

Performance



Performance Enhancements to WebSphere V4.0



- Performance Enhancement Overview
- Dynamic Caching of Servlets/JSPs
- Performance Monitoring Infrastructure (PMI)
- Performance Tuner



- ▶ A number of performance enhancements for increasing performance and monitoring and managing performance have been added for WebSphere V4.0.
- ▶ Dynamic Caching of Servlets and JSPs increases performance by caching the output of servlet and JSP request.
- ▶ Performance Monitoring Infrastructure, or PMI, is a framework built into WebSphere for monitoring performance of different modules in WebSphere.
- ▶ The Performance Tuner brings together in one place many of the different individual performance settings through a wizard format.

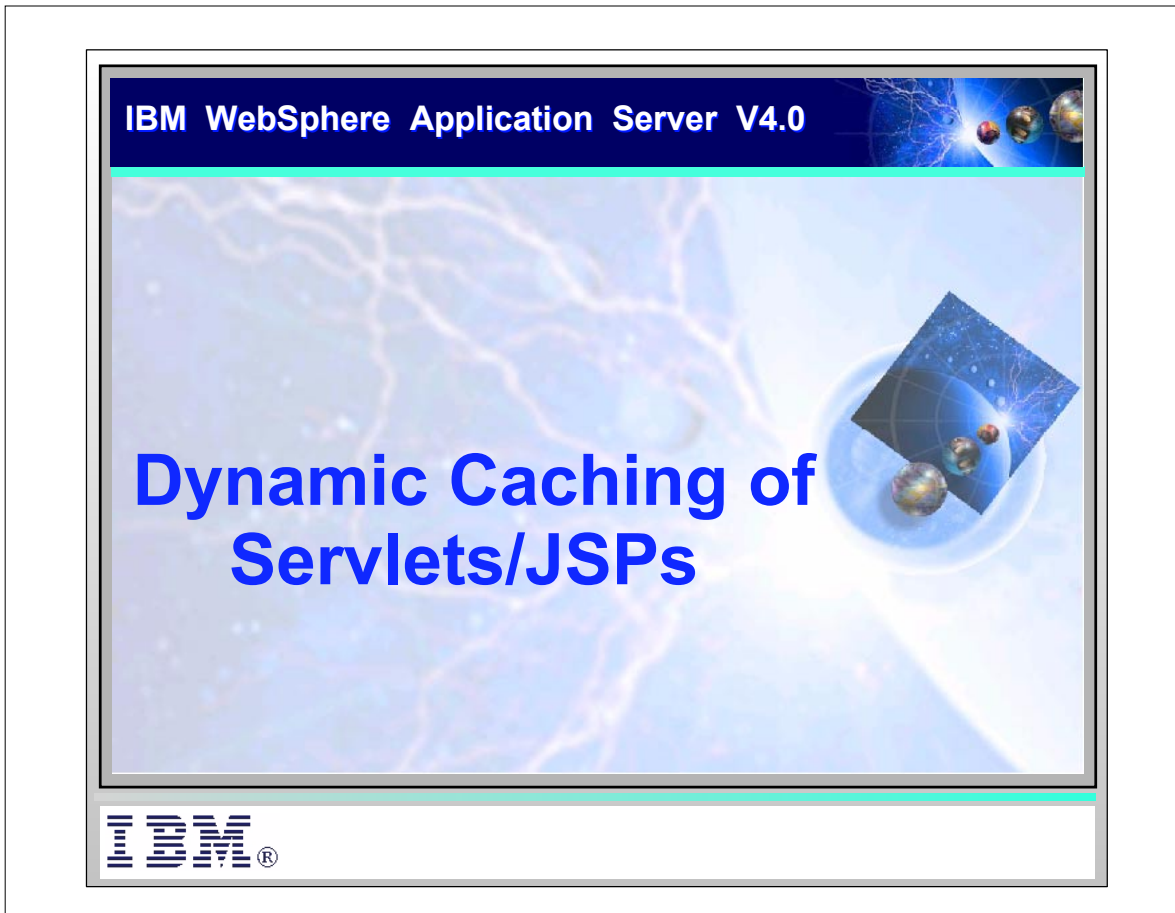
Performance Enhancement Overview



- JDK 1.3
- HTTP Transport
- Built-in Web Server
- Enhanced Scalability of large SMP systems
- Application Server and Admin Server Startup
- JSPs equal Servlets
- HTTP Session in a Cluster
- ORB Performance



- ▶ Included with Dynamic Caching of servlets and JSPs, PMI, and the Performance Tuner, there are other performance improvements in WebSphere V4.0.
- ▶ JDK 1.3
 - ▶ See Java Benchmarks (volano, specjbb).
 - ▶ Improved JIT, Garbage Collection, and memory management.
- ▶ HTTP Transport
 - ▶ General rewrite for performance and robustness
 - ▶ Connection pooling
- ▶ Built-in Web Server
 - ▶ For some configurations, e.g., DMZ using Reverse HTTP Proxy
- ▶ Enhanced Scalability on Large SMP systems
 - ▶ Removed lock contention in several areas of App Server
- ▶ Application Server and Admin Server Startup
 - ▶ AEs starts from local files
 - ▶ Tuned to handle 1000 EJBs
- ▶ JSPs
 - ▶ JSPs have been tuned to perform at the same level as comparable servlets
 - ▶ For example, a servlet using PrintWriter and Session versus a JSP
- ▶ HTTP Session in a Cluster
 - ▶ Supports Affinity without Persistence (high performance, lowered QOS)
 - ▶ Supports Batch Updates when using Persistence
- ▶ ORB Performance
 - ▶ Handling the marshaling of complex data types like Vectors, Hashtables



- ▶ Dynamic Caching of Servlets/JSPs was actually available in WebSphere 3.5.3.
- ▶ This feature for improves performance and response time of requests, caching the output from previous calls to the server.
- ▶ The caching is based on a very simple concept, which is easy to understand and implement.

Dynamic Caching Overview



- Output of JSP and Servlet requests can be cached
- Significantly reduce load on application server
- Most effective for standard output (mutual fund prices) or personalized information that is fairly static (profiles)
- Cache setup and list of entries located in XML files in `WAS_HOME\properties`
 - ▶ Ability to enable cache per Application Server based on classpath
- Cache also enabled through deployment descriptors created with the AAT
- Cache entries are created based on unique values
 - ▶ Administrator must define entries to cache
- Techniques exist for invalidating cache entries



- ▶ By caching the output of JSP and servlet requests, CPU time is not wasted processing redundant JSP and servlet calls.
- ▶ The actual HTML output from JSP and servlets, destined to be returned to the browser, is cached at the application server.
- ▶ Caching of JSP and servlet requests has a number of useful applications. It is very useful for static information, such as mutual fund prices, that are updated once a day but accessed throughout the day. Another use of caching user profile information. The profile information of a user tends to be static during a session.
- ▶ Caching is set up using XML files pointing to the appropriate Application Server classpath, allowing for different application servers to separate caches on the same node.
- ▶ Caching can also be set up through deployment descriptors using AAT (Assembly Property Extensions).
- ▶ Requests to cache are defined based on the unique values, specifically the URI value, and the arguments passed to the servlet.
- ▶ Built into the cache are a number of mechanisms allowing for invalidation of specific entry or group of entries.

Dynamic Caching Setup



- Cache enabled in dynacache.xml file or from Admin Console (AEd/AEs and AE)
 - ▶ Number of cache entries specified
 - ▶ Priority for determining how long entries are guaranteed to remain in the cache specified
 - ▶ External caches available for other applications (WebSphere Edge Server/AFPA) specified

```
<?xml version="1.0"?>
<!DOCTYPE cacheUnit SYSTEM "dynacache.dtd">
<cacheUnit>
  <cache size="10000" priority="1" />
  <externalCacheGroups> '
    <group id="afpa">
      <member address="9081"
adapterBeanName="com.ibm.servlet.dynacache.Afpa" />
    </group>
    <group id="edgeserver" type="shared">
      <member address="edgeone"
adapterBeanName="my.package.EdgeAdapter" />
    </group>
  </externalCacheGroups>
</cacheUnit>
```

- ▶ The number of cache entries is specified in the dynacache.xml file.
- ▶ The cached items are stored in the JVM Heap of the application server
- ▶ Because there is very little overhead for managing cached items, the total cache output HTML size should be added to the JVM (5000 items at 10K in size = increase maximum heap by 50 MB).
- ▶ A priority value of 1 insures a cached value remains in the cache at least once as the Least Recently Used algorithm determines which values to remove from the cache.
- ▶ For high-volume Web sites where the average output of the cached entries is difficult to determine or where a certain group of servlet and JSP calls take place frequently, the priority value should be increased.
- ▶ Priority values should not be set too high (greater than 3) or too much time will be spend calculating Least Recently Used values for expiring cache entries.
- ▶ Caches can also be available to other applications such as WebSphere Edge Server and AFPA, allowing for caching of requests before they even reach WebSphere, greatly improving performance.
- ▶ As with caching on the application server, only complete pages are cached based on unique URLs and parameters.
- ▶ Note: Adaptive Fast Path Architecture (AFPA) is a software architecture that dramatically improves the efficiency, and therefore the capacity, of Web and other network servers. The architecture includes a RAM-based cache that serves static content and a reverse proxy that can distribute requests for dynamic content to multiple servers.

Calls to Cache



- JSP and Servlet calls to cache listed in servletcache.xml
 - ▶ Specified and created by Administrator
- Multiple ways to create a unique cache ID
 - ▶ Request Parameters to the Servlet
 - ▶ Request Attributes for
 - ▶ Session Parameters

```
<?xml version="1.0"?>
<!DOCTYPE cacheUnit SYSTEM "servletcache.dtd">
<servlet>
  <path uri="/tools/Calc" />
  <request>
    <parameter id="arg1"      required="true" />
    <parameter id="arg2"      required="true" />
    <parameter id="operation" required="true" />
  </request>
  <timeout seconds="-1" />
</servlet>
```

Cache output of 5 when servlet is called for calculating 2+3 with the
URI /tools/Calc?arg1="2"&arg2="3"&operation="+"



- ▶ Calls to cache must be created by an Administrator in the servletcache.xml.
- ▶ Cache entries must be based off a unique cache ID such as request parameters, request attributes or session parameters.
- ▶ A client browser can set request parameters with a query string when submitting a form, while a servlet can set attributes on an HttpServletRequest object, then forward or include that request to another servlet. Finally, WebSphere can maintain session parameters that a servlet might want to access.
- ▶ Each one of these can be useful in creating a cache ID, as an administrator defines a variable for use in the id. With a variable declared, the value of the variable becomes a cache ID for an entry.
- ▶ For the request parameters, the URI and parameters make each request different.
- ▶ The timeout value or the length of time the item remains in the cache can also be specified, with -1 indicating that the value never times out. Even with a -1 timeout value, an entry will be removed from the cache if the cache is full and a new item is to be added or it is explicitly removed by an API call.
- ▶ You can define unlimited parameter and/or attribute objects within a <request> element, but you should only define one <request> element per <servlet> element based on the most likely call; otherwise too much time can be spent analyzing the servlet parameters.
- ▶ Information on creating cache IDs for attributes and sessions parameters available in the InfoCenter.

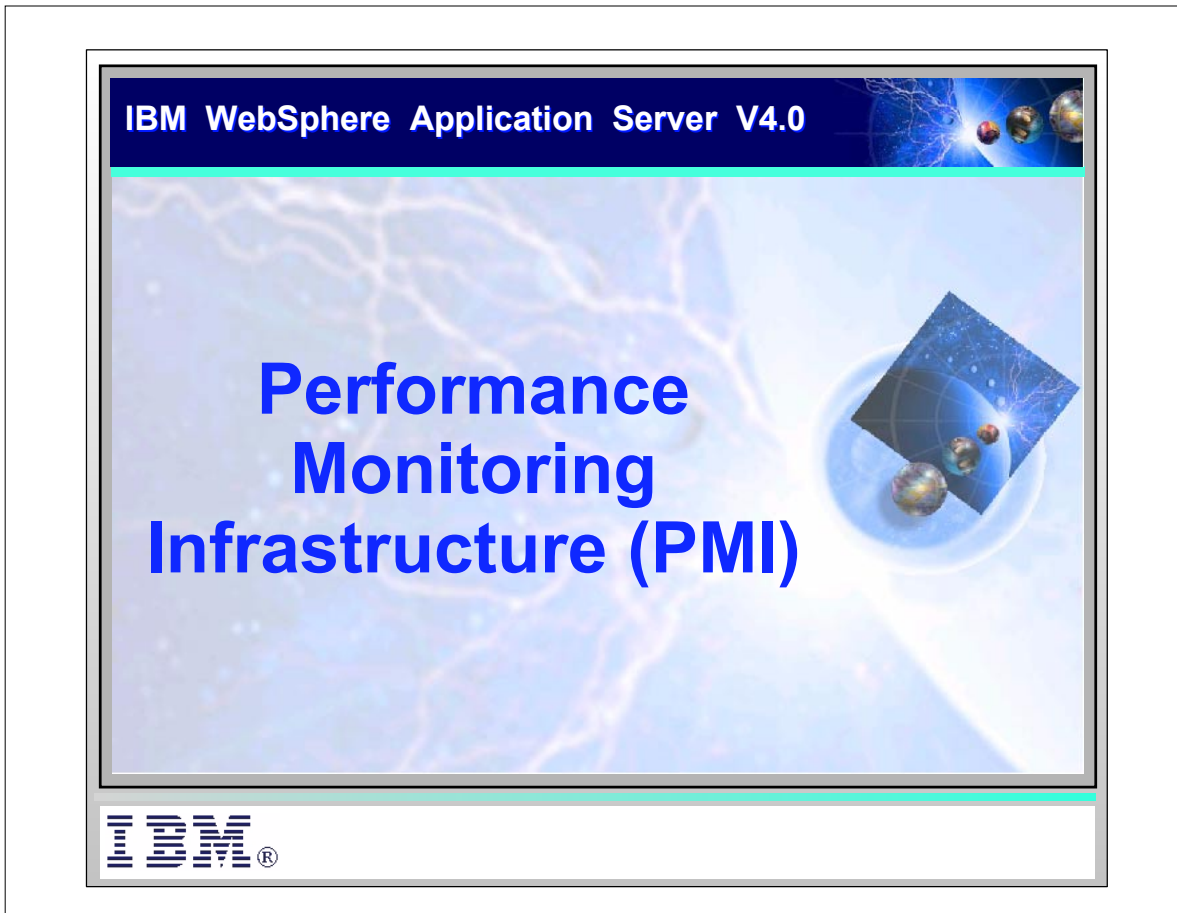
Methods of Invalidation of Cache Entry



- By definition of entry in servletcache.xml (<invalidateonly/> tag)
 - ▶ Used to invalidate a cache entry based on a specific servlet call
- By API call from another servlet
- By timeout
- Cache is full and a new entry must replace an old one based on the Least Recently Used (LRU) algorithm.



- ▶ Cache Entries can be invalidated by an end-user request or an administrator event allowing for a number of invalidation methods.
- ▶ Invalidation can be specified in the declaration of the servlet using the <invalidateonly> tag.
- ▶ Specific items or groups of items can be invalidated by calling an API from another servlet acting as a controller.
- ▶ Items in the cache can also timeout based on their definition or if the cache is full and a new entry must be added. Disposal is based on the Least Recently Used (LRU) algorithm.
- ▶ Generally, it is better to avoid using the same variable to invalidate a group of cached values and group those same entries. You will see less performance benefit than when using the variable to define a group of entries.



- ▶ PMI is new to WebSphere V4.0, providing a framework for monitoring modules within WebSphere V4.0.

PMI Overview



- Performance Monitoring Infrastructure Components
 - ▶ Server Side
 - A general extensible framework for collecting performance data on running applications
 - Incorporated into WebSphere
 - Family of products share interface; products may have different implementation
 - ▶ Wire Level Transmission
 - All WebSphere products convert server side data to same format of wire level data (XML or Java object) for transmission
 - ▶ Client side
 - All clients share same Java code for processing and displaying data
 - Examples of PMI Clients
 - ◆ Resource Analyzer
 - ◆ Custom PMI Client Application
 - ◆ Performance Servlet



- ▶ The three parts of PMI provide for a complete set of tools for monitoring specific components within WebSphere.
- ▶ The Server Side component part of WebSphere, requires no extra code instrumentation as it is part of the WebSphere code base. The framework is also built into other products in the WebSphere family, possibly with a different implementation.
- ▶ The Wire Level component of PMI communicates data in same format (standard) regardless of Server Side implementation allowing for client applications to access performance data from a number of different servers.
- ▶ Viewing of data is capable by a number of clients allowing for flexibility and easy integration to other monitoring software products. The Resource Analyzer and custom PMI client application are java clients capable of obtaining information from the PMI framework. The Performance Servlet obtains data from via an HTTP request.

Server Side Organization

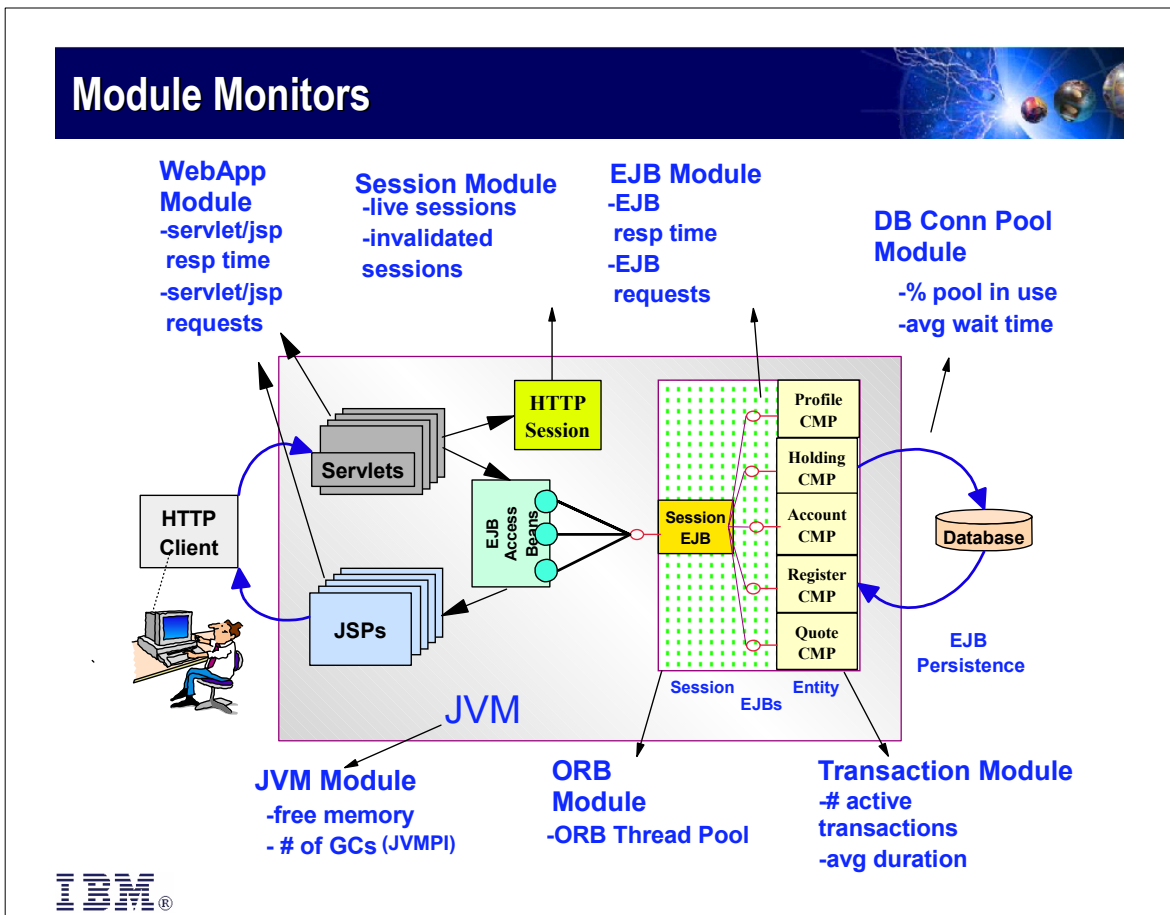


- Data organization and encapsulation
 - Data grouped into modules based on runtime components. For example, EJBs, Transaction, WebApp
 - Each module has an interface for collecting data
- Modules and instance granularity
 - EJB Module: per type, per home--method submodule
 - DB Connection Pool Module: per connection pool
 - ORB Module: per thread pool
 - Transaction Module: per transaction manager
 - WebApp Module: per web application--servlet submodule
 - Session Module: per session manager
 - JVM Module: per JVM
- Instrumentation level of amount collected for each module is set from the Administrative console



- ▶ Performance data is organized on the server based on logical components such as EJBs, Servlets, Database Connections, etc.
- ▶ Each module has interfacing data that is representative to specific parts of that module allowing for detailed monitoring.
- ▶ The Admin Console is used to set what modules in an Application server are used to report data as well as the amount of data collected or the instrumentation level.

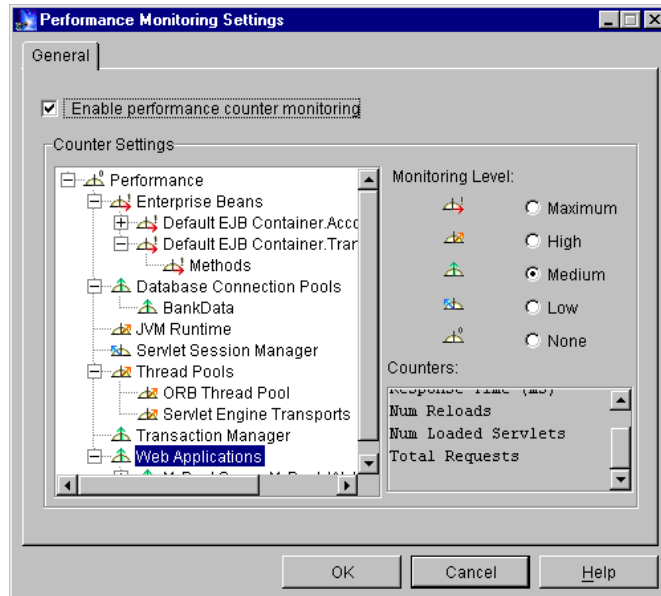
Module Monitors



- ▶ The WebSphere runtime is instrumented to provide key performance monitoring data with minimal overhead.
- ▶ The data includes counters and averages from across the runtime allowing for the monitoring of many possible bottleneck areas.

PMI Administration

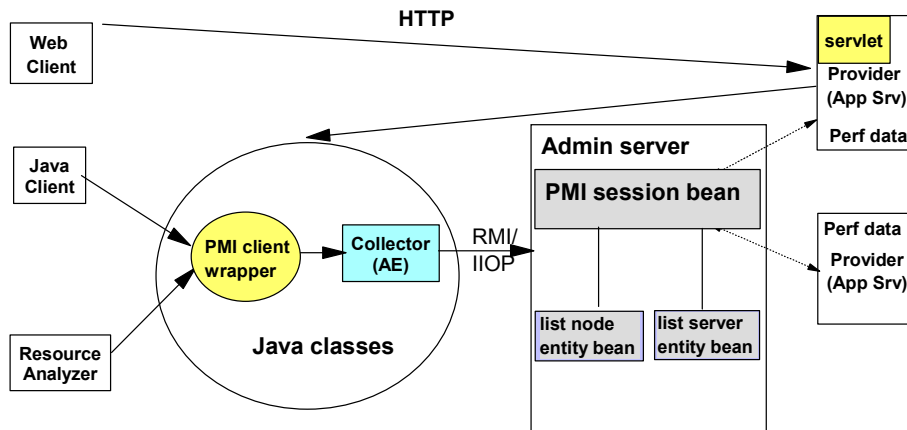
- PMI control window at Admin console
 - ▶ Allow users to change instrumentation levels when server is running
 - ▶ The level changes are persistent after server is stopped and restarted
- Instrumentation levels can also be set from the Resource Analyzer



- ▶ The instrumentation level is set from the Services table on a specific Application Server.
- ▶ There are four instrumentation levels to choose. Each level collects more data.
 - ▶ None - No measurements are recorded.
 - ▶ Low - basic incrementing and recording to a single numeric value (int, long, or double). For example, counts and sizes.
 - ▶ Medium - simple computing of data on a sample space. For example, mean, variance, standard deviation of mean .
 - ▶ High - keeps track of a level as a function of time. For example, average queue length, average concurrency.
 - ▶ Max - for method level counters in each bean home.
- ▶ Higher levels will track detailed aggregated or historical data.
- ▶ There is very little overhead and performance degradation when setting Instrumentation levels of Low, Medium, and High.
- ▶ A setting of Max, which is for performance data on EJB Methods, will show a slight degradation in overall performance.
- ▶ Instrumentation level settings are persistent after a console, Application Server, or Admin Server restart.
- ▶ The Instrumentation levels can also be set from the Resource Analyzer.

PMI Client Interfaces

- Resource Analyzer and Custom PMI Client
- Performance Servlet
- All communicate with PMI client wrapper to server side collector and PMI session bean



- ▶ The Resource Analyzer, Custom PMI Client, and the Performance Servlet all communicate with a client wrapper to a collector object which communicates with the PMI session bean.
- ▶ PMI session bean is a stateless session bean with remote interfaces for listing and obtaining performance results.
- ▶ Performance Servlet returns data in XML format obtained from the PMI session bean.
- ▶ PMI session bean is not shown in the Admin Console.

Resource Analyzer



- Built on PMI
- RMI/IIOP protocol
- Ability to log results for playback
- Included with WebSphere V4.0 product as an install option; no longer Technology Preview or separate download
- Start with ra.bat or ra.sh located in /<WAS_root>/bin directory
- Menu option in the Admin Console (AE)
- Available from the Start Menu in Windows



- ▶ The Resource Analyzer is now included with WebSphere as an install option and is no longer a Technology Preview.
- ▶ The Resource Analyzer uses the PMI client components to obtain data for display in table and chart format.
- ▶ The Resource Analyzer communicates with the PMI wrapper over RMI/IIOP.
- ▶ The Resource analyzer has the ability to log results for playback.
- ▶ The Resource Analyzer can be started from the bin directory, from the menu in the Admin Console, or from the Start menu in Windows.

Custom PMI Client Application



- Java interface to data
- PMI client classes packaged with your application
- RMI/IIOP protocol
- No additional code deployed on server side
- Convert wire level data to rich client side data
- Uses APIs to process and display data
- Application Introscope of Wily Technologies to contain PMI client



- ▶ The Custom PMI Client Application is a Java application, very similar to the Resource Analyzer, however, it is written by the end-user using the PMI client classes (j2ee.jar, perf.jar, xerces.jar, ujc.jar). These classes are shipped with WebSphere V4.0.
- ▶ The Custom application communicates with the PMI wrapper using RMI/IIOP, in the same way as the Resource Analyzer.
- ▶ The Custom PMI Client Application converts Wire Level data using the PMI Client APIs to a format suitable for display.
- ▶ Currently there are an number of third-party software vendors considering writing their own applications for monitoring WebSphere.

Using Custom PMI Client Interface



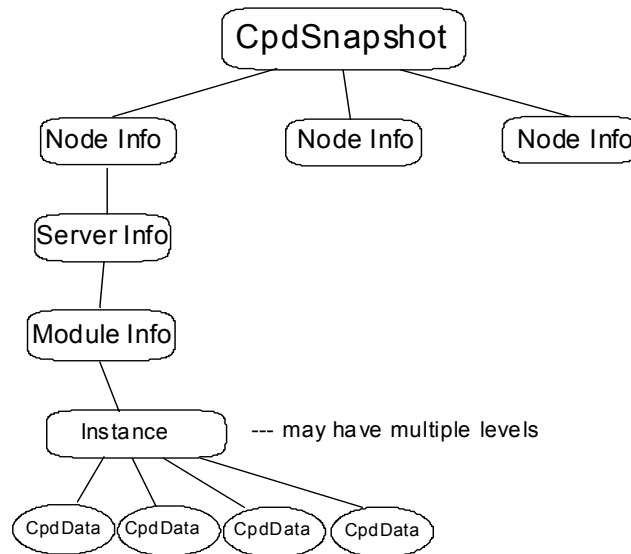
- Create an object of PMIClient
- Point the PMIClient object to the root collection
- Call methods to get performance descriptors
 - ▶ list nodes
 - ▶ list servers
 - ▶ list members: list modules, instances, and data within a server
- Get performance data - instrumentation level must be set in Admin console
- Display performance data



- ▶ The PMI client interface is very easy to use with the different modules in WebSphere organized in a hierarchy.
 - ▶ A PMIClient object can be created pointing to the root of the PMI wrapper around the collector.
 - ▶ From the root, specific modules can be accessed with hierarchical calls as components are organized by node/app server/member.
- ▶ Performance data can only be obtained if the instrumentation level has been set in the Admin console.
- ▶ With the data received from the server, it can be displayed in any format (text, GUI tables) the user chooses.

PMI Client Data Hierarchy

- Data Hierarchy allows for easy access by PMI client



- ▶ This diagram shows how the different parts are organized in a hierarchy.
- ▶ Hierarchy allows you to obtain performance for a specific component as well as components below it in the hierarchy.
- ▶ The hierarchy organization allows for easy administration and data collection.

Performance Servlet



- Servlet added by user provides performance data in XML format
- No client code required
- HTTP protocol
- Performance servlet deployed in any application server as an Enterprise Application
- Only one servlet needed to obtain performance data for all nodes in a domain
- Instrumentation level needs to be set on each module of each server manually
- Application Server with Performance Servlet must be started



- ▶ The Performance Servlet formats the performance data in XML format or an HTTP request. With the data available via HTTP and in XML format, it can be easily manipulated and used by third part applications.
- ▶ The data can simply be viewed by a browser, requiring no extra client code.
- ▶ The Performance Servlet is available as a prepackaged EAR file (perfServletApp.ear) that can be installed as an Enterprise Application.
 - ▶ The Performance Servlet can be installed on any application server as an Enterprise Application and obtain performance data from all nodes in the domain.
- ▶ Each node will need instrumentation levels set individually.
- ▶ The Application Server with the Performance Servlet installed, must be started for the servlet to return data.

Performance Servlet



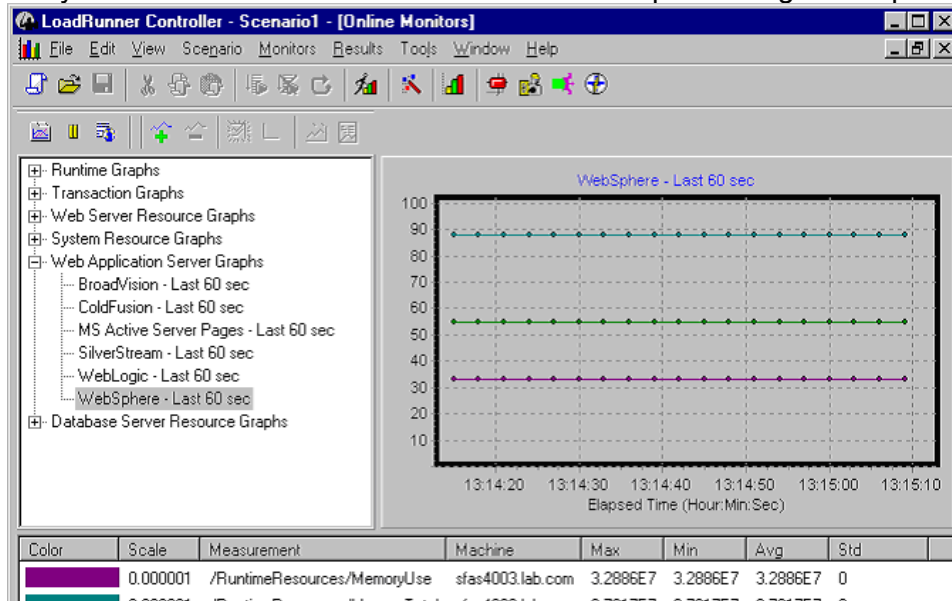
- Parameters are passed with the call to the Performance Servlet to specify what data to be returned
 - ▶ default (no parameters) - provides all performance data
 - all nodes, all servers, all modules in domain
 - instrumentation level set in Admin console
 - ▶ node - specify node
 - ▶ server - specify application server
 - ▶ module - specify PMI module
 - ▶ refreshConfig - new nodes, new servers



- ▶ With the Performance Servlet, data from specific modules can be displayed.
- ▶ Parameters passed in a hierarchical format to the servlet determine what component data is returned.
- ▶ If no parameters are passed to the servlet, all information across all nodes, servers, and modules in the domain is returned.
- ▶ Instrumentation level must be set on components before data is returned, just as with the Resource Analyzer and custom PMI client application.

Mercury Interactive WebSphere Monitor

- Uses Performance Servlet to collect data into LoadRunner Controller
- Ability to monitor end-user results and server side processing in one place



- ▶ The LoadRunner Controller from Mercury Interactive uses the Performance Servlet to monitor WebSphere.
- ▶ The WebSphere monitor and the results for test users are displayed in one convenient location.
- ▶ Because the Performance Servlet returns the data in XML, the results can be incorporated very easily into other applications.



- ▶ The Performance Tuner is a new tool to WebSphere V4.0 bringing together in one place a number of performance settings.

Performance Tuner Overview



- What does it do?
 - ▶ Brings together the different WebSphere performance settings in one place
 - ▶ Allows settings to be changed for specific App Servers and Datastores
 - ▶ Warns against extreme settings
 - ▶ Validates setting relative to other settings
- What does it not do?
 - ▶ Allow for tuning of more than one Application Server at a time
 - ▶ Provide any type of run-time monitoring
 - ▶ Provide a report indicating the affect of changes
 - ▶ Specify recommendations specific to the application

The developer knows the application best!!!



- ▶ The name "Performance Tuner" can be misleading.
- ▶ The Performance Tuner uses a wizard interface to conveniently make changes to specific performance settings within WebSphere (settings can still be manually changed individually).
- ▶ Intelligence is built into the wizard to warn against extreme settings and to validate the settings.
- ▶ Because the Performance Tuner is a simple wizard interface, it does not provide much other functionality.
- ▶ The Performance Tuner does not allow for the tuning of more than one Application Server.
- ▶ The Performance Tuner also does not provide any runtime collecting of data to provide recommendations for changes. Use Resource Analyzer for runtime monitoring.
- ▶ The Performance Tuner does not have keep track of the the affect changes have on the overall performance. Use Resource Analyzer to track the impact of the changes.
- ▶ The Performance Tuner is not the answer to all performance problems. True performance comes from efficient code!!!

Parts of Performance Tuner

■ GUI Performance Tuner Wizard

▶ Features

- Interface for entering performance settings
- Option from Wizards icon

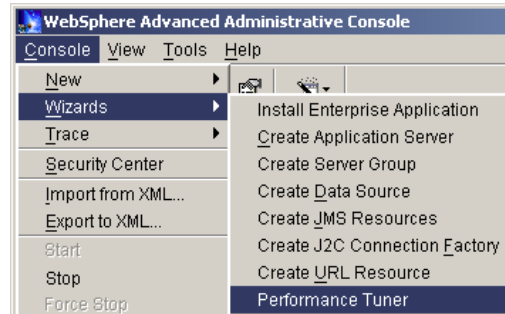
■ Stateful Session Bean

▶ Features

- Designed for one application server
- Upon request, capable of calling DB2 Smart Guide API to tune DB2 databases associated with datasources

▶ Responsibilities

- Holds values specified by administrator
- Validates settings based on knowledge the WebSphere Performance Team has gained through past experience
- Applies changes and values entered from wizard to the system's management repository objects



- ▶ The Performance Tuner uses a GUI interface available in the Admin Console for input.
- ▶ A stateful session bean performs most of the functionality, including validating settings and applying changes, as well as calling the DB2 Smart Guide API to tune specific DB2 databases (version 7.2 and higher).
- ▶ Validation is based on past experience of the WebSphere Performance Team.
- ▶ The session bean is not viewable in the administrator's console.

Adjustable Performance Values



- Servlet Container Threads
 - ▶ Available to run servlets

- EJB Server Threads
 - ▶ Threads for running EJBs
 - ▶ Pass by Reference of Java objects

- DataSource Connections
 - ▶ Threads available for connection to the datasource

- Prepared Statement Cache
 - ▶ Cache available for prepared statements in the application

- JVM Heap
 - ▶ Initial and maximum heap sizes



- ▶ The Performance Tuner allows you to change the values on a number of key settings.
- ▶ Each value is specific to a particular area where bottlenecks commonly occur.

Intelligent Validation



- Validation of Performance
 - ▶ Warnings generated in wizard for questionable settings
 - ▶ Warns of extreme values (potential overhead)
 - ▶ Compares adjacent settings (DB connections > EJB threads - is this correct?)

Warning	Description
GROWING_QUEUE_EJB	EJBS thread pool size > servlet engine thread pool size
GROWING_QUEUE_DS	DataSource connection pool size > servlet engine thread pool size OR DataSource connection pool size > EJBS thread pool size
HIGH_SETTING_SE	Servlet engine thread pool size > 100
HIGH_SETTING_EJB	EJB thread pool > 100
HIGH_SETTING_DS	DataSource connection pool size > 50
HIGH_SETTING_PREPSTMT	PreparedStatement cache size > 200
NO_DATABASE_TUNE_UNDO, and DATABASE_TUNE_REQUIRES_RESTART	When user requests to tune a DB2 database

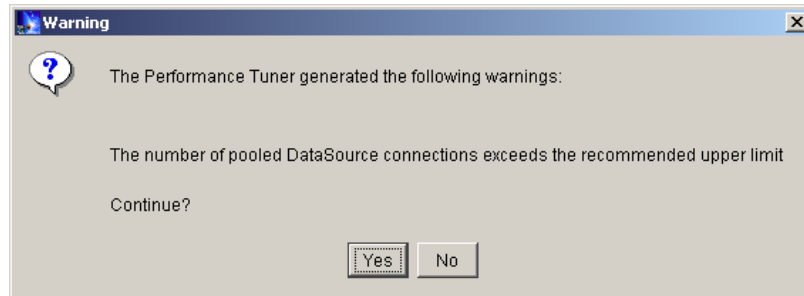


- ▶ Before the settings are implemented, validation is performed to insure that extreme values do not go unnoticed.
- ▶ The extreme limits have been put in place by the WebSphere Performance Team based on past experience.
- ▶ The Performance Tuner also checks if the settings are logical. Having more DB connections available than EJB threads to make DB connections may be incorrect and cause unnecessary overhead.

Performance Tuner Warning Message



- Warning message displayed before changes are made
- Possible to change extreme values before continuing
- Nearly all changes require a restart of the application server and the administrative server



- ▶ The Performance Tuner will display warnings about any extreme or illogical settings.
- ▶ The administrator has the ability to change the setting before any changes are committed.
- ▶ Almost all changes require the Application Server and the Administrative Server to be restarted.

Lab Preview - Performance Lab



- Put on "System Administrator" hat
- Set Instrumentation Levels from Admin Console
- Monitor with the Resource Analyzer
- Install and use Performance Servlet
- Use Performance Tuner wizard to change settings

