IBM WebSphere Commerce V7 Quarterly Releases – 2Q2015

Index Load

Index Load Complexities

- The existing WebSphere Commerce Search indexing infrastructure makes a number of assumptions, specifically:
  - Data for an index is retrieved from a single data source
    - Typically, the WC instance database
    - This can be amended by adding data from external sources in preprocessing, though
  - Data for a given Solr core is added sequentially during reindexing
    - Feature Pack 7 did add support for sharding, within the existing indexing framework
- Some clients have more complex setups, for example:
  - Vast product catalogues, requiring parallel indexing of data to lower reindexing time
  - Data stored in external sources, e.g. complex B2B prices in back-end systems
    - Although these could be loaded into the WC database, this is not always optimal
  - Different data updating at different intervals, requiring reindexing of different (parts) of cores at different intervals
- Monitoring of the overall status of index loading is not currently easy for clients
- To address these concerns we now introduce IndexLoad, a new utility in the WebSphere Commerce Search component

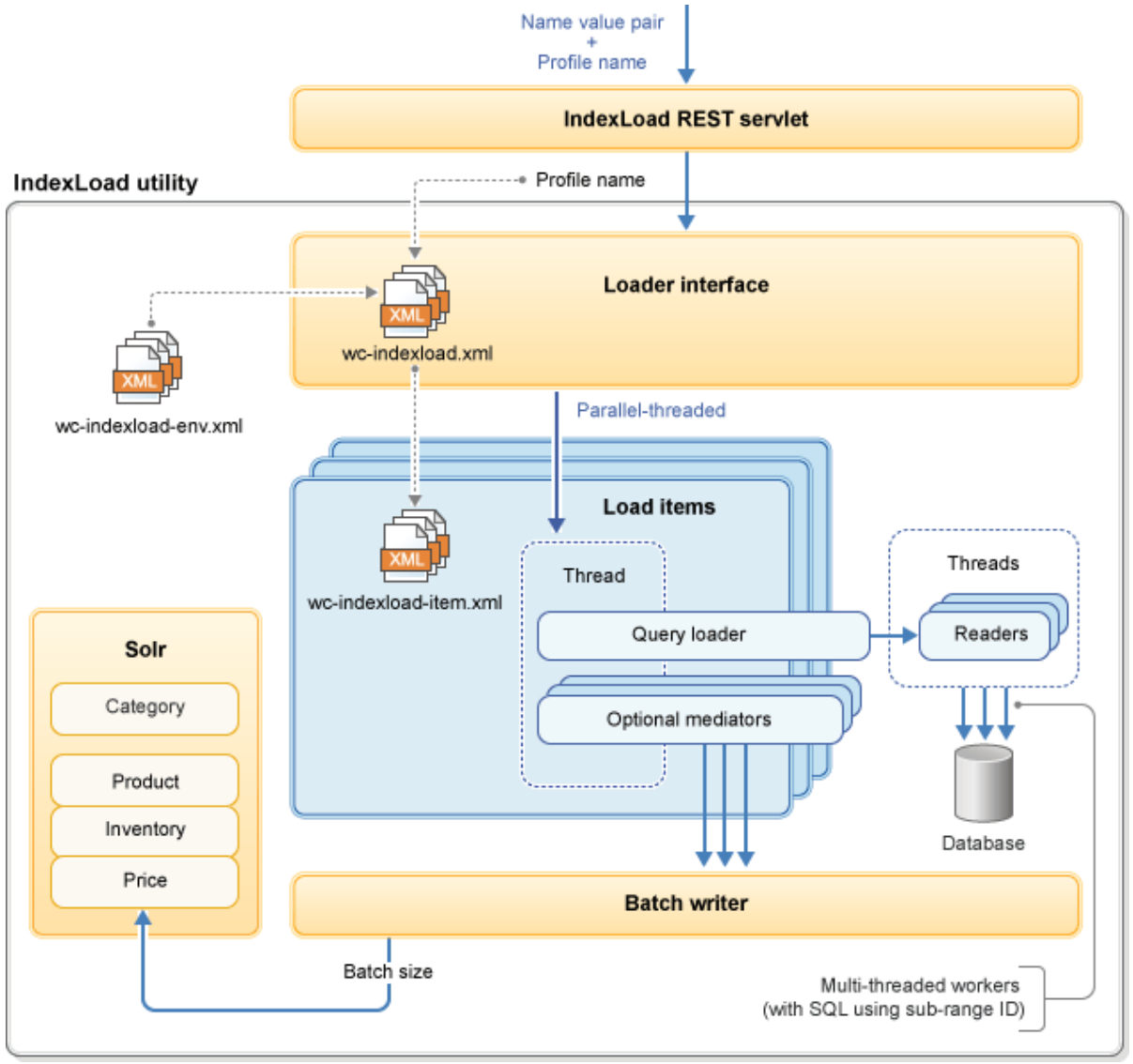
Introducing the IndexLoad utility

- IndexLoad is a new Web-based utility that supports:
  - Improved indexing performance by avoiding remote HTTP calls to store data in index
  - Indexing from parallel data feeds
  - Separate, different data sources, allowing load of data directly from a client's back-end system
  - Partitioning of source data, allowing for easy sharding at index time
  - Monitoring of indexing status
- It runs on the Search server and is invoked by hitting the following URL:
  - http://<*searchserver*>/search/indexload/<*command*>?<*parameters*>
  - Where <*command*> is one of:
    - config: display configuration
    - clear: delete all contents in the configured cores
    - merge: merges all unloaded index data directories
    - optimize: optimizes contents for all configured cores
    - start: starts indexing
    - stop: stops indexing
    - status: retrieves the status of IndexLoad

http://www.ibm.com/support/knowledgecenter/SSZLC2_7.0.0/com.ibm.commerce.developer.doc/tasks/tsdsearchindexload.htm

IndexLoad architecture

- Controlled through a REST servlet
- Loader interface controls a number of Load Items
- Each Load Item uses one or more Readers to read data from the physical source
  - Database
  - CSV file
- After loading the data, zero or more chained Mediators can modify the data
- Finally, the data is sent to the Batch Writer, which handles queuing and sending documents to Solr



http://www.ibm.com/support/knowledgecenter/SSZLC2_7.0.0/com.ibm.commerce.developer.doc/concepts/csdsearchindexload.htm

Sample Scenario: Loading prices from external source (1/2)

- Requirements:
  - Prices are kept in external back-end system
  - Prices provided from back-end system in a set of CSV files on a regular interval
- IndexLoad setup:
  - Create a new IndexLoad profile with a Load Item for each CSV file that should be loaded
  - Each LoadItem refers to a business object configuration that loads from a CSV in standard Data Load syntax

```
<_config:LoadItem name="ExternalPrice-1" businessObjectConfigFile="wc-indexload-price-csv.xml">
        <_config:property name="coreName" value="MC_10001_CatalogEntry_Price" />
        <_config:DataSourceLocation location="pricelist-1.csv" />
</_config:LoadItem>

<_config:LoadItem name="ExternalPrice-2" businessObjectConfigFile="wc-indexload-price-csv.xml">
        <_config:property name="coreName" value="MC_10001_CatalogEntry_Price" />
        <_config:DataSourceLocation location="pricelist-2.csv" />
</_config:LoadItem>

<_config:LoadItem name="ExternalPrice-3" businessObjectConfigFile="wc-indexload-price-csv.xml">
        <_config:property name="coreName" value="MC_10001_CatalogEntry_Price" />
        <_config:DataSourceLocation location="pricelist-3.csv" />
</_config:LoadItem>
```

Sample Scenario: Loading prices from external source (2/2)

- The CSV business object configuration:

```
<_config:DataLoader className="com.ibm.commerce.foundation.server.services.indexload.loader.solr.SolrIndexLoadCSVLoader" >

   <_config:DataReader className="com.ibm.commerce.foundation.server.services.indexload.reader.solr.SolrIndexLoadCSVReader"
         firstLineIsHeader="true" useHeaderAsColumnName="true" />

   <_config:BusinessObjectBuilder
         className="com.ibm.commerce.foundation.internal.server.services.indexload.builder.SolrIndexLoadMapObjectBuilder" >

      <_config:DataMapping>
         <_config:mapping xpath="catentry_id" value="catentry_id" />
         <_config:mapping xpath="price_segments" value="price_segments" />
      </_config:DataMapping>

      <_config:BusinessObjectMediator
            className="com.ibm.commerce.foundation.internal.server.services.indexload.mediator.SolrIndexLoadBusinessObjectMediator">
         <_config:extension
               className="com.ibm.commerce.foundation.server.services.indexload.mediator.solr.SolrIndexLoadExternalPriceMediator" />
         </_config:BusinessObjectMediator>

   </_config:BusinessObjectBuilder>

</_config:DataLoader>
```

Sample Scenario: Loading prices from database (1/2)

- Requirements:
  - Contract prices are loaded in the WC database, denormalized into TI_CNTRPRICE_1
- IndexLoad setup:
  - Create a new IndexLoad profile with a Load Item for loading from database

```
<_config:LoadItem name="ExternalPrice-2" businessObjectConfigFile="wc-indexload-price-sql.xml">
        <_config:property name="coreName" value="MC_10001_CatalogEntry_Price" />
</_config:LoadItem>
```

Sample Scenario: Loading prices from database (2/2)

- The SQL business object configuration is shown below
  - Prices are assumed to be preprocessed in the temporary table TI_CNTRPRICE_1

```
<_config:property name="ParallelLowerRangeSQL" value="SELECT MIN(CE.CATENTRY_ID) FROM CATENTRY CE" />
<_config:property name="ParallelUpperRangeSQL" value="SELECT MAX(CE.CATENTRY_ID) FROM CATENTRY CE" />
<_config:property name="ParallelNextRangeSQL" value="SELECT MIN(CE.CATENTRY_ID) FROM CATENTRY CE WHERE
CE.CATENTRY_ID &gt; ?" />
 <_config:DataReader
      className="com.ibm.commerce.foundation.server.services.indexload.reader.solr.SolrIndexLoadQueryMultiplexReader">
   <_config:Query>
     <_config:SQL>
       SELECT TI.CATENTRY_ID,TI.PRICE
       FROM TI_CNTRPRICE_1 TI
       WHERE TI.CATENTRY_ID &gt;= %ParallelLowerRange%
       AND TI.CATENTRY_ID &lt;= %ParallelUpperRange%
       ORDER BY TI.CATENTRY_ID
     </_config:SQL>
   <_config:ColumnMapping name="CATENTRY_ID" value="catentry_id" />
   <_config:ColumnMapping name="PRICE" value="price" />
</_config:Query>

<_config:property name="ParallelThreads" value="2" />
<_config:property name="ParallelLowerRange" value="" />
<_config:property name="ParallelUpperRange" value="" />
<_config:property name="ParallelPrefetchSize" value="100" />
```

Tuning IndexLoad

- IndexLoad provides detailed information about its performance when indexing
- Status reports provides real time indexing statistics for tuning IndexLoad settings:
  - Read and write rate
  - Indexed document average size and width
  - Cumulative read, flush, commit time
  - Memory utilization and CPU workload distribution
  - Index size and other important core specific settings
- Use this data to tune the prefetch and commit count values

Sample Tuning IndexLoad

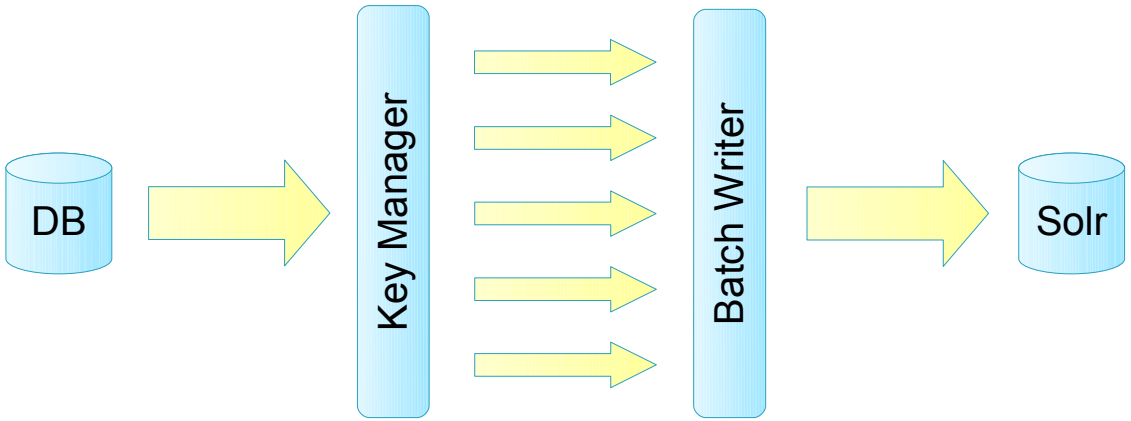
```
The full indexing task has been running for 15.626 seconds at 6/19/15 1:59 PM.

653 records have been read at an average rate of 41.789 records per second.
325 documents have been indexed at an average rate of 20.799 documents per second and 21.552 documents per second since last flushing.
Read-to-Write ratio is approximately 2.009.

Current batch queue size is 5 with a maximum size of 29.
Indexed document size average is 160,048 bytes.
Indexed document width average is 5,002 columns.

Total cumulative flush time is on average 4.532 seconds based on a batch size of 10 per thread.
No cumulative commit time is available because commit count has been disabled.
Total cumulative read time is on average 0.612 seconds based on an average prefetch size of 10 per thread.
Total cumulative mediation time is on average 12.853 seconds per thread.

Current average system load is approximately 22.335.
Total available processors are 8 with 1.762 GB of maximum amount of memory.
Total amount of used memory is 1.15 GB (65%), free memory is 0.193 GB (11%), unallocated memory is 0.419 GB (24%).
Total number of configured threads is 4 and total number of threads currently running is 4.
```

			I/O throughput per second		Cumulative processing time (seconds)	
Load item thread	Status	CPU usage	Read rate	Write rate	Reader	Mediator
ExternalPrice-1 (000)	Processing	23.52%	12.21 (67%)	6.10 (33%)	1.11 (8%)	12.00 (92%)
ExternalPrice-1 (001)	Processing	27.54%	11.45 (67%)	5.69 (33%)	0.65 (4%)	13.85 (96%)
ExternalPrice-1 (002)	Processing	24.92%	13.57 (67%)	6.75 (33%)	0.37 (3%)	12.75 (98%)
ExternalPrice-1 (003)	Processing	24.02%	11.58 (67%)	5.75 (33%)	0.32 (2%)	12.81 (98%)

```
--------------------------------------------------------------------------------
Index name: MC_10051_CatalogEntry_Price_generic
--------------------------------------------------------------------------------
Total number of active documents in index: 314
Total number of deleted documents in index: 0
Maximum internal identifier in index: 314
Total number of index segments: 14
Total number of static index fields: 1
Total number of dynamic index fields: 5001
Index data directory size: 25.0 MB
Index version number: 34190
Index RAM buffer size: 100.0 MB
Index maximum buffered document: 10
Index merge factor: 10
--------------------------------------------------------------------------------
```

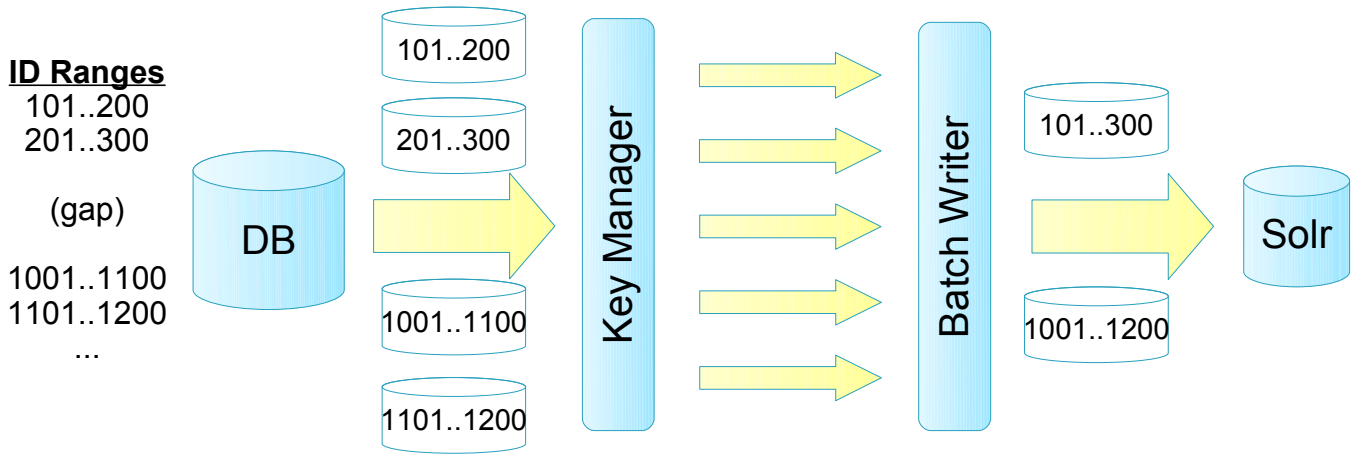
IndexLoad Performance Considerations

- The overall logic flow for IndexLoad follows a diamond shape:
  - Starts with one single source of input
  - Main body comprise multiple worker threads
  - Ends with a single output
- Important tuning parameters:
  - Prefetch size controls the input rate to avoid overloading the database
  - Number of threads control the level of parallelism
  - Batch/commit count



Parallelism and Prefetch Size

- The prefetch size controls how many rows are read from the data source at a time
  - This must be tuned to balance database load against data availability for worker threads
- The thread count controls how many parallel threads are processing these rows
  - This must be tuned to balance overall CPU load with ability to process data in parallel
- The ParallelNextRangeSQL is used to avoid gaps in input ID ranges
  - Each range is fetched and distributed across the worker threads
- Batch count controls the sizes of the batches sent to Solr

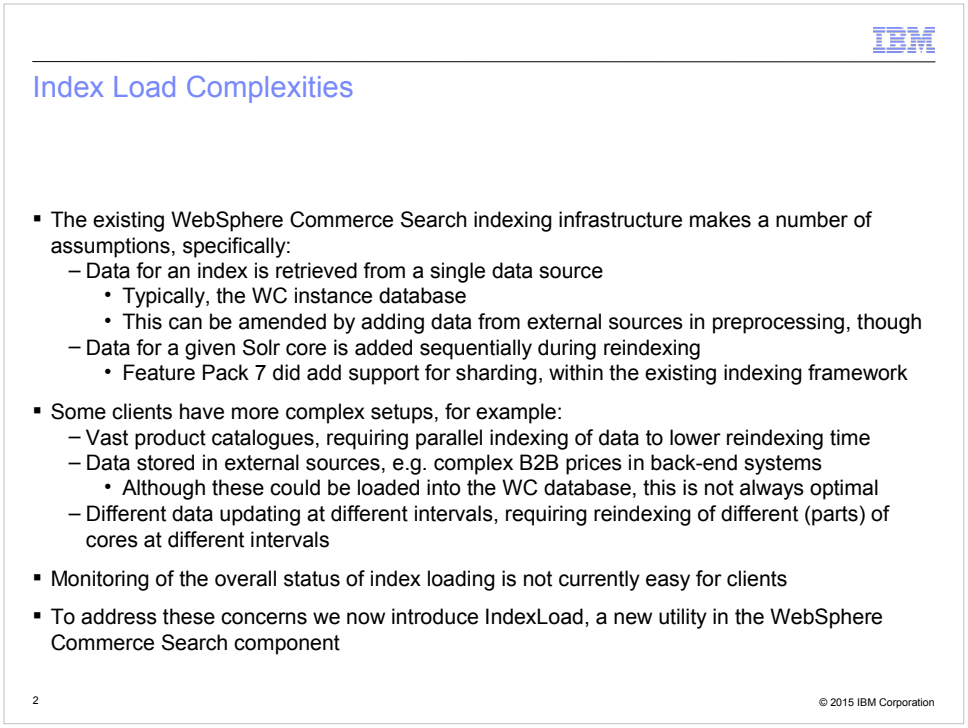


Trademarks, disclaimer, and copyright information

IBM WebSphere Commerce V7 Quarterly Releases – 2Q2015

Index Load

Welcome to the Index Load module of the technical enablement for the second quarter release 2015 of IBM WebSphere Commerce version 7.



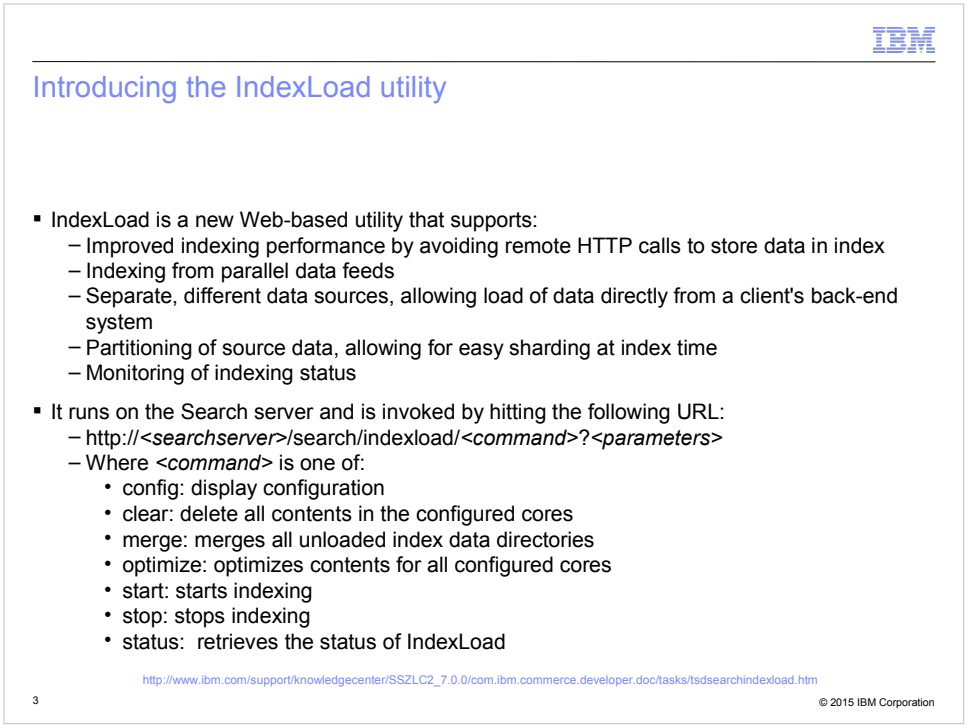
Some advanced requirements has historically been hard to implement using the existing WebSphere Commerce Search indexing infrastructure.

This is due to some assumptions made in the design of the existing solution, specifically that data mainly resides in the WebSphere Commerce database, and that data must be processed sequentially.

In some scenarios, such as very large catalog sizes, or customers where some data, for example prices, only reside in 3rd-party systems, these assumptions have caused some head-ache for implementation teams.

Although there are existing solutions that help mitigate these assumptions, see for example the index sharding topics in the Knowledge Center, we are now introducing a new indexing tool to help address these concerns at a more fundamental level.

The new tool, Index Load, is based on a more flexible architecture, allowing for better parallelization and open to external index sources.



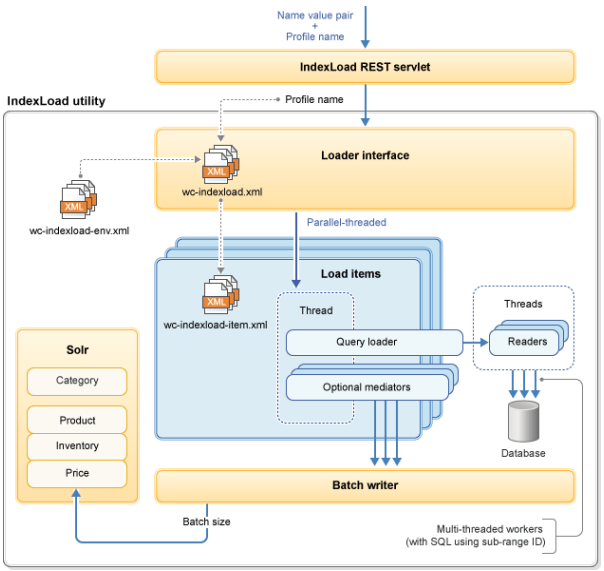
The new Index Load solution is a Web-based utility, adding functionality for parallel data feeds, separating the data source definition from the processing, partitioning of the source data to allow for sharding, as well as improved indexing status monitoring.

We have also improved the performance in the communication with Solr. Where we previously used HTTP connections, even when indexing locally on the same server as Solr, we now use the embedded client to bypass the HTTP layer.

The main interface for Index Load is a REST-like servlet that accepts a number of URL-based commands, as outlined on this slide, which makes it easier to build a graphical user interface, if so desired, on top of this interface.

IndexLoad architecture

- Controlled through a REST servlet
- Loader interface controls a number of Load Items
- Each Load Item uses one or more Readers to read data from the physical source
  - Database
  - CSV file
- After loading the data, zero or more chained Mediators can modify the data
- Finally, the data is sent to the Batch Writer, which handles queuing and sending documents to Solr



http://www.ibm.com/support/knowledgecenter/SSZLC2_7.0.0/com.ibm.commerce.developer.doc/concepts/csdsearchindexload.htm

This slide shows the architecture of the Index Load utility.

As mentioned previously, a REST interface is provided, through which all commands, such as start indexing, retrieve index status, etc., are passed.

The REST interface passes all requests to the Loader Interface. This interface retrieves the configuration for the index profile that the request relates to, and passes the incoming command to the relevant Load Item or Load Items.

Each of the Load Items uses one or more Readers to retrieve the data from the data source. A Reader can at this point read data from either a JDBC connection, or a comma-delimited CSV file.

After loading the data, the data is passed through a chain of zero or more Mediators, to allow for massaging of the data before it is sent on to Solr. Zero, because the data can be sent directly to Solr without any modifications, but the configuration can employ any number of mediators in sequence, as needed.

Instead of sending the processed data directly to Solr, the Index Load utility uses a singleton Batch Writer. This component is responsible for queuing up the data and control the flushing and committing of the data to Solr.

Sample Scenario: Loading prices from external source (1/2)

- Requirements:
  - Prices are kept in external back-end system
  - Prices provided from back-end system in a set of CSV files on a regular interval
- IndexLoad setup:
  - Create a new IndexLoad profile with a Load Item for each CSV file that should be loaded
  - Each LoadItem refers to a business object configuration that loads from a CSV in standard Data Load syntax

```
<_config:LoadItem name="ExternalPrice-1" businessObjectConfigFile="wc-indexload-price-csv.xml">
        <_config:property name="coreName" value="MC_10001_CatalogEntry_Price" />
        <_config:DataSourceLocation location="pricelist-1.csv" />
</_config:LoadItem>

<_config:LoadItem name="ExternalPrice-2" businessObjectConfigFile="wc-indexload-price-csv.xml">
        <_config:property name="coreName" value="MC_10001_CatalogEntry_Price" />
        <_config:DataSourceLocation location="pricelist-2.csv" />
</_config:LoadItem>

<_config:LoadItem name="ExternalPrice-3" businessObjectConfigFile="wc-indexload-price-csv.xml">
        <_config:property name="coreName" value="MC_10001_CatalogEntry_Price" />
        <_config:DataSourceLocation location="pricelist-3.csv" />
</_config:LoadItem>
```

5

In this part of the presentation, we will cover two sample scenarios of employing Index Load to optimize the loading of data into the WebSphere Commerce Search index.

First, we explore the configuration for a scenario where a client keep price data in a 3rd-party back-end system, instead of loading the prices into the WebSphere Commerce database.

In this scenario, the back-end system provides price updates at a regular interval through three comma-delimited CSV files, each file containing a third of the total prices to update.

To support this, you would create an Index Load profile with three Load Items. Each Load Item loads the prices from one of the three CSV files. The relevant section of the profile configuration file with the Load Items is shown here.

Note that all three Load Items refer to the same configuration file, wc-indexload-price-csv.xml, but with different CSV file names specified.

Since we are thus indexing prices separately from the products, we store the prices in a new Solr core, here referred to as MC_10001_CatalogEntry_Price.

Not shown here are the modifications needed to remove prices from the regular product index, as well as the other storefront and business logic changes to look up prices from

Sample Scenario: Loading prices from external source (2/2)

- The CSV business object configuration:

```
<_config:DataLoader className="com.ibm.commerce.foundation.server.services.indexload.loader.solr.SolrIndexLoadCSVLoader" >

   <_config:DataReader className="com.ibm.commerce.foundation.server.services.indexload.reader.solr.SolrIndexLoadCSVReader"
         firstLineIsHeader="true" useHeaderAsColumnName="true" />

   <_config:BusinessObjectBuilder
         className="com.ibm.commerce.foundation.internal.server.services.indexload.builder.SolrIndexLoadMapObjectBuilder" >

      <_config:DataMapping>
         <_config:mapping xpath="catentry_id" value="catentry_id" />
         <_config:mapping xpath="price_segments" value="price_segments" />
      </_config:DataMapping>

      <_config:BusinessObjectMediator
            className="com.ibm.commerce.foundation.internal.server.services.indexload.mediator.SolrIndexLoadBusinessObjectMediator">
         <_config:extension
               className="com.ibm.commerce.foundation.server.services.indexload.mediator.solr.SolrIndexLoadExternalPriceMediator" />
         </_config:BusinessObjectMediator>

   </_config:BusinessObjectBuilder>

</_config:DataLoader>
```


This slide shows a snippet from the wc-indexload-price-csv.xml file references on the previous slide.

Note that the configuration specifies the type of Reader used, SolrIndexLoadCSVReader, the mapping from CSV columns to Solr field names, and finally that we are using a standard price mediator, SolrIndexLoadExternalPriceMediator, to manipulate the data before it is stored directly in Solr.

Sample Scenario: Loading prices from database (1/2)

- Requirements:
  - Contract prices are loaded in the WC database, denormalized into TI_CNTRPRICE_1
- IndexLoad setup:
  - Create a new IndexLoad profile with a Load Item for loading from database

```
<_config:LoadItem name="ExternalPrice-2" businessObjectConfigFile="wc-indexload-price-sql.xml">
        <_config:property name="coreName" value="MC_10001_CatalogEntry_Price" />
</_config:LoadItem>
```


This slide introduces a second scenario for using Index Load.

In this scenario, we have a client where the contract prices are loaded into the database. We assume the prices have already been denormalized into a temporary table.

In this case, you would define a similar configuration as before, except that you would a Load Item configurations where prices are loaded from the database.

Sample Scenario: Loading prices from database (2/2)

- The SQL business object configuration is shown below
  - Prices are assumed to be preprocessed in the temporary table TI_CNTRPRICE_1

```
<_config:property name="ParallelLowerRangeSQL" value="SELECT MIN(CE.CATENTRY_ID) FROM CATENTRY CE" />
<_config:property name="ParallelUpperRangeSQL" value="SELECT MAX(CE.CATENTRY_ID) FROM CATENTRY CE" />
<_config:property name="ParallelNextRangeSQL" value="SELECT MIN(CE.CATENTRY_ID) FROM CATENTRY CE WHERE
CE.CATENTRY_ID &gt; ?" />
 <_config:DataReader
      className="com.ibm.commerce.foundation.server.services.indexload.reader.solr.SolrIndexLoadQueryMultiplexReader">
   <_config:Query>
     <_config:SQL>
       SELECT TI.CATENTRY_ID,TI.PRICE
       FROM TI_CNTRPRICE_1 TI
       WHERE TI.CATENTRY_ID &gt;= %ParallelLowerRange%
       AND TI.CATENTRY_ID &lt;= %ParallelUpperRange%
       ORDER BY TI.CATENTRY_ID
     </_config:SQL>
   <_config:ColumnMapping name="CATENTRY_ID" value="catentry_id" />
   <_config:ColumnMapping name="PRICE" value="price" />
</_config:Query>

<_config:property name="ParallelThreads" value="2" />
<_config:property name="ParallelLowerRange" value="" />
<_config:property name="ParallelUpperRange" value="" />
<_config:property name="ParallelPrefetchSize" value="100" />
```


In this example, we assume that the prices have been preprocessed and are available in the table TI_CNTRPRICE_1.

Notice that we see four SQL statements defined here:

- A SQL, called ParallelLowerRangeSQL, which retrieves the smallest product ID in the price table

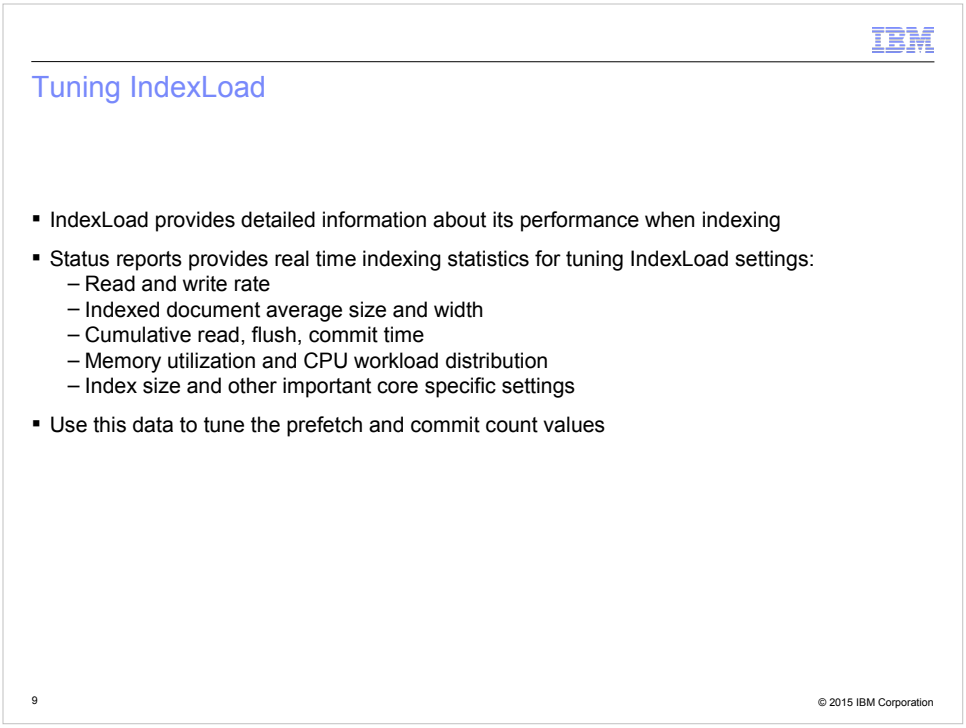
- One called ParallelUpperRangeSQL, which retrieves the largest product ID in the price table

- ParallelNextRangeSQL, which retrieves the smallest ID, above a certain value

- And finally the main SQL, which retrieves all the prices for a given product ID range

These SQL statements allows for the Index Load utility to calculate the product ID ranges and perform a sharded indexing, if requested.

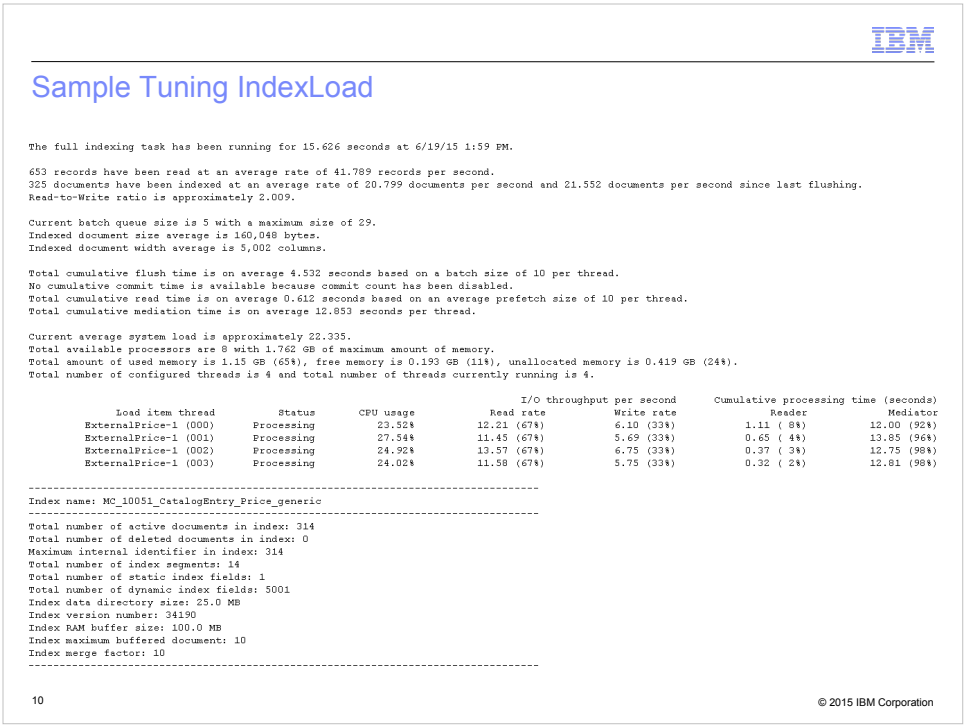
Finally, this file also provides the mapping from SQL result column names to Solr field names.



Finally, we would like to make some comments on performance tuning of Index Load.

As mentioned, Index Load provides access to status information through the Web interface. The status report contains detailed information about the current state of indexing. This information is crucial in tuning a large Index Load.

The result of any change in the configuration can be monitoring on the status report page by inspecting some of the values mentioned here, for example, the read, flush, and commit times, and the memory utilitization and CPU workload distribution.



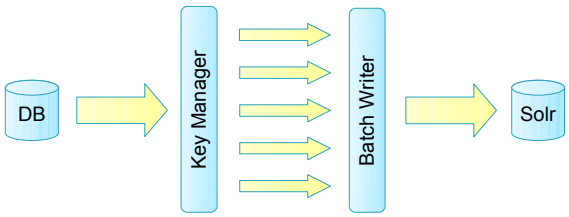
This slide shows an example of a status report during an Index Load.

Amongst other useful statistics, we can see four worker threads, their individual CPU usage, read- and write rates, as well as where the processing time is spent between the readers and mediators.

We can also see the memory utilization, overall read and write rates, as well as statistics about document counts and sizes.

IndexLoad Performance Considerations

- The overall logic flow for IndexLoad follows a diamond shape:
  - Starts with one single source of input
  - Main body comprise multiple worker threads
  - Ends with a single output
- Important tuning parameters:
  - Prefetch size controls the input rate to avoid overloading the database
  - Number of threads control the level of parallelism
  - Batch/commit count



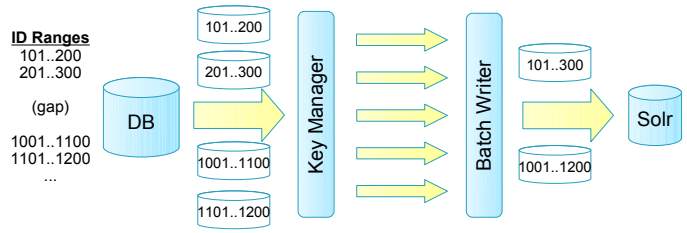
Since the processing flow in Index Load is comparable to a diamond shape, with a single reader in one end, a single writer in the other and a number of parallel worker threads in the middle, three important settings have an impact on overall performance:

- Prefetch size, controlling input buffering
- Number of worker threads, controlling the amount of parallelism; and
- Commit count, controlling the size of the batches sent to Solr for processing

The status report mentioned earlier, can be a useful aid in tuning these values.

Parallelism and Prefetch Size

- The prefetch size controls how many rows are read from the data source at a time
  - This must be tuned to balance database load against data availability for worker threads
- The thread count controls how many parallel threads are processing these rows
  - This must be tuned to balance overall CPU load with ability to process data in parallel
- The ParallelNextRangeSQL is used to avoid gaps in input ID ranges
  - Each range is fetched and distributed across the worker threads
- Batch count controls the sizes of the batches sent to Solr



Now we would like to cover the effect of the three core tuning parameters.

In this example, we are reindexing from a database source. The data we are loading has a gap in the primary IDs. Between IDs 300 and 1001, there is no data for reindexing.

Furthermore, we have set the parallel prefetch size to 100, which means that the key manager will load 100 entries at a time from the database.

We have defined five active threads by setting the ParallelThreads parameter to 5.

Finally, the batch count has been set to 200.

As a result, the ket manager will load the source data in batches of 100 at a time. The SQL defined in the business object configuration will allow for detection of gaps in the data, so we can see the four batches loaded to be IDs from 101 to 200, from 201 to 300, and then skipping to 1001, etc.

Each batch is distributed across the five worker threads and when processed, the key manager will load the next batch.

On the writer end, since the batch count is 200, we are queueing up the results in batches of 200 records before sending these to Solr for inclusion in the index.

Trademarks, disclaimer, and copyright information