

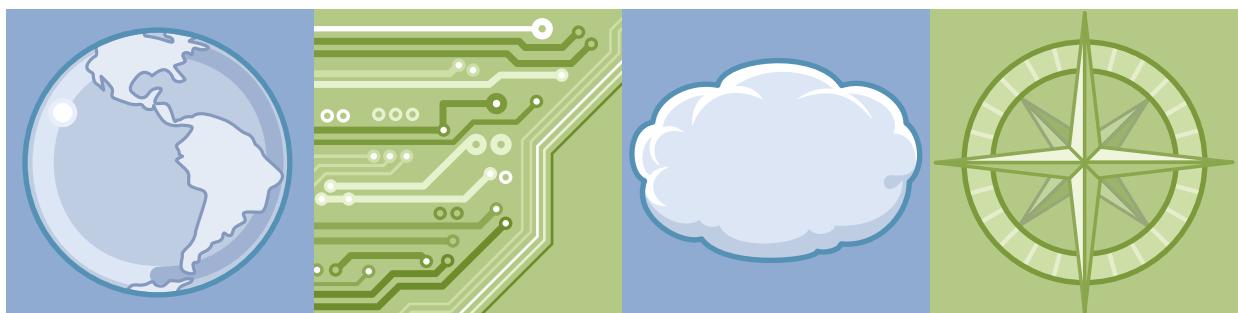


IBM Training

Student Notebook

WebSphere Application Server V8.5.5 Performance Tuning

Course code WA815 ERC 2.2



IBM Cloud

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide.

The following are trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide:

AFS™	AIX®	CICS®
DB™	DB2®	developerWorks®
Express®	HACMP™	IMS™
Informix®	Insight™	MVS™
PartnerWorld®	RACF®	Rational®
RDN®	Redbooks®	Tivoli®
WebSphere®	WPM®	z/OS®

Intel is a trademark or registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java™ and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

VMware and the VMware "boxes" logo and design, Virtual SMP and VMotion are registered trademarks or trademarks (the "Marks") of VMware, Inc. in the United States and/or other jurisdictions.

Performance View® is a trademark or registered trademark of Kenexa, an IBM Company.

Other product and service names might be trademarks of IBM or other companies.

June 2017 edition

The information contained in this document has not been submitted to any formal IBM test and is distributed on an "as is" basis without any warranty either express or implied. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will result elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Contents

Trademarks	xv
Course description	xix
Agenda	xxi
Unit 1. WebSphere Application Server systems and components	1-1
Unit objectives	1-2
Topics	1-3
1.1. Components and services	1-5
Components and services	1-6
WebSphere Application Server architecture	1-7
Web container	1-9
Enterprise JavaBeans (EJB) container	1-11
Object Request Broker service	1-13
Java EE Connector Architecture service (JCA)	1-14
WebSphere default messaging	1-16
Security service	1-17
1.2. Network Deployment components	1-19
Network Deployment components	1-20
Network Deployment concepts	1-21
Runtime flow	1-22
Administration flow	1-23
File synchronization and file transfer	1-24
Clustered application servers	1-26
Web servers	1-27
Web server plug-ins	1-28
Web servers: Managed nodes and unmanaged nodes	1-29
Edge Components	1-30
Liberty profile	1-31
Liberty profile: Composable runtime	1-32
Intelligent Management (1 of 2)	1-33
Intelligent Management (2 of 2)	1-34
1.3. Client request flow	1-37
Client request flow	1-38
Typical client request application flow	1-39
Possible performance bottlenecks	1-41
Unit summary	1-42
Checkpoint questions (1 of 2)	1-43
Checkpoint questions (2 of 2)	1-44
Checkpoint answers	1-45
Unit 2. Overview of performance concepts and tuning tasks	2-1
Unit objectives	2-2
Topics	2-3
2.1. Performance terminology	2-5
Performance terminology	2-6
Performance terms	2-7

Response time	2-8
Response time: Website example	2-9
Response time critical measurement	2-10
What is the meaning of website load?	2-11
Throughput	2-12
Maximum throughput	2-13
Throughput saturation	2-14
Response time and throughput relationship	2-15
Response time and throughput saturation	2-16
Saturated cafeteria example	2-17
Improving performance	2-18
Remove transaction steps: Website	2-19
Speed up step: Website	2-20
Path length	2-21
Path length reduction techniques	2-22
What is a bottleneck?	2-23
Bottleneck website example 1	2-24
Bottleneck website example 2	2-25
Prioritizing bottlenecks	2-26
What is scalability?	2-27
Scalability: Website example	2-28
What is capacity?	2-29
Capacity example: Website	2-30
Performance terminology summary	2-31
2.2. Different views of performance	2-33
Different views of performance	2-34
Different views of performance	2-35
User view concerns	2-36
System view	2-37
System view concerns (1 of 2)	2-38
System view concerns (2 of 2)	2-39
Application view	2-40
2.3. Test planning and design.	2-41
Test planning and design	2-42
Performance tasks: What to do and when to do it?	2-43
Test planning and design objectives	2-44
Test planning and design considerations	2-46
Test plan creation steps	2-47
Workload analysis steps (1 of 2)	2-48
Workload analysis steps (2 of 2)	2-50
Test environment setup considerations	2-51
2.4. Code analysis and profiling	2-53
Code analysis and profiling	2-54
Code analysis and profiling overview	2-55
Example: Rational Application Developer code review	2-56
Runtime analysis	2-57
Application profiling tools	2-58
2.5. Performance testing and tuning	2-59
Performance testing and tuning	2-60
Performance testing and tuning overview	2-61
What is performance testing?	2-62

Typical goals of performance testing	2-63
Types of performance tests	2-65
Test execution	2-66
Results analysis and problem resolution	2-68
Performance test development and load-driving tools	2-69
2.6. Production monitoring and tuning	2-71
Production monitoring and tuning	2-72
Performance monitoring overview	2-73
Categories of parameters to monitor	2-75
Unit summary	2-76
Checkpoint questions	2-77
Checkpoint answers	2-78
Exercise 1	2-79
Exercise objectives	2-80
Exercise instructions	2-81
Unit 3. Apache JMeter and load testing	3-1
Unit objectives	3-2
Topics	3-3
3.1. Load testing tools	3-5
Load testing tools	3-6
Reasons for load testing	3-7
Factors in load testing	3-8
Stages of load testing	3-9
Selecting load testing tools	3-10
Selecting load testing tools: More details (1 of 4)	3-11
Selecting load testing tools: More details (2 of 4)	3-12
Selecting load testing tools: More details (3 of 4)	3-13
Selecting load testing tools: More details (4 of 4)	3-14
3.2. The basics of using Apache JMeter	3-15
The basics of using Apache JMeter	3-16
What is Apache JMeter?	3-17
JMeter Test Plan view	3-19
Thread Group view: Create a thread group	3-20
Add an HTTP Request sampler	3-21
Add a timer	3-23
Add a Summary Report listener	3-24
Summary Report view	3-25
Running the test	3-26
Monitor system metrics	3-27
Use a ramp-up test to determine the saturation point	3-28
Record test results example	3-29
Graph: Ramp-up test results	3-30
Using the ramp-up test for further testing	3-31
3.3. Introduction to DayTrader	3-33
Introduction to DayTrader	3-34
DayTrader benchmark application (1 of 3)	3-35
DayTrader benchmark application (2 of 3)	3-36
DayTrader benchmark application (3 of 3)	3-37
DayTrader configuration	3-38
DayTrader compared to your application	3-39

Unit summary	3-40
Checkpoint questions	3-41
Checkpoint answers	3-42
Exercise 2	3-43
Exercise objectives	3-44
Exercise 3	3-45
Exercise objectives	3-46
Exercise 4	3-47
Exercise objectives	3-48
Unit 4. WebSphere performance data and tools	4-1
Unit objectives	4-2
Topics	4-3
4.1. WebSphere performance data.....	4-5
WebSphere performance data	4-6
High-level performance data areas	4-7
Hardware performance data	4-8
Operating system performance data	4-9
JVM performance data	4-10
Sources of JVM performance data	4-11
Web server performance data	4-12
Application server performance data	4-13
Messaging performance data	4-14
Database performance data	4-15
4.2. WebSphere performance tools	4-17
WebSphere performance tools	4-18
Performance monitoring tools	4-19
Analysis and monitoring tools: vmstat (1 of 2)	4-20
Analysis and monitoring tools: vmstat (2 of 2)	4-21
Key metrics from vmstat	4-23
Analysis and monitoring tools: nmon	4-24
nmon output of CPU and memory statistics	4-25
nmon output of kernel statistics and top processes	4-27
Web server performance tools: IBM HTTP Server server-status (1 of 3)	4-29
Web server performance tools: IBM HTTP Server server-status (2 of 3)	4-31
Web server performance tools: IBM HTTP Server server-status (3 of 3)	4-33
Web server performance tools: Web server access logs (1 of 4)	4-34
Web server performance tools: Web server access logs (2 of 4)	4-35
Web server performance tools: Web server access logs (3 of 4)	4-36
Web server performance tools: Web server access logs (4 of 4)	4-37
JVM performance tools: Verbose garbage collection	4-38
JVM performance tools: GCMV	4-40
JVM performance tools: Thread dumps (1 of 2)	4-41
JVM performance tools: Thread dumps (2 of 2)	4-42
Thread and Monitor Dump Analyzer (TMDA)	4-43
Messaging performance tools	4-45
WebSphere performance data: Performance Monitoring Infrastructure (PMI)	4-46
WebSphere performance tools: Tivoli Performance Viewer (1 of 2)	4-47
WebSphere performance tools: Tivoli Performance Viewer (2 of 2)	4-48
WebSphere performance tools: Request metrics overview	4-49
WebSphere performance tools: Request metrics (1 of 2)	4-50

WebSphere performance tools: Request metrics (2 of 2)	4-51
A final word on performance tools	4-52
Unit summary	4-53
Checkpoint questions	4-54
Checkpoint answers	4-55
Exercise 5	4-56
Exercise objectives	4-57
Unit 5. WebSphere performance tuning methods	5-1
Unit objectives	5-2
Topics	5-3
5.1. Performance tuning in practice	5-5
Performance tuning in practice	5-6
Preparation for performance tuning	5-7
Performance tuning goals	5-8
Application server parameter tuning	5-9
Typical Java EE application topology	5-10
What influences tuning?	5-12
Once in production	5-13
5.2. Gathering performance data	5-15
Gathering performance tuning data	5-16
Required performance metric	5-17
Required performance metrics: Details (1 of 2)	5-18
Required performance metrics: Details (2 of 2)	5-19
The lifecycle of performance monitoring and tuning	5-20
5.3. Top 10 monitoring hot list for WebSphere performance tuning	5-21
Top 10 monitoring hot list for WebSphere performance tuning	5-22
Top 10: Monitoring hot list	5-23
General tuning considerations (1 of 2)	5-24
General tuning considerations (2 of 2)	5-25
How to find metrics for servlets and EJBs (1 of 2)	5-26
How to find metrics for servlets and EJBs (2 of 2)	5-27
How to find metrics for thread pools and data sources	5-28
How to find metrics for JVM and system resources	5-29
5.4. Detecting bottlenecks	5-31
Detecting bottlenecks	5-32
Detecting bottlenecks (1 of 2)	5-33
Detecting bottlenecks (2 of 2)	5-34
Bottleneck considerations (1 of 4)	5-35
Bottleneck considerations (2 of 4)	5-36
Bottleneck considerations (3 of 4)	5-37
Bottleneck considerations (4 of 4)	5-38
5.5. System queues	5-39
System queues	5-40
System queues: The queuing network	5-41
WebSphere tuning: Upstream queuing	5-42
How to configure system queues (1 of 2)	5-43
How to configure system queues (2 of 2)	5-44
Determining optimum queue sizes example (1 of 2)	5-45
Determining optimum queue sizes example (2 of 2)	5-46
Determining optimum queue sizes: The throughput curve	5-47

Determining the maximum concurrency point	5-48
Unit summary	5-49
Checkpoint questions	5-50
Checkpoint answers	5-51
Unit 6. Introduction to the JVM	6-1
Unit objectives	6-2
Topics	6-3
6.1. JVM introduction	6-5
JVM introduction	6-6
Java virtual machine (JVM) features	6-7
Just-in-time compiler (JIT) basics	6-8
Ahead-Of-Time (AOT) compiler basics	6-9
JVM version (1 of 2)	6-10
JVM version (2 of 2)	6-11
Using IBM Java 7	6-12
JVM overview	6-13
JVM memory segments	6-15
6.2. Garbage collection.	6-17
Garbage collection	6-18
What is garbage collection?	6-19
Object reclamation (garbage collection)	6-20
Understanding garbage collection (GC)	6-21
IBM JDK GC process	6-22
Garbage collection phases: Mark, sweep, compact	6-24
Garbage collection policies	6-25
GC helper threads	6-27
No title	6-28
No title	6-29
Concurrent GC (optavgpause)	6-30
Generational concurrent GC (gencon)	6-31
Generational concurrent GC: Scavenge (1 of 2)	6-32
Generational concurrent GC: Scavenge (2 of 2)	6-33
Generational concurrent GC: Self-adjusting	6-34
Balanced GC	6-35
6.3. JVM heap memory usage	6-37
JVM heap memory usage	6-38
How does Java use native memory?	6-39
Memory available to the Java process	6-40
Native heap available to application	6-42
32-bit user space available to the Java process	6-43
Theoretical and advised max heap sizes for 32-bit	6-44
Using 64-bit operating systems	6-45
Compressed references summary	6-46
IBM Java 7.0 versus IBM Java 6.0	6-47
Unit summary	6-48
Checkpoint questions	6-49
Checkpoint answers	6-50
Exercise 6	6-51
Exercise objectives	6-52

Unit 7. Tuning the JVM.....	7-1
Unit objectives	7-2
Topics	7-3
7.1. Tuning the IBM JVM	7-5
Tune the IBM JVM	7-6
What factors affect memory performance the most	7-7
Runtime performance tuning: General	7-8
JVM configuration (1 of 2)	7-9
JVM configuration (2 of 2)	7-10
Runtime performance tuning: Heap size	7-12
Choosing the right GC policy (1 of 2)	7-13
Choosing the right GC policy (2 of 2)	7-14
Memory basics	7-15
Garbage collector (GC) managed Java heap size	7-16
Fixed heap sizes versus variable heap sizes (1 of 2)	7-17
Fixed heap sizes versus variable heap sizes (2 of 2)	7-18
Heap expansion and shrinkage	7-19
How to tune a generational concurrent GC setup	7-20
How to tune a generational concurrent GC nursery generation space	7-21
Sizing the nursery	7-22
7.2. Introduction to JVM problem determination.....	7-23
Introduction to JVM problem determination	7-24
JVM problem determination: Symptom analysis	7-25
JVM problem determination: Data to collect	7-26
Javacore overview	7-27
Javacore file location and naming	7-28
Javacore file subcomponents	7-29
Javacore example (JDK v6)	7-30
Verbose garbage collection (GC)	7-31
What is a java.lang.OutOfMemoryError?	7-32
Symptoms of memory leak in the Java code	7-33
JVM tool overview	7-34
Thread and Monitor Dump Analyzer (TMDA)	7-35
Garage Collection and Memory Visualizer (GCMV)	7-36
Memory Analyzer overview	7-38
Memory Analyzer leak suspects: Default report	7-40
Health Center	7-41
Health Center client	7-43
7.3. The HotSpot garbage collector.....	7-45
The HotSpot garbage collector	7-46
The HotSpot generational collector	7-47
HotSpot generations	7-48
Tuning HotSpot	7-49
Tuning: Verbose GC HotSpot	7-50
Verbose GC HotSpot: Output	7-51
Verbose GC HotSpot: Extra output	7-52
Tuning: Commerce Server: HotSpot	7-53
Controls: HotSpot	7-54
Unit summary	7-55
Checkpoint questions	7-56
Checkpoint answers	7-57

Exercise 7	7-58
Exercise objectives	7-59
Exercise 8	7-60
Exercise objectives	7-61
Unit 8. Tuning the connection pool.....	8-1
Unit objectives	8-2
Topics	8-3
8.1. Connection pools and properties.....	8-5
Connection pools and properties	8-6
What is connection pooling?	8-7
JCA connection pooling architecture	8-8
Types of connection pools in WebSphere	8-10
The need for tuning the connection pool	8-11
Key connection pool parameters	8-12
Shareable connections	8-13
Unshareable connections	8-15
Connection pool parameters in the administrative console	8-16
Monitoring the connection pool with Tivoli Performance Viewer	8-18
Tivoli Performance Viewer connection pool monitoring	8-20
Tivoli Performance Viewer connection pool example: Waiting thread count	8-21
Generating tuning advice with Tivoli Performance Viewer Performance Advisor	8-22
Tivoli Performance Viewer Performance Advisor example	8-24
Runtime Performance Advisor tuning advice	8-25
8.2. Testing, monitoring, and tuning	8-27
Testing, monitoring, and tuning	8-28
Tuning the connection pool tasks	8-29
Tuning the connection pool	8-30
Best practices for tuning the connection pool	8-31
Tuning example: Load test	8-33
Test results for untuned connection pool	8-34
Tuning example: Tune and load test again	8-35
Test results for tuned connection pool	8-36
8.3. Prepared statement cache.....	8-37
Prepared statement cache	8-38
Prepared statement cache (1 of 2)	8-39
Prepared statement cache (2 of 2)	8-40
Use Tivoli Performance Viewer to monitor cache discards	8-41
Statement cache size setting	8-42
Monitoring DayTrader's jdbc/TradeDataSource (1 of 2)	8-43
Monitoring DayTrader's jdbc/TradeDataSource (2 of 2)	8-44
Unit summary	8-45
Checkpoint questions	8-46
Checkpoint answers	8-47
Exercise 9	8-48
Exercise objectives	8-49
Unit 9. WebSphere runtime performance tuning.....	9-1
Unit objectives	9-2
Topics	9-3
Request flow and possible bottlenecks	9-4

Tuning parameter hot list	9-5
9.1. Operating systems and the Java virtual machine (JVM)	9-7
Operating systems and the Java virtual machine (JVM)	9-8
Operating systems (1 of 2)	9-9
Operating systems (2 of 2)	9-10
64-bit WebSphere Application Server (1 of 2)	9-11
64-bit WebSphere Application Server (2 of 2)	9-12
Moving transaction logs and file store to a fast disk	9-13
Key JVM tuning parameters	9-14
9.2. Threading and concurrency	9-17
Threading and concurrency	9-18
Threading and concurrency: Web server (1 of 3)	9-19
Threading and concurrency: Web server (2 of 3)	9-21
Threading and concurrency: Web server (3 of 3)	9-22
Threading and concurrency: Application server	9-23
Threading and concurrency: Server thread pools	9-24
Message listeners and activation specifications (1 of 2)	9-26
Message listeners and activation specifications (2 of 2)	9-27
9.3. Database connectivity, EJBs, and dynamic caching	9-29
Database connectivity, EJBs, and dynamic caching	9-30
Database connectivity (1 of 2)	9-31
Database connectivity (2 of 2)	9-32
Database servers	9-33
Enterprise JavaBeans (EJBs): Cache tuning	9-35
ORB pass by reference (1 of 2)	9-37
ORB pass by reference (2 of 2)	9-38
Performance benefit of ORB pass by reference	9-39
Dynamic caching (1 of 3)	9-40
Dynamic caching (2 of 3)	9-41
Dynamic caching (3 of 3)	9-42
Performance benefit of servlet caching	9-43
9.4. WebSphere default messaging (service integration bus)	9-45
WebSphere default messaging (service integration bus)	9-46
WebSphere messaging concept view	9-47
WebSphere messaging: Reliability level	9-48
Performance comparison of message reliability levels	9-49
WebSphere messaging data buffers (1 of 3)	9-50
WebSphere messaging data buffers (2 of 3)	9-51
WebSphere messaging data buffers (3 of 3)	9-52
WebSphere messaging data stores options	9-53
Unit summary	9-54
Checkpoint questions	9-55
Checkpoint answers	9-56
Unit 10. Application profiling and tuning	10-1
Unit objectives	10-2
Application bottlenecks	10-3
What resource is limited?	10-4
What is the JVM doing?	10-5
The Java Health Center	10-6
The Java Health Center	10-7

The Java Health Center perspectives	10-8
The Java Health Center Classes perspective	10-9
The Java Health Center Classes perspective	10-10
The Java Health Center environment perspective	10-11
The Java Health Center environment perspective	10-12
The Java Health Center Garbage Collection perspective	10-13
The Java Health Center Garbage Collection perspective	10-14
The Java Health Center I/O perspective	10-16
The Java Health Center I/O perspective	10-17
The Java Health Center Locking perspective	10-18
The Java Health Center Locking perspective	10-19
The Java Health Center Locking perspective	10-20
The Java Health Center Locking perspective	10-21
The Java Health Center Threads perspective	10-22
The Java Health Center Threads perspective	10-23
The Java Health Center Native memory perspective	10-24
The Java Health Center Native memory perspective	10-25
The Java Health Center Native memory perspective	10-26
The Java Health Center profiling perspective	10-27
The Java Health Center profiling perspective	10-28
Profiling perspective Method Profile view	10-29
Profiling perspective Invocation paths view	10-30
Profiling perspective Called methods view	10-31
Profiling perspective Timeline view	10-32
The Java Health Center Method trace perspective	10-33
The Java Health Center Method trace perspective	10-35
Unit summary	10-36
Checkpoint questions	10-37
Checkpoint answers	10-38
Exercise 10	10-39
Exercise objectives	10-40
 Unit 11. WebSphere clusters and scalability	 11-1
Unit objectives	11-2
Topics	11-3
11.1. Application server clusters	11-5
Application server clusters	11-6
Application server clusters	11-7
Using cluster configurations (1 of 3)	11-8
Using cluster configurations (2 of 3)	11-9
Using cluster configurations (3 of 3)	11-10
Clustering: Vertical scaling	11-11
Clustering: Horizontal scaling	11-12
Clustering: Combined	11-13
Option to start application components dynamically	11-14
Scaling with multiple processors	11-15
Clustering: Multiple processors	11-16
Clustering best practices (1 of 2)	11-17
Clustering best practices (2 of 2)	11-18
11.2. Intelligent Management	11-19
Intelligent Management	11-20

Intelligent management	11-21
Performance Management	11-23
Health management	11-24
Intelligent routing	11-25
Dynamic clusters (1 of 2)	11-26
Dynamic clusters (2 of 2)	11-27
Autonomic managers and services	11-28
Unit summary	11-30
Checkpoint questions	11-31
Checkpoint answers	11-32
Exercise 11	11-33
Exercise objectives	11-34
Exercise environment	11-35
Unit 12. Course summary	12-1
Unit objectives	12-2
Course learning objectives	12-3
To learn more on the subject	12-4
References (1 of 2)	12-5
References (2 of 2)	12-6
Unit summary	12-7
Appendix A. List of abbreviations	A-1

Trademarks

The reader should recognize that the following terms, which appear in the content of this training document, are official trademarks of IBM or other companies:

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide.

The following are trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide:

AFS™	AIX®	CICS®
DB™	DB2®	developerWorks®
Express®	HACMP™	IMS™
Informix®	Insight™	MVS™
PartnerWorld®	RACF®	Rational®
RDN®	Redbooks®	Tivoli®
WebSphere®	WPM®	z/OS®

Intel is a trademark or registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java™ and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

VMware and the VMware "boxes" logo and design, Virtual SMP and VMotion are registered trademarks or trademarks (the "Marks") of VMware, Inc. in the United States and/or other jurisdictions.

Performance View® is a trademark or registered trademark of Kenexa, an IBM Company.

Other product and service names might be trademarks of IBM or other companies.

Course description

WebSphere Application Server V8.5.5 Performance Tuning

Duration: 4 days

Purpose

This 4-day course teaches you how to monitor and tune WebSphere Application Servers for improved performance. You learn about performance tuning methods and monitoring tools that apply to both the application server environment and Java EE applications. These methods and tools for load testing, monitoring, and tuning are applied to various WebSphere components, such as the application server's Java virtual machine (JVM), JDBC connection pools, JMS resources, and the general runtime environment.

Instructor-led lectures cover topics such as performance tuning concepts and tasks, monitoring server performance, and tools and techniques for load testing. The course provides suggestions for tuning the JVM, JDBC connection pools, and application profiling.

Hands-on lab exercises throughout the course give you practical experience by using tools to generate user loads on an application and monitoring key performance metrics such as response time and throughput. You also learn how to gather information from WebSphere built-in performance advisors and how to use this information to tune, test, and monitor Java EE applications.

Audience

This course is designed for anyone who works on WebSphere-related applications and projects, including administrators, IBM Business Partners, independent software vendors, and consultants.

Prerequisites

Before taking this course, you should have:

- WebSphere administration skills, which you can learn by successfully completing an IBM WebSphere Application Server V8.5 or V8.5.5 administration course (WA585/ZA585 or WA855/ZA855), or through practical experience with administering a WebSphere Application Server environment
- Basic operating skills for the Linux operating system

Objectives

After completing this course, you should be able to:

- Apply performance tuning methods to WebSphere Application Servers
- Perform load testing by using Apache JMeter
- Monitor application server performance by using WebSphere and the IBM Support Assistant
- Monitor and tune the JVM for optimum throughput and response time
- Monitor and tune connection pools for optimum performance
- Implement best practices for general WebSphere runtime performance
- Use the IBM Health Center tool to profile and tune Java EE applications
- Load test and monitor an application server cluster

Curriculum relationship

- WA585/ZA585, WA855/ZA855
- WA591/ZA591

Agenda

Day 1

- Course introduction
- Unit 1: WebSphere Application Server systems and components
- Unit 2: Overview of performance concepts and tuning tasks
- Exercise 1: POD configuration
- Unit 3: Apache JMeter and load testing
- Exercise 2: Apache JMeter basics
- Exercise 3: DayTrader Benchmark installation

Day 2

- Exercise 4: Using Apache JMeter to load test DayTrader
- Unit 4: WebSphere performance data and tools
- Exercise 5: Performance monitoring tools
- Unit 5: WebSphere performance tuning methods
- Unit 6: Introduction to the JVM
- Exercise 6: Exploring GC policies and monitoring JVM performance

Day 3

- Unit 7: Tuning the JVM
- Exercise 7: Tuning the JVM
- Exercise 8: Troubleshooting JVM problems
- Unit 8: Tuning the connection pool
- Exercise 9: Tuning JDBC connection pools and enabling servlet caching
- Unit 9: WebSphere runtime performance tuning

Day 4

- Unit 10: Application profiling and tuning
- Exercise 10: Application profiling with Java Health Center
- Unit 11: WebSphere clusters and scalability
- Exercise 11: Load testing an application server cluster
- Unit 12: Course summary

Unit 1. WebSphere Application Server systems and components

What this unit is about

This unit provides an overview of the topology of a system. It identifies and describes key components and troubleshooting points.

What you should be able to do

After completing this unit, you should be able to:

- Describe stand-alone server architecture
- Describe Network Deployment (ND) cell architecture
- List and describe the function of IBM products that are involved in implementing stand-alone and distributed architectures
- Identify the components of the application server and describe the services that they provide
- Identify the components of an ND cell and describe the function of each
- Describe features of the Liberty profile
- Describe functions of Intelligent Management
- Describe the flow of an application request
- Describe the flow of administration requests
- Identify common bottlenecks in the end-to-end flow of client requests

How you will check your progress

- Checkpoint questions

References

SG24-7971-00 WebSphere Application Server V8: Administration and Configuration Guide

Course WA380: IBM WebSphere Application Server V8 Administration on Windows

Course WA580: IBM WebSphere Application Server V8 Administration on Linux

Unit objectives

After completing this unit, you should be able to:

- Describe stand-alone server architecture
- Describe Network Deployment (ND) cell architecture
- List and describe the function of IBM products that are involved in implementing stand-alone and distributed architectures
- Identify the components of the application server and describe the services that they provide
- Identify the components of an ND cell and describe the function of each
- Describe features of the Liberty profile
- Describe functions of Intelligent Management
- Describe the flow of an application request
- Describe the flow of administration requests
- Identify common bottlenecks in the end-to-end flow of client requests

© Copyright IBM Corporation 2016

Figure 1-1. Unit objectives

WA8152.2

Notes:



Topics

- Components and services
- Network Deployment components
- Client request flow

© Copyright IBM Corporation 2016

Figure 1-2. Topics

WA8152.2

Notes:

1.1. Components and services

Components and services



© Copyright IBM Corporation 2016

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

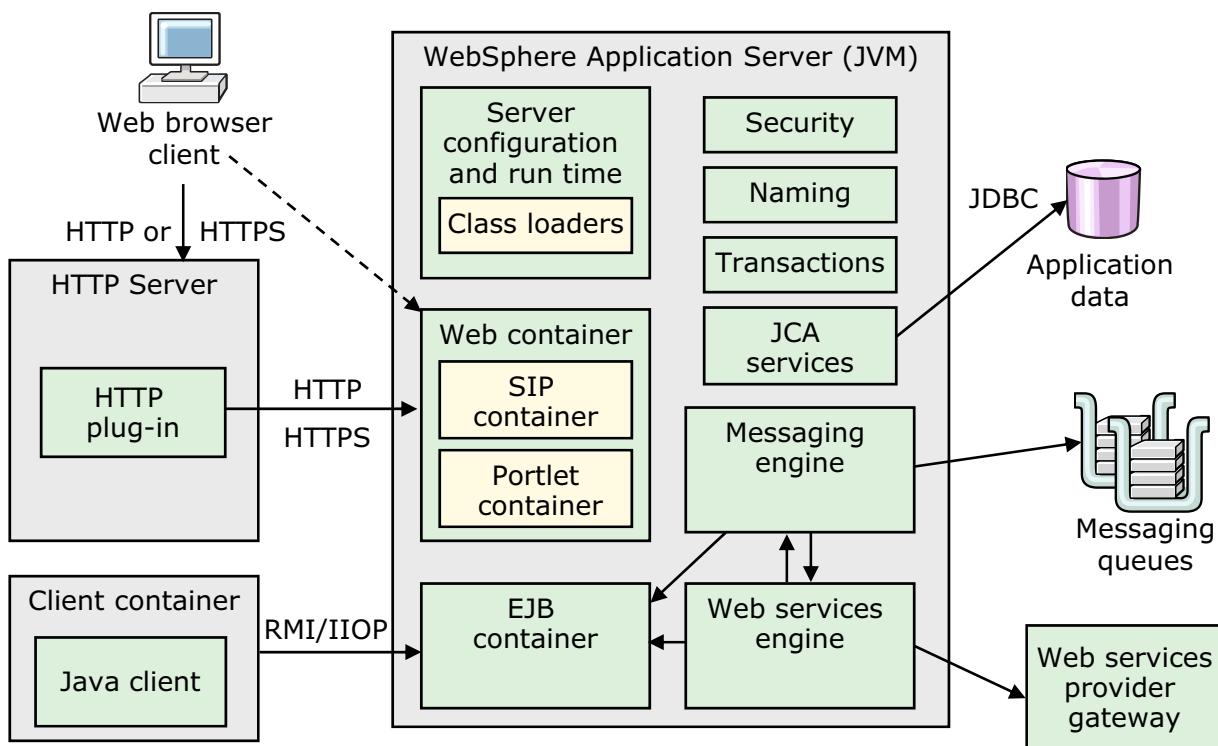
9.1

Figure 1-3. Components and services

WA8152.2

Notes:

WebSphere Application Server architecture



© Copyright IBM Corporation 2016

Figure 1-4. WebSphere Application Server architecture

WA8152.2

Notes:

This diagram illustrates the basic architecture of WebSphere Application Server, including several of the larger components.

The main element is the application server, a Java process that encapsulates many services, including the containers, where business logic runs. If you are familiar with Java EE, you recognize the web container and the EJB container. The web container runs servlets and JavaServer Pages (JSPs), both of which are Java classes that generate web markup language. Traffic into and out of the web container travels through the embedded HTTP server. While servlets and JSPs can act independently, they most commonly make calls to Enterprise JavaBeans (EJBs) to run business logic or access data. EJBs, which run in the EJB container, are easily reusable Java classes. They most commonly communicate with a relational database or other external source of application data. These EJBs return data to the web container or update the data on behalf of the servlet or JSP.

The JMS messaging engine is built into the application server. This engine is a pure Java messaging engine. JMS destinations, which are known as queues and topics, provide asynchronous messaging services to the code that is running inside the containers. JMS is covered in more depth later in this course.

As you see in more detail later on, the web services engine enables application components to be exposed as web services, which can be accessed with SOAP.

Several other services are run within the application server, including the dynamic cache, data replication, security, and others. These services are covered later in the course.

There are also some important components outside of the application server process.

WebSphere Application Server also provides a plug-in for HTTP servers that determines what HTTP traffic is intended for WebSphere to handle, and routes the requests to the appropriate server. The plug-in is critical to workload management of HTTP requests, as it can distribute the load to multiple application servers, and steer traffic away from unavailable servers. It too reads its configuration from a special XML file.

1.

Web container

- The web container processes:
 - Servlets
 - JSP files
 - Other types of server-side includes (SSI)
- Each application server run time has one logical web container, which can be modified, but not created or removed
- Each web container provides:
 - Web container transport chains
 - Servlet processing
 - JSP processing
 - HTML and other static content processing
 - Session management
 - Thread pool
 - Portlet container

© Copyright IBM Corporation 2016

Figure 1-5. Web container

WA8152.2

Notes:

- **Web container transport chains**

The web container inbound transport chain is used to direct requests to the web container. The chain consists of a TCP inbound channel that provides the connection to the network, and an HTTP inbound channel that serves HTTP 1.0 and 1.1 requests. The chain continues with a web container channel over which requests for servlets and JSPs are sent to the web container for processing.

- **Servlet processing**

When handling servlets, the web container creates a request object and a response object; it then invokes the servlet service method. The web container invokes the destroy method of the servlet when appropriate and unloads the servlet after which the JVM performs garbage collection.

- **HTML and other static content processing**

The web container inbound chain serves requests for HTML and other static content that are directed to the web container. However, in most cases, the use of an external web server and

web server plug-in as a front end to a web container is more appropriate for a production environment.

- **Session management**

Support is provided for the `javax.servlet.http.HttpSession` interface as described in the servlet application programming interface (API) specification.

- **Web services engine**

Web services are provided as a set of APIs in cooperation with the Java EE applications. Web services engines are provided to support SOAP.

The following properties allow you to configure the services that the web container provides:

- **Default virtual host**

This virtual host is the default to use for applications on the server.

- **Enable servlet caching**

You can use dynamic cache to improve application performance by caching the output of servlets, commands, and JSPs. This setting allows you to enable dynamic caching for servlets. You must first enable dynamic caching and create the appropriate cache policies so that you can use servlet caching.

- **Session management**

You can determine how the web container manages HTTP session data. This determination includes settings for the session tracking mechanism (for example, cookies), session timeout, and for the session persistence method.

- **Web container transport chains**

You can add to or configure the communication channels that are used for accessing applications in the web container. By default, you have four transport chains predefined. These transport chains are for secure and nonsecure administration console access, and for default access to the web container. The transport chains are related to port definitions seen in the communications section. Port numbers must be unique for each application server instance on a machine.

- **Custom properties**

You can specify name-value pairs for configuring internal system properties. Some components use custom configuration properties, which can be defined here. It is not common to pass information to the web container this way, but the Java EE specification indicates it as a requirement. Most configuration information can be handled programmatically, or through the deployment descriptor.

Enterprise JavaBeans (EJB) container

- The Enterprise JavaBeans (EJB) container provides the runtime services that are needed to deploy and manage enterprise beans
- A server process that handles requests for:
 - Session beans
 - Entity beans
 - Message-driven beans (MDBs)
- The enterprise beans that are packaged in EJB modules and installed in an application server, do not communicate directly with the server
 - The EJB container provides an interface between the enterprise beans and the server
- Together, the EJB container and the application server provide the enterprise bean runtime environment
- The WebSphere run time provides many low-level services, including:
 - Thread pool support
 - Transaction support
- The EJB container manages data storage and retrieval for the contained EJBs
- A single EJB container can host more than one EJB application module

© Copyright IBM Corporation 2016

Figure 1-6. Enterprise JavaBeans (EJB) container

WA8152.2

Notes:

The following properties allow you to configure the services that the EJB container provides:

- **Passivation directory**

This attribute provides the directory that you can use to store the persistent state of passivated, stateful session EJBs. If you are using the EJB container to manage session data, you should give WebSphere the ability to swap data to disk when necessary. This directory tells WebSphere where to hold EJB session data when it passivates and activates beans from the pool.

- **Inactive pool cleanup interval**

Because WebSphere builds a pool of EJBs to satisfy incoming requests, you must tell it when to remove beans from this pool to preserve resources. This attribute allows you to define the interval at which the container examines the pools of available bean instances to determine whether some instances can be deleted to reduce memory usage.

- **Default data source JNDI name**

Here you can set a default data source to use for EJBs that have no individual data source defined. This setting is not applicable for EJB-compliant CMP beans.

- **Initial state**

This attribute allows you to identify the state of the container when WebSphere is started. If you must recycle the application server, this attribute is used to determine whether to start the EJB container at server startup. You would set it to stopped only if you planned on never using the EJB container or EJBs within that specific application server instance.

- **EJB cache settings**

You can set up two types of cache settings in WebSphere:

- **Cleanup interval:** This attribute allows you to set the interval at which the container attempts to remove unused items from the cache. It reduces the total number of items in cache to the value you set in the cache size attribute.
- **Cache size:** This attribute specifies the number of buckets in the active instance list within the EJB container. WebSphere uses this attribute to determine how large the cache is and when to remove components from the cache to reduce its size.

Object Request Broker service

- Using Internet Inter-ORB Protocol (IIOP), an Object Request Broker (ORB) manages the interaction between EJB clients and EJBs
 - ORB service enables clients to make requests and receive responses from EJBs in a network-distributed environment
 - Provides a framework for clients to locate objects in the network and call operations on those objects
- The client-side ORB:
 - Creates an IIOP request that contains the operation and any required parameters for sending the request across the network
- The server-side ORB:
 - Receives the IIOP request, locates the target object, starts the requested operation, and returns the results to the client
- Commonly tuned ORB properties
 - Thread pool size
 - Pass by reference: Use when client and the EJB are on the same classloader

© Copyright IBM Corporation 2016

Figure 1-7. Object Request Broker service

WA8152.2

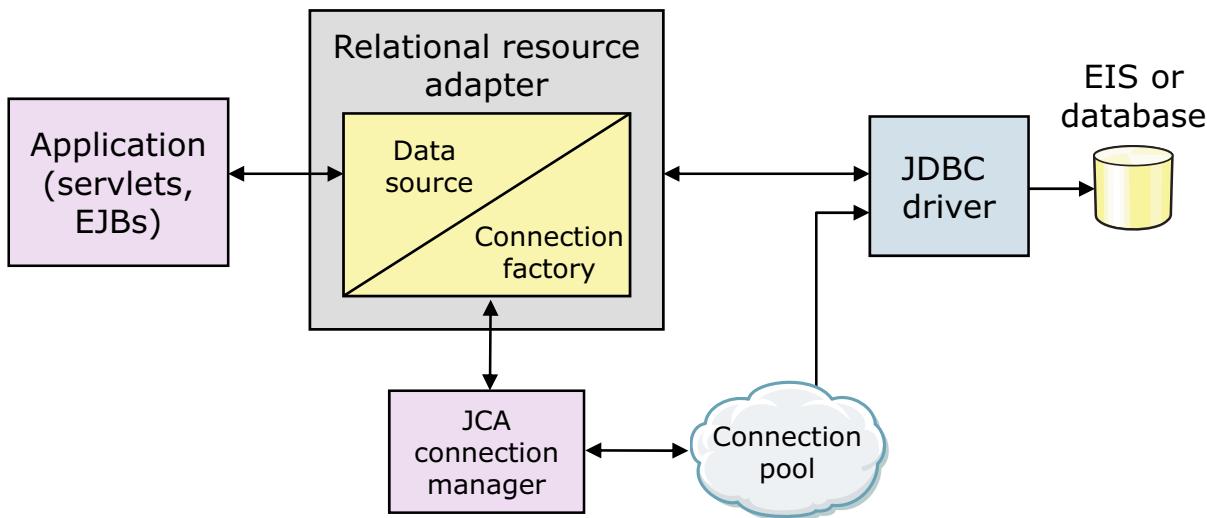
Notes:

An Object Request Broker (ORB) uses Internet Inter-ORB Protocol (IIOP) to manage the interaction between clients and servers. The ORB service enables clients to make requests and receive responses from servers in a network-distributed environment. The ORB service provides a framework for clients to locate objects in the network and call operations on those objects as though the remote objects were in the same running process as the client. The ORB service provides location transparency. The client calls an operation on a local object, which is known as a *stub*. Then, the stub forwards the request to the intended remote object, where the operation is run, and the results are returned to the client. The client-side ORB is responsible for creating an IIOP request that contains the operation and any required parameters, and for sending the request in the network. The server-side ORB receives the IIOP request, locates the target object, invokes the requested operation, and returns the results to the client. The client-side ORB unmarshals the returned results and passes the result to the stub, which returns the result to the client application, as though the operation were run locally.

When using the SCA programming model, IIOP communication is no longer the only way to invoke EJBs. When they are wrapped in SCA components, they can be invoked in various ways, including web services and JMS messaging.

Java EE Connector Architecture service (JCA)

- The JCA Connection Manager administers:
 - Connections that are obtained through resource adapters that the JCA defines
 - Data sources that the JDBC 2.0 Extensions define



© Copyright IBM Corporation 2016

Figure 1-8. Java EE Connector Architecture service (JCA)

WA8152.2

Notes:

Connection management for access to enterprise information systems (EIS) in WebSphere Application Server is based on the Java EE Connector Architecture (JCA) specification, also sometimes referred to as J2C. The connection between the enterprise application and the EIS is made by using EIS-provided resource adapters, which are plugged into the application server. The architecture specifies the connection management, transaction management, and security contracts that exist between the application server and the EIS.

Within the application server, the connection manager pools and manages connections. The connection manager administers connections that are obtained through both resource adapters that are defined according to the JCA specification and sources that are defined according to the JDBC 2.0 extensions, and later, specification.

The JCA connection manager provides the connection pooling, local transaction, and security supports. The relational resource adapter provides the JDBC wrappers and JCA CCI implementation that allow applications that use bean-managed persistence, JDBC calls, and container-managed persistence beans to access the database JDBC driver.

The JCA resource adapter is a system-level software driver that EIS vendors or other third-party vendors supply. It provides the connectivity between Java EE components (an application server or an application client) and an EIS.

One resource adapter, the WebSphere Relational Resource Adapter, is predefined for handling data access to relational databases. This resource adapter provides data access through JDBC calls to access databases dynamically. It provides connection pooling, local transaction, and security support. The WebSphere persistence manager uses this adapter to access data for container-managed persistence beans.



WebSphere default messaging

- Integrated asynchronous capabilities for WebSphere
 - Integral JMS messaging service for WebSphere Application Server
 - Fully compliant JMS 1.1 provider
 - JMS provider is the default messaging provider
- Based on service integration bus (SIBus) technology
 - Intelligent infrastructure for service-oriented integration
 - Unifies SOA, messaging, message brokering, and publish/subscribe
- Complements and extends WebSphere MQ and application server
- Other WebSphere family products use default messaging

© Copyright IBM Corporation 2016

Figure 1-9. WebSphere default messaging

WA8152.2

Notes:

The service integration functionality within WebSphere Application Server provides a highly flexible messaging fabric that supports a service-oriented architecture with a wide spectrum of quality of service options, supported protocols, and messaging patterns. It supports both message-oriented and service-oriented applications.

SOA is an architectural style whose goal is to achieve loose coupling among interacting software agents. A service is a unit of work that a service provider does to achieve specified results for a service consumer. Both provider and consumer are roles that software agents play on behalf of their owners.

Security service

- Each application server hosts a security service
- The security service uses the security settings that are stored in the configuration repository to provide:
 - Authentication and authorization
 - SSL encryption
 - Java 2 security
- User registries that contain authentication data can be configured with one of the following repositories:
 - File-based (default)
 - Local OS
 - LDAP server
 - Custom user registry
 - Federated repository - combination of file-based, LDAP servers, CUR, or databases
- Support for multiple security domains

© Copyright IBM Corporation 2016

Figure 1-10. Security service

WA8152.2

Notes:

Each application server JVM hosts a security service. The security service uses the security settings that are held in the configuration repository to provide authentication and authorization functionality.

User registries that contain authentication data can be configured by using the local OS, an LDAP directory server, a custom user registry, or a federated repository.

Federated repositories can comprise multiple user registries in LDAP servers, flat files, and databases.

The security service can also be used to enable SSL encryption between WebSphere clients and servers.

1.2. Network Deployment components

Network Deployment components



© Copyright IBM Corporation 2016

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

9.1

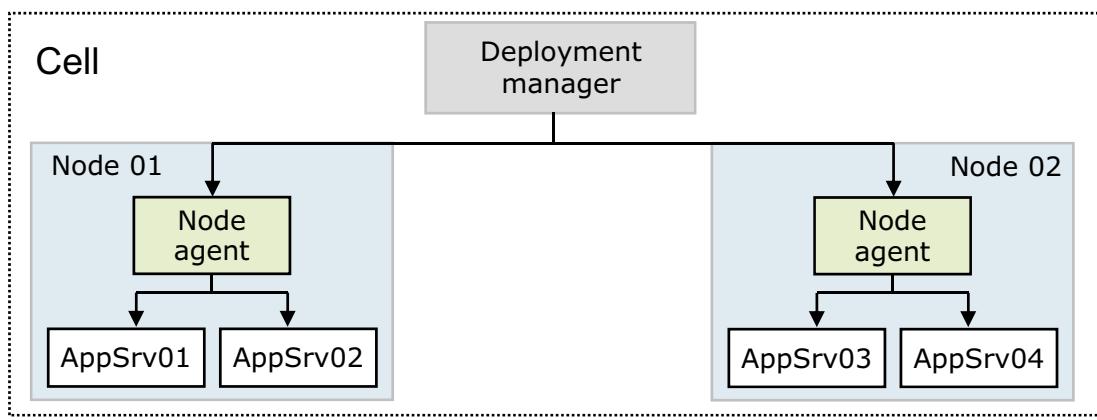
Figure 1-11. Network Deployment components

WA8152.2

Notes:

Network Deployment concepts

- A **deployment manager** process manages the node agents
 - Holds the configuration repository for the entire management domain, called a **cell**
 - In a cell, the administrative console runs inside the deployment manager
- A **node** is a logical grouping of application servers
 - A single **node agent** process manages each node
 - Through profile configuration, multiple nodes can exist on a single computer



© Copyright IBM Corporation 2016

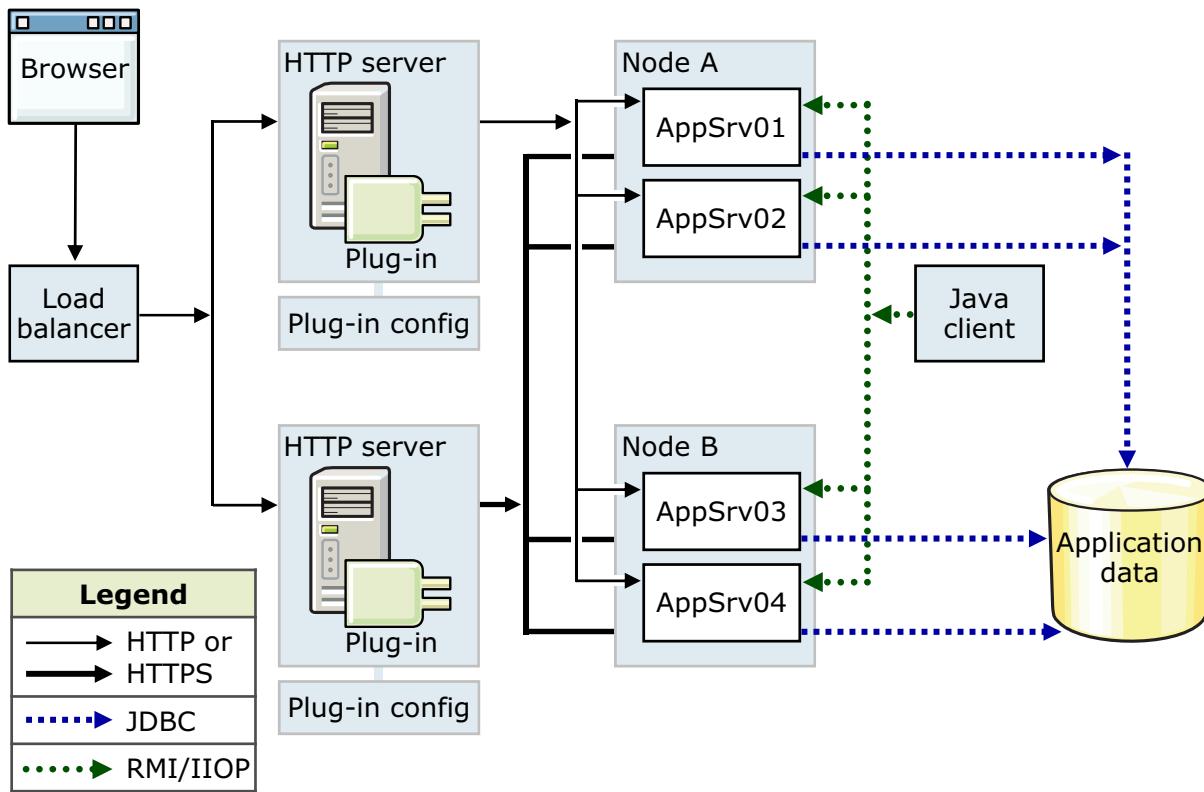
Figure 1-12. Network Deployment concepts

WA8152.2

Notes:

The deployment manager here is an application server that manages the administrative environment within a cell. A profile represents a node. Multiple nodes can exist on a single machine by using profiles. The node agent is an important process that allows for communication of administrative information (commands and configuration files) to reach the application servers.

Runtime flow



© Copyright IBM Corporation 2016

Figure 1-13. Runtime flow

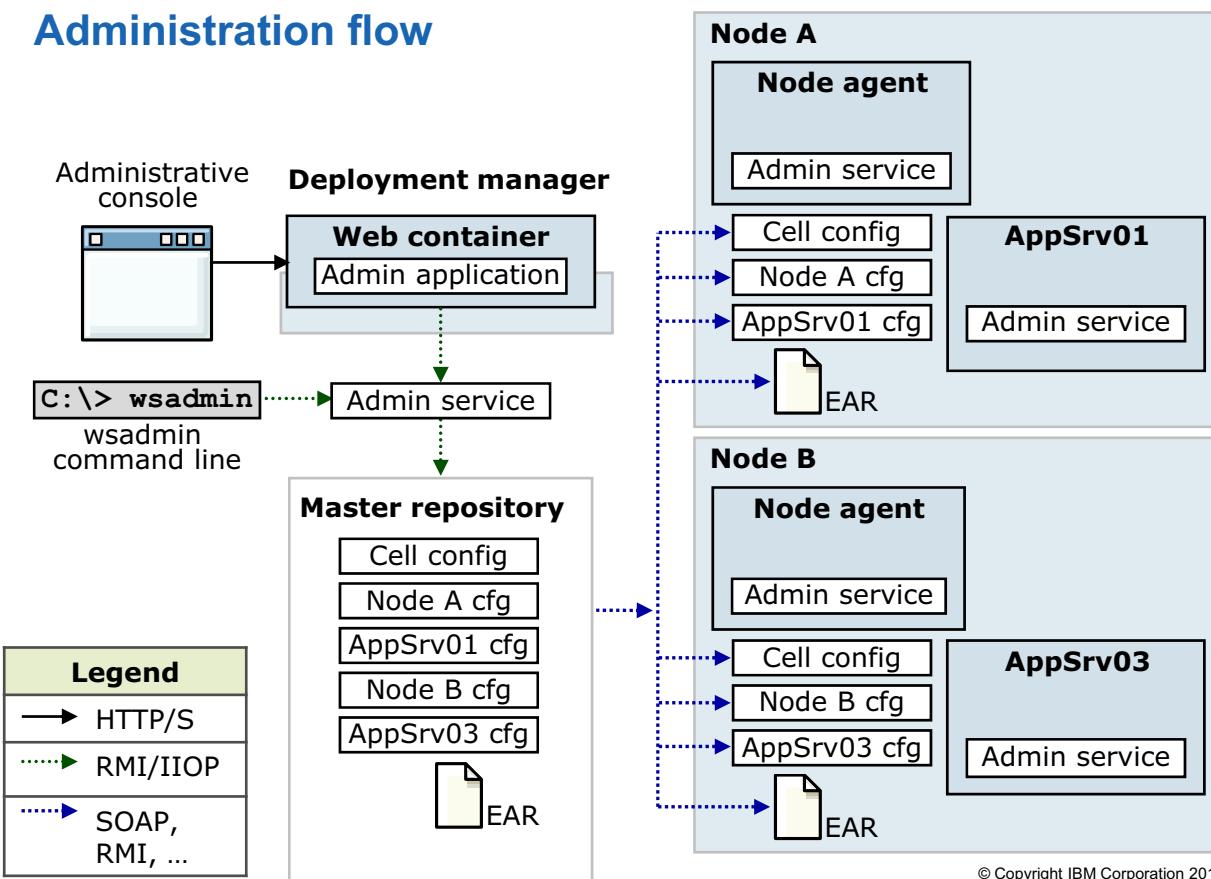
WA8152.2

Notes:

The main theme with Network Deployment is distributed applications. While the “flow” of an application remains the same, there are significant additions to the runtime of an application. Note the “load balancer”: it allows for multiple HTTP servers. Users point their browsers to the load balancer, and their requests are workload managed to an HTTP server. As soon as a request reaches one of these HTTP servers, the HTTP server plug-in load balances the request between the application servers that it is configured to serve. When the request enters the application server, the flow is identical to how it was in Express and Base.

The Java client’s requests to EJBs can also be workload managed so that the requests do not all go to one application server.

Administration flow



© Copyright IBM Corporation 2016

Figure 1-14. Administration flow

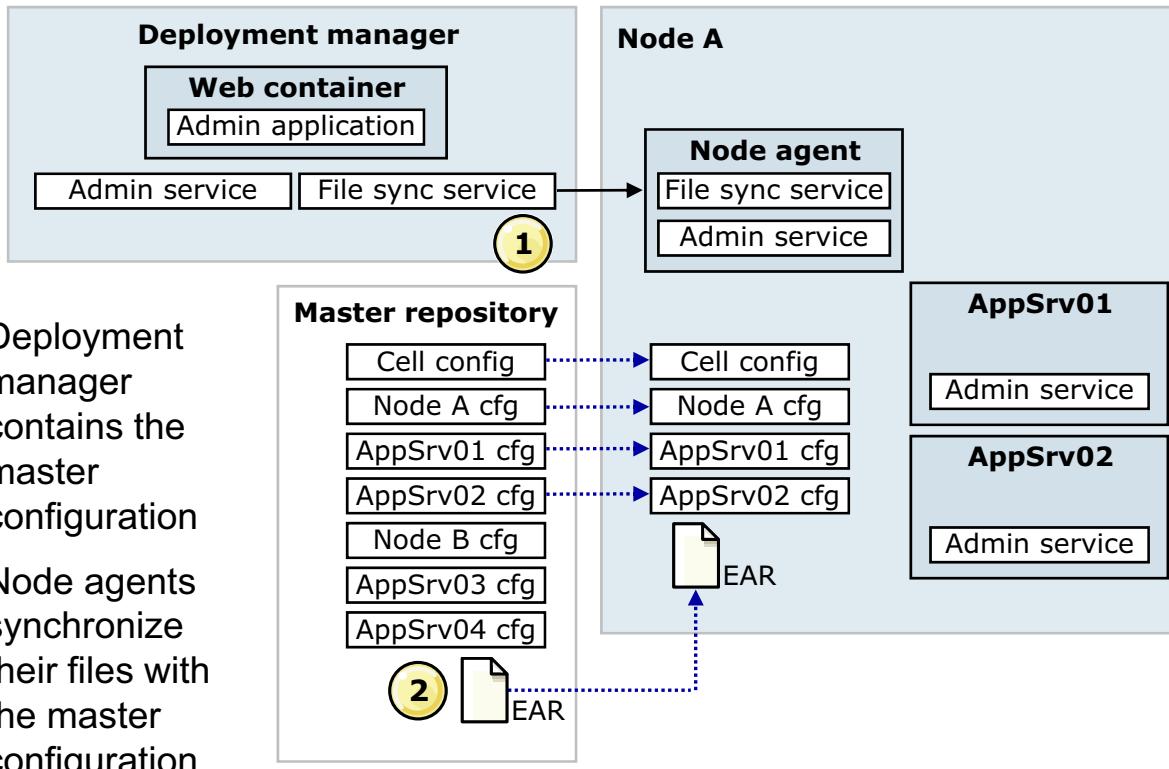
WA8152.2

Notes:

The administrative console and wsadmin are still the two ways that the environment is administered. However, take note that these tools now communicate to the deployment manager and not to the application servers directly. The communication of these commands flows from the tools to the deployment manager to the node agents, to the application servers. This communication flow allows administration of multiple nodes (each possibly containing multiple application servers) from a single focal point (the deployment manager).

There is one main repository for the configuration files within a cell, which are associated with the deployment manager. All updates to the configuration files should go through the deployment manager. You are going to see in a moment how this process works. Be careful in connecting to an application server directly with wsadmin or the administrative console, as any changes that are made to the configuration files are only temporary. The configuration files overwrite them from the master files.

File synchronization and file transfer



© Copyright IBM Corporation 2016

Figure 1-15. File synchronization and file transfer

WA8152.2

Notes:

File synchronization service

Node agents synchronize their files with the “master” configuration (repository) as follows:

- Automatically at startup
- Periodically through configuration
- Manually from the administrative console or command line

During synchronization:

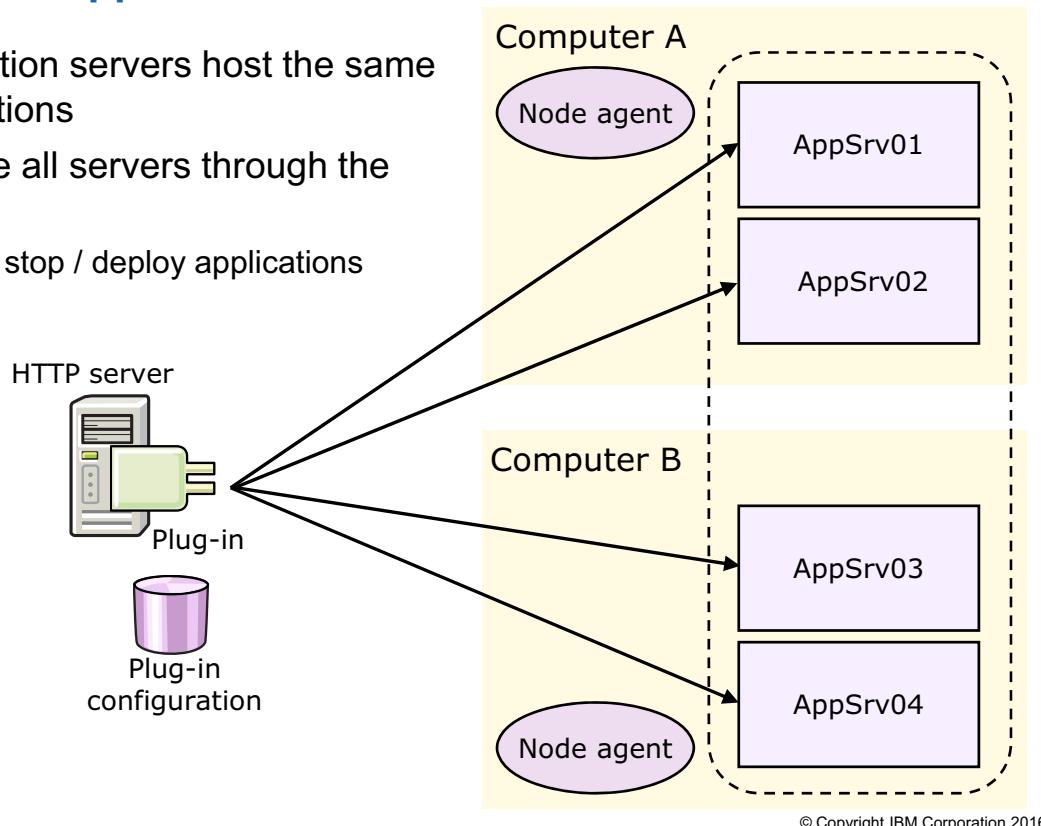
- The node agent checks for changes to master configuration
- New or updated files are copied to the node

This administrative service is responsible for keeping up-to-date the configuration and application data files that are distributed across the cell. The service runs in the deployment manager and node agents, and ensures that changes made to the master repository are propagated out to the nodes, as necessary. The file transfer system application is used for the synchronization process. File synchronization can be forced from an administration client, or can be scheduled to happen automatically. During the synchronization operation, the node agent checks with the deployment

manager to see whether any files that apply to the node are updated in the master repository. New or updated files are sent to the node, while any deleted files are also deleted from the node. Synchronization is one way. The changes are sent from the deployment manager to the node agent. No changes are sent from the node agent back to the deployment manager.

Clustered application servers

- Application servers host the same applications
- Manage all servers through the cluster
 - Start / stop / deploy applications



© Copyright IBM Corporation 2016

Figure 1-16. Clustered application servers

WA8152.2

Notes:

A cluster of application servers provides scalability, throughput, and failover. The plug-in routes web requests for the HTTP server to servers in the cluster by using a weighted or random round-robin algorithm. The plug-in configuration file contains information about clusters, members of the cluster (clones), and the applications that the servers in the cluster host. In addition, each cluster member has a weight that is stored in the configuration file. The plug-in uses this weight in its routing algorithms to determine what percentage of incoming requests are routed to each cluster member.



Web servers

- Web servers can be defined to the administrative service as web server nodes, allowing Java EE applications to be associated with one or more defined web servers
- Each web server must have a web server plug-in that is installed, which forwards requests to the application servers
- Web server nodes can be managed or unmanaged

© Copyright IBM Corporation 2016

Figure 1-17. Web servers

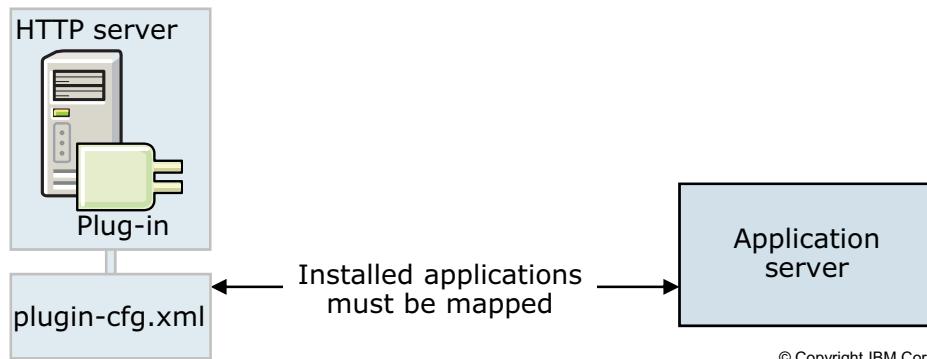
WA8152.2

Notes:

As a special case, if the unmanaged web server is an IBM HTTP Server, you can administer the web server from the administrative console. Then, you can automatically push the plug-in configuration file to the web server with the deployment manager by using HTTP commands to the IBM HTTP Server administration process. This configuration does not require a node agent.

Web server plug-ins

- When a web request requires dynamic content, such as JSP or servlet processing, it must be forwarded to the application server
- The plug-in uses the configuration file to determine whether a request is routed to the web server or an application server
- Forwards the request to the appropriate web container
 - Forwards a request that is based on its URI
 - The plug-in can use HTTP or HTTPS to transmit the request
- Can load balance among cluster members using the `plugin-cfg.xml` file



© Copyright IBM Corporation 2016

Figure 1-18. Web server plug-ins

WA8152.2

Notes:

To forward a request, you use a web server plug-in that is included with the WebSphere Application Server packages for installation on a web server. You copy an Extensible Markup Language (XML) configuration file, which is on the WebSphere Application Server, to the web server plug-in directory. The plug-in uses the configuration file to determine whether the web server or an application server should handle the request. When a web server receives a request for an application server, it forwards the request to the appropriate web container in the application server. The plug-in can use HTTP or HTTPS to transmit the request.

Web servers: Managed nodes and unmanaged nodes

- Managed nodes have a node agent on the web server computer that allows the deployment manager to administer the web server
- Administrator can:
 - Start or stop the web server from the deployment manager
 - Generate the web server plug-in configuration file for the node
 - Automatically push the plug-in configuration file to the web server
- WebSphere Application Server does not manage unmanaged web server nodes
 - Normally found outside the firewall, or in the DMZ
 - You must either manually copy, or FTP, web server plug-in configuration files to the web server
 - If you define the web server as a node, you can generate custom plug-in configuration files for it

© Copyright IBM Corporation 2016

Figure 1-19. Web servers: Managed nodes and unmanaged nodes

WA8152.2

Notes:

If the web server is an IBM HTTP Server and the IBM HTTP Server administration server is installed and properly configured, you can also:

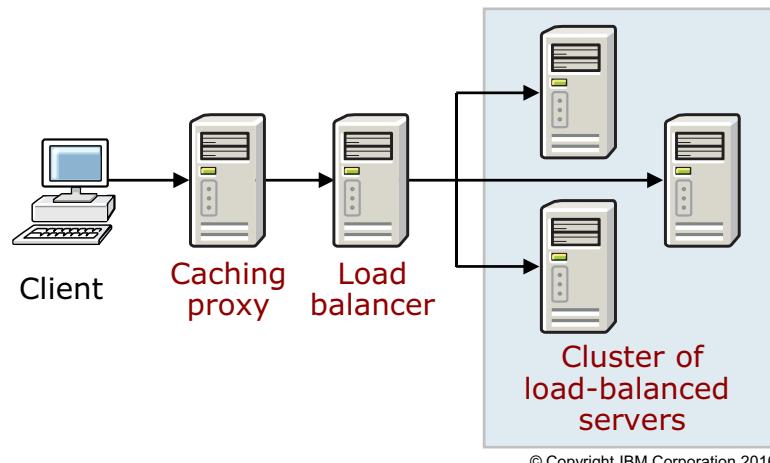
- Display the IBM HTTP Server error log (`error.log`) and access log (`access.log`) files
- Display and edit the IBM HTTP Server configuration file (`httpd.conf`)

Managed web server nodes are usually behind the firewall with WebSphere Application Server installations.

Edge Components

- WebSphere Application Server Network Deployment package contains the following Edge Components functions:
 - **Load balancer:** Responsible for balancing the load across multiple servers that can be within either local area networks or wide area networks
 - **Caching proxy** is to reduce network congestion within an enterprise by offloading security and content delivery from web servers and application servers

- Edge Components are installed separately from WebSphere Application Server



© Copyright IBM Corporation 2016

Figure 1-20. Edge Components

WA8152.2

Notes:

The caching proxy intercepts data requests from a client, retrieves the requested information from the application servers, and delivers that content back to the client. It stores cacheable content in a local cache before delivering it to the client. Subsequent requests for the same content are served from the local cache, which is much faster and reduces the network and application server load.

The load balancer provides horizontal scalability by dispatching HTTP requests among several identically configured web server or application server nodes.



Liberty profile

- The Liberty profile provides a lightweight server with a small memory footprint
- Using WebSphere Application Server Developer Tools for Eclipse, you can create a server configuration quickly
- The simple and flexible configuration is stored in the `server.xml` file and contains the configuration for the runtime instance
- You can edit the `server.xml` file directly by using an XML editor or an Eclipse-based editor
 - The configuration lists the features (capabilities or bundles) that are installed in the server
 - By defining just the features that you need, the Liberty profile provides the smallest runtime footprint for applications
- This dynamic run time allows the adding of features and updating of configuration parameters without requiring you to restart the server

© Copyright IBM Corporation 2016

Figure 1-21. Liberty profile

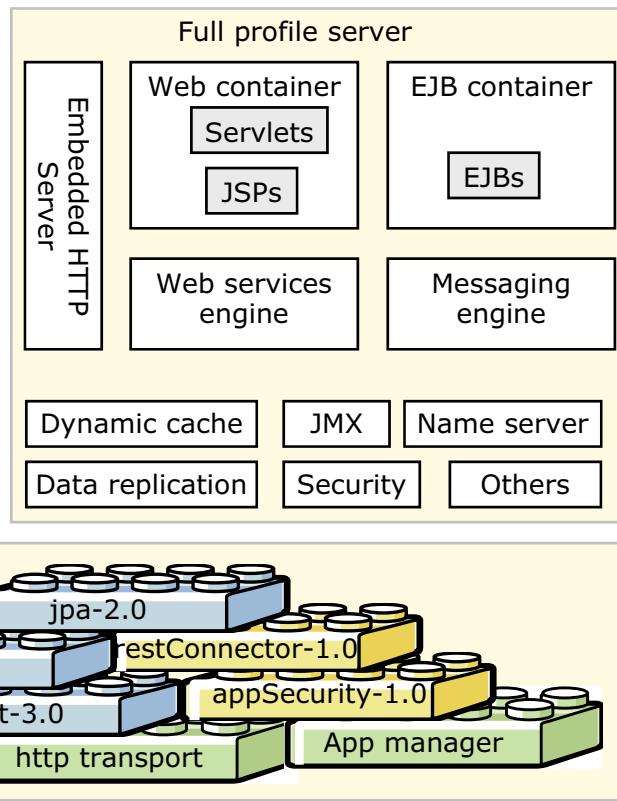
WA8152.2

Notes:

The Liberty profile provides a lightweight server with a small memory footprint. Using WebSphere Application Server Developer Tools for Eclipse, you can create a server configuration quickly with just a couple of mouse clicks. The simple and flexible configuration is stored in the `server.xml` file and contains the configuration for the runtime instance. You can edit the `server.xml` file directly by using an XML editor or an Eclipse-based editor. The configuration lists the features (capabilities or bundles) that are installed in the server. By defining just the features that you need, the Liberty profile provides the smallest runtime footprint for applications. This dynamic runtime allows the adding of features and updating of configuration parameters without requiring you to restart the server.

Liberty profile: Composable runtime

- Full profile includes everything
- Liberty profile includes only those features you add
 - Improved performance
 - Faster server starts
- **Note:** Performance tuning for Liberty is not covered in this course



© Copyright IBM Corporation 2016

Figure 1-22. Liberty profile: Composable runtime

WA8152.2

Notes:

Application servers in the full profile have a JVM shown on the left that includes all services (features) whether the applications require them or not.

Liberty profile server JVMs start only those features that you add to its `server.xml` file. By default, a server contains only the `jsp-2.2` feature to support servlet and JSP applications. You use the feature manager to add the features that you need.

The building blocks shown in the graphic on this slide show some of the Liberty profile features that can be defined for the feature manager. These features include JPA, JSP, servlet, application security, and a remote JMX connector. However, several other features can be configured for a particular server. In addition to the features, you can add HTTP port definitions for the HTTP transport, and application definitions for the application manager.

Intelligent Management (1 of 2)

- Previously the WebSphere Virtual Enterprise product
 - Basically all of the same function as WebSphere Virtual Enterprise
 - No additional license fee required
- Application infrastructure virtualization
 - Run applications on any application server in a common resource pool
 - Adjust resources quickly and seamlessly during peak periods
 - Adjust resources quickly and seamlessly in response to the peaks in unforeseen demand
 - Achieve application response times that meet business policy goals
- Extended manageability
 - Allow seamless application upgrades without experiencing an outage
 - Detect problem scenarios and act before an outage occurs
- Supports the tenets of autonomic computing: self-managing, self-healing, self-configuring, and self-optimizing

© Copyright IBM Corporation 2016

Figure 1-23. Intelligent Management (1 of 2)

WA8152.2

Notes:

There are two major functions of Intelligent Management.

The first is the ability to virtualize your application infrastructure. You might already be using hardware virtualization. Intelligent Management provides another layer of virtualization: virtualizing middleware. Hardware virtualization allows you to share hardware resources, and make efficient use of the resources by using them where they are most needed. Middleware virtualization does the same for your middleware applications. It allows you to share middleware server resources, and make the best use of those resources to serve client requests. Resources can be allocated dynamically as needed to meet unexpected or peak demand.

The second major function of Intelligent Management is extended manageability. As you move to a virtualized environment, you want to make it easier to manage that environment. Although there are many features, there are two key features to consider. The first is the ability to install multiple editions of the same application simultaneously, and easily roll out the new edition or roll back to the previous edition. The second is the ability to watch for health problems with servers, and take steps to fix the server before clients are affected.

Intelligent Management (2 of 2)

- Middleware virtualization
 - Middleware servers are started and stopped dynamically to meet business goals
 - Entire nodes can be brought online and offline to meet business goals – Single cell and multi-cell performance management
- Request flow management
 - Prioritizing and routing traffic that is based on business policies
- Enhanced management capabilities
 - Middleware servers are monitored for health problems (health management)
 - Multiple editions of the same application can be managed
 - Real-time and historical details of the performance of the environment can be captured and viewed
 - Servers can be intelligently quiesced
 - Centralized log management for problem determination
- Health management and health policies are covered in a later unit

© Copyright IBM Corporation 2016

Figure 1-24. Intelligent Management (2 of 2)

WA8152.2

Notes:

With middleware virtualization, you use a pool of middleware servers that can be dynamically started and stopped in response to client request flows. You can start more servers as more requests for applications on those servers arrive, or, if requests are not meeting response time goals, you can start more server instances to add more resources for the requests. There is also the ability to add and remove entire nodes in the cell. This ability can bring more resources online, or completely remove servers from the cell if they are no longer needed. Intelligent Management can manage resources within a single cell, and can route requests between multiple cells. It is important to understand that Intelligent Management does not dynamically start and stop applications; it dynamically starts and stops application servers.

Intelligent Management also uses Request Flow Management to optimize the use of resources in your environment. Higher priority requests can be given more resources, and application servers can be protected from overload.

Intelligent Management also provides enhanced management of the middleware environment. Health monitoring can detect problems with servers and take corrective action before clients become aware that there is a problem. Multiple editions of the same application are managed intelligently. The statistics that Intelligent Management uses for decision making and the statistics

about the performance of your environment are logged, and the information can be analyzed to help with setting service policies and with capacity planning. Servers can be put into maintenance mode, so that client traffic can be shifted to other servers, before taking a server offline. There is also the ability to gather logs in a central location, and to dynamically enable trace strings as based on the arrival of particular requests.

1.3. Client request flow

Client request flow



© Copyright IBM Corporation 2016

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

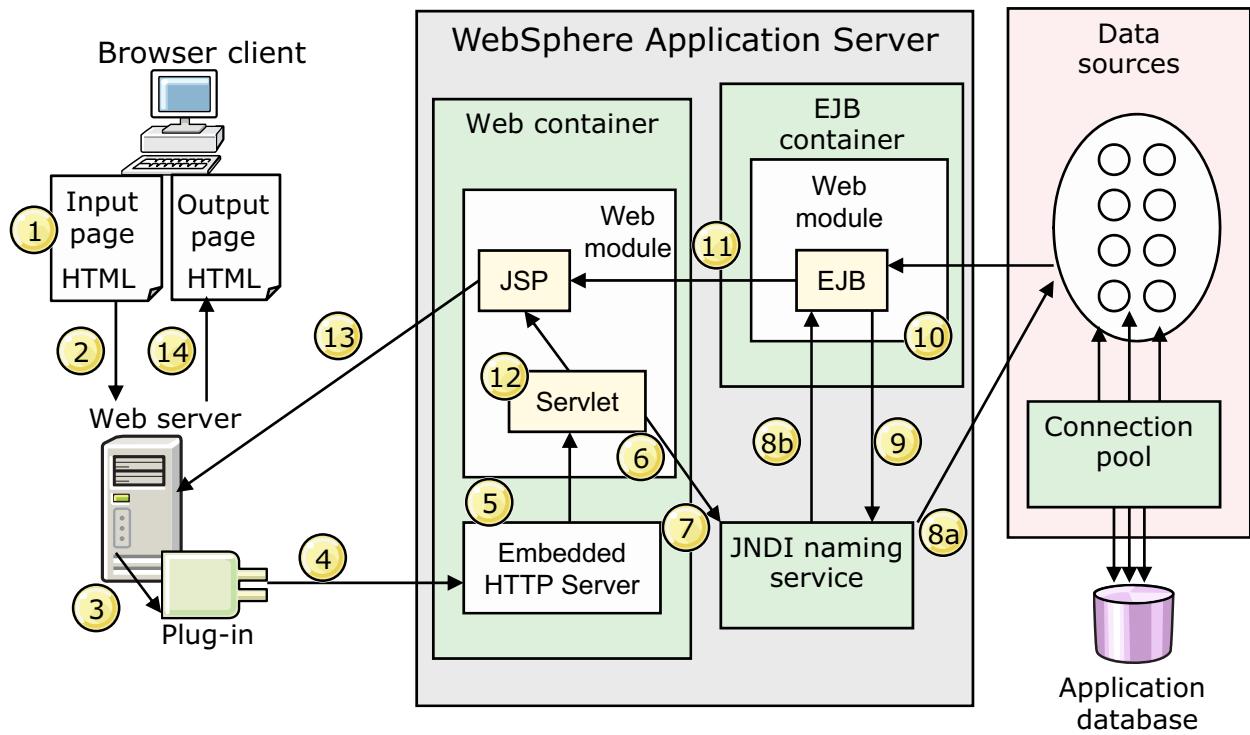
9.1

Figure 1-25. Client request flow

WA8152.2

Notes:

Typical client request application flow



© Copyright IBM Corporation 2016

Figure 1-26. Typical client request application flow

WA8152.2

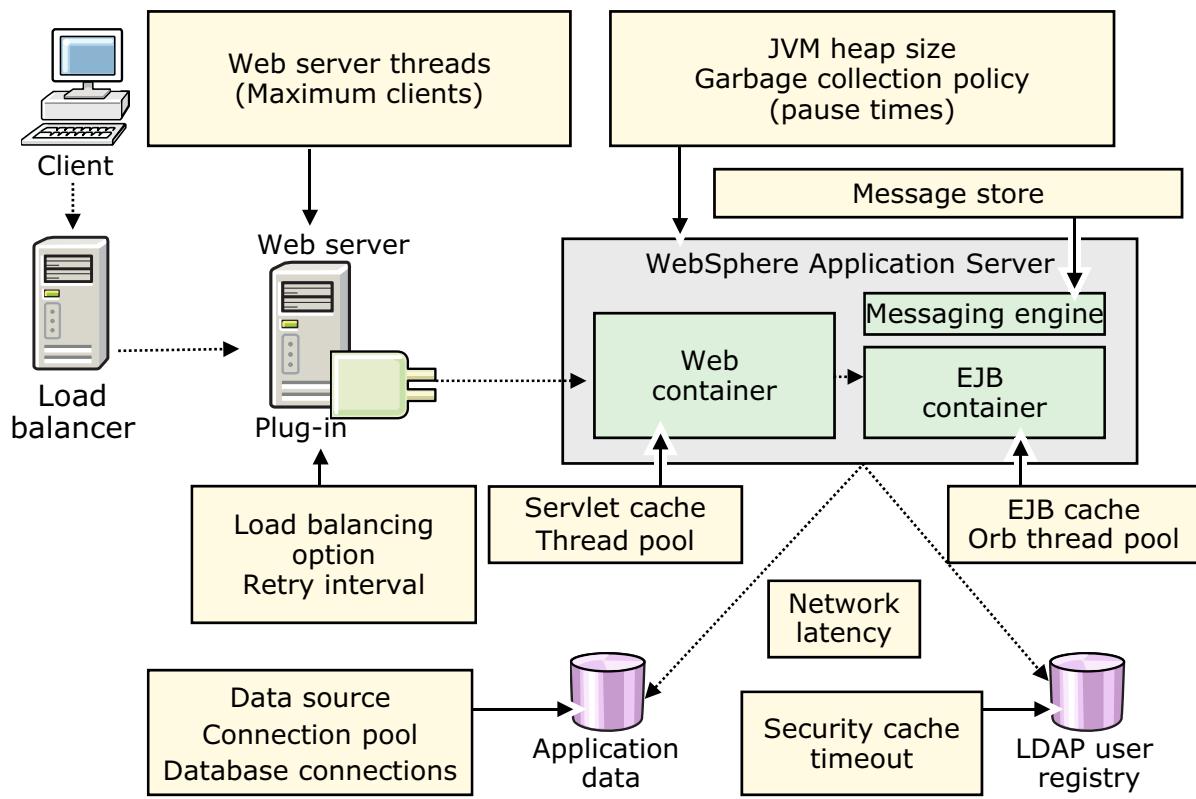
Notes:

The typical application flow is as follows:

1. A web client requests a URL in the browser input page.
2. The request is routed to the web server over the Internet.
3. The web server immediately passes the request to the web server plug-in. All requests go to the WebSphere plug-in first.
4. The web server plug-in examines the URL, verifies the list of host name aliases from which can accept traffic (based on the virtual host information), and chooses a server to handle the request.
5. A stream is created. A *stream* is a connection to the web container. It is possible to maintain a stream over a number of requests. The web container receives the request and, based on the URL, dispatches it to the correct servlet.
6. If the servlet class is not loaded, the dynamic class loader loads the servlet (`servlet init()`, then `doGet()` or `doPost()`).
7. JNDI is used for lookup of either data sources or EJBs required by the servlet.

8. Depending upon whether a data source is specified or an EJB is requested, the JNDI directs the servlet:
 - a. To the corresponding database and gets a connection from its connection pool in the case of a data source
 - b. To the corresponding EJB container, which then instantiates the EJB when an EJB is requested
9. If the EJB requested involves an SQL transaction, it goes back to the JNDI to look up the data source.
10. The SQL statement is executed and the data that is retrieved is sent back either to the servlet or to the EJB.
11. Data beans are created and handed off to JSPs in the case of EJBs.
12. The servlet sends data to JSPs.
13. The JSP generates the HTML that is sent back through the WebSphere plug-in to the web server.
14. The web server sends the output HTML page to the browser.

Possible performance bottlenecks



© Copyright IBM Corporation 2016

Figure 1-27. Possible performance bottlenecks

WA8152.2

Notes:

This diagram shows various techniques to verify that all components in an application request flow are accessible. When troubleshooting problems reported by clients, one strategy that can be used is to try to reproduce the problem, follow the request flow, and isolate one or more components that might be failing.

Unit summary

Having completed this unit, you should be able to:

- Describe stand-alone server architecture
- Describe Network Deployment (ND) cell architecture
- List and describe the function of IBM products that are involved in implementing stand-alone and distributed architectures
- Identify the components of the application server and describe the services that they provide
- Identify the components of an ND cell and describe the function of each
- Describe features of the Liberty profile
- Describe functions of Intelligent Management
- Describe the flow of an application request
- Describe the flow of administration requests
- Identify common bottlenecks in the end-to-end flow of client requests

© Copyright IBM Corporation 2016

Figure 1-28. Unit summary

WA8152.2

Notes:

Checkpoint questions (1 of 2)

1. Which of the following provides an environment for running servlets?
 - a. Web container
 - b. EJB container
 - c. The dynamic cache

2. Which of the following components are contained within the JVM of the application server? (Choose two)
 - a. Messaging engine
 - b. Embedded HTTP Server
 - c. HTTP Server plug-in
 - d. DB2 database

© Copyright IBM Corporation 2016

Figure 1-29. Checkpoint questions (1 of 2)

WA8152.2

Notes:

Write your answers here:

1.

2.

Checkpoint questions (2 of 2)

3. What protocol does an external web server use to communicate with the application server?
 - a. JDBC
 - b. SOAP
 - c. HTTP

4. The configuration for a Liberty profile server is stored in which file?
 - a. server.xml
 - b. plugin-cfg.xml
 - c. profile.xml

© Copyright IBM Corporation 2016

Figure 1-30. Checkpoint questions (2 of 2)

WA8152.2

Notes:

Write your answers here:

3.

4.



Checkpoint answers

1. (a) Web container
2. (a) Messaging engine, and (b) Embedded HTTP server
3. (c) HTTP
4. (a) server.xml

© Copyright IBM Corporation 2016

Figure 1-31. Checkpoint answers

WA8152.2

Notes:

Unit 2. Overview of performance concepts and tuning tasks

What this unit is about

This unit describes the need for performance monitoring and tuning. It defines and gives examples of common performance concepts and explains several views of performance such as: user, system, and application.

What you should be able to do

After completing this unit, you should be able to:

- Describe performance planning and design tasks that are performed during development
- Explain the value of application profiling, static code analysis, and runtime analysis that is performed during development
- Explain the goals and value of automated performance testing
- Describe the goal of performance monitoring

How you will check your progress

- Checkpoint questions

References

Performance Analysis for Java Web Sites, Joines, Willenborg, Hygh,
Addison-Wesley Longman Publishing Co.

Unit objectives

After completing this unit, you should be able to:

- Describe performance planning and design tasks that are performed during development
- Explain the value of application profiling, static code analysis, and runtime analysis that is performed during development
- Explain the goals and value of automated performance testing
- Describe the goal of performance monitoring

© Copyright IBM Corporation 2016

Figure 2-1. Unit objectives

WA8152.2

Notes:



Topics

- Performance terminology
- Different views of performance
- Test planning and design
- Code analysis and profiling
- Performance testing and tuning
- Production monitoring and tuning

© Copyright IBM Corporation 2016

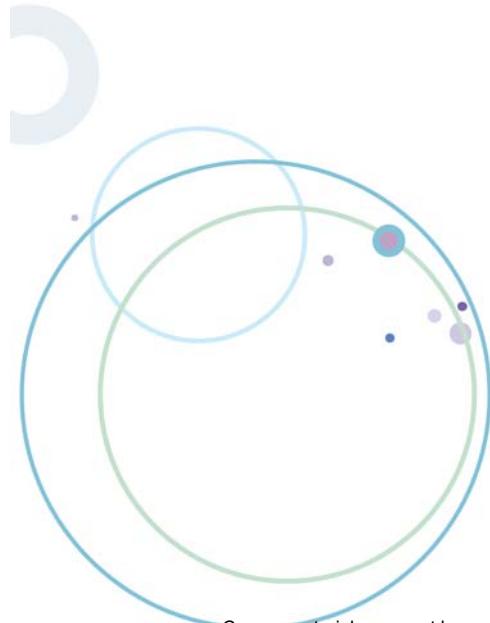
Figure 2-2. Topics

WA8152.2

Notes:

2.1. Performance terminology

Performance terminology



© Copyright IBM Corporation 2016

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

9.1

Figure 2-3. Performance terminology

WA8152.2

Notes:

After completing this topic, you should be able to:

- Describe performance terms
- Provide examples to illustrate the concepts

Performance terms

- Fundamental vocabulary for performance specialists includes these key terms:
 - Response time
 - Load
 - Throughput
 - Path length
 - Bottleneck
 - Scalability
 - Capacity

© Copyright IBM Corporation 2016

Figure 2-4. Performance terms

WA8152.2

Notes:

Fundamental vocabulary that performance specialists use includes key terms such as response time, load, throughput, path length, bottleneck, scalability, and capacity.

Response time

- Response time measures an **individual** user's wait for a request
 - Usually expressed as average
- Major components of response time:
 - Processing time
 - Transit time (usually part of processing time)
 - Any wait time in queues
- What is acceptable response time?
 - Set by current industry standards

© Copyright IBM Corporation 2016

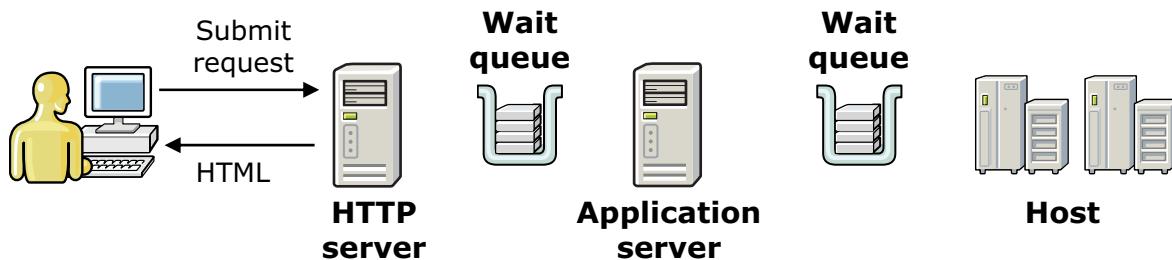
Figure 2-5. Response time

WA8152.2

Notes:

Response time measures an individual user's wait for a request and is usually expressed as average. Major components of response time include processing time, transit time (usually part of processing time), and any wait time in queues. What is an acceptable response time? Current industry standards typically set acceptable response time.

Response time: Website example



- Measured from when request is made to when HTML returns
- Website response time is a function of:
 - Raw processing time
 - Plus wait time at any number of queues
 - Plus transfer time between multiple components

© Copyright IBM Corporation 2016

Figure 2-6. Response time: Website example

WA8152.2

Notes:

Response time is measured from when a request is made to when the HTML returns.

Website response time is a function of raw processing time, the wait time at any number of queues, and the transfer time between multiple components.

Response time critical measurement

- Response time is a critical measurement
- Poor response time results in dissatisfied customers
- Many websites fail because of response time issues

- Consider response time:
 - Under peak loading
 - Under pathological loading (extreme market days)
 - Over modest dial-up or slow connections

© Copyright IBM Corporation 2016

Figure 2-7. Response time critical measurement

WA8152.2

Notes:

Response time is a critical measurement. Poor response time results in dissatisfied customers. Many websites fail because of response time issues. You should consider response time under peak loading, under pathological loading (extreme market days), and modest dial-up or slow connections.



What is the meaning of website load?

- Load is the pressure against the website
- User activity
 - Users arriving
 - Users logging in
 - Users sending requests
- Request activity
 - Requests per second, pages per hour, and so forth

© Copyright IBM Corporation 2016

Figure 2-8. What is the meaning of website load?

WA8152.2

Notes:

Load is the pressure against the website. It is expressed as user activities such as arriving, logging in, and sending requests. Request activity can be measured as requests per second, pages per hour, and so forth.



Throughput

- Measures activities that are completed in a unit of time
 - Example: Website pages that are served per second
 - Usually measured as requests per second
- Applies to many concepts, not just websites:
 - Restaurant: customers that are served per hour
 - Highway tunnels: cars through per minute
 - Department store gift wrap: packages that are wrapped per day

© Copyright IBM Corporation 2016

Figure 2-9. Throughput

WA8152.2

Notes:

Throughput measures activities that are completed in a unit of time; for example, website pages that are served per second.

Maximum throughput

- Capacity measurement of the maximum obtainable system output in a unit of time
- Restaurant example:
 - If the restaurant has only one server
 - and it takes 1 minute to serve a customer
 - then, the maximum throughput is one customer per minute
- Not a measurement of requests; only how many are fulfilled
 - Excess requests might be queued, discarded, or they might leave

© Copyright IBM Corporation 2016

Figure 2-10. Maximum throughput

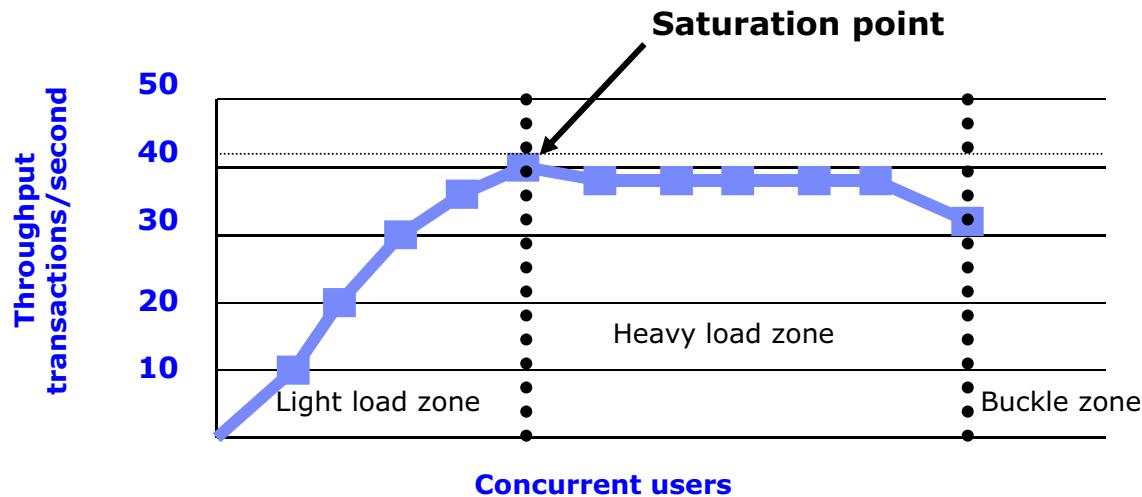
WA8152.2

Notes:

Maximum throughput is a capacity measurement. It is the maximum obtainable system output in a unit of time.

Throughput saturation

- At maximum throughput, more load does not yield more throughput
- Maximum throughput is a saturation point
 - 100% CPU utilization in the ideal case



© Copyright IBM Corporation 2016

Figure 2-11. Throughput saturation

WA8152.2

Notes:

The chart does not take into account response time. Ideally response time does not increase until the saturation point is reached.

Response time and throughput relationship

- Response time is closely tied to maximum throughput
- Beyond maximum throughput:
 - New arrivals begin to queue
 - Time in queue is added to overall response time
 - Wait is linear beyond maximum throughput

© Copyright IBM Corporation 2016

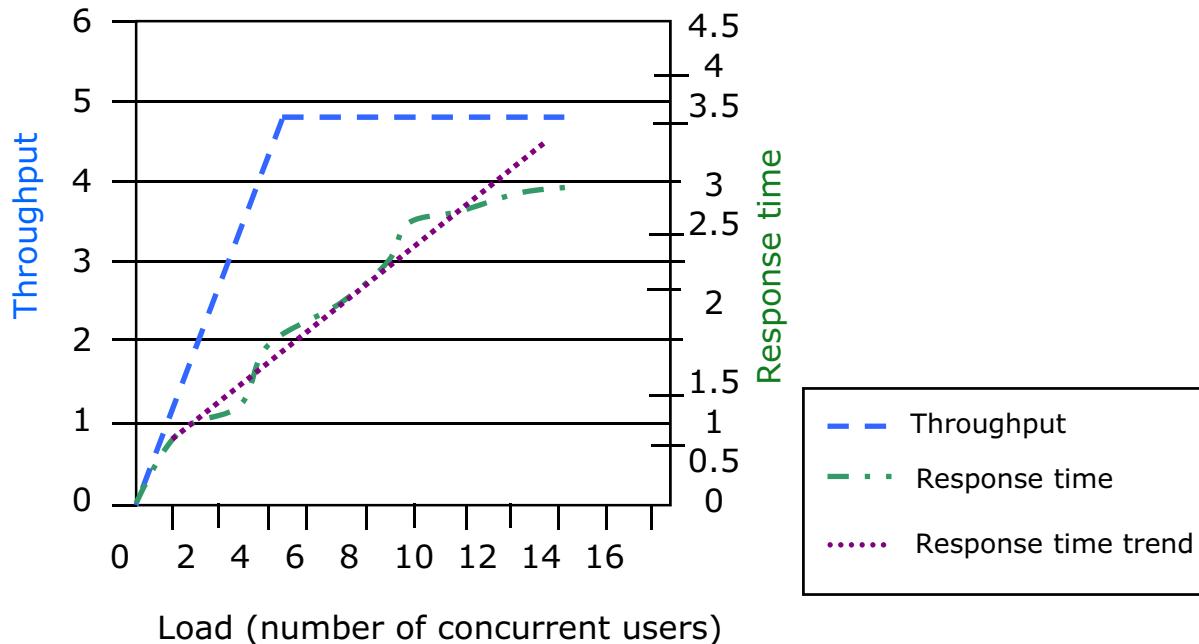
Figure 2-12. Response time and throughput relationship

WA8152.2

Notes:

Response time is closely tied to maximum throughput. When maximum throughput is exceeded, new arrivals begin to queue up. The time that is spent in the queue is added to overall response time. Wait time is linear beyond maximum throughput.

Response time and throughput saturation



© Copyright IBM Corporation 2016

Figure 2-13. Response time and throughput saturation

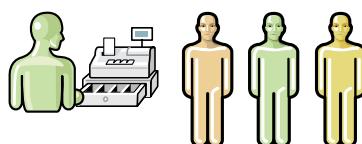
WA8152.2

Notes:

This graph shows throughput and response time as a function of concurrent users. Throughput increases linearly as the number of concurrent users increases up to maximum capacity or throughput saturation. When maximum capacity is exceeded, throughput remains constant. Response time generally shows a linearly increasing trend beyond maximum capacity as the number of concurrent users increases.

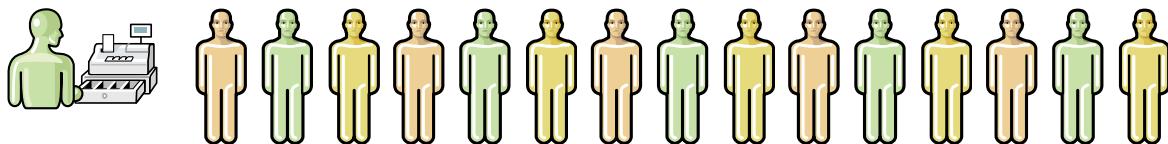
Saturated cafeteria example

- A system can support more load beyond maximum throughput
- Taking a lesson from the cafeteria:



- If customers arrive slightly faster than the server serves:
 - A line begins (queuing)
 - Response time remains good
Response time = (Time in queue + service time)
 - Throughput is unchanged (service time is constant)

- If traffic arrives significantly faster than the service rate:
 - Response time increases as wait time increases
 - Throughput eventually becomes compromised



© Copyright IBM Corporation 2016

Figure 2-14. Saturated cafeteria example

WA8152.2

Notes:

A system might support more load beyond maximum throughput. Using the example of customers who are waiting to be served in a cafeteria, two scenarios involve maximum throughput.

In the first case, customers arrive only slightly faster than the server serves. A small waiting line begins, but response time is still good. Because the service time is constant, the wait in line slightly increases the response time. This case works only for bursts of activity. Inactive periods are needed to keep the waiting line under control.

In the second case, customers arrive significantly faster than the service rate. A long line forms and response time increases as the wait time increases.

Improving performance

- How can performance be improved?
- Some techniques:
 - Reduce the number of steps that are required in a transaction
 - Speed up one or more transaction steps

© Copyright IBM Corporation 2016

Figure 2-15. Improving performance

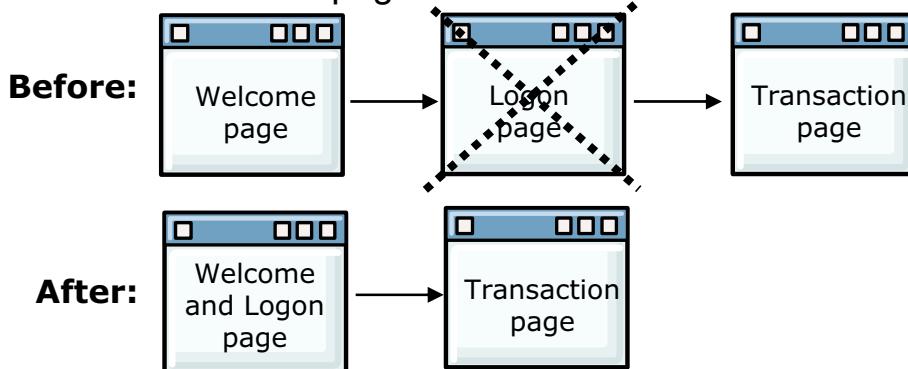
WA8152.2

Notes:

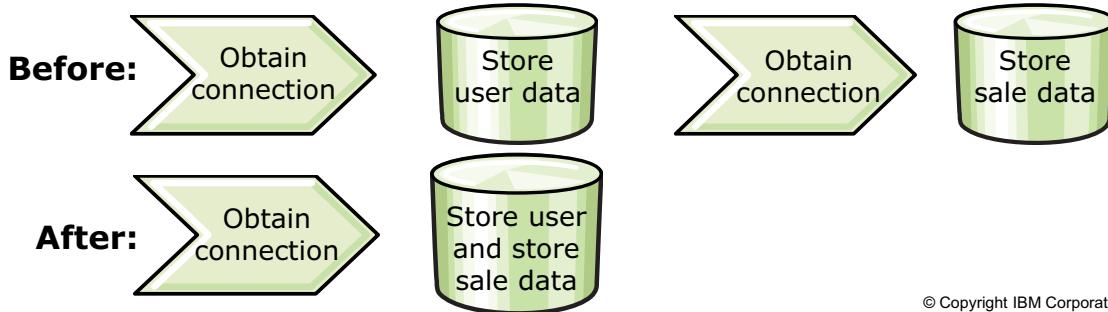
A few techniques for improving performance include reducing the number of steps that are required in a transaction and speeding up one or more transaction steps.

Remove transaction steps: Website

Consolidate website pages



Reduce application activity (path length)



© Copyright IBM Corporation 2016

Figure 2-16. Remove transaction steps: Website

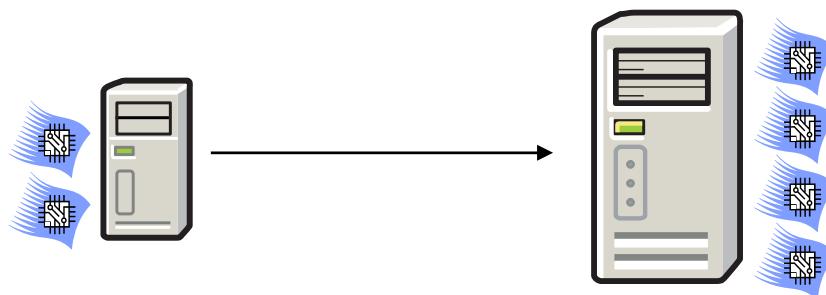
WA8152.2

Notes:

Another example is related to accessing a website. You might be able to consolidate web pages by combining the Welcome page with the logon page. Or you might be able to reduce application activities by using a single database connection to update multiple database tables.

Speed up step: Website

- Increase the speed of a step
 - Add more CPUs or faster CPUs to a server



© Copyright IBM Corporation 2016

Figure 2-17. Speed up step: Website

WA8152.2

Notes:

Using the website example, you can increase the speed of the transaction by adding more CPUs or faster CPUs to a server.



Path length

- Refers to the number of steps an action takes
- Reducing the path length speeds up a website or application
 - Speed up steps
 - Reduce the number of steps an activity takes
- Example:
 - Move a static statement out of a loop, which can reduce statement execution n times

© Copyright IBM Corporation 2016

Figure 2-18. Path length

WA8152.2

Notes:

The path length refers to the number of steps that are involved in an action. Reducing the path length speeds up a website or application. Path length reduction can be achieved by speeding up the steps and reducing the number of steps an activity takes.

For example, in the application code you can move a static statement out of a loop, which reduces the statement execution n times.

Path length reduction techniques

- Does not always require load testing to determine
 - Path length often the same for single-user and multi-user cases
 - Problems that multiple users amplify
- Best analyzed with tools
 - Debuggers
 - Code profilers
- Code reviews are also helpful

© Copyright IBM Corporation 2016

Figure 2-19. Path length reduction techniques

WA8152.2

Notes:

Load testing is not always required to determine path length because it is often the same for single-user and multi-user cases. However, multiple users might amplify problems.

Path length is best analyzed with tools such as debuggers and code profilers. Code review is also helpful.

What is a bottleneck?

- Defines a point of congestion or blockage in the system
- Appears only in multi-threaded or multi-user programming
- Users queued waiting for a shared resource
 - CPU
 - Pooled resource (threads, database connections, JMS connections)
 - Disk I/O
- Threads waiting for some task to complete
- Resolve bottlenecks in order of their severity
- Conclusion:
 - Your system is as fast as your slowest component

© Copyright IBM Corporation 2016

Figure 2-20. What is a bottleneck?

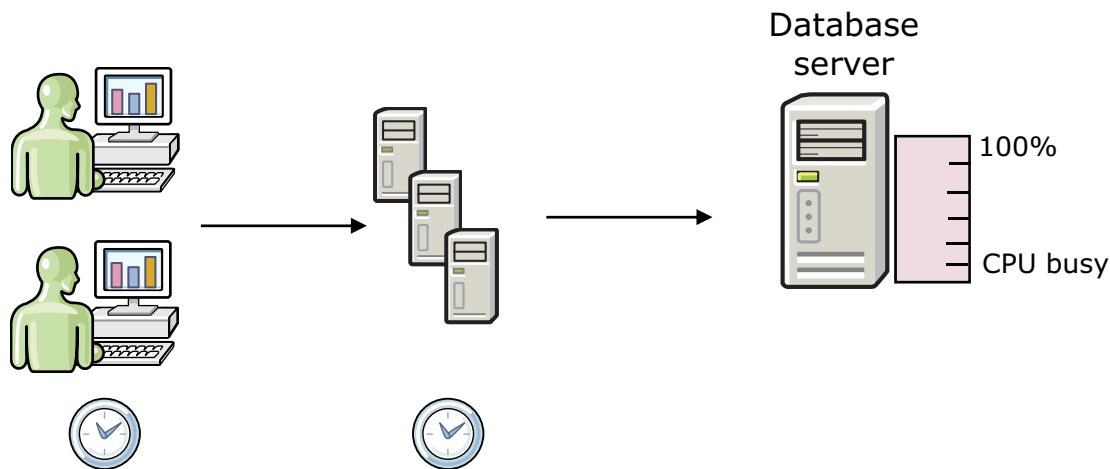
WA8152.2

Notes:

Bottleneck defines a point of congestion or blockage in the system. It appears only in multi-threaded or multi-user programming. Users are queued waiting for a shared resource such as CPU, a pooled resource, or disk I/O. Threads might be waiting for some task to complete. Resolve bottlenecks in order of their severity. Your system is as fast as your slowest component.

Bottleneck website example 1

- A slow component, such as a database server



© Copyright IBM Corporation 2016

Figure 2-21. Bottleneck website example 1

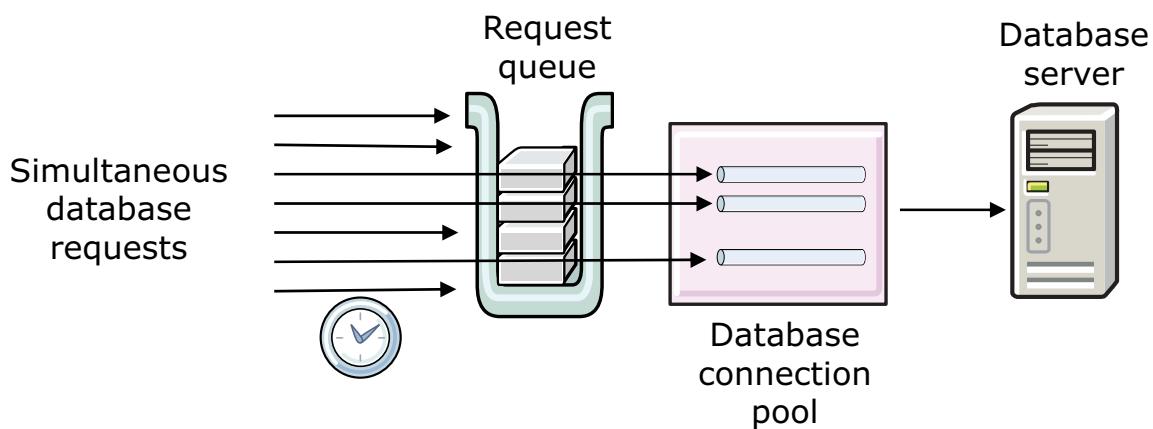
WA8152.2

Notes:

Any slow component in the request flow such as a web server or a remote database server can cause website bottlenecks.

Bottleneck website example 2

- Insufficient resources, such as a small connection pool



© Copyright IBM Corporation 2016

Figure 2-22. Bottleneck website example 2

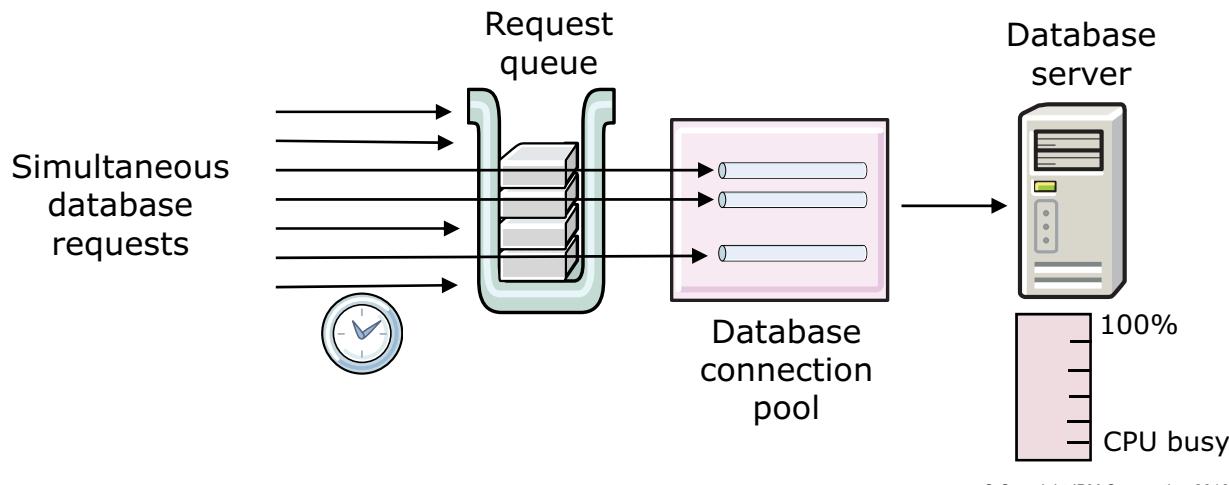
WA8152.2

Notes:

Insufficient resources, such as connection pools that are too small, can cause website bottlenecks.

Prioritizing bottlenecks

- Almost every system has bottlenecks
 - Eliminating every bottleneck is usually not feasible
 - Not every bottleneck has the same performance impact
- Eliminate bottlenecks according to severity
 - The primary activity of performance analysis



© Copyright IBM Corporation 2016

Figure 2-23. Prioritizing bottlenecks

WA8152.2

Notes:

Almost every system has bottlenecks. Eliminating every bottleneck is usually not feasible. Not every bottleneck has the same performance impact. The process of identifying and eliminating bottlenecks according to severity is one of the primary activities of performance analysis.

What is scalability?

- Defines how easily a site expands
- Sites must expand, sometimes with little warning
- Grow to support increased load
- Load can come from many sources
 - New markets
 - Normal growth
 - Extreme peaks
- Good scalability makes site growth possible and easy

© Copyright IBM Corporation 2016

Figure 2-24. What is scalability?

WA8152.2

Notes:

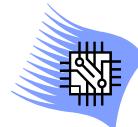
Scalability defines how easily a site expands. Sites must expand, sometimes with little warning, to support increased load. Load can come from many sources: new markets, normal growth, and extreme peaks in activity. Good scalability makes site growth possible and easy.

Scalability: Website example

- How can you grow a website to handle more load?

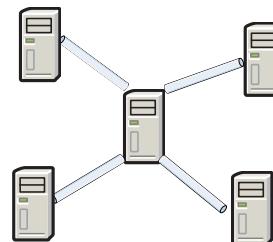
- Vertical scaling

- Add CPUs to existing machines
- Can your JVM use these CPUs?
- Can you fully use the CPUs you already have?
- Do you need more JVMs?



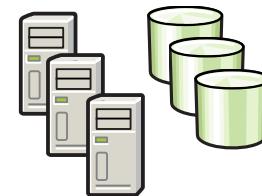
- Horizontal scaling

- Add machines to the site cluster
- How many do you need?
- Each machine hosts one or more new JVMs



- Remember the infrastructure

- Network, HTTP servers, databases, and so forth



© Copyright IBM Corporation 2016

Figure 2-25. Scalability: Website example

WA8152.2

Notes:

For website scalability, the following must be considered. How does a website grow to handle more load?

There are two approaches to scaling:

- vertical
- horizontal

For vertical scaling, you can add CPUs to existing machines, if your JVM can use these CPUs. Perhaps you can more fully use the CPUs you already have. Consider whether you need more JVMs.

For horizontal scaling, you can add machines to the site cluster. Consider how many more machines you need. Each machine might host one or more new JVMs.

Finally, you might combine both vertical and horizontal scaling if necessary.

What is capacity?

- Describes how much load the site supports
- The result of performance and load testing
- Start with simplest elements and work up
 - How much load will one server support?
 - How much load will 2, 8, or 10 servers support?
- Capacity
 - Determines hardware and infrastructure needed today
 - Leaves enough space for emergency capacity
 - Considers growth plans for future load expansion

© Copyright IBM Corporation 2016

Figure 2-26. What is capacity?

WA8152.2

Notes:

Capacity describes how much load the site supports. Discovering the site's capacity is the result of performance and load testing.

Start with simplest elements and work up. How much load will one server support? How much load can 2, 8, or 10 servers support?

Capacity dictates current hardware and infrastructure requirements. However, you should leave enough space for emergency capacity and consider growth plans for future load expansion.

Capacity example: Website

- How many servers are needed to handle 10,000 visitors per day?
- Do you need extra capacity for high-volume days?
 - Market runs
 - Holiday shopping season
 - Should you buy a few large servers or several smaller ones?
- When do you exceed this capacity?
- How can you grow the site?
 - Add servers or CPUs?
 - At what point, do you upgrade the infrastructure?

© Copyright IBM Corporation 2016

Figure 2-27. Capacity example: Website

WA8152.2

Notes:

Here are some capacity considerations for a website:

- How many servers are needed to handle 10,000 visitors per day?
- Do you need extra capacity for high-volume days that market runs and holiday shopping seasons cause?
- Should you buy a few large servers or several smaller ones?
- When do you exceed this capacity? Can you grow the site by adding servers or CPUs or both?
- At what point should you upgrade the infrastructure?

Performance terminology summary

- Each concept is probably suitable for a scholarly dissertation
 - More theoretical and mathematical information is available
- Keep terminology accessible for external groups
 - What are they trying to find in their testing?
 - What goals are important to them?
- Keep terminology consistent within the team
 - Many shops misuse some of these terms such as throughput, capacity, and scalability
 - Make sure that you understand what your customer means

© Copyright IBM Corporation 2016

Figure 2-28. Performance terminology summary

WA8152.2

Notes:

In summary, keep in mind the following about performance terminology.

Keep terminology accessible for external groups. What are they trying to find in their testing? What are the goals important to them?

Keep terminology consistent within the team. Many people misuse some of these terms. Make sure that you understand what your customer means.

2.2. Different views of performance

Different views of performance



© Copyright IBM Corporation 2016

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

9.1

Figure 2-29. Different views of performance

WA8152.2

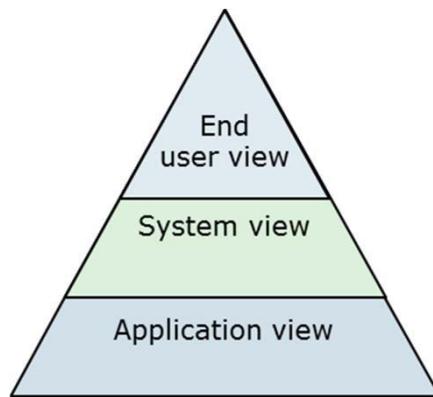
Notes:

After completing this topic, you should be able to:

- Describe the three different views of performance
- Explain the value of monitoring each of them

Different views of performance

- User asks: Why is the system so slow?
- Administrator asks: Is WebSphere running OK?
- Developer asks: Why is the Order EJB consuming so much time?



© Copyright IBM Corporation 2016

Figure 2-30. Different views of performance

WA8152.2

Notes:

The user view

The user view looks at the overall performance of the website from the user's perspective. Monitoring it allows you to ensure that response times are meeting user requirements and provides an external alert when problems arise.

The system view

This view is concerned with the basic health of all systems that are involved in processing the request. Monitoring it allows you to identify any system that might be experiencing problems and is causing performance degradation.

The application view

The application view looks at the performance of the individual application components that are involved in satisfying the request. Monitoring their resource usage allows you to identify poorly performing components within the application flow.



User view concerns

- Response time
 - Amount of time that elapses from when a request is sent to when a response is received
- Load
 - Number of users who concurrently use the website
- Throughput
 - Number of requests per second the website can handle

© Copyright IBM Corporation 2016

Figure 2-31. User view concerns

WA8152.2

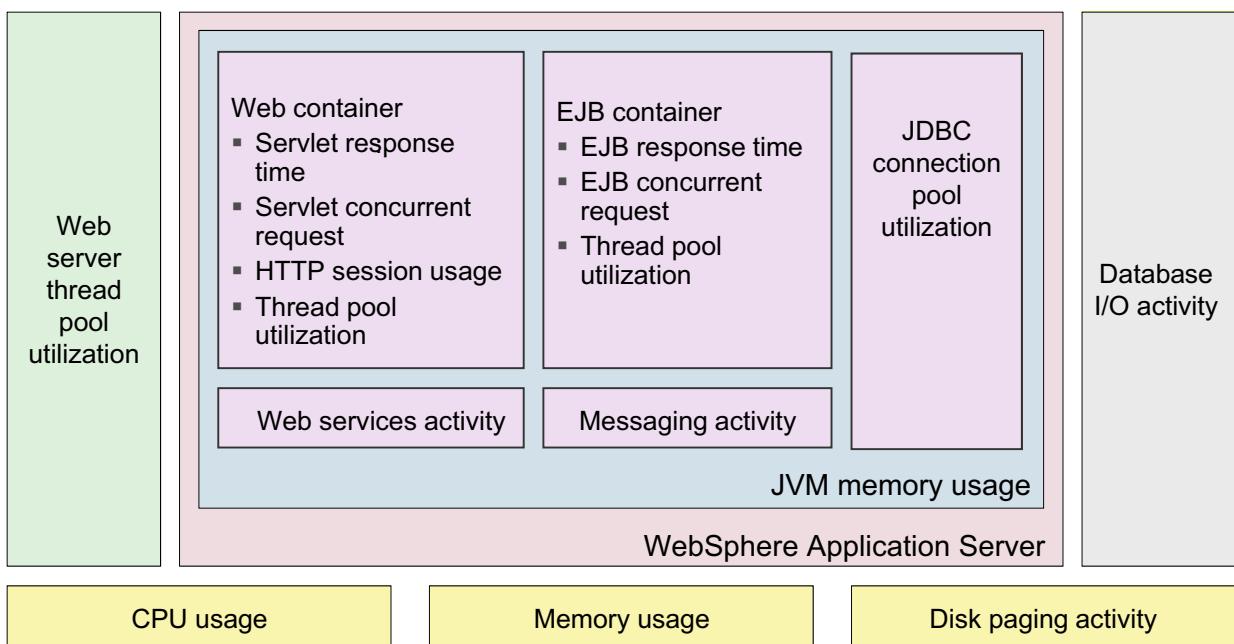
Notes:

The user's view of performance is formed from the following metrics:

- **Response time:** Amount of time that is elapsed from when a request is sent to when a response is received
- **Load:** Number of users who concurrently use the website
- **Throughput:** Number of requests per second that the website can handle



System view



© Copyright IBM Corporation 2016

Figure 2-32. System view

WA8152.2

Notes:

The system view focuses on the overall health of the system by looking at critical system components and resources to make sure that they are running correctly. If a problem occurs, it helps identify which resources are constrained and should be investigated further.



System view concerns (1 of 2)

- Core system performance:
 - Is the CPU over-utilized?
 - Is the system paging memory to disk excessively?
 - Is much time spent accessing the disk?
- Web server health:
 - How many requests is the web server currently processing (web server thread pool utilization)?
- WebSphere Application Server health:
 - How much time does the JVM spend doing garbage collection?
 - How many requests are the web or EJB containers currently processing (container thread pool utilization)?

© Copyright IBM Corporation 2016

Figure 2-33. System view concerns (1 of 2)

WA8152.2

Notes:

Core system performance: Is the CPU over-used? Is the system paging memory to disk excessively? How much time is spent accessing the disk?

Web server health: How many requests is the web server currently processing (web server thread pool use)?

WebSphere Application Server health: How much time does the JVM spend in doing garbage collection? How many requests are the web or EJB containers currently processing (container thread pool use)?



System view concerns (2 of 2)

- Application resource health:
 - Is there a servlet or EJB component that runs unusually slow (servlet or EJB response time)?
 - Is a servlet or EJB component over-utilized (servlet or EJB concurrent request)?
 - How many live HTTP session objects are currently allocated (HTTP session usage)?
 - Is a JDBC connection pool over-utilized?
- Database health:
 - Are the buffer pools sized adequately?
 - Are there enough listener agents?

© Copyright IBM Corporation 2016

Figure 2-34. System view concerns (2 of 2)

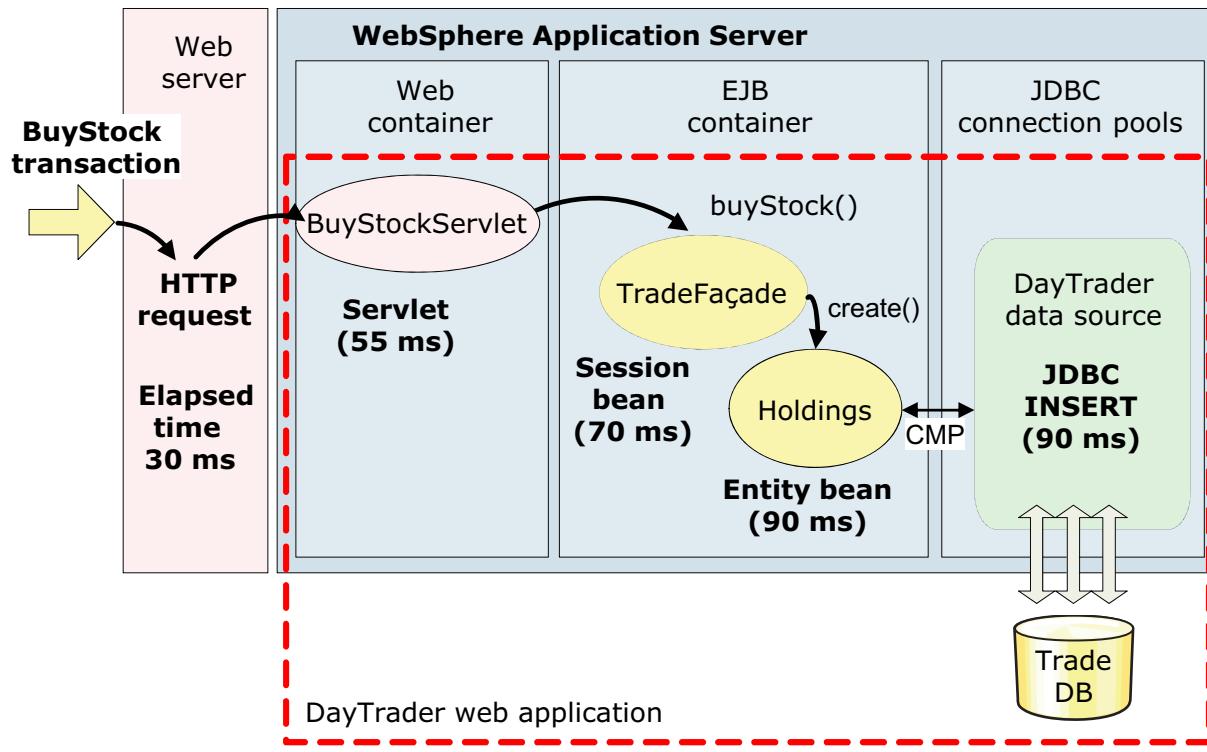
WA8152.2

Notes:

Application resource health: Is there a servlet or EJB component that runs unusually slowly (servlet and EJB response time)? Is a servlet or EJB component over-utilized (servlet or EJB concurrent request)? How many live HTTP session objects are currently allocated (HTTP session usage)? Is a JDBC connection pool over-utilized?

Database health: Are the buffer pools sized adequately? Are there enough listener agents?

Application view



© Copyright IBM Corporation 2016

Figure 2-35. Application view

WA8152.2

Notes:

The application view measures the performance of the individual application components that are involved in processing the user request. It allows you to follow the execution flow and monitor the components that participate in it. It can be used during problem diagnosis to drill down and find who is using the particular system resource that is exhibiting poor performance.

Monitoring the application view allows you to understand the breakdown of time that is spent in the components that a user transaction traverses across system boundaries.

2.3. Test planning and design

Test planning and design



© Copyright IBM Corporation 2016

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

9.1

Figure 2-36. Test planning and design

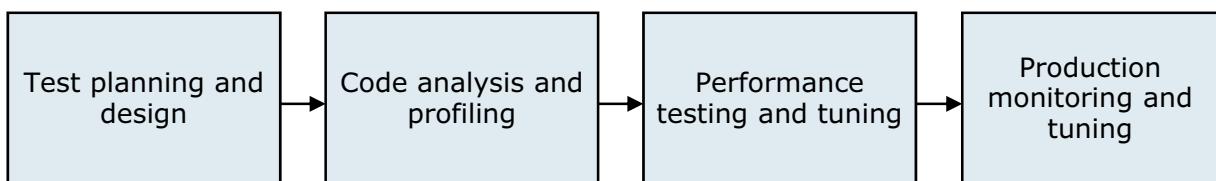
WA8152.2

Notes:

Performance tasks: What to do and when to do it?

Application lifecycle

Analysis and design Implementation Pre-deployment Post-deployment



© Copyright IBM Corporation 2016

Figure 2-37. Performance tasks: What to do and when to do it?

WA8152.2

Notes:

Test planning and design should be done during the analysis and design phase of the application lifecycle.

Test planning and design objectives

- Develop a performance test plan
 - Define performance test scope
 - Describe performance goals and success criteria
 - Identify required resources
 - Generate a test plan
- Perform workload analysis
 - Identify critical business functions to be measured
 - Specify performance measurement metrics
 - Define scenarios and user types
 - Generate a workload analysis document

© Copyright IBM Corporation 2016

Figure 2-38. Test planning and design objectives

WA8152.2

Notes:

The goal of test planning is to create a test plan document that describes what is tested, how it is tested and measured, and who and what are needed to successfully implement it.

Workload analysis (as defined in Rational Unified Process) is a design time task that includes the following activities:

- Clarify the objectives of performance testing and the use cases
- Identify the use cases to be implemented in the model
- Identify the actors and actor characteristics to be simulated or emulated in the performance tests
- Identify the workload to be simulated or emulated in the performance tests (in terms of number of actors, actor classes, and actor profiles)
- Determine the performance measures and criteria
- Review the use cases to be implemented and identify the execution frequency
- Select the most frequently called use cases and those cases that generate the greatest load on the system

- Generate test cases for each of the use cases that are identified in the previous step
- Identify the critical measurement points for each test case

The main objective of workload analysis is to define a realistic representative workload that allows performance risks to be accurately assessed. The generated artifact, a workload analysis document, communicates in greater detail than the test plan what is tested, how it is run, and how the tests are verified.



Test planning and design considerations

- Involve all of the necessary people
 - Business analysts
 - Architects
 - Developers
 - Testers and QA team
 - Administrators
- Think real world or production level testing
 - Ensure that tests represent real user activity
 - Ensure that the test environment is representative of production

© Copyright IBM Corporation 2016

Figure 2-39. Test planning and design considerations

WA8152.2

Notes:

All five groups are stakeholders and need to be included in the planning phase to make sure that the performance test plan is going to test the system properly:

- **Developers** understand what pieces of the code do the work and provide the major functions.
- **Architects** know how the application was designed and the layout of the code.
- **Business analysts** know what they are marketing and how the public plans to use the application.
- **Testers and QA team members** know the test environment and do most of the work to prepare for the performance test.
- **Administrators** are called to set up the performance test environment.

This group facilitates and constructs the performance test plan document that is used throughout the testing efforts.



Test plan creation steps

- Identify test requirements
 - Define what is tested
 - Establish test priorities
- Define test strategy
 - Describe testing approach
 - Define what makes the test a success
 - Specify what level of performance is required
- Identify resources that are needed for testing
 - Time
 - Money
 - Hardware and software
- Create a schedule
 - Identify who does what
 - Define order of tests
- Document the test plan

© Copyright IBM Corporation 2016

Figure 2-40. Test plan creation steps

WA8152.2

Notes:

The test plan should specify the types of performance test to run. Different types are discussed in the **Performance testing and tuning** topic.

Other important resources that are needed for a successful performance test should also be planned:

- Time for testing should be scheduled following development. This schedule includes both time to develop test content and time to run the test.
- Money to buy the necessary performance test tools should be allocated. Test tools can be expensive, but this resource is money that is spent well.
- Hardware that is comparable to the deployment environment should be used. If it is not available, you must extrapolate data from smaller systems to the deployment systems.

Make sure to document the test plan and adjust it as needed as testing progresses.



Workload analysis steps (1 of 2)

- Identify business processes to be tested
 - Include major business processes
 - Cover at least 70% to 80% of your code
 - Define test cases and data
- Define workloads
 - Identify user types and characteristics
 - Define the percentage of users that run each business process
 - Approximate data access patterns
 - Define scenarios

© Copyright IBM Corporation 2016

Figure 2-41. Workload analysis steps (1 of 2)

WA8152.2

Notes:

A workload is a complete test scenario (target load) to be placed on the system under tests that consist of:

- User attributes (types and characteristics)
- Transaction mix and frequency
- Data set
- Test environment configuration
- Measurement criteria

It should be a realistic representation of actual users and transactions. It is important to define what is monitored and what is acceptable for the application testing. Well-chosen measurement points make performance analysis easier.

The first step in workload definition is to clarify and document the goals of the performance-testing project. Testing goals are defined with user or customer involvement and are documented to ensure that everyone agrees with the goals. The major activity of workload definition is identifying and documenting the critical business functions that make up most of the user activity. Each business

function or task is first identified and then described in terms of user input data and data accessed. The workload definition must describe the details about how the user works through the scenario, including average typing rates and user think-time intervals. The definition should also describe an acceptable approximation for data access frequencies and patterns for data that is retrieved from the computer database as part of the user scenario.

Document the workload analysis and make sure that it includes a user percentage breakdown by business process that reflects how the application is used in production. By documenting all of the user scenarios, the project members have complete user documentation for later creating test scripts and schedules.



Workload analysis steps (2 of 2)

- Select measurement criteria
 - Response times
 - Throughput
 - Validity of transactions
- Generate workload analysis document
 - Used to create actual tests and scenarios

© Copyright IBM Corporation 2016

Figure 2-42. Workload analysis steps (2 of 2)

WA8152.2

Notes:

Select measurement criteria that includes:

- Response times
- Throughput
- Validity of transactions

During workload analysis, a workload analysis document should be generated which is used to create actual tests and scenarios.

Test environment setup considerations

- Start test environment setup once test plan and workload analysis is documented
- Servers should:
 - Use production-level code
 - Duplicate production configuration
 - Not be a production system
- Client systems should:
 - Be able to drive the planned maximum number of virtual users
 - Assign a different IP address for each virtual user
- Network should be set up with:
 - A dedicated LAN between client and application
 - A separate LAN for testing tool and client communication
- Environment should be:
 - Representative of deployment
 - Controlled
 - Dedicated

© Copyright IBM Corporation 2016

Figure 2-43. Test environment setup considerations

WA8152.2

Notes:

It is a good idea to start setting up the test environment as soon as the test plan and workload analysis documents are produced. This plan allows more time to procure, install, and configure all of the hardware, system software, and testing tools required.

The following needs to be set up in the test environment:

- Lab hardware
- System software
- Data set
- Test and monitoring tools

Ideally, the servers are similar to servers that are used in production.

The client systems are used to drive the simulated requests.

The network should be dedicated. In the best circumstances, the user traffic and application communication is on separate LANs.

Most importantly, the environment needs to be controlled and dedicated. Only certain people are allowed to implement modifications so that everything can be documented.

2.4. Code analysis and profiling

Code analysis and profiling



© Copyright IBM Corporation 2016

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

9.1

Figure 2-44. Code analysis and profiling

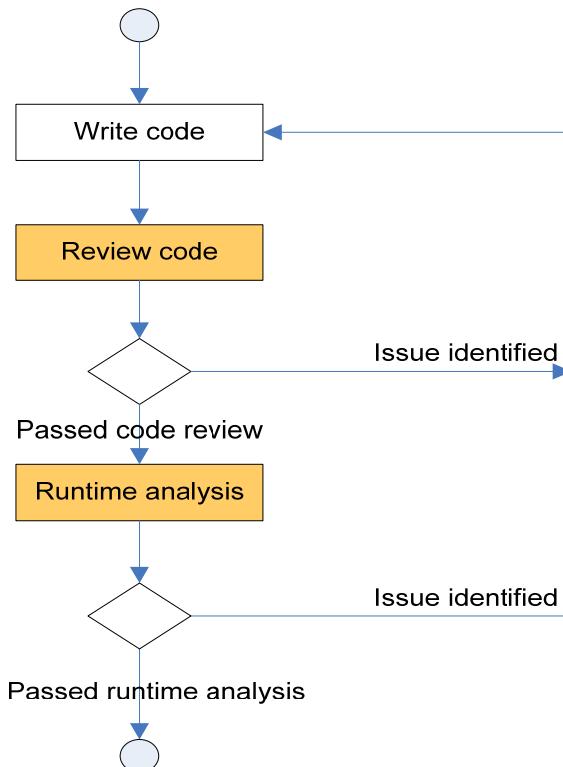
WA8152.2

Notes:

Code analysis and profiling overview

Two types of code analysis should be performed:

- Static code analysis
 - Review the code to implement performance best practices
- Runtime analysis
 - Profile the application to uncover and resolve potential performance problems such as memory leaks, performance bottlenecks, and threading problems



© Copyright IBM Corporation 2016

Figure 2-45. Code analysis and profiling overview

WA8152.2

Notes:

Two types of code analysis should be done: static code analysis and runtime analysis.

Static code analysis involves reviewing the code to implement performance best practices.

Runtime analysis involves profiling the application to uncover and resolve potential performance problems such as memory leaks, performance bottlenecks, and threading problems.



Example: Rational Application Developer code review

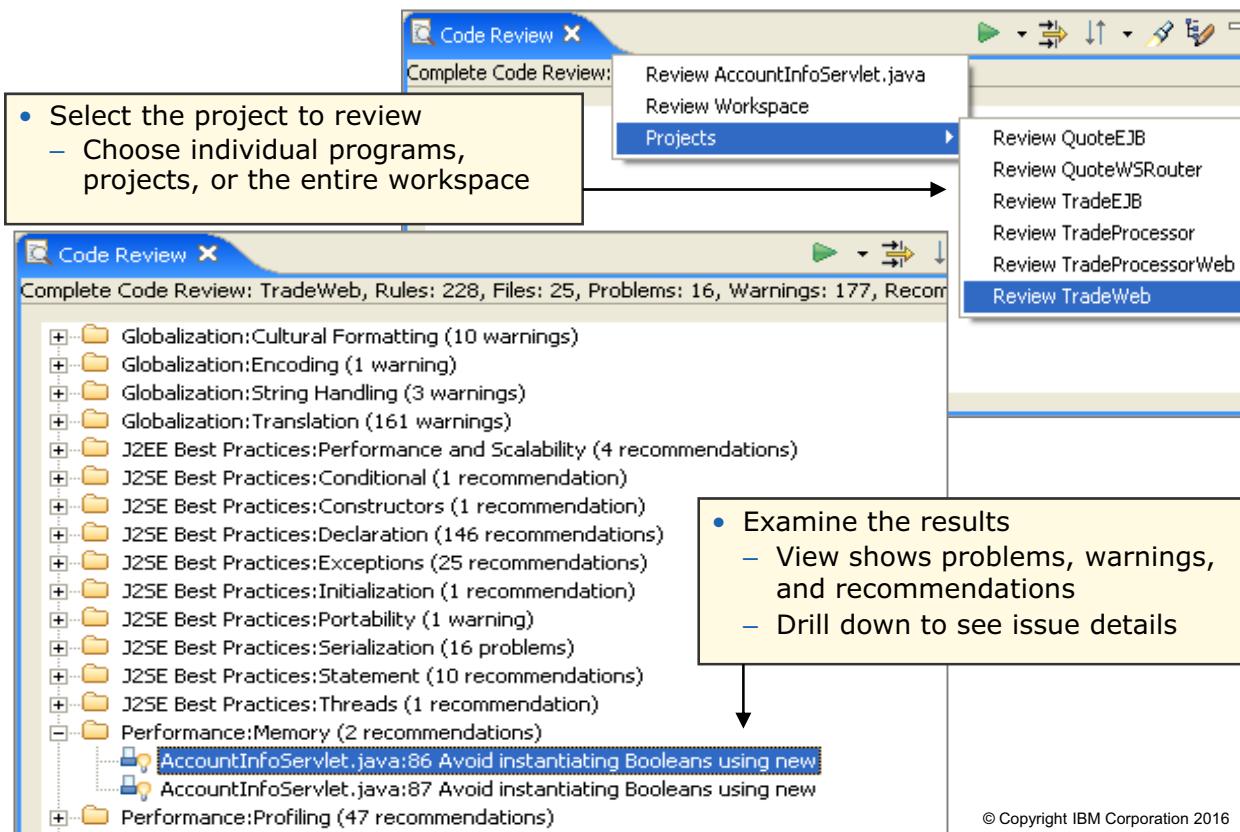


Figure 2-46. Example: Rational Application Developer code review

WA8152.2

Notes:

Static code analysis can be facilitated by using the right tools. Static code analysis can be done by using the IBM Rational Application Developer tool.

Runtime analysis

- The goal is to understand the code's runtime behavior through:
 - **Performance analysis** that can detect performance bottlenecks by looking at method flow and execution times
 - **Memory analysis** to identify memory leaks by monitoring object allocations in the heap and garbage collection
 - **Thread analysis** to detect thread bottlenecks by watching for lock contention and deadlock situations
- Profiling allows developers to recognize, isolate, and fix the following types of problems:
 - Performance bottlenecks
 - Object leaks, loitering objects
 - Excessive object creation, garbage collection
 - HTTP session management
 - Path length
 - System resource limits
 - Thread bottlenecks

© Copyright IBM Corporation 2016

Figure 2-47. Runtime analysis

WA8152.2

Notes:

Runtime analysis consists of running instrumented code and analyzing the collected data to find programming errors and improve software performance.

Performance bottlenecks are areas within the code that slow down or halt execution.

A memory leak is an error in a program's dynamic store allocation logic that causes it to fail to reclaim discarded memory. That is, objects that are no longer required are not reclaimed.

Unexpectedly large numbers of such instances might suggest a memory leak. A memory leak, if severe, can lead to the collapse of your application due to its running out of memory. Objects that continue to hold references to other objects cause memory leaks, thus preventing garbage collection from reclaiming the held objects.

A thread contention, or race conditions, occurs when a thread is waiting for a lock or resource that another thread holds. Programmers often add synchronization mechanisms to avoid these contentions, but it is possible that the synchronization itself can lead to deadlocks.

A thread deadlock is a condition in which two independent threads of control are blocked, each waiting for the other to do something. Two threads that wait for each other can bring an application, or one section of the application, to a complete halt.

Application profiling tools

- Rational Application Developer has a built-in profiler that can help developers isolate and fix performance problems
- Capabilities include:
 - Execution history analysis, which shows how call chains were executed and their execution times
 - Memory analysis visualizes overall memory usage and enables garbage collection analysis
 - Memory leak analysis allows you to take heap dumps and generate a list of leak candidates
 - Thread analysis shows the thread-level behavior of code execution
- IBM Support Assistant: Health Center tool
 - Java method profiling
 - Lock analysis
 - Garbage collection
 - Threading

© Copyright IBM Corporation 2016

Figure 2-48. Application profiling tools

WA8152.2

Notes:

The Rational Application Developer profiling tool can collect data across multiple servers to support distributed Java EE applications.

Health Center is a low memory use monitoring tool. It runs alongside an IBM Java application with a small affect on the application's performance. Health Center monitors several application areas, and uses the information to provide recommendations and analysis that help you improve the performance and efficiency of your application. Health Center can save the data that is obtained from monitoring an application and load it again for analysis later.

2.5. Performance testing and tuning

Performance testing and tuning



© Copyright IBM Corporation 2016

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

9.1

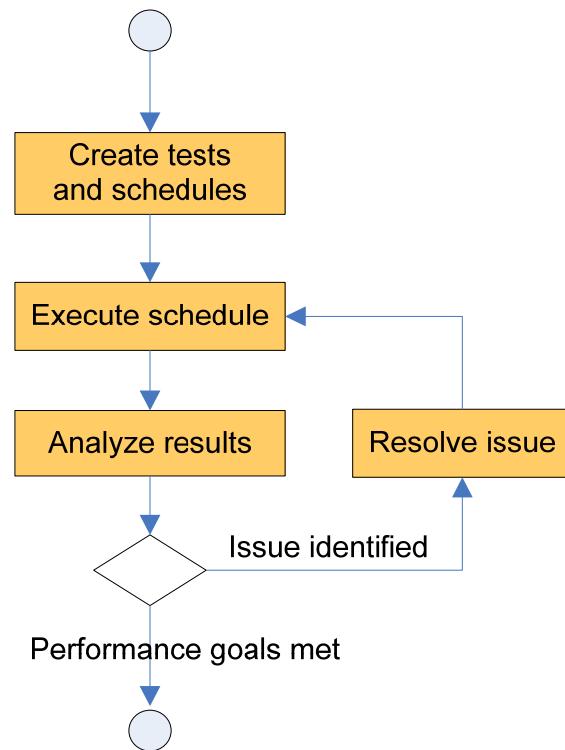
Figure 2-49. Performance testing and tuning

WA8152.2

Notes:

Performance testing and tuning overview

- Test development and validation
 - Create tests and schedules
- Test execution and results analysis
 - Execute schedules
 - Analyze results
 - Resolve performance problems



© Copyright IBM Corporation 2016

Figure 2-50. Performance testing and tuning overview

WA8152.2

Notes:

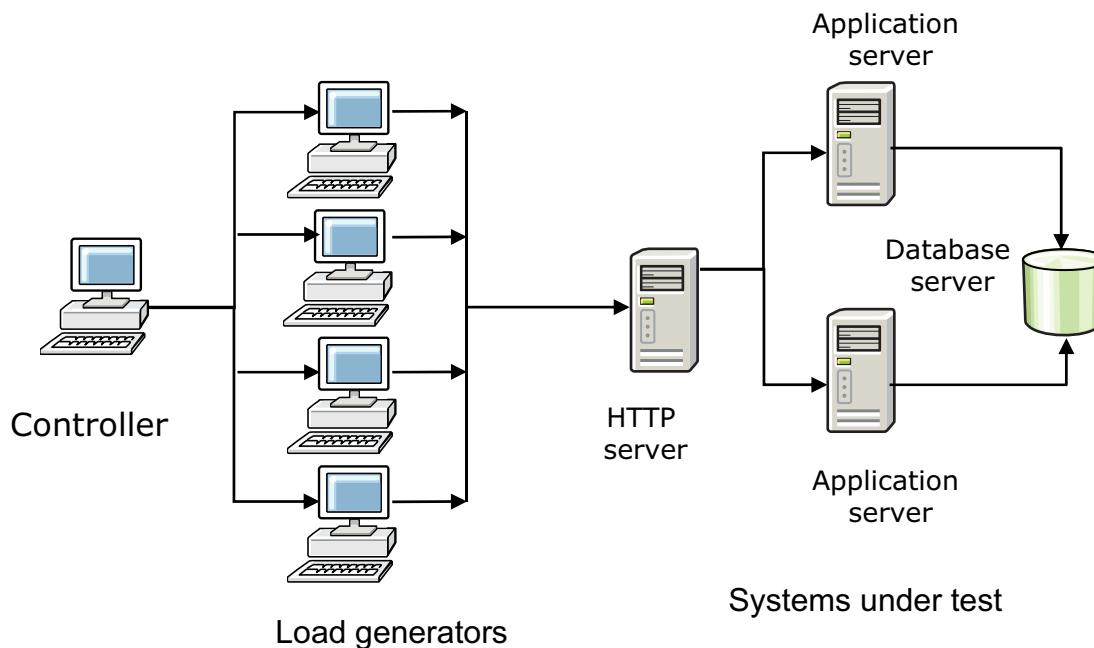
The test execution and results analysis steps are iterative. The length of time that is spent depends on the number of bottlenecks that need to be resolved.

The results analysis step uses multiple tools that are based on specific problems. These tools include:

- Performance testing tools such as IBM Rational Performance Tester and Apache JMeter
- WebSphere tools such as Tivoli Performance Viewer
- Monitoring tools such as ITCAM for Application Diagnostics
- System tools such as Perfmon for Microsoft Windows

What is performance testing?

- The process of exercising an application by emulating actual users with a load generation tool for finding system bottlenecks



© Copyright IBM Corporation 2016

Figure 2-51. What is performance testing?

WA8152.2

Notes:

A load generation tool creates the load that simulates actual users. Typically, a master computer controls other computer, and “virtual testers”, to emulate this production load. When the test run is complete, the data is returned to the master where the user analyzes the data to find the performance bottlenecks.

Performance testing includes not only testing the application and application server but also all of the other pieces that help create the entire environment. Typical activities to emulate include:

- User transactions and scenarios
- Peak activity periods
- Batch jobs and other internal system processes
- Administrator tasks (for example, system backups)



Typical goals of performance testing

- Identify system response times
 - Validation of requirements and performance goals
 - Benchmarking
 - Service level agreements (SLAs)
- Determine the maximum number of users for a system
 - Capacity planning
 - Scalability
- Discover optimum or minimum configurations

© Copyright IBM Corporation 2016

Figure 2-52. Typical goals of performance testing

WA8152.2

Notes:

Performance testing allows you to:

- Determine the response time of an application
- Determine how many users the system can support
- Determine the optimum system configuration
- Verify that different hardware and software works satisfactorily
- Find out what happens when a system is put under heavy load

Beyond observation and gathering, engineers use the data and metrics to tweak the target application to improve its performance. This discovery and resolution of bottlenecks (instances of slow or unacceptable performance) is the real return on investment (ROI) of a performance testing tool.

Benchmarking is the process of comparing a system's performance to an established level to match or improve upon it.

Performance testing can help to reduce system costs by determining what system resources, such as memory and disk resources, are needed to deliver acceptable performance. Understanding the

minimum requirements enables you to make better decisions about what hardware and software needs to be purchased.

Types of performance tests

- Load testing
 - Simulates anticipated user load and actual business operations
 - Identifies load level with acceptable response times
 - Provides metrics on performance bottlenecks
- Stress testing
 - Evaluates performance at a level well beyond estimated loads
 - Performs sustained tests at high loads or short-term burst loading
 - Determines the load under which a system fails and how it fails
- Longevity and endurance testing
 - Simulates normal loads for a long time
 - Identifies performance problems, such as slow memory leaks, that might appear after a system is running for an extended time

© Copyright IBM Corporation 2016

Figure 2-53. Types of performance tests

WA8152.2

Notes:

Load testing refers to more realistic loads for an application. It is based on business patterns and should simulate actual business operations. It is suggested that you do load testing at 10 to 20 percent above the projected user load to accommodate for growth.

Stress testing refers to long runs at saturation levels. It also refers to short-term burst loading, such as a large numbers of logins at one time. In general during stress testing, you ignore poor response times.

Other types of performance tests include:

- **Failover testing:** Simulates a hardware or software failure to test failover mechanisms.
- **Configuration testing:** Tests for compatibility issues, determines minimal and optimal configuration of hardware and software, and determines the effect of adding or modifying resources such as memory, disk drives, and processor.



Test execution

- Iterative process to identify performance problems and resolve them
- Guidelines include:
 - Document each test run
 - Obtain a baseline
 - Use identical workloads for comparison tests
 - Change only one variable at a time when comparing results
 - Take measurements in a controlled environment (no extraneous activity)
 - Ensure that a steady state or a stable load is achieved for critical measurements

© Copyright IBM Corporation 2016

Figure 2-54. Test execution

WA8152.2

Notes:

In the execution phase, repeated runs are made to capture performance data, identify bottlenecks, and to do further monitoring to help resolve those bottlenecks. Remember that your tests need to model accurately all of the system users, not just a single user type, or a single set of tasks. Realism is the key.

The first step is to obtain a one-user baseline measurement so you know what the performance was at the beginning.

It is suggested that you change only one variable (user load, system setting, database software, server hardware, and so forth) at a time for tests that are compared. If you change more than one variable, it becomes unclear as to which element caused the difference. Also, use of a controlled test environment is suggested to accurately evaluate changes in software or hardware.

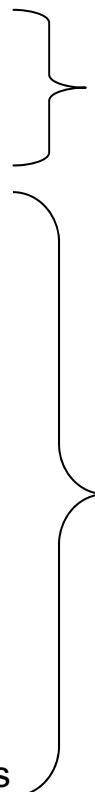
If you begin taking measurements at the start of a test run, only a few users are active. If what you want to know is the response time under the maximum load of your test, then you must wait until all users are active.

The application environment, when under consistent load, reaches a stable memory usage after several minutes of operation. When this event occurs, the system is said to be in a steady state of

operation. To take performance measurements that predict expected user response times, the system must first reach its steady state. Typically the system resource usage (especially memory usage, swapping and paging, and disk usage) is used to determine when steady state is achieved in the course of a performance test. Sometimes the response time data is also checked to make sure that consistent values are being measured when the system resource usage indicates that system steady state is achieved.

Results analysis and problem resolution

- Analyze results to find:
 - Slow pages
 - Resource bottlenecks
- Resolve application bottlenecks
 - Design and architecture issues
 - Code implementation issues
 - Poorly tuned queries
- Tune run time "Outside In"
 - HTTP server queues
 - Database connection pool
 - Application server (connections, heap sizes)
- Optimize hardware settings
 - Server system resources (memory, CPU)
 - Network health
- Adjust operating system settings



Identify performance problems

Resolve problems in order of severity

© Copyright IBM Corporation 2016

Figure 2-55. Results analysis and problem resolution

WA8152.2

Notes:

You need to be sure to verify the results. Performance bottlenecks can come from a wide variety of sources. Identify bottlenecks by severity; then work to resolve them in the order of severity. By working in this way, you do not chase problems that result from larger, more significant problems.

Implement the changes to fix the problem and rerun the same test to verify the result of the modification. When that issue is resolved, move to the next issue in order of severity with the newly implemented changes.

This graphic shows a general outline of the order in which problems might be resolved. Your own order of resolution might be different from the order that is shown in the outline. Each application performance tuning exercise is unique and different from the next.



Performance test development and load-driving tools

- Rational Performance Tester
- Apache JMeter
- Others
- Most tools allow you to:
 - Create test scripts without programming
 - Run load tests
 - Monitor the test as it runs
 - Generate and save performance reports

© Copyright IBM Corporation 2016

Figure 2-56. Performance test development and load-driving tools

WA8152.2

Notes:

IBM Rational Performance Tester is a performance testing solution that validates the scalability of web and server applications. Rational Performance Tester identifies the presence and cause of system performance bottlenecks and reduces load testing complexity.

Apache JMeter is used to test performance both on static and dynamic resources. It is used to simulate a heavy load on a server, group of servers, network, or object to test its strength or to analyze overall performance under different load types. You can use it to make a graphical analysis of performance or to test your application and server behavior under different concurrent user loads.

2.6. Production monitoring and tuning

Production monitoring and tuning



© Copyright IBM Corporation 2016

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

9.1

Figure 2-57. Production monitoring and tuning

WA8152.2

Notes:



Performance monitoring overview

- The goal of performance monitoring is to ensure that your applications are running optimally
 - If not, to determine why
- Monitoring is central to performance diagnosis, tuning, and management solutions
 - Performance test and tuning
 - Problem identification and diagnosis
 - System health and availability management

© Copyright IBM Corporation 2016

Figure 2-58. Performance monitoring overview

WA8152.2

Notes:

The goal of performance monitoring is to collect runtime statistics about your application and its environment to quantify their performance behavior. It allows you to determine whether your application meets its performance objectives and helps to identify any performance bottlenecks. Ultimately, monitoring helps to ensure that your applications are running optimally. Monitoring is at the core of the following performance-related activities:

- **Problem identification and diagnosis:** Java EE application designs are often complex. They typically include many interacting layers and use different types of component technologies such as servlets and Enterprise JavaBeans (EJBs), each with their own unique complexities. After deployment, the application runs in an environment that consists of firewalls, web servers, web application servers, databases, and many other necessary systems. This application and runtime architecture presents many areas for performance bottlenecks and single points of failure. You need to monitor each aspect to facilitate problem identification and diagnosis. When a problem occurs, monitoring helps to quickly identify the failing resource and determine its root cause.
- **Performance testing and tuning:** Your application and its runtime environment should be tuned optimally. This process entails conducting many iterations of a monitor-tune-test cycle.

Monitoring helps identify where to tune first (the trouble spots) and validate the results of your tuning changes with each iteration.

- **System Health and Availability Management:** System health and availability management solutions help ensure that resources are running within their established operational limits. When thresholds are exceeded, they can automatically generate alerts and even initiate corrective actions to resolve the problem. Monitoring provides the foundation for this proactive management of applications and resources. It also helps validate compliance with the performance terms defined in applicable service level agreements (SLAs).

Categories of parameters to monitor

- User view
 - Response time
 - Load
 - Throughput
- Why is the system so slow?
-
- System view
 - CPU usage
 - Disk activity
 - Memory usage
 - JVM heap
- Is WebSphere running OK?
-
- Application view
 - Servlet/EJB response time
 - Servlet/EJB concurrent requests
 - Number of live sessions
- Why is the Order EJB consuming so much time?

© Copyright IBM Corporation 2016

Figure 2-59. Categories of parameters to monitor

WA8152.2

Notes:

In an environment with possibly thousands of different parameters, the key to monitoring is to logically partition the environment into different views. Each view looks at the environment from a different perspective, allowing you to drill down on a potential problem to find its root cause.

Unit summary

Having completed this unit, you should be able to:

- Describe performance planning and design tasks that are performed during development
- Explain the value of application profiling, static code analysis, and runtime analysis that is performed during development
- Explain the goals and value of automated performance testing
- Describe the goal of performance monitoring

© Copyright IBM Corporation 2016

Figure 2-60. Unit summary

WA8152.2

Notes:

Checkpoint questions

1. True or False: Performance should not be considered until the application is monitored in a production environment.
2. True or False: Typical goals of performance testing include identifying system response times, determining the maximum number of concurrent users, and discovering optimal configurations.
3. True or False: Response time measures an individual user's wait for a request.
4. True or False: The two types of code analysis are known as static code and runtime analysis.
5. True or False: Load testing evaluates performance at a level well beyond estimated loads.
6. True or False: From the user's view, the most important parameters to monitor are: CPU usage, disk activity, memory usage, and the JVM heap.

© Copyright IBM Corporation 2016

Figure 2-61. Checkpoint questions

WA8152.2

Notes:

Write your answers here:

- 1.
- 2.
- 3.
- 4.
- 5.
- 6.

Checkpoint answers

1. False: Performance needs to be considered in every step from design through deployment.
2. True
3. True
4. True
5. False: Load testing simulates anticipated user load and actual business operations. Stress testing evaluates performance at a level well beyond estimated loads.
6. False: These parameters are from the System's view. For the user monitor: response time, load, and throughput.

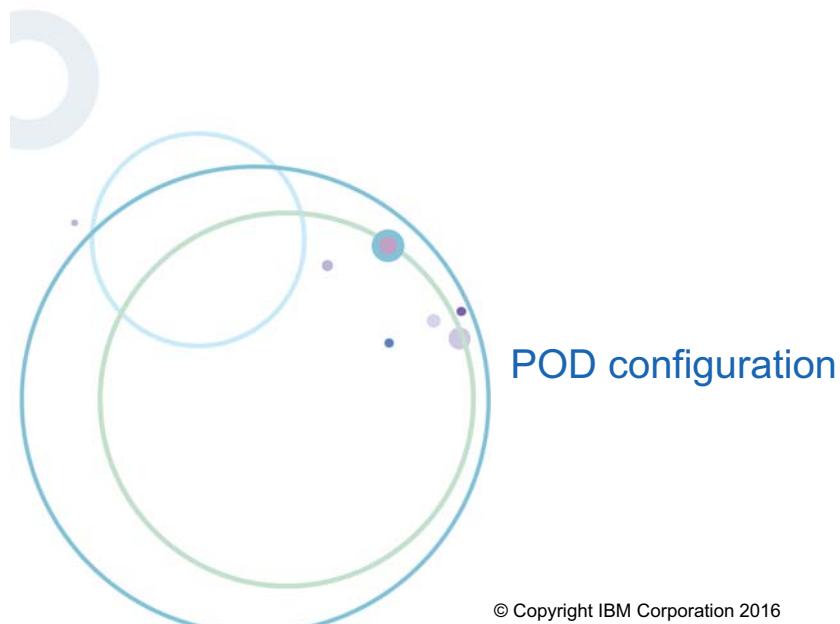
© Copyright IBM Corporation 2016

Figure 2-62. Checkpoint answers

WA8152.2

Notes:

Exercise 1



© Copyright IBM Corporation 2016

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

9.1

Figure 2-63. Exercise 1

WA8152.2

Notes:

Exercise objectives

After completing this exercise, you should be able to:

- Use the POD of three machines for the lab exercises

© Copyright IBM Corporation 2016

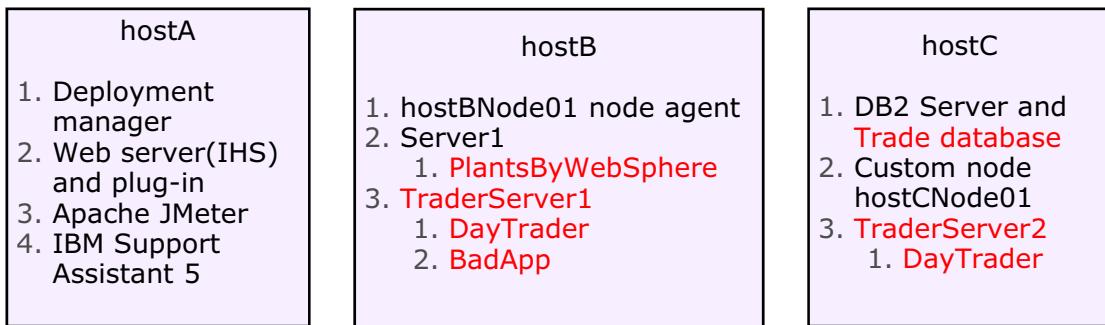
Figure 2-64. Exercise objectives

WA8152.2

Notes:

Exercise instructions

- This exercise covers the setup for your exercise POD. The term “POD” describes the group of three machines that work together for the lab exercises.
 - You use a set of scripts to configure the three machines to work together.
- You create the items in red during later lab exercises



© Copyright IBM Corporation 2016

Figure 2-65. Exercise instructions

WA8152.2

Notes:

Unit 3. Apache JMeter and load testing

What this unit is about

This unit describes how to use Apache JMeter as a load testing tool.

What you should be able to do

After completing this unit, you should be able to:

- Describe the features and functions of Apache JMeter
- Create, run, and analyze load tests by using Apache JMeter
- Run a ramp-up test
- Describe the features of the DayTrader benchmark application

How you will check your progress

- Checkpoint questions
- Lab exercises

References

Apache JMeter User Manual

<http://jmeter.apache.org/usermanual/index.html>

Case study: Tuning WebSphere Application Server V7 and V8 for performance

Performance testing and analysis with WebSphere Application Server, David Hare, http://www.ibm.com/developerworks/websphere/techjournal/1208_hare/1208_hare.html

Unit objectives

After completing this unit, you should be able to:

- Describe the features and functions of Apache JMeter
- Create, run, and analyze load tests by using Apache JMeter
- Run a ramp-up test
- Describe the features of the DayTrader benchmark application

© Copyright IBM Corporation 2016

Figure 3-1. Unit objectives

WA8152.2

Notes:



Topics

- Load testing tools
- The basics of using Apache JMeter
- Introduction to DayTrader

© Copyright IBM Corporation 2016

Figure 3-2. Topics

WA8152.2

Notes:

3.1. Load testing tools

Load testing tools



© Copyright IBM Corporation 2016

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

9.1

Figure 3-3. Load testing tools

WA8152.2

Notes:

Reasons for load testing

- Code often runs well under light loads in the development environment, but it might break when a heavier load is applied
- Exercise the code under a heavy load to:
 - Expose threading bugs
 - Verify that the code meets performance criteria
 - Verify that the code runs for long periods of time with no ill effects
 - Expose possible bottlenecks in the application
 - Help with capacity planning
 - Enable future diagnosis and debug

© Copyright IBM Corporation 2016

Figure 3-4. Reasons for load testing

WA8152.2

Notes:

There are a number of reasons for load-testing your application. Code often runs well under light loads in the development environment, but it might break when a heavier load is applied. You should exercise the code under a heavy load to:

- Expose threading bugs
- Verify that the code meets performance criteria
- Verify that the code runs for long periods of time with no ill effects
- Expose possible bottlenecks in the application
- Help with capacity planning
- Enable future diagnosis and debug

Factors in load testing

- Number of users and transaction rate
 - Normal load versus peak load
- Transaction mix
 - Use different tests to simulate most common transactions
- Data set size and contents
 - Provide realistic user data
- Think time
 - Simulate realistic user think times
- Session handling
 - Test both HTTP and HTTPS sessions
 - Test sessions that use persistent HTTP session objects

© Copyright IBM Corporation 2016

Figure 3-5. Factors in load testing

WA8152.2

Notes:

Stages of load testing

- Verification of the environment
 - No errors in the application?
- Are enough machine resources allocated for testing?
 - Configuration settings correct?
- Warm-up
 - "Cold" machines might perform poorly because data is not in memory or cache
 - Data structures might need to be created
 - Java code needs to be compiled
- Data collection and recording
 - Collect as much data as possible
- Verifying the results
 - Must be reliable and repeatable

© Copyright IBM Corporation 2016

Figure 3-6. Stages of load testing

WA8152.2

Notes:

There are several stages of load testing. First, verify the application server environment and check that there are no errors in the application. Make sure that enough machine resources are allocated for testing and that the configuration settings are correct. Warm up the system and application server since “cold” machines might perform poorly because data is not in memory or cache, data structures might need to be created, and so forth. When you run the test load, collect as much performance data as possible. Finally, verify that the results of the load testing are reliable and repeatable.



Selecting load testing tools

- The primary purpose of load testing tools is to discover under what conditions your application's performance becomes unacceptable
- Some of the points to consider when evaluating load testing tools include:
 - Client interaction and simulation of multiple clients
 - Scripted execution with the ability to edit scripts
 - Session support
 - Configurable numbers of users
 - Reporting: success, errors, and failures
 - Page display and playback, exporting test results
 - Think time
 - Variable data
 - Script recording
 - Analysis tools
 - Load distribution
 - Measuring server-side statistics

© Copyright IBM Corporation 2016

Figure 3-7. Selecting load testing tools

WA8152.2

Notes:

Selecting load testing tools: More details (1 of 4)

- Client interaction
- Simulation of multiple clients
- Think time
- Session support

© Copyright IBM Corporation 2016

Figure 3-8. Selecting load testing tools: More details (1 of 4)

WA8152.2

Notes:

There are many things to consider when selecting test tools.

- **Client interaction:** The load testing tool must be able to handle the features and protocols that your application uses.
- **Simulation of multiple clients:** This capability is the most basic functionality of a stress testing tool.
- **Think time:** Real-world users do not request one web page immediately after another.
- **Session support:** If a stress testing tool does not support sessions or cookies, it is not useful and might not be able to stress test Java and WebSphere applications.



Selecting load testing tools: More details (2 of 4)

- Configurable numbers of users
- Scripted execution with the ability to edit scripts
- Script recording

© Copyright IBM Corporation 2016

Figure 3-9. Selecting load testing tools: More details (2 of 4)

WA8152.2

Notes:

There are several other things to consider when selecting test tools.

- **Configurable numbers of users:** The load testing tool should let you specify how many simulated users are running each script or set of tasks. Many load testing tools enable you to start with a few users and ramp up slowly to a higher numbers of users.
- **Scripted execution with the ability to edit scripts:** If you cannot script the interaction between the client and the server, then you cannot handle anything except the simplest client requests. The ability to edit the scripts is essential.
- **Script recording:** Rather than writing scripts, it is much easier to manually run through a session with your browser and have this session recorded for later editing. Most load testing tools include provisions for capturing manual interaction with your application.

Selecting load testing tools: More details (3 of 4)

- Reporting: success, errors, and failures
- Exporting test results
- Analysis tools

© Copyright IBM Corporation 2016

Figure 3-10. Selecting load testing tools: More details (3 of 4)

WA8152.2

Notes:

Other things to consider when selecting test tools:

- **Reporting: success, errors, and failures:** The tool that you choose must have a defined way to identify a successful interaction, and failure and error conditions.
- **Exporting test results:** You might want to be able to analyze the test results by using various tools that are external to the load testing tool, including spreadsheets and custom analysis scripts. Most load testing tools include extensive built-in analysis functions, but being able to export the data gives you more flexibility to analyze and catalog the data in arbitrary ways.
- **Analysis tools:** Measuring performance is only half the story. The other, and more important, half of load testing is analyzing the performance data. The type of analysis and degree of detailed analysis you can perform depends directly on what analysis tools you select.



Selecting load testing tools: More details (4 of 4)

- Load distribution
- Measuring server-side statistics

© Copyright IBM Corporation 2016

Figure 3-11. Selecting load testing tools: More details (4 of 4)

WA8152.2

Notes:

And the final considerations for considering a testing tool are:

- **Load distribution:** Your deployed application might need to support hundreds of concurrent users once in production. A typical workstation that runs a stress testing tool is likely to begin to create a bottleneck when approximately 200 virtual users are running. To simulate a greater number of users, you can distribute the stress testing load across multiple workstations if your tool supports this feature.
- **Measuring server-side statistics:** The basic stress testing tool measurement is client-based response times from client/server interactions. You might also want to gather other statistics, such as the CPU utilization or page faulting rates. With this server-side data, you can then view client response times in the context of server load and throughput statistics.

3.2. The basics of using Apache JMeter

The basics of using Apache JMeter



© Copyright IBM Corporation 2016

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

9.1

Figure 3-12. The basics of using Apache JMeter

WA8152.2

Notes:

What is Apache JMeter?

- A load testing tool from Apache group, Jakarta project
- Apache JMeter can test performance on both static and dynamic resources (files, servlets, Java objects, databases, and queries)
 - It can be used to simulate a heavy load on a server to test its strength or to analyze performance under different loading conditions
 - You can create a graphical analysis of performance or test your server's behavior under heavy concurrent user loads
- Scenarios can be saved and reloaded
 - Modular, customizable
- Many tools
 - Samplers
 - Listener
 - Graphical visualizers
 - Cookie support
 - Timers

© Copyright IBM Corporation 2016

Figure 3-13. What is Apache JMeter?

WA8152.2

Notes:

Apache JMeter is open source software, a 100% pure Java desktop application that is designed to load test functional behavior and measure performance. It was originally designed for testing web applications but expanded to other test functions. It is available from <http://jmeter.apache.org>

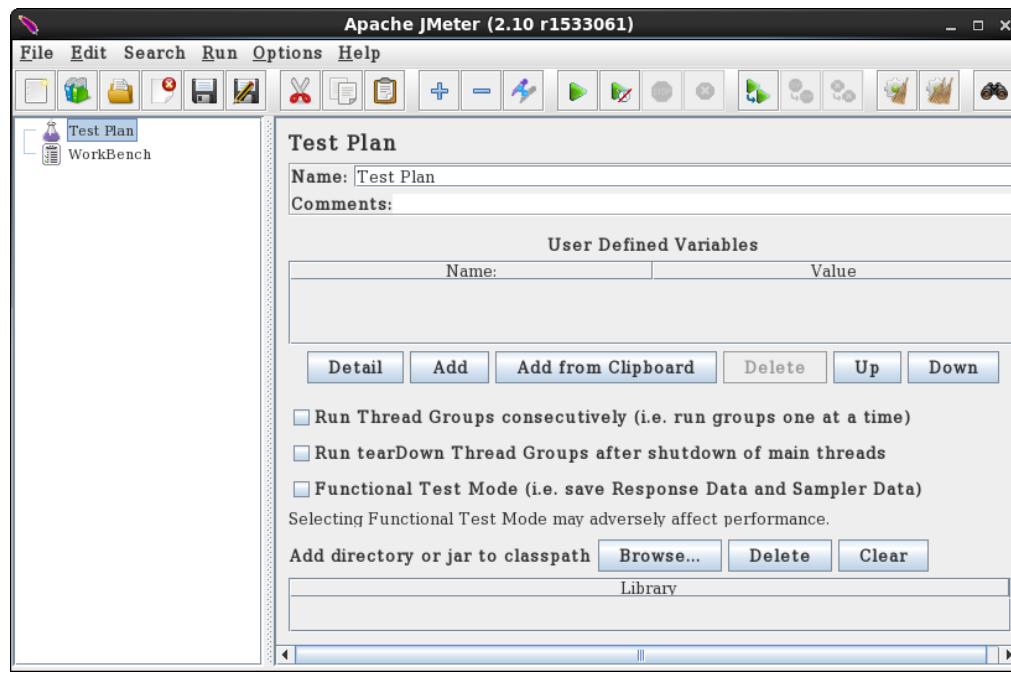
Apache JMeter features include:

- Ability to load and performance test many different server types:
 - Web: HTTP, HTTPS
 - SOAP
 - Database via JDBC
 - LDAP
 - JMS
 - Mail: POP3(S) and IMAP(S)
- Complete portability and **100% Java purity**

- Full **multithreading** framework allows concurrent sampling by many threads and simultaneous sampling of different functions by separate thread groups
- Careful **GUI** design allows faster operation and more precise timings
- Caching and offline analysis, replaying of test results
- **Highly extensible:**
 - Pluggable samplers allow unlimited testing capabilities
 - Several load statistics can be chosen with **pluggable timers**
 - Data analysis and **visualization plug-ins** allow great extensibility and personalization
 - Functions can be used to provide dynamic input to a test or provide data manipulation
 - Scriptable samplers (Bean Shell is fully supported; and a sampler is available that supports BSF-compatible languages)

JMeter Test Plan view

- Start JMeter from the command line by using `jmeter.sh` or `jmeter.bat`



© Copyright IBM Corporation 2016

Figure 3-14. JMeter Test Plan view

WA8152.2

Notes:

You can provide a more descriptive name for the test plan that shows up in the tree view.

A test plan describes a series of steps that JMeter does when the test is run. A test plan can consist of one or more thread groups, logic controllers, sample generating controllers, listeners, timers, assertions, and configuration elements.

The workbench is a place to temporarily store test elements that you are not using, for copy and paste purposes. When you save the test plan, the workbench is not saved with it, but you can save it independently. To save the workbench, right-click workbench and click **Save**.



Thread Group view: Create a thread group

- Right-click **Test Plan** and select **Add > Threads (Users) > Thread Group**
 - Define the number of users (threads) to drive the load, ramp-up period, and loop count (number of times to run the test)

The screenshot shows the 'Thread Group' configuration dialog. It includes fields for 'Name' (Thread Group), 'Comments', and 'Action to be taken after a Sampler error' (with 'Continue' selected). Under 'Thread Properties', there are fields for 'Number of Threads (users)' (40), 'Ramp-Up Period (in seconds)' (1), 'Loop Count' (checkboxes for 'Forever' (selected) and 'Scheduler'), and checkboxes for 'Delay Thread creation until needed' and 'Scheduler'.

© Copyright IBM Corporation 2016

Figure 3-15. Thread Group view: Create a thread group

WA8152.2

Notes:

You can provide a more descriptive name for the thread group that shows up in the tree view.

Begin developing a test plan by defining one or more Thread group elements. Controllers and samplers must be under a thread group. Other elements, such as Listeners, can be placed directly under the test plan, in which case they apply to all the thread groups. The thread group element controls the number of threads JMeter uses to execute your test. The controls for a thread group allow you to define:

- Number of threads (users)
- Ramp-up period
- Loop count (number of times to run the test)

Each thread executes the complete test plan independently of other test threads. Multiple threads are used to simulate concurrent connections to your server application.



Add an HTTP Request sampler

- Right-click the thread group and select **Add > Sampler > HTTP Request**
 - Enter server name, port, and path

The screenshot shows the 'HTTP Request' configuration dialog in JMeter. Key settings visible include:

- Name:** HTTP Request
- Server Name or IP:** hostB
- Port Number:** 9080
- Implementation:** dropdown menu
- Protocol:** http
- Method:** GET
- Path:** /snoop
- Checkboxes:** Redirect Automatically, Follow Redirects, Use KeepAlive, Use multipart/form-data for POST, Browser-compatible headers
- Buttons:** Detail, Add, Add from Clipboard, Delete, Up, Down

- A sampler defines the type of load to drive: HTTP, FTP, JDBC, and so on

© Copyright IBM Corporation 2016

Figure 3-16. Add an HTTP Request sampler

WA8152.2

Notes:

A sampler tells JMeter to send requests to a server and wait for a response. They are processed in the order that they appear in the tree. Controllers can be used to modify the number of repetitions of a sampler.

JMeter samplers include:

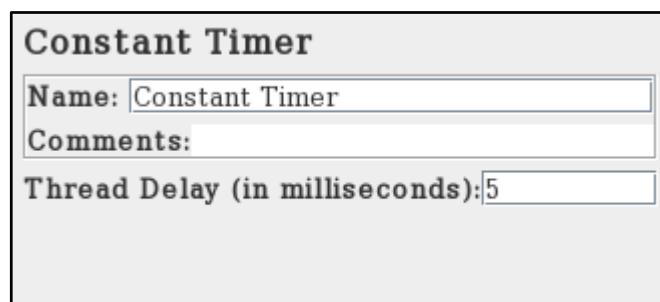
- FTP Request
- HTTP Request
- JDBC Request
- Java object request
- LDAP Request
- SOAP/XML-RPC Request
- WebService (SOAP) Request

You can use the HTTP Request sampler to send an HTTP/HTTPS request to a web server. You can control whether JMeter parses HTML files for images and other embedded resources and sends HTTP requests to retrieve them. The following types of embedded resource are retrieved:

- images
- applets
- Style sheets
- External scripts
- Frames, iframes
- Background images
- Background sound

Add a timer

- Right-click the **HTTP Request** and select **Add > Timer > Constant Timer**
- A timer adds “think time” between requests
- Simulates a user who clicks a page, and then pauses to read some information on the page, before making another request
 - JMeter provides several pre-defined timers



- Use the Constant Timer to have each thread pause for the same amount of time between requests

© Copyright IBM Corporation 2016

Figure 3-17. Add a timer

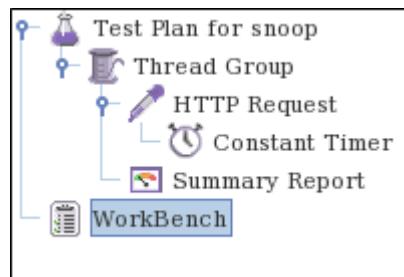
WA8152.2

Notes:

Right-click the HTTP request that you created, and click **Add > Timer**. A timer adds “think time” in between requests. This action simulates a user who clicks a page, and then pauses to read some information on the page, before making another request. Many predefined timers are available that you can choose from that range in complexity from a constant fixed timer to a Gaussian or Poisson distributed timer. The JMeter user manual documents each of the available timers.

Add a Summary Report listener

- Right-click the thread group and select **Add > Listener > Summary Report**
- The Summary Report shows the response times, throughput, and other statistics while the test is running
- The elements of the simple test plan can be expanded and viewed in the tree



© Copyright IBM Corporation 2016

Figure 3-18. Add a Summary Report listener

WA8152.2

Notes:

Most of the listeners do several roles besides “listening” to the test results. They also provide the means to view, save, and read saved test results. Listeners are processed at the end of the scope in which they are defined.

Listeners can use much memory if there are many samples. Some listeners include:

- Simple Data Writer
- Summary Report



Summary Report view

- You can run the test plan from the Summary Report view
 - Click green arrow to start the test, click red stop sign to stop

Summary Report

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	KB/sec	Avg. Bytes
Thread Group:HTTP Request	8000	730	30	1101	96.21	0.00%	106.9/sec	1121.49	10746.6
TOTAL	8000	730	30	1101	96.21	0.00%	106.9/sec	1121.49	10746.6

- Performance metrics
 - **Average:** Average response time in milliseconds
 - **Throughput:** Number of requests per second
- Statistics
 - **Min:** Minimum response time; **Max:** Maximum response time
 - **Std. Dev:** Standard deviation of the average response time

© Copyright IBM Corporation 2016

Figure 3-19. Summary Report view

WA8152.2

Notes:

The summary report creates a table row for each uniquely named request in your test.

The throughput is calculated from the point of view of the sampler target (For example, the remote server in the case of HTTP samples). JMeter takes into account the total time over which the requests are generated. If other samplers and timers are in the same thread, these elements increase the total time, and therefore reduce the throughput value. So two identical samplers with different names have half the throughput of two samplers with the same name. It is important to choose the sampler labels correctly to get the best results from the Report.

Running the test

- Always make sure the application works through a web browser before starting the load-driving tool
 - When ready, click the green arrow or click **Run > Start**
- JMeter now drives the number of users that you specified, to the server path that you used, and waits for the specified thread delay between each request
 - If you click **Summary Report** in the tree, you can view the results as the test runs
- You typically notice that the throughput increases over time
 - This length of time is called the “warm-up” period
- The response time and throughput eventually stabilize over time and reach a nearly fixed value

© Copyright IBM Corporation 2016

Figure 3-20. Running the test

WA8152.2

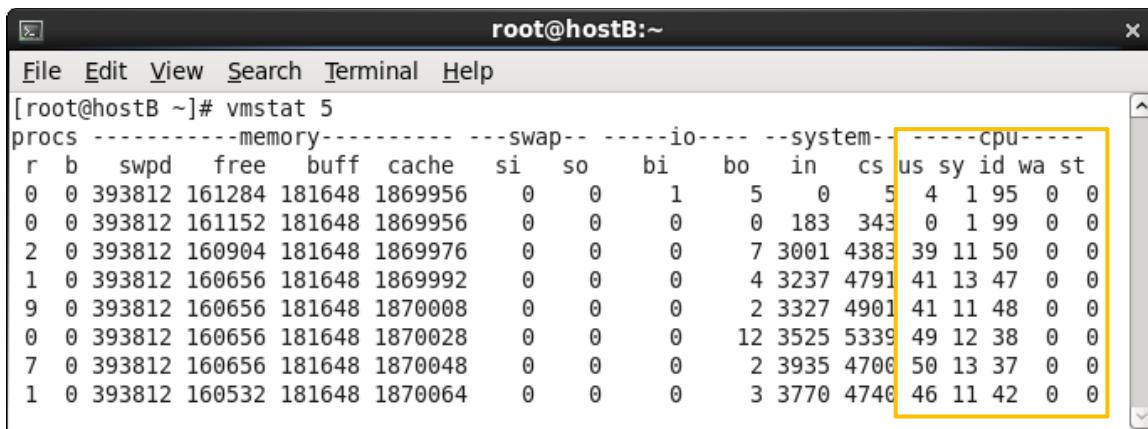
Notes:

You typically notice that the throughput increases over time; this interval of time is called the “warm-up” period. The JRE needs some warm-up time to load all the Java classes and let the JIT compiler make some optimizations by compiling methods. The response time and throughput eventually stabilize and reach a nearly fixed value.

Depending on how complex your test is, this warm-up period might be ~30 seconds or ~30 minutes.

Monitor system metrics

- As the test runs, open a terminal where the application runs, and use the command `vmstat 5` to display system metrics every 5 seconds



```
root@hostB:~#
File Edit View Search Terminal Help
[root@hostB ~]# vmstat 5
procs -----memory----- swap-- io--- system--cpu-----
r b swpd free buff cache si so bi bo in cs us sy id wa st
0 0 393812 161284 181648 1869956 0 0 1 5 0 5 4 1 95 0 0
0 0 393812 161152 181648 1869956 0 0 0 0 183 343 0 1 99 0 0
2 0 393812 160904 181648 1869976 0 0 0 7 3001 4383 39 11 50 0 0
1 0 393812 160656 181648 1869992 0 0 0 4 3237 4791 41 13 47 0 0
9 0 393812 160656 181648 1870008 0 0 0 2 3327 4901 41 11 48 0 0
0 0 393812 160656 181648 1870028 0 0 0 12 3525 5339 49 12 38 0 0
7 0 393812 160656 181648 1870048 0 0 0 2 3935 4706 50 13 37 0 0
1 0 393812 160532 181648 1870064 0 0 0 3 3770 4746 46 11 42 0 0
```

- Focus on the cpu usage metrics: us (user), sy (system), id (idle)
- Typically the “us” value shows how much cpu the application is using, and is usually larger than “sy” (user + system = total cpu usage)
- For the period monitored, the average cpu usage is 55%

© Copyright IBM Corporation 2016

Figure 3-21. Monitor system metrics

WA8152.2

Notes:

On Windows, right-click the taskbar, click **Task Manager**, and click the **Performance** tab.

Use a ramp-up test to determine the saturation point

1. Conduct a ramp-up test
 - Run the test plan first with one user
 - Double the number of users, and continue running the test plan
 - Monitor application server CPU usage and database usage
2. Record results of each test that is run in a table or spread sheet
 - Record: Number of users, throughput, response time, and CPU usage
3. Plot the results in graph
 - Throughput and application server CPU versus number of users
4. Find the point where increasing the number of users no longer increases the throughput
 - This point is the saturation point
 - Beyond the saturation point, adding more users increases only the response time

The saturation point shows you that you reached the maximum capacity for an application with the current tuning, configuration, and environment

© Copyright IBM Corporation 2016

Figure 3-22. Use a ramp-up test to determine the saturation point

WA8152.2

Notes:

The saturation point shows that you reached your maximum capacity for this application, tuning, configuration, and environment. Adding more users beyond this point increases only client response times, but does not increase the overall application throughput. To achieve better performance beyond this point, an application code change, tuning change, or environmental change must occur.

Record test results example

No of users	Throughput (req/sec)	Response time (ms)	WAS %CPU	DB2 %CPU
1	10	2	1	0
2	24	2	1	0
4	44	2	1	0
8	86	2	2	0
16	162	3	3	0
32	315	4	9	1
64	624	6	12	1
125	1208	10	23	3
250	2366	13	45	5
500	4613	23	82	9
1000	5559	32	99	11
1500	5587	34	99	11
2000	5565	34	99	11

© Copyright IBM Corporation 2016

Figure 3-23. Record test results example

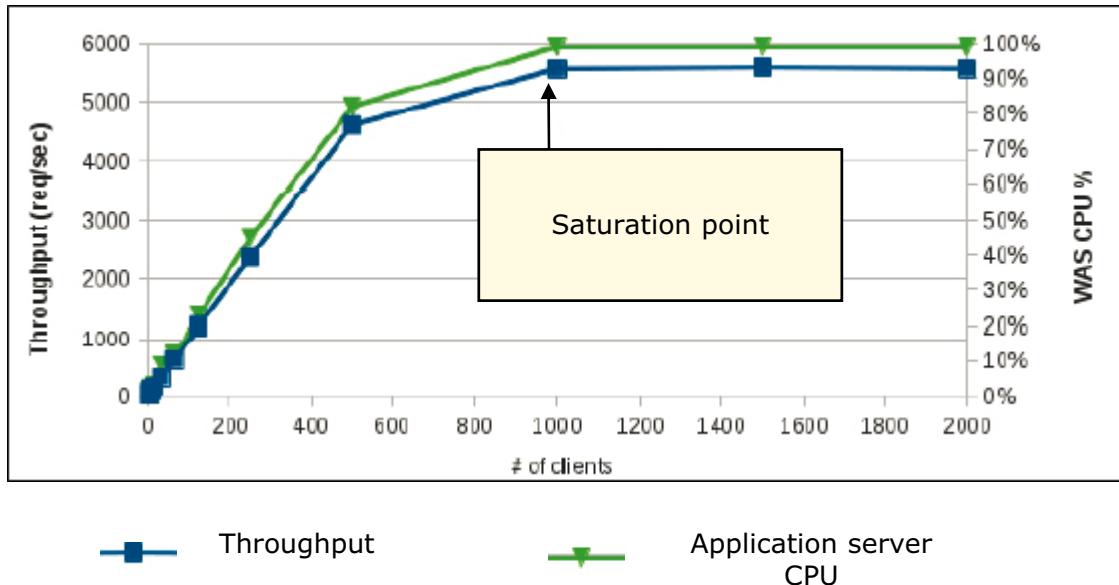
WA8152.2

Notes:

After completing the steps described, you might have a spreadsheet that looks similar to the one shown on this slide. These results depend on the DayTrader application and the environment in which this test was run. Your actual results might look different.

Graph: Ramp-up test results

- The throughput curve and % CPU usage curve are a close match
- Application throughput scales linearly from one client up to about 500 clients



© Copyright IBM Corporation 2016

Figure 3-24. Graph: Ramp-up test results

WA8152.2

Notes:

The throughput curve and CPU % curve match closely together. You see that the application throughput scales linearly from one client up to about 500 clients. This behavior is the result that is required. However, somewhere in between 500 clients and 1,000 clients, the increase in throughput starts to slow down. (More tests can be done with user loads in 500 - 1,000 to find out exactly where this slowdown occurs.) Increasing the client load beyond 1,000 clients does not improve your overall application throughput. This point is called the saturation point.

Using the ramp-up test for further testing

- Save a new test in JMeter with the number of users (threads) that you found as your saturation point
 - You might call this new test the baseline test
- Use the baseline as a quick way to compare performance as you change your application or environment, without going through the entire scalability test again
- Ideally, you should run the scalability test each time you change something that might affect performance
 - But running a test load at the saturation point is a good way to compare changes that might not show any difference at lower loads where the CPU is not fully used
- A good practice is to:
 - Run the saturation point load for minor application or tuning changes
 - Repeat the entire scalability test for major application or environment changes

© Copyright IBM Corporation 2016

Figure 3-25. Using the ramp-up test for further testing

WA8152.2

Notes:

This type of testing and analysis is important for sizing and capacity planning. By identifying the saturation point, you can accurately estimate the total capacity that is needed to support a production environment. Too often, someone might say, "I need to support 10,000 users in this environment" and then run tests with that client load. Typically, this approach leads to various overloaded conditions in one or more components, either in WebSphere Application Server or in other infrastructure components (network, database, and so on). A more productive approach is to determine what is achievable in terms of client load and throughput with a single application server and then proceed with run time and application tuning based on these results. When tuning is complete, you can then turn your attention to determining how many application server processes and physical servers are required to satisfy the scalability requirements.

3.3. Introduction to DayTrader

Introduction to DayTrader



© Copyright IBM Corporation 2016

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

9.1

Figure 3-26. Introduction to DayTrader

WA8152.2

Notes:



DayTrader benchmark application (1 of 3)

- The Apache DayTrader Performance Benchmark Sample Application simulates a simple stock trading application that allows users to:
 - Login and logout
 - View a portfolio
 - Look up stock quotes
 - Buy and sell stock shares
 - Manage account information
- An excellent application for functional testing
- Provides a standard set of workloads for characterizing and measuring application server and component level performance

© Copyright IBM Corporation 2016

Figure 3-27. DayTrader benchmark application (1 of 3)

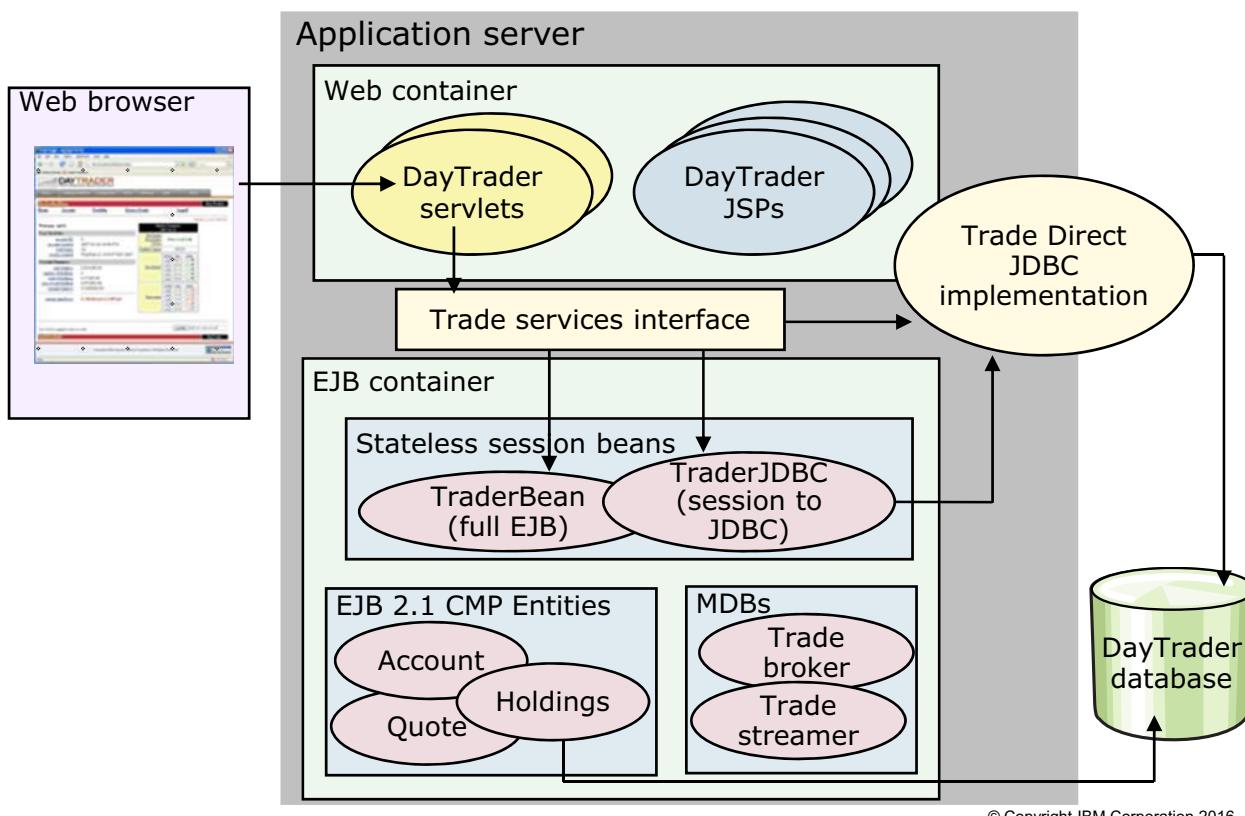
WA8152.2

Notes:

Originally based on the Trade Performance Benchmark Sample Application for WebSphere Application Server, the application was donated to the Apache Geronimo open source community with IBM's delivery of WebSphere Application Server Community Edition in 2006. DayTrader not only serves as an excellent application for functional testing, but also provides a standard set of workloads for characterizing and measuring application server and component level performance.

DayTrader was not written to provide optimal performance. Instead, it was meant to conduct relative performance comparisons between application server releases and alternative implementation styles and patterns.

DayTrader benchmark application (2 of 3)



© Copyright IBM Corporation 2016

Figure 3-28. DayTrader benchmark application (2 of 3)

WA8152.2

Notes:

DayTrader is built on a core set of Java EE technologies. These technologies include Java servlets and JavaServer Pages (JSPs) for the presentation layer. For the back-end business logic and persistence layer, the following technologies are used: Java database Connectivity (JDBC), Java Message Service (JMS), Enterprise JavaBeans (EJBs), and message-driven beans (MDBs). This graphic provides a high-level overview of the application architecture.

DayTrader benchmark application (3 of 3)

- In order to evaluate common Java EE persistence and transaction management patterns, DayTrader provides three implementations of the business services

Runtime mode	Pattern	Description
Direct	Servlet-to-JDBC	Create, read, update, and delete operations are performed directly against the database by using custom JDBC code. Connections, commits, and rollbacks are managed manually in the code.
Session direct	Servlet-to-stateless session bean -to-JDBC	Create, read, update, and delete operations are performed directly against the database by using custom JDBC code. Connections are managed manually in the code. The stateless session bean automatically manages commits and rollbacks.
EJB	Servlet-to-stateless session bean -to-entity bean	The EJB container assumes responsibility for all queries, transactions, and connections.

© Copyright IBM Corporation 2016

Figure 3-29. DayTrader benchmark application (3 of 3)

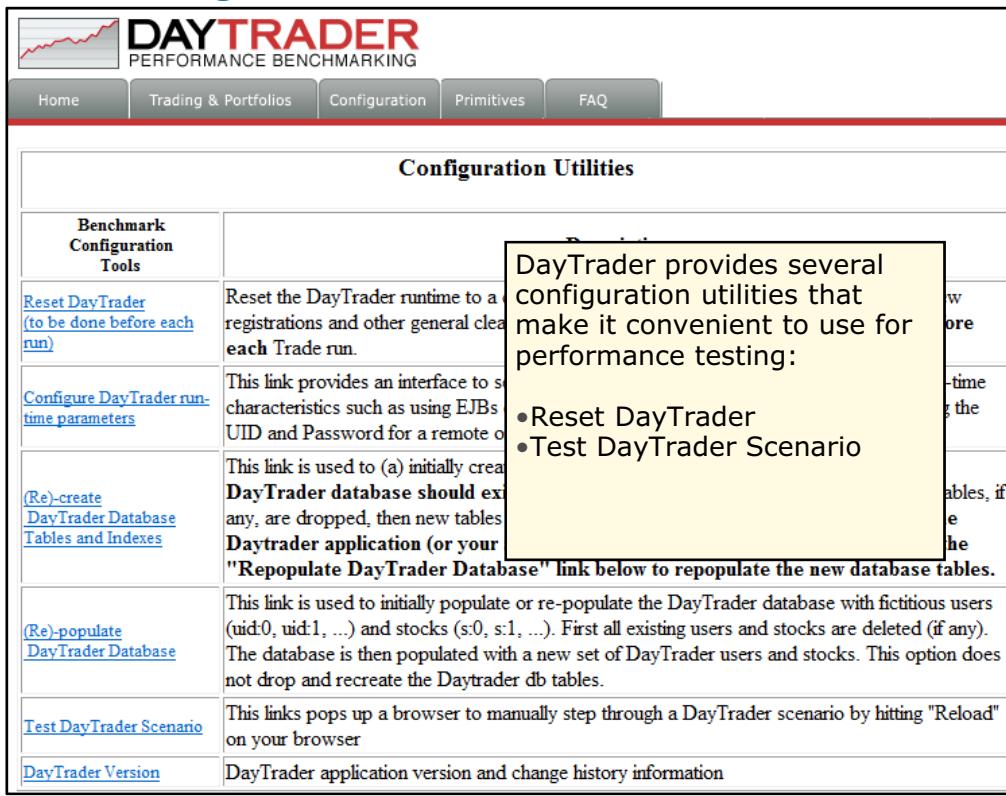
WA8152.2

Notes:

These runtime modes demonstrate the impact each tuning change has on these three common Java EE persistence and transaction implementation styles.




DayTrader configuration



Configuration Utilities

Benchmark Configuration Tools	
Reset DayTrader (to be done before each run)	Reset the DayTrader runtime to a clean starting point by logging off all users, removing new registrations, and other general cleanup. For consistent results, this URL should be run before each Trade run.
Configure DayTrader runtime parameters	This link provides an interface to set DayTrader runtime characteristics such as using EJBs, UID and Password for a remote connection.
(Re)-create DayTrader Database Tables and Indexes	This link is used to (a) initially create the DayTrader database tables if they do not exist, or (b) drop any existing tables and recreate them. The Daytrader application (or your own application) must be stopped before running this link. If you have a new set of tables, click the "Repopulate DayTrader Database" link below to repopulate the new database tables.
(Re)-populate DayTrader Database	This link is used to initially populate or re-populate the DayTrader database with fictitious users (uid0, uid1, ...) and stocks (s:0, s:1, ...). First all existing users and stocks are deleted (if any). The database is then populated with a new set of DayTrader users and stocks. This option does not drop and recreate the Daytrader db tables.
Test DayTrader Scenario	This link pops up a browser to manually step through a DayTrader scenario by hitting "Reload" on your browser.
DayTrader Version	DayTrader application version and change history information

© Copyright IBM Corporation 2016

Figure 3-30. DayTrader configuration

WA8152.2

Notes:

One of the important features of DayTrader is the **TestTradeScenario** link under the **Configuration** tab. This link invokes a servlet that emulates a population of web users by randomly generating a specific DayTrader operation for each user that accesses the servlet. For example, one user might view their portfolio, one user might perform a stock buy operation, and one user might look up a stock quote. This behavior ensures that each of the main operations in DayTrader is executed during the test, and over time. Because it is random, each operation is executed approximately the same number of times.

The Reset DayTrader utility resets the DayTrader run time to a clean starting point by logging off all users, removing new registrations, and other general cleanup. For consistent results, this URL should be run before each Trade run.

Using the configure DayTrader runtime parameters utility, you can choose which of the runtime modes to use: EJB, Session Direct, or Direct.

DayTrader compared to your application

- TestTradeScenario link invokes a servlet that emulates a population of web users by randomly generating a specific DayTrader operation for each user that accesses the servlet
 - For example, one user might view their portfolio, one might perform a stock buy operation, one might look up a stock quote
 - This behavior ensures that each of the main operations in DayTrader is executed during the test, and over time
 - Because it is random, each operation is executed approximately the same number of times
- DayTrader is not representative of all applications
- Your application might reach the saturation point with less concurrent users
 - If so, you might be experiencing an application bottleneck, which requires an analysis of the application to eliminate

© Copyright IBM Corporation 2016

Figure 3-31. DayTrader compared to your application

WA8152.2

Notes:

The type of testing and analysis that is described in this unit is important in sizing and capacity planning activities. By identifying the saturation point, you can accurately estimate the total capacity that is needed to support a production environment. With this approach, you can determine what is achievable in terms of client load and throughput with a single application server and then proceed with tuning the run time and application that is based on this information. When tuning is complete, you can then turn your attention to determining how many application server processes and physical servers are required to satisfy the scalability requirements.



Unit summary

Having completed this unit, you should be able to:

- Describe the features and functions of Apache JMeter
- Create, run, and analyze load tests by using Apache JMeter
- Run a ramp-up test
- Describe the features of the DayTrader benchmark application

© Copyright IBM Corporation 2016

Figure 3-32. Unit summary

WA8152.2

Notes:



Checkpoint questions

1. True or False: The first element to define in a JMeter test plan is a thread group.
2. True or False: Every JMeter test plan must define an HTTP Request sampler.
3. True or False: After the saturation point of an application server is reached, throughput increases.
4. True or False: The DayTrader benchmark simulates a simple stock trading application.

© Copyright IBM Corporation 2016

Figure 3-33. Checkpoint questions

WA8152.2

Notes:

Write your answers here:

- 1.
- 2.
- 3.
- 4.



Checkpoint answers

1. True
2. False: A JMeter test plan can use any of a number of pre-defined samplers.
3. False: After the saturation point of an application server is reached, throughput no longer increases.
4. True

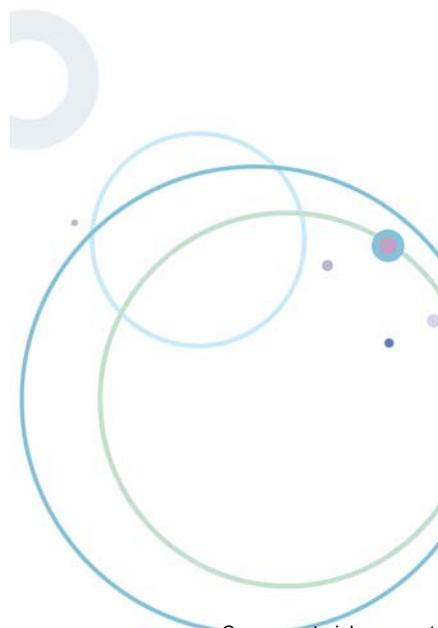
© Copyright IBM Corporation 2016

Figure 3-34. Checkpoint answers

WA8152.2

Notes:

Exercise 2



© Copyright IBM Corporation 2016

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

9.1

Figure 3-35. Exercise 2

WA8152.2

Notes:

Exercise objectives

After completing this exercise, you should be able to:

- Start Apache JMeter
- Create a simple test plan
- Run the test plan and monitor application performance
- Use JMeter to record a test script by navigating a remote website

© Copyright IBM Corporation 2016

Figure 3-36. Exercise objectives

WA8152.2

Notes:

Exercise 3

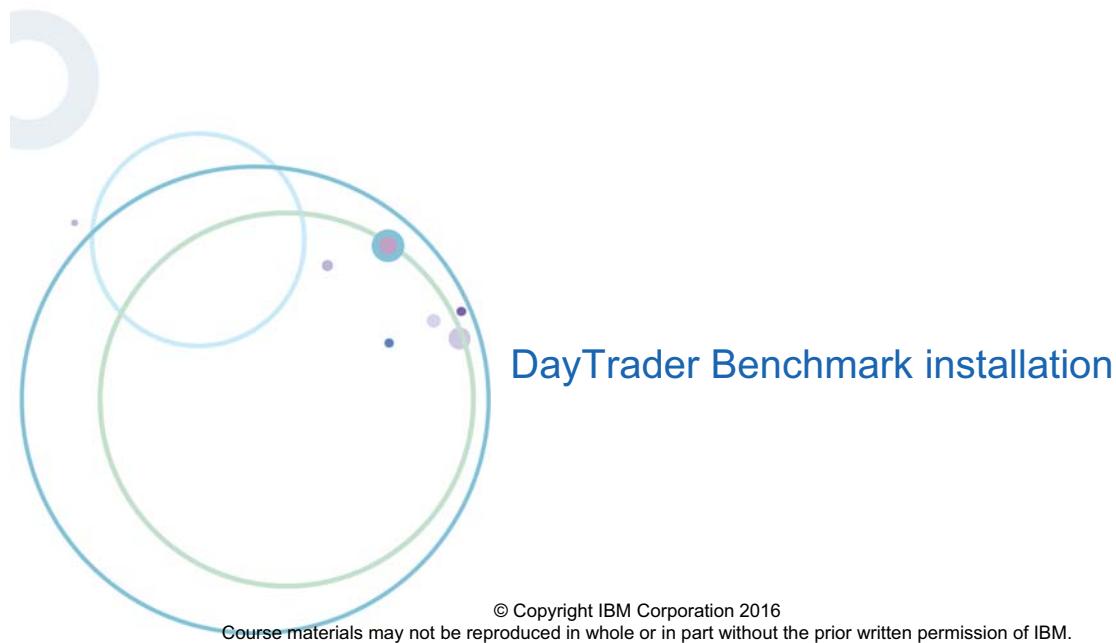


Figure 3-37. Exercise 3

WA8152.2

Notes:



Exercise objectives

After completing this exercise, you should be able to:

- Set up the WebSphere Application Server ND environment
- Install and configure the DayTrader application
- Test and explore the DayTrader application

© Copyright IBM Corporation 2016

Figure 3-38. Exercise objectives

WA8152.2

Notes:

Exercise 4



Using Apache JMeter to load test
DayTrader

© Copyright IBM Corporation 2016
Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Figure 3-39. Exercise 4

WA8152.2

Notes:



Exercise objectives

After completing this exercise, you should be able to:

- Tune and back up the DayTrader database
- Run a ramp-up test on DayTrader to find the saturation point
- Monitor CPU usage on the application server host and the database server host
- Establish a baseline of performance for the DayTrader application server

© Copyright IBM Corporation 2016

Figure 3-40. Exercise objectives

WA8152.2

Notes:

Unit 4. WebSphere performance data and tools

What this unit is about

This unit describes performance data and performance data analysis tools.

What you should be able to do

After completing this unit, you should be able to:

- Describe important performance data for monitoring WebSphere based applications
- Identify useful performance tools for monitoring performance data that describes the behavior of WebSphere based applications

How you will check your progress

- Checkpoint questions
- Machine Exercise

References

WebSphere Application Server V7 Network Deployment Information Center,
Performance Tuning article,

<http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/welc6toptuning.html>

Unit objectives

After completing this unit, you should be able to:

- Describe important performance data for monitoring WebSphere based applications
- Identify useful performance tools for monitoring performance data that describes the behavior of WebSphere based applications

© Copyright IBM Corporation 2016

Figure 4-1. Unit objectives

WA8152.2

Notes:



Topics

- WebSphere performance data
- WebSphere performance tools

© Copyright IBM Corporation 2016

Figure 4-2. Topics

WA8152.2

Notes:

4.1. WebSphere performance data

WebSphere performance data



© Copyright IBM Corporation 2016

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

9.1

Figure 4-3. WebSphere performance data

WA8152.2

Notes:



High-level performance data areas

- Hardware
- Operating system
- JVM
- Web servers
- Application servers
- Messaging
- Databases

© Copyright IBM Corporation 2016

Figure 4-4. High-level performance data areas

WA8152.2

Notes:

The areas of performance data can be mapped against the layers that exist in any Java EE application server-based application. Obviously, any software must run on the host hardware, and the hardware must have an operating system. The WebSphere Application Server is primarily written in Java, and Java must run on a Java virtual machine (JVM). Java EE applications are divided into two tiers: a web tier that serves static content and an application server tier that serves dynamic and customized content. Finally, Java EE applications access data from database servers and might use messaging to pass data between application components.

Hardware performance data

- Number of physical processors
- Amount of physical memory
- Total disk space and free disk space
- I/O channel utilization
- Network interface card utilization and configuration

© Copyright IBM Corporation 2016

Figure 4-5. Hardware performance data

WA8152.2

Notes:

Performance data at the hardware level needs to include configuration and usage levels of all hardware components. These components include the number of processors, the amount of physical memory, the amount, and usage level of disk space, the usage level of I/O channels, and the number and usage level of network interface cards.

Operating system performance data

- System configuration
 - Logical processors per physical processor
- CPU utilization
 - Whole partition
 - Per process
 - Per logical CPU
- Physical memory utilization
 - Paging
 - Swap space utilization

© Copyright IBM Corporation 2016

Figure 4-6. Operating system performance data

WA8152.2

Notes:

Performance data at the operating system level needs to include configuration and usage levels of all operating system level components. These components include the number of logical processors per physical processor; processor usage at multiple levels, including “whole partition”, per process, and per logical processor. Other components are physical memory usage levels; and virtual memory usage levels, including swap space usage levels.

JVM performance data

- Heap utilization as in current heap size versus maximum heap size
 - Current nursery size versus maximum nursery size
 - Current tenured size versus maximum tenured size
- Live object utilization
 - Current size of live versus current heap size
 - Current nursery that is used versus current nursery size
 - Recent minimum tenured used versus current tenured size
- Garbage collection (GC) cost
 - Time in GC versus time between GCs
- Level of heap fragmentation, as in “free before GC”
- Average and peak size of requested objects

© Copyright IBM Corporation 2016

Figure 4-7. JVM performance data

WA8152.2

Notes:

Performance data at the Java virtual machine (JVM) level needs to include information on the configuration of the JVM, particularly in the area of heap size parameters. This data also needs to include statistics about the behavior of JVM garbage collection. Configuration data needs to include garbage collection policy, minimum and maximum heap size, minimum and maximum nursery generation size, and minimum and maximum tenured generation size. JVM garbage collection data needs to include the following statistics:

- Current heap size
- Current nursery size and usage
- Current tenured size and usage
- Time in garbage collection
- Time between garbage collections
- Level of heap fragmentation
- Size of requested object

Sources of JVM performance data

- JVM configuration, such as maximum heap size, are Java command-line arguments
 - For example: `-Xmx512m`
- Statistics regarding garbage collection behavior are available via verbose garbage collection output

```
<gc type="scavenger" id="1" totalid="1" intervalms="0.000">
  <flipped objectcount="20386" bytes="1146496" />
  <tenured objectcount="0" bytes="0" />
  <finalization objectsqueued="16" />
  <scavenger tiltratio="50" />
  <nursery freebytes="5321632" totalbytes="6553600" percent="81" tenureage="10" />
  <tenured freebytes="38817912" totalbytes="39321600" percent="98" >
    <soa freebytes="36851832" totalbytes="37355520" percent="98" />
    <loa freebytes="1966080" totalbytes="1966080" percent="100" />
  </tenured>
  <time totalms="7.753" />
</gc>
```

- Graphing verbose garbage collection output is the best way to view garbage collection behavior

© Copyright IBM Corporation 2016

Figure 4-8. Sources of JVM performance data

WA8152.2

Notes:

JVM performance data about the configuration of the JVM can be obtained by examining the command line parameters that are used to start the JVM. Statistics regarding garbage collection behavior can be obtained by enabling the JVM's verbose garbage collection facility and by examining the verbose garbage collection logs produced during application operation.



Web server performance data

- History of incoming HTTP requests
 - Size of request
 - Time of request
 - Time to respond
 - URL of target server and content
- Source of web server performance data
 - Web server access log

© Copyright IBM Corporation 2016

Figure 4-9. Web server performance data

WA8152.2

Notes:

Requests typically arrive at Java EE applications in HTTP requests. Data regarding these requests is critical to monitoring the behavior of a Java EE application. The data that represents the behavior of the application include:

- Size of request
- Time of request
- Time to respond
- URL of target server and content

Data describing these behaviors are collected in the web server access log.



Application server performance data

- Data source utilization levels
 - Current number of database connections per data source versus maximum number of database connections per data source
- Caching
 - Maximum number of cache entries
 - Cache hit rate
- Thread pool utilization levels
 - Current number of active threads per thread pool versus maximum number of threads per thread pool
- Source of application server performance data
 - Specialized tools are required to monitor data sources, caching, and thread pools
 - You can use the Tivoli Performance Viewer

© Copyright IBM Corporation 2016

Figure 4-10. Application server performance data

WA8152.2

Notes:

Dynamic content is typically served from the application server layer of a two-tier Java EE application. The following application server performance data is important in monitoring the behavior of a Java EE application:

- Data source usage levels to databases
- Cache behavior and usage levels
- Thread pool usage levels

Specialized monitoring tools are required to monitor these application server performance statistics.

Messaging performance data

- Names of queues
- Location (server) for queues
- Queue utilization levels
 - Current queue depth versus maximum queue depth
- Average and peak message sizes
- Current and peak queue depths
 - Queue depth trends
- Message arrival rates
- Sources of messaging performance data
 - Administrative console for WebSphere Default Messaging (System Integration Bus)
 - Tivoli Performance Viewer (SIB service)
 - Specialized tools for other messaging providers such as WebSphere MQ

© Copyright IBM Corporation 2016

Figure 4-11. Messaging performance data

WA8152.2

Notes:

Some Java EE applications are called by using messaging-oriented bindings. Some Java EE applications contact other applications and services by using message-based bindings. Some Java EE applications communicate internally by using queue-based bindings.

The following data is important to monitor the behavior of Java EE applications that use queue-based bindings:

- List and names of queues
- Location of queues
- Queue usage levels (current queue depth) and peak queue depths
- Average and peak message sizes
- Message arrival rates

The administrative console can be used to monitor WebSphere messaging queues. Specialized tools are required for monitoring queues from other queue providers, including WebSphere MQ.

Database performance data

- Names (and counts) of databases and schemas
- Database connection utilization levels
 - Current number of database connections versus maximum number of database connections
- Longest running SQL queries
- Buffer utilization levels
- Sources of database performance data
 - Specialized tools are required for database performance monitoring
 - Typically provided by the database product

© Copyright IBM Corporation 2016

Figure 4-12. Database performance data

WA8152.2

Notes:

Most Java EE applications access data from a database. The following performance data is important to determine the behavior of databases that support Java EE applications:

- List and names of databases and schemas
- Database connection usage levels at the database server
- List of longest running queries
- Buffer usage levels

Specialized tools are required to collect and monitor database performance data. Typically, tools that can collect and monitor this data are included with the database product.

4.2. WebSphere performance tools

WebSphere performance tools



© Copyright IBM Corporation 2016

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

9.1

Figure 4-13. WebSphere performance tools

WA8152.2

Notes:



Performance monitoring tools

- High-end monitoring tools can be useful
 - IBM Tivoli for Composite Application Management (ITCAM) suite
 - Other non-IBM products such as CA Wily Introscope
- Much monitoring and tuning can be performed with freely available tools
 - Are already available or can be installed easily
 - Focus on the problem at hand instead of the tools
- Many tuning and problem determination tools can be downloaded with the IBM Support Assistant

© Copyright IBM Corporation 2016

Figure 4-14. Performance monitoring tools

WA8152.2

Notes:

High-end monitoring tools are available for WebSphere and can provide visibility into key WebSphere performance data. Additionally, WebSphere performance data can also be obtained by using freely available tools.

Analysis and monitoring tools: vmstat (1 of 2)

- `vmstat` is a lightweight monitoring tool for key system resources and is especially useful to monitor CPU usage
- Available on all major UNIX operating systems and Linux distributions
 - Use `mpstat` on multi-processor AIX systems to monitor all logical processor statistics
- Command format:
 - `vmstat [-options] [interval [count]]`
 - Where `interval` is the number of seconds between measurements and `count` is the number of measurements performed

© Copyright IBM Corporation 2016

Figure 4-15. Analysis and monitoring tools: vmstat (1 of 2)

WA8152.2

Notes:

The `vmstat` command provides compact information about various system resources and their related performance status. This command reports information that includes kernel thread status, memory, paging, disks, interrupts, system calls, context switches, and CPU usage levels. The reported processor usage is a percentage breakdown of user mode, system mode, idle time, and waits for disk I/O.

The `mpstat` command collects and displays performance statistics for all logical processors in the system. Users can define both the number of times the statistics are displayed, and the interval at which the data is updated. When the `mpstat` command is used, it displays two sections of statistics. The first section displays the system configuration, which is displayed when the command starts and whenever there is a change in the system configuration. The second section displays the usage statistics, which are displayed in intervals and at any time the values of these metrics are deltas from previous interval.

For more information about the `mpstat` command, see the following website:
<http://publib.boulder.ibm.com/infocenter/aix/v6r1/index.jsp?topic=%2Fcom.ibm.aix.cmds%2Fdoc%2Faixcmds3%2Fmpstat.htm>

Analysis and monitoring tools: vmstat (2 of 2)

- To print system resource usage every 10 seconds, run the following command:

```
# vmstat 10
procs -----memory----- swap-- io--- system-- cpu-----
r b swpd free buff cache si so bi bo in cs us sy id wa st
4 0 99864 60000 34404 451260 0 2 40 32 269 927 6 7 86 1 0
0 0 99848 60572 34416 451524 0 0 0 87 264 1514 17 9 74 1 0
0 0 99828 60356 34428 451728 6 0 6 50 263 1479 13 6 81 1 0
10 0 103020 51880 34372 446612 0 302 0 376 267 1259 12 35 53 0 0
2 0 111644 53328 26968 443712 6 844 7 880 265 1396 24 72 4 0 0
40 0 111636 53884 26992 442916 0 0 0 80 262 1460 17 10 69 5 0
0 0 111612 52724 27008 443412 3 0 3 74 264 1576 13 7 80 0 0
5 0 111568 51532 27020 444420 6 0 6 70 266 1614 20 17 63 0 0
```

- Output from a Linux system

© Copyright IBM Corporation 2016

Figure 4-16. Analysis and monitoring tools: vmstat (2 of 2)

WA8152.2

Notes:

- r: processes waiting to run but that are held up by the processor
- b: processes sleeping (usually waiting for I/O)
- swpd: total swap that is used (in KB)
- free: total free memory
- buff: total buffer memory
- cache: total disk cache memory usage
- si: memory that is swapped in from disk (in KB/sec)
- so: memory that is swapped out to disk (in KB/sec)
- bi: blocks that are read in from I/O devices (blocks per sec)
- bo: blocks that are written to I/O devices (blocks per sec)
- in: interrupts per second
- cs: context switches per second

- us: user processor usage in %
- sy: kernel space processor usage in %
- id: processor idle %
- wa: processor I/O waits %
- st: time that is spent in involuntary wait

Key metrics from vmstat

- CPU
 - Is the system quiet before the test begins?
 - How much CPU does the test use?
 - Is it spent productively (mostly in the “User” component of CPU)?
- Memory
 - Watch the free list for sufficient memory
 - Use `ps ev` for more detailed process memory reports
- Paging
 - The JVM must stay in memory during run time
- Context switching and system calls
 - High numbers indicate issues with excessive disk I/O or network problems

© Copyright IBM Corporation 2016

Figure 4-17. Key metrics from vmstat

WA8152.2

Notes:

The `vmstat` command reports on many operating system level performance characteristics, including:

- Processor usage
- Memory usage
- Memory paging
- Context switches

A context switch is the process of storing and restoring the state (context) of a processor so that execution can be resumed from the same point later. In this way multiple processes can share a single processor. Context switches usually require many processor cycles. A context switch can refer to a register context switch, a task context switch, a thread context switch, or a process context switch. The processor and the operating system determine the context.

Analysis and monitoring tools: nmon

- Free and powerful tool to monitor AIX and Linux systems
- Can be run in interactive mode and non-interactive mode
- Monitors most operating system resources that include:
 - CPU
 - Memory
 - Network I/O
 - Disk I/O
- Very useful on platforms that support hardware virtualization
 - Monitors LPAR resources on AIX System p machines
- Non-interactive mode probably most useful during performance test runs
 - Monitors and logs to `.csv` file
 - Use Excel Analyzer spreadsheet to import the `.csv` file and generate graphs

© Copyright IBM Corporation 2016

Figure 4-18. Analysis and monitoring tools: nmon

WA8152.2

Notes:

The nmon tool is helpful in presenting all of the important performance tuning information on one screen. The tool also updates the information dynamically. This efficient tool works on any character-based screen such as terminal window or a telnet session. In addition, the tool does not use many processor cycles. The processor consumption of the tool is usually less than 2%.

Data is displayed on the screen and updated every 2 seconds by default. You can also easily change this interval to a longer or shorter time period. If you stretch the window to full screen, the nmon tool can output a great deal of information in one place.

The nmon tool can also capture the same data to a text file for later analysis and graphing for reports. The output of the tool is in a spreadsheet format (“.CST”).

Additional information about nmon can be obtained at the following websites:

- <http://publib.boulder.ibm.com/infocenter/aix/v6r1/index.jsp?topic=%2Fcom.ibm.aix.cmds%2Fdoc%2Faixcmds4%2Fnmon.htm>
- http://www.ibm.com/developerworks/aix/library/au-analyze_aix/

nmon output of CPU and memory statistics

- nmon cm provides CPU utilization and memory statistics

The screenshot shows the nmon application window with a blue title bar and menu bar. The main area displays two sections: 'CPU Utilisation' and 'Memory Stats'. The 'CPU Utilisation' section shows a table with columns: CPU, User%, Sys%, Wait%, Idle, and %Us. The data row shows values: 1, 4.0, 4.0, 0.0, 92.0, and |Us. The 'Memory Stats' section shows various memory metrics in MB, including Total, Free, and Swap usage, as well as Cached, Active, Inactive, and Mapped memory. A warning at the bottom states: 'Warning: Some Statistics may not shown'.

CPU	User%	Sys%	Wait%	Idle	%Us
1	4.0	4.0	0.0	92.0	Us

	RAM	High	Low	Swap
Total MB	1979.3	1103.9	875.3	1521.7
Free MB	180.0	1.6	178.4	1310.7
Free Percent	9.1%	0.1%	20.4%	86.1%
	MB	MB	MB	
Buffers =	45.3	Cached = 662.0	Active = 1401.8	
Dirty =	0.2	Swapcached = 39.7	Inactive = 342.9	
Slab =	41.3	Writeback = 0.0	Mapped = 221.7	
	Commit_AS = 3554.6	PageTables = 5.8		

Warning: Some Statistics may not shown

© Copyright IBM Corporation 2016

Figure 4-19. nmon output of CPU and memory statistics

WA8152.2

Notes:

The nmon tool provides processor usage data and memory statistics. The following four types of processor workload always add up to 100% (the processor must be doing something).

- User = application code (kernel programmers call this user mode); Includes programs and RDBMS.
- Sys = AIX kernel code: Code that started by either a system call or hardware interrupt, which includes the regular clock interrupts.
- Wait = waiting for IO. This value is similar to idle, but there is outstanding disk I/O.
- Idle = nothing else to run.

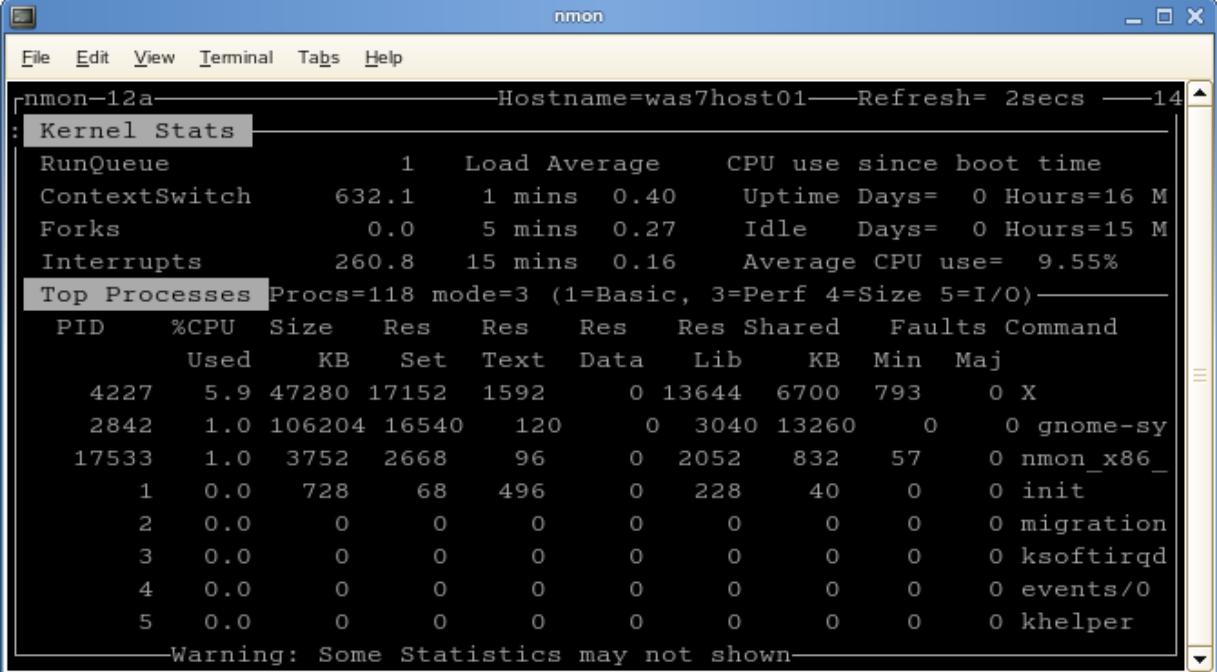
Memory use:

- Virtual: memory that is backed up by paging space and called virtual memory
- Physical: the actual RAM in the system
- Paging of memory that transfers between RAM and disk: In = to RAM; Out = to disk
 - %: Used percentage of memory that is allocated and being used

- %: Free percentage of memory that is not allocated and available
- MB: Used amount of memory that is allocated and being used in megabytes
- MB: Free amount of memory that is not allocated and available in megabytes
- Total (MB): All columns added up

nmon output of kernel statistics and top processes

- nmon kt provides kernel statistics and top processes



The screenshot shows the nmon application window with the title 'nmon'. The menu bar includes File, Edit, View, Terminal, Tabs, and Help. The main window displays 'Kernel Stats' and 'Top Processes' data.

Kernel Stats:

	RunQueue	1	Load Average	CPU use since boot time
ContextSwitch	632.1	1 mins	0.40	Uptime Days= 0 Hours=16 M
Forks	0.0	5 mins	0.27	Idle Days= 0 Hours=15 M
Interrupts	260.8	15 mins	0.16	Average CPU use= 9.55%

Top Processes:

PID	%CPU	Size	Res	Res	Res	Shared	Faults	Command
Used	KB	Set	Text	Data	Lib	KB	Min Maj	
4227	5.9	47280	17152	1592	0	13644	6700	793 0 X
2842	1.0	106204	16540	120	0	3040	13260	0 0 gnome-sy
17533	1.0	3752	2668	96	0	2052	832	57 0 nmon_x86_
1	0.0	728	68	496	0	228	40	0 0 init
2	0.0	0	0	0	0	0	0	0 migration
3	0.0	0	0	0	0	0	0	0 ksoftirqd
4	0.0	0	0	0	0	0	0	0 events/0
5	0.0	0	0	0	0	0	0	0 khelper

Warning: Some Statistics may not shown

© Copyright IBM Corporation 2016

Figure 4-20. nmon output of kernel statistics and top processes

WA8152.2

Notes:

Kernel statistics include information about: RunQueue, ContextSwitch, forks, and interrupts.

Top processes identified by PID, amount of processor used, size in memory, and so forth.

Kernel internal statistics:

- RunQueue: Run queue length (processes that are waiting for the processor)
- SwapIn: processes swapped back in after thrashing
- iget: inode (JFS file descriptor) gets from the disk
- namei: file and directory lookup within the JFS
- dirblk: directory block that is read within the JFS
- pswitch: process switches (changes between user applications)
- syscall: system calls (application that requests AIX services)
- rawch: raw character that is read in from tty
- canch: canonical character that is read in and processes

- outch: character output to tty
- read: read system call (all types of device disk, network, socket, pipe)
- write: write system call (all types of device disk, network, socket, pipe)
- fork: new process creation (clone current process)
- exec: new program code started (over writing current process with a new program)
- readch: characters read
- writech: characters written
- msg: shared messages that are written between applications
- sem: shared semaphore operations between applications

In addition, nmon can provide the following performance data:

- Processor usage
- Memory use
- Kernel statistics and run queue
- Disks I/O rates, transfers, and read/write ratios
- File system size and free space
- Disk adapters
- Network I/O rates, transfers, and read/write ratios
- Paging space and paging rates
- Machine details, processor, and OS specification
- Top processors
- User-defined disk groups
- Asynchronous I/O: AIX only
- Workload manager: AIX only
- ESS and other disk subsystem: AIX only
- Dynamic LPAR changes: AIX and Linux (on POWER hardware)



Web server performance tools: IBM HTTP Server server-status (1 of 3)

- `server-status` allows easy monitoring of a web server through a browser
- Enable by adding the following lines to `httpd.conf` file

```
<Location /server-status>
  SetHandler server-status
</Location>
```

- After a web server restart, the `server-status` is available
 - `http://yourhost/server-status`
- Optionally, specify a refresh interval
 - For example 10 seconds `http://yourhost/server-status?refresh=10`
- Optionally, monitor extended status
 - Use `ExtendedStatus On` directive
 - Provides more detailed child level statistics at some performance cost

© Copyright IBM Corporation 2016

Figure 4-21. Web server performance tools: IBM HTTP Server server-status (1 of 3)

WA8152.2

Notes:

The HTTP server status module allows a server administrator to find out how well their web server is running. An HTML page is presented that gives the current server statistics in an easily readable form. If required, this page can be made to automatically refresh (given a compatible browser). Another page gives a simple machine-readable list of the current server state.

The details that are given are:

- The number of workers that server requests
- The number of idle workers
- The status of each worker, the number of requests that worker completes and the total number of bytes served by the worker (extended status)
- A total number of accesses and byte count served (extended status)
- The time the server was started or restarted and the length of time it is running
- Averages giving the number of requests per second, the number of bytes served per second, and the average number of bytes per request (extended status)

- The current percentage of processor that is used by each worker and in total by the web server (extended status)
- The current hosts and requests that are processed (extended status)

Additional information on the topic of IBM HTTP Server “server-status” can be obtained at the following website:

• http://pic.dhe.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=%2Fcom.ibm.websphere.base.doc%2Finfo%2Faes%2Fae%2Ftprf_tunewebserv.html

The screenshot shows a Mozilla Firefox browser window displaying the 'Apache Status' page for an IBM HTTP Server running on localhost. The title bar reads 'Apache Status - Mozilla Firefox'. The address bar shows 'localhost/server-status?refresh=10'. The main content area is titled 'Apache Server Status for localhost'. It provides server statistics: Server Version: IBM_HTTP_Server/8.5.5.0 (Unix), Server Built: Feb 20 2013 13:54:09. Below this, it lists current time (Sunday, 16-Mar-2014 16:11:12 EDT), restart time (Wednesday, 12-Mar-2014 19:51:06 EDT), parent server generation (0), server uptime (3 days 20 hours 20 minutes 5 seconds), total accesses (264935), total traffic (2.7 GB), CPU usage (u80.05 s47.96 cu0 cs0 - .0385% CPU load), request rates (.797 requests/sec - 8.6 kB/second - 10.8 kB/request), and the number of requests currently being processed (21) and idle workers (29). A yellow callout box on the left side of the screenshot contains the text: 'Example server-status data with 10-second refresh'.

© Copyright IBM Corporation 2016

Figure 4-22. Web server performance tools: IBM HTTP Server server-status (2 of 3)

WA8152.2

Notes:

This screen capture is a sample of the server status report page.

The status module allows server administrators to find out how well their server is doing the work. An HTML page is presented that gives the current server statistics in an easily readable form. If required, this page can be made to automatically refresh (given a compatible browser). Another page gives a simple machine-readable list of the current server state.

The details that are given are:

- The number of workers that serve requests
- The number of idle workers
- The status of each worker, the number of requests that worker performed, and the total number of bytes served by the worker (*)
- A total number of accesses and byte count served (*)
- The time the server was started or restarted and the time it is running
- Averages giving the number of requests per second, the number of bytes served per second and the average number of bytes per request (*)

- The current percentage processor that is used by each worker and in total by Apache (*)
- The current hosts and requests that are processed (*)

The lines marked “(*)” are only available if ExtendedStatus is On.

The screenshot shows a Mozilla Firefox browser window titled "Apache Status - Mozilla Firefox". The address bar displays "localhost/server-status?refresh=10". The main content area is titled "WebSphere Plugin status (pid 15584)". Below this, a section titled "Server groups" lists two groups:

- Server group TradeCluster
 - Server hostCNode01_TradeServer2 is marked up (current conns 3, total conns 28964)
 - Server hostBNode01_TradeServer1 is marked up (current conns 6, total conns 72412)
- Server group server1_hostBNode01_Cluster
 - Server hostBNode01_server1 is marked up (current conns 0, total conns 0)

At the bottom of the page, it says "IBM_HTTP_Server at localhost Port 80". A small note in the top left corner of the screenshot area says "Example server-status WebSphere Plugin status".

© Copyright IBM Corporation 2016

Figure 4-23. Web server performance tools: IBM HTTP Server server-status (3 of 3)

WA8152.2

Notes:

This screen capture is a sample of the server status report page that is available when the plug-in configuration contains information about application server clusters.

Web server performance tools: Web server access logs (1 of 4)

- Apache response time logging is a powerful tool
 - Used to obtain the actual response times at the server side
 - Does not depend on the speed of the network or any other unknowns such as proxy servers

- Enable using standard Apache web server directives in the `httpd.conf` file

```
LogFormat "%t \"%r\" Response time (us): %D" responsetime
CustomLog logs/response.log responsetime
```

- Logs response times accurately in μ s (microseconds)
 - Almost no performance cost so can be used in production
 - Uses various tools to analyze response time characteristics

© Copyright IBM Corporation 2016

Figure 4-24. Web server performance tools: Web server access logs (1 of 4)

WA8152.2

Notes:

The web server access log records performance statistics about all requests that the web server processes. The information that is logged is configurable, and the menu of statistics to choose from is extensive. The “common” log format includes the following information:

- The IP address of the client that made the request to the web server
- The remote “logname”
- The user ID of the person who requests the content as determined by HTTP authentication
- The time the web server finished processing the request
- The HTTP request line from the client
- The size in bytes of the response content that is returned to the client, not including response headers

More information about the web server access log can be obtained at the following website:
<http://httpd.apache.org/docs/1.3/logs.html#accesslog>

Web server performance tools: Web server access logs (2 of 4)

- Example of response.log data

```
[Time Stamp] "POST /daytrader/app HTTP/1.1" Response time (us): 26566366
[TS] "GET /daytrader/images/arrowup.gif HTTP/1.1" Response time (us): 551
[TS] "POST /daytrader/app HTTP/1.1" Response time (us): 26596598
[TS] "POST /daytrader/app HTTP/1.1" Response time (us): 26597933
[TS] "POST /daytrader/app HTTP/1.1" Response time (us): 27072537
.
.
.
[TS] "GET /daytrader/app?action=buy&symbol=s%3A3&quantity=100 HTTP/1.1" Response time (us): 24067107
[TS] "GET /daytrader/app?action=buy&symbol=s%3A4&quantity=100 HTTP/1.1" Response time (us): 11942057
[TS] "GET /daytrader/app?action=quotes&symbols=s:0,s:1,s:2,s:3,s:4 HTTP/1.1" Response time (us): 28465179
[TS] "GET /daytrader/app?action=quotes&symbols=s:0,s:1,s:2,s:3,s:4 HTTP/1.1" Response time (us): 28755041
[TS] "GET /daytrader/app?action=buy&symbol=s%3A4&quantity=100 HTTP/1.1" Response time (us): 18075762
```

© Copyright IBM Corporation 2016

Figure 4-25. Web server performance tools: Web server access logs (2 of 4)

WA8152.2

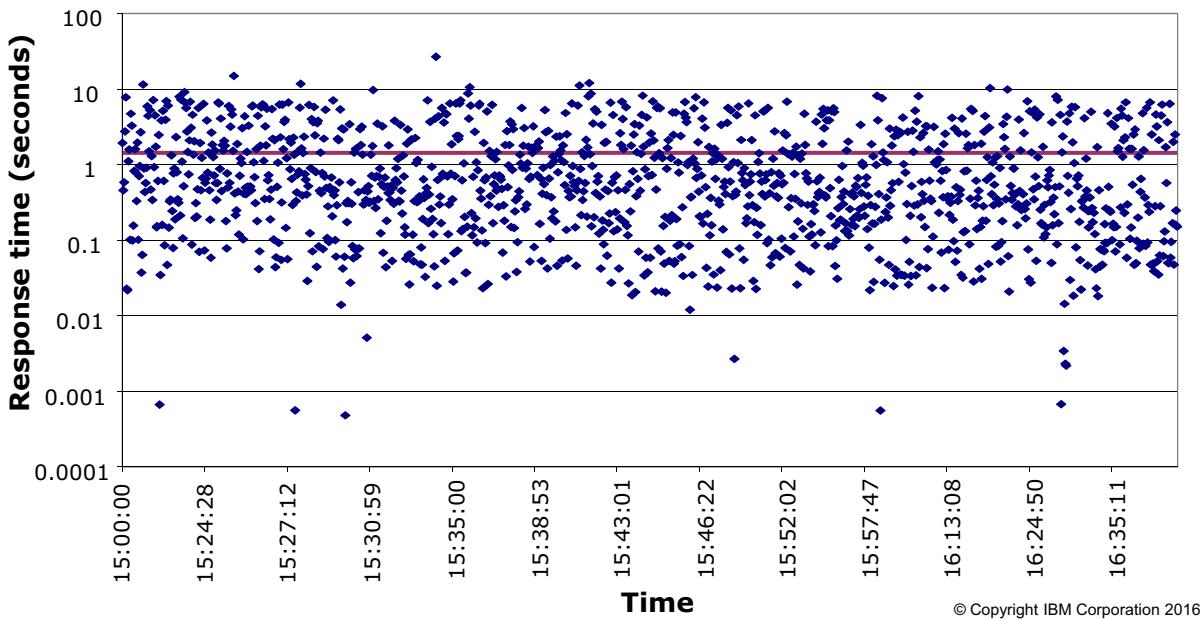
Notes:

The amount of data in the response log file can be large in a production environment. Be careful to manage log rotation and disk space.



Web server performance tools: Web server access logs (3 of 4)

- Distribution of response times is important in real life
 - Provides more insight into your application than the average
 - More credible since you can easily determine what percentage is faster than x seconds



© Copyright IBM Corporation 2016

Figure 4-26. Web server performance tools: Web server access logs (3 of 4)

WA8152.2

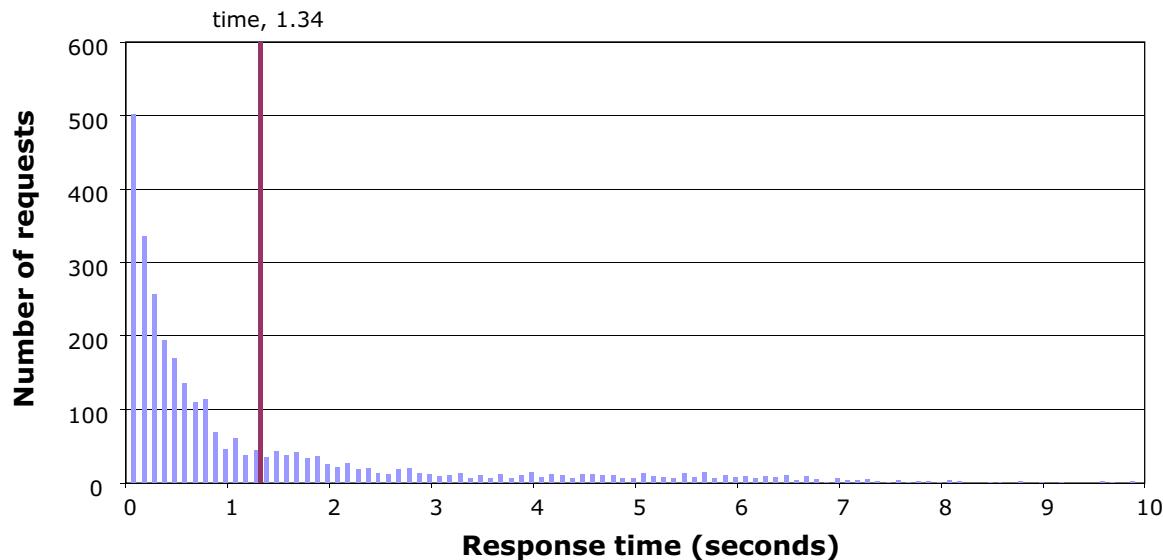
Notes:

This chart shows an example of a graph that can be produced from the statistics in a web server access log.

Web server performance tools: Web server access logs (4 of 4)

Analysis of Apache response time logs can be powerful

- Use standard tools such as Microsoft Excel
- Use various graphing options:
 - Logarithmic
 - Histogram



© Copyright IBM Corporation 2016

Figure 4-27. Web server performance tools: Web server access logs (4 of 4)

WA8152.2

Notes:

The bar chart shows number of requests that are plotted against response time.



JVM performance tools: Verbose garbage collection

- The best way to analyze garbage collection behavior is to enable and collect verbose GC data and graph it
 - Garbage collection time data
- The IBM Support Assistant includes a tool that is called Garbage Collection and Memory Visualizer (GCMV) which will graph verbose GC
- GCMV uses a powerful statistical analysis engine that provides guidance in the following areas:
 - Memory leak detection
 - Optimize garbage collection performance
 - Fine-tuning of Java heap size

© Copyright IBM Corporation 2016

Figure 4-28. JVM performance tools: Verbose garbage collection

WA8152.2

Notes:

The Java virtual machine (JVM) reports statistics about each garbage collection cycle when verbose garbage collection (“verbose GC”) is enabled. These statistics are written to an application server log file. This log file can be inspected by using a simple editor, but the best way to view garbage collection behavior is to graph verbose GC output.

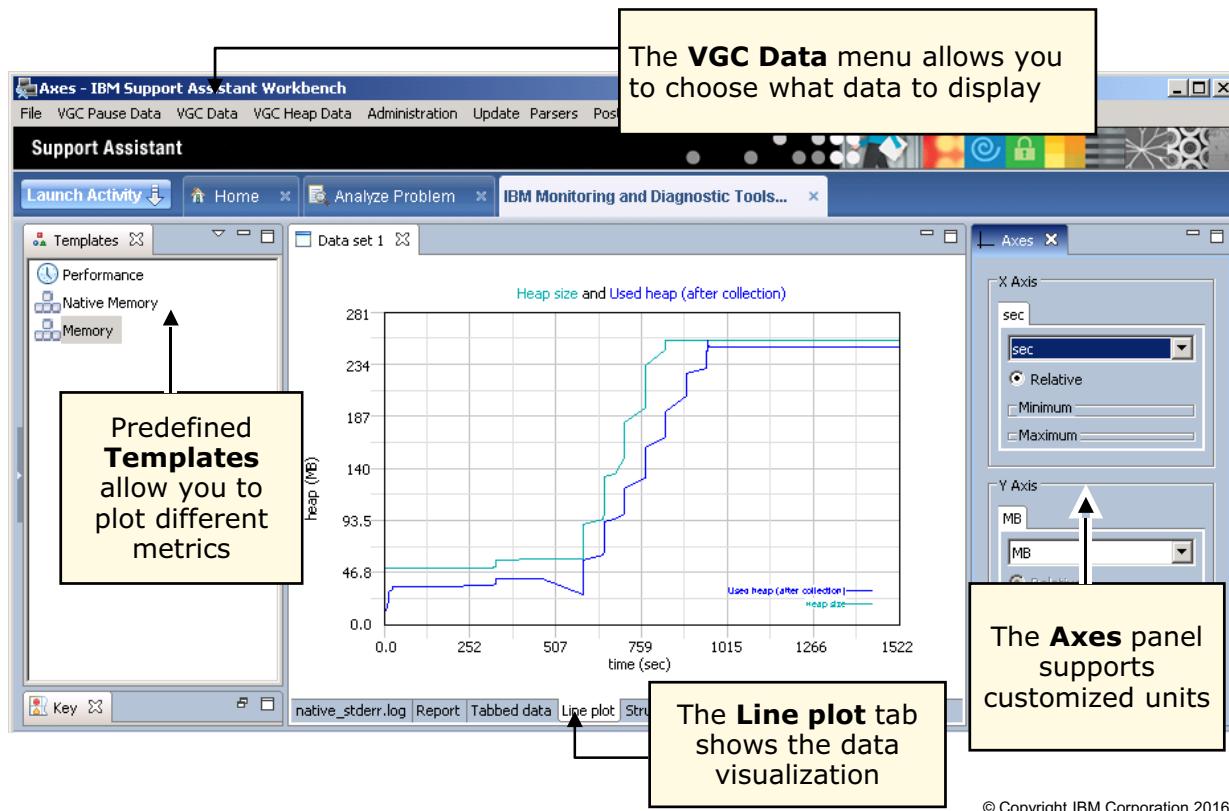
Garbage Collection and Memory Visualizer (GCMV) provides guidance in the following areas:

- Memory leak detection:
 - Detect Java heap exhaustion and memory leaks
 - Detect native heap (malloc) exhaustion and memory leaks
- Optimize garbage collection performance:
 - Determine garbage collection frequency
 - Detect long or frequent garbage collection cycles and suggest configuration changes to address these issues
 - Suggest optimal garbage collection policy

- Fine-tuning of Java heap size:
 - Determine peak and average memory usage
 - Suggest Java heap settings



JVM performance tools: GCMV



© Copyright IBM Corporation 2016

Figure 4-29. JVM performance tools: GCMV

WA8152.2

Notes:

One of the tools that can be used to graph verbose GC output is the Garbage Collection and Memory Visualizer (GCMV) tool that is part of the IBM Support Assistant. This tool includes the following features:

- Parses and plots IBM verbose GC logs
- Provides a graphical display of a wide range of verbose GC data values
- Supports various garbage collection policies that include “optthuput”, “optavgpause”, and “gencon”
- Provides “report”, “raw log”, “tabulated data”, and graph views
- Supports saving data to HTML reports, .jpg images, or “.csv” files
- Supports the comparison of multiple logs

The latest version includes the following enhancements:

- More accurate determination of memory leaks
- Automatic axis unit selection
- Improved summary report



JVM performance tools: Thread dumps (1 of 2)

- Thread dumps provide a snapshot of the running JVM
- Useful when:
 - CPU utilization is not as expected (lower or higher)
 - Throughput hits a plateau
- Thread dump produces a stack trace of each thread's activity
- Things to keep in mind:
 - A thread dump is a snapshot
 - Take several thread dumps, giving the JVM time to recover after each
 - They vary in format and detail among platforms

© Copyright IBM Corporation 2016

Figure 4-30. JVM performance tools: Thread dumps (1 of 2)

WA8152.2

Notes:

A thread memory dump is designed to provide a snapshot of the execution point of each thread that currently exists in a JVM process. The information that is provided for each thread is a Java call stack that indicates all Java calls that were processed, from the time when each thread first started to the current state.

JVM performance tools: Thread dumps (2 of 2)

- Analyzing thread dumps with a plain editor can be difficult
 - Use a thread dump viewer
- IBM Thread and Monitor Dump Analyzer for Java
 - Simplifies the process of analyzing a thread dump by presenting a list of the threads that existed at the time the thread dump was triggered
 - Simplifies the process of examining the Java call stack from the list of threads
 - Provides information about locking
 - Shows the threads that own locks
 - Shows the threads that are blocked waiting on a lock

© Copyright IBM Corporation 2016

Figure 4-31. JVM performance tools: Thread dumps (2 of 2)

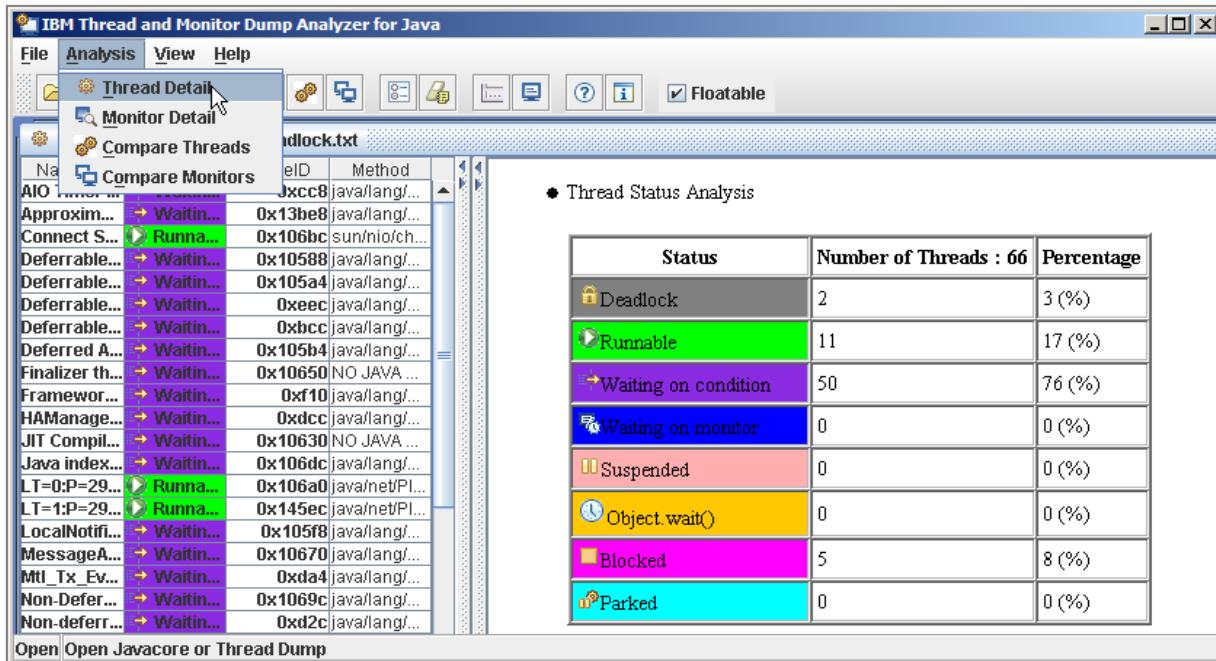
WA8152.2

Notes:

Tools are available that help with the analysis of a thread memory dump. One of the tools is called the IBM Thread and Monitor Dump Analyzer for Java. This tool simplifies the process of analyzing a thread memory dump by presenting a list of the threads that existed at the time the thread memory dump was triggered. The tool also provides information about locking; it shows the threads that own locks and the threads that are blocked waiting on a lock.

Thread and Monitor Dump Analyzer (TMDA)

- Import one or more javacore files into TMDA and view analysis



© Copyright IBM Corporation 2016

Figure 4-32. Thread and Monitor Dump Analyzer (TMDA)

WA8152.2

Notes:

The Analysis menu allows you to display thread and monitor details for a single javacore. If you open multiple Java core files, you can display a comparative thread or monitor analysis. The Thread Detail Analysis displays the following information:

- Thread status analysis: the number of threads in each state: deadlocked, runnable, blocked, and so forth.
- Thread method analysis
- Thread aggregation analysis

The left pane lists all thread in the thread memory dump. Threads can be sorted by thread name, state, native ID, or current method. When you select a thread from the left pane, the right pane shows the Thread Detail view. The Thread Detail view provides the following information:

- Thread name: the name of a thread
- Thread state: the state of a thread; for example, Runnable, Waiting, or Suspended
- Method name: The latest method that is invoked or predefined status or stack trace pattern; for example, IDLE, LISTEN and KEEP-ALIVE

- Java stack trace: Java stack trace is shown when a thread is selected
- Native stack trace: Native stack trace is shown after the Java stack trace if it is available

Thread states:

- Runnable: thread that is able to run or is running
- Conditional wait (CW): thread that is waiting on a condition variable
- Monitor wait (MW): thread that is waiting on a monitor lock
- Suspended (S): thread suspended

An IDLE thread is a thread that is ready to receive work but does not have a connection that is established.

A KEEP-ALIVE thread is an idle thread that is ready to receive work and does have a connection that is established.

A LISTEN thread listens on a port.



Messaging performance tools

- WebSphere MQ
 - WebSphere MQ Explorer can be used to monitor WebSphere MQ hosted queues
 - WebSphere MQ Explorer comes with the WebSphere MQ product
- WebSphere default messaging (Service Integration Bus)
 - Service Integration Bus destination handler
 - Various aspects of Service Integration Bus status can be monitored through the administrative console and with Tivoli Performance Viewer
- Functions available in both tools
 - Provide a list of queues
 - Report current queue depths
 - Manage queue contents

© Copyright IBM Corporation 2016

Figure 4-33. Messaging performance tools

WA8152.2

Notes:

Tools are available that provide information on the status of messaging subsystem. For WebSphere MQ, you can use the WebSphere MQ Explorer tool that comes with the WebSphere MQ product.

You can obtain more information regarding MQ Explorer from the following website:

- <http://www.ibm.com/software/integration/wmq/explorer/>

Service Integration Bus Destination Handler is a tool for the default messaging provider for WebSphere Application Server V6.0, V6.1, V7.0, and V8.5.

You can use the tool to complete various actions on messages in a Service Integration Bus without writing any custom code.

These actions can be run once to do an individual task. The tool can also be deployed as a WebSphere Application Server scheduler task to regularly check the contents of an exception destination and provide appropriate handling of messages that are based on the set of rules configured.

This tool can be downloaded from:

<http://www.ibm.com/support/docview.wss?rs=180&uid=swg24021439>

WebSphere performance data: Performance Monitoring Infrastructure (PMI)

- When PMI is enabled for an application server, monitoring of individual WebSphere components can be enabled or disabled dynamically
- WebSphere PMI provides four predefined statistic sets that can be used to enable a set of statistics:

Statistics sets	Description
Basic	Statistics that are specified in Java EE specification, as well as top statistics like CPU usage and live HTTP sessions, are enabled. This set is enabled by default and provides basic performance data about runtime and application components (performance cost of up to 2%).
Extended	Basic statistics plus key statistics from various WebSphere components like workload management (WLM) and dynamic caching (performance cost of up to 3%).
All	All statistics are enabled (performance cost of up to 6%).
Custom	Allows the user to enable or disable statistics selectively.

- PMI levels can be set via the administrative console or the `wsadmin` command

© Copyright IBM Corporation 2016

Figure 4-34. WebSphere performance data: Performance Monitoring Infrastructure (PMI)

WA8152.2

Notes:

WebSphere Application Server is instrumented to collect statistics that show application behavior by using a facility that is called the Performance Monitoring Infrastructure (PMI). PMI can report on a comprehensive set of data that illustrates the runtime resource usage levels. For example, PMI can report statistics on the following resources:

- Database connection pool size
- Servlet response time
- EJB method response time
- JVM heap size
- Processor usage

This data can be viewed by using the Tivoli Performance Viewer that comes in the administrative console.

Additional information about PMI can be obtained in the WebSphere Application Server product documentation.



WebSphere performance tools: Tivoli Performance Viewer (1 of 2)

- Tivoli Performance Viewer is part of the administrative console
- Allows graphical monitoring with PMI of:
 - WebSphere managed resources
 - Thread pools
 - Connection pools
 - Connection factories
 - Messaging engine
 - JVM heap usage
 - Others
- Response times
 - JSPs and servlets
 - EJBs

Start Monitoring Stop Monitoring					
Select	Server	Node	Host Name	Version	Collection Status
<input type="checkbox"/>	TradeServer1	was7host01Node01	was7host01	ND 7.0.0.9	Monitored
<input type="checkbox"/>	nodeagent	was7host01Node01	was7host01	ND 7.0.0.9	Monitored
<input type="checkbox"/>	server1	was7host01Node01	was7host01	ND 7.0.0.9	Unavailable, server st

Total 3

© Copyright IBM Corporation 2016

Figure 4-35. WebSphere performance tools: Tivoli Performance Viewer (1 of 2)

WA8152.2

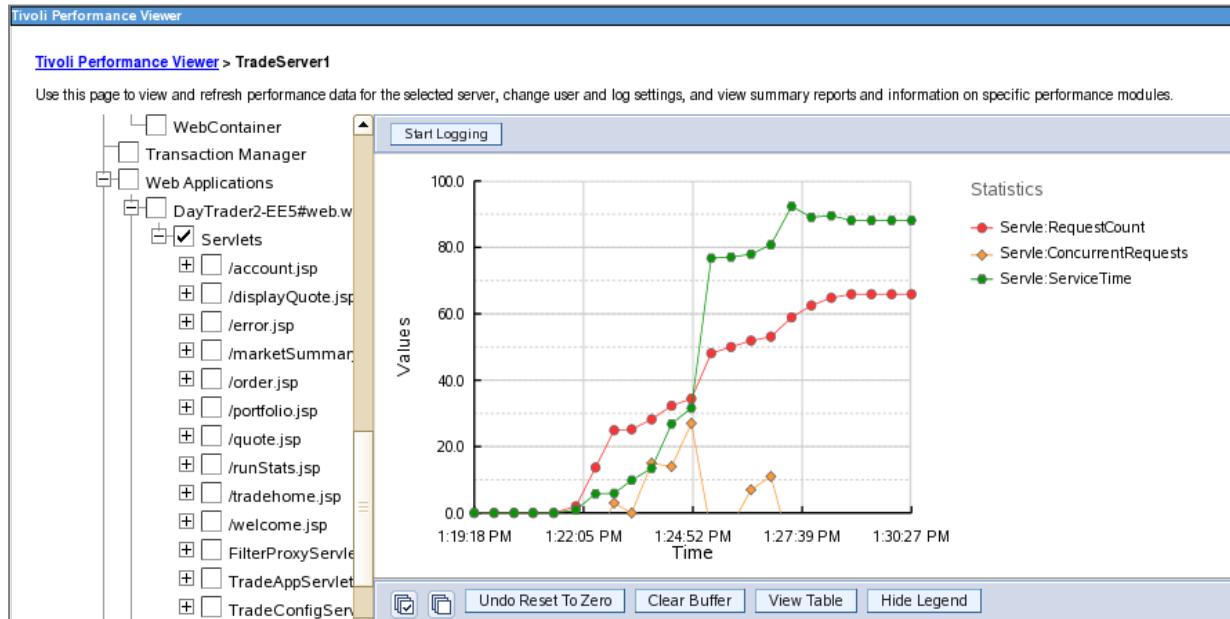
Notes:

The Tivoli Performance Viewer enables administrators and programmers to monitor the overall health of WebSphere Application Server from within the administrative console. Tivoli Performance Viewer helps users monitor the behavior of WebSphere-based applications. The statistics that can be monitored include servlet statistics, enterprise bean statistics, thread pool statistics, connection pool statistics, and many others. This information can then be used to optimize the tuning of the application server.

More information about the Tivoli Performance Viewer can be obtained in the WebSphere Application Server product documentation.

WebSphere performance tools: Tivoli Performance Viewer (2 of 2)

- Chart view of TPV monitoring DayTrader servlets



© Copyright IBM Corporation 2016

Figure 4-36. WebSphere performance tools: Tivoli Performance Viewer (2 of 2)

WA8152.2

Notes:

This Tivoli Performance Viewer chart shows servlets metrics over several minutes of activity. The servlets statistics that are monitored are:

- Request count: total number of requests that are processed by a servlet
- Concurrent requests: number of requests concurrent processed
- Service time: the average response time in milliseconds, in which a servlet request is completed

WebSphere performance tools: Request metrics overview

- Tracing facility that allows you to measure the amount of time a request spends in each component that is traversed during its execution
- Captured information includes:
 - Elapsed time in the web server
 - Response time of invoked components in the web and EJB containers
 - Response time of related JDBC calls
- Writes trace records to `SystemOut.log` or sends metrics to an Application Response Measurement (ARM) agent

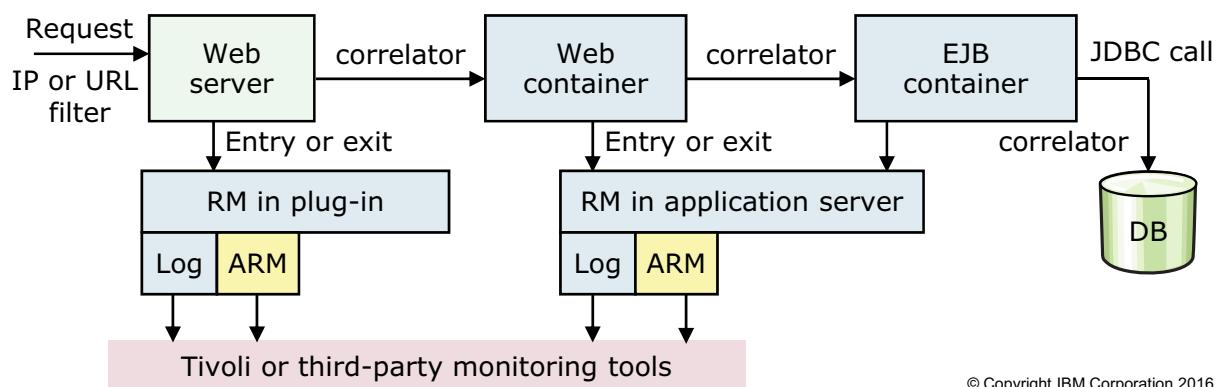


Figure 4-37. WebSphere performance tools: Request metrics overview

WA8152.2

Notes:

Request metrics allow you to monitor the transaction flow and analyze the response time of the components that are involved in running it. This analysis can help you target performance problem areas and debug resource constraint problems. For example, it can help determine whether a transaction spends most of its time in the web server plug-in, the web container, the Enterprise JavaBeans (EJB) container, or the database. The response time that is collected for each level includes the time that is spent at that level and the time that is spent in the lower levels. For example, if the total response time for the servlet is 130 milliseconds, and it includes 38 milliseconds from the enterprise beans and JDBC, then 92 milliseconds can be attributed to the servlet process.

An ARM agent does not come with WebSphere Application Server. Third-party tools usually provide an ARM agent.

WebSphere performance tools: Request metrics (1 of 2)

- Provides a hierarchical view of each request by response time against the following application server components:
 - Web server plug-in (for requests that arrive at the web server)
 - Proxy server
 - Web container
 - EJB container
 - JDBC calls
 - Java Connector Architecture (JCA)
 - Web services
 - JMS interactions
 - Service Integration Bus (SIBus)
 - Portlet container
 - Asynchronous beans

© Copyright IBM Corporation 2016

Figure 4-38. WebSphere performance tools: Request metrics (1 of 2)

WA8152.2

Notes:

ARM = application request/response metrics

Request metrics is a facility that enables tracking of individual Java EE requests recording the processing time (response time and latency) in each of the major WebSphere Application Server components. The information that request metrics tracks can be either saved to log files or sent to ARM agents (or both).

Additional information about the topic of request metrics can be obtained in the WebSphere Application Server product documentation.

WebSphere performance tools: Request metrics (2 of 2)

- Collecting performance data at all of the instrumented application components can have an adverse effect on performance
- The impact of monitoring can be minimized by using filtering to limit the amount of monitoring per transaction
- Request metrics provide filtering against the following transaction details:
 - Source IP
 - URI
 - EJB method name
 - JMS parameters
 - Web services parameters

© Copyright IBM Corporation 2016

Figure 4-39. WebSphere performance tools: Request metrics (2 of 2)

WA8152.2

Notes:

Since request metrics track the response time for each individual request at multiple layers of the application server architect, monitoring with request metrics can have an adverse impact on performance. This impact can be mitigated by using request metric's request filtering capabilities. The following types of filters are provided:

- Source IP
- URI
- EJB method name
- JMS parameters
- Web services parameters

When filtering is enabled, only requests that match the configured filter generate request metrics data, create log records, call the ARM interfaces, or all of the previously listed actions.



A final word on performance tools

- The tools that are discussed in this unit are only a small subset of the tools that are used to monitor WebSphere performance
- Investigate all available tools and use tooling that makes your team the most productive
- Use the right tool at the right time
 - Use the tool with which you are familiar
 - Use the tool that does not require extensive setup
 - Use the tool that is cost effective

© Copyright IBM Corporation 2016

Figure 4-40. A final word on performance tools

WA8152.2

Notes:

The unit presented a small sampling of tools for gathering performance data and analyzing the data.



Unit summary

Having completed this unit, you should be able to:

- Describe important performance data for monitoring WebSphere based applications
- Identify useful performance tools for monitoring performance data that describes the behavior of WebSphere based applications

© Copyright IBM Corporation 2016

Figure 4-41. Unit summary

WA8152.2

Notes:

Checkpoint questions

1. True or False: Critical hardware performance data includes amount of physical memory, free disk space, and I/O channel utilization.
2. True or False: JVM performance data includes connection pools size, HTTP response time, and native heap size.
3. True or False: Some of the key OS metrics that the vmstat tool provides are gathered from CPU, memory, and paging.
4. True or False: There are currently no tools available for analyzing verbose garbage collection data. Analysis of verbose GC must be done manually.

© Copyright IBM Corporation 2016

Figure 4-42. Checkpoint questions

WA8152.2

Notes:

Write your answers here:

- 1.
- 2.
- 3.
- 4.



Checkpoint answers

1. True
2. False: JVM performance data includes verbose GC data, thread dumps, and Java heap dumps.
3. True
4. False: There are several tools available. The Garbage Collection and Memory Visualizer (GCMV) is one example.

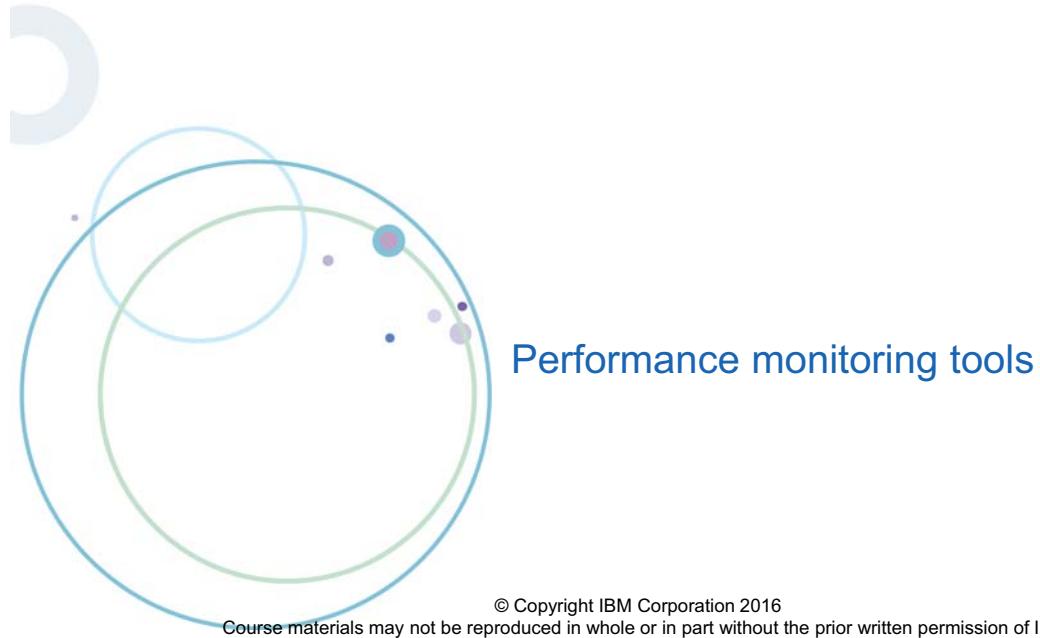
© Copyright IBM Corporation 2016

Figure 4-43. Checkpoint answers

WA8152.2

Notes:

Exercise 5



© Copyright IBM Corporation 2016

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

9.1

Figure 4-44. Exercise 5

WA8152.2

Notes:



Exercise objectives

After completing this exercise, you should be able to:

- Monitor performance by using Tivoli Performance Viewer
- Use the performance advisors, including runtime Performance Advisor and Performance Advisor in Performance Viewer
- Use the Health Center tool in the IBM Support Assistant to monitor a running JVM

© Copyright IBM Corporation 2016

Figure 4-45. Exercise objectives

WA8152.2

Notes:

Unit 5. WebSphere performance tuning methods

What this unit is about

The purpose of this unit is to discuss various techniques and methods for monitoring and tuning performance.

What you should be able to do

After completing this unit, you should be able to:

- Describe performance tuning goals, strategies, and considerations
- Evaluate a list of the top 10 monitoring considerations
- Gather performance data for WebSphere Application Server components
- Detect performance bottlenecks
- Determine optimum queue sizes
- Configure WebSphere Application Server system queues
- Identify and evaluate WebSphere tuning parameters

How you will check your progress

- Checkpoint questions

References

WebSphere Application Server V8.5 Network Deployment Information Center, Performance Tuning article,

http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/index.jsp?topic=%2Fcom.ibm.websphere.nd.doc%2Fae%2Ftfullp_tun.html

Case study: Tuning WebSphere Application Server V7 for performance

http://www.ibm.com/developerworks/websphere/techjournal/0909_blythe/0909_blythe.html

IBM Redbooks: WebSphere Application Server V6 Scalability and Performance Handbook, SG24-6392-00

Unit objectives

After completing this unit, you should be able to:

- Describe performance tuning goals, strategies, and considerations
- Evaluate a list of the top 10 monitoring considerations
- Gather performance data for WebSphere Application Server components
- Detect performance bottlenecks
- Determine optimum queue sizes
- Configure WebSphere Application Server system queues
- Identify and evaluate WebSphere tuning parameters

© Copyright IBM Corporation 2016

Figure 5-1. Unit objectives

WA8152.2

Notes:



Topics

- Performance tuning in practice
- Gathering performance tuning data
- Top 10 monitoring hot list for WebSphere performance tuning
- Detecting bottlenecks
- System queues

© Copyright IBM Corporation 2016

Figure 5-2. Topics

WA8152.2

Notes:

5.1. Performance tuning in practice

Performance tuning in practice



© Copyright IBM Corporation 2016

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

9.1

Figure 5-3. Performance tuning in practice

WA8152.2

Notes:

Preparation for performance tuning

- Before you start tuning and managing the performance of any system
 - Understand the system components and how they can affect the overall performance
- To tune performance in a WebSphere Application Server environment
 - Know every part of the system that is involved in request flow of your application
- Maintaining good performance requires you to:
 - Understand your deployment environment
 - Plan for scalability and capacity
 - Meet performance requirements for throughput and response time

© Copyright IBM Corporation 2016

Figure 5-4. Preparation for performance tuning

WA8152.2

Notes:

Before you can start tuning and managing the performance of any system, you must understand your system's components and how they can affect system overall performance. To tune performance in a WebSphere Application Server environment, you must know every part of your system that is involved in this process. Maintaining good performance requires you to understand, plan, manage performance policies, and track them.



Performance tuning goals

- Improve throughput
 - Maximize the number of transactions that are served in a certain amount of time
- Improve response time
 - Minimize the time that it takes to respond to client requests
- Performance expectations are often expressed in terms of "number of clients"
 - Need to translate those figures in terms of throughput and response time

© Copyright IBM Corporation 2016

Figure 5-5. Performance tuning goals

WA8152.2

Notes:

The goals of performance tuning are as follows:

- Improve throughput to maximize the number of transactions that are served in a certain amount of time.
- Improve response time to minimize the time it takes to respond to clients' requests.
- Performance expectations are often expressed in terms of "number of clients". You need to translate those figures in terms of transactions/hour and response time.

Application server parameter tuning

- The art of changing WebSphere Application Server settings with the goal of improving performance
- WebSphere Application Server offers an extensive list of tuning knobs and parameters that can be used to enhance the performance of an application
- Default values for the most commonly used tuning parameters in the server are set to ensure adequate performance, without customization, for the broadest range of applications
 - No two applications are alike and use the server in the same fashion
 - Cannot guarantee that a single set of tuning parameters are perfectly suited for every application
 - It is important to conduct focused performance testing and tuning against your own application

© Copyright IBM Corporation 2016

Figure 5-6. Application server parameter tuning

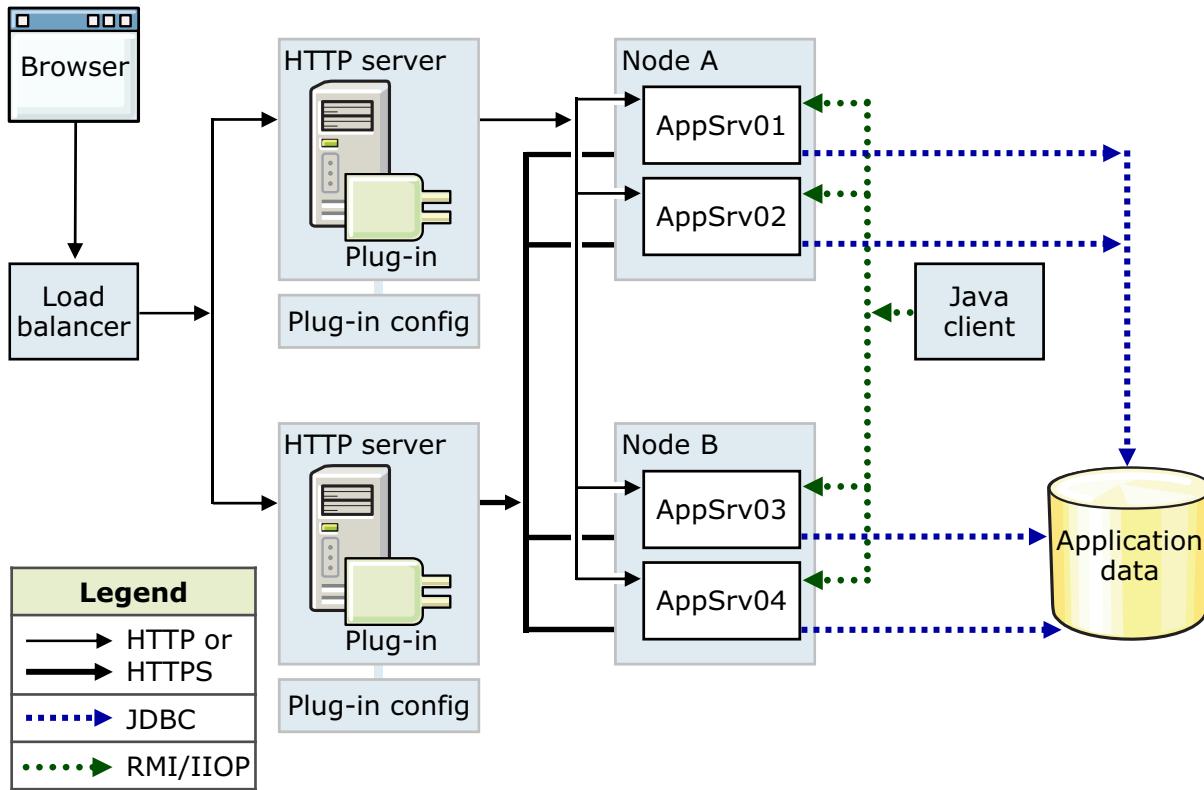
WA8152.2

Notes:

There are two types of tuning:

- **Application tuning** sometimes offers the greatest tuning improvements; this course first focuses on tuning individual performance parameters and the interactions between them.
- **Application server parameter tuning** is the art of changing WebSphere Application Server settings with the goal of improving performance.

Typical Java EE application topology



© Copyright IBM Corporation 2016

Figure 5-7. Typical Java EE application topology

WA8152.2

Notes:

In a typical Java EE application topology, one or more of the following components can cause performance issues:

- Web server
- Load balancer
- Firewalls
- Hardware
- Network
- Scaling
- Application server JVM
- Application design
- EJBs
- Database

- JMS queues

The problem can be almost anywhere within an application. The problem can be network and hardware-related, back-end, or system-related. The problem can be actual product bugs, or often, application design issues.

With increasingly complex applications, it is getting more difficult to track down these problems.

Typical e-business applications now have one or several of the following attributes:

- Multiple logical or physical tiers
- Mixture of operating system platforms
- Mixture of hardware architectures
- Fuzzy application boundaries

Software can produce massive quantities of numbers: CPU usage statistics, memory usage statistics, queuing statistics, transaction rate statistics, buffer pool statistics, and so forth. The problem is that the large number of statistics can be overwhelming and might hamper, rather than help, a person's ability to understand and analyze performance data.

What influences tuning?

- The following are parameters that can affect the performance of WebSphere Application Server:
 - Application that is used
 - Hardware capacity and settings
 - Operating system settings
 - Web server
 - WebSphere Application Server process
 - Java virtual machine (JVM)
 - Database server
- Each parameter has its own tuning options, varying in importance and affect on performance

© Copyright IBM Corporation 2016

Figure 5-8. What influences tuning?

WA8152.2

Notes:

Once in production

- Use statistics that are taken from the production machine as feedback to your performance testing environment:
 - Was the number of users and transactions per user correct?
 - Was the transaction mix correct?
 - In general, did the performance testing realistically simulate the production environment? If not, what can be changed?
- The production environment can also be used for performance testing:
 - The internal logging features of the application can be used for response time statistics and the composition of the transactions
 - Performance tools can be used, depending on their impact on the environment
 - Collected data can be used to drive further application changes

© Copyright IBM Corporation 2016

Figure 5-9. Once in production

WA8152.2

Notes:

Important performance data can be collected when the application is running in production. Use statistics that are taken from the production machine as feedback to your performance testing environment:

- Was the number of users and transactions per user correct?
- Was the transaction mix correct?
- In general, did the performance testing realistically simulate the production environment? If not, what can be changed?

The production environment can also be used for performance testing:

- The internal logging features of the application can be used for response time statistics and the composition of the transactions
- Performance tools can be used, depending on their impact on the environment
- Collected data can be used to drive further application changes

5.2. Gathering performance data

Gathering performance tuning data



© Copyright IBM Corporation 2016

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

9.1

Figure 5-10. Gathering performance tuning data

WA8152.2

Notes:



Required performance metric

- When testing a web application, several metrics must be determined:
 - Through interpretation of data from an existing application that performs similar work
 - From best-guess estimates
- These requirements are:
 - Average request rate
 - Peak request rate
 - Average concurrent users
 - Peak concurrent users
 - Regular usage hours
 - Peak usage hours
 - Site abandonment rate

© Copyright IBM Corporation 2016

Figure 5-11. Required performance metric

WA8152.2

Notes:

Required performance metrics: Details (1 of 2)

- Average request rate
 - What is the expected number of users who access this application?
 - This number is generally expressed in hits per month, day, hour, or minute, depending on volumes
 - This rate should be reevaluated regularly
- Peak request rate
 - How many pages need to be served per second?
 - This rate also should be reevaluated regularly
- Average number of concurrent users
 - What is the average number of users who access the application at the same time during regular usage hours?
 - This number should be planned for, expected, and reevaluated regularly

© Copyright IBM Corporation 2016

Figure 5-12. Required performance metrics: Details (1 of 2)

WA8152.2

Notes:



Required performance metrics: Details (2 of 2)

- Peak concurrent users
 - Maximum number of concurrent users that visit your site during peak time
- Regular usage hours
 - Defines your off-peak hours
 - Required to simulate a realistic workload
- Peak usage hours
 - Time when most traffic occurs and a performance degradation impacts most of your users
- Site abandonment rate
 - How long does a user stay on your page before they leave the site or closes the browser?

© Copyright IBM Corporation 2016

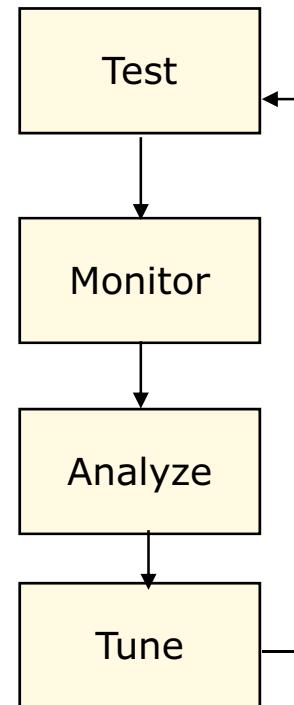
Figure 5-13. Required performance metrics: Details (2 of 2)

WA8152.2

Notes:

The lifecycle of performance monitoring and tuning

- Load testing with the stress tools and benchmarking applications
 - Rational Performance Tester, Jmeter, DayTrader
- Collecting application server data
 - PMI, request metrics, ITCAM
- Displaying application server data
 - Tivoli Performance Viewer, Performance monitoring servlet, user-written PMI clients
- Analyzing application server data
 - Performance Advisors
- Tuning the application servers
 - IBM Support Assistant tools
 - Health Center



© Copyright IBM Corporation 2016

Figure 5-14. The lifecycle of performance monitoring and tuning

WA8152.2

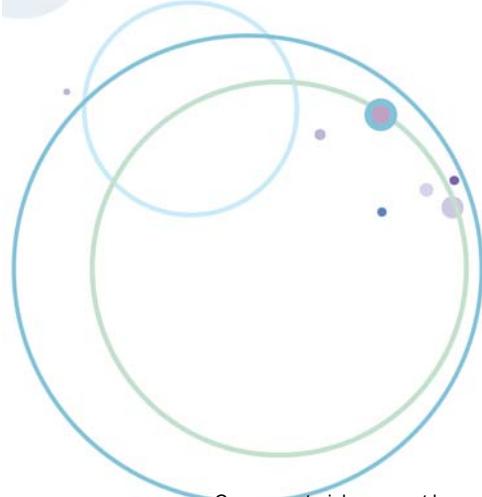
Notes:

The lifecycle of performance monitoring and tuning includes the following steps:

- Load testing with the stress tools and benchmarking applications by using tools such as Rational Performance Tester, JMeter, Load Runner, DayTrader.
- Collect application server data by using PMI, request metrics, ITCAM.
- Display application server data by using Tivoli Performance Viewer, Performance monitoring servlet, user-written PMI clients.
- Analyze application server data. The Tivoli Performance Viewer Performance Advisors might be helpful.
- Tune the application servers. The Java Health Center might be helpful.

5.3. Top 10 monitoring hot list for WebSphere performance tuning

Top 10 monitoring hot list for WebSphere performance tuning



© Copyright IBM Corporation 2016

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

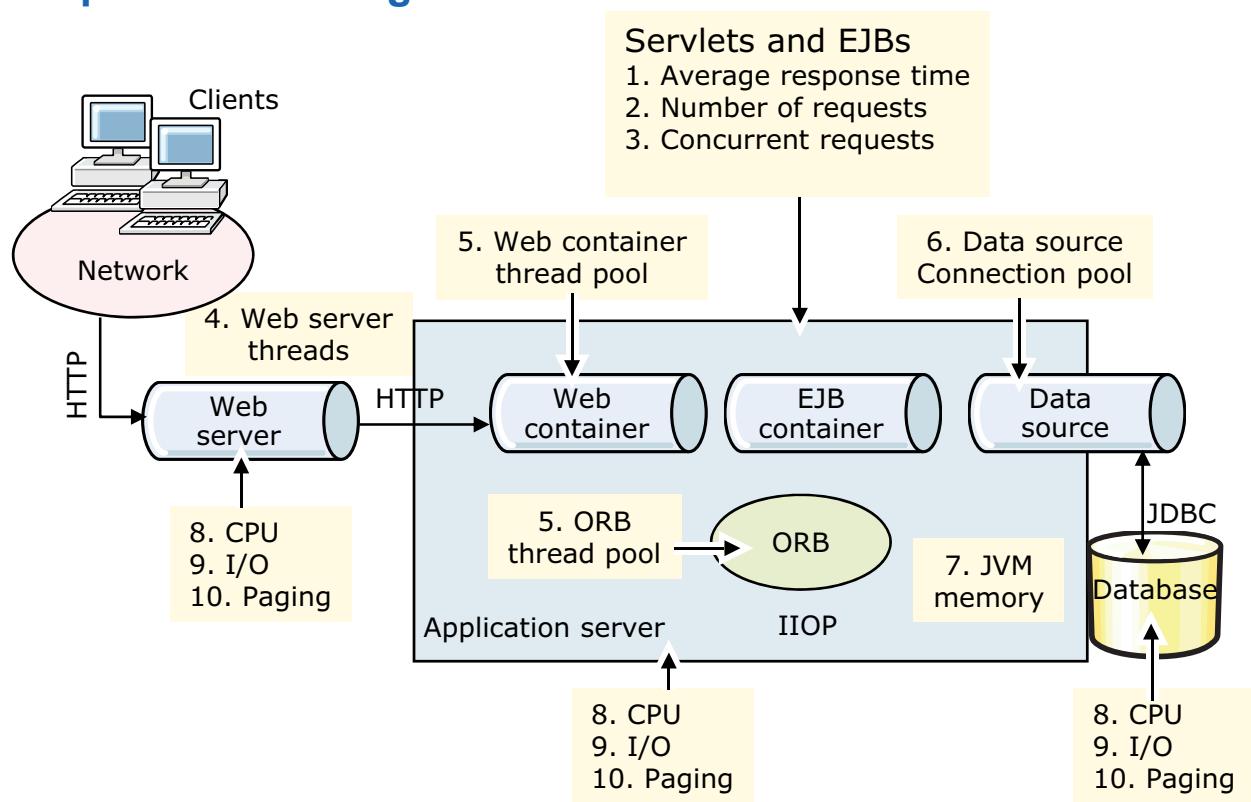
9.1

Figure 5-15. Top 10 monitoring hot list for WebSphere performance tuning

WA8152.2

Notes:

Top 10: Monitoring hot list



© Copyright IBM Corporation 2016

Figure 5-16. Top 10: Monitoring hot list

WA8152.2

Notes:

From end-to-end, web browser client to back-end database, the top 10 list of things to monitor are:

1. Average response time for servlets and EJBs
2. Number of client requests for servlets and EJBs
3. Concurrent requests for servlets and EJBs
4. Number of web server threads used
5. Size of the web container thread pool
6. Size of data source connection pools
7. Size of JVM heap memory
8. CPU usage
9. Operating system I/O
10. OS paging

General tuning considerations (1 of 2)

- Because of the sheer magnitude of monitors and tuning parameters, it is difficult to know where to start
 - What should you monitor?
 - Which component should you tune first?
- When you experience performance problems:
 - Use the Tivoli Performance Viewer to walk through the top 10 monitoring items checklist
- Use the Performance Advisors together with Tivoli Performance Viewer
 - Load test your application
 - Monitor items by using Tivoli Performance Viewer and Health Center
 - Gather and implement performance advice

© Copyright IBM Corporation 2016

Figure 5-17. General tuning considerations (1 of 2)

WA8152.2

Notes:

General tuning considerations (2 of 2)

- Make sure that you have a comprehensive understanding of the environment, the applications, and all components involved
- Before starting with WebSphere tuning, ensure that you do not have any performance bottlenecks at the OS level
 - Paging
 - Disk I/O
 - Network
- Focus your tuning actions on extraordinary behaviors such as
 - An over-utilized thread pool
 - A JVM that spends 50% of its time in garbage collection during peak load
- Perform monitoring when the system is under typical production level load and record the observed values
 - Record Tivoli Performance Viewer sessions and save to the file system
 - Use the monitoring data for recurring analysis

© Copyright IBM Corporation 2016

Figure 5-18. General tuning considerations (2 of 2)

WA8152.2

Notes:

Remember: A set of tuning parameters that work well for one application might not work the same way for another application. Tuning is application and environment-specific.

Server tuning cannot compensate for a poorly written application.



How to find metrics for servlets and EJBs (1 of 2)

- Average servlet response time
 - In Tivoli Performance Viewer, click **Performance Modules > Web Applications** and view the value that is named `ServiceTime`
- Number of servlet requests (transactions)
 - In Tivoli Performance Viewer, click **Performance Modules > Web Applications** and view the value that is named `RequestCount`
- Concurrent servlet requests (extended statistics set)
 - In Tivoli Performance Viewer, click **Performance Modules > Web Applications** and view the value that is named `ConcurrentRequests`
- Live number of HTTP sessions
 - In Tivoli Performance Viewer, click **Performance Modules > Servlet Session Manager** and view the value that is named `LiveCount`

© Copyright IBM Corporation 2016

Figure 5-19. How to find metrics for servlets and EJBs (1 of 2)

WA8152.2

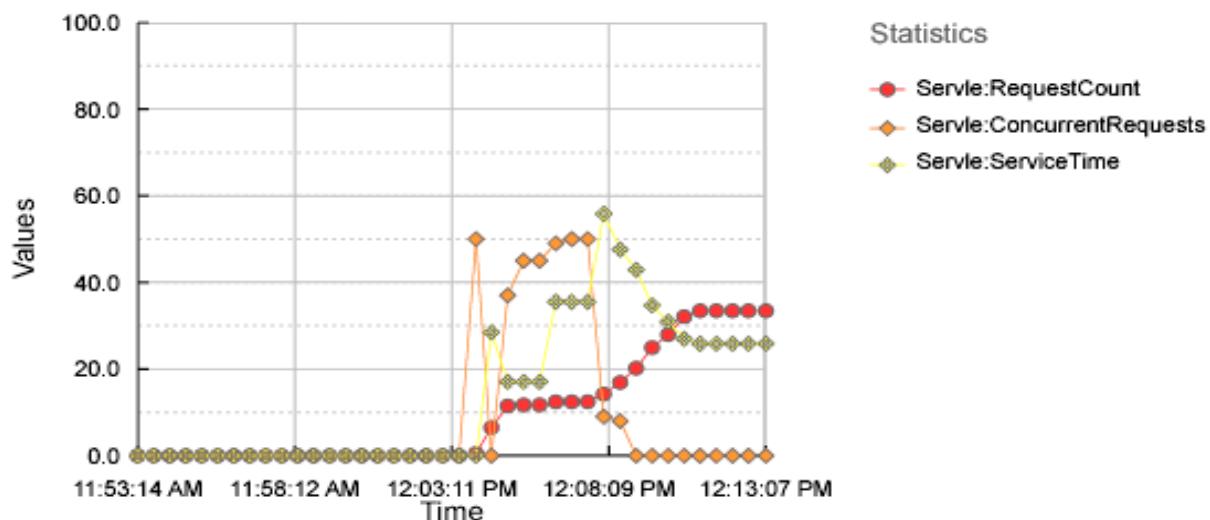
Notes:

Definitions:

- **ServiceTime:** The average response time, in milliseconds, in which a servlet request is finished.
- **RequestCount:** The total number of requests that a servlet processed.
- **ConcurrentRequests:** The number of requests that are concurrently processed.
- **LiveCount:** The total number of sessions that are currently live.

How to find metrics for servlets and EJBs (2 of 2)

- From Tivoli Performance Viewer, click **Performance Modules > Web Applications**



© Copyright IBM Corporation 2016

Figure 5-20. How to find metrics for servlets and EJBs (2 of 2)

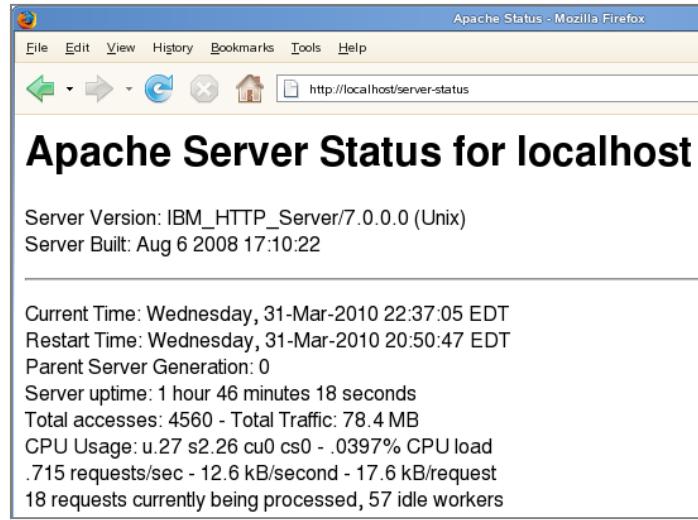
WA8152.2

Notes:

This graph was generated by running a load test of 50 current users. As you can see from the graph, the test ran for approximately 7 minutes, during which the service time gradually increased.

How to find metrics for thread pools and data sources

- Web server threads
 - Depends on the web server; consult the web server manuals
 - For IBM and Apache HTTP Server, use the "server-status" page
- Web container and EJB container thread pool
 - In Tivoli Performance Viewer, look at the Thread Pools summary report
- Data source connection pool size
 - In Tivoli Performance Viewer, look at the Connection Pools summary report
- To see the summary reports, open Tivoli Performance Viewer and expand Summary in the navigation tree



© Copyright IBM Corporation 2016

Figure 5-21. How to find metrics for thread pools and data sources

WA8152.2

Notes:

Finding web server threads is dependent upon the web server. Consult the web server manuals.

Use the Thread Pools summary report in Tivoli Performance Viewer to locate web container and EJB container thread pool.

Data source connection pool size can be found in the Connection Pools summary report.

To see the summary reports, open Tivoli Performance Viewer and expand Summary in the navigation tree.

For more information about tuning web servers, see the following article:

http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/index.jsp?topic=%2Fcom.ibm.websphere.nd.doc%2Fae%2Ftprf_tunewebserv.html



How to find metrics for JVM and system resources

JVM memory and garbage collection statistics

- Some metrics depend on JVMTI
 - Add `-agentlib:pmiJvmtiProfiler` as a Generic JVM argument
- In Tivoli Performance Viewer, go to **Performance Modules > JVM Runtime**
- Enable verbose GC for the application server
 - GC data is written to `native_stderr.log`
- System resources on web, application, and database servers
 - CPU utilization
 - Disk and network I/O
 - Paging activity
 - Use OS-provided tools or third-party monitoring tools for system resource monitoring
 - Per JVM process CPU utilization in Tivoli Performance Viewer Basic statistics set (`ProcessCpuUsage`)

© Copyright IBM Corporation 2016

Figure 5-22. How to find metrics for JVM and system resources

WA8152.2

Notes:

5.4. Detecting bottlenecks

Detecting bottlenecks



© Copyright IBM Corporation 2016

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

9.1

Figure 5-23. Detecting bottlenecks

WA8152.2

Notes:

Detecting bottlenecks (1 of 2)

- Always monitor OS resources:
 - CPU
 - Memory
 - I/O
- In general, your systems are CPU bound, meaning that adding resources (another CPU or node) can increase throughput
- If CPUs are not fully used under load, requests are probably waiting on some other resource
 - Are you sure that the HTTP server is running enough threads?
 - Database server might be undersized and therefore lowers overall performance
- Does the application establish network connections to external servers?
 - Those servers might be slow in response or not available on the network
 - Slow transactions on back-end systems consume application server resources (for example, web container threads)
 - Accommodate for such extra threads in thread pools or use async beans in code

© Copyright IBM Corporation 2016

Figure 5-24. Detecting bottlenecks (1 of 2)

WA8152.2

Notes:

Always monitor OS resources: CPU, memory, and I/O. Ideally, your systems are CPU bound, meaning that adding resources (for example, another CPU or another node) increases throughput.

CPU bound refers to a condition when the speed of the CPU determines the time that a computer takes to complete a task. During the task, CPU utilization is high or at 100% for several seconds. Interrupts that are issued by peripheral devices are processed slowly, or indefinitely delayed. When you determine that a computer is often CPU bound, this condition implies that upgrading the CPU can improve the overall computer performance.

If CPUs are not fully used under load, requests are probably waiting on some other resource. Are you sure that the HTTP server is running enough threads? The database server might be undersized and lowering overall performance.

Does the application establish network connections to external servers? Those servers might be slow in response or not available on the network at all. Slow transactions on back-end environments eat up resources (for example, web container threads).

Accommodate for such extra threads in thread pools or use async beans in code. What about disk I/O? Too much logging can kill performance. Be sure that you check the network performance.

Detecting bottlenecks (2 of 2)

- What about disk I/O?
 - Too much logging can kill performance
- Check the network performance
- If CPU is fully used, does every process get enough CPU time?
 - Too many services and processes on a single machine compete for CPU time
 - SSL encryption of many connections consumes CPU resources
 - Bad application coding or bugs might consume CPU
 - Other running services (not WebSphere Application Server-related)
- Bottom line: Adding CPUs should help

© Copyright IBM Corporation 2016

Figure 5-25. Detecting bottlenecks (2 of 2)

WA8152.2

Notes:

If CPU is fully used, does every process get enough CPU time?

- Too many services or processes on a single machine compete for CPU time
- SSL encryption of many connections eats up CPU resources
- Bad application coding
- Bugs in code
- Other running services (not WebSphere Application Server related)

Adding CPUs should help.

Bottleneck considerations (1 of 4)

- Oversized queues and thread pools
 - Too many concurrent requests can flood the system and performance drops
- Lowering concurrent request processing usually gives better response times and thus increases throughput
- HTTP server is not accepting new connections
 - Increasing HTTP server processes (threads) might not help
 - Just postpones the problem
- Find the root cause of the bottleneck
 - Poor database performance
 - Security: poor performance because of a slow authentication (LDAP) server
 - Garbage collection might be taking too long on application servers; tune the JVM heap
 - Hanging (synchronous) backend connections

© Copyright IBM Corporation 2016

Figure 5-26. Bottleneck considerations (1 of 4)

WA8152.2

Notes:

Oversized queues and thread pools: Too many concurrent requests can flood the system and performance drops.

Lower concurrent request processing usually gives better response times and thus increases throughput and overall performance.

HTTP server is not accepting new connections: Increasing HTTP server processes or threads might not help it can merely postpone the problem.

Find the root cause:

- Poor database performance
- Security: low performance because of slow authentication server
- Garbage collection is taking too long on application servers: tune the JVM heap
- Hanging (synchronous) back-end connections

Bottleneck considerations (2 of 4)

Database server

- Have your database administrator check database performance
- Most database servers have performance introspection tools
- CPU, memory, I/O performance
 - Databases perform better with multiple, fast disks and CPUs
 - DB cache ratio should be high
- Watch for long-running SQL queries
 - SQL queries in application code can be formulated badly
 - Use DB tools to analyze your statements (for example, `db2 explain`)
 - Index creation can help

OS tuning

- Tune the OS on each system as demanded by its usage
- Tuning OS for WebSphere and web server is different than for database servers

© Copyright IBM Corporation 2016

Figure 5-27. Bottleneck considerations (2 of 4)

WA8152.2

Notes:

Tuning OS for WebSphere and web server is different than for database servers. For example, tune the AIX memory manager: A DB system does not need large filesystem/buffer caches as provided by the AIX memory manager. DB software manages caches better itself.

What is DB2 `explain`? When an SQL is executed against a DB2 database, the DB2 Optimizer tool defines the path that is used to access the data. The access path is defined based on table statistics that are generated by DB2 runstats tool.

The `explain` command provides details of the access path and allows you to analyze how the data is accessed. Knowing these details, you might be able to improve the command's performance.

Bottleneck considerations (3 of 4)

- Storage considerations
 - Extensive logging on each node and component can kill performance
 - Set appropriate (low) log level for load testing and production machines
 - Watch out for high disk I/O on web servers and application servers
 - Use SAN storage if necessary
- Caching
 - Can be done in various places
 - Understand when and which type of caching to use
 - Do you serve static content from the application servers?
 - Perhaps a caching proxy server can speed up things without application changes

© Copyright IBM Corporation 2016

Figure 5-28. Bottleneck considerations (3 of 4)

WA8152.2

Notes:

Storage considerations include:

- Extensive logging on each node and component can kill performance.
- Set appropriate (low) log level for load testing and production.
- Watch out for high disk I/O on web servers and application servers.
- Using SAN storage is becoming increasingly popular, but it also becomes a black box. You need a SAN storage expert to diagnose problems.

A storage area network (SAN) is a high-speed subnetwork of shared storage devices. A storage device is a dedicated machine that contains one or more disks for storing data. A SAN's architecture works such that all storage devices are available to all servers on a LAN or WAN.

Bottleneck considerations (4 of 4)

- Bugs in application code are easy to make, but hard to find
 - Root cause analysis is best done together with developers
 - “You cannot tune your way out of a bad application”
- Use a monitoring tool that shows historical data
 - Compare system performance for different application releases
 - Look for changes in resource usage
 - Use OS monitoring tools as well as web server, application server, and database server monitoring tools
- Still having performance problems?
 - Use a profiler to look into the application code
 - Perform execution-time analysis to find out which parts of application take too long to execute
 - Profiling requires Java development and application knowledge

© Copyright IBM Corporation 2016

Figure 5-29. Bottleneck considerations (4 of 4)

WA8152.2

Notes:

Bugs in application code are easy to make but hard to find. Root cause analysis is best done together with developers. But remember that you cannot tune your way out of a bad application.

Use a monitoring tool that shows historical data. Compare system performance at different application releases during production phase to spot changes in resource usage. Use OS monitoring tools and web, WebSphere, or DB monitoring tools.

If you are still having performance problems, use a profiler to look into the application code. Perform execution-time analysis to find out which parts of your application execute too long and dig down deeper. Proper analysis requires a high skill level in Java development and application knowledge.

5.5. System queues

System queues



© Copyright IBM Corporation 2016

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

9.1

Figure 5-30. System queues

WA8152.2

Notes:



System queues: The queuing network

- WebSphere Application Server establishes a queuing network
- Goal: You must accept a high number of client connections, but the system usually performs better if not all clients are processed simultaneously
- The queuing network represents queues of requests that are waiting to use a resource
- The queues include:
 - Network
 - Web server
 - Web container
 - EJB container
 - Data source connection pools

© Copyright IBM Corporation 2016

Figure 5-31. System queues: The queuing network

WA8152.2

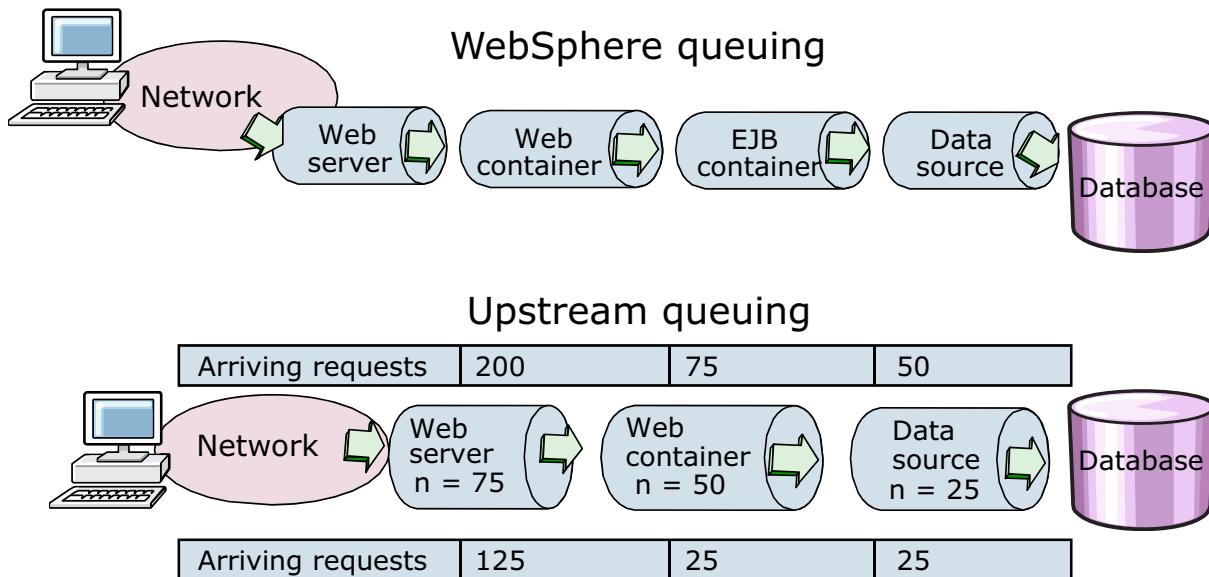
Notes:

WebSphere Application Server establishes a queuing network. An application needs to accept a high number of client connections, but usually the system performs better if not all clients are processed simultaneously.

The queuing network represents queues of requests that are waiting to use a resource. The queuing network includes the network, web server, web container, EJB container, and data source connection pools.

WebSphere tuning: Upstream queuing

- Upstream queuing attempts to allow more work to be done by limiting the number of connections at each tier of the application



© Copyright IBM Corporation 2016

Figure 5-32. WebSphere tuning: Upstream queuing

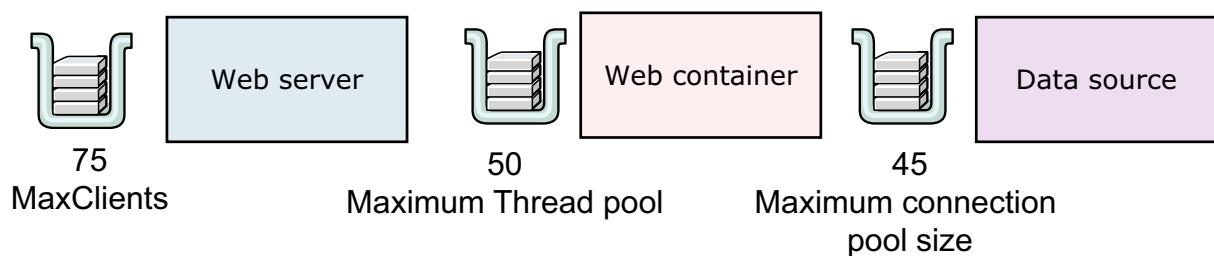
WA8152.2

Notes:

Design the queuing system to allow as much work on the database and application server as possible without over-stressing each area. It is important not to overload any one tier because then you have poor performance due to waiting threads. This result is one of the most difficult things to achieve during performance testing.

How to configure system queues (1 of 2)

- Most users should wait in the network
- Specify the queues farthest upstream (closest to the client) to be slightly larger
- Specify the queues farther downstream (farthest from the client) to be progressively smaller



© Copyright IBM Corporation 2016

Figure 5-33. How to configure system queues (1 of 2)

WA8152.2

Notes:

Most users should wait in the network; therefore, specify the queues farthest upstream (closest to the client) to be slightly larger, and specify the queues farther downstream (farthest from the client) to be progressively smaller. For example, given a Maximum Application Concurrency value of 48, start with system queues at the following values: web server 75, web container 50, data source 45. Perform a set of extra experiments by adjusting these values slightly higher and lower to find the best settings.

How to configure system queues (2 of 2)

- IBM HTTP Server
 - In recent versions, the queue mechanism of the web server is greatly improved in its scalability, and tuning can focus primarily on the web server itself
 - MaxClients for UNIX and Linux is the maximum number of simultaneous client connections
 - Use ThreadsPerChild for Windows systems
- Web container
 - Thread pool maximum size
 - HTTP transport channel: maximum persistent requests
- ORB (object request broker)
 - Thread pool maximum size
- Data source
 - Maximum connection pool size
 - Prepared statement cache size

© Copyright IBM Corporation 2016

Figure 5-34. How to configure system queues (2 of 2)

WA8152.2

Notes:

IBM HTTP Server: In V6, the queue mechanism of the web server was greatly improved in its scalability, and tuning can focus primarily on the web server itself. Configure the MaxClients parameter for UNIX and the ThreadsPerChild for Windows systems.

Web container: Configure thread pool maximum size and HTTP transport channel, maximum persistent requests parameters.

ORB (object request broker): Configure thread pool maximum size.

Data source: Configure connection pool size and prepared statement cache size.



Determining optimum queue sizes example (1 of 2)

- Procedure:
 - Perform a number of load tests against the application server environment with large queue settings to ensure maximum concurrency through the system
- Set the queue sizes for the web server, web container, and data source to an initial value
 - For example, 100

Simulate many typical user interactions that concurrent users enter in an attempt to fully load the WebSphere environment

- In this context, concurrent users means simultaneously active users who:
 - Send a request
 - Wait for the response
 - Immediately resend a new request

© Copyright IBM Corporation 2016

Figure 5-35. Determining optimum queue sizes example (1 of 2)

WA8152.2

Notes:

The procedure should be to perform a number of load tests against the application server environment with large queue settings to ensure maximum concurrency through the system.

1. Set the queue sizes for the web server, web container, and data source to some initial value, for example 100.
2. Simulate many typical user interactions that are entered by concurrent users in an attempt to fully load the WebSphere environment.

In this context, “concurrent users” means simultaneously active users that send a request, wait for the response, and immediately resend a new request upon response reception, without think time.

Determining optimum queue sizes example (2 of 2)

- Use any load testing tool to simulate this workload
- Measure overall throughput
 - Determine at what point the system capabilities are fully stressed (the saturation point)
- Repeat the process, increasing the user load each time
- After each run, record:
 - Throughput (requests per second)
 - Response times (seconds per request)
 - Plot the throughput curve

© Copyright IBM Corporation 2016

Figure 5-36. Determining optimum queue sizes example (2 of 2)

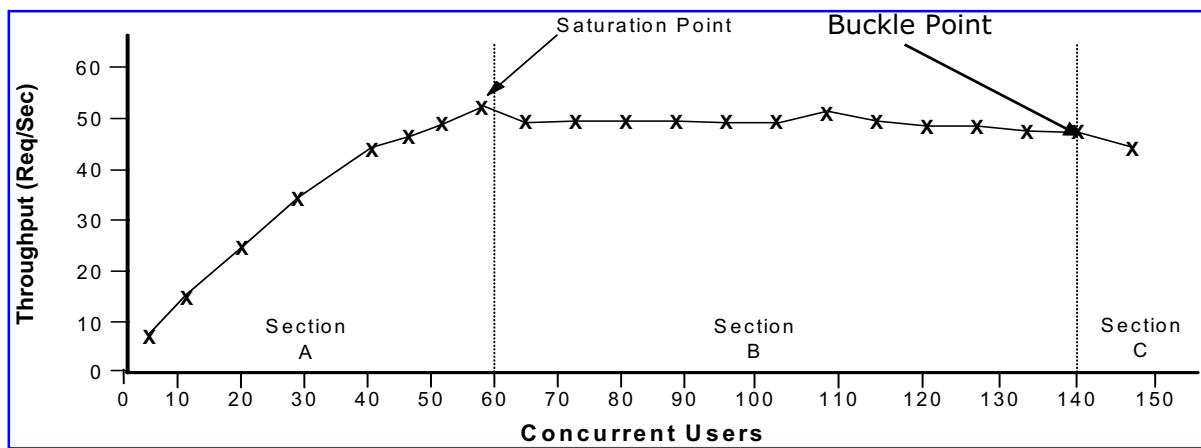
WA8152.2

Notes:

1. Use any stress tool to simulate this workload.
2. Measure overall throughput and determine at what point the system capabilities are fully stressed (the saturation point).
3. Repeat the process, each time and increase the user load.
4. After each run, record the throughput (requests per second) and response times (seconds per request) and plot the throughput curve.

Determining optimum queue sizes: The throughput curve

- Throughput is a function of the number of concurrent requests present in the total system
- At some load point:
 - Congestion starts to develop because of a bottleneck
 - Throughput increases at a much lower rate until a saturation point is reached (maximum throughput value)
- The throughput curve should help you identify this load point



© Copyright IBM Corporation 2016

Figure 5-37. Determining optimum queue sizes: The throughput curve

WA8152.2

Notes:

Section A=Lighter user load zone, Section B=Heavy load zone, Section C=Buckle zone

The throughput maximum (saturation point) usually represents a bottleneck that might be a “saturated” resource. The CPU can often become the constraining resource. When your CPUs reach 100%, the server does not have the capacity to handle more requests.

As client load increases in the heavy load zone, throughput remains nearly constant, but the response time increases in proportion to the user load.

At some point (the buckle point) one of the system components becomes exhausted, and throughput starts decreasing. For example, the buckle point might be reached when the network connections to your web server exceed the limits of the network adapter, or your OS might run out of file handles.

Determining the maximum concurrency point

- The number of concurrent users at the throughput saturation point represents the maximum concurrency of the application
 - For example, if the application saturated the application server at 50 users, 48 users might give the best combination of throughput and response time
- This value is called the maximum application concurrency value
- Maximum application concurrency becomes the preferred value for adjusting the WebSphere Application Server system queues

© Copyright IBM Corporation 2016

Figure 5-38. Determining the maximum concurrency point

WA8152.2

Notes:

The number of concurrent users at the throughput saturation point represents the maximum concurrency of the application. For example, if the application saturated the application server at 50 users, 48 users might give the best combination of throughput and response time. This value is called the Maximum Application Concurrency value.

Maximum Application Concurrency becomes the preferred value for adjusting the WebSphere Application Server system queues.

Unit summary

Having completed this unit, you should be able to:

- Describe performance tuning goals, strategies, and considerations
- Evaluate a list of the top 10 monitoring considerations
- Gather performance data for WebSphere Application Server components
- Detect performance bottlenecks
- Determine optimum queue sizes
- Configure WebSphere Application Server system queues
- Identify and evaluate WebSphere tuning parameters

© Copyright IBM Corporation 2016

Figure 5-39. Unit summary

WA8152.2

Notes:

Checkpoint questions

1. True or False: Performance bottlenecks at the OS level include Java heap size and database connection pool size.
2. True or False: Site abandonment rate refers to how long a user stays on a web page before they leave the site or closes the browser.
3. True or False: The throughput of WebSphere Application Server is a function of the number of concurrent requests present in the total system.
4. True or False: Upstream queuing attempts to allow more work to be done by using equal queue and thread pool sizes along the request flow.

© Copyright IBM Corporation 2016

Figure 5-40. Checkpoint questions

WA8152.2

Notes:

Write your answers here:

- 1.
- 2.
- 3.
- 4.



Checkpoint answers

1. False: Performance bottlenecks at the OS level include paging, swapping, and Disk I/O.
2. True
3. True
4. False: Upstream queuing attempts to allow more work to be done by limiting the number of connections at each tier of the application.

© Copyright IBM Corporation 2016

Figure 5-41. Checkpoint answers

WA8152.2

Notes:

Unit 6. Introduction to the JVM

What this unit is about

This unit provides an overview of the JVM and garbage collection policies.

What you should be able to do

After completing this unit, you should be able to:

- Describe components of the JVM and overall architecture
- Describe JIT and AOT compilers
- Describe the garbage collection process
- Describe garbage collection policies such as generational concurrent and optimize for throughput
- Describe JVM heap memory usage

How you will check your progress

- Checkpoint
- Machine exercise

References

IBM Developer Kit and Runtime Environment, Java Technology Edition,
Version 6 Diagnostics Guide

<http://download.boulder.ibm.com/ibmdl/pub/software/dw/jdk/diagnosis/diag60.pdf>

WebSphere Application Server ND V8.5 Information Center article: Tuning
the JVM

http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/index.jsp?topic=%2Fcom.ibm.websphere.nd.doc%2Fae%2Fcontainer_tune_jvm.html

Unit objectives

After completing this unit, you should be able to:

- Describe components of the JVM and overall architecture
- Describe JIT and AOT compilers
- Describe the garbage collection process
- Describe garbage collection policies such as generational concurrent and optimize for throughput
- Describe JVM heap memory usage

© Copyright IBM Corporation 2016

Figure 6-1. Unit objectives

WA8152.2

Notes:



Topics

- JVM introduction
- Garbage collection
- JVM heap memory usage

© Copyright IBM Corporation 2016

Figure 6-2. Topics

WA8152.2

Notes:

6.1. JVM introduction

JVM introduction



© Copyright IBM Corporation 2016

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

9.1

Figure 6-3. JVM introduction

WA8152.2

Notes:

Java virtual machine (JVM) features

Almost every WebSphere process runs in a JVM

The JVM provides:

- Class loading
 - A Classloader verifies and loads classes into memory
 - Multiple Classloaders are involved in loading the required libraries for an application to run
 - Each Classloader loads its own classes
- Garbage collection (GC)
 - Garbage collection takes care of memory management for the entire application server
 - The GC process searches memory to reclaim space from program segments or inactive Java objects
- Execution management
 - Manages the bookkeeping work for all the Java threads
- Execution engine (Interpreter)
 - Interprets the Java methods

© Copyright IBM Corporation 2016

Figure 6-4. Java virtual machine (JVM) features

WA8152.2

Notes:

The Java virtual machine (JVM) is an interpretive computing engine that is responsible for running the bytecode in a compiled Java program. The JVM translates the Java bytecode into the native instructions of the host computer. The application server, being a Java process, requires a JVM to run and to support the Java applications that are running on it. JVM settings are part of an application server configuration.

It is called “virtual” because it provides an interface that is independent of the underlying operating system and computer hardware architecture. This independence from hardware and operating system is a cornerstone of the write-once-run-anywhere value of Java programs. Java programs are compiled into bytecode that targets the abstract virtual machine; the JVM is responsible for running the bytecode on the specific operating system and hardware combinations.

Just-in-time compiler (JIT) basics

- The just-in-time compiler (JIT) is essential for a high-performing Java application
 - Java is write-once-run-anywhere
 - It is interpretive by nature and without the JIT, cannot compete with native code applications
- The JIT compiler is enabled by default for an application server
- The JIT works by compiling bytecode that is loaded from the Classloader when an application accesses it
 - Because different operating systems have different JIT compilers, there is no standard procedure for when a method is compiled
 - As your code accesses methods, the JIT determines how frequently specific methods are accessed
 - Methods that are used often are compiled to optimize performance

© Copyright IBM Corporation 2016

Figure 6-5. Just-in-time compiler (JIT) basics

WA8152.2

Notes:

All the JVMs that are currently used commercially come with a supplementary compiler that takes bytecode and outputs platform-dependent computer code. This compiler works with the JVM to select parts of the Java program that would benefit from the compilation of bytecode. It replaces these areas of bytecode with concrete code for the JVM. This process is called just-in-time (JIT) compilation.

Ahead-Of-Time (AOT) compiler basics

- Ahead-Of-Time (AOT) compiles Java classes into native code for subsequent executions of the same program
 - The AOT compiler works with the class data sharing framework
- The AOT compiler generates native code dynamically while an application runs and caches any generated AOT code in the shared data cache
 - Subsequent JVMs that run the method, can load and use the AOT code from the shared data cache without incurring the performance decrease experienced with JIT-compiled native code
- The AOT compiler is enabled by default, but is only active when shared classes are enabled
 - By default, shared classes are disabled so that no AOT activity occurs
 - The `-Xshareclasses` command-line option can be used to enable shared classes
- When the AOT compiler is active, the compiler selects the methods to be AOT compiled with the primary goal of improving startup time

© Copyright IBM Corporation 2016

Figure 6-6. Ahead-Of-Time (AOT) compiler basics

WA8152.2

Notes:

Because AOT code must persist over different program executions, AOT-generated code does run as fast as JIT-generated code. AOT code usually runs faster than interpreted code.

In a JVM without an AOT compiler or with the AOT compiler disabled, the JIT compiler selectively compiles frequently used methods into optimized native code.

A time cost is associated with compiling methods because the JIT compiler operates while the application is running. Because methods begin by being interpreted and most JIT compilations occur during startup, startup times can be increased.

Startup performance can be improved by using the shared AOT code to provide native code without compiling. There is a small time cost to load the AOT code for a method from the shared data cache and bind it into a running program. The time cost is low compared to the time it takes the JIT compiler to compile that method.

The `-Xshareclasses` option can be used to enable shared classes, which might also activate the AOT compiler if AOT is enabled.

JVM version (1 of 2)

- WebSphere supports several JVMs based on version and operating system type
 - Windows, AIX, and Linux: IBM supplied
 - Can use only IBM SDK that zWSAS provides on z/OS
 - Solaris and HP-UX: Hybrid of IBM add-ons and vendor supplied JVM
- For a comprehensive list of the supported JVMs, check
 - <http://www.ibm.com/support/docview.wss?uid=swg27038218>
- To determine the JVM version in use:
 - Look in the `SystemOut.log` file of one of the profile instances
`<profile_home>/logs/server1/SystemOut.log`

© Copyright IBM Corporation 2016

Figure 6-7. JVM version (1 of 2)

WA8152.2

Notes:

JVMs differ on the available tools that are supported and can have different behavior, even between different versions that are targeted for the same platform. Always look at the documentation for the specific JVM for behavioral descriptions and the options to control the JVM.

WebSphere Application Server V8.5.5 supports the Java Development Kit (JDK) Version 6.0 and Version 7.0.

For Oracle Solaris, WebSphere Application Server contains an embedded copy of the Oracle Solaris JVM along with some IBM add-ons, such as security, XML, and ORB packages. The WebSphere Application Server Solaris SDK is, therefore, a hybrid of Oracle and IBM products. However, the core JVM and JIT are Oracle Solaris.

For HP-UX, WebSphere Application Server contains an embedded copy of the HP JVM alongside some IBM add-ons, such as security packages. The WebSphere Application Server HP SDK is, therefore, a hybrid of HP and IBM products. However, the core JVM and JIT are HP software.

JVM version (2 of 2)

- To determine the JVM version in use:
 - Run `java -fullversion` (IBM JVMs only) from the command line

```
[root@washost bin]# ./java -fullversion
java full version "JRE 1.6.0 IBM Linux build pxa6460_26sr5fp1ifix-
20130408_02 (SR5 FP1)"
```

- Run `java -version` from the command line

```
[root@washost bin]# ./java -version
java version "1.6.0"
Java(TM) SE Runtime Environment (build pxa6460_26sr5fp1ifix-
20130408_02(SR5 FP1+IV38399+IV38578))
IBM J9 VM (build 2.6, JRE 1.6.0 Linux amd64-64 Compressed References
20130301_140166 (JIT enabled, AOT enabled)
J9VM - R26_Java626_SR5_FP1_20130301_0937_B140166
JIT - r11.b03_20130131_32403
GC - R26_Java626_SR5_FP1_20130301_0937_B140166_CMPRSS
J9CL - 20130301_140166)
JCL - 20130408_01
```

© Copyright IBM Corporation 2016

Figure 6-8. JVM version (2 of 2)

WA8152.2

Notes:

The Java virtual machine (JVM) used by the IBM SDK for Java is the IBM J9 virtual machine (J9 VM).

- JIT is just-in-time compiler.
- GC is garbage collector.
- J9CL is IBM J9 class library.
- JCL is Java Class Library.



Using IBM Java 7

- In WebSphere Application Server V8.5 and V8.5.5, IBM Java 6 is the default Java run time
 - You have the option of installing and switching to IBM Java 7
- A new command-line tool that is called `managesdk` is added to the `<profile_root>/bin` folder
- List installed run times:
 - `managesdk.sh -listAvailable`
- List run times that are configured for all profiles:
 - `managesdk.sh -listEnabledProfileAll`
- Change all configured run times to Java 7, 64-bit:
 - `managesdk.sh -enableProfileAll -sdkname 1.7_64 -enableServers`
- Change the default configured run time for future servers that are created in a profile:
 - `managesdk.sh -setNewProfileDefault -sdkname 1.7_64`

© Copyright IBM Corporation 2016

Figure 6-9. Using IBM Java 7

WA8152.2

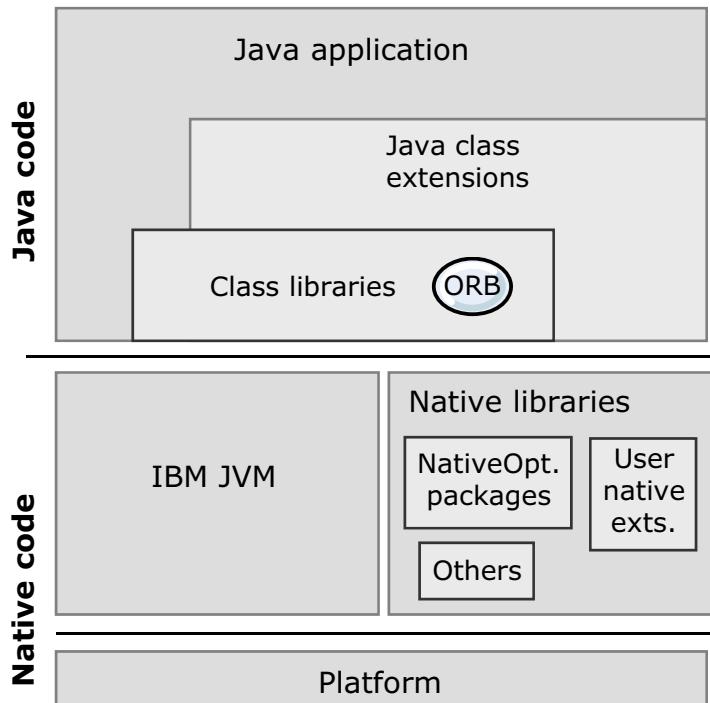
Notes:

A new command line tool that is named `managesdk` is added to the `bin` folder of WebSphere Application Server (note that you must separately install IBM Java 7 before `managesdk` lists this run time). Use `managesdk.sh -help` to see a list of all tasks and task parameters.

JVM overview

- JVM is built by using object-oriented design
- The core run times of JVMs are developed in C and run most functions in native code
 - Garbage collector and memory management
 - JIT
 - I/O subroutines, OS calls
- The J2SE and Java EE APIs all exist at the Java code layer
 - Makes data structures available
 - Gives users access to needed function
 - Allows interactions with system

Java application stack



© Copyright IBM Corporation 2016

Figure 6-10. JVM overview

WA8152.2

Notes:

A Java application uses the Java class libraries that the JRE provides to implement the application-specific logic. The class libraries, in turn, are implemented in terms of other class libraries. Eventually, the JVM provides primitive native operations. In addition, some applications must access native code directly.

The diagram on this slide shows the components of a typical Java Application Stack and the IBM JRE.

The JVM facilitates the invocation of native functions by Java applications and a number of well-defined Java Native Interface (JNI) functions for manipulating Java from native code.

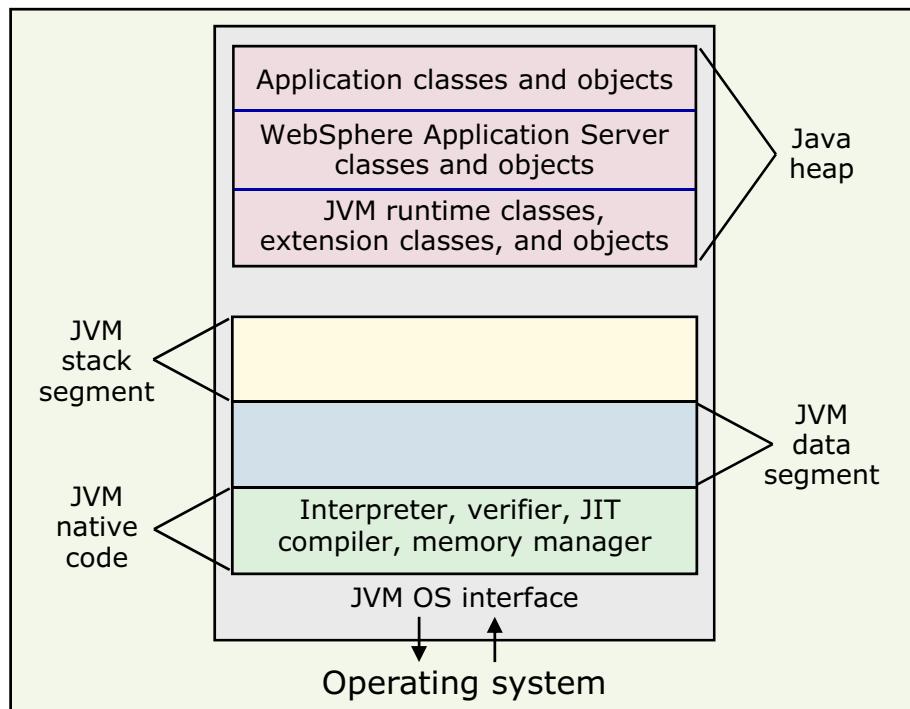
Keep in mind that WebSphere Application Server is a Java application.

The just-in-time (JIT) compiler works by compiling bytecode that is loaded from the Classloader when it is accessed by an application.

- Due to different platforms that have different JITs, there is no standard method for when a method is compiled.

- As your code accesses methods, the JIT determines how frequently specific methods are accessed; to optimize performance, it quickly compiles methods that are touched most frequently.

JVM memory segments



Typical JVM architecture

© Copyright IBM Corporation 2016

Figure 6-11. JVM memory segments

WA8152.2

Notes:

The important points to remember on this slide:

1. The JVM is a process whose memory layout depends on the OS platform. The diagram that is shown is a typical abstraction of process memory layout.
2. As a process, the JVM itself needs its own heap (JVM native heap) from which it gets its own dynamic memory needs.
3. A portion of that heap is allocated for the Java heap where classes and objects are stored. This memory is the heap that Java programmers are familiar with.
4. Garbage collection occurs only in the Java heap.
5. The Java heap can grow and shrink. When there is enough available memory in the JVM heap, the Java heap can be extended easily without acquiring memory from the OS. Otherwise, a request for more memory from the OS is made and the JVM process size increases as necessary.

The data segment contains many buffers that native memory uses for the network I/O facility and a database buffer if using a type 2 JDBC driver.

Memory that is not allocated to the Java heap is available to the native heap: Available process memory space minus the Java heap memory equals the native heap.

6.2. Garbage collection

Garbage collection



© Copyright IBM Corporation 2016

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

9.1

Figure 6-12. Garbage collection

WA8152.2

Notes:

What is garbage collection?

- Responsible for allocation and freeing of:
 - Java objects, array objects, and Java classes
- Allocates objects by using a contiguous section of the Java heap
- Ensures that the object remains on the heap while it is in use or live
 - Determination is based on a reference from another live object or from outside of the heap
- Reclaims objects that are no longer referenced
- Ensures that any `finalize` method is run before the object is reclaimed
 - It is recommended that you avoid the use of finalizers in your Java code

© Copyright IBM Corporation 2016

Figure 6-13. What is garbage collection?

WA8152.2

Notes:

Every Java class inherits the `finalize()` method from `java.lang.Object`. The garbage collector calls this method when it determines that no more references to the object exist.

However, avoid finalizers. You cannot guarantee when a finalizer runs, and often they cause problems. If you do use finalizers, try to avoid allocating objects in the finalizer method. A **verbose GC** trace shows whether finalizers are called.

Object reclamation (garbage collection)

Occurs under two scenarios:

- An allocation failure (af)
 - An object allocation is requested and not enough contiguous memory is available
- A programmatically requested garbage collection cycle
 - Call is made to `System.gc()` or `Runtime.gc()`
 - The distributed garbage collector is running
 - Call to JVMTI is made
- Two main technologies are used to remove the garbage:
 - Mark and sweep collector
 - Copy collector
- IBM uses a mark and sweep collector
 - Or a combination of mark, sweep, and copy for generational concurrent (gencon)

© Copyright IBM Corporation 2016

Figure 6-14. Object reclamation (garbage collection)

WA8152.2

Notes:

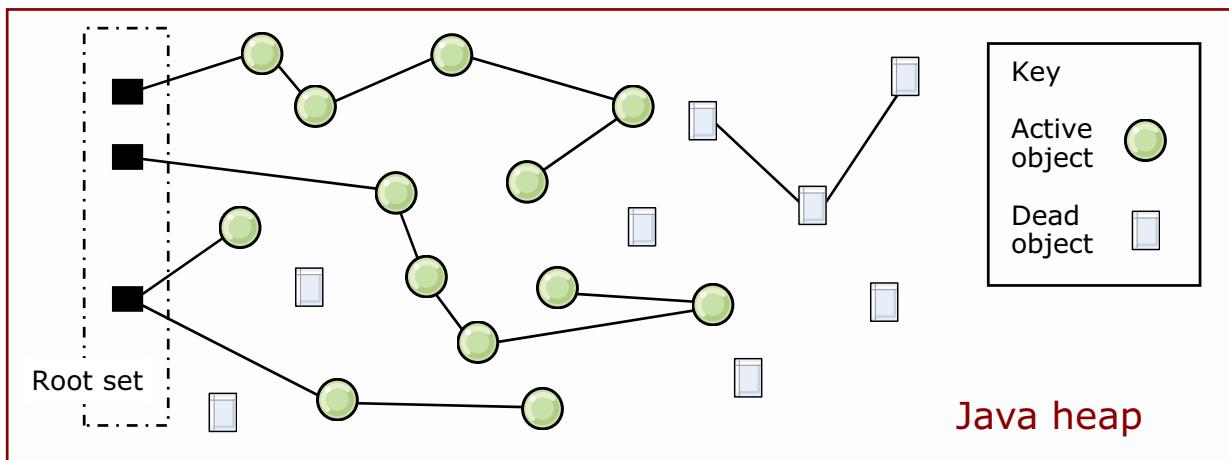
`System.gc()` internally calls `Runtime.gc()`. It is used to schedule a call to the garbage collector. A distributed garbage collector is used to collect objects in a distributed (client/server) environment.

The JVM tool Interface (JVMTI) is a new native programming interface. It provides a way to inspect the state and to control the execution of applications that are running in the JVM. JVMTI supports tools that need to access the state of the JVM, including profiling, debugging, monitoring, and thread analysis tools.

A `ForceGarbageCollection()` call can force the JVM to perform a garbage collection.

Understanding garbage collection (GC)

- JVM manages the Java heap and does garbage collection
- Object is eligible for GC when there are no more references from root to that object
- Roots set: references in the stack and registers
- Reference is dropped when variable goes out of scope



© Copyright IBM Corporation 2016

Figure 6-15. Understanding garbage collection (GC)

WA8152.2

Notes:

The active state of the JVM is made up of the set of stacks that represent the threads, the static objects that are inside Java classes, and the set of local and global JNI references.

The GC scans the thread stacks and looks for pointers that appear to be pointing to Java objects (that is, within the Java heap range). It is likely that there are attributes (for example, floats) being pushed onto the stack that look like Java object references; that is why GC marks them as “dosed” to make them unmovable during compaction.

The root set also includes pinned objects, JNI references, and more.

All the objects that pointers reference are reachable objects; they are listed in a mark vector, which is used to compare with the allocbits vector. The allocbits vector contains the information the GC needs for all of the objects created. From the difference between the mark vector and the allocbits vector, the GC can determine what needs to be collected.

IBM JDK GC process

- **Mark:** Recursively marks all the live objects, starting with the registers and thread stacks
- **Sweep:** Frees all the objects that were not marked in the mark phase
- **Compaction:** Reduces heap fragmentation
 - This phase attempts to move all live objects to one end of the heap, freeing up large areas of contiguous free space at the other end
 - Compaction stops JVM activity while it occurs
 - Not every GC cycle results in a compaction
- **Parallel** mark and sweep process uses main and multiple helper threads (number of processors minus one) to process tasks
- **Concurrent** mark and sweep can be configured
 - It starts a concurrent marking and sweeping phase before the heap is full
 - The mark and sweep phase runs while the application is still running

© Copyright IBM Corporation 2016

Figure 6-16. IBM JDK GC process

WA8152.2

Notes:

The IBM Garbage Collector is by default a “stop-the-world” (STW) operation because all application threads are stopped while the garbage is collected.

Concurrent mark can be configured. It starts a concurrent marking phase before the heap is full. The mark phase runs while the application is still running, in effect, attempting to trade application performance for possible smaller garbage collection times.

Mark stack overflow (MSO) is a rare event that can occur. Because the mark stack has a fixed size, it can overflow. This overflow has a negative impact on pause time:

- Mark process resumes where the overflow occurred
- Repeats <n> times until no more objects that require marks

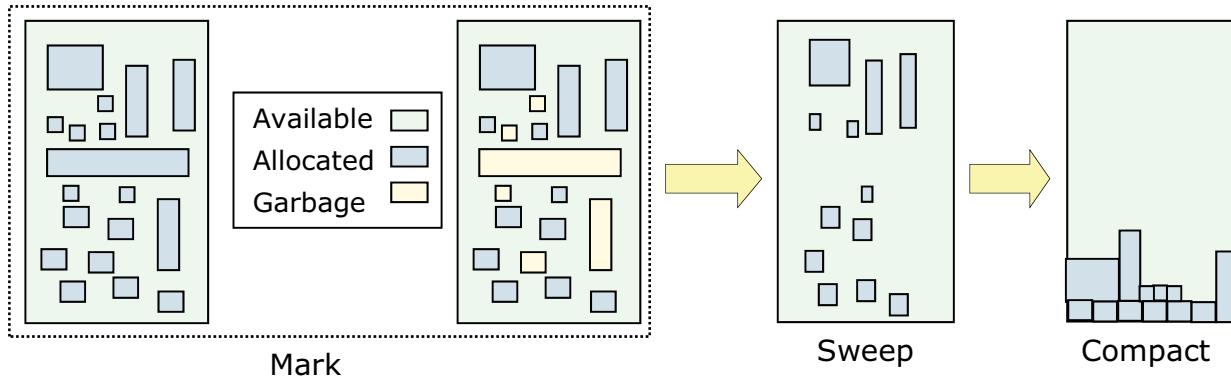
A parallel version of garbage collector mark exists. The time spent marking objects is decreased through the addition of helper threads and a facility that shares work between those threads.

Incremental compaction is a way of spreading compaction work across a number of garbage collection cycles, reducing pause times. Another important task for Incremental Compaction is the removal of dark matter. Dark matter is the term for small pieces of free space (currently less than

512 bytes in size) that are not on the free list and therefore are not available for allocation of objects.

Garbage collection phases: Mark, sweep, compact

- **Mark** phase identifies objects that are not used
- **Sweep** phase frees objects not marked to meet garbage collection target (satisfy the allocation request)
- **Compact** phase collects and compacts the resulting freed-up spaces



© Copyright IBM Corporation 2016

Figure 6-17. Garbage collection phases: Mark, sweep, compact

WA8152.2

Notes:

When the JVM cannot allocate an object from the current heap due to the lack of contiguous space, a memory allocation fault occurs, and the GC is invoked. The first task of the GC is to collect all of the garbage that is in the heap. This process starts when any thread calls the GC either indirectly as a result of allocation failure, or directly by a specific call to `System.gc()`. The first step is to acquire exclusive control on the virtual machine to prevent any further Java operations. Garbage collection can then begin.



Garbage collection policies

Memory management is configurable by using four different policies with varying characteristics

- **Generational concurrent:** (new default) Divides heap into “nursery” and “tenured” segments that provide fast collection for short lived objects
 - Can provide maximum throughput with minimal pause times
- **Optimize for throughput:** Flat heap collector that is focused on maximum throughput
- **Optimize for pause time:** Flat heap collector with concurrent mark and sweep to minimize GC pause time
- **Balanced:** New policy
 - Uses a region-based layout for the Java heap
 - These regions are individually managed to reduce the maximum pause time on large heaps and increase the efficiency of garbage collection
- **Subpool:** This option is now deprecated and is treated as an alias for optimize for throughput

© Copyright IBM Corporation 2016

Figure 6-18. Garbage collection policies

WA8152.2

Notes:

The former default policy was “optimize for throughput”. It is a “stop-the-world” policy where the JVM does not do any other work. To optimize the throughput, the garbage collection does all of its work during the GC cycle and does not affect the application when not active.

The `-Xgcpolicy` command line parameter is used to enable and disable concurrent mark:

`-Xgcpolicy:<optthrput | optavgpause | gencon | balanced | subpool>`

The `-Xgcpolicy` options have these effects:

- **optthrput** disables concurrent mark. If you do not have pause time problems (as seen by erratic application response times), you get the best throughput with this option. *Optthrput* is the default setting.
- **optavgpause** enables concurrent mark with its default values. If you are having problems with erratic application response times that normal garbage collections cause, you can reduce those problems at the cost of some throughput, by using the *optavgpause* option.
- **gencon** requests the combined use of concurrent and generational GC to help minimize the time that is spent in any garbage collection pause.

- ***balanced*** is a new policy that uses a region-based layout for the Java heap. These regions are individually managed to reduce the maximum pause time on large heaps and increase the efficiency of garbage collection. The policy also uses a different object allocation strategy that improves application throughput on large systems that have Non-Uniform Memory Architecture (NUMA) characteristics
- ***subpool*** is used before version 8. A flat heap technique to help increase performance on large SMP systems with 16 or more processors by optimizing the object allocation. Only available on IBM pSeries and zSeries.

This option is now deprecated. The subpool option is treated as an alias for optimize for throughput. Therefore, if you use this option, the effect is the same as optimize for throughput.

GC helper threads

- Parallelism is achieved by using GC Helper Threads
 - “Parked” set of threads that wake to share GC work
 - Main GC thread generates the root set of objects
 - Helper threads share the work for the rest of the phases
 - Number of helpers is one less than the number of processing units CPUs
 - The number of helper threads plus main GC thread equals the number of processing units (CPUs)
 - Configurable using `-Xgcthreads`

© Copyright IBM Corporation 2016

Figure 6-19. GC helper threads

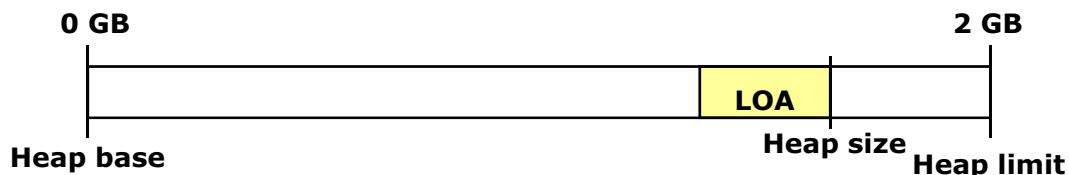
WA8152.2

Notes:

By default, a platform with n processors also has $n - 1$ new helper threads, which work with the master thread to complete the marking phase of garbage collection. You can override the default number of threads by using the `-Xgcthreads` option. If you specify a value of 1, there are no helper threads. The `-Xgcthreads` option accepts any value greater than 0, but clearly you gain nothing by setting it to more than $n - 1$.

Parallel GC (optthrput)

- Parallel mark sweep collector, with compaction avoidance
 - Developed to use extra processors on server systems
 - Designed to increase performance for SMP and not degrade performance for uniprocessor systems
- Optimized for throughput
 - Best policy for “batch” type applications
- Consists of a single flat Java heap



© Copyright IBM Corporation 2016

Figure 6-20. No title

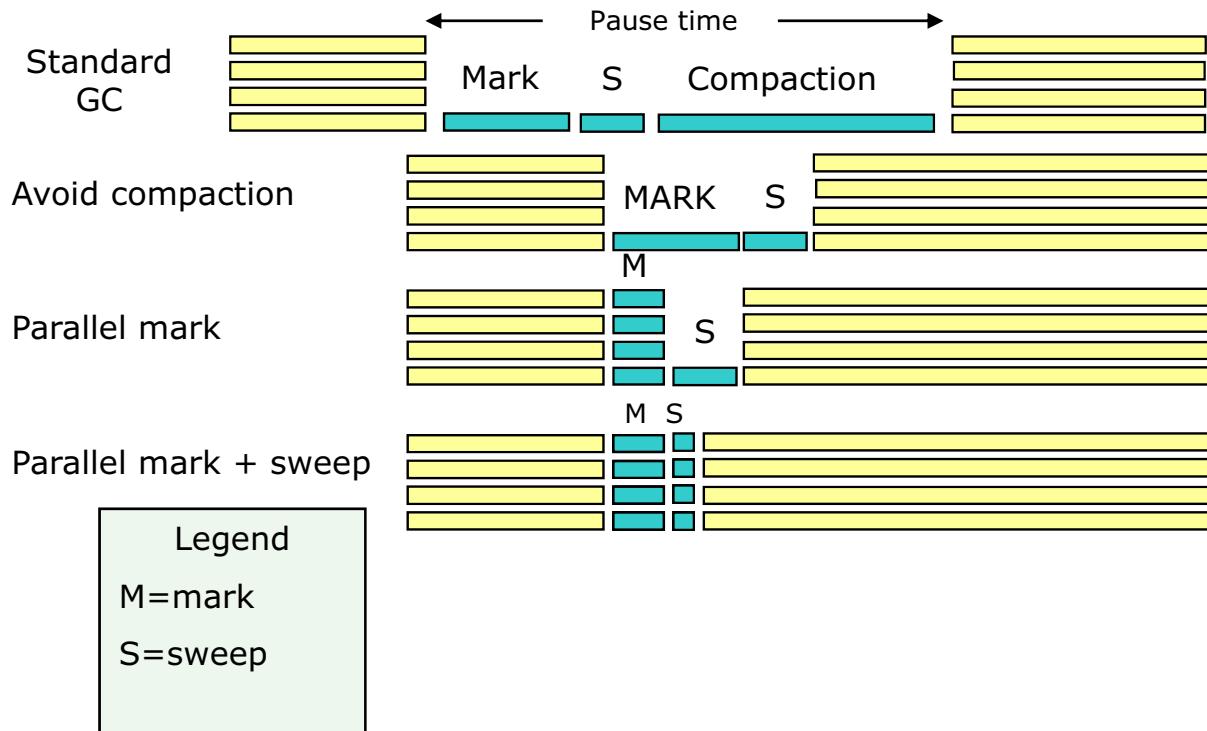
WA8152.2

Notes:

The optthrput policy provides high throughput but with longer garbage collection pause times. During a garbage collection, all application threads are stopped for mark, sweep, and compaction, when compaction is needed. The optthrput policy is sufficient for most applications.

The large object area (LOA) is an area of the tenured section of the heap set used to satisfy allocations for large objects that cannot be satisfied in the small object area (SOA), of the tenured heap.

Parallel mark and parallel sweep view of GC



© Copyright IBM Corporation 2016

Figure 6-21. No title

WA8152.2

Notes:

This graphic is meant to show how pause time can be reduced during GC by avoiding compaction. Also, the graphic shows how pause time can be reduced with the use of helper threads during both the mark phase and the sweep phase.

Pause time

When an application's attempt to create an object cannot be satisfied immediately from the available space in the heap, the garbage collector is responsible for identifying unreferenced objects (garbage), deleting them. The heap state is then available for immediate and subsequent allocation requests.

Such garbage collection cycles introduce occasional unexpected pauses in the execution of application code. Because applications grow in size and complexity, and heaps become correspondingly larger, this garbage collection pause time tends to grow in size and significance.

Concurrent GC (optavgpause)

- Reduces the time spent inside stop-the-world (STW) GC
 - Reduction usually 90 - 95%
- Achieved by carrying out some of the stop-the-world work while the application is running
 - 1.4.2 Concurrent marking
 - 5.0 and 6.0 Concurrent marking and concurrent sweeping
- Slight reduction of throughput for greatly reduced STW times
- Policy is ideal for systems with responsiveness criteria
 - Example: portal applications

© Copyright IBM Corporation 2016

Figure 6-22. Concurrent GC (optavgpause)

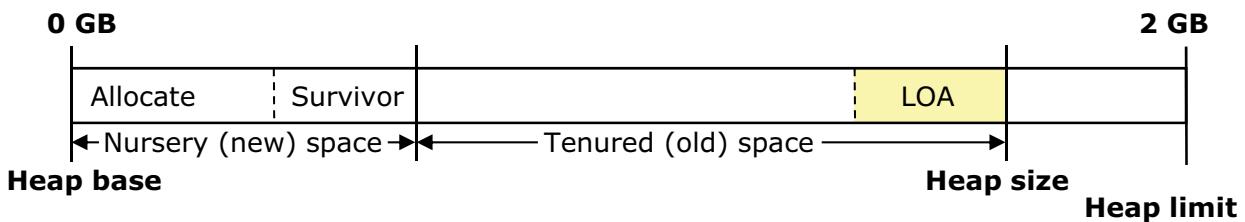
WA8152.2

Notes:

Optavgpause is the policy that reduces garbage collection pause time by performing the mark and sweep phases of garbage collection while an application is running. This policy causes a small performance impact to overall throughput.

Generational concurrent GC (gencon)

- Similar in concept to the mode that Solaris and HP-UX use
 - Parallel copy and concurrent global collects by default
- Motivation: Objects die young so focus collection efforts on recently created objects
 - Divide the heap up into a two areas: “new” and “old”
 - Perform allocations from the new area
 - Collections focus on the new area
 - Objects that survive a number of collects in new area are promoted to old area (tenured)



- Ideal for transactional and high data throughput workloads

© Copyright IBM Corporation 2016

Figure 6-23. Generational concurrent GC (gencon)

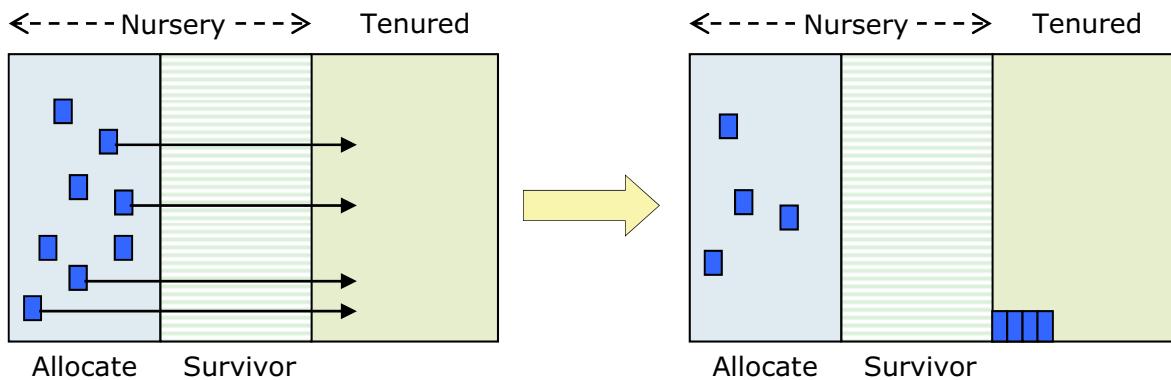
WA8152.2

Notes:

The Generational Concurrent Garbage Collector was introduced in Java 5.0 from IBM. A generational garbage collection strategy is suited to an application that creates many short-lived objects, as is typical of many transactional applications.

Generational concurrent GC: Scavenge (1 of 2)

- When the allocate space is full, a GC is triggered
- The allocate space is traced
- Objects that reach the tenured age (survived a specific number of scavenge operations; maximum is 14) are copied into the tenured area



© Copyright IBM Corporation 2016

Figure 6-24. Generational concurrent GC: Scavenge (1 of 2)

WA8152.2

Notes:

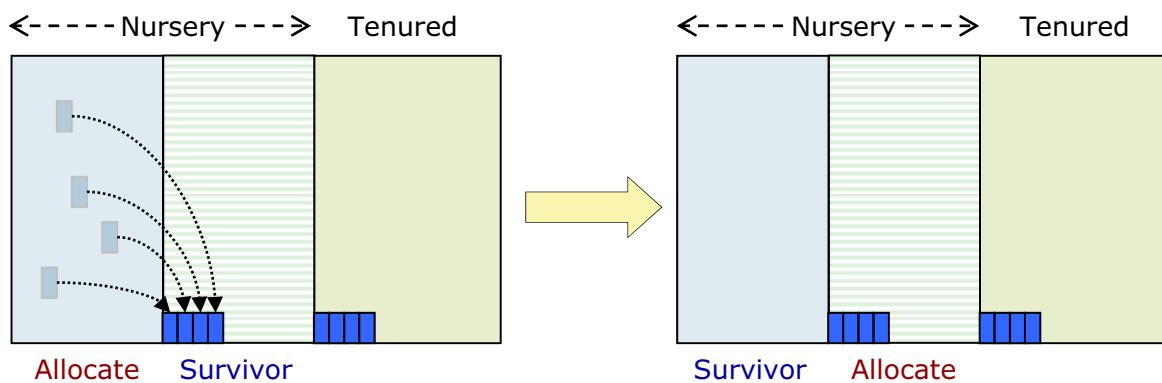
Different algorithms are used in each generation. For young generations, compaction or by-copy algorithms are used. For older tenured generations, the normal mark and sweep algorithms are used.

The Java heap is split into two areas, a new (or nursery) area and an old (or tenured) area. Objects are created in the new area and, if they live long enough, they are moved or promoted into the old area. Objects are promoted when they reach a certain age (known as the tenure age). This age is a count of the number of garbage collections for which they are alive.

Tenure age is a measure of the object age at which it should be promoted to the tenure area. The JVM dynamically adjusts this age, and reaches a maximum value of 14. The age of an object is incremented on each scavenge. A tenure age of x implies that, if the object survived x flips between survivor and allocate space, it is promoted. The threshold is adaptive and adjusts the tenure age that is based on the percentage of space that is used in the new area.

Generational concurrent GC: Scavenge (2 of 2)

- Remaining live objects are copied from the allocate space into the survivor space, and a count of the number of times each object was scavenged is incremented
- The copied objects are placed contiguously in the survivor space so that an effective compaction occurs
- At the end of scavenge, the survivor space and allocate space pointers are flipped
 - The survivor space becomes the new allocate space and is empty



© Copyright IBM Corporation 2016

Figure 6-25. Generational concurrent GC: Scavenge (2 of 2)

WA8152.2

Notes:

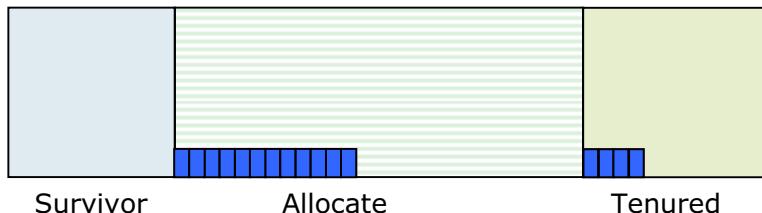
The new area is split into two logical spaces: allocate and survivor. Objects are allocated into the allocate space. When that space is filled, a GC process that is called scavenge is triggered. During a scavenge, live objects are copied either into the survivor space or into the tenured space if they are old enough to reach the tenured age. Then, the roles are reversed: the survivor space becomes the allocate space.

Dead objects in the nursery remain untouched. When all the live objects are copied, the spaces in the new area switch roles. The new survivor space is now entirely empty of live objects and is available for the next scavenge. Any objects that were not moved into the old or tenured space are now in the “new” allocate space.

A technique that is called *tilting* maximizes the size of the allocate space in the new area. Tilting controls the relative sizes of the allocate and survivor spaces. Based on the amount of data that survives, the ratio can be adjusted to make the survivor space smaller.

Generational concurrent GC: Self-adjusting

- The scavenger automatically adjusts the relative sizes of the allocate and survivor spaces by using history and predictions (Tilt)



- The generational collector is designed to respond dynamically to varying conditions so that its behavior is adaptive
- These mechanisms are:
 - Concurrent collection runs in the tenured area
 - Rarely is there a full collection (mark, sweep, compact) in the tenured area
 - The sizes of the young and tenured generations will self-adjust over time, which is based on memory pressure
 - The scavenger adjusts the flip count (number of scavenges that an object must survive before tenure) based on history
 - The tilt ratio self-adjusts based on history

© Copyright IBM Corporation 2016

Figure 6-26. Generational concurrent GC: Self-adjusting

WA8152.2

Notes:

A technique that is called tilting maximizes the size of the allocate space in the new area. Tilting controls the relative sizes of the allocate and survivor spaces. Based on the amount of data that survives, the ratio can be adjusted to make the survivor space smaller.

Balanced GC

- The Balanced Garbage Collection policy uses a region-based layout for the Java heap
 - Regions are individually managed to reduce the maximum pause time on large heaps
 - Benefits from Non-Uniform Memory Architecture (NUMA) characteristics of modern server hardware
- The Balanced Garbage Collection policy is intended for environments where heap sizes are greater than 4 GB
 - The policy is available only on 64-bit platforms
 - You activate this policy by specifying `-Xgcpolicy:balanced` on the command line
- When to use the Balanced garbage collection policy
 - If using the gencon policy, and the performance is good but compactions occur
 - The application experiences large global collection pause times
 - The global collections are frequent enough to be disruptive

© Copyright IBM Corporation 2016

Figure 6-27. Balanced GC

WA8152.2

Notes:

The Java heap is split into potentially thousands of equal sized areas that are called “regions”. Each region can be collected independently, which allows the collector to focus only on the regions, which offer the best return on investment.

Objects are allocated into a set of empty regions that the collector selects. This area is known as an Eden space. When the Eden space is full, the collector stops the application to perform a Partial Garbage Collection (PGC). The collection might also include regions other than the Eden space, if the collector determines that these regions are worth collecting. When the collection is complete, the application threads can proceed, allocating from a new Eden space, until this area is full. This process continues for the life of the application.

From time to time, the collector starts a Global Mark Phase (GMP) to look for more opportunities to reclaim memory. Because PGC operations see only subsets of the heap during each collection, abandoned objects might remain in the heap. This issue is like the “floating garbage” problem that is seen by concurrent collectors. However, the GMP runs on the entire Java heap and can identify object cycles that are inactive for a long period. These objects are reclaimed.

When to use the Balanced garbage collection policy?

There are a number of situations when you should consider the Balanced garbage collection policy. Generally, if you are currently using the gencon policy, and the performance is good but the application still experiences large global collection (including compaction) pause times frequently enough to be disruptive, consider the Balanced policy.

6.3. JVM heap memory usage

JVM heap memory usage



© Copyright IBM Corporation 2016

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

9.1

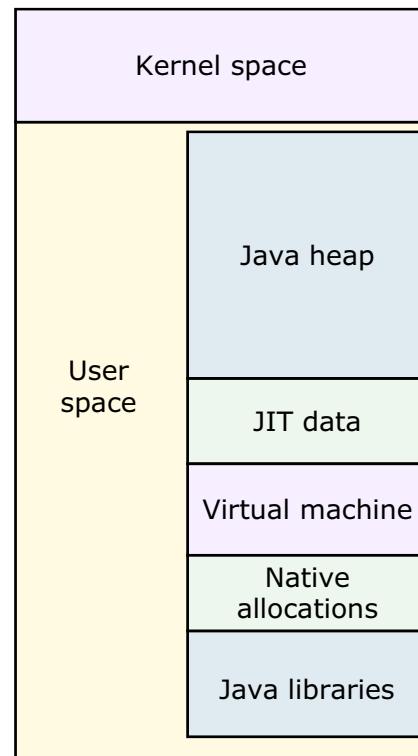
Figure 6-28. JVM heap memory usage

WA8152.2

Notes:

How does Java use native memory?

- Java heap:
 - Occupies the `-Xmx` value from startup
- Just In Time (JIT) compiler
 - Runtime data, executable code
- Virtual machine resources:
 - Dump and trace engines
 - GC infrastructure
- JNI allocations:
 - Native (`malloc`, `new`) in JNI functions
 - Used by the interface itself
- Resources to underpin Java objects;
 - Classes and class loaders
 - Threads
 - Direct `java.nio.ByteBuffers`
 - Sockets



© Copyright IBM Corporation 2016

Figure 6-29. How does Java use native memory?

WA8152.2

Notes:

Virtual machine resources: Most are static in size such as the execution engine and the debug engine. Some dependent on the application, the GC infrastructure, and the Java heap size.

JIT compiler resources include the JIT code cache and JIT data cache.

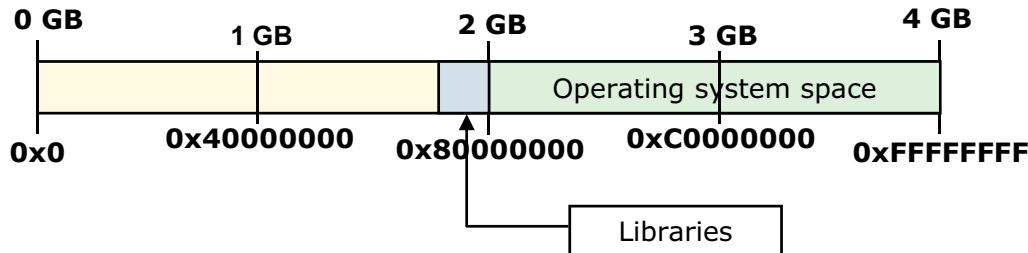
JNI allocations: Direct allocation calls allocate onto the native heap, and are not tracked by the JVM. For example, calls to `malloc()`, `calloc()`, and `realloc()`. While these calls are not prohibited, they require careful management to ensure that memory is not leaked.

Resources to underpin Java objects: Some object types have associated native resources:

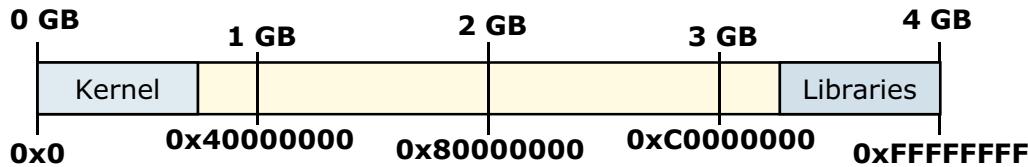
- Thread objects have native threads.
- AWT objects have widgets and shells.
- Socket objects have native sockets.
- Class objects have class structure resources such as: Constant pool, method table, method blocks, and compiled code.
- The Classloader has a class cache and constraint table.

Memory available to the Java process

- On Windows 32-bit:



- On AIX 32-bit:



© Copyright IBM Corporation 2016

Figure 6-30. Memory available to the Java process

WA8152.2

Notes:

Hardware imposes many of the restrictions that a native process experiences, not the OS. Every computer has a processor and some random-access memory (RAM), also known as physical memory. A processor interprets a stream of data as instructions to execute; it has one or more processing units that perform integer and floating-point arithmetic and more advanced computations. A processor has a number of *registers*, fast memory elements that are used as working storage for the calculations that are performed; the register size determines the largest number that a single calculation can use.

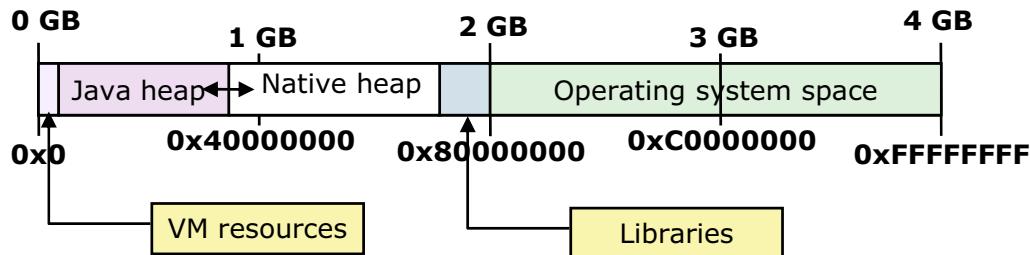
The processor is connected to physical memory by the memory bus. The size of the physical address (the address that is used by the processor to index physical RAM) limits the amount of memory that can be addressed. For example, a 16-bit physical address can address from 0x0000 to 0xFFFF, which gives $2^{16} = 65536$ unique memory locations. If each address references a byte of storage, a 16-bit physical address would allow a processor to address 64 KB of memory.

Processors are described with a bit size. This number normally refers to the size of the registers, although there are exceptions such as 390 31 bit, where it refers to the physical address size. For desktop and server platforms, this number is 31, 32, or 64; for embedded devices and microprocessors; it can be as low as 4. The physical address size can be the same as the register

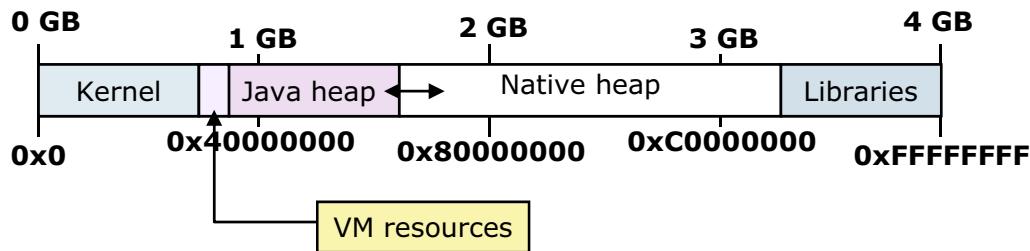
width but might be larger or smaller. Most 64-bit processors can run 32-bit programs when running a suitable OS.

Native heap available to application

- On Windows:



- On AIX:



© Copyright IBM Corporation 2016

Figure 6-31. Native heap available to application

WA8152.2

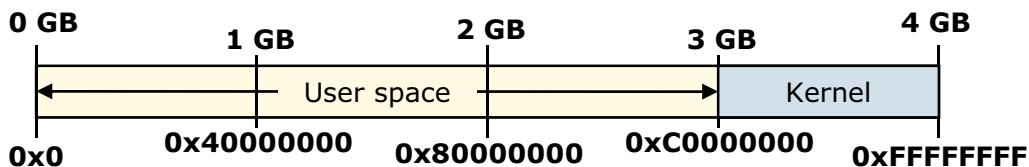
Notes:

The heap's size is controlled from the Java command line by using the `-Xmx` and `-Xms` options (`m` is the maximum size of the heap, `m` is the initial size). Although the logical heap (the area of memory that is actively used) can grow and shrink according to the number of objects on the heap and the amount of time that is spent in GC, the amount of native memory that is used remains constant and depends on the `-Xmx` value: the maximum heap size. Most GC algorithms rely on the allocation of heap from a contiguous slab of memory, so it is impossible to allocate more native memory when the heap needs to expand. All heap memory must be reserved up front.

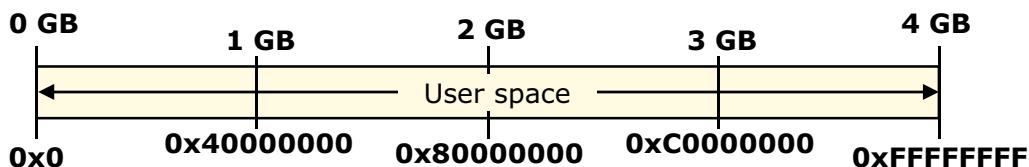
Reserving native memory is not the same as allocating it. When native memory is reserved, it is not backed with physical memory or other storage. Although reserving chunks of the address space does not exhaust physical resources, it does prevent that memory from being used for other purposes. A leak that is caused by reserving memory that is never used is as serious as leaking allocated memory.

32-bit user space available to the Java process

- Linux:



- Linux with hugemem kernel:



- Separate address space for the kernel

© Copyright IBM Corporation 2016

Figure 6-32. 32-bit user space available to the Java process

WA8152.2

Notes:

Reserving native memory is not the same as allocating it. When native memory is reserved, it is not backed with physical memory or other storage. Although reserving chunks of the address space does not exhaust physical resources, it does prevent that memory from being used for other purposes. A leak that is caused by reserving memory that is never used is as serious as leaking allocated memory.

Theoretical and advised max heap sizes for 32-bit

- The larger the Java heap, the more constrained the native heap
- Advised limits to prevent native heap from becoming overly restricted, leading to OutOfMemoryError exception

Platform (32-bit)	Additional options	Maximum possible	Advised maximum
AIX	automatic	3.25 GB	2.5 GB
Linux		2 GB	1.5 GB
	Hugemem kernel	3 GB	2.5 GB
Windows		1.8 GB	1.5 GB
	3 GB	1.8 GB	1.8 GB
z/OS		1.7 GB	1.3 GB

- Exceeding advised limits is possible, but should be done only when native heap usage is understood
 - Native heap usage can be measured by using OS tools

© Copyright IBM Corporation 2016

Figure 6-33. Theoretical and advised max heap sizes for 32-bit

WA8152.2

Notes:

The Linux hugemem kernel supports a 4-GB per process user space (versus 3 GB for the other kernels), and a 4-GB direct kernel space.

The 32-bit IBM JVM for Windows includes support for the /LARGEADDRESSAWARE switch, also known as the /3GB switch. This switch increases the amount of space available to a process, 2–3 GB. The switch is a Windows boot parameter, not a command line option to the JVM. After enabling the /3GB switch, the JVM gains 1 GB of extra memory space. This extra space does not increase the theoretical maximum size of the Java heap. However, it does allow the Java heap to grow closer to its theoretical maximum size (2 GB - 1 byte) because the extra memory can be used for the native heap.

Using 64-bit operating systems

- Moving to 64-bit effectively removes the Java heap size limit
 - However, ability to use more memory is not free
- 64-bit applications perform slower
 - More data has to be manipulated
 - Cache performance is reduced
- 64-bit applications require more memory
 - Java object references are larger
 - Internal pointers are larger
 - Therefore, slightly larger footprint for the same application
- Major improvements to 64-bit in Java 6.0 and 7.0 because of compressed references
 - Reduce the size of address references on 64-bit IBM J9

© Copyright IBM Corporation 2016

Figure 6-34. Using 64-bit operating systems

WA8152.2

Notes:

WebSphere Application Server V7.0 introduced compressed reference (CR) technology to dramatically improve performance on 64-bit operating systems. IBM CR technology for Java allows 64-bit WebSphere deployments to allocate large Java heaps without the memory footprint growth and performance cost that is generally incurred with larger, 64-bit address references. Using CR technology, WebSphere Application Server instances can allocate heap sizes up to 28 GB with the same physical memory performance cost as an equivalent 32-bit deployment.

Compressed references summary

- Compressed references reduce the size of address references on 64-bit IBM J9 for Java 6 and 7
 - Offsets increased memory footprint and the associated performance loss
- Application server instances can allocate heap sizes up to 28 GB
 - Same physical memory performance cost as an equivalent 32-bit deployment
- Applications on 64-bit are able to achieve about 95% of 32-bit performance when using the same heap size
 - Even with compressed references, the performance hit for 64-bit may be 5-10%
- These command-line options control compressed references feature:
 - `-Xcompressedrefs` – Enables the compressed references feature, and uses 32-bit wide memory references to address the heap (default)
 - `-Xnocompressedrefs` – Explicitly disables the compressed references feature, and uses full 64-bit wide memory references to address the heap

© Copyright IBM Corporation 2016

Figure 6-35. Compressed references summary

WA8152.2

Notes:

The CR technology provides this improvement by reducing the width of 64-bit Java heap references to 32 bits through an efficient bit shifting algorithm. The 4 byte (32-bit) compressed references are converted to 64-bit values and stored in processor registers in a “just-in-time manner” at run time. 64-bit deployments are thus able to reap the benefits of large Java heaps and 64-bit processor extensions while maintaining a comparable 32-bit memory footprint. Heap sizes below 4 GB might not require any shift operations, since the effective virtual address range requires only 32 bits.

64-bit WebSphere Application Server can provide dramatic performance improvements for applications that take advantage of large heaps. Historically, the memory footprint cost of 8-byte Java references limits the performance gains. CR technology removes this limitation. For applications that do not require a large heap, the performance of WebSphere Application Server 64-bit with CR technology is generally within 95% of WebSphere Application Server 32-bit, when using the same heap size settings and equivalent hardware configurations. The minimal performance cost is incurred from reference compression and decompression.

For more information, see the document, IBM WebSphere Application Server V7 64-bit performance: Introducing WebSphere Compressed Reference Technology.



IBM Java 7.0 versus IBM Java 6.0

- WebSphere Application Server V8.5.5 supports two Java versions:
 - IBM Java 6 (default)
 - IBM Java 7 SR4
- Performance tests with DayTrader3 EJB mode show that Java 7 provides a significant (~10%) throughput improvement over Java 6
- Application server startup slightly longer (~2.8%) with Java 7
- Memory footprint is slightly larger (~1.2%) with Java 7

© Copyright IBM Corporation 2016

Figure 6-36. IBM Java 7.0 versus IBM Java 6.0

WA8152.2

Notes:

The system configuration for the performance tests that yielded these results was:

- Hardware Intel Westmere with 4 cores enabled, 12GB RAM
- RedHat Linux 6, 64-bit

Unit summary

Having completed this unit, you should be able to:

- Describe components of the JVM and overall architecture
- Describe JIT and AOT compilers
- Describe the garbage collection process
- Describe garbage collection policies such as generational concurrent and optimize for throughput
- Describe JVM heap memory usage

© Copyright IBM Corporation 2016

Figure 6-37. Unit summary

WA8152.2

Notes:

Checkpoint questions

1. True or False: The JIT compiler must be explicitly enabled for an application server.
2. True or False: Garbage collection occurs when there is an allocation failure or it has been programmatically requested via a `System.GC()` call.
3. True or False: In addition to a mark and sweep, a compaction occurs for every garbage collection cycle.
4. True or False: Generational concurrent (gencon) is the default GC policy.

© Copyright IBM Corporation 2016

Figure 6-38. Checkpoint questions

WA8152.2

Notes:

Write your answers here:

- 1.
- 2.
- 3.
- 4.



Checkpoint answers

1. False: The JIT compiler is enabled by default for an application server.
2. True
3. False: A compaction occurs only if the mark and sweep phases cannot recover enough heap space.
4. True

© Copyright IBM Corporation 2016

Figure 6-39. Checkpoint answers

WA8152.2

Notes:

Exercise 6



Exploring GC policies and monitoring
JVM performance

© Copyright IBM Corporation 2016

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

9.1

Figure 6-40. Exercise 6

WA8152.2

Notes:

Exercise objectives

After completing this exercise, you should be able to:

- Enable verbose GC for an application server
- Manually view verbose GC logs
- Configure the optthruput GC policy
- Configure the optavgpause GC policy
- Use Apache JMeter to load test an application server
- Use IBM Garbage Collection and Memory Visualizer (GCMV) to analyze and compare GC data

© Copyright IBM Corporation 2016

Figure 6-41. Exercise objectives

WA8152.2

Notes:

Unit 7. Tuning the JVM

What this unit is about

This unit describes how to tune the JVM and how to troubleshoot common JVM problems.

What you should be able to do

After completing this unit, you should be able to:

- Size the JVM heap
- Tune various GC policies
- Troubleshoot common JVM problems

How you will check your progress

- Checkpoint questions
- Lab exercises

References

WebSphere Application Server V8.5 Information Center, Tuning the JVM,
http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/index.jsp?topic=%2Fcom.ibm.websphere.nd.doc%2Fae%2Fcontainer_tune_jvm.html

Unit objectives

After completing this unit, you should be able to:

- Size the JVM heap
- Tune various GC policies
- Troubleshoot common JVM problems

© Copyright IBM Corporation 2016

Figure 7-1. Unit objectives

WA8152.2

Notes:



Topics

- Tune the IBM JVM
- Introduction to JVM problem determination
- The HotSpot garbage collector

© Copyright IBM Corporation 2016

Figure 7-2. Topics

WA8152.2

Notes:

7.1. Tuning the IBM JVM

Tune the IBM JVM



© Copyright IBM Corporation 2016

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

9.1

Figure 7-3. Tune the IBM JVM

WA8152.2

Notes:

What factors affect memory performance the most

- Memory management:
 - How efficiently does the system manage memory?
 - GC policy
- Total available memory:
 - Is there enough memory to satisfy every request for memory?
 - Heap size
- Allocation rate:
 - How often does the application issue requests for memory?
 - Application design
- Object size:
 - How large are these objects?
 - Application design
- Object lifetime:
 - How long do these objects stay reserved by the application?
 - Application design

© Copyright IBM Corporation 2016

Figure 7-4. What factors affect memory performance the most

WA8152.2

Notes:

Enterprise applications that are written in the Java language involve complex object relationships and use large numbers of objects. Although the Java language automatically manages memory that is associated with object lifecycles, understanding the application usage patterns for objects is important. In particular, verify that the following conditions exist:

- The application is not leaking objects.
- The Java heap parameters are set properly to handle a specific object usage pattern.
- The application is not over-utilizing objects.

Runtime performance tuning: General

- Tuning the JVM properly is a process that takes time and must be tailored to your application
 - You can typically get 80% of the maximum performance with 20% of the work by ensuring that you are making good choices on a few key settings
 - To truly get maximum performance from your application, you must know your application's memory allocation and runtime needs
- The JVM must be tuned in two iterative steps over a testing cycle
 - Step 1: Tuning the heap size
 - Step 2: Select and tune GC policy
- Tuning the JVM is an iterative process
- Measure the results of each tuning action that is taken
- Change one parameter at a time
- Tuning is application and environment dependent

© Copyright IBM Corporation 2016

Figure 7-5. Runtime performance tuning: General

WA8152.2

Notes:

Garbage collection can be used to evaluate application performance health. By monitoring garbage collection during the execution of a fixed workload, users gain insight whether the application is over-utilizing objects. Garbage collection can even be used to detect the presence of memory leaks.

Use the garbage collection and heap statistics in Tivoli Performance Viewer to evaluate application performance health. You must enable JVMTI to get detailed garbage collection statistics. By monitoring garbage collection, memory leaks and overly used objects can be detected.

JVM configuration (1 of 2)

- Administrative console JVM configuration tab

The screenshot shows the 'Java Virtual Machine' configuration page. At the top, there's a breadcrumb navigation: Application servers > TradeServer1 > Process definition > Java Virtual Machine. Below it, a message says 'Use this page to configure advanced Java(TM) virtual machine settings.' Two tabs are present: 'Configuration' (selected) and 'Runtime'. Under 'Runtime', several options are listed with checkboxes: 'Verbose class loading' (unchecked), 'Verbose garbage collection' (checked), and 'Verbose JNI' (unchecked). Below these are input fields for 'Initial heap size' (128 MB) and 'Maximum heap size' (512 MB). There's also a checkbox for 'Run HProf' (unchecked) and a 'HProf Arguments' input field.

© Copyright IBM Corporation 2016

Figure 7-6. JVM configuration (1 of 2)

WA8152.2

Notes:

You need to select the **Verbose garbage collection** check box on the Runtime tab. Selecting this check box enables verbose GC logging immediately without having to restart the application server.

The initial heap size specifies, in megabytes, the initial heap size available to the JVM code. If this field is left blank, the default value is used.

The maximum heap size specifies, in megabytes, the maximum heap size that is available to the JVM code. If this field is left blank, the default value is used.

The default maximum heap size is 256 MB. This default value applies for both 32-bit and 64-bit configurations.

Increasing the maximum heap size setting can improve startup. When you increase the maximum heap size, you reduce the number of garbage collection occurrences with a 10% gain in performance.

Increasing this setting usually improves throughput until the heap becomes too large to reside in physical memory. If the heap size exceeds the available physical memory, and paging occurs, there is a noticeable decrease in performance. Therefore, it is important that the value you specify for this property allows the heap to be contained within physical memory.



JVM configuration (2 of 2)

- Administrative console JVM configuration tab

Debug Mode

Debug arguments

```
-agentlib:jdwp=transport=dt_socket,server=y,suspend=n,address=7777
```

Generic JVM arguments

```
-Xhealthcenter -Xgcpolicy:optthruput
```

Executable JAR file name

Disable JIT

- The Generic JVM arguments window is where you can specify:
 - GC policy (-XgcPolicy)
 - Use several garbage collection threads (-XgcThreads)
 - Disable class garbage collection (-XnoClassGC)
 - Disable explicit garbage collection (-XdisableExplicitGC)
 - Many more

© Copyright IBM Corporation 2016

Figure 7-7. JVM configuration (2 of 2)

WA8152.2

Notes:

The Generic JVM arguments window is where you specify command line arguments to pass to the Java virtual machine code that starts the application server process. You can enter the optional command line arguments in the Generic JVM arguments field. If you enter more than one argument, enter a space between each argument.

A manual call to the garbage collector (for example, through the `System.gc()` call) suggests that a garbage collection cycle runs. In fact, the call is interpreted as a request for full garbage collection scan unless a garbage collection cycle is already running or explicit garbage collection is disabled by specifying: `-XdisableExplicitGC`

-XnoClassGC By default, the JVM unloads a class from memory whenever there are no live instances of that class left. The cost of loading and unloading the same class multiple times, can decrease performance.

You can use the `-XnoClassGC` argument to disable class garbage collection. However, the performance cost of class garbage collection is typically minimal. Turning off class garbage collection in a Java EE-based system, might effectively create a memory leak of class data, and cause the JVM to throw an Out-of-Memory Exception. If you use this option, whenever you

redeploy an application, you should always restart the application server to clear the classes and static data from the previous version of the application.

Runtime performance tuning: Heap size

- Key setting for the IBM JVM
- Has a large impact on performance for all Java applications
- Setting the heap size correctly should bring you near to 80% of your maximum performance

Follow these rules for setting the heap size:

- Ensure that you set your heap size (`-Xms` and `-Xmx`) to values that allow you to have a substantially large interval between GCs
 - What counts is the percentage of execution time in GC (<= 5% to 20%)
 - Typical low end limit on frequency of GCs is 10 sec
 - Typical high end limit on duration of GCs is 1-2 sec
 - Too large heap sizes might result in long GC durations while too small heaps result in short intervals between the GC cycles
- Avoid paging (at any cost)
 - Monitor with `vmstat`
 - You cannot expect good (or even acceptable) performance when the system is paging

© Copyright IBM Corporation 2016

Figure 7-8. Runtime performance tuning: Heap size

WA8152.2

Notes:

The JVM heap size parameters directly influence garbage collection behavior. Increasing the JVM heap size allows more objects to be created before an allocation failure occurs and triggers a garbage collection. A larger JVM heap naturally allows the application to run for a longer period between garbage collection (GC) cycles. Unfortunately, an associated downside to increased heap size is a corresponding increase in the amount of time that is needed to find and process objects that should be collected. Tuning the JVM heap size often involves a balance between the interval between garbage collections and the pause time that is needed to complete the garbage collection.

Choosing the right GC policy (1 of 2)

There are four GC policies and each optimizes for different scenarios

GC policy	Description
-Xgcpolicy:gencon	<ul style="list-style-type: none"> (Default) Optimized for highly transactional workloads Use when fragmentation can be an issue (applications create large objects that are short lived)
-Xgcpolicy:optthrput	<ul style="list-style-type: none"> Optimized for batch type applications
-Xgcpolicy:optavgpause	<ul style="list-style-type: none"> Optimized for applications with responsiveness criteria
-Xgcpolicy:balanced	<ul style="list-style-type: none"> Optimized to reduce the maximum pause time on large heaps and increase the efficiency of garbage collection

© Copyright IBM Corporation 2016

Figure 7-9. Choosing the right GC policy (1 of 2)

WA8152.2

Notes:

Setting `gcpolicy` to `optthrput` disables concurrent mark. If you do not have pause time problems, denoted by erratic application response times, you should get the best throughput by using this option. Setting `gcpolicy` to `optavgpause` enables concurrent mark with its default values. This setting alleviates erratic application response times that result during normal garbage collection. However, this option might decrease the overall throughput.

Choosing the right GC policy (2 of 2)

- How do you know which GC policy to use?
- Monitor GC activity
 - Use the Health Center or Garbage Collection and Memory Visualizer tools
- Use of verbose GC logging
 - Provides data that is required for GC performance tuning
- Look for certain characteristics
 - Frequency of allocation failures
 - Time to process allocation failures

© Copyright IBM Corporation 2016

Figure 7-10. Choosing the right GC policy (2 of 2)

WA8152.2

Notes:

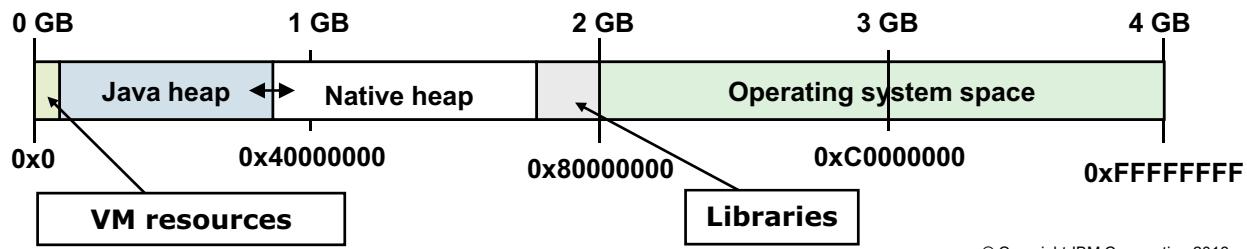
While each policy provides unique benefits, for WebSphere Application Server Version 8.0 and later, `gencon` is the default garbage collection policy. Previous versions of the application server specify that `optthruput` is the default garbage collection policy.

This `gencon` policy works with the generational garbage collector. The generational scheme attempts to achieve high throughput along with reduced garbage collection pause times. To accomplish this goal, the heap is split into new and old segments. Long lived objects are promoted to the old space while short-lived objects are garbage collected quickly in the new space. The `gencon` policy provides significant benefits for many applications. However, it is not suited for all applications, and is typically more difficult to tune.

Memory basics

Two heaps:

- Native heap (system heap)
 - Has objects whose lifetime is the same as the JVM (such as class file)
 - Contains objects that are allocated via JNI; for example, database (type 2) connections
 - Thread stacks
 - Not controlled by the garbage collector
- Java heap
 - Contains all Java objects
 - `-Xmx` and `-Xms` options bound the Java heap



© Copyright IBM Corporation 2016

Figure 7-11. Memory basics

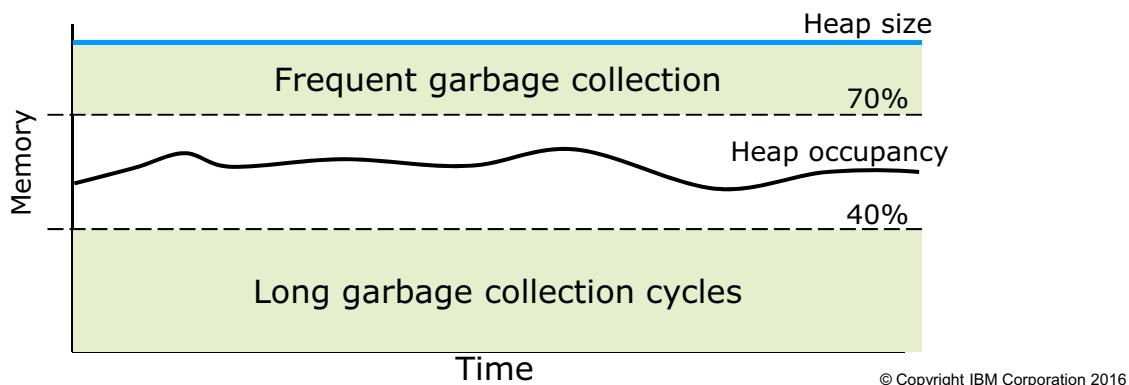
WA8152.2

Notes:

The native heap available to applications is different from one operation system to another operation system.

Garbage collector (GC) managed Java heap size

- GC adapts heap size to keep occupancy between 40% and 70%
 - Heap occupancy over 70% causes frequent GC cycles, which generally means reduced performance
 - Heap occupancy below 40% means infrequent GC cycles, but cycles longer than they need to be; the result is longer pause times than necessary, and reduced performance
- The maximum heap size setting should therefore be 43% larger than the maximum occupancy of the application
 - $(\text{Maximum occupancy}) + (43\%) = \text{occupancy at 70\% of total heap}$
 - **Example:** For 70 MB occupancy, 100 MB Maximum heap that is required, which is $70 \text{ MB} + 43\% \text{ of } 70 \text{ MB}$



© Copyright IBM Corporation 2016

Figure 7-12. Garbage collector (GC) managed Java heap size

WA8152.2

Notes:

Math lesson:

MH = maximum heap size

MO = maximum heap occupancy

You want MH such that $70\% \text{ of } \text{MH} = \text{MO}$, or $0.7 \text{ MH} = \text{MO}$

Doing the math,

$\text{MH} = 1.43 \text{ MO}$

$\text{MH} = \text{MO} + 43\% \text{ of } \text{MH}$

Fixed heap sizes versus variable heap sizes (1 of 2)

- Fixed heap size:
 - Initial heap size (-Xms) = Maximum heap size (-Xmx)
 - Fixed heap sizes do not expand or shrink the Java heap
 - Generally recommended
 - Avoids compact GC phase, which must run each time that the heap shrinks
 - Trade off: Startup of the JVM is slower since a larger heap must be allocated

© Copyright IBM Corporation 2016

Figure 7-13. Fixed heap sizes versus variable heap sizes (1 of 2)

WA8152.2

Notes:

It is generally suggested to set the initial and maximum heap sizes to the same value to prevent the JVM from dynamically resizing the heap. This setting not only makes GC analysis easier by fixing a key variable to a constant value, but also avoids the performance cost that is associated with allocating and deallocating more memory. The trade-off for setting the minimum and maximum heap sizes to the same value is that the initial startup of the JVM is slower since the JVM must allocate the larger heap. There are scenarios where setting the minimum and maximum heap settings to different values is advantageous.

Refer to: WebSphere Application Server Performance Tuning Case Study,
http://www.ibm.com/developerworks/websphere/techjournal/0909_blythe/0909_blythe.html

Fixed heap sizes versus variable heap sizes (2 of 2)

Variable heap sizes:

- GC adapts the heap size to keep occupancy between 40%–70%
 - Expands and shrinks the Java heap as necessary
- Allows for scenario where usage varies over time
 - Where variations would take usage outside the 40%–70% window
- Advised for large object situations
 - Creates head room such that heap can be grown with contiguous heap space if a large object needs to be allocated
 - If heap size equals maximum (-Xmx) heap size, you are subject to fragmentation issues and are at risk of encountering an out of memory for a large object

Each option has advantages and disadvantages, and you must select which is right for your particular application

© Copyright IBM Corporation 2016

Figure 7-14. Fixed heap sizes versus variable heap sizes (2 of 2)

WA8152.2

Notes:

A common performance-tuning measure is to make the initial heap size (`-Xms`) equal to the maximum heap size (`-Xmx`). Because no heap expansion or contraction occurs, this setting can result in significant performance gains in some situations. Usually, only the applications that need to handle a surge of allocation requests keep a substantial difference between initial and maximum heap sizes. Remember, though, that if `-Xms100m -Xmx100m` is specified, the JVM uses 100 megabytes of memory for its complete lifetime, even if the heap usage never exceeds 10%.

Heap expansion and shrinkage

- The act of heap expansion and shrinkage is relatively cheap
- A compaction of the Java heap is sometimes required
 - Expansion: GC might compact to try to allocate the object before expansion for some expansion
 - Shrinkage: GC might need to compact to move objects from the area of the heap that is shrinking
- Expansion and shrinkage optimize the heap occupancy
 - It usually does so at the cost of a compaction cycle

© Copyright IBM Corporation 2016

Figure 7-15. Heap expansion and shrinkage

WA8152.2

Notes:

When and why does the Java heap expand?

The JVM starts with a small default Java heap, and it expands the heap as a result of an application's allocation requests until it reaches the value that `-Xmx` specifies. Expansion occurs after GC if GC is unable to free enough heap storage for an allocation request, or if the JVM determines that expanding the heap is required for better performance.

When does the Java heap shrink?

Heap shrinkage occurs when GC determines that there is large amount of free heap storage, and releasing some heap memory is beneficial for system performance. Heap shrinkage occurs after GC, but when all of the threads are still suspended.

How to tune a generational concurrent GC setup

- Tenured space must be large enough to hold all persistent data of the application
 - Too small causes excessive GC or even out-of-memory conditions
 - For a typical WebSphere Application Server application: 100–400 MB
- One way to determine the tenure space size:
 - The amount of free heap that exists after each GC in default mode
 - (% free heap) x (total heap size)
- Analyze GC logs
 - Understand how frequently the tenured space gets collected
 - An optimal generational application has infrequent collection in the tenured space

© Copyright IBM Corporation 2016

Figure 7-16. How to tune a generational concurrent GC setup

WA8152.2

Notes:

Size the tenure space:

- Sizing the tenure space is analogous to sizing overall heap size in a mark, sweep, compact collector.
 - For a fixed size use `-Xmo` with `xx` megabytes
 - For a dynamic sized tenure space within set boundaries, use `-Xmos` and `-Xmox` to set the minimum and maximum tenure sizes
- The fixed size options are not compatible with the minimum and maximum settings. If you specify both, the JVM does not start and it throws an error message.
- There is no specific control on the flip count. The scavenger adjusts the flip count as needed.
- There is no specific control on the tilt ratio. The scavenger adjusts the tilt ration as needed.

How to tune a generational concurrent GC nursery generation space

- Large nursery:
 - Large nursery is good for throughput
 - If it is too large, it delays the scavenge operations
- Small nursery:
 - Small nursery is good for low pause times
 - If it is too small, it causes frequent scavenge operations over a short span of time
- Good WebSphere performance (throughput) requires a large nursery
 - A good starting point is 512 MB
 - Move up or down to determine optimal value
 - Measure throughput and response times
- Analyze GC logs to understand frequency and length of scavenges

© Copyright IBM Corporation 2016

Figure 7-17. How to tune a generational concurrent GC nursery generation space

WA8152.2

Notes:

- If the nursery is too small, there are many scavenge operations over short spans of time. The result is high processor usage. Because there is a maximum flip count of 14, many objects reach the tenure age quickly. Many of these objects are short-lived and are tenured incorrectly. This occurrence fills and fragments the old generation, and forces a full garbage collect in tenure space (mark, sweep, and the nearly always deadly compaction).
- If the nursery is too large, it delays scavenge operations. Most objects are no longer live and can be reclaimed. Many long-lived objects need to be copied to the survivor space. Since scavenge runs infrequently, it takes too long for these objects to reach the flip count and get tenured.

Sizing the nursery

- Copying from allocate to survivor or to tenured space is expensive
 - Physical data is copied (similar to compaction, which is also expensive)
- Ideally survival rates should be as low as possible
 - Less data needs to be copied
 - Less tenured and global collections occur
- The larger the nursery:
 - The greater the time between collections
 - The fewer objects that should survive
 - However, the longer copying can potentially take
- Recommendation is to have a nursery as large as possible
 - But not so large that nursery collection times affect the application responsiveness
 - Tune or increase the maximum nursery size around the default
 - The default maximum nursery size on IBM JVM V6 is 64 MB, which is too small for heaps larger than 256 MB

© Copyright IBM Corporation 2016

Figure 7-18. Sizing the nursery

WA8152.2

Notes:

There are several controls on the size of the nursery:

- If you want a fixed size nursery, use `-Xmn` with the number of megabytes required.
- For a dynamic sized nursery, you use `-Xmns` and `-Xmnx` to set the minimum and maximum nursery size. The collector adjusts the size between these settings as required.

7.2. Introduction to JVM problem determination

Introduction to JVM problem determination



© Copyright IBM Corporation 2016

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

9.1

Figure 7-19. Introduction to JVM problem determination

WA8152.2

Notes:

JVM problem determination: Symptom analysis

Application server stops responding under the following conditions:

- Server crash
 - Application server stops or exits unexpectedly
 - Tools are available to determine the cause of crash
- Hung process
 - Verify that application server process is still running
 - Threads might be deadlocked
 - Code might be running in a loop
 - Tools available to determine cause of the hang
- Out of memory condition
 - Errors and exceptions that are logged without process exit
 - At times, can result in unexpected process exit
- Performance degradation
 - Application server might crash and the node agent restarts it
 - Check to see whether the process ID is continually changing

© Copyright IBM Corporation 2016

Figure 7-20. JVM problem determination: Symptom analysis

WA8152.2

Notes:

An application server that does not respond might be hung, or the process might be terminated. Users see hung applications or are not able to access new applications.

An application server can suffer performance degradation when an application server is repeatedly failing and being restarted automatically.

If CPU activity is low but the application server is not terminated, you most likely have a hang or deadlock situation. If CPU activity is high and the application server is using the cycles, you most likely have a loop or inefficient code.

An often encountered condition is out-of-memory (OOM), which manifests itself either as an unexpected process exit with an “`OutOfMemoryException`” or just a list of errors and exceptions but no immediate process exit. This situation can occur when the application server is running low on memory, due to an application problem such as a memory leak, a hardware memory failure, or higher than usual demand.

JVM problem determination: Data to collect

- Core files
 - Also known as process memory dumps or system core files
 - Complete memory dump of the virtual memory for the process
 - Can be large
 - Usually required by IBM support
 - Tools available to parse these files
- Javacore files
 - Also known as Java dump or thread dump files
 - Text file that is created during an application server failure
 - Can also be generated manually
 - Error condition is given at top of memory dump
 - Format of the file is specific to the IBM JDK
- VerboseGC logs
 - Provides detailed information about garbage collection cycles
- Heap memory dump
 - Shows the objects that use Java heap memory
 - Needed for memory leak determination

© Copyright IBM Corporation 2016

Figure 7-21. JVM problem determination: Data to collect

WA8152.2

Notes:

These data artifacts are presented here as an overview on the types of data available.

Javacore creation is enabled by default and can be disabled with option “DISABLE_JAVADUMP”. The steps to initiate and control the output of the javacore file vary by JVM type. It always best to look at the documentation that is provided with the specific JVM. On Solaris and HP-UX platforms, the thread dump goes to native_stdout.

Javacore overview

- What is a javacore?
 - A snapshot of the running Java process
 - Small diagnostic text file that the JVM produces contains vital information about the running JVM process
 - Lists the JVM command line, environments, and loaded libraries
 - Provides a snapshot of all the running threads, their stack traces, and the monitors (locks) held by the threads
 - GC history and storage management (memory) information
 - Necessary for detecting hung threads and deadlock conditions
 - Useful for detecting some categories of native memory leaks

- Also, helpful for detecting performance problems
 - Take at least three snapshots of the JVM (about 2 – 3 minutes apart)
 - Analyze the javacores to see what different threads are doing in each snapshot
 - **Example:** A series of snapshots where container threads are in the same method or waiting on the same monitor or resource might indicate a bottleneck, hang, or a deadlock

© Copyright IBM Corporation 2016

Figure 7-22. Javacore overview

WA8152.2

Notes:

A javacore file is a snapshot of a running Java process. The IBM Java SDK produces a javacore file in response to specific operating system signals. Javacore files show the state of every thread in the Java process, in addition to the monitor information that identifies Java synchronization locks. If deadlocks are detected, it is noted in the file.

It provides various storage management values (in hexadecimal), including the free space in heap, the size of current heap, and details on other internal memory that the JVM is using. It also contains garbage collection history data. This feature is new for JDK 5.

You can use javacore files to resolve the problems that are listed here. More information on capturing data for problem determination can be found in the MustGather documents for these problems:

- 100% CPU usage
- Crash
- Hang or performance degradation

Javacore files can also provide an initial insight for memory problems.

Javacore file location and naming

- The javacore file is stored in the first viable location of:
 - Xdump:java:file=<path_name>/<filename.txt> (command-line argument)
 - The setting of the IBM_JAVACOREDIR environment variable (deprecated)
 - <WAS_install_root>/profiles/<profile>
 - TMPDIR or TEMP environment variable
 - Windows only: If the javacore cannot be stored in any of the above, it is put to STDERR
 - A new option –Xdump:java:defaults:file=<path/filename> can be used to change the default path and name of all javacore dump agents
- Javacore naming
 - Windows and Linux: javacore.YYYYMMDD.HHMMSS.PID.txt where YYYY = year, MM = month, DD = day, SS = second, PID = processID
 - AIX: javacorePID.TIME.txt where PID = processID, TIME = time since 1/1/1970
 - z/OS: Javadump.YYYYMMDD.HHMMSS.PID.txt where YYYY = year, MM = month, DD = day, SS = second, PID = processID

© Copyright IBM Corporation 2016

Figure 7-23. Javacore file location and naming

WA8152.2

Notes:

A javacore file can be generated on demand through the wsadmin command line interface:

1. From the command prompt, enter the command wsadmin.bat/sh to get a wsadmin command prompt.
2. Get a handle to the problem application server. (Note the contents in brackets "[...]", along with the brackets, are not optional. It must be entered to set the JVM object. Also, notice that there is a space between completeObjectName and type):


```
wsadmin> set jvm [$AdminControl completeObjectName type=JVM,process=server1,*]
```
3. Generate the thread dump:


```
wsadmin>$AdminControl invoke $jvm dumpThreads
```

Javacore file subcomponents

Subcomponent	Description
TITLE	Shows basic information about the event that caused the generation of the javacore, the time it was taken, and the file name.
GPINFO	Shows general information about the OS. If the memory dump resulted from a general protection fault (GPF), information about the fault module is provided.
ENVINFO	Shows information about the JRE level, details about the command line that started the JVM process, and the JVM environment.
NATIVEMEMINFO	Provides information about the native memory that the Java Runtime Environment (JRE) allocates. Requested from the OS by using library functions such as malloc() and mmap().
MEMINFO	Shows the free space in heap, the size of current heap, details on other internal memory that the JVM is using, and garbage collection history data.
LOCKS	Shows the locks threads hold on resources, including other threads. A lock (monitor) prevents more than one entity from accessing a shared resource.
THREADS	Identifies the current thread, provides a complete list of Java threads that are alive, and provides their stack traces.
CLASSES	Shows information about the class loaders and specific classes loaded.

© Copyright IBM Corporation 2016

Figure 7-24. Javacore file subcomponents

WA8152.2

Notes:

GPF stands for general protection fault.

Javacore example (JDK v6)

```

0SECTION      TITLE subcomponent dump routine
NULL
1TICCHARSET   =====
1TISIGINFO    1252
1TIDATETIME   Dump Requested By User (00100000) Through com.ibm.jvm.Dump.JavaDump
1TIFILENAME   Date:          2012/03/06 at 17:08:36
               Javacore filename: C:\ProgramFiles\IBM\WebSphere\AppServer\profiles\
                           profile1\javacore.20120306.170836.6404.0001.txt
1TIREQFLAGS   Request Flags: 0x81 (exclusive+preempt)
1TIPREPSTATE  Prep State:   0x106 (vm_access+exclusive_vm_access+)
NULL
0SECTION      GPINFO subcomponent dump routine
NULL
2XHOSLEVEL    OS Level       : Windows XP 5.1 build 2600 Service Pack 3
2XHCPUUS     Processors - 
3XHCPUARCH   Architecture  : x86
3XHNUMCPUS   How Many      : 1
3XHNUMASUP   NUMA is either not supported or has been disabled by user
1XERROR2     Register dump section only produced for SIGSEGV, SIGILL or SIGFPE.
NULL
0SECTION      ENVINFO subcomponent dump routine
NULL
1CIJAVAVERSION JRE 1.6.0 Windows XP x86-32 build 20111113_94967 (pwi3260_26sr1-
20111114_01(SR1))
1CIVMVERSION  VM build R26_Java626_SR1_20111113_1649_B94967
1CIJITVERSION r11_20111028_21230
1CIGCVERSION  GC - R26_Java626_SR1_20111113_1649_B94967
1CIJITMODES   JIT enabled, AOT disabled, FSD disabled, HCR disabled
1CIRUNNINGAS  Running as a standalone JVM

```

© Copyright IBM Corporation 2016

Figure 7-25. Javacore example (JDK v6)

WA8152.2

Notes:

Verbose garbage collection (GC)

- Verbose GC is an option that the JVM run time provides
- Provides a garbage collection log:
 - Interval between collections
 - Duration of collection
 - Whether compaction was required
 - Memory size, memory that was freed, memory available
- Select **Servers > Server Types > WebSphere application servers > <serverName>**
- Under Server Infrastructure, click **Java and Process Management > Process Definition > Java Virtual Machine**
 - On the **Configuration** tab, select the **Verbose Garbage Collection** check box, restart server
 - On the **Runtime** tab, select the **Verbose Garbage Collection** check box, effective for current server instance
- IBM JVM writes to `native_stderr.log`

© Copyright IBM Corporation 2016

Figure 7-26. Verbose garbage collection (GC)

WA8152.2

Notes:

It is often a good idea to have verbose GC enabled permanently in production. The performance cost on a reasonably well-tuned JVM is small. The benefits of having it on when something happens are considerable (no need to reproduce the problem a second time after enabling). It is also good to keep an eye on the verbose GC regularly, as a way to monitor the health of the system, even when nothing bad is noticed.

Enabling verbose GC by default is a decision that each system administrator must make consciously. But it is no longer plainly “not recommended as a normal production setting.”

It is possible to enable verbose GC output from the **Runtime** tab. This setting allows the initiation of verbose GC output on a running application server.

What is a java.lang.OutOfMemoryError?

- Java virtual machine error
- Not enough memory to allocate an object; some of the causes are:
 - Java heap is too small
 - Memory is available in the heap, but it is fragmented (rare for Java 6)
 - Memory leak in the Java code
 - Not enough space in the native memory
- If available, first analyze the javacore file for memory issues:
 - Verify OutOfMemory as the cause for the javacore
 - Check heap size information
- Generate and analyze garbage collection and memory information:
 - Verbose GC
 - Javacore
 - Heap dump
 - System dumps

© Copyright IBM Corporation 2016

Figure 7-27. What is a java.lang.OutOfMemoryError?

WA8152.2

Notes:

Symptoms of memory leak in the Java code

- Regardless of the JVM maximum heap, the heap is still going to run out of memory
- Increasing the maximum heap size merely causes the problem to take longer to occur
- One or more objects are taking up a high percentage of the JVM heap:
 - A few large objects
 - Thousands of instances of small objects
- Memory leaks can also occur in native code
 - “Iceberg objects”: Threads, classes, and direct bytebuffers that have a small Java heap footprint, but a large native heap footprint

© Copyright IBM Corporation 2016

Figure 7-28. Symptoms of memory leak in the Java code

WA8152.2

Notes:

JVM tool overview

- Assess the status of a running Java application
 - IBM Monitoring and Diagnostic Tools for Java – Health Center
 - The Health Center can also be used to monitor processor usage and lock contention
- WebSphere internal thread pools and heap usage statistics
 - Tivoli Performance Viewer
- Thread activity snapshot: Use javacore files
 - IBM Thread and Monitor Dump Analyzer for Java
- Memory and garbage collection: Use verbose GC output
 - IBM Monitoring and Diagnostic Tools for Java – Garbage Collection and Memory Visualizer (GCMV)
 - IBM Pattern Modeling and Analysis tool for Java Garbage Collector
- Memory use by object: Use heap dumps
 - Memory Analyzer tool (MAT)
 - IBM HeapAnalyzer

© Copyright IBM Corporation 2016

Figure 7-29. JVM tool overview

WA8152.2

Notes:

Most tuning and debugging can be accomplished with free tools that many customers have on hand or ones available from the IBM Support Assistant. In addition, high-end tools can also be used, such as IBM Tivoli Composite Application Manager for WebSphere or other third-party products.

There are tools available to parse system core files.



Thread and Monitor Dump Analyzer (TMDA)

The screenshot shows the IBM Thread and Monitor Dump Analyzer for Java interface. The left pane displays a list of threads with their names, states (Runnable, Blocked, Waiting), and methods. A red arrow points from the 'WebContainer : 2' entry in the list to the right pane. The right pane has three sections: 'Waiting Threads : 3' (listing WebContainer: 0, 1, 3), 'Thread Detail' (showing 'WebContainer : 2' with State 'Deadlock/Blocked' and Monitor 'Owning Monitor Lock on java/lang/Object@023EF880/023EF88C Waiting for Monitor Lock on java/lang/Object@023EF870/023EF87C'), and 'Blocked by : 1' (listing WebContainer: 1). A callout box highlights the following features:

- Import javacore file
- Multiple javacores can be imported and compared
- Useful for troubleshooting hung threads

© Copyright IBM Corporation 2016

Figure 7-30. Thread and Monitor Dump Analyzer (TMDA)

WA8152.2

Notes:

In the left pane, each thread name can be selected, and the details of the thread are displayed in the right pane.

Notice that since the WebContainer: 2 thread is selected in the screen capture. The middle pane shows threads that are waiting and the threads that are currently blocked by WebContainer: 2.

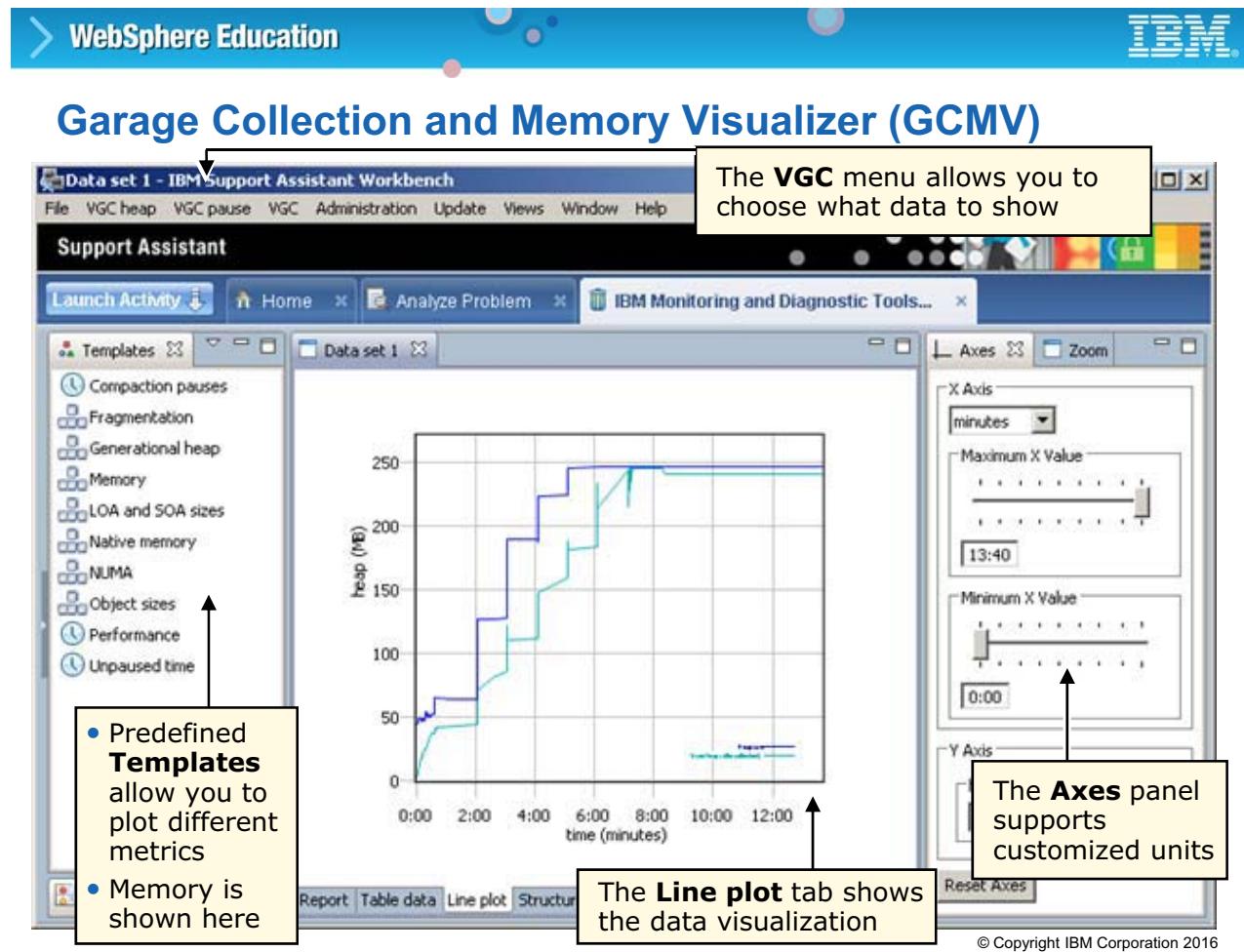


Figure 7-31. Garage Collection and Memory Visualizer (GCMV)

WA8152.2

Notes:

Templates

The IBM Monitoring and Diagnostic Tools for Java - Garbage Collection and Memory Visualizer (GCMV) provides the following templates, which cover a range of common collections of data types to plot:

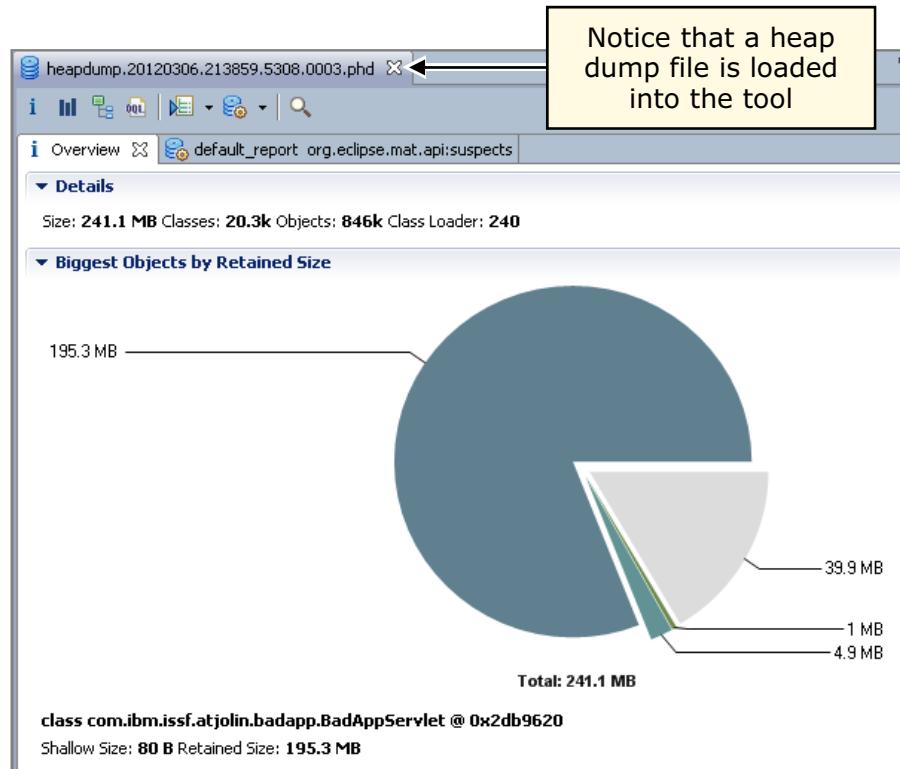
- **Memory:** Plots Heap size, Used heap (after collection), and JVM restarts. Useful for diagnosing memory leaks or poor choice of heap size.
- **Performance:** Plots Mark, Sweep, Pause, Compact times, and JVM restarts. Useful for investigating performance issues.
- **Unpaused Time:** Plots Intervals between garbage collection triggers and Pause times. This template can help determine whether an application is spending long periods of time in garbage collection rather than running.
- **Object Sizes:** Plots Heap size, Used heap (after collection), and Requested object sizes that trigger allocation failures. This template can help determine whether an application is requesting large objects and triggering excessive garbage collection.

- **LOA and SOA Sizes:** Plots Heap size, Used SOA (small object area), Used LOA (large object area), Used tenured heap, and Used nursery heap (after collection). This template can help determine whether there is a problem with the `-Xlорatio` setting.
- **Generational Heap:** Plots Heap size, Free heap, Tenured heap size, Free tenured heap, Nursery size, Free nursery heap, and Tenure age. Useful for diagnosing memory leaks or poor nursery or tenured heap sizings.
- **Fragmentation:** Plots Heap size, Free heap (before collection), and Free heap (after collection). A large amount of free heap before collection might indicate heap fragmentation. A small amount of free heap after collection might indicate that the heap size is too small or the application is holding onto too much data.
- **Compaction Pauses:** Plots Heap size, Used heap (after collection), Compact times, and Pause times. This template can help identify whether a restricted heap is causing excessive time that is spent in compaction.
- **Native Memory:** Plots native memory usage. Useful for diagnosing out-of-memory errors.
- **NUMA:** Non-uniform memory architecture information.

WebSphere Education

Memory Analyzer overview

- Pie chart of biggest objects and links to common analysis options
 - Histogram
 - Denominator tree
 - Top consumers
 - Duplicate classes



© Copyright IBM Corporation 2016

Figure 7-32. Memory Analyzer overview

WA8152.2

Notes:

This screen capture shows the Overview tab, which the Memory Analyzer displays after it analyzes a heap dump. This view presents a pie chart of the largest objects by retained size (reach size).

After an OutOfMemoryError was thrown, WebSphere generated the heap dump used for this analysis.

- **Histogram:** Lists number of instances per class
- **Denominator Tree:** Lists the biggest objects and what they keep alive
- **Top Consumers:** Prints the most expensive objects that are grouped according to class and package
- **Duplicate Classes:** Detects classes that multiple class loaders load

There is no exact algorithm for memory analysis. Use the overview, histogram, and dominator tree functions to do initial analysis. Use reports and queries for more detailed analysis.

When you open a heap dump file, the Overview tab provides a first analysis of the heap dump.

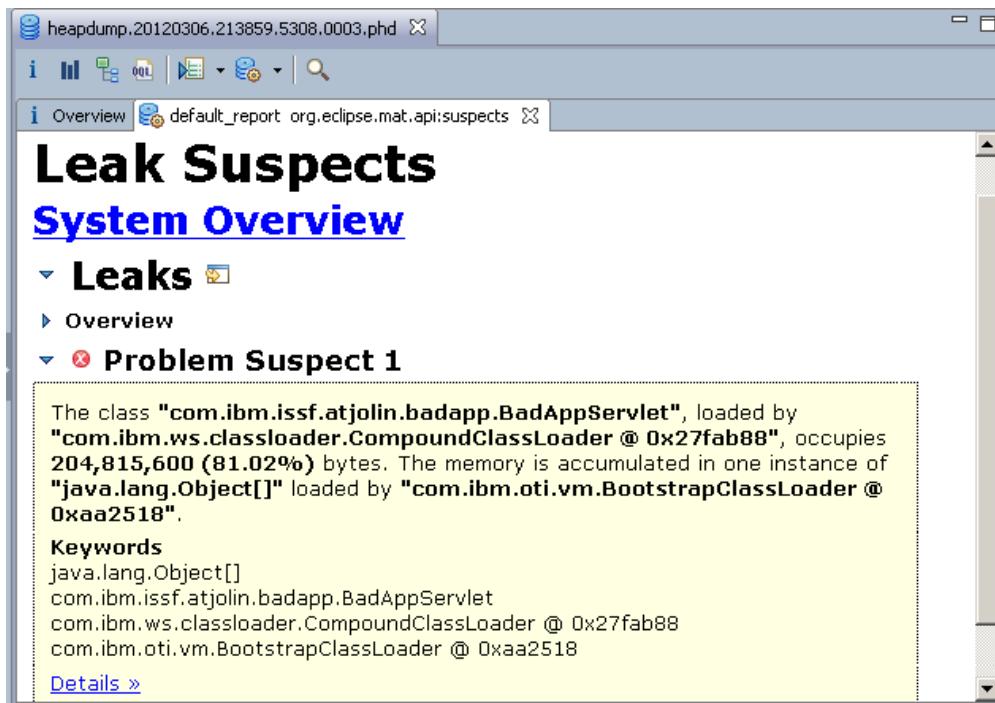
The general procedure is as follows:

1. Examine the overview.
2. Check the size of the heap dump, and the numbers of objects, classes, and class loaders that it contains.
3. Look at the biggest objects.
4. In the **Overview** tab, click the **Histogram** link. Use the Histogram view to group objects, examine or calculate the size of those objects, or run queries on those objects.
5. In the **Overview** tab, click the **Dominator Tree** link. Use the Dominator Tree view to find the largest objects, and to examine the dependencies between objects.



Memory Analyzer leak suspects: Default report

- One or more leak suspects are listed



© Copyright IBM Corporation 2016

Figure 7-33. Memory Analyzer leak suspects: Default report

WA8152.2

Notes:

The Leak Suspects view helps you to identify possible memory leak suspects. The report contains the same information as the heap dump overview report, and also a section about possible leak suspects. A summary pie chart shows the leak suspects and the proportion of the heap that each suspect is consuming.

Clicking the **Details** link reveals the call tree views for each of the suspects and more information such as:

- Shortest paths to the accumulation point
- Accumulated objects
- Accumulated objects by class



Health Center

- The IBM Monitoring and Diagnostic Tools for Java - Health Center is a diagnostic tool for monitoring the status of a running JVM
 - Available in the IBM Support Assistant
- The Health Center uses a small amount of processor time and memory, thus a small impact on the application's performance
- The tool is provided in two parts:
 - The Health Center agent that is part of the WebSphere run time and collects data from an application server
 - An Eclipse-based client that connects to the agent
- The client interprets the data and provides recommendations to improve the performance of the monitored application server

© Copyright IBM Corporation 2016

Figure 7-34. Health Center

WA8152.2

Notes:

Health Center provides visibility, monitoring and profiling in the following application areas:

- **Performance**
 - **Java method profiling:** The Health Center uses a sampling method profiler to diagnose applications that show high CPU usage. It is a low overhead, which means there is no need to specify in advance which parts of the application to monitor, the Health Center simply monitors everything. It works without recompilation or byte code instrumentation and shows where the application is spending its time, by giving full call stack information for all sampled methods.
 - **Lock analysis:** Synchronization can be a significant performance bottleneck on multi-CPU systems. It is often difficult to identify a hot lock or to assess the impact that locking is having on your application. Health Center records all locking activity and identifies the objects with most contention. Health Center analyses this information, and uses it to provide guidance about whether synchronization is impacting performance.
 - **Garbage collection:** The performance of Garbage Collection (GC) affects the entire application. Tuning GC correctly can potentially deliver significant performance gains.

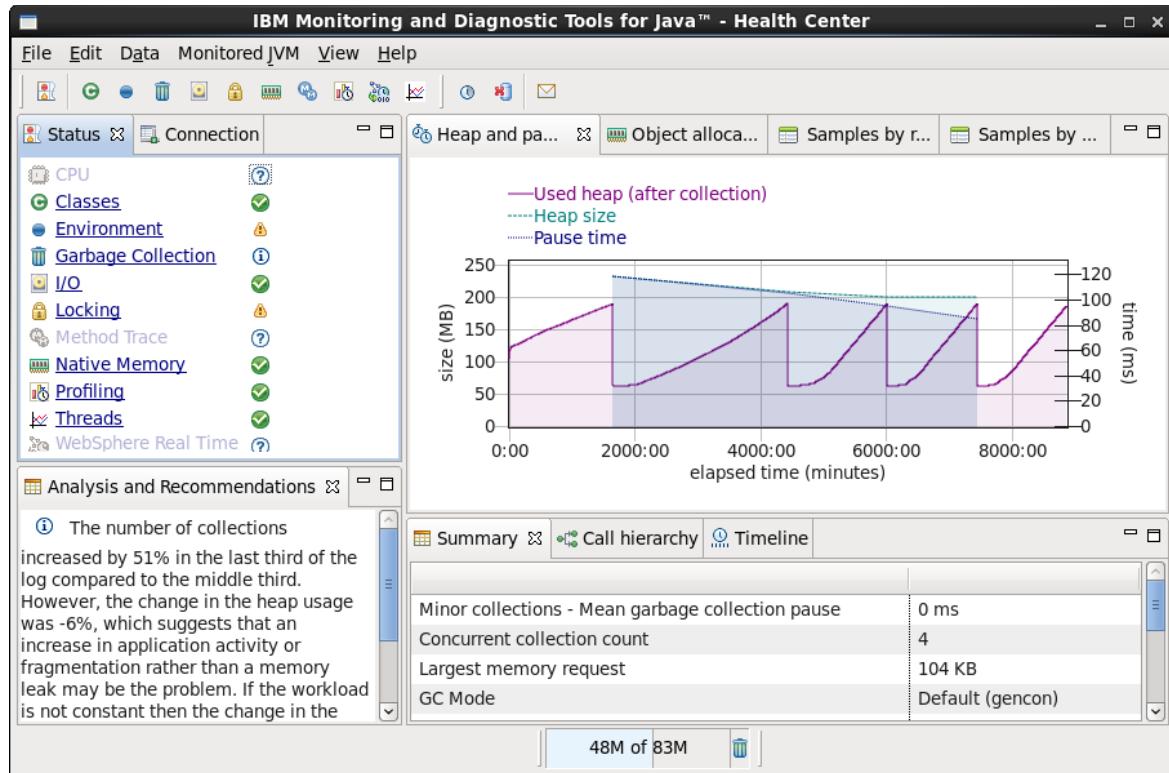
Health Center identifies where garbage collection is causing performance problems and suggests more appropriate command line options.

- **Threading:** Thread behavior in an application can have a major impact on your applications performance. Thread contention issues can often be difficult to find. Health Center identifies which threads are holding onto locks and which ones are blocked, helping the resolution of contention issues.
- **Memory usage** The Health Center identifies if your application is using more memory than seems reasonable, or where memory leaks occur. It then suggests solutions to memory issues, and Java heap sizing guidance.
- **System Environment** Health Center uses an 'environment perspective' to provide details of the Java version, Java classpath, boot classpath, environment variables, and system properties. This data is useful for identifying problems on remote systems or systems where you do not control the configuration. If the Health Center detects misconfigured applications, it provides recommendations on how to fix it.
- **Java Class loading** Health Center provides class loading information, showing exactly when a class is loaded and whether it is cached or not. This information helps you determine whether your application is affected by excessive class loading.
- **File input and output** Health Center uses an 'I/O perspective' to monitor application input or output (I/O) tasks as they run. You can use this perspective to monitor how many and which files are open, and to help solve problems such as when the application fails to close files.
- **Real-Time Monitoring** The WebSphere® Real-Time perspective (WRTP) helps you identify unusual or exceptional events that might occur when you run a WRT application. The trace information can be presented in various ways, including linear or logarithmic scales, and histograms. WRTP provides some pre-defined trace point views that are especially helpful.
- **Object Allocations** Health Center allows you to view the size, time, and code location of an object allocation request that meets specific threshold criteria.

<http://www.ibm.com/developerworks/java/jdk/tools/healthcenter/>



Health Center client



© Copyright IBM Corporation 2016

Figure 7-35. Health Center client

WA8152.2

Notes:

The Health Center client is split into subsystems, each representing a component of the JVM. The following subsystems are available:

- *Classes*: Information about classes that are loaded
- *Environment*: details of the configuration and system of the monitored application
- *Garbage collection*: Information about the Java heap and pause times
- *I/O*: Information about I/O activities that take place
- *Locking*: Information about contention on inflated locks
- *Memory*: Information about the native memory usage
- *Profiling*: provides a sampling profile of Java methods and includes call paths
- *WebSphere Real Time for Linux*: Information about real-time applications

Subsystems are represented as Eclipse perspectives. The first subsystem that you see is the Status perspective, listing the subsystems and their overall status. When you connect to a running application, subsystems with data available become links, and any recommendations are

displayed. The Health Center updates the displayed data and recommendations every 10 seconds. Switch to the subsystem perspectives by using the links or the toolbar icons. You can return to the Status perspective by using the farthest left toolbar icon.

The heap usage, pause times, summary table, and tuning recommendation sections are in the Health Center garbage collection perspective.

The Health Center garbage collection perspective has the following sections:

- A graph of heap usage
- A graph of pause times
- A summary table of important GC metrics
- Tuning recommendations
- A table that shows the call stacks for large object allocations

Heap usage

Use the graph of heap usage to identify trends in application memory usage. If the memory footprint is larger than expected, a heap analysis tool can identify areas of excessive memory usage. If the used heap is increasing over time, the application might be leaking memory. A memory leak happens when Java applications hold references to objects that are no longer required. Because these objects are still referenced, they cannot be collected and contribute to memory requirements. As the memory consumption grows, more processor resources are required for garbage collection, leaving fewer for application work. Eventually the memory requirements can fill the heap, leading to an `OutOfMemoryError` exception, and an application failure.

7.3. The HopSpot garbage collector

The HopSpot garbage collector



Reference slides for Oracle Solaris users

© Copyright IBM Corporation 2016

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

9.1

Figure 7-36. The HopSpot garbage collector

WA8152.2

Notes:



The HotSpot generational collector

- This collector is based on the same generational theory as the IBM version
- Its implementation predates the IBM generational collector
- It also uses two generations:
 - The young generation is typically smaller and is collected more often
 - The tenured generation is typically larger and collected less often
 - Third generation, the permanent, used to contain loaded classes and has its own tuning options

© Copyright IBM Corporation 2016

Figure 7-37. The HotSpot generational collector

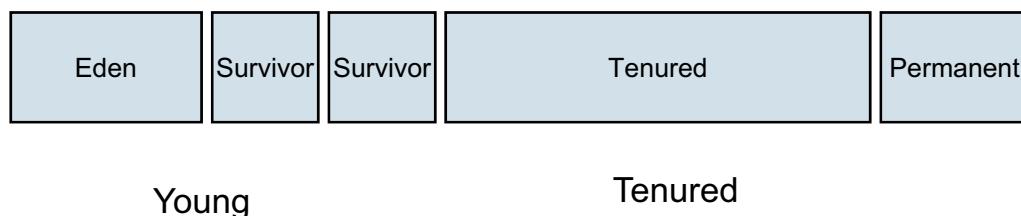
WA8152.2

Notes:

The permanent generation had a bad leak for a long time. It often shows up as a growing number of classes in memory. Starting and stopping an application in WebSphere can cause this leak to manifest. If you see this leak, you can increase the size of permanent space, but frequent starts and stops fill permanent space. To avoid the leak, recycle the entire JVM and not just the application.

HotSpot generations

- The young generation is divided into three spaces: Eden-space, and two survivor spaces (to-space, and from-space)
- Allocations are done from Eden-space and “from-space”
 - When those spaces fill, a young generation collection occurs
 - The surviving objects are copied to the “to-space” and thus compacted
 - Objects that exceed the age count are tenured
 - The pointers are flipped, and from-space and to-space switch roles
- The tenured generation is collected with a mark-sweep-compact collection
 - There is an option to collect the tenured generation concurrently



© Copyright IBM Corporation 2016

Figure 7-38. HotSpot generations

WA8152.2

Notes:

The architecture of the Sun-developed, HP-ported HotSpot Java virtual machine (JVM) evolved differently than the IBM-developed software development kit (SDK). Its internal structure, for young or old generation and permanent regions, arises to primarily support generational garbage collection and other garbage collection modes as necessary.

A tenured collection also tries to collect from permanent space.

This process looks much more complex than the IBM version, but the theory is the same.

Tuning HotSpot

- The tuning goals are the same as for the IBM collector
- The relationship between Eden, from, and to spaces are complex
- White papers on the Oracle website are helpful
- Many manual controls are available that you can use to adjust the collector's behavior
- Before tuning, you should understand what is happening inside the collector
- Verbose GC is good but harder to activate
- Oracle has a free download of a visual real-time GC tool (VisualGC)
- You can also use GC processing tools from IBM Support Assistant
 - IBM Monitoring and Diagnostic Tools for Java - Garbage Collection and Memory Visualizer (GCMV)
 - IBM Pattern Modeling and Analysis Tool for Java Garbage Collector (PMAT)

© Copyright IBM Corporation 2016

Figure 7-39. Tuning HotSpot

WA8152.2

Notes:

WebSphere support found that sizing the young or tenured generations correctly was 80% or more of the benefit. After that is accomplished, one is free to begin to study and control the more exotic parameters (later chart) but this study can turn into a science project.

Tuning: Verbose GC HotSpot

- In order to view garbage collection details, you must set several parameters on the JVM in addition to setting verbose GC:
 - `-XX:+PrintGCDetails`
 - `-XX:+PrintGCTimeStamps`
 - `-XLoggc:/path` (where path points to the target file)

© Copyright IBM Corporation 2016

Figure 7-40. Tuning: Verbose GC HotSpot

WA8152.2

Notes:

Any garbage collection decisions should be based on the behavior of the garbage collectors. You should identify the correct garbage collection mode to suit the operational needs of your application. You should also verify that you are meeting your performance requirements, and are efficiently recycling enough memory resources to consistently meet the demands of your application. Any changes that you make to garbage collection parameter settings should produce sufficiently different results and show benefits that are derived from using different regions of the HotSpot Java heap.

The `-XX:+PrintHeapAtGC` parameter generates a large amount of logging to the `native_stdout.log`. Ensure that there is adequate file system space when enabling this option. All of the information that this option produces is known to the JVM; therefore, no extra processing is required as a result of enabling this parameter. However, there is a slight amount of extra I/O performance cost, which should amount to less than 3%.

Verbose GC HotSpot: Output

- A typical verbose GC output line looks like:

```
[GC[DefNew: 8128K->64K(8128K), 0.0453670 secs] 13000K->7427K(32704K), 0.0454906 secs]
```

- The format is:

```
[GC [<collector>: <starting occupancy1> -> <ending occupancy1>, <pause time1> secs] <starting occupancy3> -> <ending occupancy3>, <pause time3> secs]
```

Where:

- <collector> is an internal name for the collector that is used in the minor collection
- <starting occupancy1> occupancy of the young generation before the collection
- <ending occupancy1> occupancy of the young generation after the collection
- <pause time1> pause time in seconds for the minor collection
- <starting occupancy3> occupancy of the entire heap before the collection
- <ending occupancy3> occupancy of the entire heap after the collection
- <pause time3> pause time for the entire garbage collection; includes the time for a major collection if one was done

© Copyright IBM Corporation 2016

Figure 7-41. Verbose GC HotSpot: Output

WA8152.2

Notes:

Gather and analyze data to evaluate the configuration, typically by using `verbosegc`. Continue to gather and analyze data as you make tuning changes until you are satisfied with the performance of the JVM.

Verbose GC HotSpot: Extra output

- If you add the following options (in addition to those options already active):
 - `-XX:+PrintGCApplicationConcurrentTime`
 - `-XX:+PrintGCApplicationStoppedTime`
- Then, two more lines are output to the verbose GC file:

```
[GC [DefNew: 8128K->64K(8128K), 0.0453670 secs] 13000K->7427K(32704K),  
0.0347865 secs]  
Total time for which application threads were stopped: 0.04677333 seconds  
Application time: 0.5564736 seconds  
[GC [DefNew:8128K->64K(8128K), 0.0453670 secs] 13000K->7427K(32704K),  
0.0495676 secs]
```

- These lines detail the elapsed time that GC blocked thread execution and the elapsed time that the application (all JVM work) ran between GC cycles

© Copyright IBM Corporation 2016

Figure 7-42. Verbose GC HotSpot: Extra output

WA8152.2

Notes:

This output really helps to understand the effects of your tuning efforts.

Tuning: Commerce Server: HotSpot

- In practice, the following parameters provide optimum performance in a HotSpot JVM running Commerce server and can be used as a starting point for your tuning efforts:
 - `-XX:+UseConcMarkSweepGC` (for the tenured generation)
 - `-XX:+CMSParallelRemarkEnabled`
 - `-XX:-UseParallelGC`
 - `-XX:-UseParallelOldGC`
 - `-server` (generic JVM argument to enable server optimization)
- Additionally, you might find via experimentation that a new generation size of half the total heap works

© Copyright IBM Corporation 2016

Figure 7-43. Tuning: Commerce Server: HotSpot

WA8152.2

Notes:

- **-XX:+UseConcMarkSweepGC**
Use concurrent mark-sweep collection for the tenured generation.
- **-XX:+UseParallelGC**
Use this parameter to enable the default throughput or parallel scavenge collector.
- **-XX:-UseParallelOldGC**
Use parallel garbage collection for the full collections. Enabling this option automatically sets **-XX:+UseParallelGC**

Controls: HotSpot

- `-XX:+HandlePromotionFailure` The youngest generation collection does not require a guarantee of full promotion of all live objects
- `-XX:+ScavengeBeforeFullGC` Do young generation GC before a full GC
- `-XX:-UseConcMarkSweepGC` Use concurrent mark-sweep collection for the old generation
- `-XX:-UseParallelGC` Use parallel garbage collection for scavenges
- `-XX:-UseParallelOldGC` Use parallel garbage collection for the full collections
- `-XX:MaxNewSize=size` Maximum size of new generation (in bytes)
- `-XX:MaxPermSize=64m` Size of the permanent generation
- `-XX:NewRatio=2` Ratio of new/old generation sizes
- `-XX:NewSize=2.125m` Default size of new generation (in bytes)
- `-XX:SurvivorRatio=8` Ratio of eden/survivor space size
- `-XX:TargetSurvivorRatio=50` Wanted percentage of survivor space that is used after scavenge

© Copyright IBM Corporation 2016

Figure 7-44. Controls: HotSpot

WA8152.2

Notes:

The latest HotSpot JVM can set targets for performance, and it attempts to self-adjust to these targets. You can set a target for maximum pause or maximum performance cost, and it tries to reach those targets.



Unit summary

Having completed this unit, you should be able to:

- Size the JVM heap
- Tune various GC policies
- Troubleshoot common JVM problems

© Copyright IBM Corporation 2016

Figure 7-45. Unit summary

WA8152.2

Notes:



Checkpoint questions

1. True or False: A javacore file is a text file that contains vital information about the running JVM process.
2. True or False: The initial heap size should always be set equal to the maximum heap size.
3. True or False: The default garbage collection policy is gencon, which is optimized for highly transactional workloads.
4. True or False: The Thread and Monitor Dump Analyzer tool is used to analyze Java heap dumps.

© Copyright IBM Corporation 2016

Figure 7-46. Checkpoint questions

WA8152.2

Notes:

Write your answers here:

- 1.
- 2.
- 3.
- 4.



Checkpoint answers

1. True
2. False: Sometimes it is advantageous to use a variable heap size where the initial and maximum heap sizes are not equal.
3. True
4. False: The Thread and Monitor Dump Analyzer tool is used to analyze javacore files.

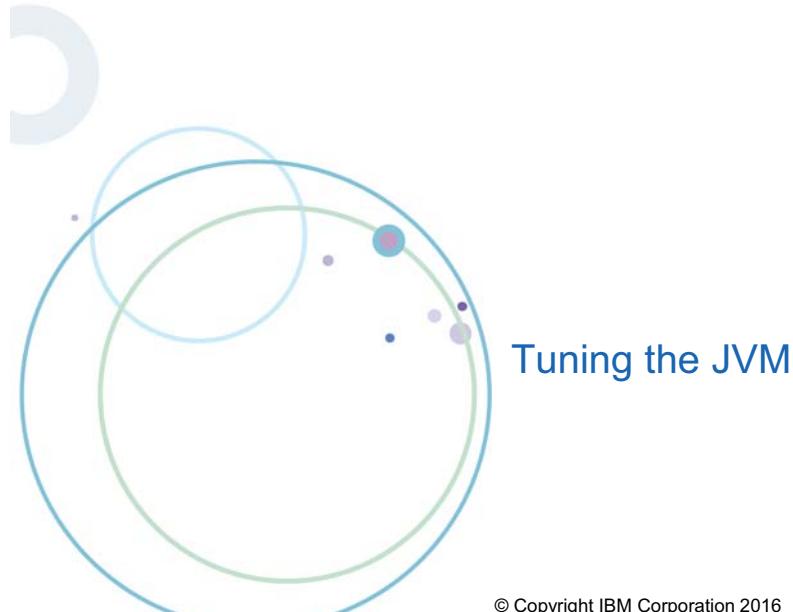
© Copyright IBM Corporation 2016

Figure 7-47. Checkpoint answers

WA8152.2

Notes:

Exercise 7



© Copyright IBM Corporation 2016

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

9.1

Figure 7-48. Exercise 7

WA8152.2

Notes:

Exercise objectives

After completing this exercise, you should be able to:

- Configure the initial and maximum JVM heap size
- Use Apache JMeter to load test various heap sizes
- Use Health Center to monitor garbage collection and heap usage
- Use GCMV to gather tuning recommendations for the JVM

© Copyright IBM Corporation 2016

Figure 7-49. Exercise objectives

WA8152.2

Notes:

Exercise 8



Troubleshooting JVM problems

© Copyright IBM Corporation 2016

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

9.1

Figure 7-50. Exercise 8

WA8152.2

Notes:



Exercise objectives

After completing this exercise, you should be able to:

- Use the IBM Support Assistant Health Center tool to monitor Java heap size and usage
- Detect and troubleshoot a server that is experiencing out-of-memory exceptions
- Use the IBM Support Assistant Memory Analyzer tool to analyze heap dumps
- Detect and troubleshoot a server with hung threads
- Use the IBM Support Assistant Thread Monitor Dump Analyzer tool to analyze thread dumps

© Copyright IBM Corporation 2016

Figure 7-51. Exercise objectives

WA8152.2

Notes:

Unit 8. Tuning the connection pool

What this unit is about

This unit describes connection pooling and how to tune it to maximize the chances that an application will be able to get a connection to a data source when it needs one.

What you should be able to do

After completing this unit, you should be able to:

- Configure connection pool parameters in the administrative console
- Use Tivoli Performance Viewer to monitor the connection pool
- Use the Tivoli Performance Viewer Performance Advisor to generate tuning advice
- Test, monitor, and tune a connection pool
- Monitor and tune the prepared statement cache

How you will check your progress

- Checkpoint questions
- Machine exercise

References

WebSphere Application Server ND V8.5 Information Center article Data access tuning parameters,
http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/index.jsp?topic=%2Fcom.ibm.websphere.base.doc%2Fe%2Frdat_datobjtune.html

Case study: Tuning WebSphere Application Server V7 for performance
http://www.ibm.com/developerworks/websphere/techjournal/0909_blythe/0909_blythe.html



Unit objectives

After completing this unit, you should be able to:

- Configure connection pool parameters in the administrative console
- Use Tivoli Performance Viewer to monitor the connection pool
- Use the Tivoli Performance Viewer Performance Advisor to generate tuning advice
- Test, monitor, and tune a connection pool
- Monitor and tune the prepared statement cache

© Copyright IBM Corporation 2016

Figure 8-1. Unit objectives

WA8152.2

Notes:



Topics

- Connection pools and properties
- Testing, monitoring, and tuning
- Prepared statement cache

© Copyright IBM Corporation 2016

Figure 8-2. Topics

WA8152.2

Notes:

8.1. Connection pools and properties

Connection pools and properties



© Copyright IBM Corporation 2016

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

9.1

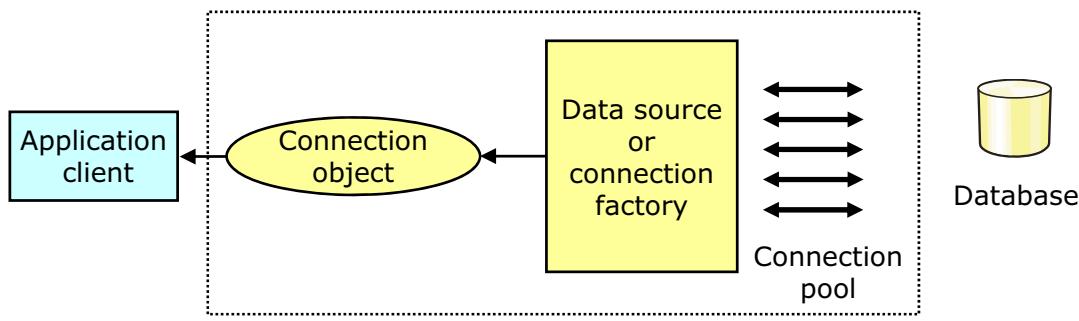
Figure 8-3. Connection pools and properties

WA8152.2

Notes:

What is connection pooling?

- The application server maintains a pool of ready-to-use connections to a data store
 - Application client requests a connection by using a data source or connection factory object
 - Connection is retrieved from pool
- Benefits:
 - Minimizes database session setup and tear-down
 - Improves application database access performance
 - Spreads out connection cost over repeated uses



© Copyright IBM Corporation 2016

Figure 8-4. What is connection pooling?

WA8152.2

Notes:

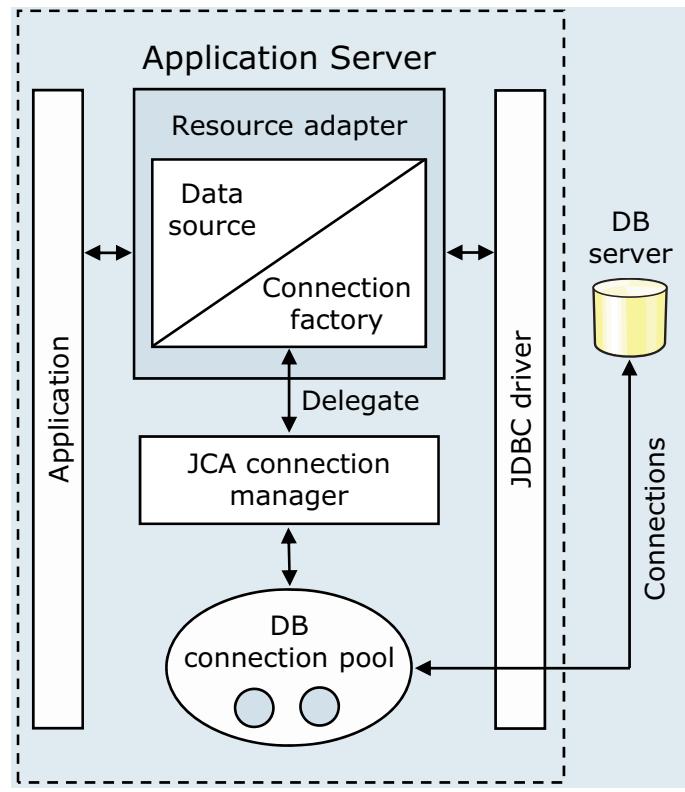
Creating a connection to a database is an expensive operation.

- Connections are large objects (1–2 MB each) and take up system resources.
- Creating a connection from scratch is relatively time consuming.

The goal of connection pooling is to improve the efficiency and performance of database access by maintaining a pool of readily available persistent connections to the database.

JCA connection pooling architecture

- Connection pooling supports two components:
 - JCA connection manager
 - Relational resource adapter
- How it works:
 - An application acquires a data source or connection factory object from the relational resource adapter
 - The data source or connection factory delegates the connection allocation request to the JCA connection manager
 - The JCA connection manager retrieves a free connection from the pool or creates a new one if none is available
 - The retrieved or created connection is returned to the application



© Copyright IBM Corporation 2016

Figure 8-5. JCA connection pooling architecture

WA8152.2

Notes:

J2C = Java EE Connector architecture

JDBC = Java database Connectivity

A resource adapter is a system-level software driver that a Java application uses to connect to an enterprise information system (EIS). A resource adapter connects with an application server and provides connectivity between the EIS, the application server, and the enterprise application.

WebSphere Application Server supports any resource adapter that implements version 1.0 or 1.5 of this specification.

The JCA connection manager provides connection pooling, local transaction support, and security services. The IBM WebSphere Application Server J2C component provides connection management that is based on the JCA specification. IBM supplies resource adapters for many enterprise systems separately from the WebSphere Application Server package.

WebSphere Application Server does provide the WebSphere Relational Resource adapter implementation that provides access to relational databases. This resource adapter provides data access through JDBC calls to access the database dynamically. The connection management is based on the JCA connection management architecture and provides connection pooling,

transaction, and security support. The WebSphere RRA is preinstalled and runs as part of WebSphere Application Server, and needs no further administration.

The relational resource adapter provides the JDBC wrapper or JCA Common Client Interface (CCI) implementation that allows applications to access the database JDBC driver.

The following steps describe how an application gets a connection from the pool:

1. An application acquires a data source or connection factory object from the relational resource adapter.
2. The data source or connection factory delegates the connection allocation request to the JCA connection manager.
3. The JCA connection manager retrieves a free connection from the pool or creates a new one if none is available.
4. The retrieved or created connection is returned to the application.

Types of connection pools in WebSphere

- JDBC provider connection pool
 - Enables access to a relational database
 - Provides a connection by using a JDBC data source
 - Managed in the administrative console under JDBC Providers
- JMS provider connection pool
 - Enables access to the data store used by the default messaging engine or a WebSphere MQ provider
 - Provides a connection by using a JMS connection factory
 - Managed in the administrative console under JMS Providers
- EIS connection pool
 - Enables access to enterprise information systems such as CICS, legacy databases such as IMS, and other back-end systems
 - Provides a connection by using a data source or connection factory
 - Managed in the administrative console under resource adapters
- The WebSphere J2C connection pool manager manages each one

© Copyright IBM Corporation 2016

Figure 8-6. Types of connection pools in WebSphere

WA8152.2

Notes:

JMS = Java Message Service

EIS = enterprise information system

CICS = Customer information Control System

IMS = Information Management System

The J2C connection pool manager pools and manages all connections to a JCA-compliant data store.

Support for WebSphere Version 4 data sources is also provided for compatibility with previous versions, but uses the old WebSphere connection manager architecture. This support is important because version 4 data sources have their own problem areas and generate their own set of error messages.

The WebSphere *default messaging engine* manages messaging resources and provides a pure JMS implementation that is fully integrated with the application server process. It uses a JDBC database to store messages, transaction states, and delivery records.

The need for tuning the connection pool

- Keeping connections in a pool consumes resources
 - Connections are large objects (1–2 MB each)
- An improperly tuned connection pool can result in:
 - Poor user response if the client is consistently waiting for a free connection
 - Application exceptions if the client cannot get a connection within the specified wait timeout interval
 - Reduced server throughput if unused connections are wasting system resources
- Connection pools need to be properly tuned to ensure optimal performance:
 - Maximize the chances that connections are available when needed
 - Minimize the number of idle connections
 - Minimize the number of orphaned connections

© Copyright IBM Corporation 2016

Figure 8-7. The need for tuning the connection pool

WA8152.2

Notes:

Keep in mind that each connection also ties up resources on the database server itself. If multiple application servers are connected to the same database server and one application server is tying up too many connections, other application servers might start seeing problems too.

Sometimes executing queries take longer than expected and cause `getConnection()` requests to fail due to low connection timeout values or not enough connections in the pool. If pool parameters are set too small or too large, performance problems result.

An orphaned connection is one that is allocated to a client but is not being used and is therefore abandoned, most likely because the client experiences an unexpected problem.

Key connection pool parameters

- Maximum connections
 - Specifies the maximum number of connections that can be created in the pool
 - Default value is 10
 - A value of 0 allows the number of physical connections to grow infinitely and causes the Connection timeout value to be ignored
- Minimum connections
 - Specifies the minimum number of physical connections to maintain
 - Default value is 1
 - If the size of the connection pool is at or below the minimum connection pool size, an Unused Timeout thread does not discard physical connections
 - However, the pool does not create connections solely to ensure that the minimum connection pool size is maintained
- Connection timeout
 - Specifies the interval, in seconds, after which a connection request times out and a `ConnectionWaitTimeoutException` is thrown
 - Default value is 180 seconds (3 minutes)
 - A value of 0 instructs the pool manager to wait as necessary until a connection becomes available

© Copyright IBM Corporation 2016

Figure 8-8. Key connection pool parameters

WA8152.2

Notes:

The pool contains physical connections to the back-end resource. When the maximum number is reached, no new connections are created, and the requester waits until one that used is returned to the pool. If connections are not returned before the *Connection timeout* time interval, a `ConnectionWaitTimeoutException` is thrown.

The *Connection timeout* value indicates the number of seconds that a request for a connection waits when there are no connections available in the free pool. If the maximum connections value is reached, no new connections are created. If a connection is not available within this time, the pool manager throws a `ConnectionWaitTimeoutException`.



Shareable connections

- Single connection that is used for multiple `getConnection()` calls within the same transactional context or container boundary
 - The connection is not released back to the pool even when closed
 - Connection is released when transaction or parent method completes
- Default behavior
 - Often more scalable, improved performance
- Common issues
 - Shared connections might be held too long, exhausting the pool, showing symptoms of a connection leak

© Copyright IBM Corporation 2016

Figure 8-9. Shareable connections

WA8152.2

Notes:

Connection pooling in WebSphere Application Server is managed according to the Java EE Connector Architecture (JCA) and enables the use of shareable or unshareable connections. In WebSphere Application Server, connections are shareable by default. The use of shareable connections means that, if conditions allow it, different `get connection` requests by the application receive a handle (indirectly) for the same physical connection to the resource. The benefits of this behavior are improved performance and a reduction in the number of physical connections that need to be managed.

When the application closes a shareable connection, it is not returned to the free pool. Rather, it remains ready for another request within the same transaction for a connection to the same resource. Shareable connections that an application obtains within a local transaction containment are kept reserved in the shared pool for use within that local transaction containment until it ends. The connections remain in the shared pool even if the application explicitly closes the connection. This behavior is beneficial for many applications, but can have unintended consequences for others.

The use of shareable connections can provide performance benefits in many situations, but you need to be aware of this default behavior and take it into account when developing applications.

Otherwise, the application might experience problems under heavy workload. Possible drawbacks of shared connections include:

- Sharing within a single component (such as an enterprise bean and its related Java objects) is not always supported.
- The current specification allows resource adapters the choice of allowing only one active connection handle at a time.
- A connection might be held long after the application has called the `close` method.

Unshareable connections

- Each `getConnection()` call results in a connection that is allocated from the connection pool
- The connection is released back to the pool upon a `close()` call
- Many different connections can be used within the same transaction
- When to use unshareable connections:
 - You do things to the connection that could affect another application that is sharing the connection
 - For example, unexpectedly changing the isolation level
- Common issues:
 - Unshared connections might exhaust the pool as many might be used within a transaction
 - Wasted connections

© Copyright IBM Corporation 2016

Figure 8-10. Unshareable connections

WA8152.2

Notes:

Possible drawbacks of unshared connections include:

- Inefficient use of your connection resources. For example, if within a single transaction you get more than one connection, then you use multiple physical connections when you use unshareable connections.
- Wasted connections. It is important not to keep the connection handle open (that is, your application does not call the `close()` method) any longer than it is needed. While an unshareable connection is open, the physical connection is unavailable to any other component, even if your application is not currently using that connection. Unlike a shareable connection, an unshareable connection is not closed at the end of a transaction or servlet call.

Connection pool parameters in the administrative console

General Properties

Scope
cells:was7:host01Cell01:nodes:was7:host01Node01:servers:server1

* Connection timeout
180 seconds

* Maximum connections
10 connections

* Minimum connections
1 connections

* Reap time
180 seconds

* Unused timeout
1800 seconds

* Aged timeout
0 seconds

Purge policy
EntirePool

EntirePool
FailingConnectionOnly

Apply OK Reset Cancel

© Copyright IBM Corporation 2016

Figure 8-11. Connection pool parameters in the administrative console

WA8152.2

Notes:

The following connection pool properties can be configured in the administrative console:

- **Minimum connections:** Specifies the minimum number of physical connections to be maintained. Until this number is reached, the pool maintenance thread does not discard any physical connections. However, no attempt is made to bring the number of connections up to this number. The default value is one connection.

For example, if *Minimum connections* property is set to three, and one physical connection is created, the Unused timeout thread does not discard that connection. By the same token, the thread does not automatically create two more connections to reach the *Minimum connections* setting.

- **Reap time:** Specifies the interval, in seconds, between runs of the pool maintenance thread. The default value is 180 seconds.

For example, if *Reap time* is set to 60, the pool maintenance thread runs every 60 seconds. The *Reap time* interval affects the accuracy of the *Unused timeout* and *Aged timeout* settings. The smaller the interval you set, the greater the accuracy.

When the pool maintenance thread runs, it discards any connections that are unused for longer than the time value specified in *Unused timeout* until it reaches the number of connections that are specified in *Minimum connections*. It also discards any connections that remain active longer than the time value specified in *Aged timeout*.

- **Unused timeout:** Specifies the interval in seconds after which an unused or idle connection is discarded. The default value is 1800 seconds.

For example, if the *Unused timeout* value is set to 120, and the pool maintenance thread is enabled (*Reap time* is not 0), any physical connection that remains unused for 120 seconds (2 minutes) is discarded. The Reap time value affects the accuracy of this timeout, and its performance.

- **Aged timeout:** Specifies the interval in seconds before a physical connection is discarded, regardless of recent usage activity. The default value is 0, which indicates that active connections are to remain in the pool indefinitely.

For example, if the Aged timeout value is set to 1200, and the Reap time value is not 0, any physical connection that remains in existence for 1200 seconds (20 minutes) is discarded from the pool. The Reap time value affects the accuracy of this timeout, and its performance.

- **Purge policy:** Specifies how to purge connections when a stale connection or fatal connection error is detected. Valid values are *EntirePool* (the default) and *FailingConnectionOnly*.

If you use *EntirePool*, all physical connections in the pool are destroyed when a stale connection is detected. Final destruction of connections that are in use at the time of the error might be delayed. However, these connections are never returned to the pool.

If you choose *FailingConnectionOnly*, the pool manager attempts to destroy only the stale connection.

 WebSphere Education



Monitoring the connection pool with Tivoli Performance Viewer

Metric name	Description	What to look for
PoolSize	Size of the connection pool	<ul style="list-style-type: none"> Increases as new connections are created (up to the value of <i>maximum connections</i>) and decreases when connections are destroyed A significant number of creates and destroys is an indication that the pool size (<i>maximum connections</i>) should be adjusted Counter is already enabled as part of the <i>basic</i> (default) PMI statistic set
PercentUsed	Average percent of the pool that is in use	<ul style="list-style-type: none"> If consistently low, you might want to decrease the pool size Counter is already enabled as part of the <i>basic</i> (default) PMI statistic set
WaitingThreadCount	Average number of threads that are concurrently waiting for a connection	<ul style="list-style-type: none"> The optimal value for the pool size is that which reduces this value Counter is already enabled as part of the <i>basic</i> (default) PMI statistic set
PercentMaxed	Average percent of the time that all connections are in use	<ul style="list-style-type: none"> Ensure you are not consistently maxed at 100% Counter requires the selection of either <i>all</i> or <i>custom</i> PMI statistic set

© Copyright IBM Corporation 2016

Figure 8-12. Monitoring the connection pool with Tivoli Performance Viewer

WA8152.2

Notes:

PMI collects these metrics and they are available for display in Tivoli Performance Viewer under the *JDBC Connection Pools* and *JCA Connection Pools* modules.

To access these modules, in the administrative console:

1. Navigate to **Monitoring and Tuning > Performance Viewer > Current Activity**.
2. Select the server that you want to monitor.
3. Expand **Performance Modules > JDBC Connection Pools or JCA Connection Pools**.

To view the collected metrics for a JDBC data source:

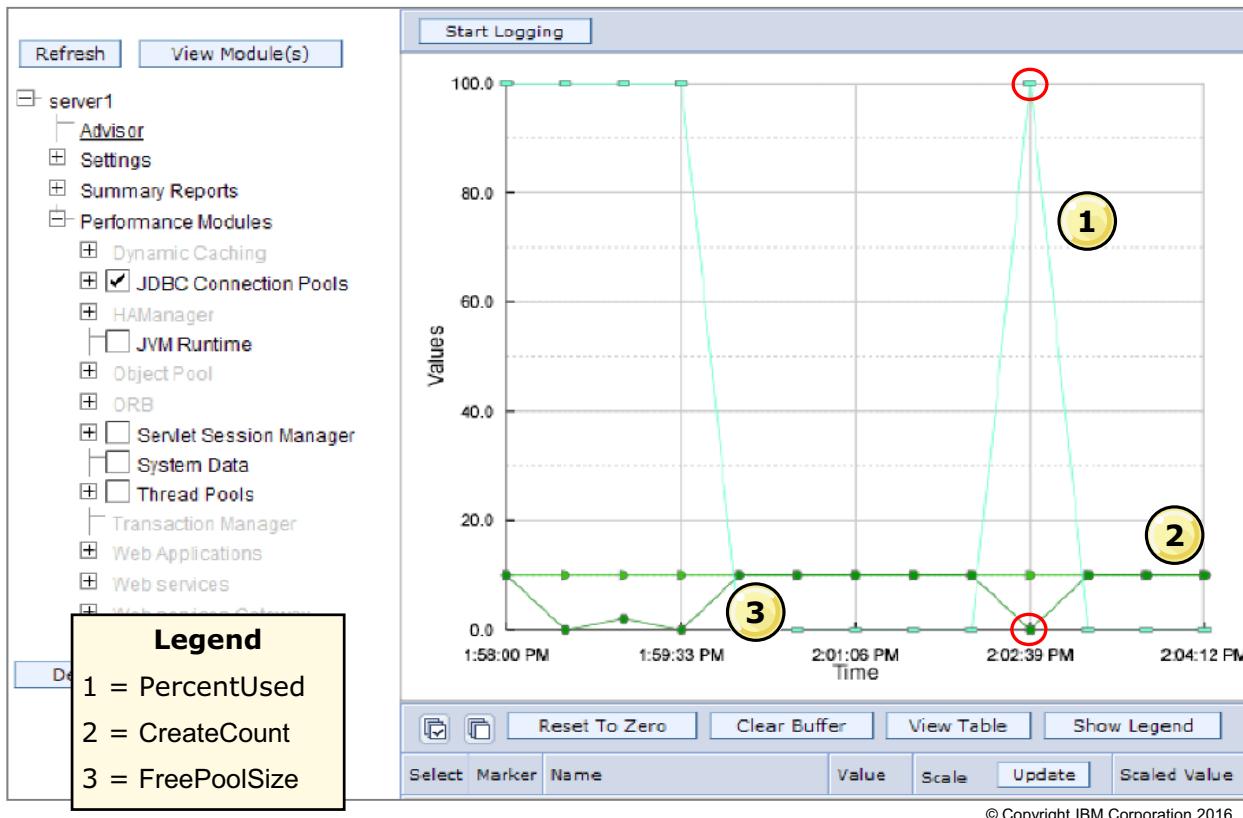
1. Expand the JDBC provider name for the data source.
2. Select the check box for the data source. A chart appears showing graphs for selected data source counters.

For a complete list of the available PMI connection pool counters, refer to the WebSphere product documentation chapter on “J2C connection pool counters” at:

http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/index.jsp?topic=%2Fcom.ibm.websphere.nd.doc%2Fae%2Frprf_datacounter3.html

Tivoli Performance Viewer connection pool monitoring



© Copyright IBM Corporation 2016

Figure 8-13. Tivoli Performance Viewer connection pool monitoring

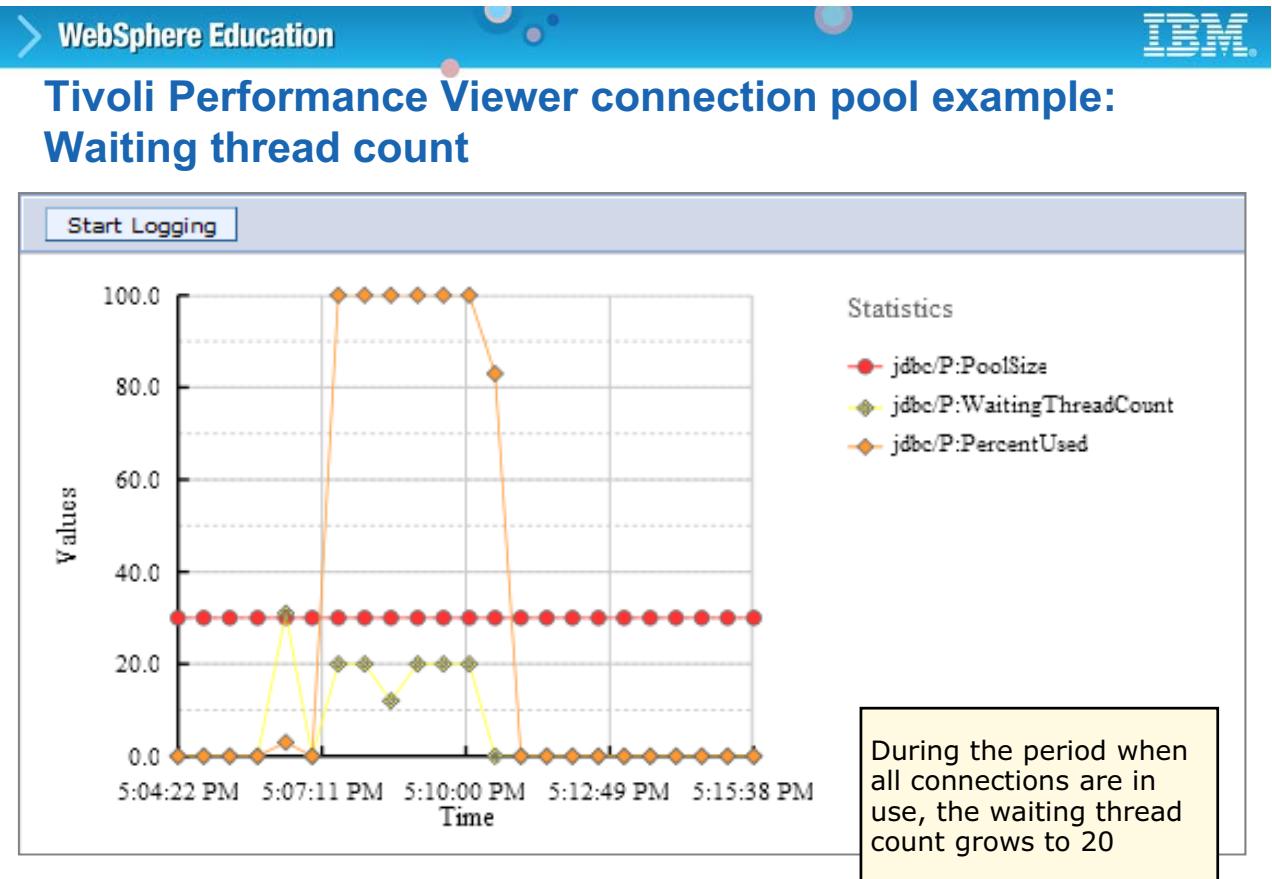
WA8152.2

Notes:

This screen capture shows an example of connection pool graphs for a data source:

- The cyan colored graph (1) plots the *PercentUsed* metric (average percent of the pool that is in use).
- The green graph (2) plots the *CreateCount* metric (total number of connections created). Ten connections are created, reaching the default *Maximum* connections value for the pool.
- The dark green graph (3) plots the *FreePoolSize* metric (number of free connections in the pool).

Notice that, as expected, when the *FreePoolSize* is 0, indicating that no connection is available in the pool, the *PercentUsed* value is at 100%.



© Copyright IBM Corporation 2016

Figure 8-14. Tivoli Performance Viewer connection pool example: Waiting thread count

WA8152.2

Notes:

This slide shows a chart graphing connection pool metrics over time. The metrics that are graphed in this chart are:

- Pool size (which remains constant)
- Percent of connections used (which reaches 100% for several minutes)
- Waiting thread count (which grows to 20 during the 100% connection usage period).

If this behavior is consistent under load testing, then the maximum connection pool size should probably be increased.



Generating tuning advice with Tivoli Performance Viewer Performance Advisor

- Tivoli Performance Viewer Performance Advisor can provide configuration advice for connection pool size:
 - Advice appears in the Performance Advisor section of Tivoli Performance Viewer
 - Based on collected PMI data over the last 1-minute interval
 - Uses IBM-defined rules of thumb for advice basis
- Limitations:
 - Pool sizing advice might not be generated if your timeout values are too high (pools are not returning to minimum values)
 - Tivoli Performance Viewer Advisor gives recommendations only when CPU usage is greater than or equal to 50%

© Copyright IBM Corporation 2016

Figure 8-15. Generating tuning advice with Tivoli Performance Viewer Performance Advisor

WA8152.2

Notes:

WebSphere provides two separate tuning advisors:

- Tivoli Performance Viewer Performance Advisor: Runs on demand and outputs advice to the Tivoli Performance Viewer graphical user interface in the administrative console
- Performance and Diagnostic Advisor (Runtime Performance Advisor): Runs in the background and outputs advice to SystemOut.log and the administrative console under **Troubleshooting > Runtime Messages > Warning**

For the Performance and Diagnostic Advisor, it is possible to configure the **Minimum CPU** for Working System parameter to specific the threshold at which usage advice is generated. This configuration allows for a lower threshold of 50% (the default value).

The following examples illustrate the advice that Tivoli Performance Viewer Performance Advisor would give in certain situations:

- **Situation:** The number of connections is low (equal to *Minimum connections*).
Advice: Decrease pool size.
- **Situation:** All data source connections are heavily used and heap space is available.

Advice: Increase maximum pool size.

- **Situation:** The size of the pool is fluctuating a lot (high variance), possibly indicating batch processing, and wasted resources.

Advice: Decrease pool size.

To run the Tivoli Performance Viewer Performance Advisor, in the administrative console:

1. Navigate to **Monitoring and Tuning > Performance Viewer > Current Activity**.
2. Select the server for which you want to generate advice.
3. Click **Advisor** and look at the generated advice list.

Message

TUNE5033W: For data source jdbc/NoTxTradeDataSource, the maximum connection pool size is unusually large. Typically, the connection pool size is no more than 30.

Severity

Config

Description

In general, a very large connection pool reduces performance, although it might be necessary for some applications.

User Action

To change the connection pool properties, open the administrative console and click JDBC Providers > JDBC_provider > Data Sources > data_source > Connection pool properties. See the information center for more information about queuing.

Detail

Currently, the size of the minimum connection pool is 10 and the maximum pool size is 50.

© Copyright IBM Corporation 2016

Figure 8-16. Tivoli Performance Viewer Performance Advisor example

WA8152.2

Notes:

The Tivoli Performance Viewer Performance Advisor generates a table of messages according to different severity levels: Alert and Config. In this example the message is: TUNE5033W: For data source jdbc/NoTxTradeDataSource, the maximum connection pool size is unusually large. Typically, the connection pool size is no more than 30. The Tivoli Performance Viewer Performance Advisor also provides Description, User Action, and Detail.

For this example:

- **Severity:** Config
- **Description:** In general, a large connection pool reduces performance, although it might be necessary for some applications.
- **User action:** To change the connection pool properties, open the administrative console and click **JDBC Providers > JDBC_provider > Data Sources > data_source > Connection pool properties**. See the product documentation for details about queuing.
- **Detail:** Currently, the size of the minimum connection pool is 10 and the maximum pool size is 50.

Runtime Performance Advisor tuning advice

Runtime Events

[Runtime Events > Message Details](#)

Use this page to view runtime events that propagate from the server.

General Properties

Message

TUNE0201W: The rate of discards from the prepared statement cache is high. Increase the size of the prepared statement cache for the data source jdbc/TradeDataSource. Additional explanatory data follows. Discards/sec: 3.4. This alert has been issued 1 time(s) in a row. The threshold will be updated to reduce the overhead of the analysis.

Message type

Runtime warning

Explanation

No explanation found for ID=TUNE0201W

User action

No user action found for ID=TUNE0201W

Message Originator

com.ibm.ws.performance.tuning.serverAlert.TraceResponse

Source object type

RasLoggingService

© Copyright IBM Corporation 2016

Figure 8-17. Runtime Performance Advisor tuning advice

WA8152.2

Notes:

This graphic shows an example of a connection pool advice that the Runtime Performance Advisor generates. The message is as follows:

TUNE0201W: The rate of discards from the prepared statement cache is high. Increase the size of the prepared statement cache for the data source jdbc/TradeDataSource. More explanatory data follows. Discards/sec: 3.4. This alert was issued one time in a row. The threshold is updated to reduce the expense of the analysis.

The message type is runtime warning. No explanation or user action is provided for this event.

8.2. Testing, monitoring, and tuning

Testing, monitoring, and tuning



© Copyright IBM Corporation 2016

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

9.1

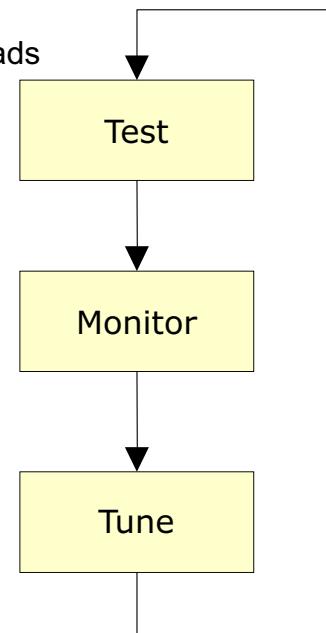
Figure 8-18. Testing, monitoring, and tuning

WA8152.2

Notes:

Tuning the connection pool tasks

- Test application
 - Use a load generation tool to simulate production-like loads
 - Compare results with original baseline
 - Document results
- Monitor connection pool runtime behavior
 - Enable PMI and select wanted statistic set
 - View connection pool performance metrics with Tivoli Performance Viewer
 - Generate tuning advice with Tivoli Performance Viewer Performance Advisor or Runtime Performance Advisor
- Tune connection pool parameters
 - Make one change at a time
 - Apply recommendations and best practices
- Repeat Test-Monitor-Tune cycle until problem is resolved and performance goals are met



© Copyright IBM Corporation 2016

Figure 8-19. Tuning the connection pool tasks

WA8152.2

Notes:

Performance tuning, in general, is an iterative and incremental process that consists of multiple test-monitor-tune cycles. Having the right tools, including a load generation tool to simulate real-world users for load testing, is a must to ensure successful results.

The *art* of performance tuning is a mixture of documentation, test data, and experience. Some tools can assist with this practice such as the *Tivoli Performance Advisor* embedded in WebSphere Application Server, but the suggestions that it offers still need to be verified through load testing. The general method for getting the correct value is to *divide and conquer* by increasing the timeout and connection parameters until the timeout issue disappears and then backing them off until any wasted resources are recovered.

There is also a runtime performance advisor available known as the Performance and Diagnostic Advisor.

The Performance Monitoring Infrastructure (PMI) is enabled by default.

Tuning the connection pool

- Goal is to create a large enough pool that can handle a peak load but does not unnecessarily take up system resources
 - Unused connections during non-peak periods can be controlled with the Minimum connections parameter
- In order to successfully tune the connection pool, you need to know two pieces of information:
 - The requests per second that occur during a peak
 - How long the database takes to respond to each type of operation: SELECT, INSERT, UPDATE, and so on
- Key parameters to tune:
 - Maximum connections: Optimal value for pool size is that which reduces the value of concurrent waiters for a connection (WaitingThreadCount)
 - Connection timeout: Value should be based on a combination of how long the database operations take to complete and the number of concurrent waiters for a connection

© Copyright IBM Corporation 2016

Figure 8-20. Tuning the connection pool

WA8152.2

Notes:

Tuning the connection pool settings for optimal performance during peak load is an iterative activity. It is only through trial and error that the correct parameter values are discovered. In particular, the two parameters that have the greatest effect on correcting connection pool configuration errors are:

- Connection timeout
- Maximum connections

Sometimes the time that is taken to complete a database operation is greater than the amount of time a thread can wait for a resource (connection timeout). In this case, increasing only the number of available connections is not the best approach. Conversely, if the connections are short-lived, then increasing their number might lead to the application server overload in other areas during a peak because the extra connections are unnecessarily consuming resources. Also, the number of idle connections during off-peak periods should be weighed against the pool ramp-up time when a peak occurs. By understanding the nature of these parameters and the nature of the database operations that occur during a peak load, an optimal configuration can be achieved.

Best practices for tuning the connection pool

Maximum connections property:

- As a first guess, try doubling the number of the maximum connections
- If this action solves the problem, then start reducing the number of connections to determine where the failure or bottleneck threshold is
- Better performance is generally achieved if the connections value is set lower than the value for the maximum size of the web container thread pool

Connection timeout setting:

- If a `ConnectionWaitTimeoutException` is found in the WebSphere logs:
 - Obtain the average database operations duration for the application
 - Start with a value that is 5 seconds longer than the average
 - Gradually increase it until problem is resolved or it is at the highest value that the client tolerates

Before you increase the pool size, consult the database administrator to ensure that the database server is configured to handle the maximum pool size setting

© Copyright IBM Corporation 2016

Figure 8-21. Best practices for tuning the connection pool

WA8152.2

Notes:

The database connection pool *Minimum* and *Maximum connections* values are often misunderstood. If you set a maximum of 40 connections and a minimum of 10 connections, the pool does not start with 10 connections created. The value of 10 connections minimum is a low-water mark. Until 10 connections that required concurrently, the pool contains only the maximum number of concurrent connections that are required up to that point. Therefore, if the number of concurrent connections reaches only six, then the pool contains six connections. When the number of connections that are needed exceeds 10, the number of connections in the pool does not drop below 10 until the pool is cleaned out. In other words, after the reap time expires, all unused connections are destroyed until the *Minimum* connections threshold is reached.

Configuration of the data sources should be done in consultation with the database administrator. For instance, the connection pool size should not be larger than the number of agents or connections that are allowed on the database server. This situation can become an issue, especially if cloning is used, because each application server allocates its own pool. To compute the maximum connections that the database might see, multiply the connection pool size by the size of the cluster. For example, assume that the connection pool maximum is 10 (the default), and you have a deployment of two instances of an application server on each of two hosts. There is a potential for up to 40 connections to be open against the database simultaneously.

An application that does significantly more INSERT and UPDATE operations than SELECT operations requires greater resources at the database server. If auto-indexing is activated, then the database might be spending much of its time reindexing after each INSERT or UPDATE. If auto-indexing is turned off, then any SELECT operations might become more expensive because the indexes might become stale. In either case, greater performance cost is incurred for applications that are modifying the data in the database.

Tuning example: Load test

- The load test uses 100 virtual users to browse the catalog on the Plants By WebSphere website and order a plant
- Each user must get a connection to the Derby database
- The test is first run with all default values for the data source connection pool properties
 - Maximum connections = 10
 - Minimum connections = 1
- The Tivoli Performance Viewer is used to monitor the connection pool statistics
- The Rational Performance Tester is used run the test and to view a summary of the test results

© Copyright IBM Corporation 2016

Figure 8-22. Tuning example: Load test

WA8152.2

Notes:

The load test uses 100 virtual users to browse the catalog on the Plants By WebSphere website and order a plant. Each user must have a connection to the Derby database. The test is first run with all default values for the data source connection pool properties.

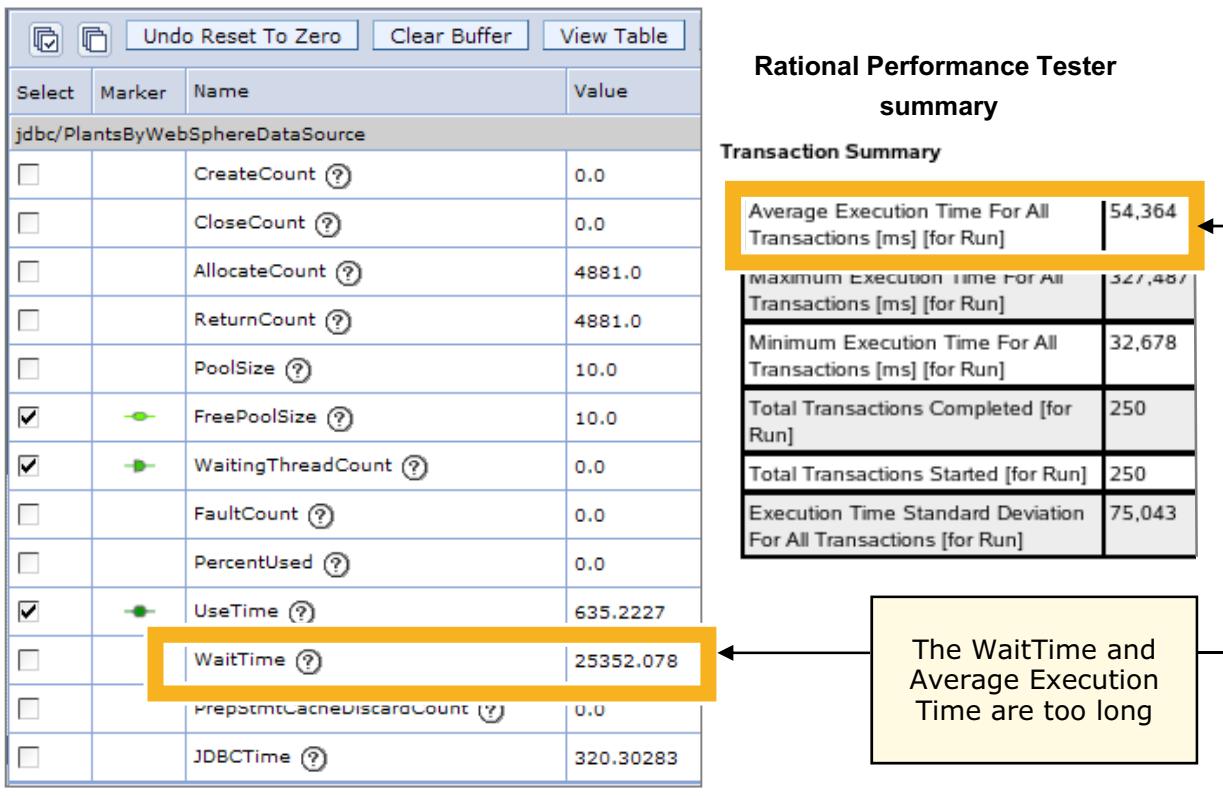
- Maximum connections = 10
- Minimum connections = 1

The Tivoli Performance Viewer is used to monitor the connection pool statistics. The Rational Performance Tester is used to run the test and to view a summary of the test results.

 WebSphere Education



Test results for untuned connection pool



© Copyright IBM Corporation 2016

Figure 8-23. Test results for untuned connection pool

WA8152.2

Notes:

This graphic shows the Tivoli Performance Viewer statistics for the `jdbc/PlantsByWebSphereDataSource` and the Transaction summary from the Rational Performance Tester summary report. The `WaitTime` (the average time to wait until a connection is granted) is 25,352 ms and the `Average Execution Time For All Transactions` is 54,364 ms. The `WaitTime` and `Average Execution Time` are too long.



Tuning example: Tune and load test again

- Advice from Tivoli Performance Viewer Advisor suggests that increasing the maximum connection pool to 30 should increase performance
- The web container thread pool is at the default size of 50; therefore the maximum connections property is set to 30
- The application server is restarted
- The load test is run three times, and the first run results are discarded

© Copyright IBM Corporation 2016

Figure 8-24. Tuning example: Tune and load test again

WA8152.2

Notes:

After restarting the server, the first test run is discarded in accordance with load testing methodology.

Test results for tuned connection pool

Tivoli Performance Viewer statistics

Select	Marker	Name	Value
jdbc/PlantsByWebSphereDataSource			
<input type="checkbox"/>		CreateCount (?)	0.0
<input type="checkbox"/>		CloseCount (?)	0.0
<input type="checkbox"/>		AllocateCount (?)	4883.0
<input type="checkbox"/>		ReturnCount (?)	4883.0
<input type="checkbox"/>		PoolSize (?)	30.0
<input checked="" type="checkbox"/>		FreePoolSize (?)	30.0
<input checked="" type="checkbox"/>		WaitingThreadCount (?)	0.0
<input type="checkbox"/>		FaultCount (?)	0.0
<input type="checkbox"/>		PercentUsed (?)	0.0
<input checked="" type="checkbox"/>		UseTime (?)	1454.7072
<input type="checkbox"/>		WaitTime (?)	11224.816
<input type="checkbox"/>		PrepStmtCacheDiscardCount (?)	0.0
<input type="checkbox"/>		JDBCTime (?)	691.445

Rational Performance Tester summary

Transaction Summary

Average Execution Time For All Transactions [ms] [for Run]	38,977.9
Maximum Execution Time For All Transactions [ms] [for Run]	92,421
Minimum Execution Time For All Transactions [ms] [for Run]	32,746
Total Transactions Completed [for Run]	250
Total Transactions Started [for Run]	250
Execution Time Standard Deviation For All Transactions [for Run]	17,600.8

Using a larger pool size, WaitTime and Average Execution Time are reduced 55% and 28%

© Copyright IBM Corporation 2016

Figure 8-25. Test results for tuned connection pool

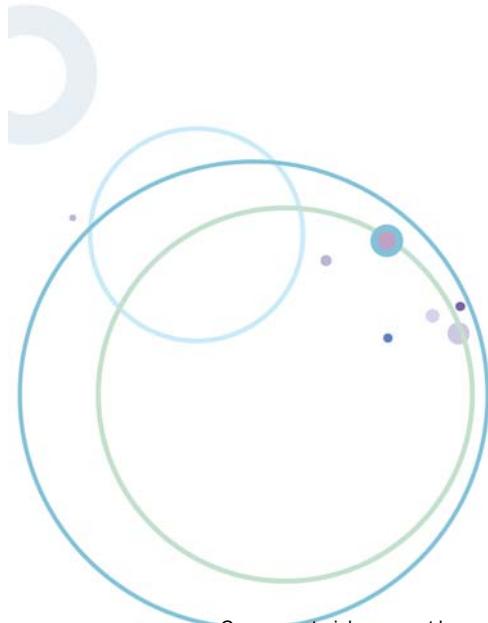
WA8152.2

Notes:

This graphic shows the Tivoli Performance Viewer statistics for jdbc/PlantsByWebSphereDataSource and the Transaction summary form the Rational Performance Tester summary report. The WaitTime (the average time to wait until a connection is granted) is 11,225 ms and the Average Execution Time For All Transactions is 38,978 ms. Using a larger pool size, WaitTime and Average Execution Time are reduced to 55% and 28%.

8.3. Prepared statement cache

Prepared statement cache



© Copyright IBM Corporation 2016

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

9.1

Figure 8-26. Prepared statement cache

WA8152.2

Notes:

Prepared statement cache (1 of 2)

- A prepared statement is a precompiled SQL statement that is stored in a prepared statement object
 - This object is used to efficiently execute the SQL statement multiple times
 - For example, in a loop
- The data source optimizes the processing of prepared statements by caching those statements that are not being used in an active connection
- It is important to configure the cache size of the data source to gain optimal statement execution efficiency

© Copyright IBM Corporation 2016

Figure 8-27. Prepared statement cache (1 of 2)

WA8152.2

Notes:

The data source optimizes the processing of prepared statements to help make SQL statements process faster. It is important to configure the cache size of the data source to gain optimal statement execution efficiency. A prepared statement is a precompiled SQL statement that is stored in a prepared statement object. This object is used to efficiently execute the SQL statement multiple times. If the JDBC driver specified in the data source supports precompilation, the creation of the prepared statement sends the statement to the database for precompilation. Some drivers might not support precompilation, and the prepared statement might not be sent until the prepared statement is executed.



Prepared statement cache (2 of 2)

- If the cache is not large enough, useful entries are discarded to make room for new entries
- In general, the more prepared statements your application has, the larger the cache should be
- For example, if the application has five SQL statements, set the prepared statement cache size to 5 so that each connection has five statements

© Copyright IBM Corporation 2016

Figure 8-28. Prepared statement cache (2 of 2)

WA8152.2

Notes:

If the cache is not large enough, useful entries are discarded to make room for new entries. In general, the more prepared statements your application has, the larger the cache should be. For example, if the application has five SQL statements, set the prepared statement cache size to 5 so that each connection has five statements.



Use Tivoli Performance Viewer to monitor cache discards

- Tivoli Performance Viewer can help tune this setting to minimize cache discards
- Use a standard workload that represents a typical number of incoming client requests, use a fixed number of iterations, and use a standard set of configuration settings
- Watch the PrepStmtCacheDiscardCount counter of the JDBC Connection Pools module
- The optimal value for the statement cache size is the setting that is used to get either a value of zero or the lowest value for PrepStmtCacheDiscardCount

© Copyright IBM Corporation 2016

Figure 8-29. Use Tivoli Performance Viewer to monitor cache discards

WA8152.2

Notes:

Tivoli Performance Viewer can help tune this setting to minimize cache discards. Use a standard workload that represents a typical number of incoming client requests, use a fixed number of iterations, and use a standard set of configuration settings. Watch the PrepStmtCacheDiscardCount counter of the JDBC Connection Pools module. The optimal value for the statement cache size is the setting that is used to get either a value of zero or the lowest value for PrepStmtCacheDiscardCount.



Statement cache size setting

- As with the connection pool size, the statement cache size setting requires resources at the database server
- Specifying too large a cache could have an impact on database server performance
- It is highly recommended that you consult your database administrator for determining the best setting for the prepared statement cache size
- The statement cache size setting defines the maximum number of prepared statements that are cached per connection

A screenshot of a web-based configuration interface for a database connection. The title bar says "Data sources". Below it, the path is "Data sources > PLANTSDB > WebSphere Application Server data source properties". A sub-header says "Use this page to set WebSphere(R) Application Server connection management-specific properties that affect a connection pool." There are tabs for "Configuration" and "General Properties". Under "General Properties", there is a section for "Statement cache size" with a text input field containing the value "10" and the unit "statements".

© Copyright IBM Corporation 2016

Figure 8-30. Statement cache size setting

WA8152.2

Notes:

As with the connection pool size, the statement cache size setting requires resources at the database server. Specifying too large a cache might have an impact on database server performance. It is highly recommended that you consult your database administrator for determining the best setting for the prepared statement cache size. The statement cache size setting defines the maximum number of prepared statements that are cached per connection.

Monitoring DayTrader's jdbc/TradeDataSource (1 of 2)

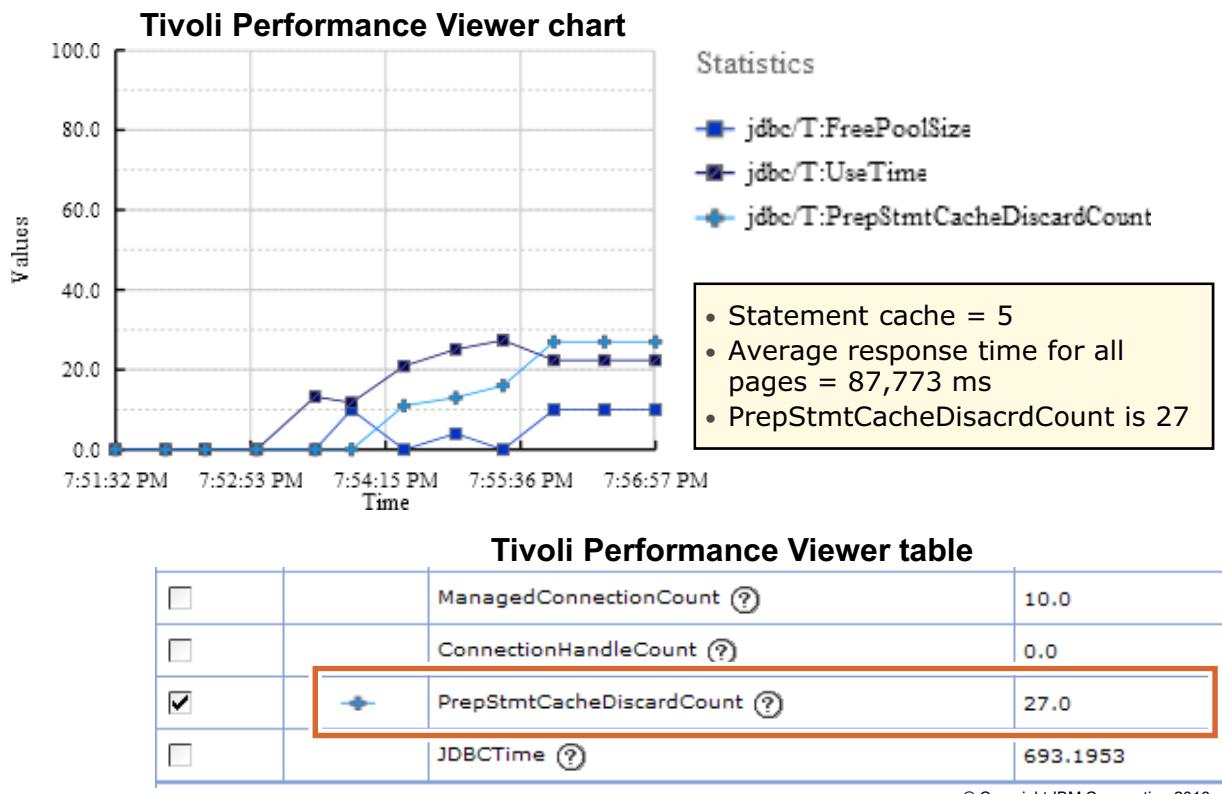


Figure 8-31. Monitoring DayTrader's jdbc/TradeDataSource (1 of 2)

WA8152.2

Notes:

This graphic shows the Tivoli Performance Viewer Chart and Table for the jdbc/TradeDataSource statistics. The statement cache for the application server was set to 5, and the average response time for all pages that Rational Performance Tester reported as 87,773 ms. The PrepStmtCacheDiscardCount is 27. PrepStmtCacheDiscardCount is the number of prepared statements that are discarded because the cache is full.



Monitoring DayTrader's jdbc/TradeDataSource (2 of 2)

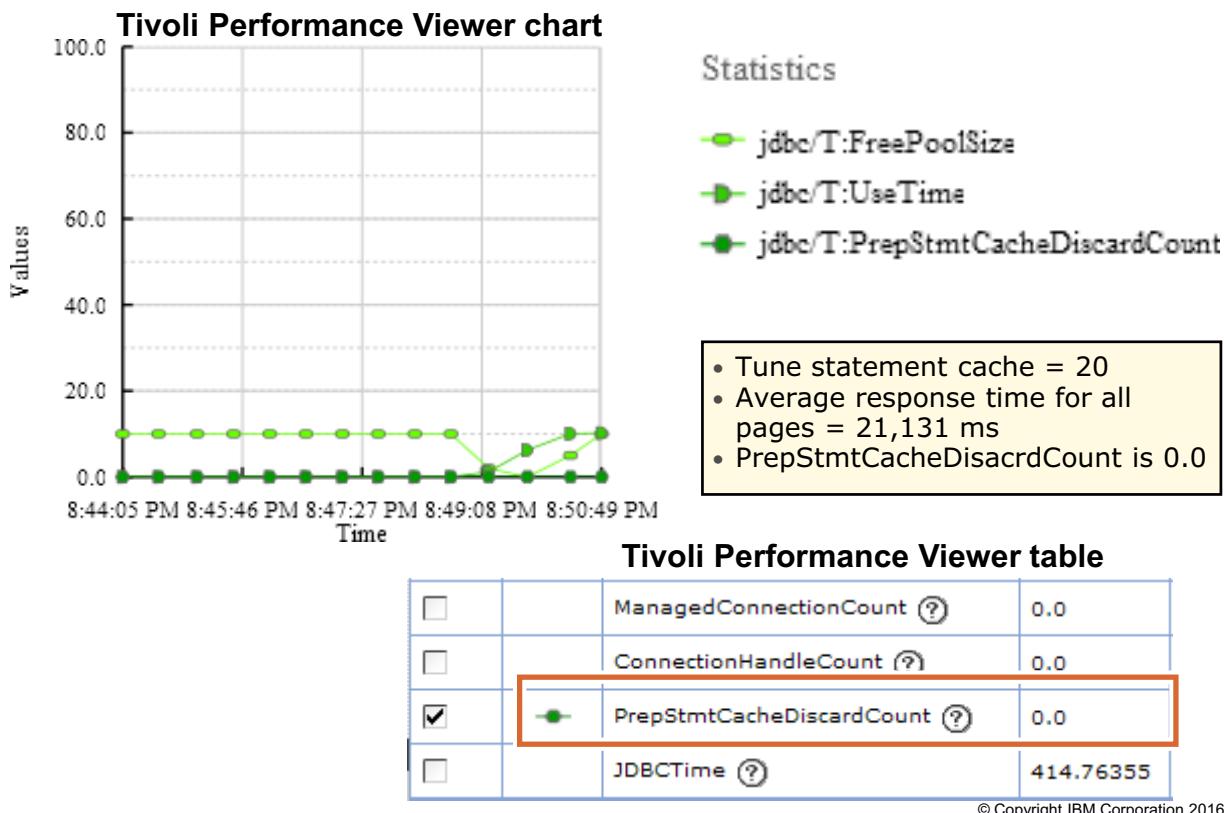


Figure 8-32. Monitoring DayTrader's jdbc/TradeDataSource (2 of 2)

WA8152.2

Notes:

This graphic shows the Tivoli Performance Viewer Chart and Table for the jdbc/TradeDataSource statistics after the statement cache for the application server was set to 20. The average response time for all pages that Rational Performance Tester reported as 21,131 ms, and the PrepStmtCacheDiscardCount is now zero.



Unit summary

Having completed this unit, you should be able to:

- Configure connection pool parameters in the administrative console
- Use Tivoli Performance Viewer to monitor the connection pool
- Use the Tivoli Performance Viewer Performance Advisor to generate tuning advice
- Test, monitor, and tune a connection pool
- Monitor and tune the prepared statement cache

© Copyright IBM Corporation 2016

Figure 8-33. Unit summary

WA8152.2

Notes:

Checkpoint questions

1. True or False: JDBC data sources and JMS connection factories are both managed by the WebSphere J2C connection pool manager.
2. True or False: An improperly tuned connection pool can result in reduced server throughput.
3. True or False: When tuning a connection pool, you must change several related parameters in order to properly monitor any performance gains.
4. True or False: A prepared statement is a JIT compiled Java method that is stored in native memory.

© Copyright IBM Corporation 2016

Figure 8-34. Checkpoint questions

WA8152.2

Notes:

Write your answers here:

- 1.
- 2.
- 3.
- 4.



Checkpoint answers

1. True
2. True
3. False: It is usually best to make one change at a time.
4. False: A prepared statement is a precompiled SQL statement that is stored in a prepared statement object.

© Copyright IBM Corporation 2016

Figure 8-35. Checkpoint answers

WA8152.2

Notes:

Exercise 9



Tuning JDBC connection pools and enabling servlet caching

© Copyright IBM Corporation 2016

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

9.1

Figure 8-36. Exercise 9

WA8152.2

Notes:



Exercise objectives

After completing this exercise, you should be able to:

- Examine and configure connection pool performance parameters by using the administrative console
- Configure the Tivoli Performance Viewer to monitor connection pool performance statistics
- Use Apache JMeter to load test an application
- Monitor an application by using Tivoli Performance Viewer and Apache JMeter
- Tune the connection pool performance parameters after analyzing results from the Tivoli Performance Viewer
- Enable servlet caching for an application server
- Use the Dynamic Cache Monitor to view caching activity and cache contents
- Monitor the performance of the DayTrader application when servlet caching is enabled

© Copyright IBM Corporation 2016

Figure 8-37. Exercise objectives

WA8152.2

Notes:

Unit 9. WebSphere runtime performance tuning

What this unit is about

This unit presents tuning topics covering the end-to-end WebSphere Application Server runtime environment.

What you should be able to do

After completing this unit, you should be able to:

- Identify and tune parameters that are associated with database connectivity, EJBs, and dynamic caching
- Identify and tune parameters that are associated with WebSphere messaging
- Implement performance best practices for WebSphere clusters
- Explain performance considerations associated with using 64-bit WebSphere

How you will check your progress

- Checkpoint questions

References

WebSphere Application Server ND V8.5 Information Center article: Tuning Performance

<http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/index.jsp?topic=%2Fcom.ibm.websphere.nd.doc%2Fae%2Fwelc6toptuning.html>

IBM Redbooks: WebSphere Application Server V6 Scalability and Performance Handbook, SG24-6392-00

Unit objectives

After completing this unit, you should be able to:

- Identify and tune parameters that are associated with database connectivity, EJBs, and dynamic caching
- Identify and tune parameters that are associated with WebSphere messaging
- Implement performance best practices for WebSphere clusters
- Explain performance considerations associated with using 64-bit WebSphere

© Copyright IBM Corporation 2016

Figure 9-1. Unit objectives

WA8152.2

Notes:



Topics

- Operating systems and the Java virtual machine (JVM)
- Threading and concurrency
- Database connectivity, EJBs, dynamic caching
- WebSphere messaging (service integration bus)

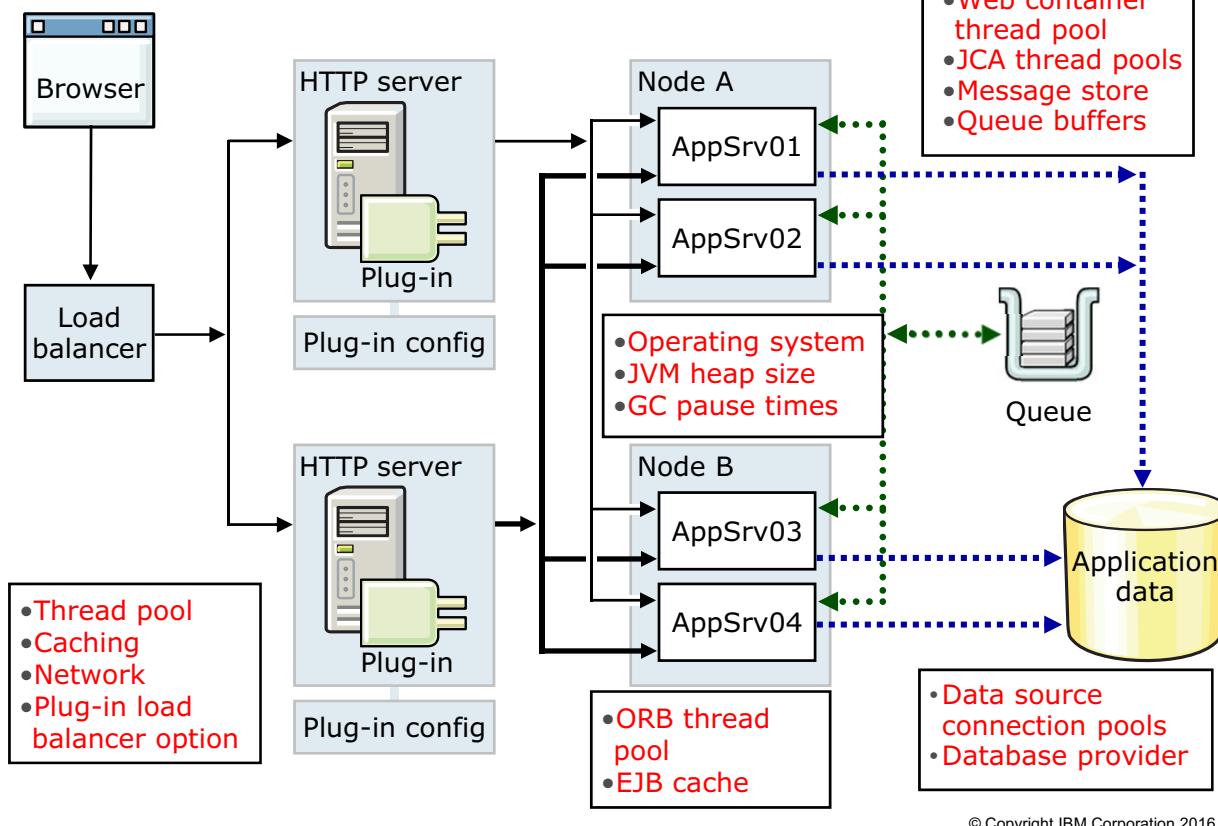
© Copyright IBM Corporation 2016

Figure 9-2. Topics

WA8152.2

Notes:

Request flow and possible bottlenecks



© Copyright IBM Corporation 2016

Figure 9-3. Request flow and possible bottlenecks

WA8152.2

Notes:

LB = load balancer

WC = web container

In a typical Java EE application topology, one or more of the following components can cause performance issue:

- Web server
- Load balancer
- Firewalls
- Hardware
- Network
- Scaling
- Application server JVM
- Application design
- EJBs
- Database
- JMS queues

Tuning parameter hot list

- Review hardware and software requirements
- Install the current refresh pack, fix pack, and the recommended interim fixes
- Verify network interconnections and hardware configuration
- Tune the operating system
- Set the minimum and maximum Java virtual machine (JVM) heap sizes
- Set the garbage collection level appropriately
- Use type 4 (pure Java) JDBC driver
- Tune WebSphere Application Server data sources and connection pools
- Enable the pass by reference option
- Ensure that the transaction log is assigned to a fast disk
- Disable functions that are not required
- Review the application design

© Copyright IBM Corporation 2016

Figure 9-4. Tuning parameter hot list

WA8152.2

Notes:

The WebSphere Application Server ND V8.5 product documentation contains a list of hot tuning parameters with details about each:

http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/index.jsp?topic=%2Fcom.ibm.websphere.nd.doc%2Fae%2Frprf_hotparameters.html

Many of these topics were discussed in previous units. A few more are covered in this presentation.

9.1. Operating systems and the Java virtual machine (JVM)

Operating systems and the Java virtual machine (JVM)



© Copyright IBM Corporation 2016

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

9.1

Figure 9-5. Operating systems and the Java virtual machine (JVM)

WA8152.2

Notes:

Operating systems (1 of 2)

- Refer to documentation for the operating system that hosts WebSphere for details about recommended tuning approaches
- Tuning for the following operating systems is covered in the WebSphere Application Server V8.5 Information Center
 - Windows, Linux, AIX, Solaris, HP-UX
- Refer to the relevant information center article for tuning details
- Large page support
 - Operating systems can provide a large contiguous section of memory by using memory pages that are larger than the default memory page size
 - You must define and enable large memory pages within the OS
 - Depending on the OS, large memory page sizes can range from 4 MB (Windows) to 16 MB (AIX) versus the default page size of 4 KB
 - Many Java based applications often benefit from large pages due to a reduction in CPU cost that is associated with managing smaller numbers of large pages

© Copyright IBM Corporation 2016

Figure 9-6. Operating systems (1 of 2)

WA8152.2

Notes:

Refer to documentation about the operating system that hosts WebSphere for details about recommended tuning approaches. For more information about tuning the operating system, see the following website:

http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/index.jsp?topic=%2Fcom.ibm.websphere.nd.doc%2Fae%2Ftprf_tuneopsys.html

Operating systems (2 of 2)

Example settings and variables for Linux

- TCP/IP timeouts:
 - timeout_timewait parameter
 - TCP_KEEPALIVE_INTERVAL
 - TCP_KEEPALIVE_PROBES
- Linux file descriptors (ulimit)
 - Specifies the number of open files that are supported
 - Default setting is sufficient for most applications
 - If the ulimit is too low, a file open error, memory allocation failure, or connection establishment error might be displayed
- Connection backlog: Change the following parameters when a high rate of incoming connection requests result in connection failures:
 - echo 3000 > /proc/sys/net/core/netdev_max_backlog
 - echo 3000 > /proc/sys/net/core/somaxconn

© Copyright IBM Corporation 2016

Figure 9-7. Operating systems (2 of 2)

WA8152.2

Notes:

- **timeout_timewait parameter:** Determines the time that must elapse before TCP/IP can release a closed connection and reuse its resources. This interval between closure and release is known as the **TIME_WAIT** state or twice the maximum segment lifetime (2MSL) state.
- **TCP_KEEPALIVE_INTERVAL:** Determines the wait time between isAlive interval probes.
- **TCP_KEEPALIVE_PROBES:** Determines the number of probes before timing out.
- **Linux file descriptor (ulimit):** Specifies the number of open files that are supported. The default setting is typically sufficient for most applications. If the value set for this parameter is too low, a file open error, memory allocation failure, or connection establishment error might be displayed.

64-bit WebSphere Application Server (1 of 2)

- As of V7, a 64-bit version of WebSphere Application Server is available on almost all supported hardware and operating system platforms
- Allows the JVM to grow well beyond 32-bit process size boundaries
- Applications that experience greatest benefit are either memory-constrained or computationally expensive
 - Memory constrained
 - Extra memory supports better caching strategy
 - Avoids expensive queries
 - Computationally expensive code
 - Security algorithms
- Might be required for cases that need a large heap or many threads

© Copyright IBM Corporation 2016

Figure 9-8. 64-bit WebSphere Application Server (1 of 2)

WA8152.2

Notes:

A 32-bit JVM can access 4 GB of memory like any 32-bit process. Many operating systems reserve 1.5 GB of the 4 GB address space of a 32-bit process, leaving 2.5 GB for the application. The JVM is the application. A JVM needs memory outside the heap, like for a stack per thread. When the JVM has more threads, more memory is required outside the heap. When the Java heap is set to a maximum of 1.8 GB, approximately 700 MB is left outside the heap. A JVM runs optimally at 50% live occupancy. The size of live memory for a 1.8 GB heap is 900 MB.

Applications that run within Java based servers have “size of live” objects that approach or exceed 900 MB. Because the 64-bit JVM is supported on most hardware and operating system platforms for WebSphere Application Server V7, the 64-bit JVM is an option in almost all cases for almost all applications. The 64-bit JVM should be considered when an application can benefit from a JVM heap larger than 1.8 GB. These applications include ones that would benefit from a large cache or that have a “size of live” objects that are more than 900 MB. The 64-bit JVM should also be considered when an application needs a high number of threads, which reduces the maximum heap size that should be run for the 32-bit JVM.

64-bit WebSphere Application Server (2 of 2)

- Stay with 32-bit WebSphere if the hosted application operates efficiently in a heap that is 1.8 GB (1800 MB) or smaller
 - JVMs automatically size the heap to operate at 50% live size occupancy (within the maximum heap size parameter)
 - 50% live size occupancy is a good tradeoff between time in garbage collection and time between garbage collections
 - A JVM sizes the heap at 1.8 GB automatically if the “size of live” is 900 MB
 - Collect and graph verbose GC to determine the “size of live” of an application that runs under expected and peak loads
 - Stay with the 32-bit JVM if the “size of live” is 900 MB or less
- Consider moving to the 64-bit JVM if “size of live” is larger than 900 MB and if more caching improves the performance of the application
- Note: “size of live” refers to the number of live objects in the heap

© Copyright IBM Corporation 2016

Figure 9-9. 64-bit WebSphere Application Server (2 of 2)

WA8152.2

Notes:

Although the 64-bit JVM enables heap sizes beyond 1.8 GB, the 32-bit JVM is more efficient for applications that do not need a heap larger than 1.8 GB.

The “size of live” objects for an application should be monitored and measured by collecting verbose GC output from a JVM. If the “size of live” objects for an application is less than 900 MB, the 32-bit JVM is more efficient than the 64-bit JVM for that application (assuming relatively normal memory usage outside the heap).



Moving transaction logs and file store to a fast disk

- Since disk I/O operations are costly, storing log files on fast disks such as a RAID can greatly improve performance
- The Transaction log directory can be set in the administrative console at **Servers > Application Servers > server_name > Container Services > Transaction Service**
- The File store log directory can be specified during the creation of a SIBus member by using either:
 - The `-logDirectory` option in the `AdminTask addSIBusMember` command
 - The administration console SIBus member creation panels

© Copyright IBM Corporation 2016

Figure 9-10. Moving transaction logs and file store to a fast disk

WA8152.2

Notes:

In most RAID configurations, the task of writing data to the physical media is shared across multiple drives. This technique yields more concurrent access to storage for persisting transaction information, and faster access to that data from the logs.

Key JVM tuning parameters

- Heap size
 - Heap size parameters influence the behavior of garbage collection
 - Increasing the heap size supports more object creation
 - Since a large heap takes longer to fill, the application runs longer before a garbage collection occurs (increases time between garbage collections)
 - A larger heap also takes longer to compact and causes garbage collection to take longer (increases garbage collection pause times)
- Garbage collection policy
 - Garbage collection typically consumes 5 - 20 percent of total run time of a properly functioning application
 - If not managed, garbage collection is one of the biggest bottlenecks for an application
 - Test and tune available policies: gencon, optthruput, optavgpause, balanced

© Copyright IBM Corporation 2016

Figure 9-11. Key JVM tuning parameters

WA8152.2

Notes:

In general, the topic of tuning Java virtual machines warrants its own dedicated treatment. However, this graphic contains a high-level description of the most important JVM parameters.

One of the most important JVM parameters is maximum heap size. In general, the size of the Java heap should be tuned so that the JVM is able to operate at 50% live occupancy. Live occupancy can be defined as the current “size of live objects” divided by the current size of the heap. For example, if the current “size of live objects” is 500 MB and the current size of the heap is 1 GB (1000 MB), live occupancy is 50%.

Another key JVM parameter for the IBM JVM is “`gc policy`”. The IBM JVM has three GC policies:

- “`optthruput`” where the heap is treated as one large block of memory.
- “`optavgpause`” where portions of the garbage collection process are performed in parallel with Java execution.
- “`gencon`” where the heap is divided into nursery and tenured generations.

In general, the “`gencon`” GC policy achieves the highest levels of throughput and the shortest garbage collection pause times.

There are two high-level guidelines for tuning the IBM JVM when running the “`gencon`” garbage collection policy:

- The size of the nursery should be 1/4 to 1/2 the overall size of the heap.
- The size of the tenured generation should be large enough to hold application caching.

9.2. Threading and concurrency

Threading and concurrency



© Copyright IBM Corporation 2016

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

9.1

Figure 9-12. Threading and concurrency

WA8152.2

Notes:

Threading and concurrency: Web server (1 of 3)

A single Java EE application frequently supports thousands of concurrent clients

- Consider the use of multiple web server instances
 - An application needs to support more concurrent clients than the number of threads that the host OS supports

The web server might refuse a session if a sudden spike of incoming requests occurs.

- Increasing the `ListenBacklog` and `StartServers` parameters might reduce or eliminate this behavior
 - The `ListenBacklog` parameter configures the maximum number of pending connections
 - The `StartServers` parameter configures number of IBM HTTP Server processes to start initially

Avoid frequent creation and destruction of client threads or processes as the number of users changes

- Configure the parameters:
 - `MinSpareServers`
 - `MaxSpareServers`

© Copyright IBM Corporation 2016

Figure 9-13. Threading and concurrency: Web server (1 of 3)

WA8152.2

Notes:

All of the web servers that are typically used with WebSphere allocate a thread to handle each client connection. Ensuring that enough threads are available for the maximum number of concurrent client connections helps prevent the web tier from being an application bottleneck. Even if the web server is tuned properly, the web server might refuse a session if a sudden spike of incoming requests occurs. Increasing the “`ListenBacklog`” and “`StartServer`” parameters might reduce or eliminate this behavior.

The “`MinSpareServers`” and “`MaxSpareServers`” can be used to specify the minimum and maximum number of servers (client threads and processes) that can exist in an idle state. Frequent creation and destruction of client threads and processes can result from too wide variations in the number of incoming users. To control this situation, set the range of these parameters large enough to include the maximum number of simultaneous users.

Because logging represents a performance cost, minimize logging in to the web server’s access log to maximize application operational efficiency.

The main goal of load-balancing algorithms is to provide an even distribution of work across cluster members. The default “round-robin” load balancing algorithm works best with web servers that have a single process that sends requests to the application server. If the web server is using

multiple processes to send requests to the application server, the “random” load balancing algorithm can sometimes yield a more even distribution of work across a cluster.

The “**RetryInterval**” parameter specifies the length of time a web server waits before trying to connect to an application server that has previously been marked as temporarily unavailable. The default value of this parameter is 60 seconds, but you might want to reduce this value so that the web server uses a recovered application server as soon as possible.

Threading and concurrency: Web server (2 of 3)

- Minimize logging to the web server's access log
- Consider the use of random plug-in load-balancing instead of weighted round-robin for WebSphere clusters
 - Might provide more even distribution across the cluster when multiple processes are sending requests
- Consider reducing the plug-in retry interval to application servers
 - Retry interval value specifies the length of time to wait before trying to connect to a server that is marked temporarily unavailable
 - If the interval is too high, an available server might not be used
- Concurrency tuning at web servers depends on the expected peak number of simultaneous TCP connections
 - The number of requests currently being processed is the number of simultaneous connections
 - Monitor `mod_status` reports to determine the maximum number of connections that must be handled
 - Configure via directives in the `httpd.conf` file

© Copyright IBM Corporation 2016

Figure 9-14. Threading and concurrency: Web server (2 of 3)

WA8152.2

Notes:

Consider the use of **random** load-balancing instead of **weighted round-robin** for WebSphere clusters. The goal of the default load balance option, round-robin, is to provide an even distribution of work across cluster members. Round-robin works best with web servers that have a single process that sends requests to the application server. If the web server is using multiple processes to send requests to the application server, the random option can sometimes yield a more even distribution of work across the cluster.

Retry interval specifies the length of time, in seconds, that should elapse from the time an application server is marked down to the time that the plug-in tries to connect again.

Consider reducing the retry interval to application servers. How can lowering the retry interval affect throughput? If the plug-in attempts to connect to a particular application server and that application server is offline or while restarting, the requests must wait for a timeout period. This process causes delayed responses for those requests. If you set the retry interval value too high, then an available application server is not used.

Threading and concurrency: Web server (3 of 3)

- Web server tuning recommendations
 - ThreadsPerChild: Default (25 on many operating systems)
 - MaxClients: Maximum number of simultaneous connections rounded up to an even multiple of ThreadsPerChild
 - StartServers: 2
 - MinSpareThreads: The greater of 25 or 10% of MaxClients
 - MaxSpareThreads: Equal to MaxClients in cases where the level of web server resource requirements is not a concern when the web server is idle (high performance configurations)
 - ServerLimit: MaxClients divided by ThreadsPerChild
 - ThreadLimit: Equal to ThreadsPerChild
- Refer to your web server documentation for more details

© Copyright IBM Corporation 2016

Figure 9-15. Threading and concurrency: Web server (3 of 3)

WA8152.2

Notes:

This slide shows some suggested values of tuning parameters specific to web servers. The listed parameters might not apply to all of the supported web servers. Check your web server documentation before using any of these parameters.

Threading and concurrency: Application server

- Often an HTTP request arrives at the application server and is dispatched to a web container thread pool where it remains during its entire processing
- Thus, the size of the web container thread pool is a critical tuning parameter
 - Typically 50 - 75 web container threads are normal for efficient dispatching of work to CPUs
- General application server thread pools
 - Web container thread pool (HTTP requests)
 - ORB thread pool (RMI/IOP requests to EJBs)
 - Work manager thread pools (asynchronous beans)
- Messaging thread pools
 - Message listener thread pool
 - Default thread pool
- J2C and JCA thread pools
 - Default thread pool
 - User-defined thread pools

© Copyright IBM Corporation 2016

Figure 9-16. Threading and concurrency: Application server

WA8152.2

Notes:

Tuning the thread pool is critical to achieving the required level of throughput. It is also required to have enough threads to process enough simultaneous requests to maximize usage of the provided CPU capacity if required.

To tune the thread pool, consider the threading model of the deployed applications. Commonly for Java EE applications, the threading model is simple where an incoming request arrives as an HTTP request, which is dispatched to a thread from the web container thread pool where all processing is performed. In some cases, one application server calls an EJB with RMI on another application server. In these cases, the target, downstream application server processes the RMI request on a thread from the ORB thread pool. If applications use asynchronous beans to execute portions of request processing, the asynchronous beans execute on threads from work manager thread pools.

In all cases, tuning the thread pool size should be based on the expected number of simultaneous requests and the number of physical (and logical) CPUs hosting the application server.

Typically, 50 - 70 web container threads are sufficient to consume all of the CPUs provided in a “logical partition”.



Threading and concurrency: Server thread pools

You can administer the following resources:

Select	Name	Description	Minimum Size	Maximum Size
<input type="checkbox"/>	Default		20	20
<input type="checkbox"/>	ORB.thread.pool		10	50
<input type="checkbox"/>	SIBFAPInboundThreadPool	Service integration bus FAP inbound channel thread pool	4	50
<input type="checkbox"/>	SIBFAPThreadPool	Service integration outbound channel		
<input type="checkbox"/>	SIBJMSRAThreadPool	Service Integration Resource Adapter t		
<input type="checkbox"/>	TCPChannel.DCS			
<input type="checkbox"/>	WMQCommonServices	WebSphere MQ common services thread pool		
<input type="checkbox"/>	WMQJCAResourceAdapter	wmqJcaRaThreadP		
<input type="checkbox"/>	WebContainer			
<input type="checkbox"/>	server.startup	This pool is used b during server startu		
Total 10				

Configuration

Application servers > server_name > Thread pools

General Properties

* Name: Custom properties

Description:

* Minimum Size: threads

* Maximum Size: threads

* Thread inactivity timeout: milliseconds

Allow thread allocation beyond maximum thread size

© Copyright IBM Corporation 2016

Figure 9-17. Threading and concurrency: Server thread pools

WA8152.2

Notes:

Three of the most commonly used (and tuned) thread pools within the application server are Default, Orb, and WebContainer. The graphic shows the default values for the Minimum and Maximum Size.

- **WebContainer thread pool:** Used when requests come in over HTTP. From the DayTrader architecture above, you can see that most traffic comes into DayTrader through the WebContainer thread pool.
- **Default thread pool:** Used when requests come in for a message-driven bean or if a particular transport chain is defined to a specific thread pool.
- **ORB thread pool:** Used when remote requests come in over RMI/IOP for an enterprise bean from an EJB application client, remote EJB interface, or another application server.
- **Minimum Size:** The minimum number of threads to allow in the pool. When an application server starts, no threads are initially assigned to the thread pool. Threads are added to the thread pool when the workload assigned to the application server requires more threads. Threads are added until the number of threads in the pool equals the number that is specified in the Minimum Size field. After this point in time, more threads are added and removed when the

workload changes. However, the number of threads in the pool never decreases below the number that is specified in the Minimum Size field, even if some of the threads are idle.

- **Maximum Size:** Specifies the maximum number of threads to maintain in the default thread pool.
- **Thread inactivity timeout:** Specifies the number of milliseconds of inactivity that should elapse before a thread is reclaimed. A value of 0 indicates not to wait, and a negative value (less than 0) means to wait forever.

Message listeners and activation specifications (1 of 2)

- Message-driven beans (MDBs) accessing WebSphere MQ queues are assigned listeners and connection factories
 - Listeners and connection factories should be tuned for the concurrency that is required to achieve application throughput requirements
 - Tuning parameter: Connection factory's connection pool parameters
- MDBs based on JCA are assigned activation specifications
 - Activation specifications should be tuned for the concurrency that is required to achieve application throughput requirements
 - Tuning parameter: Maximum concurrent MDB invocations per endpoint (default 10)

© Copyright IBM Corporation 2016

Figure 9-18. Message listeners and activation specifications (1 of 2)

WA8152.2

Notes:

MDBs based on WebSphere MQ queues are assigned “listeners”. MDBs based on J2C or JCA are assigned “activation specifications”. The number of threads that are used by an MDB depends on the “listeners” for WebSphere MQ queues and “activation specifications” for JCA or J2C-based queues.

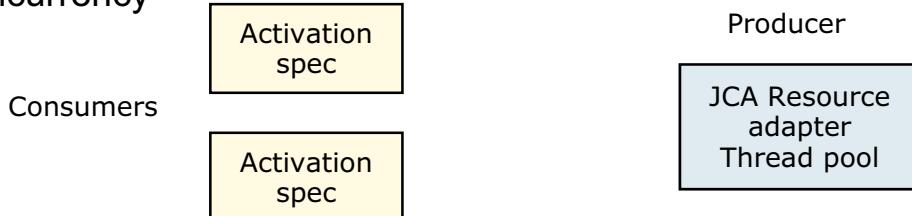
All thread pool consumers should be considered when configuring the maximum size of a “providing” thread pool.

The maximum concurrent MDB invocations per endpoint are the maximum number of endpoints to which messages are delivered concurrently.

Increasing this number can improve performance but can increase the number of threads that are in use at any one time. If message order must be retained across failed deliveries, set the maximum concurrent endpoints to 1. Message order applies only if the destination that the message-driven bean is consuming from is not a partitioned destination. Partitioned destinations are used in a workload-sharing scenario in a cluster.

Message listeners and activation specifications (2 of 2)

- Listeners and activation specifications configure the consumer side of concurrency



- The provider side of concurrency is the thread pool
- The provider side is shared among the set of listeners and activation specifications
- The concurrency configuration of all listeners or activation specs needs to be considered when tuning the thread pool provider
 - Consider all thread pool consumers in aggregate when tuning the providing thread pool

© Copyright IBM Corporation 2016

Figure 9-19. Message listeners and activation specifications (2 of 2)

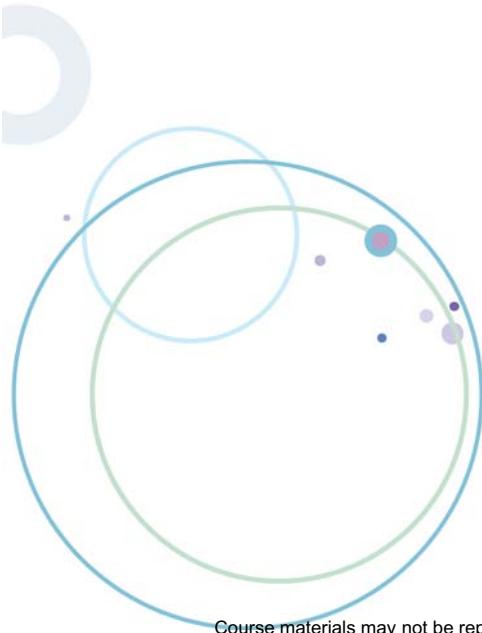
WA8152.2

Notes:

Listeners and activation specifications configure the consumer side of MDB concurrency. In other words, the number of threads that are allocated to a listener or activation spec is the maximum number of threads the associated MDB uses from a thread pool that provides the threads. For listeners, the thread pool that provides the threads is the Message Listener thread pool. For activation specifications, the thread pool that provides the threads is the thread pool that is assigned to the corresponding JCA or J2C resource adapter. By default, resource adapters in WebSphere are configured to use the Default thread pool.

9.3. Database connectivity, EJBs, and dynamic caching

Database connectivity, EJBs, and dynamic caching



© Copyright IBM Corporation 2016

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

9.1

Figure 9-20. Database connectivity, EJBs, and dynamic caching

WA8152.2

Notes:

Database connectivity (1 of 2)

- One of the most important parameters to tune in the area of database connectivity is database connections
- There is a producer-consumer relationship for database connections:
 - WebSphere data sources are the consumer
 - Database providers are the producer of database connections



- The number of database connections that are offered at the database provider must be sufficient to handle the maximum number of database connections that are requested from JDBC data sources

© Copyright IBM Corporation 2016

Figure 9-21. Database connectivity (1 of 2)

WA8152.2

Notes:

Database connections take a significant amount of time to create. Database connection pooling enables applications to avoid the cost of creating new database connections by using an existing database connection from a pool.

There is a producer-consumer relationship with database connections. The consumer side of database connections is the data source at the application server. The producer side of database connections is at the database provider. In general, the number of database connections that are requested at the consumer side should not be more than the number of database connections that are provided at the producer side.

The maximum connections parameter for a JDBC data source specifies the maximum number of database connections in a database connection pool.

A general rule for tuning database connections is to make at least one connection for each type of thread that uses a database connection. Avoid creating more threads that require database connections than there are available database connections.



Database connectivity (2 of 2)

Data sources

[Data sources > TradeDataSource > Connection pools](#)

Use this page to set properties that impact the timing of connections which can affect the performance of your application. Consider your application requirements might warrant changing these settings.

Configuration

General Properties

Scope: cells:was7host01Cell01:nodes:was7host01Node01

- * **Connection timeout**: 180 seconds
- * **Maximum connections**: 30 connections
- * **Minimum connections**: 10 connections
- * **Reap time**: 180 seconds
- * **Unused timeout**: 1800 seconds
- * **Aged timeout**: 0 seconds

Purge policy: EntirePool

- Connection pool configuration
- Most important tuning parameters
 - Maximum connections
 - Minimum connections
- General rule for tuning database connections:
 - One database connection per thread that requires a database connection
 - The number of database connections should be close to the maximum number of threads that can run on the application server

© Copyright IBM Corporation 2016

Figure 9-22. Database connectivity (2 of 2)

WA8152.2

Notes:

- **Connection timeout**: Specifies the interval, in seconds, after which a connection request times out and a Connection Wait Timeout Exception is thrown.
- **Maximum connections**: Specifies the maximum number of physical connections that you can create in this pool.
- **Minimum connections**: Specifies the minimum number of physical connections to maintain.
- **Reap time**: Specifies the interval, in seconds, between runs of the pool maintenance thread.
- **Aged timeout**: Specifies the interval in seconds before a physical connection is discarded.
- **Purge policy**: Specifies how to purge connections when a **stale connection** or **fatal** connection error is detected. Valid values are **EntirePool** and **FailingConnectionOnly**.

Database servers

- General tuning recommendations for database providers
 - Keep database statistics up-to-date using the tools that are part of the database product
 - Place database log files on high-speed disk subsystems
 - Place logs on a separate disk device from table space containers
 - Maintain table indexing that the tools that are a part of the database product recommends
 - Size log files appropriately for the level of load against the databases
 - Refer to performance documentation for the database product that is used for a comprehensive set of tuning recommendations
- Consider the use of DB2 Performance Expert Extended Insight (PEEI) for DB2 databases
 - Provides end-to-end monitoring for your WebSphere to DB2 environment

© Copyright IBM Corporation 2016

Figure 9-23. Database servers

WA8152.2

Notes:

Databases often have a wide variety of available choices when determining the best approach for accessing data. Statistics, which describe the “shape” of the data, are used to guide the selection of an efficient data access strategy. Examples of statistics include the number of rows in a table and the number of distinct values in a certain column. A set of representative statistics allows good performance over a long span of time. However, it might be necessary to refresh statistics periodically if the data population shifts dramatically.

Changes to table data might not be written immediately to disk and instead might be made to database log files when log buffers fill at transaction commit time, and possibly after a maximum interval of time. More importantly, log writes hold commit operations that are pending, meaning that the application is synchronously waiting for the write to complete. Thus, it is important to place database log files on fast disk subsystems that employ a write-back cache.

A basic strategy for database storage configurations is to place the database logs on dedicated physical disks separate from the table space containers. This configuration reduces contention between disk access I/O between the table space containers and the database logs.

Database providers such as IBM DB2 and Oracle provide tools that provide index recommendations in the context of a specific workload. The tools should be applied to representative workloads, and if properly used, their recommendations should be applied.

Increasing the available log space gives asynchronous cleaning of pages more time to write dirty buffer pool pages and avoid log switch pauses. A longer interval between cleanings allows multiple changes to be coalesced on a page before it is written, which reduces the required write throughput by making cleaning of pages more efficient.

Consider the use of DB2 Performance Expert Extended Insight (PEEI) for DB2 databases:

- <http://www.ibm.com/developerworks/data/tutorials/dm-0906db2expertinsight1/>

For more information about IBM DB2, see the following website:

- <http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/index.jsp>

For more information about Oracle, see the following websites:

- <http://www.oracle.com/pls/db111/homepage>
- <http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP100883>

Enterprise JavaBeans (EJBs): Cache tuning

- Cache size = Maximum concurrent active instances
 - (Entity beans that are required per transaction) \times (maximum expected concurrent transaction)
 - Add maximum active stateless session bean instances
 - Tuning parameters: cache size and cleanup interval
- Use large EJB caches if you have sufficient heap
 - Benchmark caches routinely sized at 30 K entries or more
 - Monitor to find the optimal size
- If the cache fills up, the container tries to passivate based on LRU (least recently used)
 - Passivation triggers disk I/O: Slow
 - Tivoli Performance Viewer shows a very high rate of the `ejbStores` method calls
- Remember that some EJBs can be long lived
 - Stateful session beans
- Use Tivoli Performance Viewer to monitor EJBs

© Copyright IBM Corporation 2016

Figure 9-24. Enterprise JavaBeans (EJBs): Cache tuning

WA8152.2

Notes:

WebSphere Application Server includes a caching facility that caches EJB instances. Significant performance improvements can be realized by leveraging this caching capability. However, a balance needs to be struck between the performance improvements from caching and the memory cost for an application.

There is a diagnostic trace service for the EJB cache that helps determine the optimal size for the EJB cache.

The cache size specifies the number of buckets in the active instance list within the EJB container.

A bucket can contain more than one active enterprise bean instance, but performance is maximized if each bucket in the table has a minimum number of instances that are assigned to it. When the number of active instances within the container exceeds the number of buckets, that is, the cache size, the container periodically attempts to reduce the number of active instances in the table by passivating some of the active instances. For the best balance of performance and memory, set this value to the maximum number of active instances that are expected during a typical workload.

The cleanup interval specifies the interval at which the container attempts to remove unused items from the cache to reduce the total number of items to the value of the cache size. This setting applies to the cache only.

ORB pass by reference (1 of 2)

- The Object Request Broker (ORB) Pass by reference option determines whether pass by reference or pass by value semantics should be used when handling parameter objects involved in an EJB request
- Pass by reference option can be found in the administrative console at **Application Servers > server_name > Container Services > ORB service**
 - Enable this property with caution because unexpected behavior can occur
- By default this option is disabled and a copy of each parameter object is made and passed to the invoked EJB method
- This action is considerably more expensive than passing a simple reference to the existing parameter object

© Copyright IBM Corporation 2016

Figure 9-25. ORB pass by reference (1 of 2)

WA8152.2

Notes:

The ORB Pass by reference option basically treats the invoked EJB method as a local call (even for EJBs with remote interfaces) and avoids the requisite object copy. If remote interfaces are not necessary, a slightly simpler alternative, which does not require tuning, is to use EJBs with local interfaces.

Enable this property with caution because unexpected behavior can occur. If the callee modifies an object reference, the caller's object is modified as well, since they are the same object.

Usually, any application code that passes an object reference as a parameter to an enterprise bean method must be scrutinized to determine whether passing that object reference results in loss of data integrity or in other problems.

ORB pass by reference (2 of 2)

- This option provides a benefit only when the EJB client (servlet) and invoked EJB are located within the same Classloader
 - This requirement means that both the EJB client and EJB must be deployed in the same EAR file and running on the same application server instance
 - If the EJB client and EJB modules are mapped to different application server instances (often referred to as split-tier), the EJBs must be invoked remotely using by pass by value semantics
- The DayTrader application contains both the web and EJB modules in the same EAR file and both are deployed to the same application server instance
 - Therefore, the ORB Pass by reference option can be used to realize a performance gain

© Copyright IBM Corporation 2016

Figure 9-26. ORB pass by reference (2 of 2)

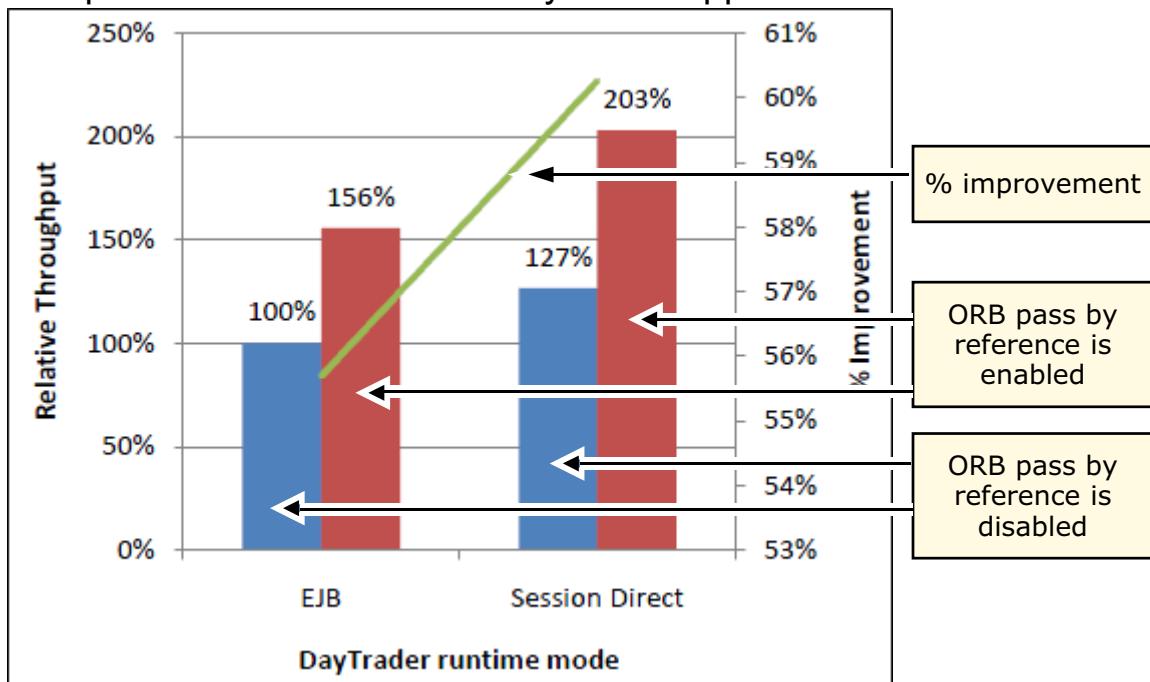
WA8152.2

Notes:

The DayTrader application contains both the web and EJB modules in the same EAR file and both are deployed to the same application server instance. Therefore, the ORB Pass by reference option can be used to realize a performance gain.

Performance benefit of ORB pass by reference

- This option is beneficial for the DayTrader application



© Copyright IBM Corporation 2016

Figure 9-27. Performance benefit of ORB pass by reference

WA8152.2

Notes:

As indicated by the measurements that are shown on this slide, this option is beneficial for DayTrader. In EJB mode, all requests from the servlets are passed to the stateless session beans over a remote interface. In direct mode, the EJB container is bypassed in lieu of direct JDBC and manual transactions.



Dynamic caching (1 of 3)

- To configure the dynamic cache service, click **Servers > Server Types > WebSphere application servers > server_name > Container services > Dynamic cache service**
- The dynamic cache service improves performance by caching the output of:
 - Servlets
 - Commands (Java object)
 - JavaServer Pages (JSP)
- Dynamic caching features include:
 - Cache replication among clusters
 - Cache disk offload
 - Edge-side include caching
 - External caching, which is the ability to control caches outside of the application server such as the web server

© Copyright IBM Corporation 2016

Figure 9-28. Dynamic caching (1 of 3)

WA8152.2

Notes:

Caching the output of servlets, commands, and JavaServer Pages (JSP) improve application performance. WebSphere Application Server consolidates several caching activities, including servlets, web services, and WebSphere commands, into one service that is called the *dynamic cache*. These caching activities work together to improve application performance, and share many configuration parameters that are set in the dynamic cache service of an application server. You can use the dynamic cache to improve the performance of servlet and JSP files by serving requests from an in-memory cache. Cache entries contain servlet output, the results of a servlet after it runs, and metadata.

The “commands” refer to a bean command object.

For more information about the command object, see “Caching a command object”:
http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/index.jsp?topic=/com.ibm.websphere.nd.multiplatform.doc/ae/rdyn_commandxmp.html

Dynamic caching (2 of 3)

- Consider dynamic caching during application design
 - Should be part of design rather than retro-fitting later
- Organize page layout and design for caching
 - Servlet and JSP caching to hold repetitive, expensive page elements
 - Stateful elements versus general use elements
- API caching via distributed map services
 - Leverage dynamic caching services within the application
- Leverage the caching structure
 - Multiple internal caches are supported beginning in WebSphere V6.0
 - Split objects into different caches
 - Distribute cache data more efficiently

© Copyright IBM Corporation 2016

Figure 9-29. Dynamic caching (2 of 3)

WA8152.2

Notes:

The dynamic cache service works within an application server Java virtual machine (JVM), intercepting calls to cacheable objects. For example, it intercepts calls through a servlet service method, and either stores the output of the object to the cache or serves the content of the object from the dynamic cache.

For more information about dynamic caching, see the following website:

http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/index.jsp?topic=%2Fcom.ibm.websphere.nd.doc%2Fae%2Fwelc6tech_dyn_intro.html

Dynamic caching (3 of 3)

Two types of caching functions:

- Policy based (defined via the `cachespec.xml` file)
 - Servlet, JSP
 - Commands (Java object)
 - Web service
 - Web service client
- API based
 - Distributed map
 - Cacheable servlet
 - Cacheable command
- Distribution features
 - Distribute cache contents to peer instances
 - Push content to HTTP servers and other edge components

© Copyright IBM Corporation 2016

Figure 9-30. Dynamic caching (3 of 3)

WA8152.2

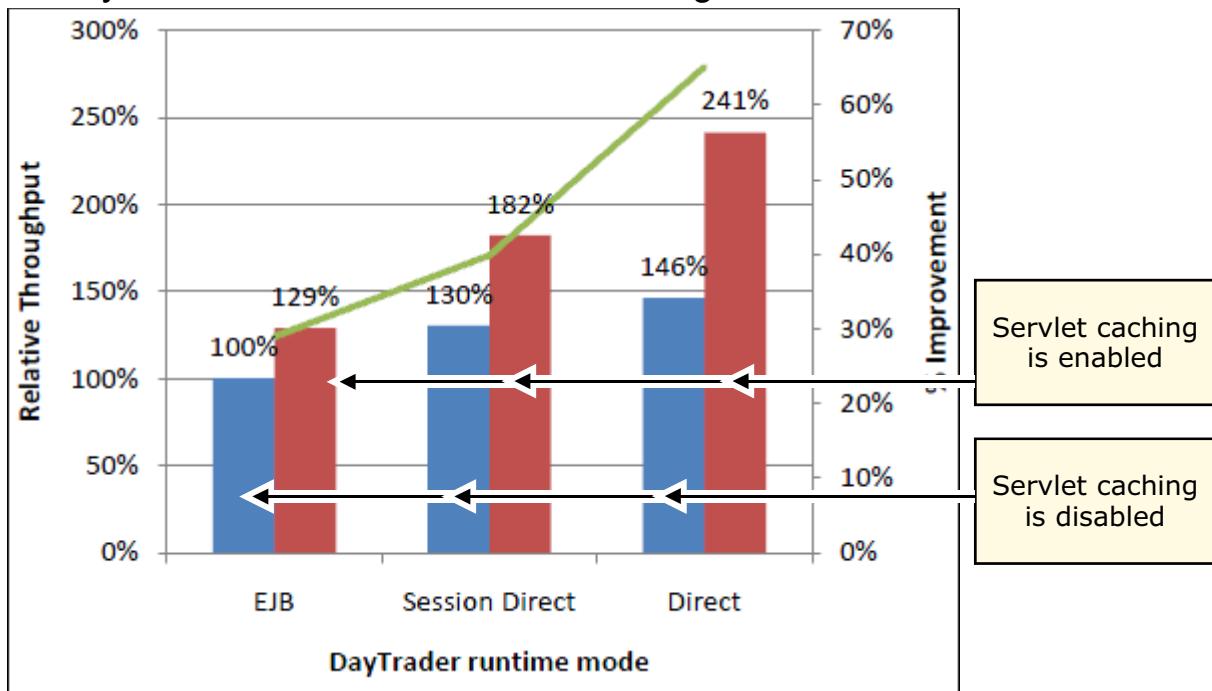
Notes:

Dynamic caching in WebSphere Application Server supports two types of caching: policy-based caching where the policy is defined in a file named “`cachespec.xml`” and API-based caching.

Dynamic caching supports cache distribution features, including the ability to distribute cache contents to peer instances and pushing content to HTTP servers and other “edge” components.

Performance benefit of servlet caching

- DayTrader benefits from servlet caching



© Copyright IBM Corporation 2016

Figure 9-31. Performance benefit of servlet caching

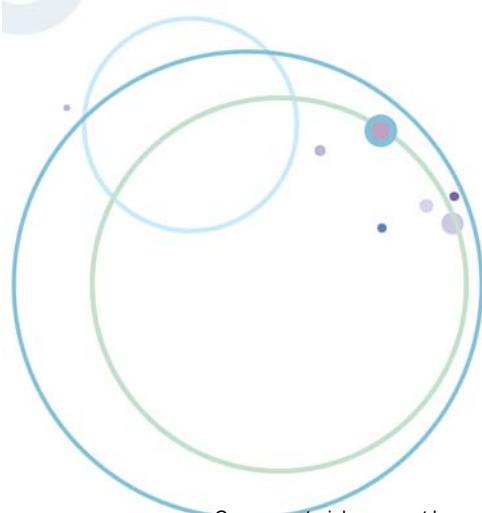
WA8152.2

Notes:

In the DayTrader application, a market summary is displayed on every access of a user's home page. This summary contains a list of the top five gaining and losing stocks, and the current stock index and trading volume. This list requires the execution of several database lookups and therefore significantly delays the loading of the user's home page. With servlet caching, the `marketSummary.jsp` can be cached, virtually eliminating these expensive database queries to improve the response time for the user home page. The refresh interval for the cached object can be configured. DynaCache might also be used to cache other servlet or JSP fragments and data within DayTrader. This example demonstrates the improvement one can achieve through caching to avoid complex server operations.

9.4. WebSphere default messaging (service integration bus)

WebSphere default messaging (service integration bus)



© Copyright IBM Corporation 2016

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

9.1

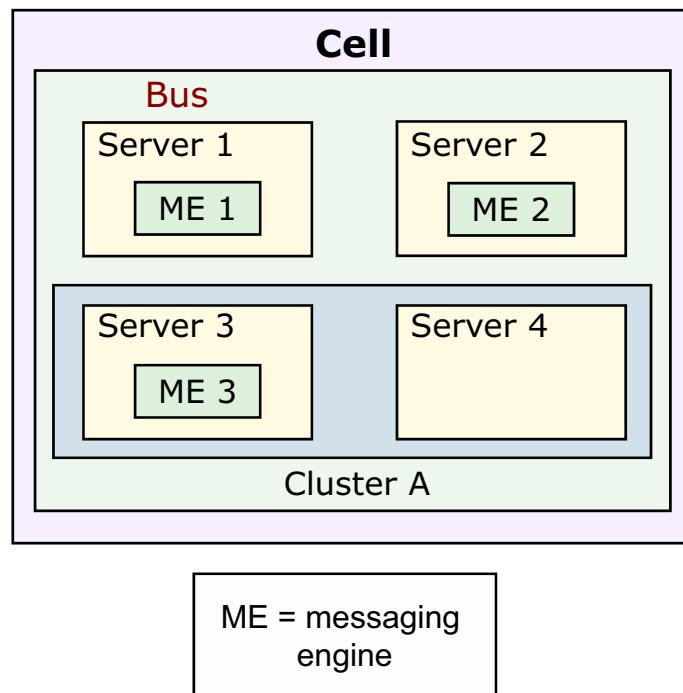
Figure 9-32. WebSphere default messaging (service integration bus)

WA8152.2

Notes:

WebSphere messaging concept view

- Applications connect to a bus
- Bus destinations are:
 - Queues
 - Topics
- An application that sends messages attaches to a destination as a producer
- An application that receives messages attaches to a destination as a consumer



© Copyright IBM Corporation 2016

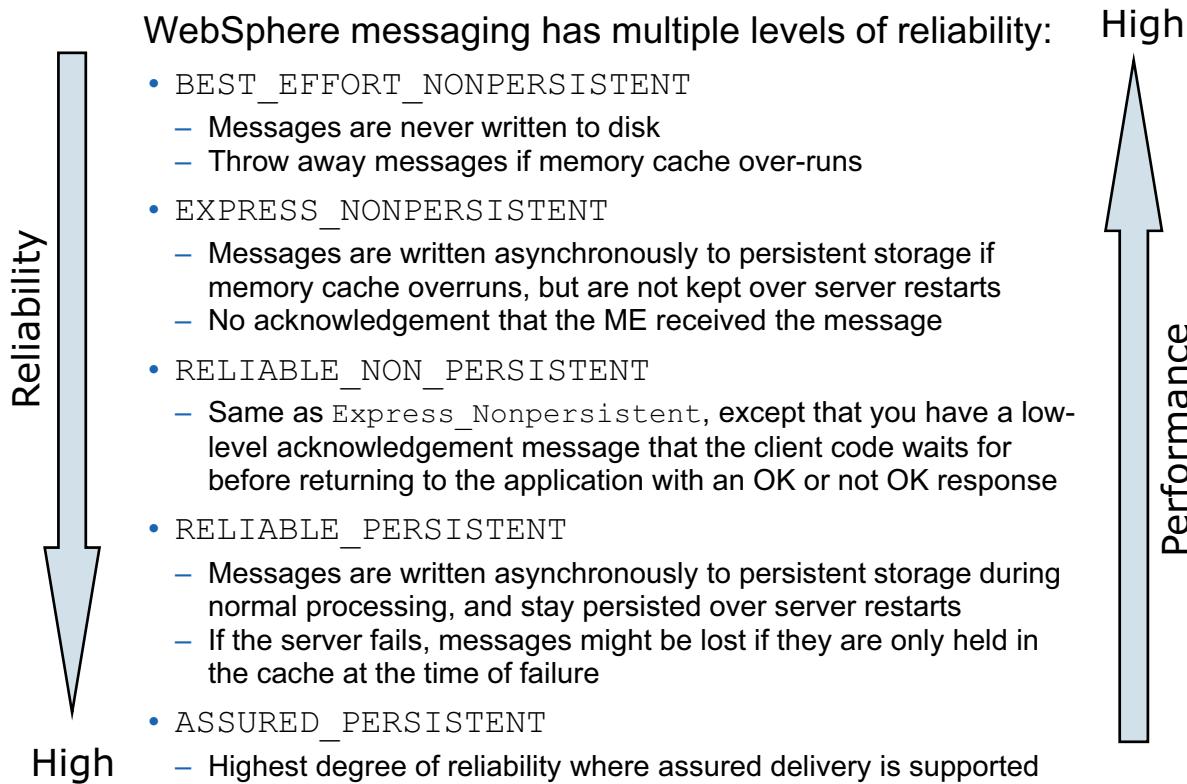
Figure 9-33. WebSphere messaging concept view

WA8152.2

Notes:

The way to think about a bus is to imagine all of its members interconnected. This way of thinking is in contrast to the WebSphere MQ world, where a queue is in a static place, hosted by only one Queue Manager. In WebSphere Default Messaging, the application connects to the bus as a whole and not an individual Queue Manager. As a result, all queues or topics on the bus are available to any application connected to the bus. It does not matter where the destinations were originally defined. Not pictured here is the concept of a message store. A message store is the place where a messaging engine stores state information and persistent message data.

WebSphere messaging: Reliability level



© Copyright IBM Corporation 2016

Figure 9-34. WebSphere messaging: Reliability level

WA8152.2

Notes:

WebSphere messaging has five levels of reliability:

- BEST_EFFORT_NONPERSISTENT
- EXPRESS_NONPERSISTENT
- RELIABLE_NON_PERSISTENT
- RELIABLE_PERSISTENT
- ASSURED_PERSISTENT

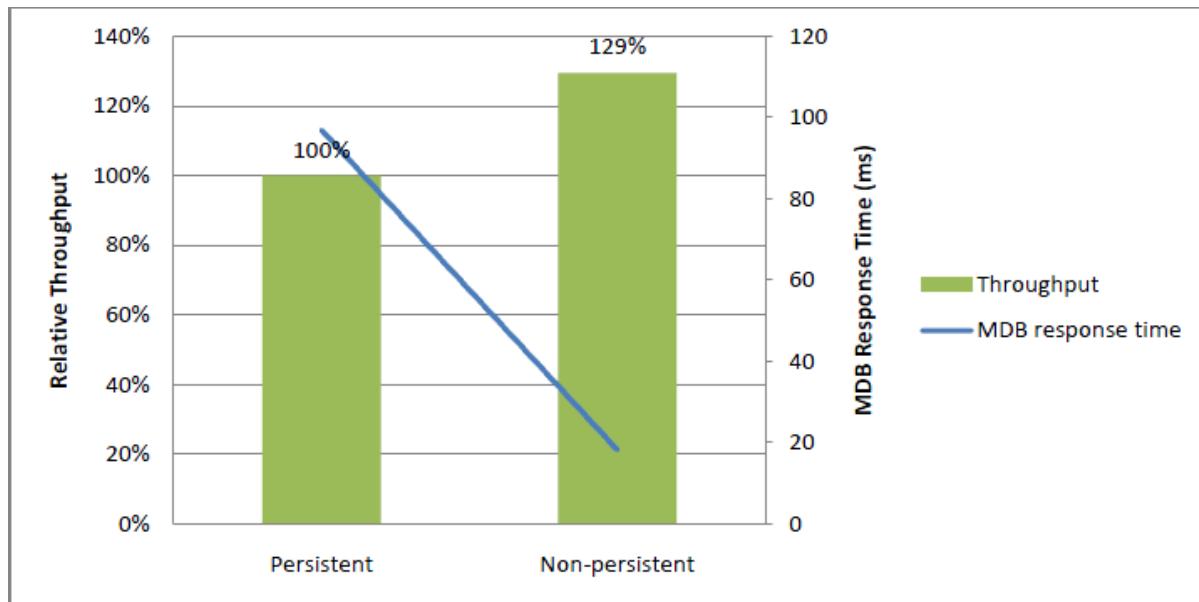
Lower levels of reliability are more efficient than higher levels of reliability. Reliability equates to CPU burn. Thus, higher levels of reliability burn more CPU per request or message than lower levels of reliability. Specifically, higher levels of reliability require more CPU capacity and utilization at a specific level of load than lower levels of reliability. Additionally, lower levels of reliability achieve a higher level of throughput at maximum (100%) CPU utilization than higher levels of reliability.

For more information about tuning WebSphere Messaging, see the following website:

http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/index.jsp?topic=%2Fcom.ibm.websphere.base.doc%2Fae%2Ftmj_tuning.html

Performance comparison of message reliability levels

- Persistent versus non-persistent



© Copyright IBM Corporation 2016

Figure 9-35. Performance comparison of message reliability levels

WA8152.2

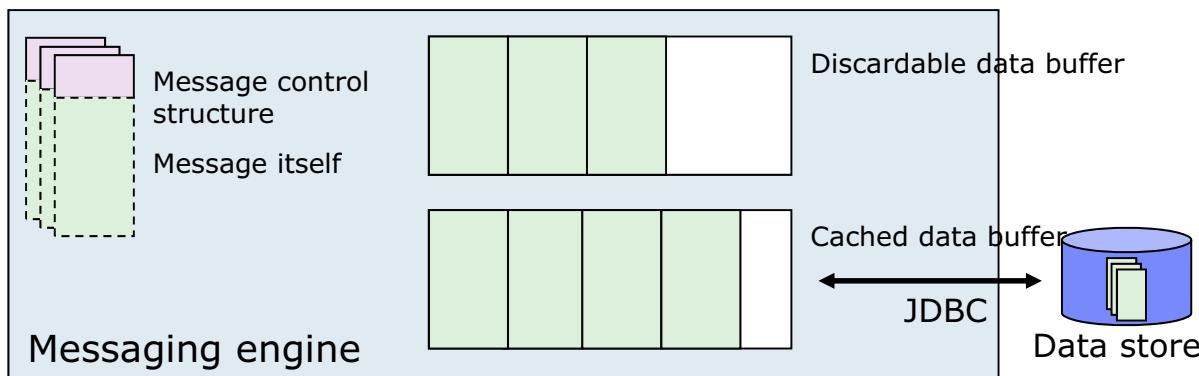
Notes:

Persistent messages are always stored to some form of persistent data store, while non-persistent messages are generally stored in volatile memory. There is a trade-off between reliability of message delivery and the speed with which messages are delivered. This slide shows the results for two test runs that use a file store with different reliability levels, "Assured persistent" and "Express non-persistent". The results are clear that as the reliability level decreases, the messages can be processed faster.

For more information, see the following WebSphere Application Server Performance Tuning Case Study:

http://www.ibm.com/developerworks/websphere/techjournal/0909_blythe/0909_blythe.html

WebSphere messaging data buffers (1 of 3)



- Each messaging engine has two data buffers to store messages:
 - Discardable data buffer
 - Cached data buffer
- Messages are discarded from the data buffers by using FIFO
 - Messages from cached data buffer are still available from the database
 - Messages from the discardable data buffer are lost when discarded
 - The discardable data buffer must be sized according to the expected level of load

© Copyright IBM Corporation 2016

Figure 9-36. WebSphere messaging data buffers (1 of 3)

WA8152.2

Notes:

WebSphere messaging uses “in memory” caching and buffering for both discardable and persistent messages. Discardable messages are buffered (in memory) in a data buffer that is named the “discardable data buffer”. The discardable messages are not written to the WebSphere messaging “data store”. Persistent messages are cached (in memory) in a data buffer that is named the “cached data buffer”. The size of both of these data buffers needs to be tuned. Specifically, the maximum size of the “discardable data buffer” should be tuned when using a level of reliability that might discard messages. The maximum size of the “cached data buffer” should be tuned when using a level of reliability that persists messages in the WebSphere messaging “data store”.

WebSphere messaging data buffers (2 of 3)

- The default size for both the cached data buffer and the discardable data buffer is 320 KB
 - Establish an estimate for the average message size to determine the number of messages that fill the data buffers
 - Size according to the wanted maximum number of messages
 - The default size of the buffers is likely to be too small
- Discarded messages from the discardable data buffer throw an exception
 - Look for `com.ibm.ws.sib.msgstore.OutOfCacheSpace` in `SystemOut.log`

© Copyright IBM Corporation 2016

Figure 9-37. WebSphere messaging data buffers (2 of 3)

WA8152.2

Notes:

The default size of both the “cached data buffer” and the “discardable data buffer” are likely to be too small for optimal performance for many applications. To configure these buffers, establish an estimate for average size of messages and then select the number of messages that need to be buffered. Then, multiply to determine the buffer size that is required in terms of bytes. An exception is thrown and recorded in the WebSphere “`SystemOut.log`” file when messages are discarded (when using a level of reliability that might discard messages). Thus, “`SystemOut.log`” should be monitored when using a level of reliability that might discard messages. No exceptions are thrown (or recorded) when using a level of reliability that writes messages to the WebSphere messaging data store.

WebSphere messaging data buffers (3 of 3)

- Discarded messages from the cached data buffer do **not** throw an exception
 - But messages might need to be retrieved from the database, hurting performance
 - Actively monitor PMI data
- PMI: **Performance Modules > SIB Service > SIB Messaging Engines > Messaging Engine > Storage Management > Cache**
 - CacheStoredDiscardCount (for cache data buffer)
 - CacheNotStoredDiscardCount (for discardable data buffer)

© Copyright IBM Corporation 2016

Figure 9-38. WebSphere messaging data buffers (3 of 3)

WA8152.2

Notes:

The number of messages that are discarded from the WebSphere messaging data buffers can be monitored with PMI.

WebSphere messaging data stores options

- The data store serves as a persistent repository for messages and operating information
- File-based data store (default)
 - Messages are persisted to the file system
 - Can perform as fast as a remote database
 - Performance is improved when a fast disk such as RAID is used
- High availability considerations file-based data store
 - Place file on highly available, shared drive
 - Use IBM File System Locking Protocol Test for verification
- Remote database data stores
 - Messages are persisted to a remote database
 - Frees up cycles for the application server JVM process that were previously used to manage the file-based stores
 - Can use high performance, production level database servers (DB2, Oracle)

© Copyright IBM Corporation 2016

Figure 9-39. WebSphere messaging data stores options

WA8152.2

Notes:

RAID = Redundant Array of Independent Disks

A third option is the **Local Derby database data store**. With this option, a local, in-process Derby database is used to store the operational information and messages that are associated with the messaging engine. Although convenient for development purposes, this configuration uses valuable cycles and memory within the application server to manage the stored messages.

One technical advantage of using a database for the data store is that some Java EE applications can share JDBC connections to benefit from one-phase commit optimization. See “Sharing connections to benefit from one-phase commit optimization”. File store does not support this optimization.

WebSphere messaging has two data store options: “Database” and “flat file”. The “flat file” data store option might be faster than the “database” data store option in some cases. Issues other than performance should be considered when choosing the best data store option. Consider high availability concerns when considering the “flat file” data store option. The file system lock protocol should be tested if the “flat file” data store option is being considered.

Unit summary

Having completed this unit, you should be able to:

- Identify and tune parameters that are associated with database connectivity, EJBs, and dynamic caching
- Identify and tune parameters that are associated with WebSphere messaging
- Implement performance best practices for WebSphere clusters
- Explain performance considerations associated with using 64-bit WebSphere

© Copyright IBM Corporation 2016

Figure 9-40. Unit summary

WA8152.2

Notes:

Checkpoint questions

1. True or False: MinSpareServers and MaxSpareServers are web server tuning parameters.
2. True or False: The maximum connections parameter specifies the maximum number of physical connections that you can create in a data source connection pool.
3. True or False: The dynamic cache caches JIT-compiled Java methods.
4. True or False: As the WebSphere messaging level of reliability increases, message processing performance also increases.

© Copyright IBM Corporation 2016

Figure 9-41. Checkpoint questions

WA8152.2

Notes:

Write your answers here:

- 1.
- 2.
- 3.
- 4.



Checkpoint answers

1. True
2. True
3. False: The dynamic cache service improves performance by caching the output of servlets, commands, and JSPs.
4. False: The higher levels of messaging reliability cause a decrease in general message processing performance.

© Copyright IBM Corporation 2016

Figure 9-42. Checkpoint answers

WA8152.2

Notes:

Unit 10. Application profiling and tuning

What this unit is about

This unit describes two tools that can be used to help tune application performance.

What you should be able to do

After completing this unit, you should be able to:

- Describe areas of performance bottlenecks for applications
- Use the IBM Java Health Center to investigate memory issues in applications
- Use IBM Java Health Center to profile application execution
- Use IBM Java Health Center to examine application thread information
- Use IBM Java Health Center to examine application lock information

How you will check your progress

- Checkpoint
- Lab Exercise

References

Application profiling performance considerations,

http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/index.jsp?topic=%2Fcom.ibm.websphere.nd.doc%2Fae%2Frprf_appprofiletune.html

IBM Redbooks, Rational Application Developer V7.5 Programming Guide,
SG24-7672-00

Unit objectives

After completing this unit, you should be able to:

- Describe areas of performance bottlenecks for applications
- Use the IBM Java Health Center to investigate memory issues in applications
- Use IBM Java Health Center to profile application execution
- Use IBM Java Health Center to examine application thread information
- Use IBM Java Health Center to examine application lock information

© Copyright IBM Corporation 2016

Figure 10-1. Unit objectives

WA8152.2

Notes:

Application bottlenecks

- Caused by limited resources
- Applications can be:
 - CPU bound
 - I/O bound
 - Space bound
 - “Lock bound” (contention)

© Copyright IBM Corporation 2016

Figure 10-2. Application bottlenecks

WA8152.2

Notes:

Limited resources cause application bottlenecks. There are four basic kinds of bottlenecks:

- CPU bound applications must wait for processor resource time to complete work.
- I/O bound applications wait for I/O operations to finish before the application can complete work.
- Space bound applications spend too much time on memory management rather than completing the work of the application.
- Lock bound applications have multiple threads that are trying to access the same application data at the same time; and some of the threads must wait before the thread can complete its work.

What resource is limited?

- CPU bound
 - Processor utilization consistently high
- I/O bound
 - Processor utilization not consistently high
- Space bound
 - Processor utilization could be high, or not
- Lock bound
 - Processor utilization not consistently high
- Level of processor utilization is not enough to know where the problem is
- Need to refine this analysis further

© Copyright IBM Corporation 2016

Figure 10-3. What resource is limited?

WA8152.2

Notes:

There are some typical indicators of the bottleneck type:

- CPU bound applications obviously have high processor utilization.
- I/O bound applications are waiting for I/O operations to complete rather than using the processor, so processor utilization is not consistently high.
- Space bound applications might use CPU for garbage collection, or might be blocked on I/O operations if memory is being paged to disk.
- Lock bound applications cannot use all of the processors in a multi-processor architecture, so CPU utilization is not consistently high.

The CPU utilization metric is not enough to narrow down the actual bottleneck.

What is the JVM doing?

Use tools to understand the behavior of the JVM

Many tools available through ISA:

- Garbage Collection and Memory Visualizer
- Thread and Monitor Dump Analyzer
- The Java Health Center
 - Analyses information from a running JVM
 - Helps find application bottlenecks
 - Makes recommendations
- Rational Application Developer profiling perspective
 - Analyzes application runtime behavior
 - Indicates what parts of an application to focus on for tuning
 - Not covered in this course

© Copyright IBM Corporation 2016

Figure 10-4. What is the JVM doing?

WA8152.2

Notes:

To narrow down where the bottleneck is, you must understand what is going on inside the JVM. There are many tools available to you. Garbage collection and heap analysis tools were presented earlier. The Java Health Center was used for tuning the JVM in general. The Java Health Center is also used to analyze particular applications that are running inside a WebSphere Application Server JVM. The Rational Application Developer profiling perspective provides a complete suite of functions for analyzing application behavior. This unit focuses only on the Java Health Center for application analysis.

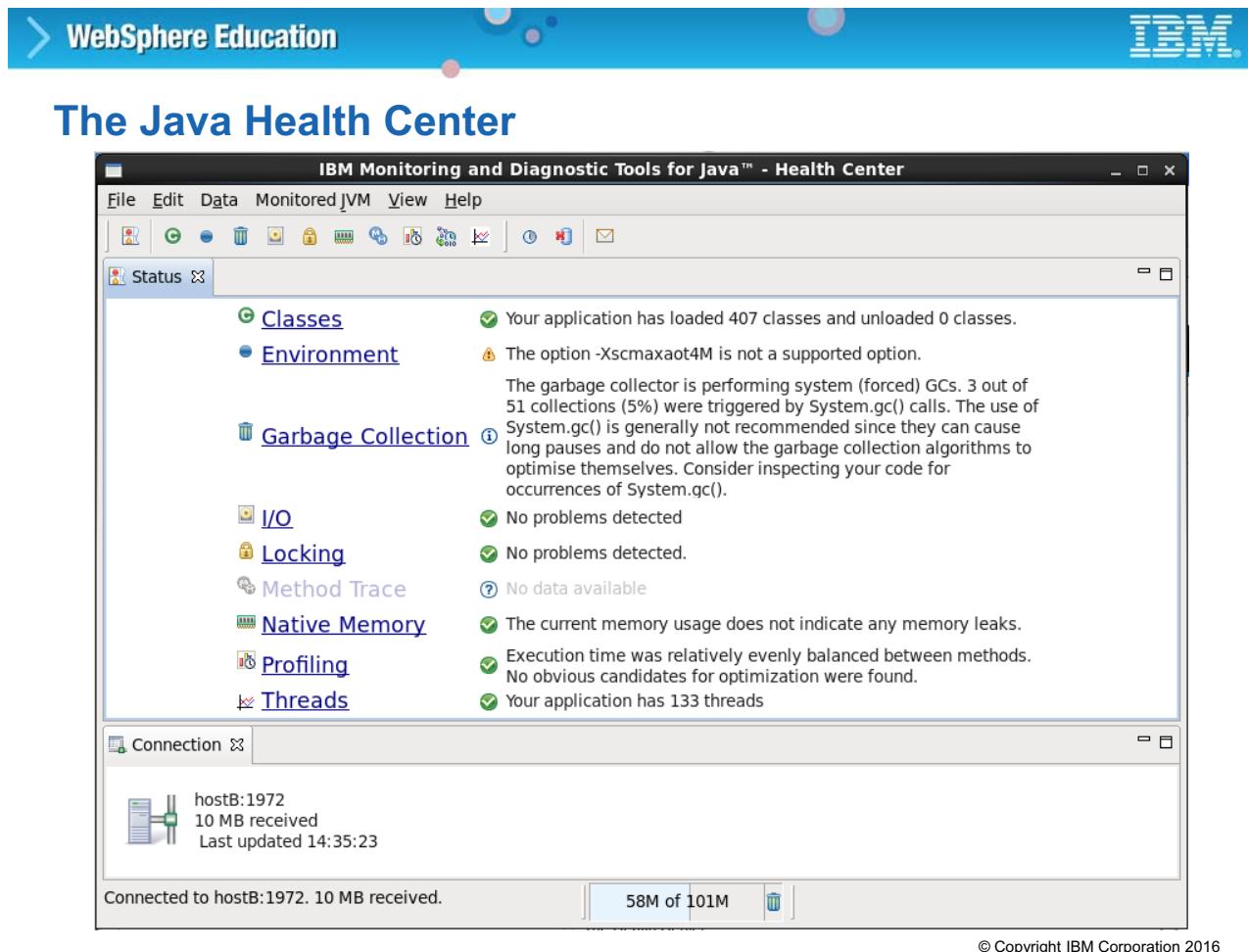


Figure 10-5. The Java Health Center

WA8152.2

Notes:

The Java Health Center has different perspectives for analyzing different aspects of the JVM. Each perspective has the high-level results of the analysis of that perspective. Icons indicate whether there is anything of interest for each perspective.



The Java Health Center

- Helps answer
 - What is the JVM doing?
 - Is anything wrong?
 - Why is an application slow?
 - Why does an application not scale?
 - Do the options that are used make sense?
- Analyzes data and gives recommendations
- Live monitoring with low performance impact

© Copyright IBM Corporation 2016

Figure 10-6. The Java Health Center

WA8152.2

Notes:

The different perspectives of the Java Health Center help answer questions about what is happening in the JVM. Each perspective provides analysis and gives recommendations for what might be going wrong.

The Java Health Center has a low performance impact, less than 1% or 2% on a loaded system.



The Java Health Center perspectives

- Classes
- Environment
- Garbage Collection
- File I/O
- Locking
- Method Trace
- Native Memory
- Profiling
- Threads

© Copyright IBM Corporation 2016

Figure 10-7. The Java Health Center perspectives

WA8152.2

Notes:

The class loading perspective shows information on how classes are loaded over time.

The environment perspective shows system and configuration information about the monitored JVM.

The garbage collection perspective provides a view of how garbage collection is behaving.

The file I/O perspective provides information about file open and close activities that the JVM performs.

The locking perspective allows you to view lock usage and identify possible points of contention.

The method trace perspective shows accurate timings for method calls, and an analysis of method use across the running threads.

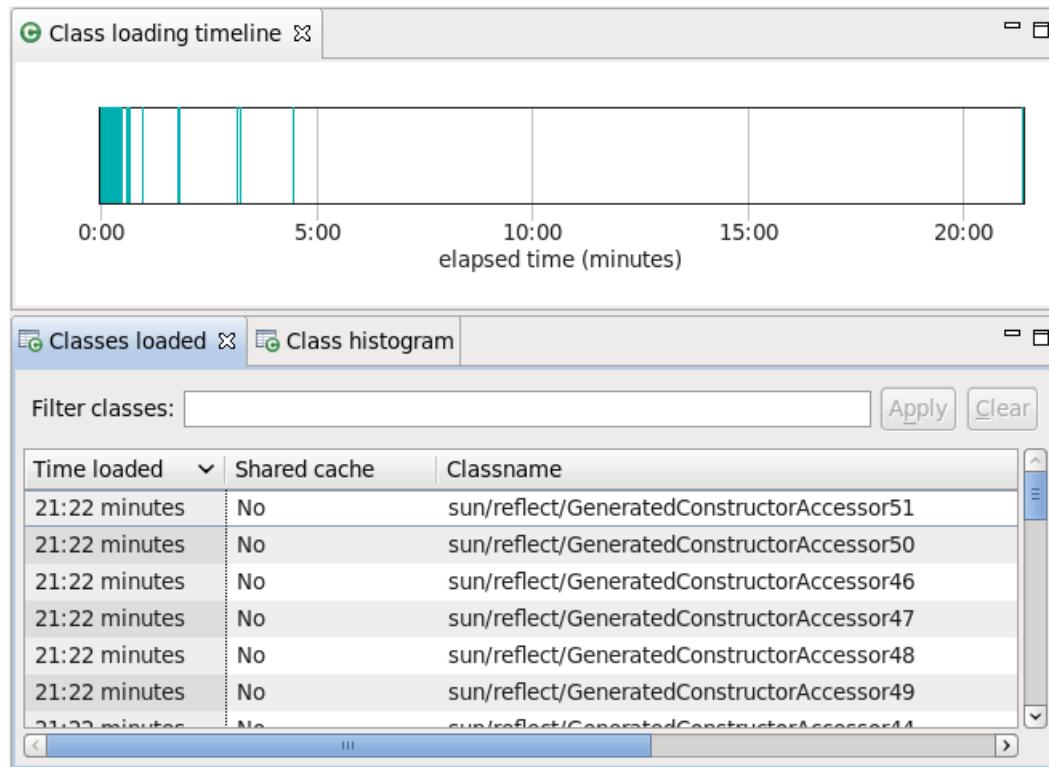
The native memory perspective provides information about the native memory usage of the process and system that is being monitored.

The profiling perspective shows you which methods are run most often, and in which order.

The threads perspective provides information, in graph and table form, about all the live threads that are attached to the monitored JVM.



The Java Health Center Classes perspective



© Copyright IBM Corporation 2016

Figure 10-8. The Java Health Center Classes perspective

WA8152.2

Notes:

The classes perspective displays the density of class loading over time, which classes were loaded at which time, and whether a class was loaded from the class sharing cache. You can also request the collection of class histogram data to see a snapshot of the classes that are in the heap, and the amount of heap space that the class instances are occupying.



The Java Health Center Classes perspective

- Class loading graph
 - How much class loading occurred
 - Use to identify when classes were loading at a rate you did not expect
- Class loading table
 - A more detailed view of when classes were loaded
 - Whether the class was loaded from the shared classes cache
 - Use to identify what classes were loaded at a particular time
- Class histogram
 - Which classes are on the heap
 - Number of instances of a class
 - Amount of heap space that instances of a class are occupying
 - Can help diagnose memory problems

© Copyright IBM Corporation 2016

Figure 10-9. The Java Health Center Classes perspective

WA8152.2

Notes:

The class loading graph gives a visual indication of how much class loading occurred in your application over time. Use the graph to identify points in time at which classes were loading at a rate you did not expect.

The classes loaded table gives a more detailed view of which classes are loaded at which times. This table also indicates whether the class was loaded from the shared classes cache.

The class histogram shows which classes are on the heap, the number of instances that exist for each class, and the amount of heap space that those instances are occupying. You can use this information to help diagnose memory problems.

This data comes from a snapshot of the heap; to trigger the snapshot and populate the histogram, click **Collect histogram data** in the Class histogram view. You can also trigger a snapshot from the Monitored JVM menu: Click **Monitored JVM > Collect histogram data....** There might be a short delay before the data is displayed.



The Java Health Center environment perspective

The screenshot shows the Java Health Center environment perspective interface. It includes tabs for Configuration, System properties, and Environment variables. The Configuration tab displays a list of properties like Boot classpath, Classpath, Dump options, Java command line, Java parameters, and Ulimit parameters. The Java runtime environment tab shows details such as Java virtual machine name (IBM J9 VM), Version (1.6), Process id (24880), Health Center Agent library build date (Nov 21 2012 14:01:11), Full version (JRE 1.6.0 IBM Linux bu...), Health Center Agent version (2.1.0.20121121), and Java home (/opt/IBM/WebSphere/...). The System tab provides system-level information like Host name (hostB), Operating system version (2.6.32-71.el6), Number of available processors (1), Operating system (Linux), and Architecture (amd64). A copyright notice at the bottom right reads "© Copyright IBM Corporation 2016".

Figure 10-10. The Java Health Center environment perspective

WA8152.2

Notes:

The environment perspective shows system and configuration information about the monitored JVM.

The Java Health Center environment perspective

- Shows system and configuration information about the monitored JVM
 - Version information for the JVM
 - Operating system and architecture information for the monitored system
 - Process ID
 - All system properties
 - All environment variables
- Use to confirm that you are monitoring the intended JVM
- Identifies JVM parameters that might adversely affect performance
 - Shows a warning

© Copyright IBM Corporation 2016

Figure 10-11. The Java Health Center environment perspective

WA8152.2

Notes:

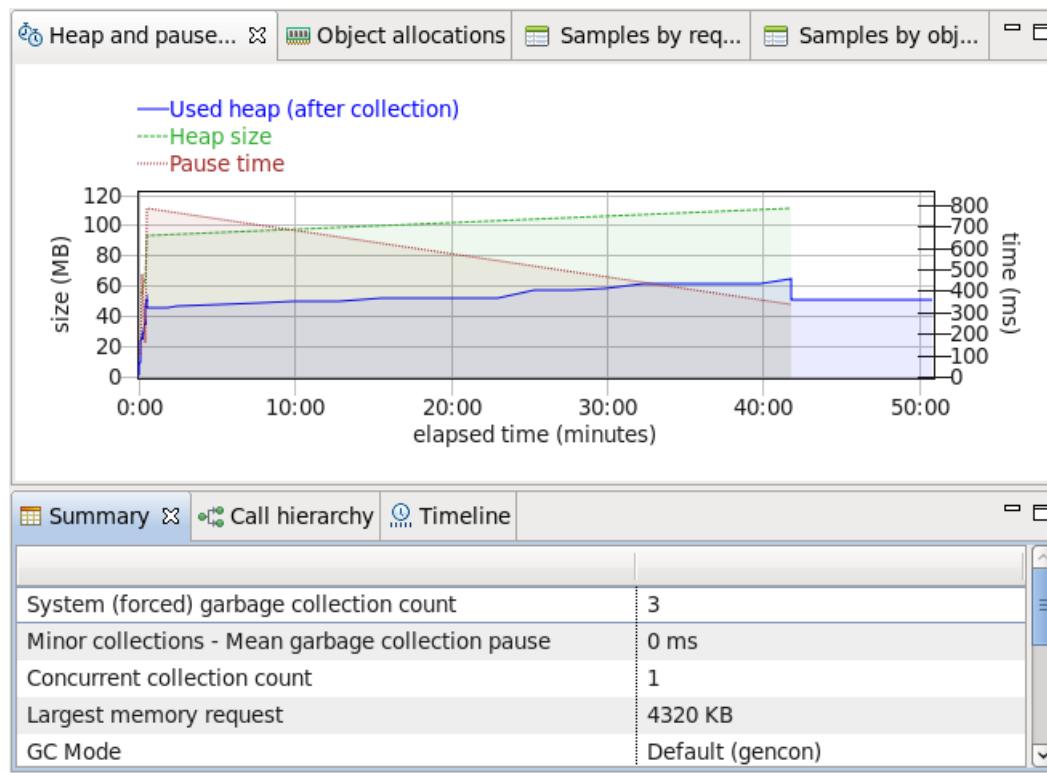
The information that is provided includes:

- Version information for the JVM
- Operating system and architecture information for the monitored system
- Process ID
- All system properties
- All environment variables

This information can be useful in confirming that the intended JVM is being monitored. You can also use this information to help diagnose some types of problems.

The Health Center identifies JVM parameters that might adversely affect system performance, stability, and serviceability. If any of these parameters are detected, a warning is displayed.

The Java Health Center Garbage Collection perspective



© Copyright IBM Corporation 2016

Figure 10-12. The Java Health Center Garbage Collection perspective

WA8152.2

Notes:

You can view the heap usage, pause times, summary table, object allocations, and tuning recommendation in the Health Center garbage collection perspective.



The Java Health Center Garbage Collection perspective

- Heap and pause times
 - Use to identify trends in application memory usage
 - Use to assess the performance affect of garbage collection
 - Use with heap analysis tools
- Object allocations
 - Use to identify which code is allocating large objects
- Samples by request site
 - Use to find code that is issuing allocation requests most frequently
- Samples by object
 - Use to find the objects that are being allocated most frequently
- Summary
 - Shows garbage collection metrics
- Call hierarchy
 - Displays the call paths to a selected allocation request site
- Timeline
 - A visual representation of when an allocation request was made

© Copyright IBM Corporation 2016

Figure 10-13. The Java Health Center Garbage Collection perspective

WA8152.2

Notes:

Use the graph of heap usage to identify trends in application memory usage. If the memory footprint is larger than expected, a heap analysis tool can identify areas of excessive memory usage. If the used heap is increasing over time, the application might be leaking memory.

Use the graph of pause times to assess the performance affect of garbage collection. When garbage collection is running, all application threads are paused. The aim of tuning garbage collection is to reduce pause times when low response times are required.

Use the object allocations view to identify which code is allocating large objects. You can use low threshold values and high threshold values to specify the object range that triggers collection of the call stack information.

For the Samples by request site and Samples by object views, data is displayed only when you enable the collection of call stacks for sampled object allocation events.

Use the Samples by request site view to find code that is issuing allocation requests most frequently.

Select a row in the table to display the call paths to the allocation request site in the Call hierarchy view, and a visual representation of when the allocation requests were made in the Timeline view.

Use the Samples by object view to find the objects that are being allocated most frequently.

Select a row in the table to display the call paths to the allocation request site in the Call hierarchy view.

The summary table shows garbage collection metrics, including mean pause time, mean interval between garbage collections, and the amount of time that is spent in garbage collection.

The Health Center provides general tuning recommendations and advice. The Health Center does not know what your quality of service requirements are; therefore the recommendations are not always useful. For example, a suggested change might improve application efficiency but increase pause times, which might not be best for your application. The tuning recommendations also indicate whether the application seems to be leaking memory. However, the Health Center cannot distinguish between naturally increasing memory requirements and memory that is being held when it is no longer required.

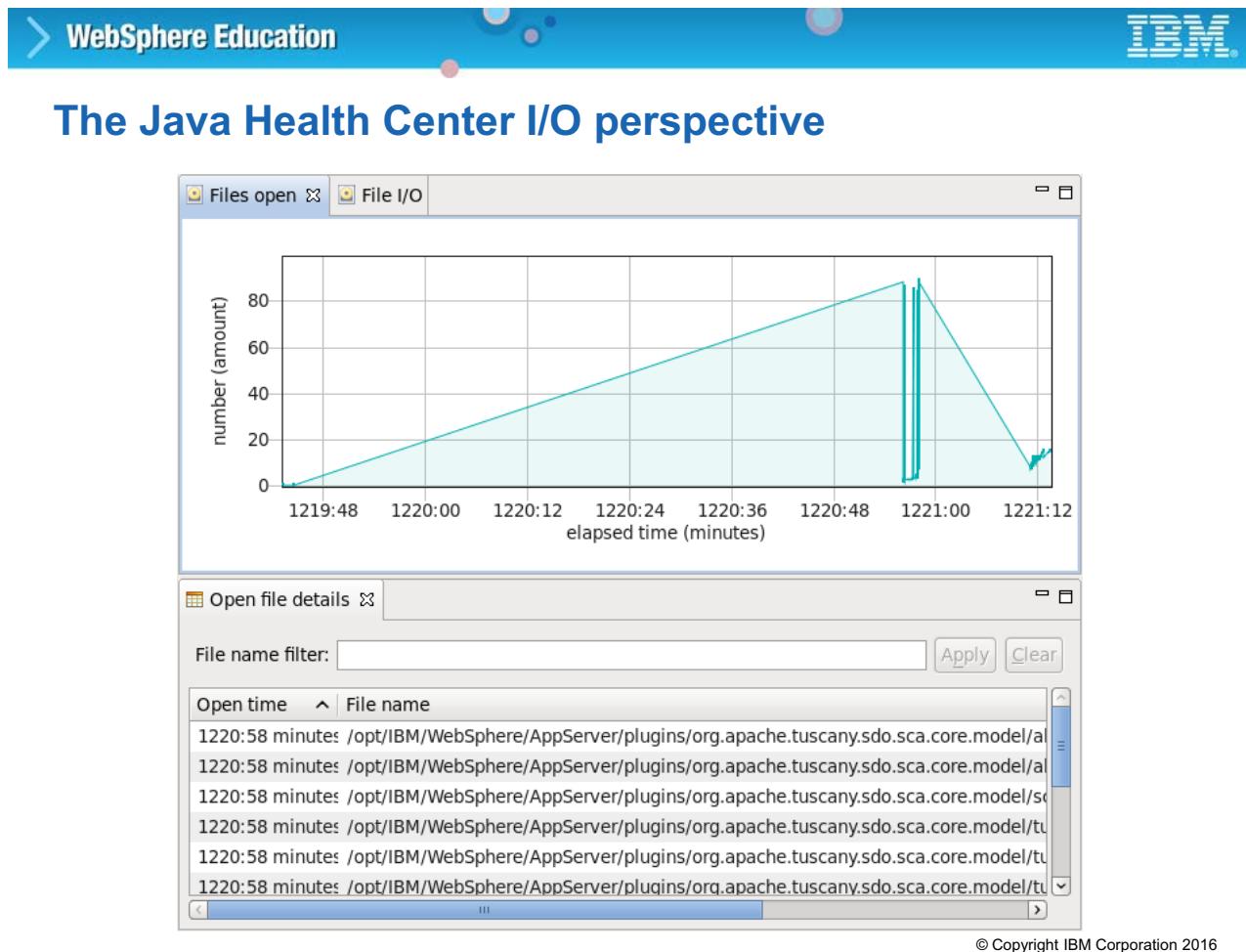


Figure 10-14. The Java Health Center I/O perspective

WA8152.2

Notes:

The I/O perspective provides information about three aspects.

- File open events
- File close events
- Details of files that are currently open



The Java Health Center I/O perspective

File open

- The number of files held open by the target application
- Use to find out whether the number of open files is increasing
 - Indicates that the application might not be closing file handles after use

File I/O

- Shows information about each file open or file close event
- Use to help you identify problems with I/O "bottlenecks"
- Does not show read/write I/O requests – use other tools

Open file details

- Shows information about the files that are held open by the target application
 - File name
 - Time it was opened

© Copyright IBM Corporation 2016

Figure 10-15. The Java Health Center I/O perspective

WA8152.2

Notes:

The file open view reports the number of files currently held open by the target application. Use this view to find out whether the number of open files is increasing. An increasing number indicates that the application might not be closing file handles after use.

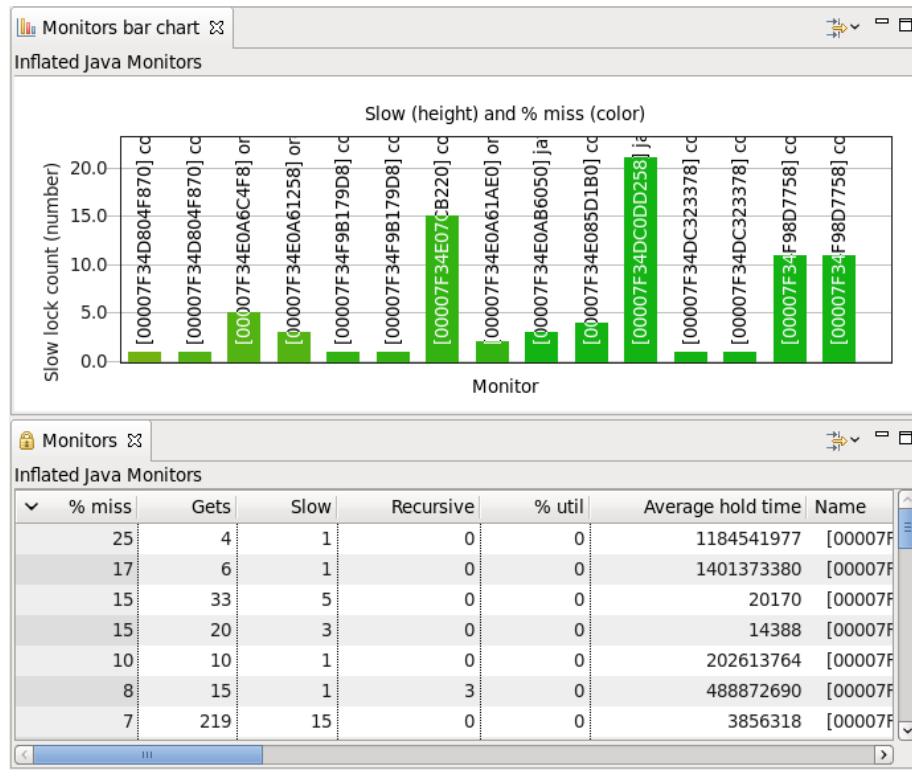
The file I/O view shows information about each file open or file close event. Use this view to help you identify problems with I/O "bottlenecks". This view does not show information about read or write operations. Too many read or write operations can cause performance problems. Use operating system tools to examine the amount of read/write activity for files.

Open file details view

The open file details view shows information about the files currently held open by the target application. The information includes the file name, and the time it was opened. You can filter the information in this view by using the text box before the view.



The Java Health Center Locking perspective



© Copyright IBM Corporation 2016

Figure 10-16. The Java Health Center Locking perspective

WA8152.2

Notes:

Use the locking perspective to review lock usage and identify possible points of contention.

The Java Health Center Locking perspective

Java monitors

- Synchronized Objects in the Java application
 - Java class libraries, middleware, Independent software packages, application code

System monitors

- Locks that are part of the Java runtime environment itself

Chart

- An overview of how contended the application locks are
- The height of the bars represents the slow lock count
 - A slow count occurs when the requesting thread is blocked
- The color of each bar is based on the value of the % miss column in the table
 - The gradient moves from red (100%), through yellow (50%), to green (0%)
 - A red bar indicates that the thread blocks every time
 - A green bar indicates a thread that never blocks

Only the most contended monitors are shown

© Copyright IBM Corporation 2016

Figure 10-17. The Java Health Center Locking perspective

WA8152.2

Notes:

The Java monitors view shows synchronized objects in the Java application, provided as part of the Java Class Libraries, middleware, independent software packages, or application code.

System monitors are locks that are part of the Java runtime environment itself.

Java monitors are shown by default and are most useful in resolving application contention issues. To show the system monitors, use the **Show...** icon in the toolbar of the table or chart.

The bar chart gives an overview of how contended the application locks are.

The height of the bars represents the slow lock count and is relative to all the columns in the graph. A slow count occurs when another thread already owns the requested monitor and the requesting thread is blocked.

The color of each bar is based on the value of the % miss column in the table. The gradient moves from red (100%), through yellow (50%), to green (0%). A red bar indicates that the thread blocks every time that the monitor is requested. A green bar indicates a thread that never blocks.

Only the most contended monitors are shown.

The Java Health Center Locking perspective

- Java monitors table

Column heading	Description
% miss	The percentage of Gets for which the thread had to block
Gets	The number of times the lock was taken while it was inflated
Slow	The number of non-recursive lock acquires where the requesting thread had to wait because the lock was owned
Recursive	The total number of recursive acquires
% util	The amount of time the lock was held, divided by the amount of time the output was taken over
Average hold time	The average amount of time the lock was held by a thread
Name	The monitor name, blank if the name is not known

- A high % miss shows contention on the resource that the lock protects
 - This contention might be preventing the Java application from scaling further

© Copyright IBM Corporation 2016

Figure 10-18. The Java Health Center Locking perspective

WA8152.2

Notes:

The % miss column is of initial interest. A high % miss shows that frequent contention occurs on the synchronized resource that the lock protects. This contention might be preventing the Java application from scaling further.

A high % util indicates that the lock is used a lot. A high average hold time indicates that the code that uses the lock holds it for a long amount of time.



The Java Health Center Locking perspective

- If a lock has a high % miss value
 - Look at the average hold time and % util
 - If % util and average hold time are both high, try to reduce the amount of work that is done while the lock is held
 - If % util is high but the average hold time is low, try to make the resource that the lock protects more granular to separate the lock into multiple locks

© Copyright IBM Corporation 2016

Figure 10-19. The Java Health Center Locking perspective

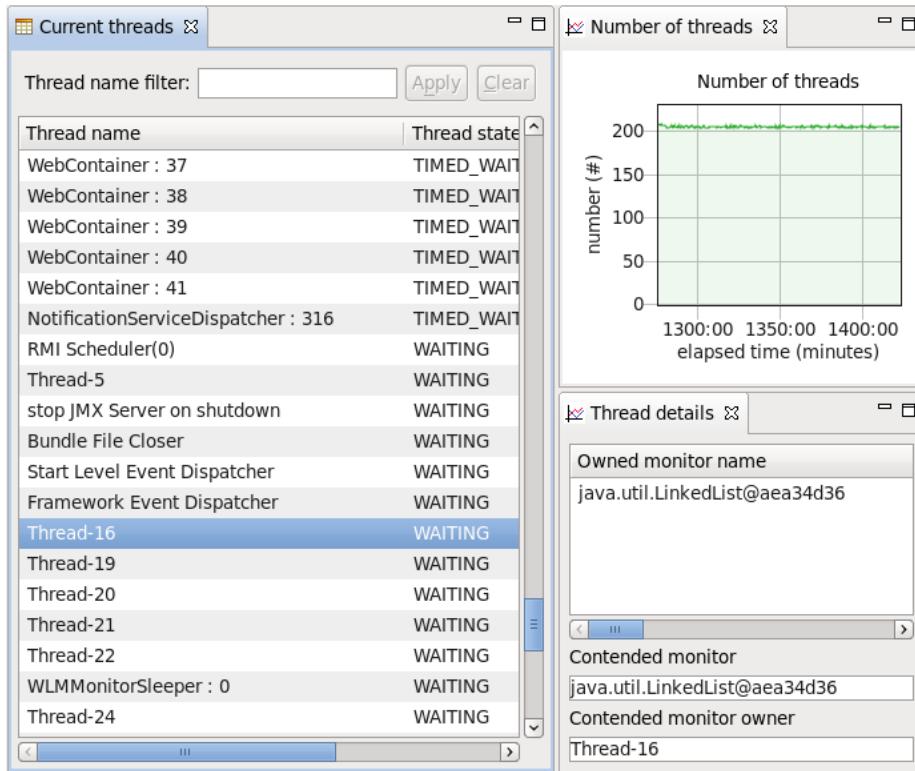
WA8152.2

Notes:

If a lock has a high % miss value, look at the average hold time and % util. If % util and average hold time are both high, you might need to reduce the amount of work that is done while the lock is held. If % util is high but the average hold time is low, you might need to make the resource that the lock protects more granular to separate the lock into multiple locks.



The Java Health Center Threads perspective



© Copyright IBM Corporation 2016

Figure 10-20. The Java Health Center Threads perspective

WA8152.2

Notes:

The threads perspective provides information, in graph and table form, about all the live threads that are attached to the monitored JVM.



The Java Health Center Threads perspective

Number of threads

- The number of live threads that are attached to the monitored JVM
- Use to see whether the number of threads is increasing
 - Might cause performance problems

Current threads

- A snapshot of the threads in the monitored JVM
 - Thread name and state
- Use to understand the activity of the threads
 - Blocked, waiting for monitors, and so on

Thread details

- Shows the details of the thread that is selected in the "Current threads" view
 - List of all the monitors that the selected thread owns
 - Contended monitor, if there is one

Thread stack

- Call stack information for a thread that you select in the "Current threads" view

© Copyright IBM Corporation 2016

Figure 10-21. The Java Health Center Threads perspective

WA8152.2

Notes:

The number of threads view shows the number of live threads that are attached to the monitored JVM. Use this view to see whether the number of threads is increasing, which might cause performance problems.

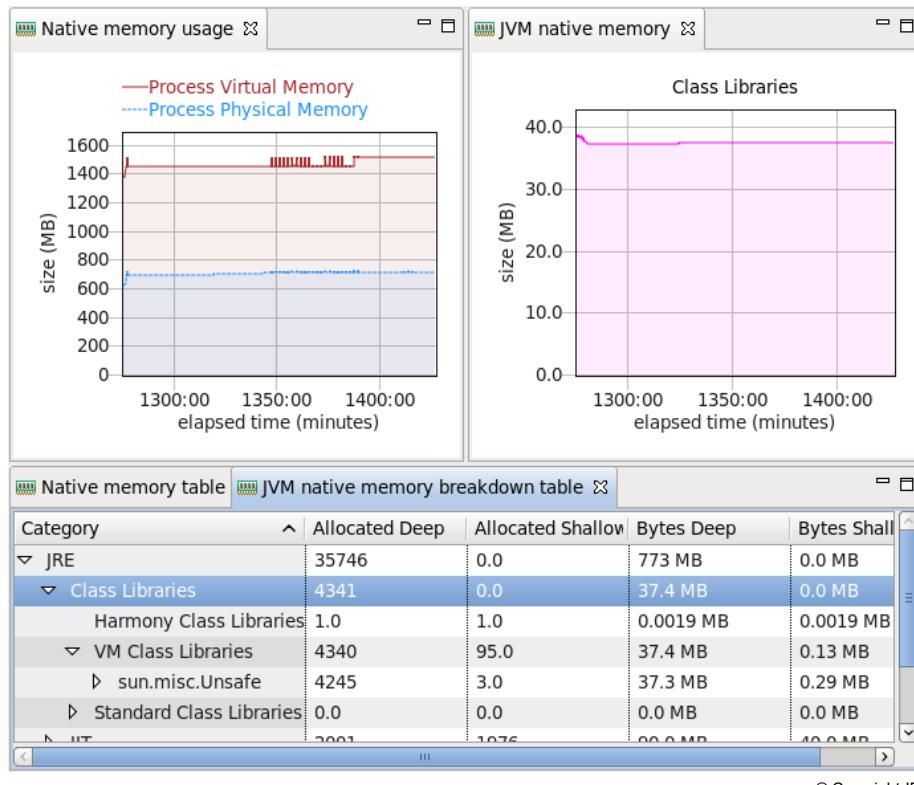
The current threads view shows a snapshot of the threads in the monitored JVM. For each thread, the view shows the thread name and state. Use this view to understand the activity of the threads, for example whether they are blocked or waiting for monitors. Select a thread in this view to see more information.

This thread details view shows the details of the thread that you selected in the "Current threads" view. This view shows a list of all the monitors that the selected thread owns, and the *contended monitor*, if there is one. The contended monitor is the object whose monitor the selected thread is waiting to enter or waiting to regain.

The thread stack view shows the call stack information for a thread that you select in the "Current threads" view. Use this information to understand the hierarchy of method calls in the thread.



The Java Health Center Native memory perspective



© Copyright IBM Corporation 2016

Figure 10-22. The Java Health Center Native memory perspective

WA8152.2

Notes:

The native memory perspective provides information about the native memory usage of the process and system that are being monitored.

The Java Health Center Native memory perspective

Native memory usage

- The process virtual memory and process physical memory
- Helps you to monitor the native memory usage by processes
- Compare to the Used Heap graph in the garbage collection perspective
 - See whether the amount of memory that is used is due to the size of the Java Heap or due to the memory allocated natively

JVM native memory

- The native memory that the selected items use in the native memory table

Native memory table

- The latest, minimum, and maximum values for native memory information
- Use to see the most recent memory usage information for your application

© Copyright IBM Corporation 2016

Figure 10-23. The Java Health Center Native memory perspective

WA8152.2

Notes:

The native memory usage view plots the process virtual memory and process physical memory on a graph. The information that is presented helps you to monitor the native memory usage by processes. Compare this graph to the Used Heap graph in the garbage collection perspective. You can see whether the amount of memory that an application uses is due to the size of the Java Heap or due to the memory allocated natively.

The JVM native memory view plots the native memory use of the items in the selected row of the JVM native memory breakdown table.

The native memory table view displays a table that contains the latest, minimum, and maximum values for all the available native memory information. You can use this table to see the most recent memory usage information for your monitored application.

The Java Health Center Native memory perspective

- JVM native memory breakdown table
 - Breakdown of the areas of the JVM that are using native memory
 - If you select a row in the table, that entry is shown as a graph of memory use over time, in the JVM native memory view

Column Name	Description
Allocated Deep	The number of objects that are allocated for that area and all of its child objects
Allocated Shallow	The number of objects that are allocated for that area only
Bytes Deep	The actual memory use for that area and all of its child objects
Bytes Shallow	The actual memory use for that area only

© Copyright IBM Corporation 2016

Figure 10-24. The Java Health Center Native memory perspective

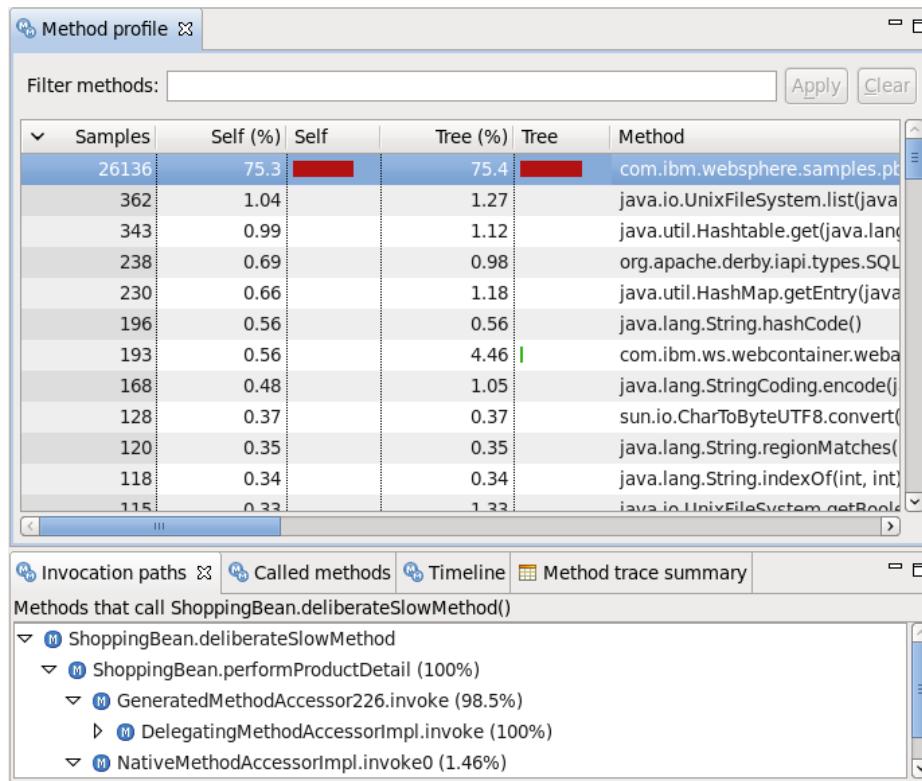
WA8152.2

Notes:

JVM native memory breakdown table shows the breakdown of the areas of the JVM that are using native memory. If you select a row in the table, that entry is shown as a graph of memory use over time, in the JVM native memory view. The table entries give information about the amount of memory that is held by each object, or by each object and all of its child objects.



The Java Health Center profiling perspective



© Copyright IBM Corporation 2016

Figure 10-25. The Java Health Center profiling perspective

WA8152.2

Notes:

The Profiling perspective shows you which methods are run most often, and in which order.



The Java Health Center profiling perspective

Sampling profiler

- Samples call stacks periodically
- Does not record every method that is run
 - Methods that do not run often might not appear
 - Methods that run quickly might not appear
 - Inline methods do not appear

Low performance impact

© Copyright IBM Corporation 2016

Figure 10-26. The Java Health Center profiling perspective

WA8152.2

Notes:

The profiling perspective shows method profiles and call hierarchies. The profiler takes regular samples to see which methods are running. Only methods that are called often, or take a long time to complete, are shown.

Profiling perspective Method Profile view

- Shows which methods are using the most processing resource
- Methods with a high Self (%) are good candidates for optimization
 - Small improvements to the efficiency of these methods might have a large effect on performance
 - Reduce the amount of work that they do
 - Reducing the number of times that they are called

Column Heading	Description
Self (%)	The percentage of samples that are taken while a method was at the top of the stack
Self	A graphical representation of the Self (%) column
Tree (%)	The percentage of samples that are taken while a method was anywhere in the call
Tree	A graphical representation of the Tree (%) column
Samples	The number of samples that are taken while a method was at the top of the stack
Method	A fully qualified representation of the method, including package name, class name, method name, arguments, and return type

© Copyright IBM Corporation 2016

Figure 10-27. Profiling perspective Method Profile view

WA8152.2

Notes:

The Method Profile table shows which methods are using the most processing resource.

Methods with a higher Self (%) value are described as "hot", and are good candidates for optimization. Small improvements to the efficiency of these methods might have a large effect on performance. Methods near the end of the table are poor candidates for optimization. Even large improvements to their efficiency are unlikely to affect performance because they do not use as much processing resource.

You can optimize methods by reducing the amount of work that they do or by reducing the number of times that they are called.

Profiling perspective Invocation paths view

- Shows the methods that called the highlighted method
- If more than one method calls the highlighted method, a weight is shown in parentheses

Example:

- Another `constructString()` method (81.3% of samples)
- `TraceBuffer()` occasionally calls the `Util.constructString()` method (18.7% of samples)
- `Util.constructString`
 - `constructString (81.3%)`
 - `processTraceBuffer (18.7%)`

In this case, you have two strategies for optimization

- Make the `Util.constructString()` method more efficient
- Reduce how often `Util.constructString()` is called
 - Reducing how often `processTraceBuffer()` calls `constructString()` makes less difference than reducing how often `constructString()` calls `Util.constructString()`

© Copyright IBM Corporation 2016

Figure 10-28. Profiling perspective Invocation paths view

WA8152.2

Notes:

The Invocations paths tab shows the methods that called the highlighted method.

If more than one method calls the highlighted method, a weight is shown in parentheses. For any method, the sum of the percentages of its calling methods is 100%.

An example shows `constructString()` often calls the `Util.constructString()` method (81.3% of the samples). Another method, `processTraceBuffer()`, occasionally calls the `Util.constructString()` method (18.7% of the samples).

In this case, you have two strategies for optimization. The first is to make the `Util.constructString()` method more efficient. The second is to reduce how often it is called. Reducing how often `processTraceBuffer()` calls `constructString()` makes less difference than reducing how often `constructString()` calls `Util.constructString()`.



Profiling perspective Called methods view

Shows the methods that the highlighted method called

- If only the highlighted method is shown, that method had no called methods that were sampled
- If the method has child methods in the tree, the percentages typically do not add up to 100%
 - The difference in percentages indicates the time that is spent in the body of the highlighted method
- The Called methods tab is less useful for performance tuning than the Invocation paths tab
 - Any inefficient child methods also typically show up in the method profile table

© Copyright IBM Corporation 2016

Figure 10-29. Profiling perspective Called methods view

WA8152.2

Notes:

The Called methods tab shows the methods that the highlighted method called. In other words, they show where the highlighted method is doing its work.

If only the highlighted method is shown, no methods that are that method called were sampled. Either the methods called ran quickly, or they were inlined. If the method has child methods in the tree, the percentages typically do not add up to 100%. The percentages for child methods never add up to more than 100%. The difference in percentages indicates the time that is spent in the body of the highlighted method.

The Called methods tab is less useful for performance tuning than the Invocation paths tab. Time that is spent processing child methods is not counted as time spent processing the parent. A lightweight method that calls some inefficient child method is not placed high in the method profile table. Any inefficient child methods typically show up in the method profile table also.



Profiling perspective Timeline view

- Shows when the methods were invoked
- Gives a visual indication of when a method was invoked in your application
- Use the graph to see whether a method is used regularly throughout the lifecycle of your application
 - Some methods might be used only at a specific stage in the lifecycle, such as startup
 - This information can help you decide whether the method is a good target for optimization

© Copyright IBM Corporation 2016

Figure 10-30. Profiling perspective Timeline view

WA8152.2

Notes:

The timeline tab shows when the methods were invoked.

The method profiling timeline gives a visual indication of when a method was invoked in your application. You can use the graph to see whether a method is used regularly throughout the lifecycle of your application. Some methods might be used only at a specific stage in the lifecycle, such as startup. This information can help you decide whether the method is a good target for optimization.



The Java Health Center Method trace perspective

- Shows accurate timings for method calls and an analysis of method use across the running threads
- You must first generate some trace data
- Data tree
 - Shows the threads that each method ran on
- Trace summary
 - Count: The number of times that the method ran
 - Total time: The total time that the method ran for
 - Max time: The longest time that the method ran for
 - Mean time: The mean average time that the method ran for
 - Method name: The name of the method
- Trace timeline
 - A visual representation of when the method was used on the displayed thread or threads

© Copyright IBM Corporation 2016

Figure 10-31. The Java Health Center Method trace perspective

WA8152.2

Notes:

This perspective shows accurate timings for method calls, and an analysis of method use across the running threads. You must enable tracing for some methods before information is available in this perspective.

The method trace data tree view shows the threads that each method ran on. Expand a method to display the associated threads. Select a thread to show it in the method trace timeline view, which shows the use of the thread by the method, over time.

The method trace summary view shows the following information for each method:

- **Count:** The number of times that the method ran.
- **Total time:** The total time that the method ran for.
- **Max time:** The longest time that the method ran for.
- **Mean time:** The mean average time that the method ran for.
- **Method name:** The name of the method.

The method trace summary view is also displayed in the Profiling perspective.

The method trace timeline view is a visual representation of when the method was used on the displayed thread or threads. Color identifies the threads. Select threads from the method trace data tree to display them in this view. You can select threads from multiple methods; move the mouse over the lines to see the name of the associated method, and the exact time.



The Java Health Center Method trace perspective

- Method tracing is not enabled by default, you must first select the methods to trace
- Enable method tracing when you start your application
 - You cannot enable tracing at run time
- Use the Profiling perspective to identify methods that you want to trace
- Stop your application, add a `-Xtrace` parameter for those methods to the command line, then restart the application
 - Health Center can generate the trace parameters from the Profiling view, select **Generate method trace parameters > Copy parameters to clipboard**
 - Add the method trace parameters from the clipboard to the Java arguments
`-Xtrace:maximal=mt,methods={"mypackage.util.constructString()"}`

© Copyright IBM Corporation 2016

Figure 10-32. The Java Health Center Method trace perspective

WA8152.2

Notes:

You enable method tracing when you start your application; you cannot enable tracing at run time. To identify methods that you want to trace, run Health Center and monitor your application for a while. Use the Profiling view to examine the methods. For any methods that you want to get accurate timings for, stop your application, add a `-Xtrace` parameter for those methods to the command line, then restart the application. Health Center can generate the required trace parameters for you, from the Profiling view.

Unit summary

Having completed this unit, you should be able to:

- Describe areas of performance bottlenecks for applications
- Use the IBM Java Health Center to investigate memory issues in applications
- Use IBM Java Health Center to profile application execution
- Use IBM Java Health Center to examine application thread information
- Use IBM Java Health Center to examine application lock information

© Copyright IBM Corporation 2016

Figure 10-33. Unit summary

WA8152.2

Notes:

Checkpoint questions

1. True or False: The Health Center can be used to analyze application runtime behavior such as object creation and garbage collection behavior.
2. True or False: You can use the profiling perspective to decide what methods are the best candidates for optimization.
3. True or False: Accurate timings from method tracing are immediately available for all methods that are profiled.

© Copyright IBM Corporation 2016

Figure 10-34. Checkpoint questions

WA8152.2

Notes:

Write your answers here:

- 1.
- 2.
- 3.



Checkpoint answers

1. True.
2. True.
3. False. You must enable specific method tracing and restart the JVM before method trace information for a method is available.

© Copyright IBM Corporation 2016

Figure 10-35. Checkpoint answers

WA8152.2

Notes:

Exercise 10



Application profiling with Java Health Center

© Copyright IBM Corporation 2016

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

9.1

Figure 10-36. Exercise 10

WA8152.2

Notes:

Exercise objectives

After completing this exercise, you should be able to:

- Use the Java Health Center to find application methods with the longest execution time
- Use the Java Health Center to examine garbage collection behavior
- Use the Java Health Center to find the location of large object allocations
- Use the Java Health Center to analyze calls to System.gc()
- Use Java tracing to find the location of calls to System.gc()
- Use the Java Health Center to examine locking behavior

© Copyright IBM Corporation 2016

Figure 10-37. Exercise objectives

WA8152.2

Notes:

Unit 11. WebSphere clusters and scalability

What this unit is about

This unit describes WebSphere Application Server clusters and scalability and Intelligent Management.

What you should be able to do

After completing this unit, you should be able to:

- Describe the static clusters and dynamic clusters
- Compare the benefits of vertical and horizontal scaling
- Explain the use of multiple processors for scalability
- Describe queuing considerations for vertical and horizontal scaling
- Describe the main features of Intelligent Management

How you will check your progress

- Checkpoint questions
- Lab Exercise

References

WebSphere Application Server V8.5 Information Center,

Clusters and workload management

http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/index.jsp?topic=%2Fcom.ibm.websphere.nd.doc%2Fae%2Fcrun_srvgrp.html

Intelligent Management overview

http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/index.jsp?topic=%2Fcom.ibm.websphere.wve.doc%2Fae%2Fcwve_xdovrvw.html

Unit objectives

After completing this unit, you should be able to:

- Describe the static clusters and dynamic clusters
- Compare the benefits of vertical and horizontal scaling
- Explain the use of multiple processors for scalability
- Describe queuing considerations for vertical and horizontal scaling
- Describe the main features of Intelligent Management

© Copyright IBM Corporation 2016

Figure 11-1. Unit objectives

WA8152.2

Notes:



Topics

- Application server clusters
- Intelligent Management

© Copyright IBM Corporation 2016

Figure 11-2. Topics

WA8152.2

Notes:

11.1. Application server clusters

Application server clusters



© Copyright IBM Corporation 2016

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

9.1

Figure 11-3. Application server clusters

WA8152.2

Notes:

Application server clusters

- Static cluster
 - A static cluster is a group of application servers that participates in workload management
 - Workload management includes load balancing, scalability, and high availability
 - Clusters enable enterprise applications to scale beyond the amount of throughput that can be achieved with a single application server
- Dynamic cluster
 - A dynamic cluster is a fundamental building block of the dynamic operations environment
 - You must configure dynamic clusters to get Intelligent Management functionality, such as high availability and service policies
 - The application placement controller starts and stops dynamic cluster members when running in supervised or automatic mode
 - Dynamic clusters are more scalable than static clusters

© Copyright IBM Corporation 2016

Figure 11-4. Application server clusters

WA8152.2

Notes:

Clusters are sets of servers that are managed together and participate in workload management. They run the same Java EE applications for load distribution. Clusters enable enterprise applications to scale beyond the amount of throughput that can be achieved with a single application server.

Application servers that are members of a cluster can be on the same or different host machines. A cell can have no clusters, one cluster, or multiple clusters.

Using cluster configurations (1 of 3)

- The capability to spread workload among application servers by using clustering can be a valuable asset in configuring highly scalable production environments
 - Often true when the application is experiencing bottlenecks that are preventing full CPU utilization of Symmetric Multiprocessing (SMP) servers
- When adjusting the WebSphere system queues in clustered configurations, remember that when servers are added to a cluster, the database server downstream receives more load

© Copyright IBM Corporation 2016

Figure 11-5. Using cluster configurations (1 of 3)

WA8152.2

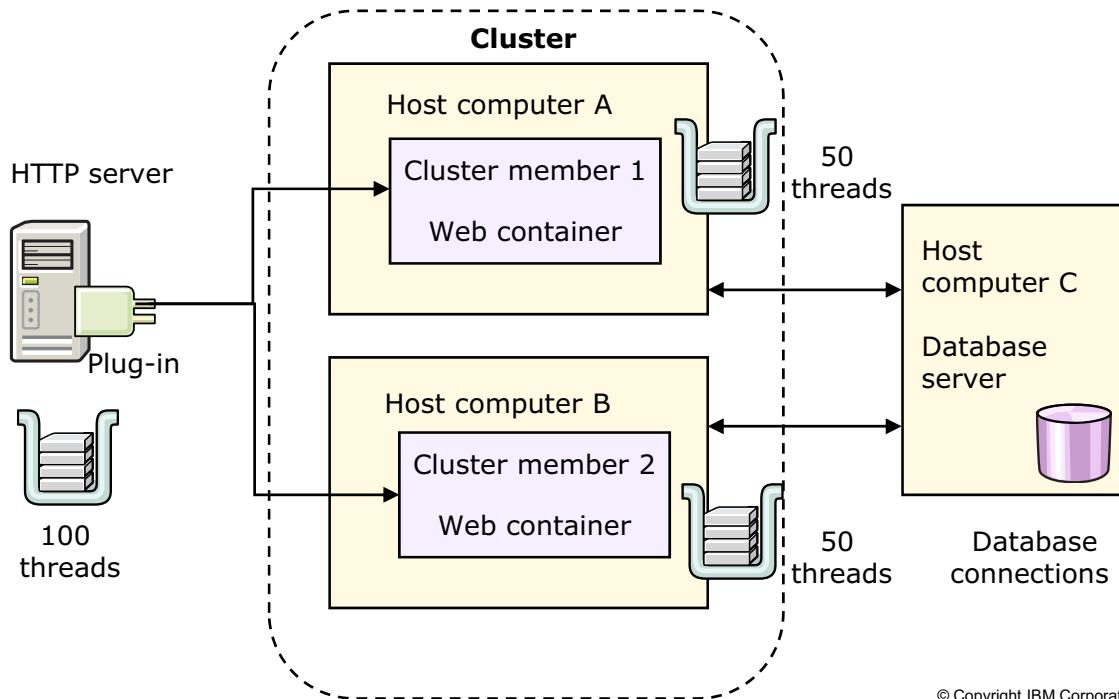
Notes:

The capability to spread workload among application servers by using clustering can be a valuable asset in configuring highly scalable production environments. This capability is especially valuable when the application is experiencing bottlenecks that are preventing full CPU utilization of Symmetric Multiprocessing (SMP) servers.

When adjusting the WebSphere system queues in clustered configurations, remember that when a server is added to a cluster, the server downstream receives twice the load.

Using cluster configurations (2 of 3)

- Two web containers within a cluster are located between a web server and a data source



© Copyright IBM Corporation 2016

Figure 11-6. Using cluster configurations (2 of 3)

WA8152.2

Notes:

If the **LoadBalanceWeight** is the same for each cluster member, the cluster members share equal proportions of the request that are routed through the web server. Therefore, if there are two cluster members, the web server queue settings can be doubled to ensure that ample work is distributed to each web container.

Using cluster configurations (3 of 3)

- Consider the following queue configurations:
 - Web server queue settings can be doubled to ensure that ample work is distributed to each web container
 - Web container thread pools can be reduced to avoid saturating a system resource such as CPU or another resource that the servlets are using
- The data source can be reduced to avoid saturating the database server
- Java heap parameters can be reduced for each instance of the application server
 - For versions of the JVM shipped with WebSphere Application Server, it is crucial that the heap from all JVMs remains in physical memory
 - If a cluster of four JVMs is running on a system, enough physical memory must be available for all four heaps

© Copyright IBM Corporation 2016

Figure 11-7. Using cluster configurations (3 of 3)

WA8152.2

Notes:

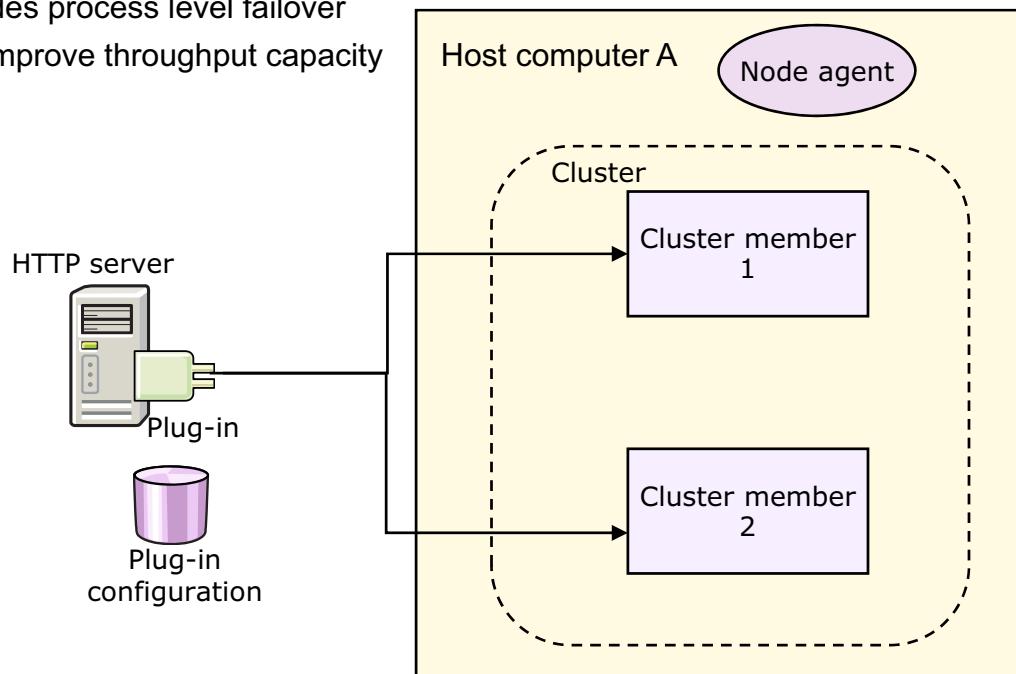
Two web containers within a cluster are located between a web server and a data source. It is assumed that the web server, web container, and data source (but not the database) are all running on a single SMP server. Given these constraints, the following queue considerations need to be made:

- Web server queue settings can be doubled to ensure that ample work is distributed to each web container.
- Web container thread pools can be reduced to avoid saturating a system resource such as CPU or another resource that the servlets are using.
- The data source can be reduced to avoid saturating the database server.
- Java heap parameters can be reduced for each instance of the application server.

For versions of the JVM shipped with WebSphere Application Server, it is crucial that the heap from all JVMs remains in physical memory. Therefore, if a cluster of four JVMs is running on a system, enough physical memory must be available for all four heaps.

Clustering: Vertical scaling

- Vertical scaling
 - Provides process level failover
 - Can improve throughput capacity



© Copyright IBM Corporation 2016

Figure 11-8. Clustering: Vertical scaling

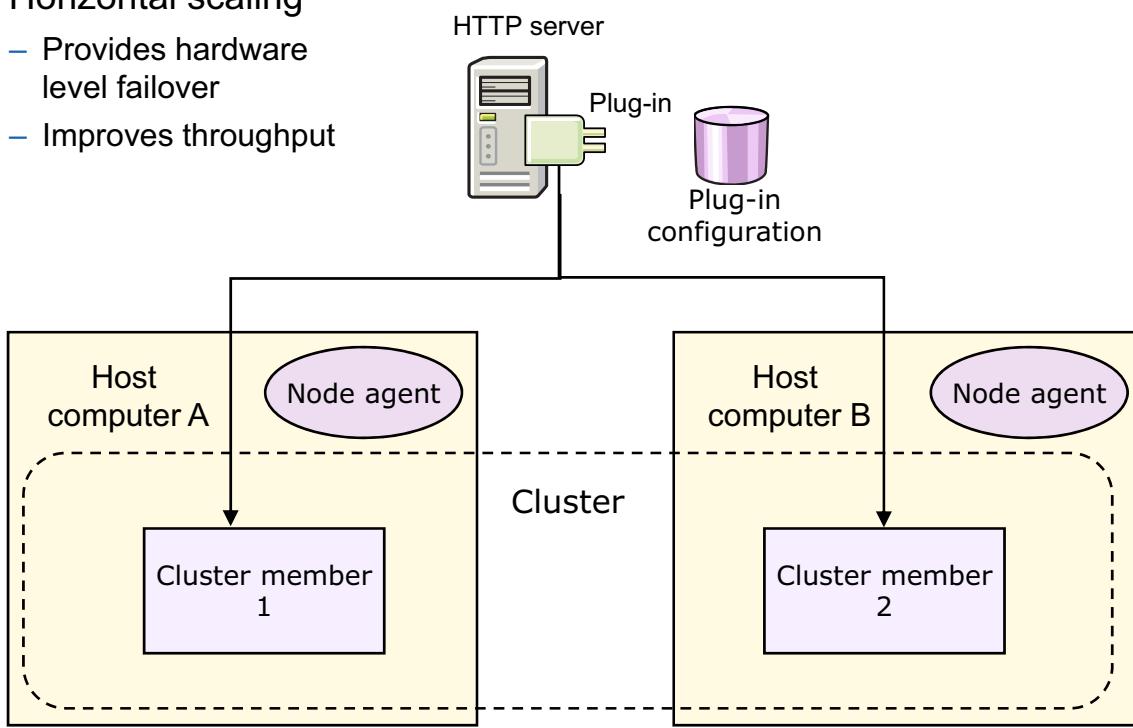
WA8152.2

Notes:

Cluster members can be scaled vertically. Vertical scaling means putting multiple cluster members on the same node (or computer). Although vertical scaling might not increase throughput (assuming that one cluster member uses the computer fully), it does provide process level failover, and might allow more efficient use of the processing power.

Clustering: Horizontal scaling

- Horizontal scaling
 - Provides hardware level failover
 - Improves throughput



© Copyright IBM Corporation 2016

Figure 11-9. Clustering: Horizontal scaling

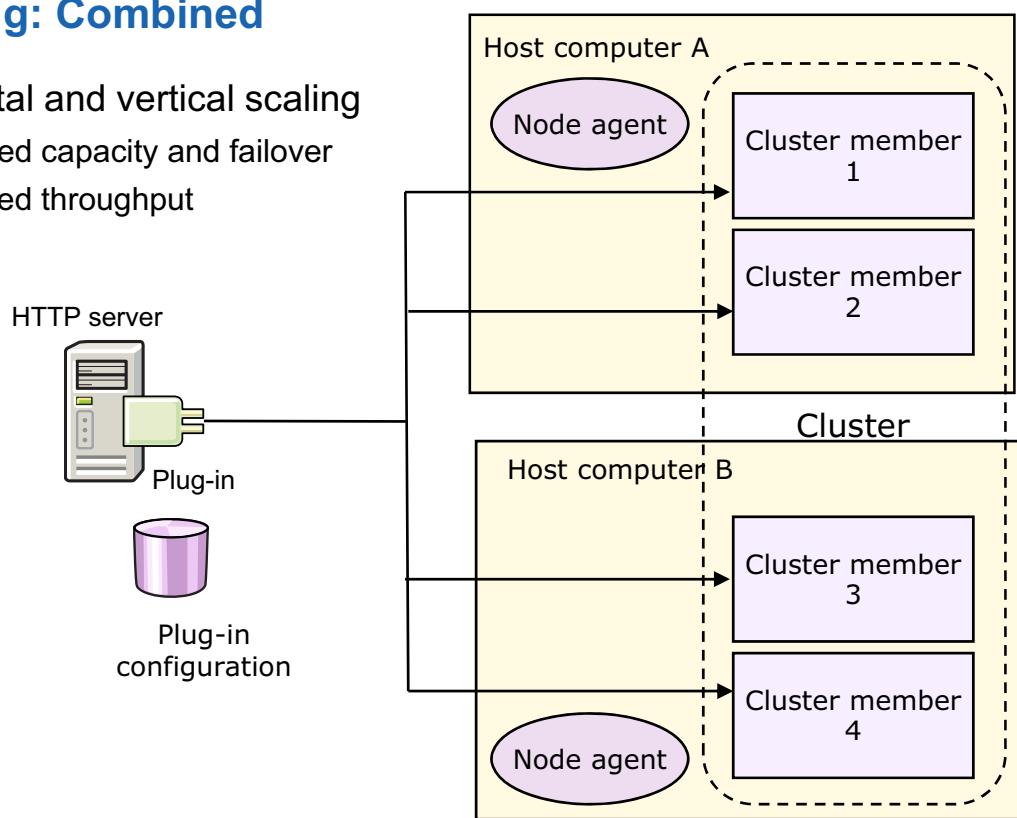
WA8152.2

Notes:

Cluster members can be scaled horizontally. Horizontal scaling means putting cluster members on different computers. This type of scaling can increase throughput, and also provide system level failover.

Clustering: Combined

- Horizontal and vertical scaling
 - Improved capacity and failover
 - Improved throughput



© Copyright IBM Corporation 2016

Figure 11-10. Clustering: Combined

WA8152.2

Notes:

Scaling both vertically and horizontally combines both system and process level failover.

Option to start application components dynamically

- You can improve system performance if you configure each cluster member to start components dynamically as needed instead of letting all components automatically start when the cluster member starts
 - Improves cluster startup time and reduces the memory footprint of the cluster members
- Starting components as they are needed is most effective if all of the applications that are deployed on the cluster are of the same type
 - Works better if all of your applications are web applications that use servlets and JavaServer Pages (JSP)
 - Works less effectively if your applications use servlets, JSPs, and Enterprise JavaBeans (EJB)

© Copyright IBM Corporation 2016

Figure 11-11. Option to start application components dynamically

WA8152.2

Notes:

When you configure cluster members such that not all of the cluster member components start when the cluster or a specific cluster member is started, the cluster member components are dynamically started as they are needed. For example, if an application module starts that requires a specific server component, that component is dynamically started.

Scaling with multiple processors

- Most systems today support multiple processors
 - Known as SMP, symmetric multiprocessing
- More processor means more computing power
- In theory, more computing power means that an application server can process more requests
- Verify that your system kernel supports SMP
 - For UNIX and Linux, use the `uname -a` command
 - Look for the SMP

```
[root@hostA ~]# uname -a
Linux hostA 2.6.32-71.el6.x86_64 #1 SMP Wed Sep 1 01:33:01
EDT 2010 x86_64 x86_64 x86_64 GNU/Linux
```

© Copyright IBM Corporation 2016

Figure 11-12. Scaling with multiple processors

WA8152.2

Notes:

Clustering: Multiple processors

- A single JVM can drive multiple processors
 - There are examples where a single JVM was able to drive a 64-way machine
 - Other examples where the throughput fell off after only four processors
- The optimal number of processors depends on the environment and can vary greatly
 - To determine what number of processors works best in your environment, requires performance testing and analysis

© Copyright IBM Corporation 2016

Figure 11-13. Clustering: Multiple processors

WA8152.2

Notes:

The correct numbers of CPUs to use within a single machine depends on the environment, including the applications that run within the application processors. Some helpful advice can be found at the following location:

http://www.ibm.com/developerworks/websphere/techjournal/0506_col_alcott/0506_col_alcott.html

Although the article is rather old, the same principals hold today.

Clustering best practices (1 of 2)

- The best values for your environment depend on your configurations and the applications you are running
- The optimal number of physical processors for a set of JVMs is about four
 - For example, on a 12-way box: three JVMs can each use four processors
- Avoid large numbers of JVMs in a single logical partition
 - Each JVM might require a significant amount of CPU capacity (over one physical processor of CPU capacity)
 - Large numbers of JVMs that require significant amounts of CPU capacity drive the number of processors per logical partition beyond the optimal number of processors per JVM
- Consider horizontal scaling instead of vertical scaling in cases where large numbers of physical processors are required

© Copyright IBM Corporation 2016

Figure 11-14. Clustering best practices (1 of 2)

WA8152.2

Notes:

The optimal number of physical processors for a JVM is around 4. Four physical processors are also optimal for multiple JVMs running in a single “logical partition”.

Running many JVMs in a single “logical partition” works, and a level of scaling is achieved, but is not optimal. If the number of physical processors that are required by a “logical partition” hosting multiple JVMs is significantly higher than 4 (16 or more), JVM operation is not optimal.

To avoid “logical partitions” with high numbers of physical CPUs, consider horizontal scaling (adding cluster members to separate “logical partitions”) instead of vertical scaling (adding CPUs to existing “logical partitions”).

When running JVMs on “logical partitions” with high numbers of physical CPUs, consider tuning the number of garbage collection threads with the “`-gcthreads`” parameter for the IBM JVM.

Clustering best practices (2 of 2)

- Horizontal scaling
 - Adding new logical partitions with new physical CPUs
 - Scales linearly in most cases
 - Note: the new logical partitions host new WebSphere instances
- Vertical scaling
 - Adding WebSphere instances and new physical CPUs in existing logical partitions
 - Does not scale linearly at high numbers of physical processors (over 8)
- Consider the use of helper threads for garbage collection if the number of logical processors that are assigned to the host logical partition is high (over 8)
 - Use the generic JVM argument `-gcthreads`
 - The optimal number of garbage collection threads is about six

© Copyright IBM Corporation 2016

Figure 11-15. Clustering best practices (2 of 2)

WA8152.2

Notes:

Horizontal scaling tends to scale linearly, whereas vertical scaling might not because vertical scaling is more dependent on sharing resources with the other running instances.

11.2. Intelligent Management

Intelligent Management



© Copyright IBM Corporation 2016

Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

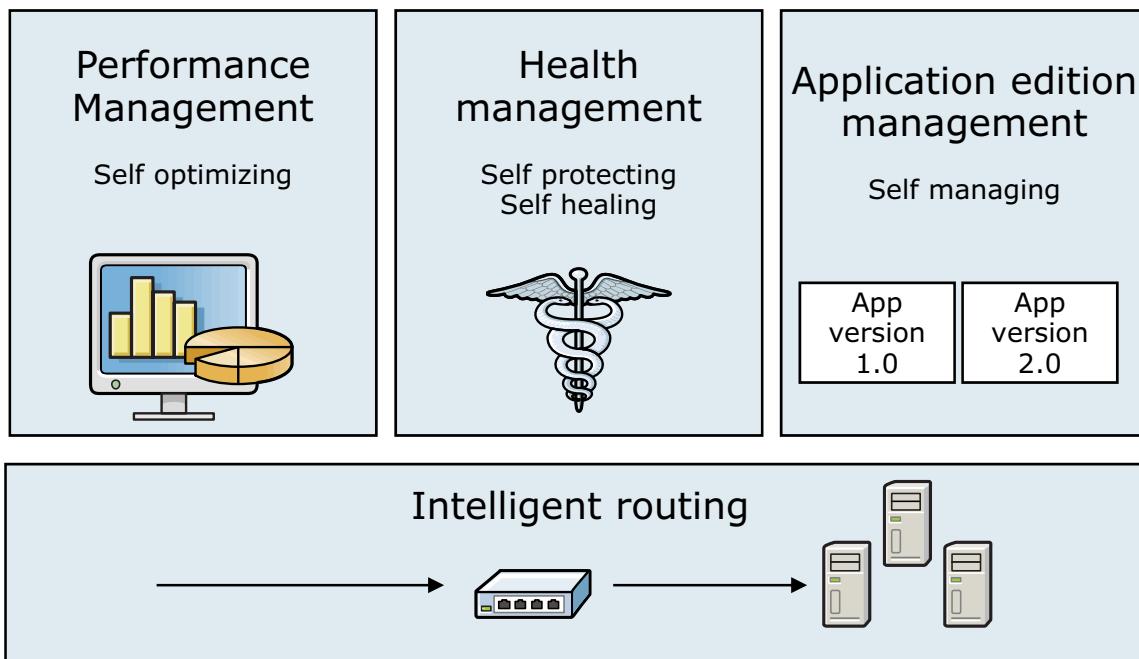
9.1

Figure 11-16. Intelligent Management

WA8152.2

Notes:

Intelligent management



© Copyright IBM Corporation 2016

Figure 11-17. Intelligent management

WA8152.2

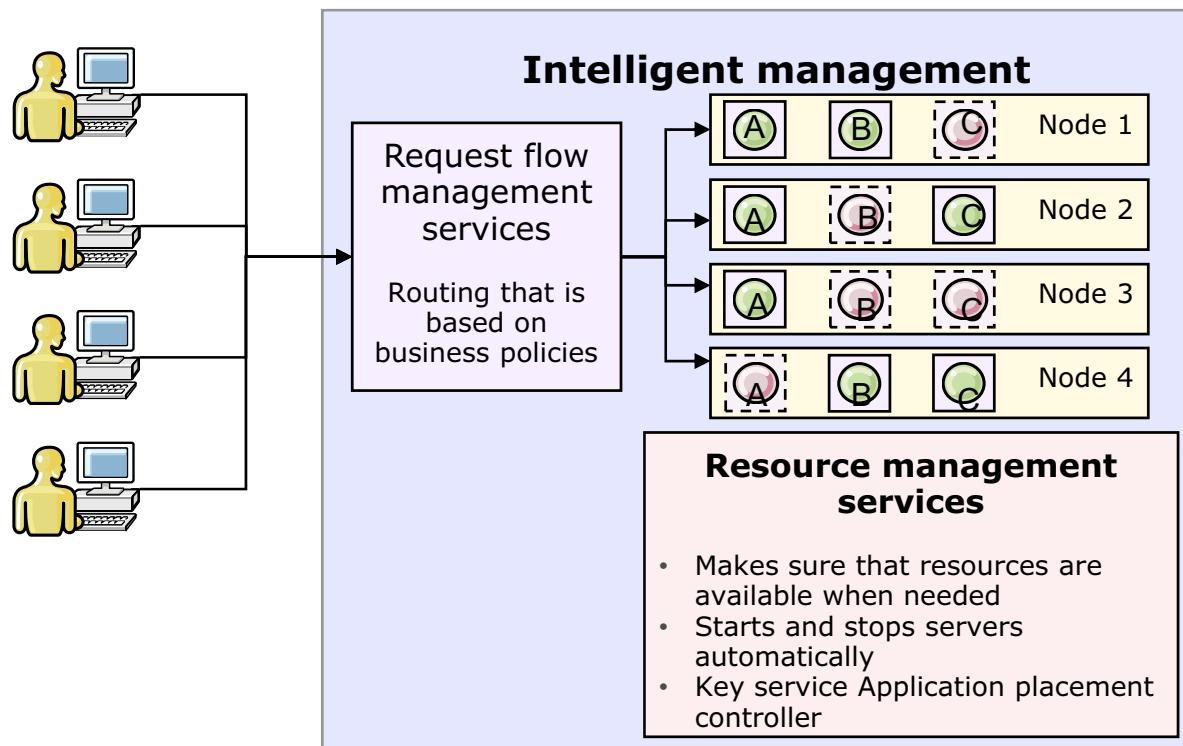
Notes:

There are four main topics on intelligent management.

- Overview of Performance Management capabilities:
 - Associate service policies with your applications and have WebSphere efficiently manage requests to achieve these goals
 - Decrease administrative work that is required to monitor and diagnose performance issues
 - Minimize the number of JVMs and virtual machines that are running to reduce processing that lightly used or idle JVMs or virtual machines incur
 - Protect your middleware infrastructure against overload
- Overview of health management capabilities:
 - Automatically detect and handle application health problems without requiring administrator time, expertise, or intervention
 - Intelligently handle health issues in a way that maintains continuous availability
 - Configure applications differently if you want to

- Approve automatic actions before actions are taken
- Set policies that are based on both high-level metrics and low-level metrics to detect problems before you notice that something is wrong
- Overview of application edition management capabilities:
 - Ability to run multiple versions of your applications concurrently
 - You do not incur any downtime while updating your applications
 - Verify that a new version of your application works in production before sending real traffic to it
 - Reduce your infrastructure costs and decrease potential outages
 - Easily update your operating system or WebSphere without incurring downtime
- Overview of intelligent routing capabilities:
 - A routing tier that is aware of what is happening on the application server tier and reacts; business critical applications are given priority
 - Automatic routing without needing to update configuration files
 - A highly scalable routing tier
 - Ease of management
 - Flexible policy-based routing to control if, when, and where requests are routed
 - A highly available deployment manager

Performance Management



© Copyright IBM Corporation 2016

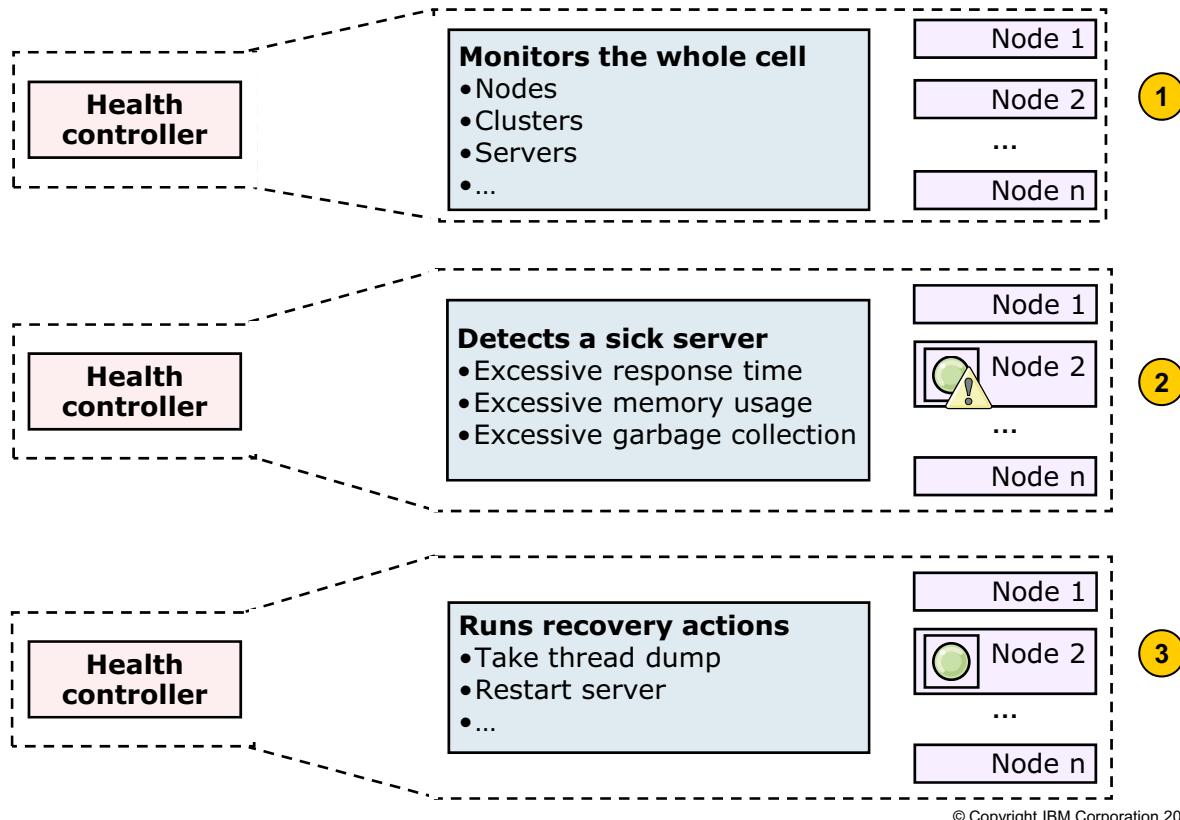
Figure 11-18. Performance Management

WA8152.2

Notes:

Performance Management provides two main categories of services. Request Flow Management services intelligently route requests to appropriate resources based on business policies you define. Resource Management services make sure that the right servers are available to meet the demand. One of the key services that Intelligent Management uses to implement Performance Management is the Application Placement Controller.

Health management



© Copyright IBM Corporation 2016

Figure 11-19. Health management

WA8152.2

Notes:

The Health Management feature of Intelligent Management allows you to monitor your environment and automatically recover when unhealthy servers are found. The autonomic manager responsible for monitoring and recovering servers is called the Health Controller. You define what unhealthy conditions to watch for, and what actions to take with Health Policies. The Health Controller works together with the other autonomic managers to ensure that actions are taken intelligently, so service is not interrupted, and service policies are met.

Intelligent routing

- A routing tier that is aware of what is happening on the application server tier
 - Knows which cluster members are currently started
 - Knows application server usage, performance, and other statistics
 - Understands service policies
 - Routes work to the application server that can do it best
 - Can route to multiple application editions
 - Provides preference for higher priority requests (some restrictions)
 - Provides processor and memory overload protection (some restrictions)
- Integrates with Performance Management, health management, and application edition management

© Copyright IBM Corporation 2016

Figure 11-20. Intelligent routing

WA8152.2

Notes:

Intelligent routing can improve the quality of service by making the best use of available resources, and ensuring that priority is given to business-critical applications and users. Requests to applications are prioritized and routed based on administrator-defined rules. Intelligent routers retrieve information on the available servers and their capacity to serve requests. Intelligent routers work with performance management, health management, and application edition management.

Dynamic clusters (1 of 2)

- A dynamic cluster is a server cluster that uses weights and workload management to balance the workloads of its cluster members dynamically
 - Based on performance information that is collected from the cluster members
- A dynamic cluster is an application deployment target that can expand and contract as a result of the workload in your environment
 - Dynamic clusters work with autonomic managers, including the application placement controller and the dynamic workload manager to maximize the use of your computing resources

© Copyright IBM Corporation 2016

Figure 11-21. Dynamic clusters (1 of 2)

WA8152.2

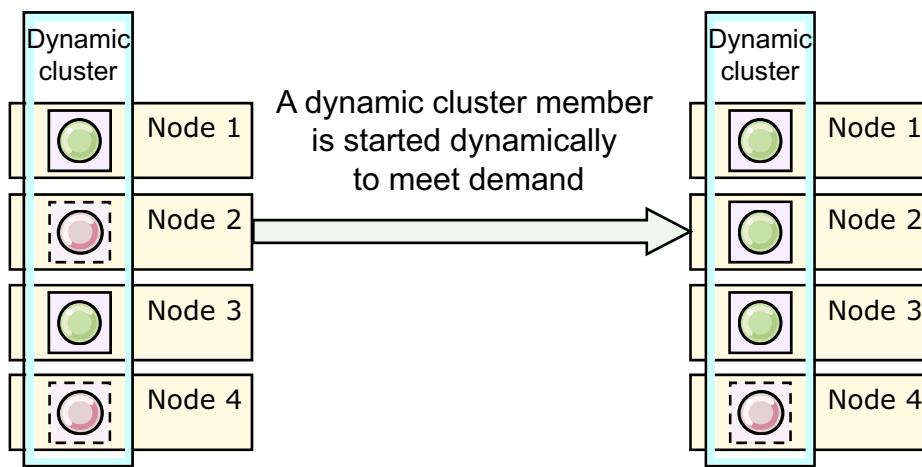
Notes:

A dynamic cluster is a server cluster that uses weights and workload management to balance the workloads of its cluster members dynamically, based on performance information that is collected from the cluster members. Dynamic clusters enable application server virtualization.

A dynamic cluster is an application deployment target that can expand and contract as a result of the workload in your environment. Dynamic clusters work with autonomic managers, including the application placement controller and the dynamic workload manager to maximize the use of your computing resources. Dynamic clusters are required for many of the Intelligent Management autonomic functions, including high availability and service policies.

Dynamic clusters (2 of 2)

- Similar to a static cluster, but number of active servers can be resized dynamically at run time
 - The number of active instances depends on business policies and current request flow
 - Managed by the Application placement controller



© Copyright IBM Corporation 2016

Figure 11-22. Dynamic clusters (2 of 2)

WA8152.2

Notes:

Dynamic clusters are static clusters under the covers, but they have extra behavioral characteristics. The dynamic part of a Dynamic Cluster is the number of instances that are started. When a Dynamic cluster is created, a set of rules is specified that determines which nodes have Dynamic Cluster Member instances that are created on them. All of the members are then statically created on those nodes when the cluster is defined. All of the members exist, but they are started and stopped dynamically.

When a new node is added to the cell, all dynamic clusters with rules that match the characteristics of the new node have a cluster member immediately created. An example of rule might be: "All nodes in the cell except the node with the Deployment Manager".

Intelligent routing makes sure that requests are routed to the currently started dynamic cluster members.

Autonomic managers and services

- Application placement controller
 - Decides when and where to start dynamic cluster members
- Autonomic request flow manager
 - Classifies incoming requests and monitors performance
- Dynamic workload management controller
 - Adjusts server weights for dynamic routing
- Health controller
 - Monitors health policies and acts to correct problems
- On-demand configuration service
 - Maintains cell topology information and keeps the other controllers informed about the environment

© Copyright IBM Corporation 2016

Figure 11-23. Autonomic managers and services

WA8152.2

Notes:

Intelligent management capabilities are delivered in a set of components that are known as autonomic managers. Autonomic managers monitor performance and health statistics through a series of sensors; they optimize system performance and manage traffic.

The application placement controller

The application placement controller is responsible for the management of the location of an application. A single application placement controller exists in the cell and is hosted in the deployment manager or in a node agent process. The application placement controller starts and stops application server instances to manage traffic. The application placement controller can dynamically address periods of intense workflow that would otherwise require the manual intervention of a system administrator.

The autonomic request flow manager

The main purpose of the autonomic request flow manager is to determine when requests are sent to the application servers. Intelligent management calculates the amount of resources that each type of request needs on each node. The ARFM uses information from the nodes, from the ODR, and from PMI statistics.

The dynamic workload controller

The dynamic workload controller dynamically adjusts server weights to even out and minimize response times across the cluster. There is one dynamic workload controller per cluster. The dynamic workload controller maintains a list of active server instances for each dynamic cluster, and assigns each server instance a routing weight according to observed performance trends. Requests are then routed to candidate server instances to balance workloads on the nodes within a cluster.

The health controller

The health controller enforces configured health policies. When policy conditions are violated, the health controller carries out the defined actions for the health policy.

The on-demand configuration service

The on-demand configuration service maintains cell topology information and keeps the other controllers aware of the environment. It tracks updates in cell topology and state, including the following changes:

- Applications that are installed and removed
- Servers started and stopped
- Nodes that added and removed
- Classification updates

The on-demand configuration component allows the intelligent routers to understand the environment.

Unit summary

Having completed this unit, you should be able to:

- Describe the static clusters and dynamic clusters
- Compare the benefits of vertical and horizontal scaling
- Explain the use of multiple processors for scalability
- Describe queuing considerations for vertical and horizontal scaling
- Describe the main features of Intelligent Management

© Copyright IBM Corporation 2016

Figure 11-24. Unit summary

WA8152.2

Notes:

Checkpoint questions

1. True or False: A static cluster is a group of application servers that participates in workload management.
2. True or False: Vertical scaling always increases throughput.
3. True or False: Use the generic JVM argument `-gcthreads` to specify the number of helper threads.
4. True or False: Performance Management provides two services, health and application edition management.

© Copyright IBM Corporation 2016

Figure 11-25. Checkpoint questions

WA8152.2

Notes:

Write your answers here:

- 1.
- 2.
- 3.
- 4.



Checkpoint answers

1. True
2. False: Vertical scaling might not increase throughput if system resources are exhausted.
3. True
4. False: The two services Performance Management provides are Request Flow Management and Resource Management.

© Copyright IBM Corporation 2016

Figure 11-26. Checkpoint answers

WA8152.2

Notes:

Exercise 11



Load testing an application server cluster

© Copyright IBM Corporation 2016
Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

9.1

Figure 11-27. Exercise 11

WA8152.2

Notes:



Exercise objectives

After completing this exercise, you should be able to:

- Install the DayTrader Application to a cluster
- Configure the IBM HTTP plug-in to route requests to the cluster
- Use Apache JMeter to load test the cluster
- Use the Apache server-status tool to monitor web server usage
- Monitor CPU usage on cluster member hosts
- Reconfigure cluster member weights to make better use of system resources

© Copyright IBM Corporation 2016

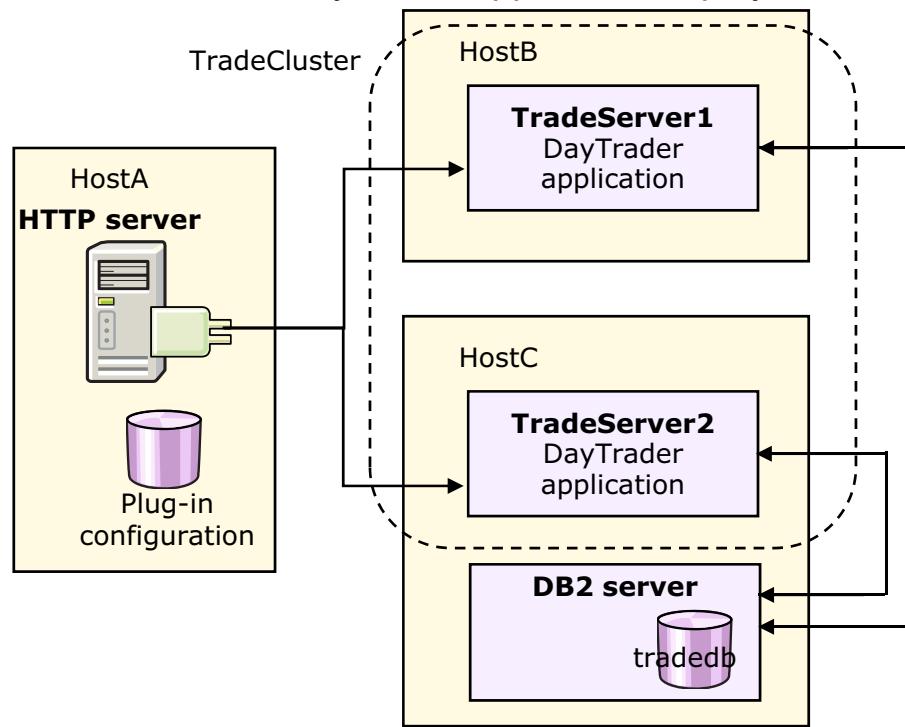
Figure 11-28. Exercise objectives

WA8152.2

Notes:

Exercise environment

- Horizontal cluster with the DayTrader application deployed



© Copyright IBM Corporation 2016

Figure 11-29. Exercise environment

WA8152.2

Notes:

Unit 12. Course summary

What this unit is about

This unit summarizes the course and provides information for future study.

What you should be able to do

After completing this unit, you should be able to:

- Explain how the course met its learning objectives
- Access the IBM Training website
- Identify other IBM Training courses that are related to this topic
- Locate appropriate resources for further study

Unit objectives

After completing this unit, you should be able to:

- Explain how the course met its learning objectives
- Access the IBM Training website
- Identify other IBM Training courses that are related to this topic
- Locate appropriate resources for further study

© Copyright IBM Corporation 2016

Figure 12-1. Unit objectives

WA8152.2

Notes:

Course learning objectives

After completing this course, you should be able to:

- Apply performance tuning methods to WebSphere Application Servers
- Perform load testing by using Apache JMeter
- Monitor application server performance by using WebSphere and the IBM Support Assistant
- Monitor and tune the JVM for optimum throughput and response time
- Monitor and tune connection pools for optimum performance
- Implement best practices for general WebSphere runtime performance
- Use the IBM Health Center tool to profile and tune Java EE applications
- Load test and monitor an application server cluster

© Copyright IBM Corporation 2016

Figure 12-2. Course learning objectives

WA8152.2

Notes:



To learn more on the subject

- IBM Training website:
www.ibm.com/training
- Web pages
 - WebSphere: <http://www.ibm.com/websphere>
 - WebSphere Support page:
<http://www.ibm.com/software/webservers/appserv/was/support>
 - IBM developerWorks: WebSphere Application Server zone
<http://www.ibm.com/developerworks/websphere/zones/was>

© Copyright IBM Corporation 2016

Figure 12-3. To learn more on the subject

WA8152.2

Notes:



References (1 of 2)

- Redbooks: www.redbooks.ibm.com
 - SG24- 8056-00 *WebSphere Application Server V8.5 Administration and Configuration Guide*
 - SG24-6392- 00 *WebSphere Application Server V6 Scalability and Performance Handbook*
 - SG24-7584-00 *IBM WebSphere Application Server V6.1 on the Solaris10 Operating System*
- developerWorks Forums
<https://www.ibm.com/developerworks/community/forums>
 - WebSphere Application Server
 - Application Performance Diagnostics
 - IBM Support Assistant forum
 - IBM Monitoring and Diagnostic Tools for Java - Health Center

© Copyright IBM Corporation 2016

Figure 12-4. References (1 of 2)

WA8152.2

Notes:



References (2 of 2)

- IBM information centers
 - WebSphere Application Server V8.5 Information Center
<http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/index.jsp>
 - IBM SDK, Java Technology Edition, Version 6
http://publib.boulder.ibm.com/infocenter/javasdk/v6r0/topic/com.ibm.java.doc.60_26/homepage/plugin-homepage-java626.html
 - IBM SDK, Java Technology Edition, Version 7
<http://pic.dhe.ibm.com/infocenter/java7sdk/v7r0/index.jsp>
- IBM Developer Kit and Runtime Environment, Java Technology Edition, Version 6 Diagnostics Guide
 - <http://download.boulder.ibm.com/ibmdl/pub/software/dw/jdk/diagnosis/diag60.pdf>
- IBM Support Assistant Team Server
 - <http://www.ibm.com/software/support/isa/teamserver.html>

© Copyright IBM Corporation 2016

Figure 12-5. References (2 of 2)

WA8152.2

Notes:



Unit summary

Having completed this unit, you should be able to:

- Explain how the course met its learning objectives
- Access the IBM Training website
- Identify other IBM Training courses that are related to this topic
- Locate appropriate resources for further study

© Copyright IBM Corporation 2016

Figure 12-6. Unit summary

WA8152.2

Notes:

Appendix A. List of abbreviations

A

- AB** ApacheBench
AFS Andrew File System
AIX Advanced IBM UNIX
AMI asynchronous message interface
Ant Another Neat Tool
AOT ahead-of-time
APAR authorized program analysis report
APC application placement controller
API application programming interface
ARM Application Response Measurement
ASCII American Standard Code for Information Interchange
AST Application Server Toolkit

B

- BSF** Bean Scripting Framework

C

- CA** certificate authority
CCI Common Client Interface
CIM Centralized Installation Management
CIP custom installation package
CMP container-managed persistence
CMS Certificate Management System
CMT Configuration Migration Tool
CORBA Common Object Request Broker Architecture
CP caching proxy
CPU central processing unit
CSlv2 Common Secure Interoperability Protocol Version 2
CW conditioned wait

D

- DB** database
DBA database administrator
DC domain controller
DCS Distribution and Consistency Services
DD deployment descriptor

DHCP Dynamic Host Configuration Protocol

DLL Dynamic Link Library

DMgr or **dmgr** deployment manager

DMZ demilitarized zone

DN distinguished name

DNS Domain Name System

DP diagnostic provider

DRS data replication service

DTD document type definition

DTFJ Diagnostic Tooling Framework for Java

E

EAR enterprise archive

EE Enterprise Edition

EIS enterprise information system

EJB Enterprise JavaBean

EJS Enterprise Java Services

EL Expression Language

ENC Enterprise Naming Context

ESB Enterprise service bus

ESI Edge Side Include

F

FFDC first-failure data capture

FIPS Federal Information Processing Standard

FQDN fully qualified domain name

FTP File transfer protocol

G

GA generally available

GB gigabyte

GC garbage collection

GCD greatest common divisor

GCMV Garbage Collection and Memory Visualizer

GIF Graphics Interchange Format

GMT Greenwich Mean Time

GPF general protection fault

GTK GNU GUI Tool Kit

GSS Generic Security Services

GUI graphical user interface

H

HA high availability or highly available
HACMP High Availability Cluster Multi-Processing
HAM high availability manager
HFS Hierarchical File System
HP Hewlett Packard
HPEL High Performance Extensible Logging
HP-UX Hewlett Packard UNIX
HTML Hypertext Markup Language
HTTP Hypertext Transfer Protocol
HTTPD HTTP Daemon
HTTPS HTTP over SSL

I

IBM International Business Machines Corporation
IDDE Interactive Diagnostic Data Explorer
IE Internet Explorer
IIM IBM Installation Manager
IIP Internet Inter-ORB Protocol
IMS Information Management System
I/O input/output
IP Internet Protocol
IPSEC IP Security
ISC Integrated Solutions Console
ISMP InstallShield MultiPlatform
ISV independent software vendor
IT information technology
ITDS IBM Tivoli Directory Server
IVT Installation Verification Tool

J

J2C Java 2 Connector
J2EE Java 2 Enterprise Edition
J2SE Java 2 Platform Standard Edition
JAAS Java Authentication and Authorization Service
JACC Java Authorization Contract for Containers
JAF Java Activation Framework
JAR Java archive
JASPIC Java Authentication Service Provider Interface for Containers
Java EE Java Platform, Enterprise Edition

JAXB Java Architecture for XML Binding

JAXP Java API for XML Processing

JAXR Java API for XML Registries

JAX-RPC Java API for XML Remote Procedure Calls

JAX-RS Java API for XML-based Remote Procedure Calls

JAX-WS Java API for XML Web Services

JCA Java EE Connector Architecture

JCE Java Cryptology Extension

JCL Java class library

JDBC Java Database Connectivity

JDE JD Edwards

JDK Java Development Kit

JHC Java Health Center

JIT just-in-time

JMS Java Message Service

JMX Java Management Extensions

JNI Java Native Interface

JNDI Java Naming and Directory Interface

JPA Java Persistence API

JPG Graphics file type or extension (lossy compressed 24-bit color image storage format developed by the Joint Photographic Experts Group)

JRE Java Runtime Environment

JSF JavaServer Faces

JSP JavaServer Pages

JSR Java Specification Request

JSSE Java Secure Sockets Extension

JSTL JavaScript Tag Library

JTA Java Transaction API

JVM Java virtual machine

JVMPPI Java Virtual Machine Profiler Interface

JVMTI Java Virtual Machine Tool Interface

K

KB knowledge base

L

LAN Local area network

LB load balancer

LDAP Lightweight Directory Access Protocol

LOA large object area

LSD location service daemon

LTPA Lightweight Third Party Authentication

M

MAC message authentication code or media access control

MAT Memory Analyzer tool

MB megabyte

MDB message-driven bean

ME messaging engine

MQ Message Queue

MQI Message Queue Interface

MVS Multiple Virtual System

N

NAS network attached storage

NAT network address translation

NFS network file system

NIC network interface card

NIM Network Installment Management

NTP Network Time Protocol

NUMA non-uniform memory access

O

OAM object authority manager

ODC on demand configuration

ODR on demand router

OLTP online transaction processing

OMG Object Management Group

OOM out-of-memory

ORB Object Request Broker

OS operating system

OSGi Open Service Gateway initiative

OVF Open Virtualization Format

P

PAR archive Index File

PCT Plug-ins Configuration Tool

PD problem determination

PFBC properties file based configuration

PGP Pretty Good Privacy

PHD Portable Heap Dump
PHP personal home page
PID process identifier
PKI public key infrastructure
PMAT Pattern Modeling and Analysis Tool
PME programming model extensions
PMI Performance Monitoring Infrastructure
PMR problem management record
PMT Program Management Tool
POJO plain old Java object
PWB PlantsByWebSphere

Q

QA quality assurance
QoS quality of service

R

RA registration authority
RACF Resource Access Control Facility
RAM random access memory
RAR resource archive
RAS reliability, availability, and serviceability
RC return code
RDBMS relational database management system
RDN relative distinguished name
REST Representational State Transfer
RM request metrics
RMI Remote Method Invocation
RMI/IOP Remote Method Invocation Over Internet Inter-ORB Protocol
RRA Relational Resource Adapter
RUP Rational Unified Process
RXA Remote Execution and Access

S

SAAJ SOAP with Attachments API for Java
SAM Security Access Manager
SAR SIP application resource
SAS Secure Association Service
SBDT Smart Business Development and Test
SCA Service Component Architecture

SCADA supervisory control and data acquisition
SCM source code management
SDO Service Data Objects
SDK software development kit
SDLC systems development lifecycle
SFSB stateful session bean
SIB or SIBus service integration bus
SIBDH Service Integration Bus Destination Handle
SIP Session Initiation Protocol
SMTP Simple Mail Transfer Protocol
SOA service-oriented architecture
SOA small object area
SOAP A lightweight, XML-based protocol for exchanging information in a decentralized, distributed environment. Usage note: SOAP is not an acronym; it is a word in itself (formerly an acronym for Simple Object Access Protocol)
SPI service provider interface
SPNEGO Simple and Protected GSS-API Negotiation Mechanism
SQL Structured Query Language
SSB stateful session bean
SSL Secure Sockets Layer
SSO single sign-on
StAX Streaming API for XML
SUSE Software und System Entwicklung (German: Software and Systems Development)
SVC supervisor call
SVG Scalable Vector Graphics
SWAM Simple WebSphere Authentication Mechanism

T

TAM Tivoli Access Manager
TCP Transmission Control Protocol
TCP/IP Transmission Control Protocol/Internet Protocol
TGC trace garbage collector
TKCARS Toolkit for Custom and Reusable Solution Information
TLS Transport Layer Security
TMDA Thread and Monitor Dump Analyzer
TP Trade Processor (application in lab exercises)

U

UDDI Universal Description, Discovery, and Integration
UNC Universal Naming Convention

UNIX Uniplexed Information and Computing System

UOW unit of work

URI Uniform Resource Identifier

URL Uniform Resource Locator

UTC Universal Test Client

UUID Universally Unique Identifier

V

VM virtual machine

VMM virtual member manager

VPN virtual private network

W

WAIT Whole-system Analysis of Idle Time

WAR web archive

WCT WebSphere Customization Toolbox

WLM workload management

WPM WebSphere Platform Messaging

WPS WebSphere Process Server

WS web services

WS-AT web services atomic transaction

WS-BA web services business activity

WS-COOR web services coordination

WSDL Web Services Description Language

WS-I Web Services Interoperability

WSIF Web Services Invocation Framework

X

XA Extended Architecture

XCT cross-component trace

XML Extensible Markup Language

XTP extreme transaction processing

Z

z/OS zSeries operating system

ZMMT z/OS Migration Management Tool

ZPMT z/OS Profile Management Tool

IBM
®