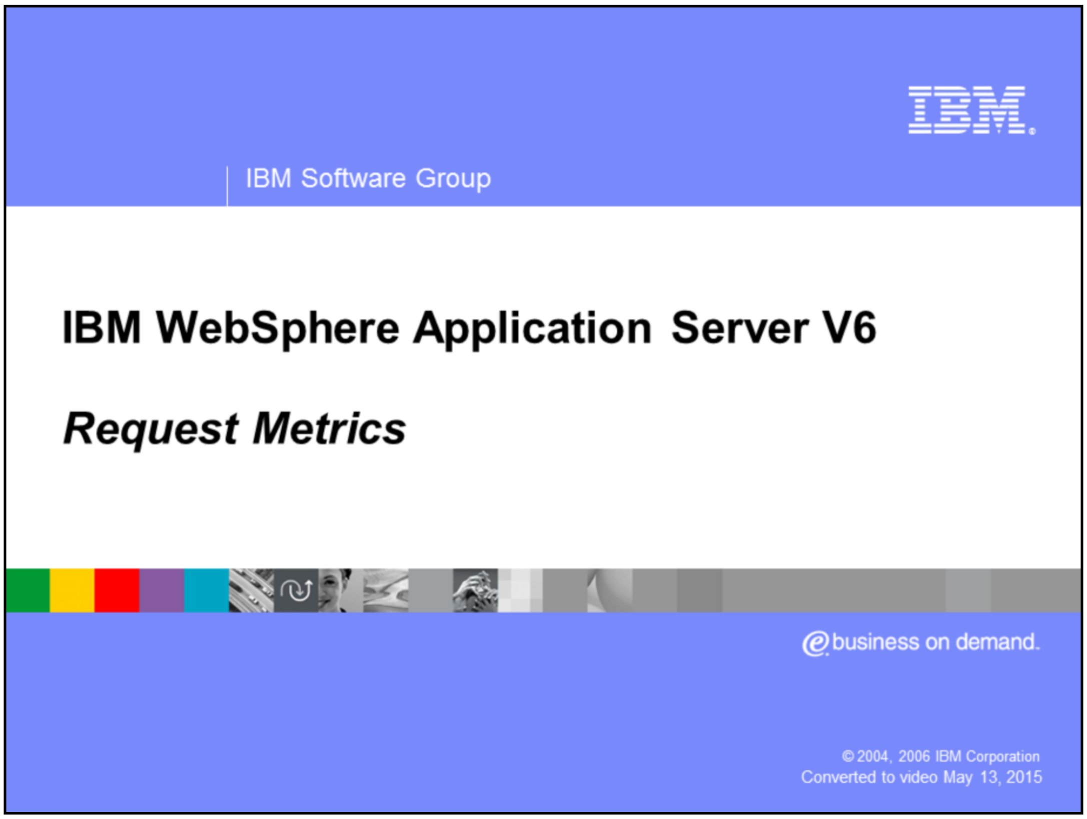


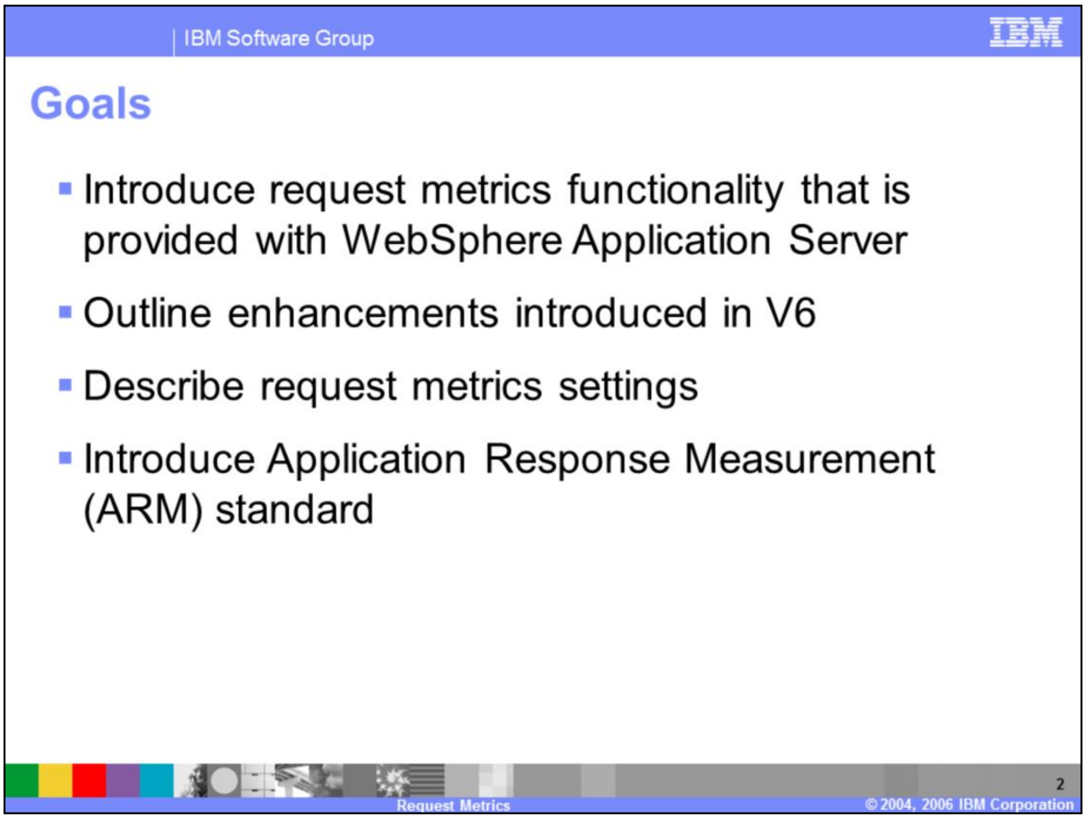
IBM Software Group

IBM WebSphere Application Server V6

Request Metrics

business on demand.

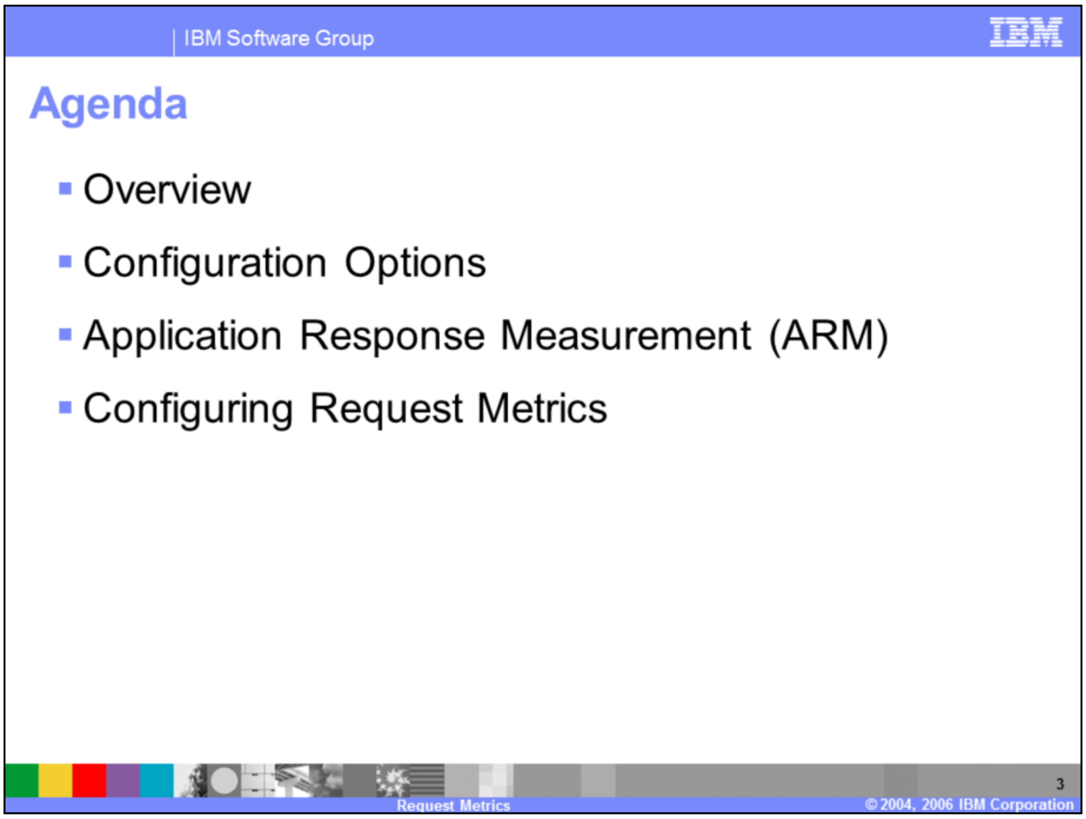
This presentation will focus on WebSphere Application Server's request metrics.



Goals

- Introduce request metrics functionality that is provided with WebSphere Application Server
- Outline enhancements introduced in V6
- Describe request metrics settings
- Introduce Application Response Measurement (ARM) standard

The goals for this presentation are to introduce the request metrics functionality that is provided with the WebSphere Application Server. The presentation will also introduce the Application Response Measurement or ARM standard and explain the part that it plays with request metrics. The goal is also to point out any differences between version 5 and version 6 along the way.

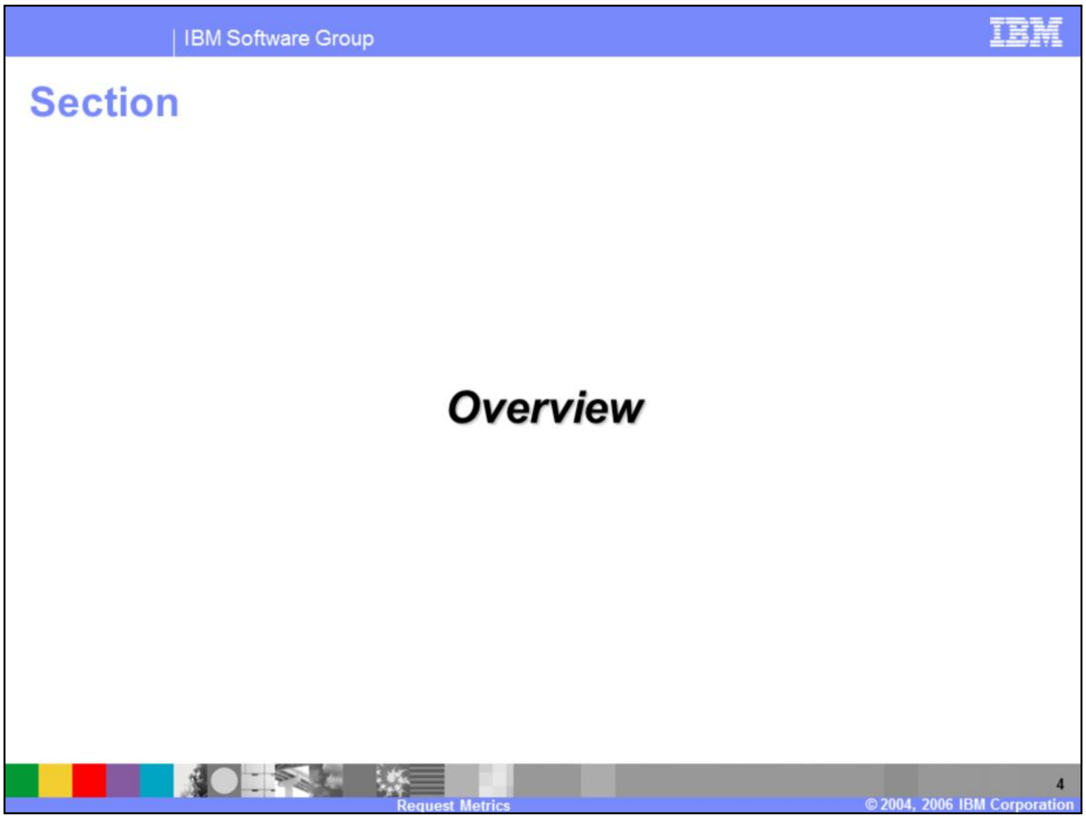


IBM

Agenda

- Overview
- Configuration Options
- Application Response Measurement (ARM)
- Configuring Request Metrics

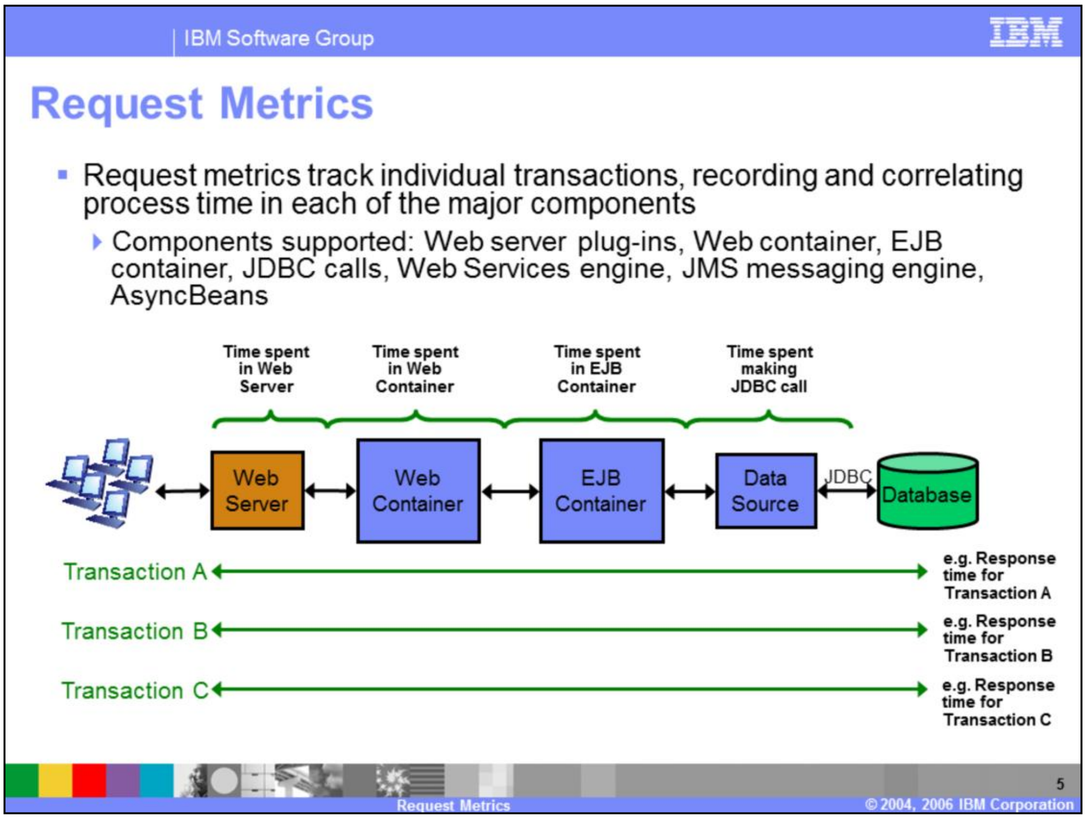
This presentation discusses request metrics, its configuration options and the Application Response Measurement standard.



Section

Overview

This section will discuss what the request metrics functionality provides to WebSphere Application Server.



IBM Software Group

Request Metrics

- Request metrics track individual transactions, recording and correlating process time in each of the major components
  - Components supported: Web server plug-ins, Web container, EJB container, JDBC calls, Web Services engine, JMS messaging engine, AsyncBeans

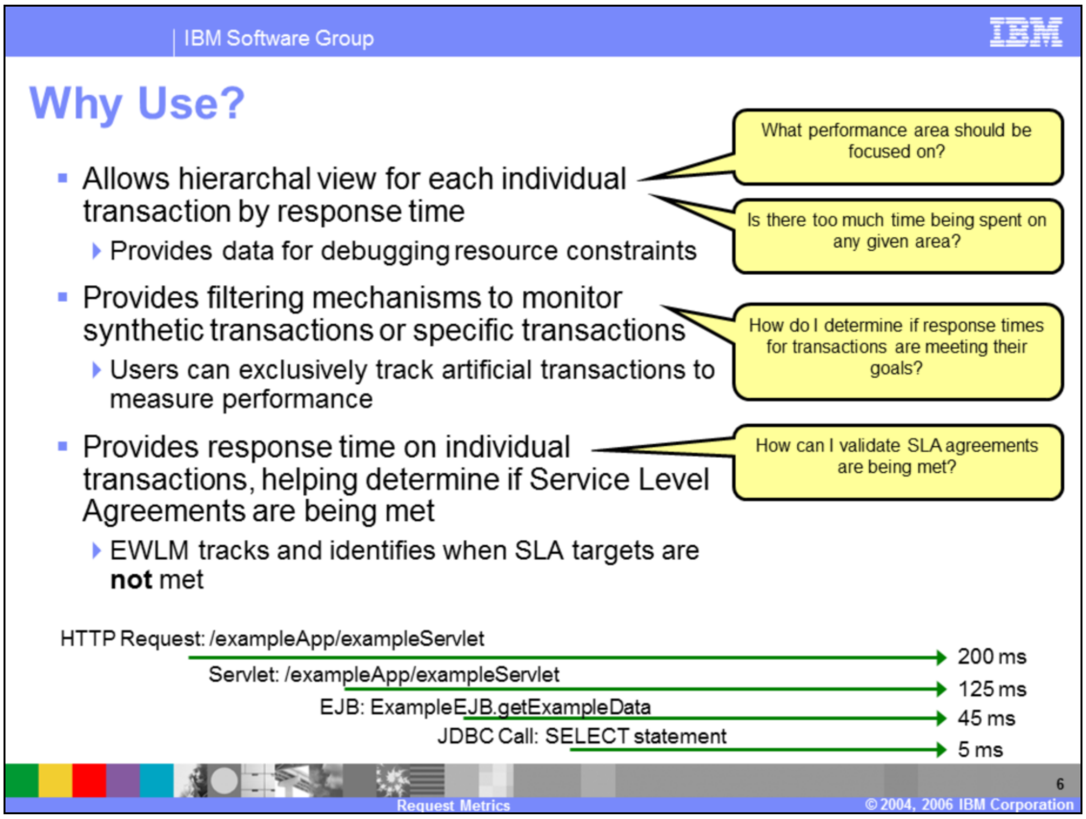
Time spent in Web Server
Time spent in Web Container
Time spent in EJB Container
Time spent making JDBC call
Web Server
Web Container
EJB Container
Data Source
JDBC
Database
Transaction A
e.g. Response time for Transaction A
Transaction B
e.g. Response time for Transaction B
Transaction C
e.g. Response time for Transaction C

5

Request Metrics © 2004, 2006 IBM Corporation

Request metrics allows you to monitor transactions as they move through the different components in an Application Server environment. Request metrics allows you to not only see the amount of time spent executing the whole request, but also the time spent executing the request in each of the supported Application Server components. The Application Server components that support request metrics functionality are the HTTP server plug-in, Web container, Web Services engine, EJB container, JDBC calls, Java™ Messaging Service and AsyncBeans. Additionally request metrics is not limited to monitoring transactions that occur on just one Application Server. Request metrics is capable of monitoring transactions as they fan out across a Application Server environment.

It is important to note that the request metrics data differs from Performance Monitoring Infrastructure or PMI data, in that request metrics data is transaction aware and PMI data is not. The PMI service collects performance data across all currently executing transactions, while request metrics collects performance data for individual transactions.


IBM Software Group

Why Use?

- Allows hierarchal view for each individual transaction by response time
  - Provides data for debugging resource constraints
- Provides filtering mechanisms to monitor synthetic transactions or specific transactions
  - Users can exclusively track artificial transactions to measure performance
- Provides response time on individual transactions, helping determine if Service Level Agreements are being met
  - EWLM tracks and identifies when SLA targets are **not** met

What performance area should be focused on?
Is there too much time being spent on any given area?
How do I determine if response times for transactions are meeting their goals?
How can I validate SLA agreements are being met?
HTTP Request: /exampleApp/exampleServlet
200 ms
Servlet: /exampleApp/exampleServlet
125 ms
EJB: ExampleEJB.getExampleData
45 ms
JDBC Call: SELECT statement
5 ms

6

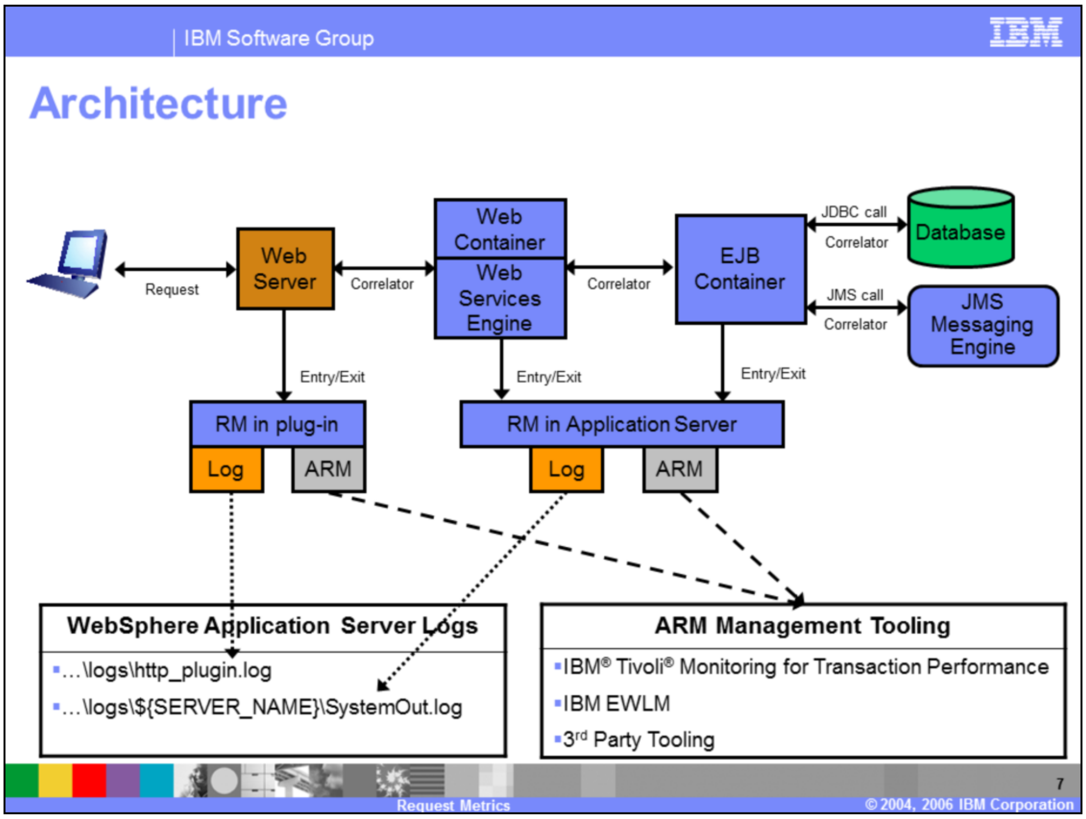
Request Metrics

© 2004, 2006 IBM Corporation

Request metrics provide valuable hierarchical data on transactions in your application. Because request metrics tracks response times across individual Application Server components or the whole application, it allows you to see if your application is executing according to design. The ability to view which components are involved with a transaction, also allows you to quickly determine which portion of your application may be to blame for poor performance. The example at the bottom of the slide represents what you may see when viewing request metrics data from your application. From the example, you can see exactly which components within your environment were involved with the request. Additionally you can see the amount of time spent in each one of these components and the amount of time overall spent responding to the request. Lastly, note that the response times at each level include the response times for the lower levels as well. This means that the time spent processing the request at the HTTP server was actually 75 milliseconds (200 minus 125).

Request metrics can be used to monitor the health and performance of your application through the use of synthetic transactions and appropriate application monitoring software. Synthetic transactions are transactions that are initiated from a known source (usually a system within your network designed to submit requests against your application) with the purpose of monitoring specific transactions. Synthetic transactions allow you to monitor your application's responsiveness and health. Along with system monitoring software, such as IBM Tivoli® Monitoring for Transaction Performance, a system administrator can use synthetic transactions and request metrics to alert them when response times are too low or transactions are failing.

Request metrics also provide valuable measurements for determining whether Service Level Agreements (SLAs) are being achieved. Because request metrics can show performance data for specific transactions in the system, it is an invaluable tool in determining whether your SLA targets are being met. In addition, request metrics can be paired with other tools, such as IBM Enterprise Workload Management (EWLM), which define business goals based on the performance data retrieved from request metrics. These business goals are then used when routing work in a business application environment.


IBM Software Group

Architecture

WebSphere Application Server Logs

- …\logs\http_plugin.log
- …\logs\${SERVER_NAME}\SystemOut.log

ARM Management Tooling

- IBM® Tivoli® Monitoring for Transaction Performance
- IBM EWLM
- 3rd Party Tooling

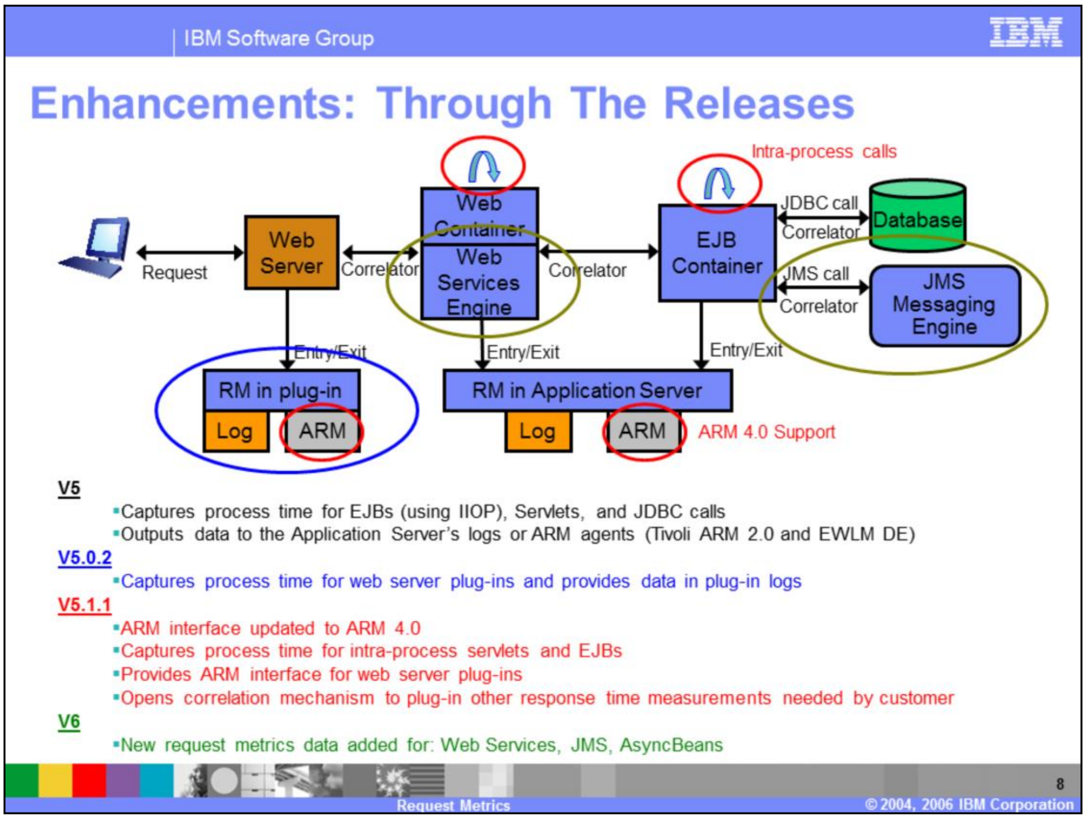
Request Metrics 7 © 2004, 2006 IBM Corporation

This slide depicts an architectural picture for the request metrics functionality. An example will be used to visualize what happens when a request enters a WebSphere Application Server. Note that in the diagram, dashed vertical lines represent filters that can be set in your Application Server environment. Request Metric filters control the types of incoming transactions that will be monitored. Request Metric filters will be discussed in detail in an upcoming section.

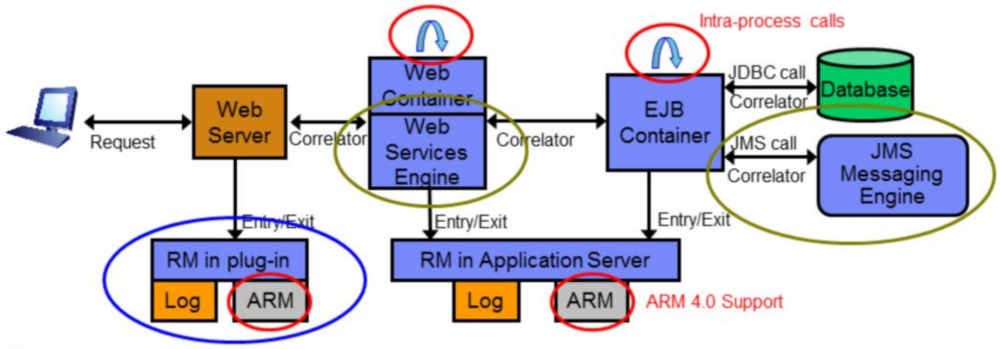
As a Web request enters the system, it will be examined by the filter settings in the web server plug-in. If the request matches one of the associated filters, an ARM transaction will be started and a correlator will be assigned in the web server plug-in. When the request moves into the Web container, another ARM transaction will be started and a correlator value will be assigned to the transaction. Additionally, the correlator value that was passed from the upstream container will be assigned as the ARM transaction's parent correlator – this allows the two separate transactions to later be correlated together. This behavior continues until the last point in the transactional chain is reached.

As the response leaves the system, it will stop the ARM transaction timers in each of the containers and record the appropriate transaction information and time in each of the activated logging mechanisms. Note that there are two possible options when logging request metrics data. You can either send it to the Application Server's logs and do the correlation yourself or you can send the data to an Application Response Measurement or ARM agent which will in turn externalize it to its management software. The Application Response Measurement standard will be discussed in an upcoming section.

In addition to what is shown in the picture above, WebSphere Application Server is also capable of passing the correlator from a request metrics transaction to a downstream ARM-enabled application that is not directly controlled by Application Server. This allows downstream non-WebSphere Application Server sub-transactions to be added to the currently in-flight request metrics transaction that took place within the Application Server. An example of this would be a transaction executing within the Application Server moving to an ARM-enabled backend, such as a CICS® server. The transaction correlator could be passed downstream so that non-WebSphere Application Server sub-transactions could be figured into the overall transaction's time.



Enhancements: Through The Releases



V5

- Captures process time for EJBs (using IIOP), Servlets, and JDBC calls
- Outputs data to the Application Server's logs or ARM agents (Tivoli ARM 2.0 and EWLM DE)

V5.0.2

- Captures process time for web server plug-ins and provides data in plug-in logs

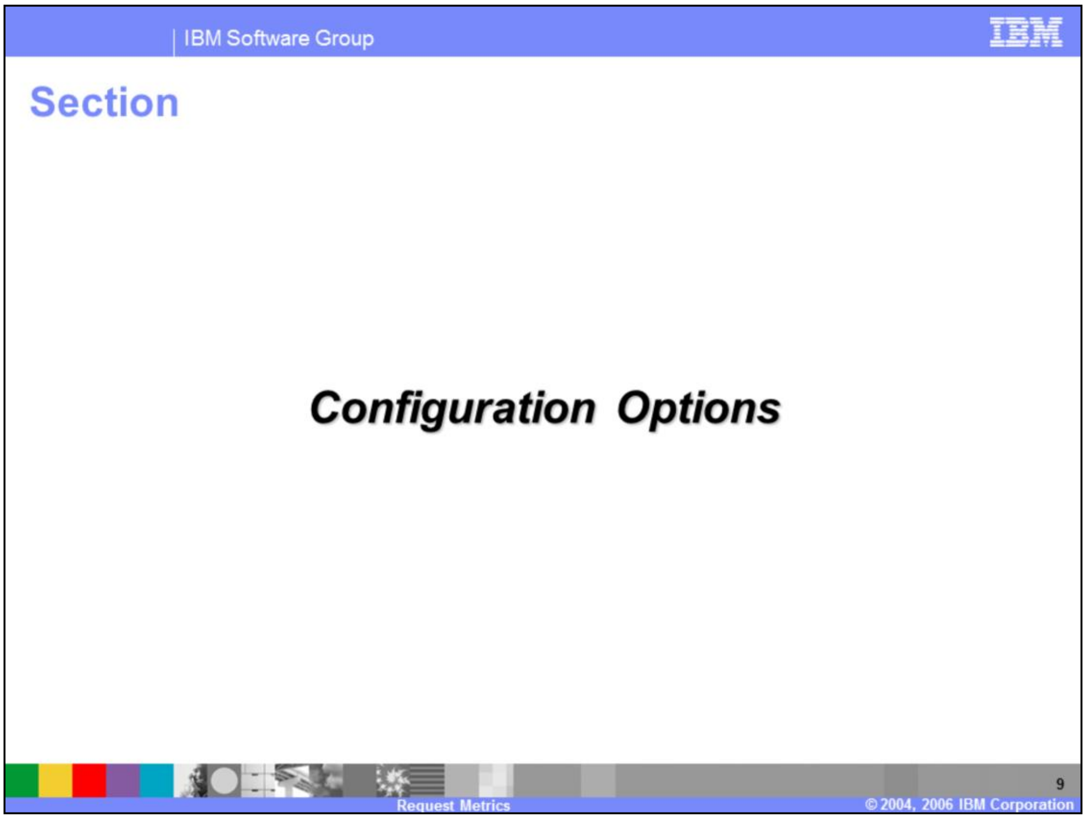
V5.1.1

- ARM interface updated to ARM 4.0
- Captures process time for intra-process servlets and EJBs
- Provides ARM interface for web server plug-ins
- Opens correlation mechanism to plug-in other response time measurements needed by customer

V6

- New request metrics data added for: Web Services, JMS, AsyncBeans

Request metrics was first introduced in WebSphere Application Server V5.0. Since then there have been minor enhancements with each release of WebSphere. In V6, request metrics support has been added to three new WebSphere Application Server components. These components are the Web Services engine, Java Messaging Service and AsyncBeans.



IBM Software Group

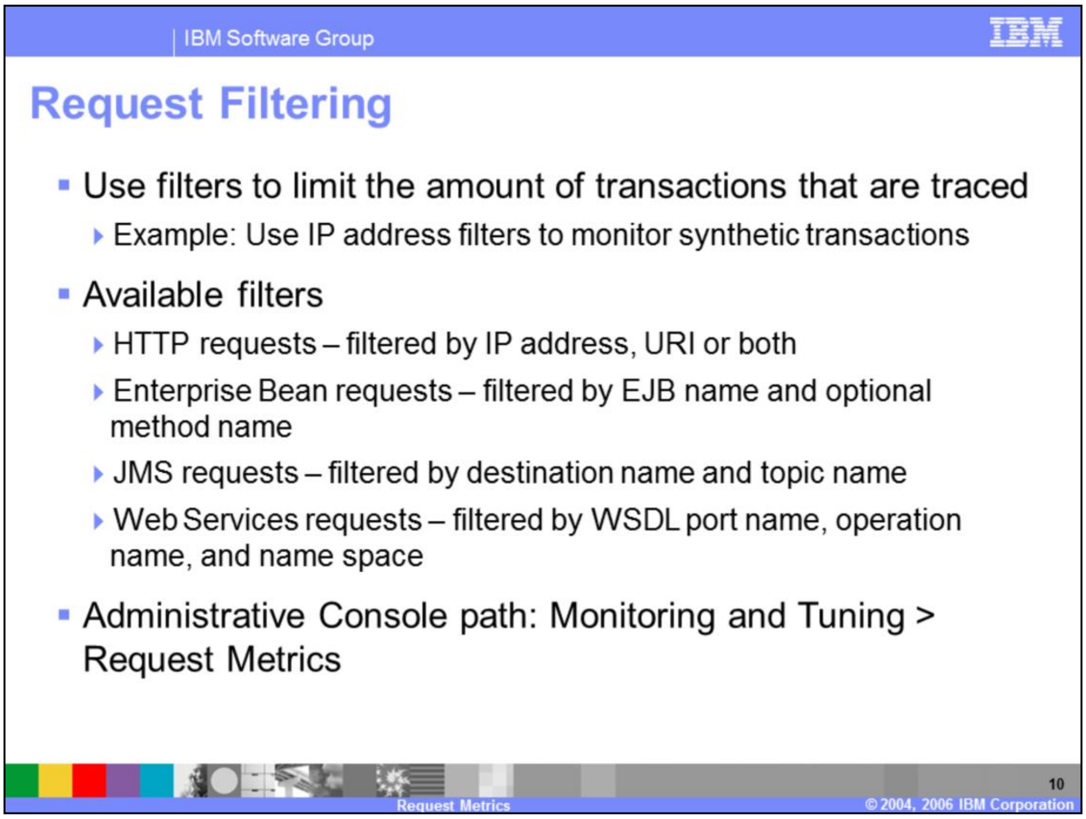
Section

Configuration Options

Request Metrics

9

The next section discusses what configuration options are available for request metrics.



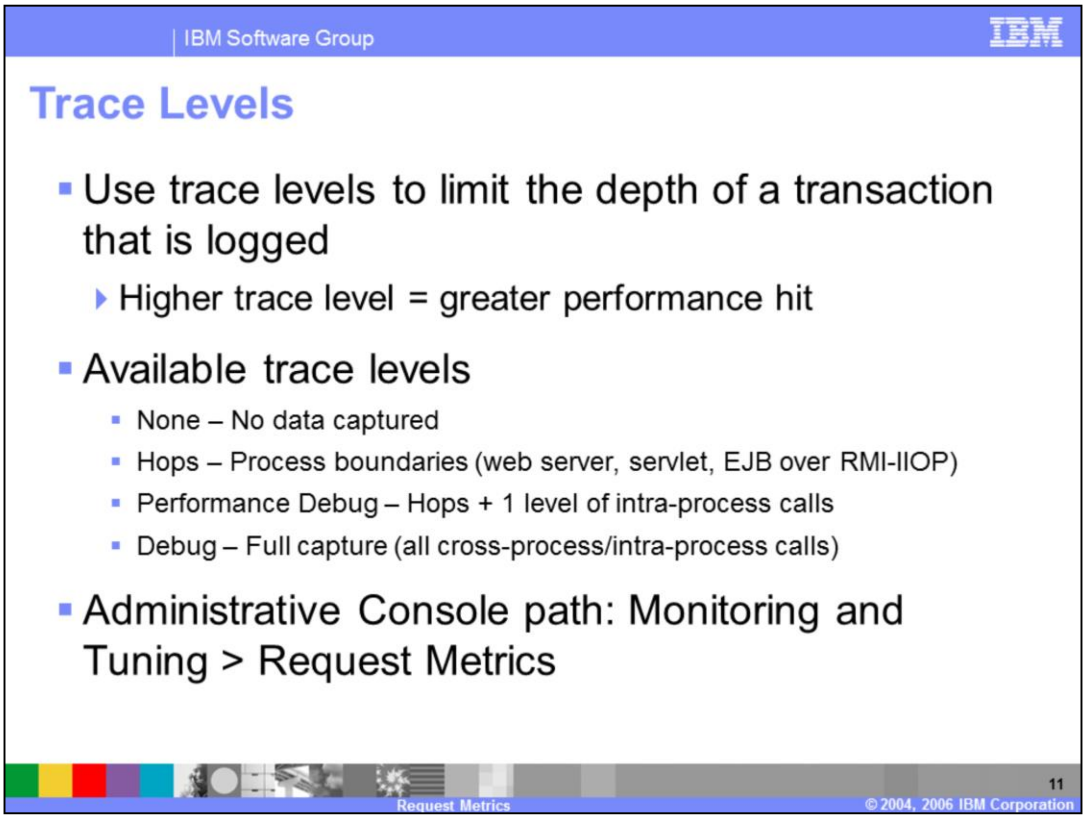
Request Filtering

- Use filters to limit the amount of transactions that are traced
  - Example: Use IP address filters to monitor synthetic transactions
- Available filters
  - HTTP requests – filtered by IP address, URI or both
  - Enterprise Bean requests – filtered by EJB name and optional method name
  - JMS requests – filtered by destination name and topic name
  - Web Services requests – filtered by WSDL port name, operation name, and name space
- Administrative Console path: Monitoring and Tuning > Request Metrics

Request metrics filters are used to limit the number of incoming requests that are monitored. In order for an incoming request to generate request metrics performance data it has to match one of the defined filters. If you do not define filters within the Application Server, all incoming requests will be monitored. An example of when you would want to use request metrics filters is monitoring synthetic transactions. In this example an IP address filter would be defined for the system that is used to generate synthetic transactions against your environment. This would result in the Application Server only monitoring requests that come from this system.

There are several types of request metrics filters available. Filters can be set for incoming HTTP requests, remote EJB requests, Web Services requests, and Java Messaging Service or JMS requests. HTTP requests can be filtered by IP address, URI or both. Remote EJB requests are filtered by EJB name and optionally the method name. JMS requests are filtered according to the specified destination and topic name. Web Services requests are filtered according to the specified WSDL port name, operation name, and name space.

The request metrics settings can be viewed and modified by navigating to Request Metrics under Monitoring and Tuning in the Administrative Console.


IBM Software Group

Trace Levels

- Use trace levels to limit the depth of a transaction that is logged
  - Higher trace level = greater performance hit
- Available trace levels
  - None – No data captured
  - Hops – Process boundaries (web server, servlet, EJB over RMI-IIOP)
  - Performance Debug – Hops + 1 level of intra-process calls
  - Debug – Full capture (all cross-process/intra-process calls)
- Administrative Console path: Monitoring and Tuning > Request Metrics

11

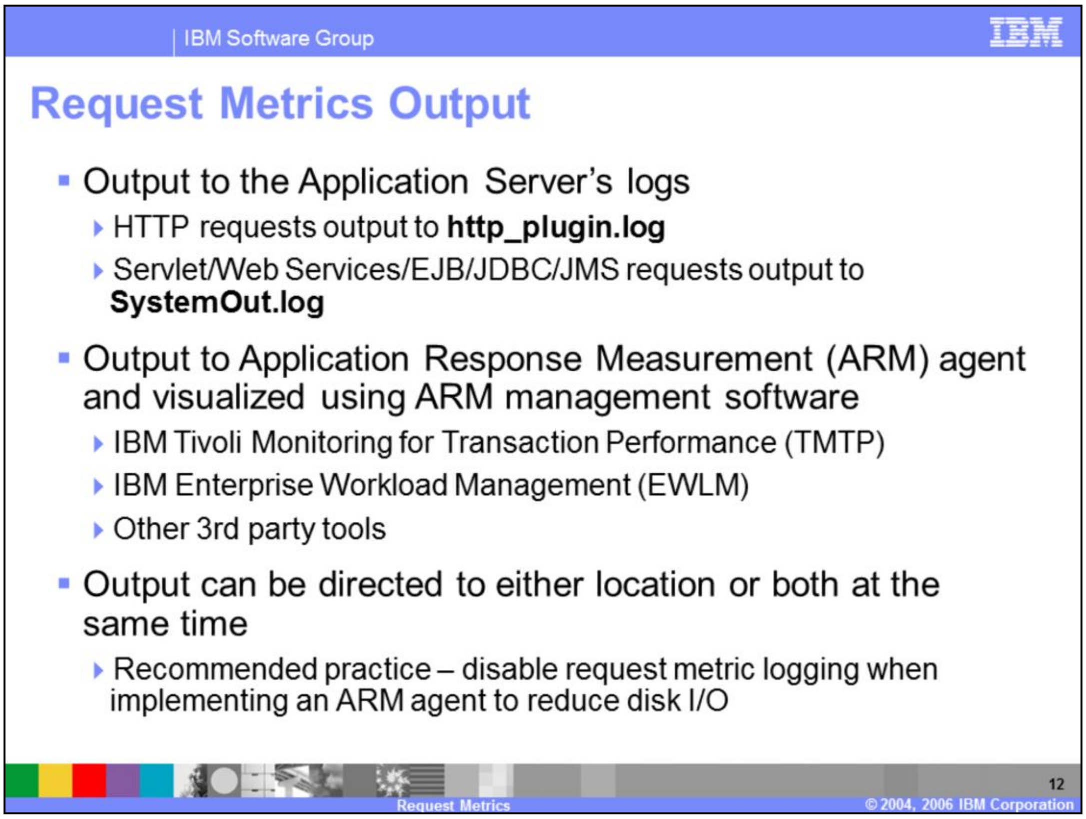
Request Metrics

© 2004, 2006 IBM Corporation

Trace levels are used to limit the depth of request metrics data that are collected. It is important to note that the higher the trace level that you set the greater the performance hit on your WebSphere Application Server.

There are four trace levels that are available. The trace level setting of “None” results in no data being captured. This has the same effect of disabling request metrics on the server. The trace level setting of “Hops” captures requests when they move between processes. This will not collect request metrics data for a servlet calling an EJB residing in the same JVM. The trace level setting of “Performance Debug” captures the same amount of data as Hops plus it captures one level of intra-process calls. This will collect the request metrics data for a servlet calling an EJB in same JVM, but won’t capture the EJB calling another EJB in same JVM. The trace level setting of “Debug” results in all cross process and intra-process calls being captured. This should only be used in extreme cases. In most cases Hops or Performance Debug should capture sufficient data.

The request metrics settings can be viewed and modified by navigating to Request Metrics under Monitoring and Tuning in the Administrative Console.



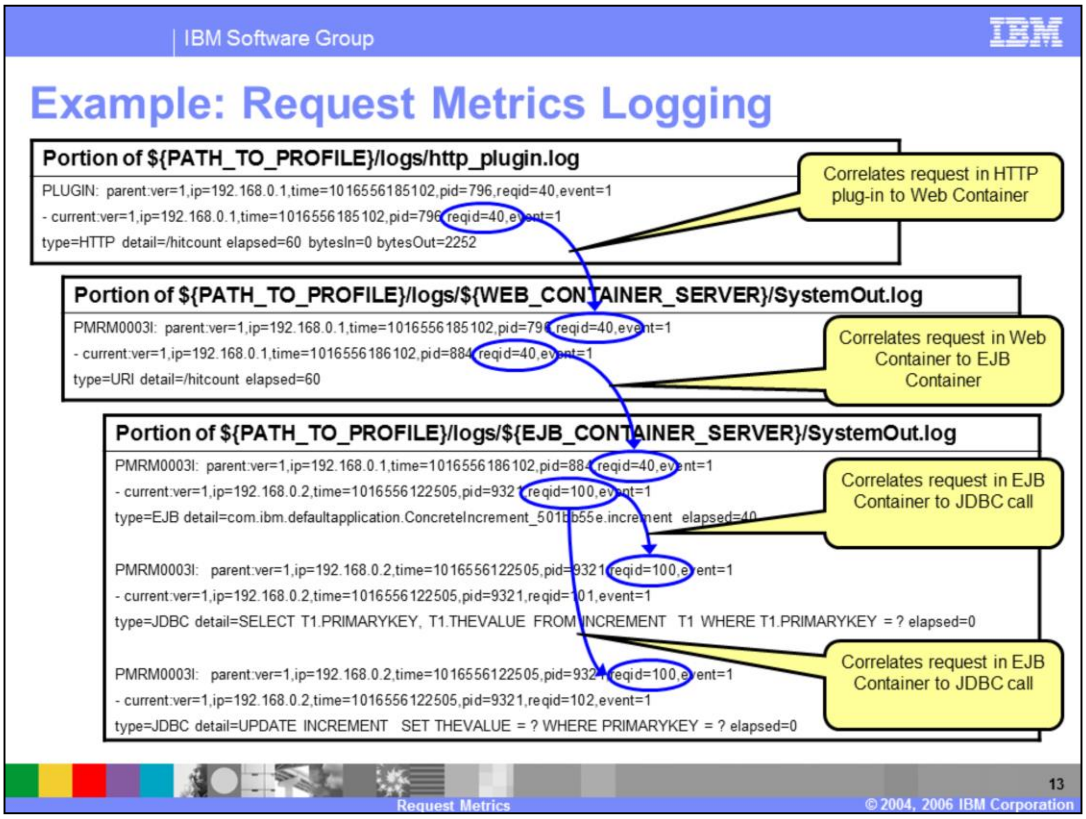
Request Metrics Output

- Output to the Application Server's logs
  - HTTP requests output to **http_plugin.log**
  - Servlet/Web Services/EJB/JDBC/JMS requests output to **SystemOut.log**
- Output to Application Response Measurement (ARM) agent and visualized using ARM management software
  - IBM Tivoli Monitoring for Transaction Performance (TMTP)
  - IBM Enterprise Workload Management (EWLM)
  - Other 3rd party tools
- Output can be directed to either location or both at the same time
  - Recommended practice – disable request metric logging when implementing an ARM agent to reduce disk I/O

Request metrics data can be output to two different locations. The first location is the Application Server's log files. If the request is a Web request and it matches the defined filters, request metric data is placed in the HTTP plug-in log. The rest of the components that support request metrics, output their data to the server's SystemOut.log file. After the data is output it is up to you to correlate the sub-transactions together to visualize the whole transaction.

The second location that request metric data can be output is an Application Response Measurement agent. Once data is written to the ARM agent it externalizes it to its associated ARM management software. It is the responsibility of the ARM management software to analyze and visualize the collected data into an understandable format. It is important to note that WebSphere Application Server does not provide an ARM agent, this is the responsibility of the tool vendors that utilize the ARM agent. Examples of IBM products that have implemented the ARM standard and that can be used with request metrics are IBM Tivoli Monitoring for Transaction Performance and IBM Enterprise Workload Management.

It is important to note that you can output request metric data to either of the locations at the same time. It is, however, recommended that when outputting data to an ARM agent that you disable request metric logging. This should be done to minimize the amount of disk I/O that is encountered when writing request metric data to the disk.


IBM Software Group

Example: Request Metrics Logging

Portion of ${PATH_TO_PROFILE}/logs/http_plugin.log

```
PLUGIN:  parent:ver=1,ip=192.168.0.1,time=1016556185102,pid=796,reqid=40,event=1
- current:ver=1,ip=192.168.0.1,time=1016556185102,pid=796,reqid=40,event=1
type=HTTP  detail=/hitcount elapsed=60  bytesIn=0 bytesOut=2252
```

Correlates request in HTTP plug-in to Web Container

Portion of ${PATH_TO_PROFILE}/logs/${WEB_CONTAINER_SERVER}/SystemOut.log

```
PMRM0003I:  parent:ver=1,ip=192.168.0.1,time=1016556185102,pid=796,reqid=40,event=1
- current:ver=1,ip=192.168.0.1,time=1016556186102,pid=884,reqid=40,event=1
type=URI detail=/hitcount elapsed=60
```

Portion of ${PATH_TO_PROFILE}/logs/${EJB_CONTAINER_SERVER}/SystemOut.log

```
PMRM0003I:  parent:ver=1,ip=192.168.0.1,time=1016556186102,pid=884,reqid=40,event=1
- current:ver=1,ip=192.168.0.2,time=1016556122505,pid=9321,reqid=100,event=1
type=EJB detail=com.ibm.defaultapplication.ConcreteIncrement_501bb55e.increment  elapsed=40

PMRM0003I:   parent:ver=1,ip=192.168.0.2,time=1016556122505,pid=9321,reqid=100,event=1
- current:ver=1,ip=192.168.0.2,time=1016556122505,pid=9321,reqid=101,event=1
type=JDBC detail=SELECT T1.PRIMARYKEY,  T1.THEVALUE  FROM INCREMENT   T1  WHERE T1.PRIMARYKEY  = ? elapsed=0

PMRM0003I:   parent:ver=1,ip=192.168.0.2,time=1016556122505,pid=9321,reqid=100,event=1
- current:ver=1,ip=192.168.0.2,time=1016556122505,pid=9321,reqid=102,event=1
type=JDBC detail=UPDATE  INCREMENT   SET THEVALUE  = ? WHERE PRIMARYKEY  = ? elapsed=0
```

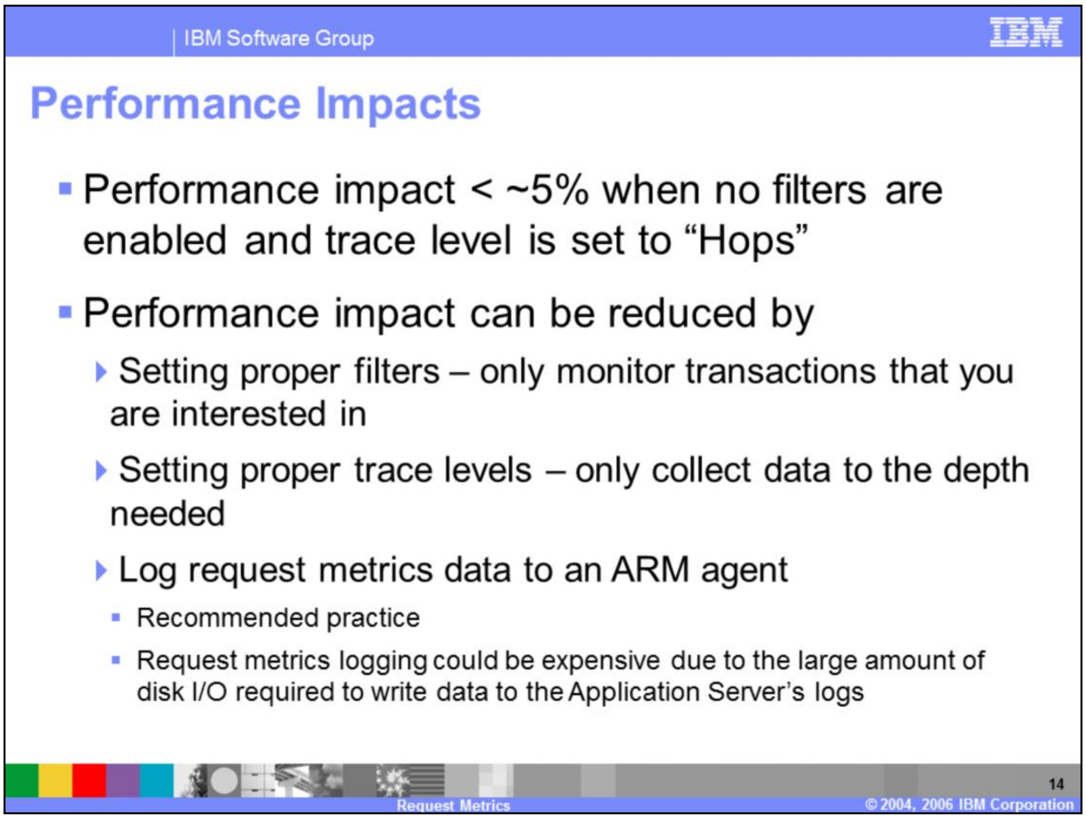
Correlates request in EJB Container to JDBC call
Correlates request in EJB Container to JDBC call

13

Request Metrics

© 2004, 2006 IBM Corporation

The graphic on this slide displays what is output to the Application Server's logs when request metric logging is enabled. The important thing to note in this slide is how the correlator identifier is passed as the request moves from the HTTP plug-in to the Web container to EJB container to the backend JDBC calls. The passed correlator allows you to view not only the whole transaction but the sub-transactions that make up the whole transaction.

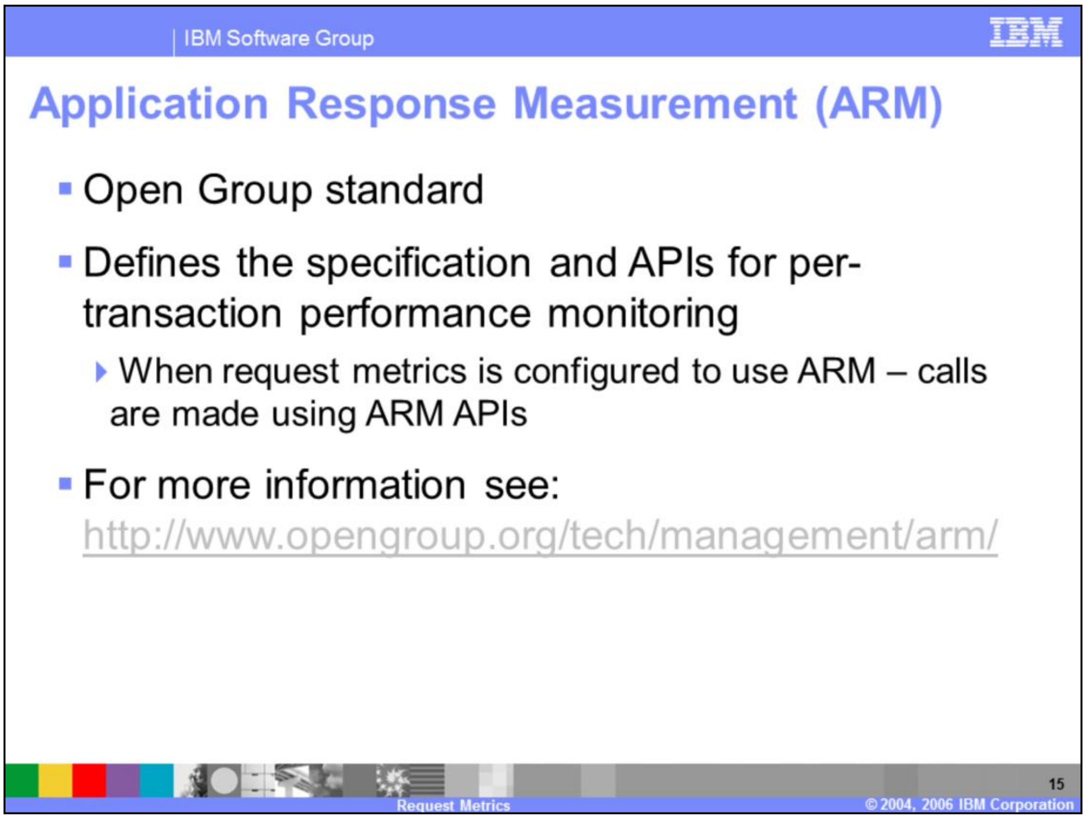


Performance Impacts

- Performance impact < ~5% when no filters are enabled and trace level is set to "Hops"
- Performance impact can be reduced by
  - Setting proper filters – only monitor transactions that you are interested in
  - Setting proper trace levels – only collect data to the depth needed
  - Log request metrics data to an ARM agent
    - Recommended practice
    - Request metrics logging could be expensive due to the large amount of disk I/O required to write data to the Application Server's logs

The performance impact of enabling request metrics on your Application Server has been improved in V6. The performance impact on the server is less than approximately five percent when no filters have been enabled and a trace level of "Hops" has been set. This is a significant improvement over V5.

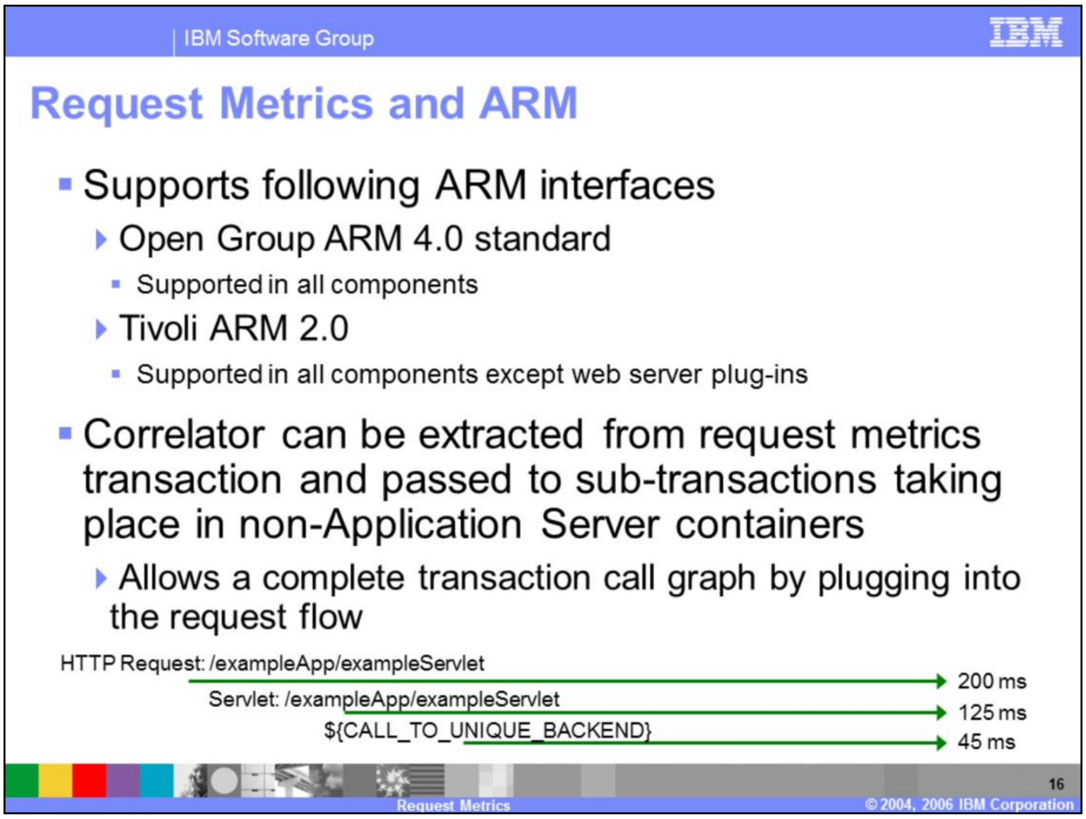
To further reduce the performance impact incurred by your Application Server, additional steps can be taken. It is recommended that you enable filters for only the transactions that you want to monitor. Additionally, set the trace level to a setting that allows you to trace performance data to the depth required. Lastly, log request metrics data to an ARM agent. Logging data to the ARM agent is the recommended practice when using request metrics and is also less expensive than request metrics logging. Remember request metrics logging is expensive due to the amount of disk I/O necessary to write the request metrics data to the Application Server's logs.



Application Response Measurement (ARM)

- Open Group standard
- Defines the specification and APIs for per-transaction performance monitoring
  - When request metrics is configured to use ARM – calls are made using ARM APIs
- For more information see: http://www.opengroup.org/tech/management/arm/

Application Response Measurement is an Open group standard. ARM defines a specification and set of APIs that should be used for per-transaction performance monitoring. The ARM APIs are used when request metrics are configured to output data using the ARM agent. If you want to learn more about the ARM standard and its API visit the link specified in the slide.



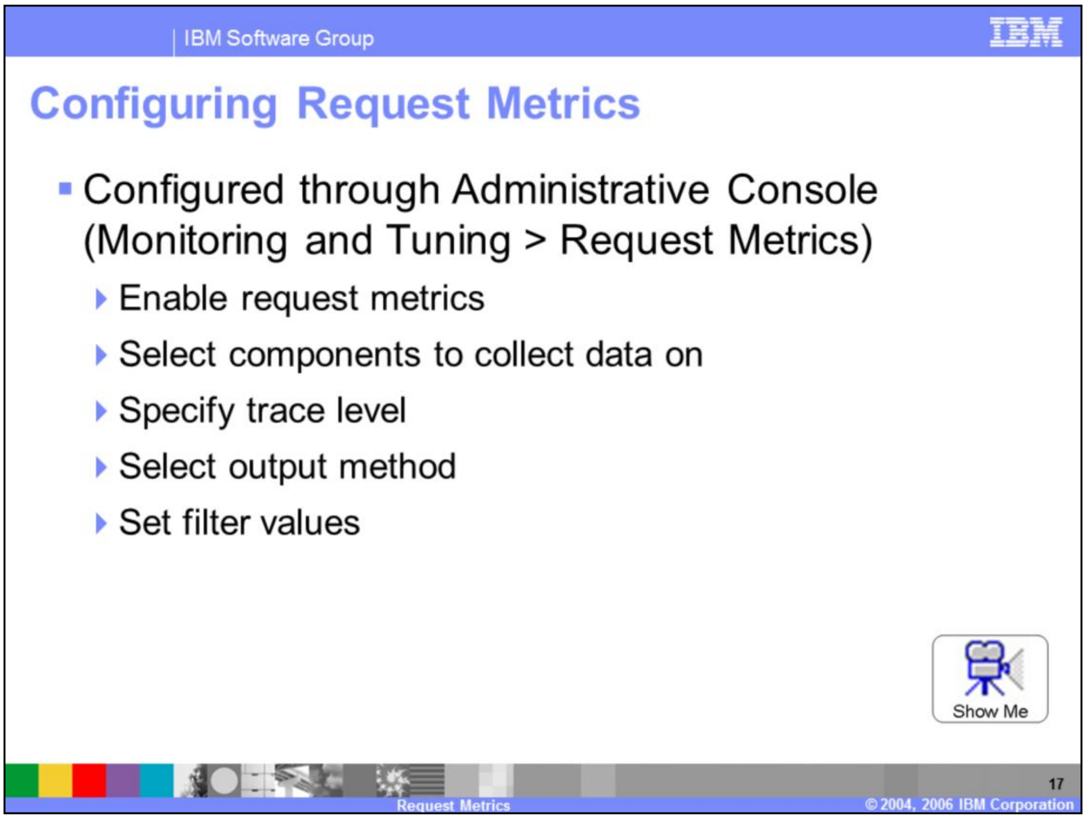
Request Metrics and ARM

- Supports following ARM interfaces
  - Open Group ARM 4.0 standard
    - Supported in all components
  - Tivoli ARM 2.0
    - Supported in all components except web server plug-ins
- Correlator can be extracted from request metrics transaction and passed to sub-transactions taking place in non-Application Server containers
  - Allows a complete transaction call graph by plugging into the request flow

HTTP Request: /exampleApp/exampleServlet
200 ms
Servlet: /exampleApp/exampleServlet
125 ms
${CALL_TO_UNIQUE_BACKEND}
45 ms

Request metrics supports two different ARM specifications. It currently supports the Open Group ARM 4.0 standard within all of its components that support request metrics collection. It also contains support for Tivoli ARM 2.0 within all of its components that support request metrics collection except for the Web server plug-ins.

Additionally the request metrics correlator can be extracted from a currently in-flight request metrics transaction and passed to a non-WebSphere Application Server container. This allows you to receive accurate time for the whole transaction even as it moves across application boundaries. In order for this to work, however, the downstream application has to be ARM-enabled. An example of where this would be useful is a business application with a WebSphere Application Server front end and a proprietary ARM-enabled backend. As the transaction prepares to leave your Application Server environment you would extract the correlator ID from the request metrics transaction and pass it to the proprietary backend. The proprietary backend would then continue its processing of the request, continuing to pass the correlator along to subsequent sub-transactions. After the transaction has finished being processed, the ARM management software would then be able to combine the sub-transactions taken from both the Application Server and the proprietary backend and draw a complete picture.


IBM Software Group

Configuring Request Metrics

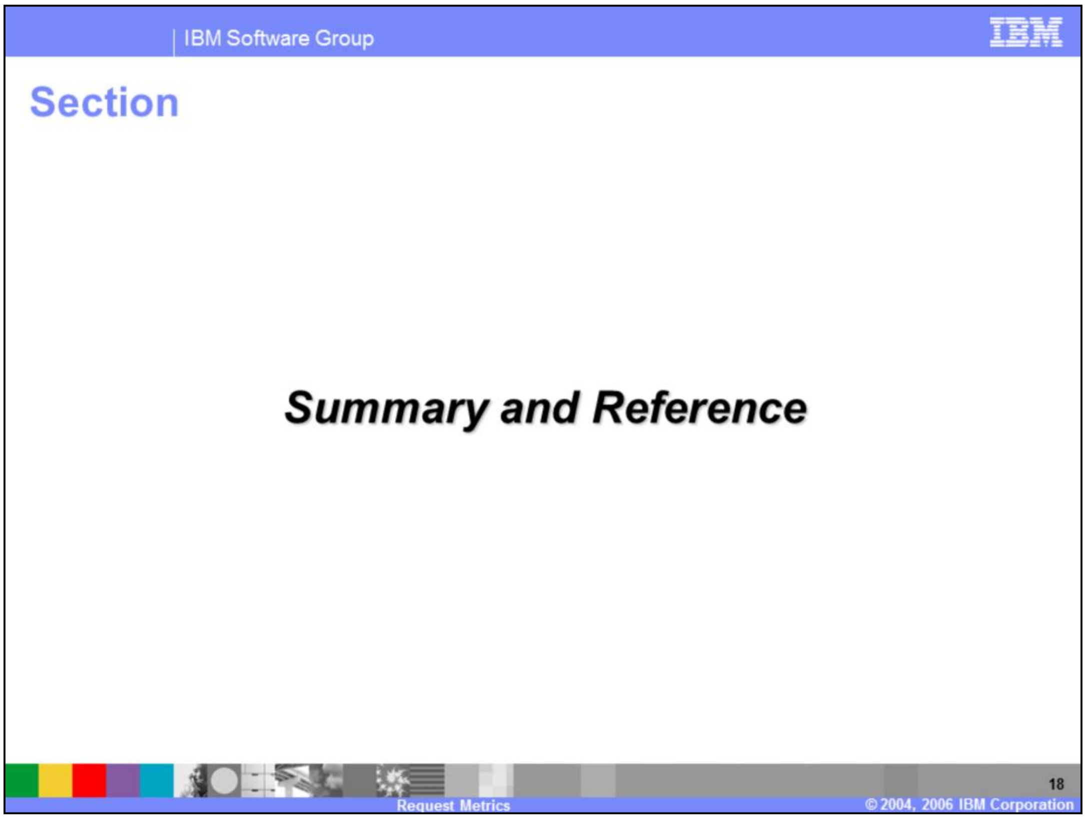
- Configured through Administrative Console (Monitoring and Tuning > Request Metrics)
  - Enable request metrics
  - Select components to collect data on
  - Specify trace level
  - Select output method
  - Set filter values

17

Request Metrics

© 2004, 2006 IBM Corporation

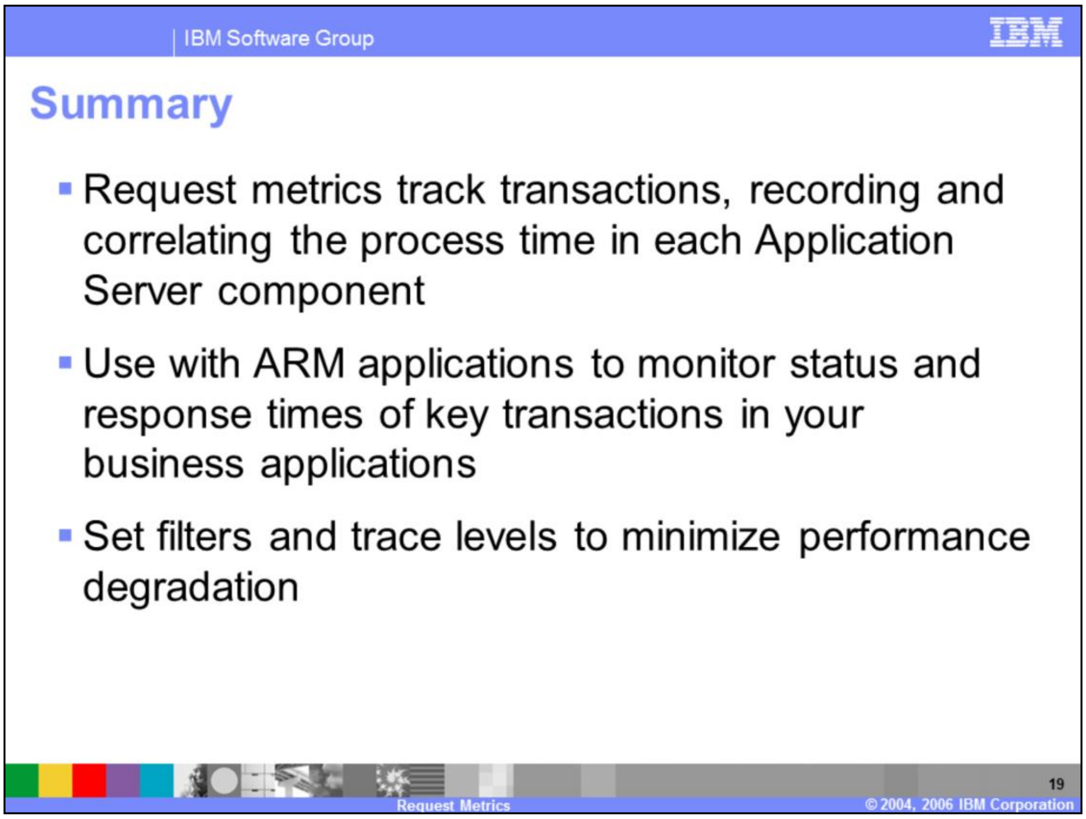
The demonstration that is available on this slide takes you through the different configuration options and where they are located within the Administrative Console. To access the demonstration, pause this presentation and click the Show Me icon.



Section

Summary and Reference

The next section will summarize the points discussed in this lecture.


IBM Software Group

IBM

Summary

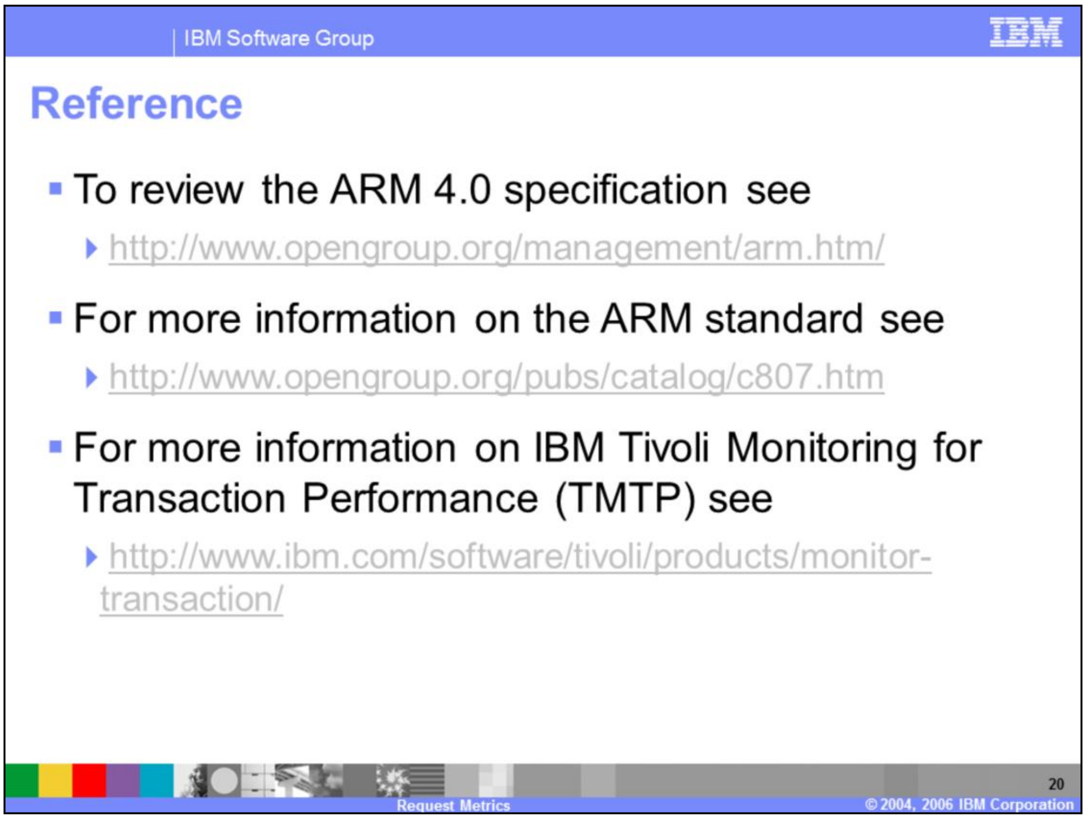
- Request metrics track transactions, recording and correlating the process time in each Application Server component
- Use with ARM applications to monitor status and response times of key transactions in your business applications
- Set filters and trace levels to minimize performance degradation

19

Request Metrics

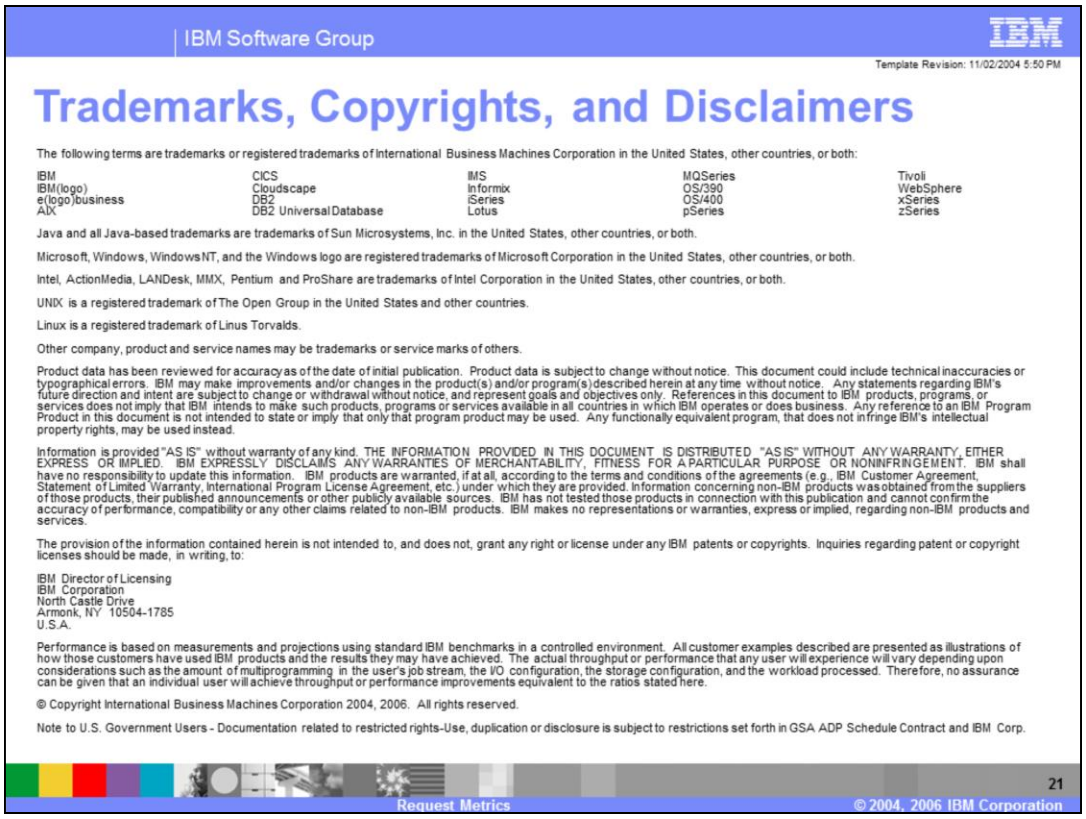
© 2004, 2006 IBM Corporation

In summary, this presentation has focused on the request metrics functionality provided with WebSphere Application Server. Request metrics allows you to track individual transaction's performance as they move through the Application Server environment. Request metrics is most effective when used in conjunction with an ARM application. Lastly, filters and trace levels should be used to minimize the performance impact that request metrics has on your Application Server.



Reference

- To review the ARM 4.0 specification see
  - http://www.opengroup.org/management/arm.htm/
- For more information on the ARM standard see
  - http://www.opengroup.org/pubs/catalog/c807.htm
- For more information on IBM Tivoli Monitoring for Transaction Performance (TMTP) see
  - http://www.ibm.com/software/tivoli/products/monitor-transaction/



Template Revision: 11/02/2004 5:50 PM

Trademarks, Copyrights, and Disclaimers