque fragment ObjectGrid ou instance BackingMap auquel ou à laquelle ils sont associés. Les beans à portée de fragment peuvent être utiles lorsque vous utilisez les beans dans d'autres contextes pour pouvoir extraire l'instance correcte.

   Pour définir un bean à portée prototype, utilisez l'attribut `scope="prototype"` sur le bean :

   ```
   <bean id="myPluginBean" class="com.mycompany.MyBean" scope="prototype">
   ...
   </bean>
   ```

   Pour définir un beans à portée de fragment, vous devez ajouter l'espace de nom `objectgrid` au schéma XML et utiliser l'attribut `scope="objectgrid:shard"` sur le bean :

   ```
   <?xml version="1.0" encoding="UTF-8"?>

   <blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
              xmlns:objectgrid="http://www.ibm.com/schema/objectgrid"

       xsi:schemaLocation="http://www.ibm.com/schema/objectgrid
                   http://www.ibm.com/schema/objectgrid/objectgrid.xsd">

       <bean id="myPluginBean" class="com.mycompany.MyBean"
      scope="objectgrid:shard">
       ...
       </bean>

   ...
   ```

3. Créez des définitions de bean PluginServiceFactory pour chaque bean de plug-in. Tous les beans eXtreme Scale doivent avoir un bean PluginServiceFactory défini pour que la portée de bean correcte puisse être appliquée. eXtreme Scale inclut une fabrique BlueprintServiceFactory que vous pouvez utiliser. Elle contient deux propriétés que vous devez définir. Vous devez affecter à la propriété blueprintContainer la référence `blueprintContainer` et attribuer à la propriété beanId le nom de l'identificateur du bean. Lorsque eXtreme Scale recherche le service pour instancier les beans appropriés, le serveur recherche l'instance du composant bean en utilisant le conteneur Blueprint.

   ```
   bean id="myPluginBeanFactory"
       class="com.ibm.websphere.objectgrid.plugins.osgi.BluePrintServiceFactory">
       <property name="blueprintContainer" ref="blueprintContainer"/>
       <property name="beanId" value="myPluginBean" />
   </bean>
   ```

4. Créez un gestionnaire de service pour chaque bean PluginServiceFactory. Chaque gestionnaire de service expose le bean PluginServiceFactory en utilisant

l'élément <service>. L'élément de service identifie le nom à exposer à OSGi, la référence au bean PluginServiceFactory et l'interface à exposer, ainsi que le classement du service. eXtreme Scale utilise le classement du gestionnaire de service pour effectuer des mises à niveau de service lorsque la grille eXtreme Scale est active. Si le classement n'est pas défini, l'infrastructure OSGi utilise le classement 0 par défaut. Consultez la rubrique relative à la mise à jour des classements de service pour plus d'informations.

Blueprint contient diverses options de configuration des gestionnaires de service. Pour définir un gestionnaire de service simple pour un bean PluginServiceFactory, créez un élément <service> pour chaque bean PluginServiceFactory :

```
<service ref="myPluginBeanFactory"
    interface="com.ibm.websphere.objectgrid.plugins.osgi.PluginServiceFactory"
    ranking="1">
</service>
```

5. Stockez le fichier XML Blueprint dans l'ensemble de plug-ins. Le fichier XML Blueprint doit être stocké dans le répertoire OSGI-INF/blueprint du conteneur Blueprint pour être découvert.

   Pour stocker le fichier XML Blueprint dans un répertoire différent, vous devez définir l'en-tête de manifeste Bundle-Blueprint suivant :

   ```
   Bundle-Blueprint: OSGI-INF/blueprint.xml
   ```

Résultats

Les plug-ins eXtreme Scale sont maintenant configurés pour être exposés dans un conteneur OSGi Blueprint. En outre, le fichier XML descripteur ObjectGrid est configuré pour référencer les plug-ins en utilisant le service OSGi Blueprint.

Installation et démarrage des plug-ins OSGi

Dans cette tâches, vous installez l'ensemble de plug-in dynamique dans l'infrastructure OSGi, puis vous démarrez le plug-in.

Avant de commencer

Cette rubrique suppose que vous avez exécuté les tâches suivantes :

- Vous avez installé l'ensemble serveur ou client eXtreme Scale dans l'infrastructure OSGi Eclipse Equinox. Voir «Installation des ensembles eXtreme Scale», à la page 184.
- Vous avez implémenté un ou plusieurs plug-ins dynamiques BackingMap ou ObjectGrid. Voir «Génération de plug-ins dynamiques eXtreme Scale», à la page 186.
- Vous avez regroupé les plug-ins dynamiques comme services OSGi dans des ensembles OSGi.

Pourquoi et quand exécuter cette tâche

Cette tâche explique comment installer l'ensemble en utilisant la console Eclipse Equinox. L'ensemble peut être installé en utilisant plusieurs méthodes différentes, y compris en modifiant le fichier de configuration config.ini. Les produits qui intègrent Eclipse Equinox incluent des méthodes alternatives de gestion des ensembles. Pour plus d'informations sur l'ajout d'ensembles dans le fichier config.ini dans Eclipse Equinox, voir les options d'exécution Eclipse.

OSGi permet de démarrer les ensembles ayant des services dupliqués. WebSphere eXtreme Scale utilise le dernier classement de service. Lors du démarrage de

plusieurs infrastructures OSGi dans une grille de données eXtreme Scale, vous devez veiller à démarrer les classements de service corrects sur chaque serveur afin que la grille ne soit pas démarrée en utilisant une combinaison de versions différentes.

Pour identifier les versions utilisées par la grille de données, utilisez l'utilitaire xscmd pour vérifier les classements en cours et disponibles. Pour plus d'informations sur les classements de service disponibles, voir Mise à jour des services OSGi pour les plug-ins eXtreme Scale avec **xscmd**.

Procédure

Installez l'ensemble de plug-in dans l'infrastructure OSGi Eclipse Equinox en utilisant la console OSGi.

1. Démarrez l'infrastructure Eclipse Equinox avec la console activée, par exemple :
   ```
   <java_home>/bin/java -jar <equinox_root>/plugins/org.eclipse.osgi_3.6.1.R36x_v20100806.jar -console
   ```
2. Installez l'ensemble de plug-in dans la console Equinox.
   ```
   osgi> install file:///<path to bundle>
   ```
   Equinox affiche l'ID du nouvel ensemble installé :
   ```
   Bundle id is 17
   ```
3. Entrez la ligne suivante pour démarrer l'ensemble dans la console Equinox, où <id> est l'ID d'ensemble affecté lors de l'installation de l'ensemble :
   ```
   osgi>  install <id>
   ```
4. Extrayez l'état du service dans la console Equinox pour vérifier que l'ensemble a démarré :
   ```
   osgi> ss
   ```
   Lorsque l'ensemble a démarré correctement, il affiche l'état ACTIVE, par exemple :
   ```
   17      ACTIVE      com.mycompany.plugin.bundle_VRM
   ```

Installez l'ensemble de plug-in dans l'infrastructure OSGi Eclipse Equonix en utilisant le fichier config.ini file.

5. Copiez l'ensemble de plug-in dans le répertoire Eclipse Equinox plug-ins, par exemple :
   ```
   <equinox_root>/plugins
   ```
6. Modifiez le fichier de configuration Eclipse Equinox `config.ini` et ajoutez l'ensemble à la propriété osgi.bundles, par exemple :
   ```
   osgi.bundles=\
   org.eclipse.osgi.services_3.2.100.v20100503.jar@1:start, \
   org.eclipse.osgi.util_3.2.100.v20100503.jar@1:start, \
   org.eclipse.equinox.cm_1.0.200.v20100520.jar@1:start, \
   com.mycompany.plugin.bundle_VRM.jar@1:start
   ```
   Important : Vérifiez qu'il existe une ligne blanche après le dernier nom d'ensemble. Chaque ensemble est séparé par une virgule.
7. Démarrez l'infrastructure Eclipse Equinox avec la console activée, par exemple :
   ```
   <java_home>/bin/java -jar <equinox_root>/plugins/org.eclipse.osgi_3.6.1.R36x_v20100806.jar -console
   ```
8. Extrayez l'état de service dans la console Equinox pour vérifier que l'ensemble est démarré. Par exemple :
   ```
   osgi> ss
   ```

Une fois l'ensemble démarré, il affiche l'état ACTIVE. Par exemple :

```
17      ACTIVE      com.mycompany.plugin.bundle_VRM
```

Résultats

L'ensemble de plug-in est maintenant installé et démarré. Le conteneur ou le client peut être maintenant démarré eXtreme Scale. Pour plus d'informations sur le développement des plug-ins eXtreme Scale, voir la rubrique API système et plug-ins.

Exécution de conteneurs eXtreme Scale avec des plug-ins dynamiques dans un environnement OSGi

Si l'application est hébergée dans l'infrastructure OSGi Eclipse Equinox avec Eclipse Gemini ou Apache Aries, vous pouvez utiliser cette tâche pour installer et configurer l'application WebSphere eXtreme Scale dans OSGi.

Avant de commencer

Avant de démarrer cette tâche, procédez comme suit :

- Installez l'infrastructure OSGi Eclipse Equinox avec Eclipse Gemini
- Créez et exécutez les plug-ins dynamiques eXtreme Scale pour les utiliser dans un environnement OSGi

Pourquoi et quand exécuter cette tâche

Avec les plug-ins dynamiques, vous pouvez mettre à niveau dynamiquement les plug-ins lorsque la grille est active. Cela vous permet de faire évoluer une application sans redémarrer les processus de conteneur de la grille. Pour plus d'informations sur le développement de plug-ins eXtreme Scale, voir API système et plug-ins.

Procédure

1. Configurez les plug-ins OSGi en utilisant le fichier XML descripteur ObjectGrid.
2. Démarrez les serveurs de conteneur eXtreme Scale en utilisant l'infrastructure OSGi Eclipse Equinox.
3. Administrez les services OSGi pour les plug-ins eXtreme Scale avec l'utilitaire xscmd.
4. Configurez les serveurs avec OSGi Blueprint.

Configuration des plug-ins OSGi en utilisant le fichier descripteur XML ObjectGrid

Dans cette tâche, vous ajoutez des services OSGi existants à un fichier XML descripteur pour que les conteneurs WebSphere eXtreme Scale puissent reconnaître et charger correctement les plug-ins OSGi.

Avant de commencer

Pour configurer vos plug-ins, veillez à :

- Créer le module et activer les plug-ins dynamiques du déploiement OSGi.
- Disposer des noms des services OSGi qui représentent les plug-ins.

Pourquoi et quand exécuter cette tâche

Vous avez créé un service OSGi pour encapsuler le plug-in. Maintenant, ces services doivent être définis dans le fichier `objectgrid.xml` pour que les conteneurs eXtreme Scale puissent charger et configurer le ou les plug-ins

Procédure

1. Les plug-ins de grille, tels que TransactionCallback, doivent être définis sous l'élément objectGrid. Voir l'exemple suivant du fichier `objectgrid.xml` :

```
<?xml version="1.0" encoding="UTF-8"?>

<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
    xmlns="http://ibm.com/ws/objectgrid/config">

    <objectGrids>
        <objectGrid name="MyGrid" txTimeout="60">
            <bean id="myTranCallback" osgiService="myTranCallbackFactory"/>
            ...
        </objectGrid>
        ...
    </objectGrids>
    ...
/objectGridConfig>
```

   Important : La valeur d'attribut `osgiService` doit correspondre à la valeur d'attribut `ref` définie dans le fichier XML `blueprint`, où le service a été défini pour myTranCallback PluginServiceFactory.

2. Les plug-ins de mappe, tels que les chargeurs ou les sérialiseurs, doivent être définis dans l'élément backingMapPluginCollections et référencés depuis l'élément backingMap. Voir l'exemple suivant du fichier `objectgrid.xml` :

```
<?xml version="1.0" encoding="UTF-8"?>

objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
    xmlns="http://ibm.com/ws/objectgrid/config">
    <objectGrids>
        <objectGrid name="MyGrid" txTimeout="60">
            <backingMap name="MyMap1" lockStrategy="PESSIMISTIC"
                copyMode="COPY_TO_BYTES" nullValuesSupported="false"
                pluginCollectionRef="myPluginCollectionRef1"/>
            <backingMap name="MyMap2" lockStrategy="PESSIMISTIC"
                copyMode="COPY_TO_BYTES" nullValuesSupported="false"
                pluginCollectionRef="myPluginCollectionRef2"/>
            ...
        </objectGrid>
        ...
    </objectGrids>
    ...
    <backingMapPluginCollections>
        <backingMapPluginCollection id="myPluginCollectionRef1">
            <bean id="MapSerializerPlugin" osgiService="mySerializerFactory"/>
        </backingMapPluginCollection>
        <backingMapPluginCollection id="myPluginCollectionRef2">
            <bean id="MapSerializerPlugin" osgiService="myOtherSerializerFactory"/>
            <bean id="Loader" osgiService="myLoader"/>
        </backingMapPluginCollection>
        ...
    </backingMapPluginCollections>
    ...
</objectGridConfig>
```

Résultats

Le fichier `objectgrid.xml` dans cet exemple demande à eXtreme Scale de créer la grille `MyGrid` avec les deux mappes `MyMap1` et `MyMap2`. La mappe `MyMap1` utilise le sérialiseur encapsulé par le service OSGi, mySerializerFactory. La mappe `MyMap2` utilise un sérialiseur depuis le service OSGi, myOtherSerializerFactory, et un chargeur depuis le service OSGi, myLoader.

Démarrage des serveurs eXtreme Scale en utilisant l'infrastructure OSGi Eclipse Equinox

Les serveurs de conteneur WebSphere eXtreme Scale peuvent être démarrés dans une infrastructure OSGi Eclipse Equinox en utilisant plusieurs méthodes.

Avant de commencer

Pour pouvoir démarrer un conteneur eXtreme Scale, vous devez exécuter les tâches suivantes :

1. L'ensemble de serveur WebSphere eXtreme Scale doit être installé dans Eclipse Equinox.
2. L'application doit être placée dans un ensemble OSGi.
3. Les plug-ins WebSphere eXtreme Scale (s'il en existe) doivent être placés dans un ensemble OSGi. Ils peuvent se trouver dans le même ensemble que l'application ou dans des ensembles séparés.

Pourquoi et quand exécuter cette tâche

Cette tâche explique comment démarrer un serveur de conteneur eXtreme Scale dans une infrastructure OSGi Eclipse Equinox. Vous pouvez utiliser n'importe laquelle des méthodes suivantes pour démarrer les serveurs de conteneur en utilisant l'implémentation Eclipse Equinox :

- Service OSGi Blueprint

  Vous pouvez inclure toute la configuration et toutes les métadonnées dans un ensemble OSGi. Voir l'illustration suivante pour comprendre le processus Eclipse Equinox de cette méthode :

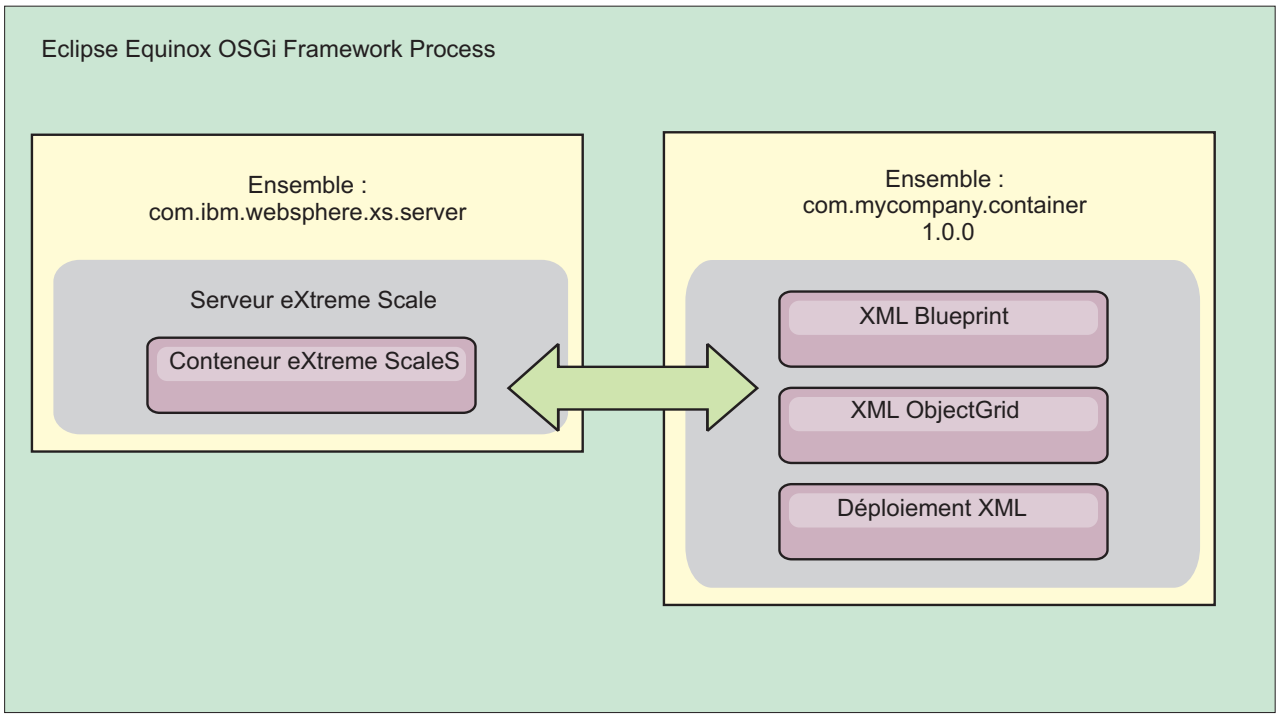


Figure 56. Processus Eclipse Equinox pour inclure toute la configuration et toutes les métadonnées dans un ensemble OSGi

- Service Admin de configuration OSGi

Vous pouvez définir la configuration et les métadonnées en dehors d'un ensemble OSGi. Voir l'image suivante pour comprendre le processus Eclipse Equinox pour cette méthode :

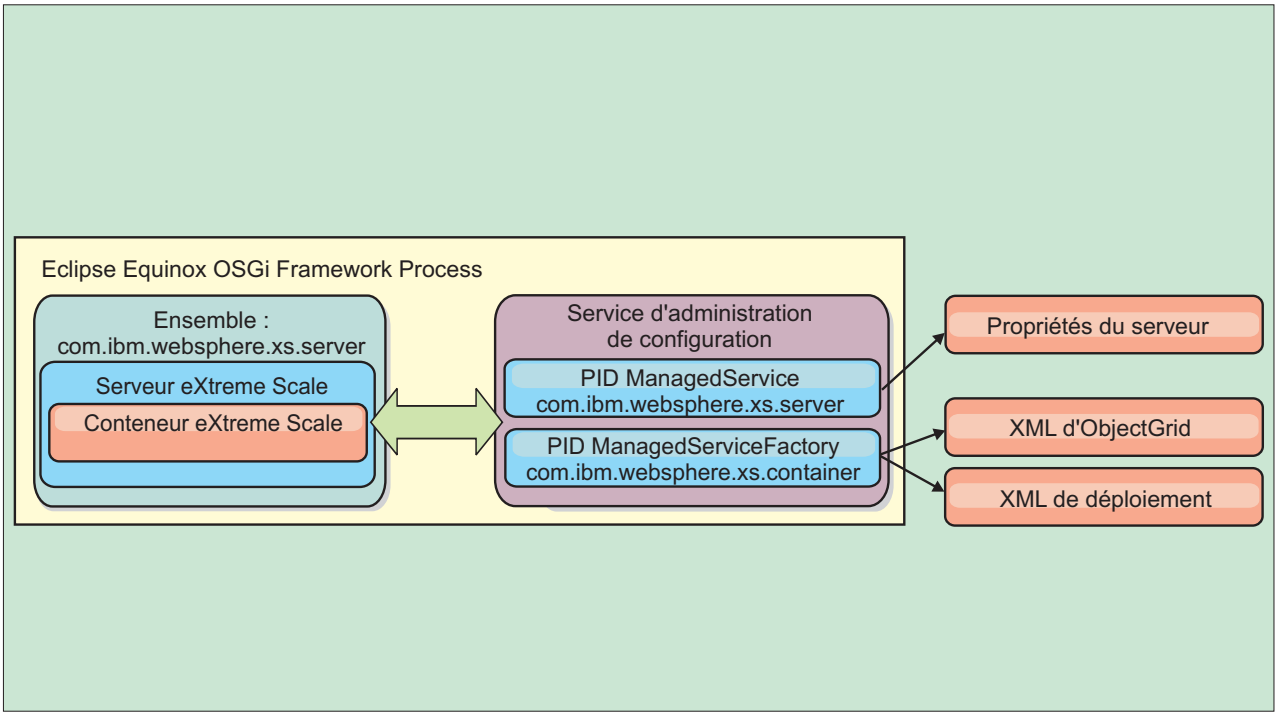


Figure 57. Processus Eclipse Equinox pour définir la configuration et les métadonnées en dehors d'un ensemble OSGi

- A l'aide d'un programme
  Prend en charge les solutions de configuration personnalisées.

Dans chaque cas, un singleton de serveur eXtreme Scale est configuré et un ou plusieurs conteneurs sont configurés.

L'ensemble de serveur eXtreme Scale, `objectgrid.jar`, contient toutes les bibliothèques nécessaires pour démarrer et exécuter un conteneur de grille eXtreme Scale dans une infrastructure OSGi. L'environnement d'exécution du serveur communique avec les plug-ins fournis par l'utilisateur et les objets de données en utilisant le gestionnaire de service OSGi.

Important : Après que l'ensemble de serveur eXtreme Scale a été démarré et le serveur eXtreme Scale initialisé, il ne peut pas être redémarré. Le processus Eclipse Equinox doit être redémarré pour redémarrer le serveur eXtreme Scale.

Vous pouvez utiliser le support eXtreme Scale pour l'espace de nom Spring pour configurer les serveurs de conteneur eXtreme Scale dans un fichier XML Blueprint. Lorsque les éléments XML de serveur et de conteneur sont ajoutés au fichier XML Blueprint, le gestionnaire d'espace de nom eXtreme Scale démarre automatiquement un serveur de conteneur en utilisant les paramètres définis dans le fichier XML Blueprint lors du démarrage de l'ensemble. Le gestionnaire arrête le conteneur lorsque l'ensemble s'arrête.

Pour configurer les serveurs de conteneur eXtreme Scale avec XML Blueprint, procédez comme suit :

Procédure

- Démarrez un serveur de conteneur eXtreme Scale en utilisant OSGi Blueprint.
  1. Créez un ensemble de conteneur.
  2. Installez l'ensemble de conteneur dans l'infrastructure OSGi Eclipse Equinox. Voir «Installation et démarrage des plug-ins OSGi», à la page 192.
  3. Démarrez l'ensemble de conteneur.
- Démarrez un serveur de conteneur eXtreme Scale en utilisant l'administrateur de configuration OSGi.
  1. Configurez le serveur et le conteneur en utilisant l'administrateur de configuration.
  2. Lorsque l'ensemble de serveur eXtreme Scale est démarré ou que les PID (persistent identifier) sont créé avec l'administrateur de configuration, le serveur et le conteneur démarrent automatiquement.
- Démarrez un serveur de conteneur eXtreme Scale en utilisant l'API ServerFactory. Voir la documentation d'API de serveur.
  1. Créez une classe d'activateur d'ensemble OSGi et utilisez l'API eXtreme Scale ServerFactory pour démarrer un serveur.

Administration des services OSGi en utilisant l'utilitaire `xscmd`

Vous pouvez utiliser l'utilitaire **xscmd** pour exécuter des tâches d'administration, telles qu'afficher les serveurs et leurs classements utilisés par chaque conteneur, et mettre à niveau l'environnement d'exécution pour utiliser les nouvelles versions des ensembles.

Pourquoi et quand exécuter cette tâche

Avec l'infrastructure Eclipse Equinox OSGi, vous pouvez installer plusieurs versions d'un même ensemble et vous pouvez mettre à jour ces ensemble lors de l'exécution. WebSphere eXtreme Scale est un environnement distribué qui exécute les serveurs de conteneur dans une multitude d'instances de l'infrastructure OSGi.

Les administrateurs doivent copier, installer et démarrer manuellement les ensembles dans l'infrastructure OSGi. eXtreme Scale contient un personnaliseur ServiceTrackerCustomizer OSGi pour suivre les services identifiés comme plug-ins eXtreme Scale dans le fichier XML descripteur. Utilisez l'utilitaire **xscmd** pour valider la version utilisée du plug-in, les versions pouvant être utilisées et exécuter des mises à niveau d'ensemble.

eXtreme Scale utilise le numéro de classement de service pour identifier la version de chaque service. Lorsque au moins deux services sont chargés avec la même référence, eXtreme Scale utilise automatiquement le service ayant le classement le plus élevé.

Procédure

- Exécutez la commande **osgiCurrent** et vérifiez que chaque serveur eXtreme Scale utilise le classement de service de plug-in correct.

  Comme eXtreme Scale choisit automatiquement la référence de service ayant le classement le plus élevé, il se peut que la grille de données démarre avec plusieurs classements d'un service de plug-in.

  Si la commande détecte une discordance de classements ou qu'elle ne trouve pas un service, un niveau d'erreur différent de zéro est défini. Si la commande aboutit, le niveau d'erreur 0 est défini.

L'exemple suivant montre la sortie de la commande **osgiCurrent** lorsque deux plus-ins sont installés dans une grille sur quatre serveurs. Le plug-in loaderPlugin utilise le classement 1 et le plug-in txCallbackPlugin, le classement 2.

```
OSGi Service Name Current Ranking ObjectGrid Name MapSet Name Server Name
----------------- --------------- --------------- ----------- -----------
loaderPlugin      1               MyGrid          MapSetA     server1
loaderPlugin      1               MyGrid          MapSetA     server2
loaderPlugin      1               MyGrid          MapSetA     server3
loaderPlugin      1               MyGrid          MapSetA     server4
txCallbackPlugin  2               MyGrid          MapSetA     server1
txCallbackPlugin  2               MyGrid          MapSetA     server2
txCallbackPlugin  2               MyGrid          MapSetA     server3
txCallbackPlugin  2               MyGrid          MapSetA     server4
```

L'exemple suivant montre la sortie de la commande **osgiCurrent** lorsque le serveur 2 a été démarré avec un nouveau classement du plug-in loaderPlugin :

```
OSGi Service Name Current Ranking ObjectGrid Name MapSet Name Server Name
----------------- --------------- --------------- ----------- -----------
loaderPlugin      1               MyGrid          MapSetA     server1
loaderPlugin      2               MyGrid          MapSetA     server2
loaderPlugin      1               MyGrid          MapSetA     server3
loaderPlugin      1               MyGrid          MapSetA     server4
txCallbackPlugin  2               MyGrid          MapSetA     server1
txCallbackPlugin  2               MyGrid          MapSetA     server2
txCallbackPlugin  2               MyGrid          MapSetA     server3
txCallbackPlugin  2               MyGrid          MapSetA     server4
```

- Exécutez la commande **osgiAll** pour vérifier que les services de plug-in ont été correctement démarrés sur chaque serveur de conteneur eXtreme Scale.

  Lorsque des ensembles contenant des services référencés par une configuration ObjectGrid démarrent, l'environnement d'exécution eXtreme Scale suit le plug-in, mais il ne l'utilise pas immédiatement. La commande **osgiAll** montre les plug-ins disponibles pour chaque serveur.

  Lorsqu'elle est exécutée sans paramètres, tous les services de toutes les grilles et de tous les serveurs sont indiqués. Des filtres supplémentaires, notamment le filtre **-serviceName** <service_name>, peuvent être définis pour limiter la sortie à un seul service ou sous-ensemble de la grille de données.

  L'exemple suivant montre la sortie de la commande **osgiAll** lorsque deux plug-ins sont démarrés sur deux serveurs. Les classements 1 et 2 du plug-in loaderPlugin sont démarrés et le classement 1 du plug-in txCallbackPlugin est démarré. Le résumé à la fin de la sortie indique que les deux serveurs voient les mêmes classements de service :

```
Server: server1
   OSGi Service Name   Available Rankings
   -----------------   ------------------
   loaderPlugin        1, 2
   txCallbackPlugin    1

Server: server2
   OSGi Service Name   Available Rankings
   -----------------   ------------------
   loaderPlugin        1, 2
   txCallbackPlugin    1

Summary - All servers have the same service rankings.
```

  L'exemple suivant montre la sortie de la commande **osgiAll** lorsque l'ensemble qui contient le plug-in loaderPlugin avec le classement 1 est arrêté sur le serveur 1. Le résumé à la fin de la sortie indique que le serveur n'a pas le plug-in loaderPlugin avec le classement 1 :

```
Server: server1
   OSGi Service Name   Available Rankings
   -----------------   ------------------
   loaderPlugin        2
   txCallbackPlugin    1

Server: server2
   OSGi Service Name   Available Rankings
   -----------------   ------------------
   loaderPlugin        1, 2
   txCallbackPlugin    1

Summary - The following servers are missing service rankings:
   Server  OSGi Service Name Missing Rankings
   ------  ----------------- ----------------
   server1 loaderPlugin      1
```

L'exemple suivant montre la sortie si le nom de service est défini avec l'argument **-sn** et que le service n'existe pas.

```
Server: server2
   OSGi Service Name Available Rankings
   ----------------- ------------------
   invalidPlugin     No service found

Server: server1
   OSGi Service Name Available Rankings
   ----------------- ------------------
   invalidPlugin     No service found

Summary - All servers have the same service rankings.
```

- Exécutez la commande **osgiCheck** pour vérifier les groupes de services de plug-in et de classements s'ils sont disponibles.

  La commande **osgiCheck** accepte un ou plusieurs groupes de classements de service de la manière suivante `-serviceRankings <service name>;<ranking>[,<serviceName>;<ranking>]`

  Lorsque les classements sont tous disponibles, la méthode retourne un niveau d'erreur 0. Si un ou plusieurs classements sont indisponibles, un niveau d'erreur différent de zéro et la table de tous les serveurs qui ne contiennent pas les classements de service définis sont spécifiés. Des filtres supplémentaires peuvent être utilisés pour limiter la vérification des services à un sous-ensemble des serveurs disponibles dans le domaine eXtreme Scale.

  Par exemple, si le classement ou le service est absent, le message suivant s'affiche :

```
Server  OSGi Service Unavailable Rankings
------  ------------ --------------------
server1 loaderPlugin 3
server2 loaderPlugin 3
```

- Exécutez la commande **osgiUpdate** pour mettre à jour le classement d'un ou de plusieurs plug-ins pour tous les serveurs dans un seul ObjectGrid et MapSet dans une seule opération.

  La commande accepte un ou plusieurs groupes de classements de service de la manière suivante : `-serviceRankings <service name>;<ranking>[,<serviceName>;<ranking>] -g <grid name> -ms <mapset name>`

  Avec cette commande, vous pouvez exécuter les opérations suivantes :

  – Vérifier que les services spécifiés sont disponibles pour la mise à niveau sur chacun des serveurs.

  – Mettre la grille hors ligne en utilisant l'interface StateManager. Pour plus d'informations, voir Gestion de la disponibilité ObjectGrid. Ce processus met

au repos la grille et attend la fin des transactions en cours en interdisant le démarrage de nouvelles transactions. Ce processus indique également aux programmes d'écoute ObjectGridLifecycleListener et BackingMapLifecycleListener d'arrêter toute activité transactionnelle. Voir Plug-in de programme d'écoute d'événement pour plus d'informations sur les plug-ins de programme d'écoute.

– Mettre à jour chaque conteneur eXtreme Scale exécuté dans une infrastructure OSGi pour utiliser les nouvelles versions de service.
– Mettre la grille en ligne pour reprendre l'exécution des transactions.

Le processus de mise à jour est idempotent de sortie que si un client n'exécute pas une tâche, l'opération est annulée. Si un client ne peut pas exécuter l'annulation ou qu'il est interrompu pendant la mise à jour, la même commande peut être réexécutée et elle reprend à l'étape appropriée.

Si le client ne peut pas continuer et que le processus est redémarré depuis un autre client, utilisez l'option `-force` pour permettre au client d'exécuter la mise à jour. La commande **`osgiUpdate`** empêche plusieurs client de mettre à jour simultanément un même groupe de mappes. Pour plus d'informations sur la commande **`osgiUpdate`**, voir Mise à jour des services OSGi pour les plug-ins eXtreme Scale avec **`xscmd`**.

Configuration des serveurs avec OSGi Blueprint

Vous pouvez configurer les serveurs de conteneur WebSphere eXtreme Scale en utilisant un fichier XML OSGi Blueprint qui permet de simplifier le regroupement et le développement d'ensembles de serveur autonomes.

Avant de commencer

Cette rubrique suppose que vous avez exécuté les tâches suivantes :

- L'infrastructure OSGi Eclipse Equinox a été installée et démarrée avec le conteneur Eclipse Gemini ou Apache Aries Blueprint.
- L'ensemble de serveur eXtreme Scale a été installé et démarré.
- L'ensemble de plug-ins dynamiques eXtreme Scale a été créé.
- Le fichier XML descripteur eXtreme Scale ObjectGrid et le fichier XML de stratégie de déploiement ont été créés.

Pourquoi et quand exécuter cette tâche

Cette tâche explique comment configurer un serveur eXtreme Scale avec un conteneur en utilisant un fichier XML Blueprint. Le résultat de la procédure est un ensemble de conteneur. Lorsque l'ensemble de conteneur est démarré, l'ensemble de serveur eXtreme Scale suit l'ensemble, analyse le fichier XML de serveur et démarre un serveur et un conteneur.

Un ensemble de conteneur peut être éventuellement combiné à l'application et aux plug-ins eXtreme Scale lorsque des mises à jour de plug-ins dynamiques ne sont pas nécessaires ou que les plug-ins ne prennent pas en charge la mise à jour dynamique.

Procédure

1. Créez un fichier XML Blueprint avec l'espace de nom `objectgrid` inclut. Vous pouvez affecter le nom de votre choix au fichier. Toutefois, il doit inclure l'espace de nom Blueprint :

```
<?xml version="1.0" encoding="UTF-8"?>

<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
           xmlns:objectgrid="http://www.ibm.com/schema/objectgrid"
           xsi:schemaLocation="http://www.ibm.com/schema/objectgrid
                http://www.ibm.com/schema/objectgrid/objectgrid.xsd">
...
</blueprint>
```

2. Ajoutez la définition XML du serveur eXtreme Scale avec les propriétés de serveur appropriées. Voir le fichier XML descripteur Spring pour plus d'informations sur toutes les propriétés de configuration disponibles. Voir l'exemple suivant de définition de fichier XML :

```
objectgrid:server
    id="xsServer"
   tracespec="ObjectGridOSGi=all=enabled"
   tracefile="logs/osgi/wxsserver/trace.log"
    jmxport="1199"
   listenerPort="2909">
    <objectgrid:catalog host="catserver1.mycompany.com" port="2809" />
    <objectgrid:catalog host="catserver2.mycompany.com" port="2809" />
</objectgrid:server>
```

3. Ajoutez la définition XML du conteneur eXtreme Scale avec la référence à la définition de serveur et les fichiers XML descripteur d'ObjectGrid et de déploiement d'ObjectGrid regroupés dans l'ensemble. Par exemple :

```
<objectgrid:container id="container"
    objectgridxml="/META-INF/objectGrid.xml"
    deploymentxml="/META-INF/objectGridDeployment.xml"
    server="xsServer" />
```

4. Stockez le fichier XML Blueprint dans l'ensemble de conteneur. Le fichier XML Blueprint doit être stocké dans le répertoire `OSGI-INF/blueprint` du conteneur Blueprint pour être trouvé.

   Pour stocker le fichier XML Blueprint dans un répertoire différent, vous devez définir l'en-tête du manifeste Bundle-Blueprint. Par exemple :

```
Bundle-Blueprint: OSGI-INF/blueprint.xml
```

5. Regroupez les fichiers dans un fichier JAR d'ensemble unique. Voir l'exemple suivant de hiérarchie de répertoires d'ensemble :

```
MyBundle.jar
    /META-INF/manifest.mf
    /META-INF/objectGrid.xml
    /META-INF/objectGridDeployment.xml
    /OSGI-INF/blueprint/blueprint.xml
```

Résultats

Un ensemble de conteneur eXtreme Scale est maintenant créé et peut être installé dans Eclipse Equinox. Lorsque l'ensemble de conteneur est démarré, l'environnement d'exécution du serveur eXtreme Scale dans l'ensemble de serveur eXtreme Scale démarre automatiquement le serveur eXtreme Scale de singleton en utilisant les paramètres définis dans l'ensemble et démarre un serveur de conteneur. L'ensemble peut être arrêté et démarré, ce qui arrête et redémarre le conteneur. Le serveur est un singleton et ne s'arrête pas lorsque l'ensemble est démarré pour la première fois.

Chapitre 4. Exemples

Plusieurs tutoriels et exemples WebSphere eXtreme Scale sont disponibles.

Exemples

Les étapes suivantes portent sur les principales fonctions WebSphere eXtreme Scale.

- Exemple d'API DataGrid
- Configuration de déploiements locaux

Exemples de communauté

Les exemples suivants expliquent comment utiliser WebSphere eXtreme Scale dans divers environnements pour montrer différentes fonctions du produit.

- **Infrastructure de services asynchrone** : elle fournit une matrice de traitement évolutive et tolérante aux pannes pour le traitement asynchrone des messages. Pour plus d'informations, et savoir comment télécharger l'exemple, voir l'exemple Galerie d'exemples : Infrastructure de service asynchrone.
- **Sécurité de l'authentification du client** : cet exemple explique comment configurer l'authentification pour demander au client de fournir des données d'identification valides pour que le serveur l'autorise à accéder à une grille. Pour plus d'informations et savoir comment télécharger l'exemple, voir l'exemple Galerie d'exemples : Sécurité de l'authentification du client
- **Création de mappes dynamiques** : cet exemple explique comment créer des mappes après l'initialisation de la grille. Pour eXtreme Scale 7.0 et les versions suivantes, vous pouvez utiliser des canevas pour extraire les mappes. Pour plus d'informations et savoir comment télécharger l'exemple, voir l'exemple Galerie d'exemples : Création de mappes dynamiques après l'initialisation de la grille.
- **Réplication multimaître** : l'exemple relatif à l'initiation à la réplication multimaître présente rapidement la réplication multimaître. Pour plus d'informations et savoir comment télécharger l'exemple, voir l'exemple Galerie d'exemples : Exemple de réplication multimaître.
- **Requêtes avec l'API Entity Manager** : cet exemple montre comment utiliser des requêtes dans une mappe partitionnée distribuée avec l'API EntityManager. Pour plus d'informations, et savoir comment télécharger l'exemple, voir l'exemple Galerie d'exemples : Exécution de requêtes dans une grille partitionnée en utilisant l'API Entity Manager.
- **Requête parallèles avec une implémentation ReduceGridAgent** : explique comment utiliser l'API Data Grid pour exécuter une requête sur chaque partition dans la grille. Pour plus d'informations et savoir comment télécharger l'exemple, voir l'exemple Galerie d'exemples : Exécution de requêtes en parallèle en utilisant ReduceGridAgent.

Articles avec tutoriels et exemples

Tableau 9. Articles disponibles par fonction

Article	Caractéristiques
Build grid-ready apps with ObjectGrid	API ObjectMap, API EntityManager, requête, agents, Java SE et EE, statistiques, partitionnement, administration/opérations, Eclipse
Highly scalable grid-style computing and data processing with the ObjectGrid component of WebSphere Extended Deployment	API EntityManager, agents
Build a scalable, resilient, high performance database alternative with the ObjectGrid component of WebSphere Extended Deployment	API ObjectMap, réplication, partitionnement, administration/opérations, Eclipse
Enhancing xsadmin for WebSphere eXtreme Scale	Administration
User's Guide to WebSphere eXtreme Scale	Toutes les rubriques

Version d'essai gratuite

Pour vous initier à WebSphere eXtreme Scale, vous pouvez télécharger une version d'essai gratuite. Vous pourrez ainsi développer des applications innovantes à hautes performances en étendant à l'aide de fonctionnalités avancées la notion de mise en cache des données.

Version d'essai à télécharger

Vous pouvez télécharger une version d'essai gratuite de WebSphere eXtreme Scale à partir de la page de téléchargement de la version d'évaluation d'eXtreme Scale.

Après avoir téléchargé et décompressé la version d'évaluation de eXtreme Scale, accédez au répertoire `gettingstarted` et lisez le fichier `GETTINGSTARTED_README.txt`. Ce tutoriel vous permet de commencer à utiliser eXtreme Scale, créer une grille de données sur plusieurs serveurs et exécuter quelques applications simples vous permettant de stocker et d'extraire des données dans une grille. Avant de déployer eXtreme Scale dans un environnement de production, plusieurs options sont à prendre en considération : nombre de serveurs à utiliser, quantité d'espace de stockage présente sur chacun de ces serveurs, et choix de la réplication, synchrone ou asynchrone.

Exemples de fichier de propriétés

Les fichiers de propriétés serveur contiennent les paramètres d'exécution de vos serveurs de catalogues et serveurs conteneurs. Vous pouvez spécifier un fichier de propriétés serveur pour une configuration autonome ou WebSphere Application Server. Les fichiers de propriétés client contiennent les paramètres de votre client.

Vous pouvez utiliser les exemples de fichiers de propriétés suivants qui se trouvent dans le répertoire *racine_install_wxs*`\properties` pour créer votre fichier de propriétés :

- `sampleServer.properties`
- `sampleClient.properties`

Exemple : utilitaire xsadmin

Avec l'utilitaire **xsadmin**, vous pouvez formater et afficher des informations textuelles relatives à votre topologie WebSphere eXtreme Scale. Il fournit une méthode pour effectuer l'analyse syntaxique et la détection des données de déploiement actuelles et peut servir de base pour l'écriture d'utilitaires personnalisés.

Avant de commencer

- 7.1.1+ L'utilitaire **xsadmin** est fourni comme exemple de création d'utilitaires personnalisés pour votre déploiement. L'utilitaire **xscmd** est fourni comme utilitaire pris en charge pour la surveillance et l'administration de votre environnement. Pour plus d'informations, voir Administration avec l'utilitaire **xscmd**.
- Pour que l'utilitaire **xsadmin** affiche des résultats, vous devez avoir créé votre topologie de grille de données Les serveurs de catalogue et les serveurs de conteneur doivent être démarrés. Pour plus d'informations, voir Démarrage et arrêt des serveurs sécurisés.
- Vérifiez que la variable d'environnement *JAVA_HOME* est définie pour utiliser l'environnement d'exécution installé avec le produit. Si vous utilisez la version d'évaluation du produit, vous devez définir la variable d'environnement *JAVA_HOME*.

Pourquoi et quand exécuter cette tâche

L'exemple d'utilitaire **xsadmin** utilise une implémentation de beans gérés (MBeans). Cet exemple d'application de surveillance active rapidement les fonctions de surveillance intégrées que vous pouvez étendre en utilisant les interfaces dans le package com.ibm.websphere.objectgrid.management. Vous pouvez analyser le code source de l'exemple d'application **xsadmin** dans le fichier *rép_base_wxs*/samples/xsadmin.jar dans une application autonome ou dans le fichier *rép_base_wxs*/xsadmin.jar dans une installation WebSphere Application Server.

Vous pouvez utiliser l'exemple d'utilitaire **xsadmin** pour afficher la structure et l'état de la grille de données (par exemple, le contenu de la grille). Dans cet exemple, la structure de la grille de données dans cette tâche est constituée d'une seule grille de données *ObjectGridA* avec une mappe *MapA* qui appartient au groupe de mappes *MapSetA*. Cet exemple montre comment afficher tous les conteneurs actifs dans une grille de données et imprimer les mesures filtrées relatives à la taille de la mappe *MapA*. Pour afficher toutes les options de la commande, exécutez l'utilitaire **xsadmin** sans arguments ou avec l'option **-help**.

Procédure

1. Accédez au répertoire bin.

   ```
   cd rép_base_wxs/bin
   ```

2. Exécutez l'utilitaire **xsadmin**.
   - Pour afficher l'aide en ligne, exécutez la commande suivante :

     UNIX

     ```
     xsadmin.sh
     ```

     Windows

     ```
     xsadmin.bat
     ```

Vous devez envoyer une seule des options listées pour que l'utilitaire fonctionne. Si aucune option **-g** ou **-m** n'est spécifiée, l'utilitaire **xsadmin** affiche les informations pour chaque grille de la topologie.

- Pour activer les statistiques pour tous les serveurs, exécutez la commande suivante :

  UNIX

  ```
  xsadmin.sh -g ObjectGridA -setstatsspec ALL=enabled
  ```

  Windows

  ```
  xsadmin.bat -g ObjectGridA -setstatsspec ALL=enabled
  ```

- Pour afficher tous les conteneurs en ligne d'une grille, exécutez la commande suivante :

  UNIX

  ```
  xsadmin.sh -g ObjectGridA -m MapSetA -containers
  ```

  Windows

  ```
  xsadmin.bat -g ObjectGridA -m MapSetA -containers
  ```

  Toutes les informations sur les conteneurs s'affichent. Ci-après, un exemple de sortie :

  ```
  Connecting to Catalog service at localhost:1099


  *** Show all online containers for grid - ObjectGridA & mapset - MapSetA

  Host: 192.168.0.186
  Container: server1_C-0, Server:server1, Zone:DefaultZone
  Partition Shard Type
          0 Primary

  Num containers matching = 1
  Total known containers = 1
  Total known hosts = 1
  ```

  Avertissement : Pour obtenir ces informations lorsque le protocole TLS/SSL (Transport Layer Security/Secure Sockets Layer) est activé, vous devez démarrer les serveurs de catalogue et de conteneur avec l'ensemble de ports de service JMX. Pour définir le port de service JMX, vous pouvez utiliser l'option **-JMXServicePort** dans le script **startOgServer** ou appeler la méthode setJMXServicePort dans l'interface ServerProperties.

- Pour vous connecter au service de catalogue et afficher les informations sur la mappe MapA, exécutez la commande suivante :

  UNIX

  ```
  xsadmin.sh -g ObjectGridA -m MapSetA -mapsizes -fm MapA
  ```

  Windows

  ```
  xsadmin.bat -g ObjectGridA -m MapSetA -mapsizes -fm MapA
  ```

  La taille de la mappe spécifiée s'affiche. Ci-après, un exemple de sortie :

  ```
  Connecting to Catalog service at localhost:1099

  ****Displaying Results for Grid - ObjectGridA, MapSet - MapSetA*****
  ```

```
*** Listing Maps for server1 ***
 Map Name Partition Map Size Used Bytes (B) Shard Type
 MapA     0         0        0              Primary
```

- Pour vous connecter au service de catalogue à l'aide d'un port JMX spécifique et afficher des informations sur la mappe MapA, exécutez la commande suivante : UNIX

```
xsadmin.sh -g ObjectGridA -m MapSetA -mapsizes -fm MapA
  -ch CatalogMachine -p 6645
```

Windows

```
xsadmin.bat -g ObjectGridA -m MapSetA -mapsizes -fm MapA
  -ch CatalogMachine -p 6645
```

L'exemple d'utilitaire **xsadmin** se connecte au serveur MBean qui s'exécute dans un serveur de catalogue. Un serveur de catalogue peut s'exécuter comme processus autonome, processus WebSphere Application Server ou être intégré dans un processus d'application personnalisé. Utilisez l'option **-ch** pour spécifier le nom d'hôte du service de catalogue et l'option **-p** pour spécifier son port de désignation.

La taille de la mappe spécifiée s'affiche. Ci-après, un exemple de sortie :

```
Connecting to Catalog service at CatalogMachine:6645

*****Displaying Results for Grid - ObjectGridA, MapSet - MapSetA*****


*** Listing Maps for server1 ***
Map Name: MapA  Partition #: 0  Map Size: 0  Shard Type: Primary
Server Total: 0
```

- Pour vous connecter à un service de catalogue hébergé dans un processus WebSphere Application Server, procédez comme suit :

L'option **-dmgr** est obligatoire lorsque vous vous connectez à un service de catalogue hébergé par tout processus ou cluster de processus WebSphere Application Server. Utilisez l'option **-ch** pour spécifier le nom d'hôte, s'il n'est pas `localhost`, et l'option **-p** pour substituer le port d'amorce du service de catalogue, qui utilise le processus BOOTSTRAP_ADDRESS. L'option **-p** est nécessaire uniquement si BOOTSTRAP_ADDRESS n' pas la valeur 9809.

Remarque : La version autonome de WebSphere eXtreme Scale ne peut pas être utilisée pour vous connecter à un service de catalogue hébergé par un processus WebSphere Application Server. Utilisez l'utilitaire **xsadmin** dont le script est inclus dans le répertoire *racine_was*`/bin` qui est disponible lorsque vous installez WebSphere eXtreme Scale sur WebSphere Application Server or WebSphere Application Server Network Deployment.

a. Accédez au répertoire bin de WebSphere Application Server :

```
cd racine_was/bin
```

b. Lancez l'utilitaire **xsadmin** avec la commande suivante :

UNIX

```
xsadmin.sh -g ObjectGridA -m MapSetA -mapsizes -fm MapA -dmgr
```

Windows

```
xsadmin.bat -g ObjectGridA -m MapSetA -mapsizes -fm MapA -dmgr
```

La taille de la mappe spécifiée s'affiche.

```
Connecting to Catalog service at localhost:9809

****Displaying Results for Grid - ObjectGridA, MapSet - MapSetA*****
```

```
*** Listing Maps for server1 ***
Map Name: MapA  Partition #: 0  Map Size: 0  Shard Type: Primary
Server Total: 0
```

- Pour afficher le placement configuré et d'exécution de votre configuration, exécutez la commande suivante :

  ```
  xsadmin -placementStatus
  xsadmin -placementStatus -g myOG -m myMapSet
  xsadmin -placementStatus -m myMapSet
  xsadmin -placementStatus -g myOG
  ```

  Vous pouvez définir la portée de la commande pour afficher les informations de placement de l'intégralité de la configuration, une grille de données unique, un groupe de mappes unique ou une combinaison de grille de données et de groupe de mappes. Ci-après, un exemple de sortie :

```
**********Printing Placement Status for Grid - Grid, MapSet - mapSet**************

<objectGrid name="Grid" mapSetName="mapSet">
  <configuration>
    <attribute name="placementStrategy" value="FIXED_PARTITIONS"/>
    <attribute name="numInitialContainers" value="3"/>
    <attribute name="minSyncReplicas" value="0"/>
    <attribute name="developmentMode" value="true"/>
  </configuration>
  <runtime>
    <attribute name="numContainers" value="3"/>
    <attribute name="numMachines" value="1"/>
    <attribute name="numOutstandingWorkItems" value="0"/>
  </runtime>
</objectGrid>
```

Création d'un profil de configuration pour l'utilitaire `xsadmin`

Vous pouvez sauvegarder les paramètres fréquemment spécifiés de l'utilitaire **xsadmin** dans un fichier de propriétés. En conséquence, les appels de l'utilitaire **xsadmin** sont plus courts.

Avant de commencer

Créez un déploiement de base de WebSphere eXtreme Scale qui inclut au moins un serveur de catalogue et au moins un conteneur de serveur. Pour plus d'informations, voir Script **startOgServer**.

Pourquoi et quand exécuter cette tâche

Voir «Référence de l'utilitaire **xsadmin**», à la page 209 pour obtenir la liste des propriétés que vous pouvez placer dans un profil de configuration pour l'utilitaire **xsadmin**. Si vous spécifiez à la fois un fichier de propriétés et un paramètre correspondant en tant qu'argument de ligne de commande, l'argument de ligne de commande remplace la valeur du fichier de propriétés.

Procédure

1. Créez un fichier de propriétés de profil de configuration. Ce fichier de propriétés doit contenir les propriétés globales que vous souhaitez utiliser dans tous vos appels de la commande **xsadmin**.

   Enregistrez le fichier de propriétés avec le nom de votre choix. Par exemple, vous pouvez placer le fichier dans le chemin `/opt/ibm/WebSphere/wxs71/ObjectGrid/security/<my.properties>`.

   Remplacez `<my.properties>` par le nom de votre fichier. Par exemple, vous pouvez définir les propriétés suivantes dans votre fichier :

   - `XSADMIN_TRUST_TYPE=jks`

- XSADMIN_TRUST_PATH=/opt/ibm/WebSphere/wxs71/ObjectGrid/bin/security/key.jks
- XSADMIN_USERNAME=ogadmin

2. Exécutez l'utilitaire xsadmin avec le fichier de propriétés que vous avez créé. Utilisez le paramètre **-profile** pour indiquer l'emplacement de votre fichier de propriétés. Vous pouvez également utiliser le paramètre **-v** pour afficher la sortie en mode prolixe.

```
./xsadmin.sh -l -v -password xsadmin -ssl -trustPass ogpass -profile
/opt/ibm/WebSphere/wxs71/ObjectGrid/security/<my.properties>
```

Référence de l'utilitaire xsadmin

Vous pouvez passer des arguments à l'utilitaire **xsadmin** selon deux méthodes différentes : avec un argument de ligne de commande ou avec un fichier de propriétés.

Arguments de xsadmin

Vous pouvez définir un fichier de propriétés pour l'utilitaire **xsadmin** avec version 7.1 Correctif 1 ou ultérieur. En créant un fichier de propriétés, vous pouvez enregistrer certains des arguments fréquemment utilisés, tels que le nom d'utilisateur. Les propriétés que vous pouvez ajouter à un fichier de propriétés se trouvent dans le tableau suivant. Si vous spécifiez une propriété à la fois dans un fichier de propriétés et dans l'argument de ligne de commande équivalent, la valeur de l'argument de ligne de commande remplace la valeur du fichier de propriétés.

Pour plus d'informations sur la définition d'un fichier de propriétés pour l'utilitaire **xsadmin**, voir «Création d'un profil de configuration pour l'utilitaire **xsadmin**», à la page 208.

Tableau 10. Arguments de l'utilitaire **xsadmin**

Arguments de ligne de commande	Nom de la propriété équivalente dans le fichier de propriétés	Description et valeurs valides
-bp	n/a	Indique le port d'écoute. **Valeur par défaut :**2809
-ch	n/a	Indique le nom d'hôte JMX pour le serveur de catalogue. **Valeur par défaut :**localhost
-clear	n/a	Efface la mappe définie. **Permet d'utiliser les filtres suivants : -**fm
-containers	n/a	Pour chaque grille de données et chaque mappe définies, affiche une liste des serveurs de conteneurs. **>Permet d'utiliser les filtres suivants : -**fnp
-continuous	n/a	Spécifiez cet indicateur si vous voulez des résultats de taille de mappe en continu pour surveiller la grille de données. Lorsque vous exécutez cette commande avec l'argument **-mapsizes** , la taille de la mappe s'affiche toutes les 20 secondes.
-coregroups	n/a	Affiche tous les groupes centraux pour le serveur de catalogue. Cet argument est utilisé pour des diagnostics avancés.
-dismissLink *<domaine_service_catalogue>*	n/a	Supprime un lien entre 2 domaine de services de catalogue. Spécifiez le nom du domaine de services de catalogue étranger auquel vous vous êtes connecté antérieurement avec l'argument **-establishLink**.
-dmgr	n/a	Indique si vous êtes connecté à un service de catalogue hébergé par WebSphere Application Server. **Par défaut :**false

Tableau 10. Arguments de l'utilitaire **xsadmin** *(suite)*

Arguments de ligne de commande	Nom de la propriété équivalente dans le fichier de propriétés	Description et valeurs valides
-empties	n/a	Spécifiez cet indicateur si vous voulez afficher les conteneurs vides dans les résultats.
-establishLink *<nom_domaine_étranger> <hôte1:port1,hôte2:port2...>*	n/a	Connecte le domaine de services de catalogue à un domaine de services de catalogue étranger. Utilisez le format suivant : `-establishLink <nom_domaine_étranger> <hôte1:port1,hôte2:port2...>`. *nom_domaine_étranger* est le nom du domaine de services de catalogue étranger et *hôte1:port1,hôte2:port2...* est une liste séparée par des virgules de noms d'hôte de serveur de catalogue et de ports ORB (Object Request Broker) qui s'exécutent dans ce domaine de services de catalogue.
-fc	n/a	Filtre pour ce conteneur uniquement. Si vous effectuez le filtrage des serveurs de conteneurs dans un environnement WebSphere Application Server Network Deployment, utilisez la syntaxe suivante : `<cell_name>/<node_name>/<serverName_containerSuffix>` **Utilisez les arguments suivants : -mapsizes, -teardown,-revisions,-getTraceSpec,-setTraceSpec,-getStatsSpec,-setStatsSpec**
-fh	n/a	Filtre pour cet hôte uniquement. **Utilisez les arguments suivants : -mapsizes, -teardown,-revisions,-getTraceSpec,-setTraceSpec,-getStatsSpec,-setStatsSpec,-routetable**
-fm	n/a	Filtre uniquement cette mappe. **Utilisez les arguments suivants : -clear, -mapsizes**
-fnp	n/a	Filtre les serveurs qui n'ont pas de fragments principaux. **Utilisez les arguments suivants : -containers**
-fp	n/a	Filtre pour cette partition uniquement. **Utilisez les arguments suivants : -mapsizes, -teardown,-revisions,-getTraceSpec,-setTraceSpec,-getStatsSpec,-setStatsSpec,-routetable**
-fs	n/a	Filtre pour ce serveur uniquement. Si vous effectuez le filtrage des serveurs d'applications dans un environnement WebSphere Application Server Network Deployment , utilisez la syntaxe suivante : `<cell_name>/<node_name>/<server_name>` **Utilisez les arguments suivants : -mapsizes, -teardown,-revisions,-getTraceSpec,-setTraceSpec,-getStatsSpec,-setStatsSpec**
-fst	n/a	Filtre pour ce type de fragment uniquement. Spécifiez P pour les fragments principaux uniquement, A pour les fragments de réplique asynchrone uniquement et S pour les fragments de réplique synchrone uniquement. **Utilisez les arguments suivants : -mapsizes, -teardown,-revisions,-getTraceSpec,-setTraceSpec,-getStatsSpec,-setStatsSpec**
-fz	n/a	Filtre pour cette zone uniquement. **Utilisez les arguments suivants : -mapsizes, -teardown,-revisions,-getTraceSpec,-setTraceSpec,-getStatsSpec,-setStatsSpec,-routetable**
-force	n/a	Force l'action qui est dans la commande, en désactivant toutes les invites préalables. Cet argument est utile pour exécuter des commandes par lot.
-g	n/a	Spécifie le nom d'ObjectGrid.
-getstatsspec	n/a	Affiche la spécification de statistique actuelle. Vous pouvez définir la spécification de statistique avec l'argument **-setstatsspec**. **Permet d'utiliser les filtres suivants :** `-fst -fc -fz -fs -fh -fp`
-getTraceSpec	n/a	Affiche la spécification de trace actuelle. Vous pouvez définir la spécification de trace avec l'argument **-settracespec**. **Permet d'utiliser les filtres suivants :** `-fst -fc -fz -fs -fh -fp`
-h	n/a	Affiche l'aide de l'utilitaire **xsadmin** qui inclut une liste d'arguments.
-hosts	n/a	Affiche tous les hôtes de la configuration.

Tableau 10. Arguments de l'utilitaire **xsadmin** *(suite)*

Arguments de ligne de commande	**Nom de la propriété équivalente dans le fichier de propriétés**	**Description et valeurs valides**
-jmxUrl	XSADMIN_JMX_URL	Spécifie l'adresse d'un serveur de connecteur d'API JMX au format suivant : `service:jmx:`*`protocole`*`:`*`sap`*. Les définitions des variables *protocole* et *sap* sont les suivantes : *protocole* Spécifie le protocole de transport à utiliser pour la connexion au serveur de connecteur. *sap* Spécifie l'adresse à laquelle le serveur de connecteur se trouve. Pour plus d'informations sur le format de l'URL du service JMX, voir Classe JMXServiceURL (Java 2 Platform SE 5.0).
-l	n/a	Affiche toutes les grilles de données et groupes de mappes connues.
-m	n/a	Spécifie le nom du groupe de mappes.
-mapsizes	n/a	Affiche la taille de chaque mappe sur le serveur de catalogue pour vérifier que la distribution des clés est uniforme sur les fragments. **Permet d'utiliser les filtres suivants :** `-fm -fst -fc -fz -fs -fh -fp`
-mbeanservers	n/a	Affiche une liste de tous les noeuds finals de serveur de bean géré.
-overridequorum	n/a	Remplace le paramètre de quorum de sorte que les événements du serveur de conteneur ne sont pas ignorés lors d'un scénario d'échec du centre de données.
-password	XSADMIN_PASSWORD	Spécifie le mot de passe pour se connecter à l'utilitaire **xsadmin**. Ne spécifiez pas le mot de passe dans votre fichier de propriétés si vous voulez que ce mot de passe reste sécurisé.
-p	n/a	Indique le port JMX pour l'hôte du serveur de catalogue. **Valeur par défaut:** 1099 ou 9809 pour un hôte WebSphere Application Server, 1099 pour les configurations autonomes.
-placementStatus	n/a	Affiche le placement configuré et le placement de l'environnement d'exécution de votre configuration. Vous pouvez définir la portée de la sortie à une combinaison de grilles de données et de groupes de mappes, ou à la configuration entière : • Configuration entière : `-placementStatus` • Pour une grille de données spécifique : `-placementStatus -g` *`ma_grille`* • Pour un groupe de mappes spécifique : `-placementStatus -m` *`mon_groupe_de_mappes`* • Pour une grille de données et un groupe de mappes spécifiques : `-placementStatus -g` *`ma_grille`* `-m` *`mon_groupe_de_mappes`*
-primaries	n/a	Affiche une liste des fragments principaux.
-profile	n/a	Spécifie un chemin complet au fichier de propriétés pour l'utilitaire **xsadmin**.
-quorumstatus	n/a	Affiche le statut du quorum pour le service de catalogue.
-releaseShard <*nom_serveur_conteneur*> <*nom_objectgrid*> <*nom_groupe_mappes*> <*nom_partition*>	n/a	Utilisé en association avec l'argument **-reserveShard**. L'argument **-releaseShard** doit être appelé après qu'un fragment a été réservé et placé . L'argument **-releaseShard** appelle la méthode ContainerMBean.release().
-reserved	n/a	Utilisé avec l'argument **-containers** pour afficher uniquement les fragments qui ont été réservés avec l'argument **-reserveShard**.
-reserveShard <*nom_serveur_conteneur*> <*nom_objectgrid*> <*nom_groupe_mappes*> <*nom_partition*>	n/a	Transfère un fragment primaire vers le serveur de conteneur défini. La méthode ContainerMBean.reserve() est appelée par cet argument.
-resumeBalancing <*objectgrid_name*> <*map_set_name*>	n/a	Tente d'équilibrer les demandes et autorise les tentatives de rééquilibrage futurs dans l'ObjectGrid et le groupe de mappes.

Tableau 10. Arguments de l'utilitaire **xsadmin** *(suite)*

Arguments de ligne de commande	Nom de la propriété équivalente dans le fichier de propriétés	Description et valeurs valides
-revisions	n/a	Affiche les identificateurs des révisions d'un domaine de services de catalogue avec chaque grille de données, le numéro de partition, le type de partition (principale ou réplique), le domaine de services de catalogue, l'ID de durée de vie et le nombre de révisions des données de chaque fragment. Vous pouvez utiliser son argument pour déterminer si une réplique asynchrone ou un domaine lié sont interceptés. Cet argument appelle la méthode ObjectGridMBean.getKnownRevisions(). **Permet d'utiliser les filtres suivants :** `-fst -fc -fz -fs -fh -fp`
-routetable	n/a	Affiche l'état actuel de la grille de données à partir d'une perspective serveur client. La table de routage est l'information qu'un serveur client ObjectGrid utilise pour communiquer avec la grille de données. Utilisez la table de routage sous la forme d'une aide au diagnostic lorsque vous tentez d'identifier les problèmes de connexion ou les exceptions `TargetNotAvailable`. **Arguments requis :** Dans un environnement autonome, vous devez spécifier les paramètres **-bp** et **-p** les avec cet argument si vous n'utilisez pas les valeurs par défaut pour le port d'écoute d'amorçage et le port JMX pour l'hôte du serveur de catalogue. **Permet d'utiliser les filtres suivants :** `-fz -fh -fp`
-settracespec *<chaîne_trace>*	n/a	Active la trace sur les serveurs pendant l'exécution. Reportez-vous à l'exemple suivant : `-setTraceSpec "ObjectGridReplication=all=enabled"` Voir Collecte de trace et Options de trace pour plus d'informations sur les chaînes de trace que vous pouvez spécifier. **Permet d'utiliser les filtres suivants :** `-fst -fc -fz -fs -fh -fp`
-swapShardWithPrimary *<container_server_name> <objectgrid_name> <map_set_name> <partition_name>*	n/a	Permute le fragment de réplique défini du serveur de conteneur indiqué et le fragment primaire. Cette commande permet d'équilibrer manuellement les fragments primaires lorsque cela est nécessaire.
-setstatsspec *<spécification_stats>*	n/a	Active la collecte des statistiques. Cet argument appelle les méthodes DynamicServerMBean.setStatsSpec et DynamicServerMBean.getStatsSpec. Voir Class StatsSpec pour plus d'informations sur les modules de statistiques que vous pouvez surveiller. **Permet d'utiliser les filtres suivants :** `-fm -fst -fc -fz -fs -fh -fp`
-suspendBalancing *<objectgrid_name> <map_set_name>*	n/a	Empêche les tentatives d'équilibrage de l'ObjectGrid et du groupe de mappes définis.
-ssl	n/a	Indique que SSL (Secure Sockets Layer) est activé.
-teardown	n/a	Arrête une liste ou un groupe de serveurs de catalogue et de conteneur. **Permet d'utiliser les filtres suivants :** `-fst -fc -fz -fs -fh -fp` **Format pour spécifier une liste de serveurs :** *nom_serveur_1*,*nom_serveur_2* ... **Pour arrêter tous les serveurs dans une zone, incluez l'argument -fz :** –fz *<nom_zone>* **Pour arrêter tous les serveurs sur un hôte, incluez l'argument -fh :** –fh *<nom_hôte>*
-triggerPlacement	n/a	Force le placement de fragment à s'exécuter, en ignorant la valeur **numInitialContainers** définie dans le fichier de déploiement XML. Vous pouvez utiliser cet argument lorsque vous effectuez des opérations de maintenance pour pouvoir continuer à placer les fragments, même si la valeur **numInitialContainers** est inférieure à la valeur définie.
-trustPass	XSADMIN_TRUST_PASS	Spécifie le mot de passe pour le fichier de clés certifiées spécifié.
-trustPath	XSADMIN_TRUST_PATH	Spécifie un chemin vers le fichier de clés certifiées. Exemple : `etc/test/security/server.public`

Tableau 10. Arguments de l'utilitaire **xsadmin** *(suite)*

Arguments de ligne de commande	Nom de la propriété équivalente dans le fichier de propriétés	Description et valeurs valides
-trustType	XSADMIN_TRUST_TYPE	Spécifie le type de fichier de clés certifiées. Les valeurs valides sont : JKS, JCEK, PKCS12, etc.
-unassigned	n/a	Affiche une liste de fragments qui ne peuvent pas être placés sur la grille de données. Les fragments ne peuvent pas être placés lorsque le service de placement a une contrainte qui empêche le placement.
-username	XSADMIN_USERNAME	Spécifie le nom d'utilisateur pour se connecter à l'utilitaire **xsadmin**.
-v	n/a	Active l'action de ligne de commande prolixe. Utilisez cet indicateur si vous utilisez des variables d'environnement, un fichier de propriétés, ou les deux pour spécifier certains arguments de ligne de commande, et si vous voulez afficher leur valeur. Pour plus d'informations, voir «Option prolixe de l'utilitaire **xsadmin**».
-xml	n/a	Affiche la sortie filtrée à partir de la méthode PlacementServiceMBean.listObjectGridPlacement(). Les autres arguments **xsadmin** filtrent la sortie de cette méthode et organisent les données dans un format plus exploitable.

Option prolixe de l'utilitaire xsadmin

Vous pouvez utiliser l'option prolixe **xsadmin** pour traiter les problèmes. Exécutez la commande `xsadmin -v` pour lister tous les paramètres définis. Cette option affiche toutes les valeurs dans toutes les portées, y compris les arguments de ligne de commande, les arguments de fichier de propriétés, et les arguments spécifiés par l'environnement. La section `Arguments effectifs` contient les paramètres qui sont utilisés dans l'environnement, si vous avez spécifié la même propriété en utilisant plusieurs portées.

Exemple d'option prolixe

Arguments de la commande **xsadmin :**

Le texte suivant est un exemple de sortie lorsque vous utilisez l'option prolixe à partir de la ligne de commande après avoir exécuté la commande suivante avec une valeur de propriété spécifiée :

```
./xsadmin -l -v -username xsadmin -password xsadmin -ssl -trustPass ogpass
-profile /opt/ibm/WebSphere/wxs71/ObjectGrid/security/my.properties
```

Arguments de fichier de propriétés :

Le contenu du fichier `/opt/ibm/WebSphere/wxs71/ObjectGrid/security/my.properties` est le suivant :

```
XSADMIN_TRUST_PASS=ogpass
XSADMIN_TRUST_TYPE=jks
XSADMIN_TRUST_PATH=/opt/ibm/WebSphere/wxs71/ObjectGrid/bin/security/key.jks
XSADMIN_USERNAME=ogadmin
XSADMIN_PASSWORD=ogpass
```

Résultats de la commande :

Dans la sortie suivante de la commande **xsadmin** précédente, le texte qui se trouve en ***italique gras*** indique les propriétés et les valeurs qui sont spécifiées à la fois sur la ligne de commande et dans le fichier de propriétés. Dans la section des `arguments de ligne de commande`, notez que les arguments définis dans la ligne de commande remplacent les valeurs dans le fichier des propriétés.

```
Arguments spécifiés de ligne de commande
**********************************
XSADMIN_USERNAME=xsadmin
XSADMIN_PASSWORD=xsadmin
XSADMIN_TRUST_PATH=<unspecified>
XSADMIN_TRUST_TYPE=<unspecified>
XSADMIN_TRUST_PASS=ogpass
XSADMIN_PROFILE=/opt/ibm/WebSphere/wxs71/ObjectGrid/security/my.properties
XSADMIN_JMX_URL=<unspecified>
**********************************
Properties file specified arguments
**********************************
XSADMIN_USERNAME=ogadmin
XSADMIN_PASSWORD=ogpass
XSADMIN_TRUST_PATH=/opt/ibm/WebSphere/wxs71/ObjectGrid/bin/security/key.jks
XSADMIN_TRUST_TYPE=jks
XSADMIN_TRUST_PASS=ogproppass
XSADMIN_JMX_URL=<unspecified>
**********************************
Environment-specified arguments
**********************************
XSADMIN_USERNAME=<unspecified>
XSADMIN_PASSWORD=<unspecified>
XSADMIN_TRUST_PATH=<unspecified>
XSADMIN_TRUST_TYPE=<unspecified>
XSADMIN_TRUST_PASS=<unspecified>
XSADMIN_JMX_URL=<unspecified>
**********************************
Effective arguments
**********************************
XSADMIN_USERNAME=xsadmin
XSADMIN_PASSWORD=xsadmin
XSADMIN_TRUST_PATH=/opt/ibm/WebSphere/wxs71/ObjectGrid/bin/security/key.jks
XSADMIN_TRUST_TYPE=jks
XSADMIN_TRUST_PASS=ogpass
XSADMIN_PROFILE=/opt/ibm/WebSphere/wxs71/ObjectGrid/security/my.properties
XSADMIN_JMX_URL=<unspecified>
SSL authentication enabled: true
**********************************
Connecting to Catalog service at localhost:1099
*** Show all 'objectGrid:mapset' names
Grid Name  MapSet Name
accounting defaultMapSet
```

Avertissement : La propriété XSADMIN_PROFILE, bien qu'elle s'affiche dans la sortie prolixe, ne constitue pas une clé valide que vous pouvez spécifier dans un fichier de propriétés. La valeur de cette propriété dans la sortie prolixe indique la valeur de propriété qui est en cours d'utilisation, comme indiqué dans l'argument de ligne de commande **-profile**.

Sortie sans l'option prolixe

Exemple de sortie de la même commande sans l'option prolixe :

```
./xsadmin -l -username xsadmin -password xsadmin -ssl -trustPass ogpass
-profile /opt/ibm/WebSphere/wxs71/ObjectGrid/security/my.properties
```

```
Connecting to Catalog service at localhost:1099
*** Show all 'objectGrid:mapset' names
Grid Name  MapSet Name
accounting defaultMapSet
```

Remarques

Les références aux produits, logiciels et services d'IBM n'impliquent pas qu'ils soient distribués dans tous les pays dans lesquels IBM exerce son activité. Toute référence à un produit, logiciel ou service IBM n'implique pas que seul ce produit, logiciel ou service puisse être utilisé. Tout autre élément fonctionnellement équivalent peut être utilisé, s'il n'enfreint aucun droit d'IBM. L'évaluation et la vérification de son fonctionnement en conjonction avec d'autres produits, hormis ceux expressément désignés par IBM, relèvent de la responsabilité de l'utilisateur.

IBM peut détenir des brevets ou des demandes de brevet couvrant les produits mentionnés dans le présent document. La remise de ce document ne vous donne aucun droit de licence sur ces brevets ou demandes de brevet. Si vous désirez recevoir des informations concernant l'acquisition de licences, veuillez en faire la demande par écrit à l'adresse suivante :

IBM Director of Licensing
IBM Corporation
500 Columbus Avenue
Thornwood, New York 10594 USA

Les licenciés souhaitant obtenir des informations permettant : (i) l'échange des données entre des logiciels créés de façon indépendante et d'autres logiciels (dont celui-ci), et (ii) l'utilisation mutuelle des données ainsi échangées, doivent adresser leur demande à :

IBM Corporation
Mail Station P300
522 South Road
Poughkeepsie, NY 12601-5400
USA
Attention: Information Requests

Ces informations peuvent être soumises à des conditions particulières, prévoyant notamment le paiement d'une redevance.

Marques

Les termes suivant sont des marques d'IBM Corporation aux Etats-Unis et/ou dans certains autres pays :

- AIX
- CICS
- Cloudscape
- DB2
- Domino
- IBM
- Lotus
- RACF
- Redbooks
- Tivoli
- WebSphere
- z/OS

Java ainsi que tous les logos et toutes les marques incluant Java sont des marques de Sun Microsystems, Inc. aux Etats-Unis et/ou dans certains autres pays.

LINUX est une marque de Linus Torvalds aux Etats-Unis et/ou dans certains autres pays.

Microsoft, Windows, Windows NT et le logo Windows sont des marques de Microsoft Corporation aux Etats-Unis et/ou dans certains autres pays.

UNIX est une marque enregistrée de The Open Group aux Etats-Unis et dans certains autres pays.

Les autres noms de sociétés, de produits et de services peuvent appartenir à des tiers.

Index

Q

R

S

T

U

V

Z