IT Resource Optimization

Orchestration Case Study: Integrating WebSphere and DB2 Orchestration with Enterprise Workload Manager

Authors: High Performance On Demand Solutions team and Enterprise Workload Management team
Web address: ibm.com/websphere/developer/zones/hvws
Technical contacts: Sandra Chang
Management contact: Larry Hsiung

Date: November 4, 2004

Status: Version 1.0

Abstract: *This paper describes a technology proof of concept led by IBM's High Performance On Demand Solutions team to demonstrate that the IBM Enterprise Workload Manager (EWLM) can be integrated with the IBM WebSphere and DB2 Orchestration solution to leverage EWLM's performance monitoring functions and to extend the orchestration solution with the response time metrics that EWLM aggregates.*

Executive summary

Momentum is growing for enterprises that want to move to an on demand operating environment (ODOE). IBM customers see the ODOE as the answer to the challenges of low resource utilization, high system management costs, long times to deploy new business functions, and nonintegrated business processes.

A typical on demand operating environment usually consists of a combination of routers, edge servers, Web servers, Web application servers, Enterprise JavaBeans (EJB) servers, legacy transaction servers, and database servers, all of which may run on different hardware and operating systems. The flow of business applications in such an environment involves flowing through multiple server processes that spread across multiple tiers of systems.

With such a complex environment, the customers are constantly pressured with the following challenges and are forced to look for proactive solutions that are flexible, scalable, and optimize resources:

- How to ensure that work requests are performing as expected in this type of multi-tiered environment? Are system-level resources being used for optimal performance? How to identify the performance bottlenecks?
- How to adapt to changing demands and react quickly when business model requires new application or resources? How to maintain business resiliency?
- How to avoid over-provisioning resource in support of application use to ensure delivery of service with acceptable performance and availability characteristics?
- How to lower the total cost of ownership (TCO) of the application while improving responsiveness to new business requirements and overall customer satisfaction? How to do more with less?

Workload and performance management, automation, and orchestration are the key capabilities that can help customers address these challenges.

IBM Enterprise Workload Manager (EWLM) provides customer with a complete workload management environment that manages, monitors, and reports performance statistics based on work requests submitted to applications and servers in a heterogeneous EWLM management domain.

WebSphere® and DB2® Orchestration (WADO) uses IBM Tivoli® Intelligent ThinkDynamic Orchestrator (ITITO) to autonomically provision and orchestrate application workloads across WebSphere and DB2 resources. WADO orchestrates IT resources based on the service level agreements for both the application and database servers. A key advantage WADO offers is that it allows enterprises to evolve their existing business and IT processes to take advantage of the benefits of orchestration, most significantly reduced costs and greater flexibility.

This paper describes the technology proof of concept (PoC) led by IBM's High Performance On Demand Solutions (HiPODS) team to demonstrate that EWLM can be integrated with WADO to leverage EWLM's performance monitoring functions and to extend WADO with the response time metrics that EWLM aggregates.

Note: Before using this information, read the information in “Notices” on the last page.

Contents

Solution summary

The autonomic control loop in the current WADO solution uses the service level agreement (SLA) that is based on transactions per second (TPS) and/or CPU utilization. This SLA specifies the TPS range that can be executed per server. The autonomic control loop compares the actual TPS against this range and decides if more servers are to be added or removed if the actual TPS is not within the range. It then executes a request to add or remove servers to or from the WebSphere server pool and rebalances servers according to the application's priority. While the WebSphere autonomic control loop adds or removes a server to or from the WebSphere server pool, it generates a resource hint for each database instance on which that workload depends. The hints reflect an awareness of the capacity relationship between the WebSphere workload and the database. The DB2 autonomic control loop, which is based on the CPU SLA, picks up these hints and thus can make a proactive decision to add or remove a CPU to prepare for the to-be-expected changes on the database traffic from the added or removed WebSphere application servers.

Because response time is another common measurement that is used to match the business application goal, it is desirable to extend WADO with an average transaction response time (AVGRSP) based SLA to make server or CPU resource adjustments.

EWLM version 1.1 provides the capability to define business-oriented performance goals and report on an end-to-end view of actual response time performance data across heterogeneous platforms. These performance statistics are reported on the overall business transaction class and on each tier of the application. The data displayed by EWLM's graphical user interface facilitates understanding application topology, resource utilization, and response time distribution and makes it easy to isolate performance problems.

This PoC leverages the performance monitoring functions of EWLM, uses the response time performance data that EWLM gathers, and feeds the data to WADO so that the WADO autonomic control loop can use an AVGRSP-based SLA to monitor the workload and adjust resources. The integration of WADO and EWLM not only extends WADO with another SLA type option but also shows that EWLM can be integrated into a WADO environment to provide an end-to-end performance management solution.

Overall architecture

The architecture for the PoC solution is shown in Figure 1.

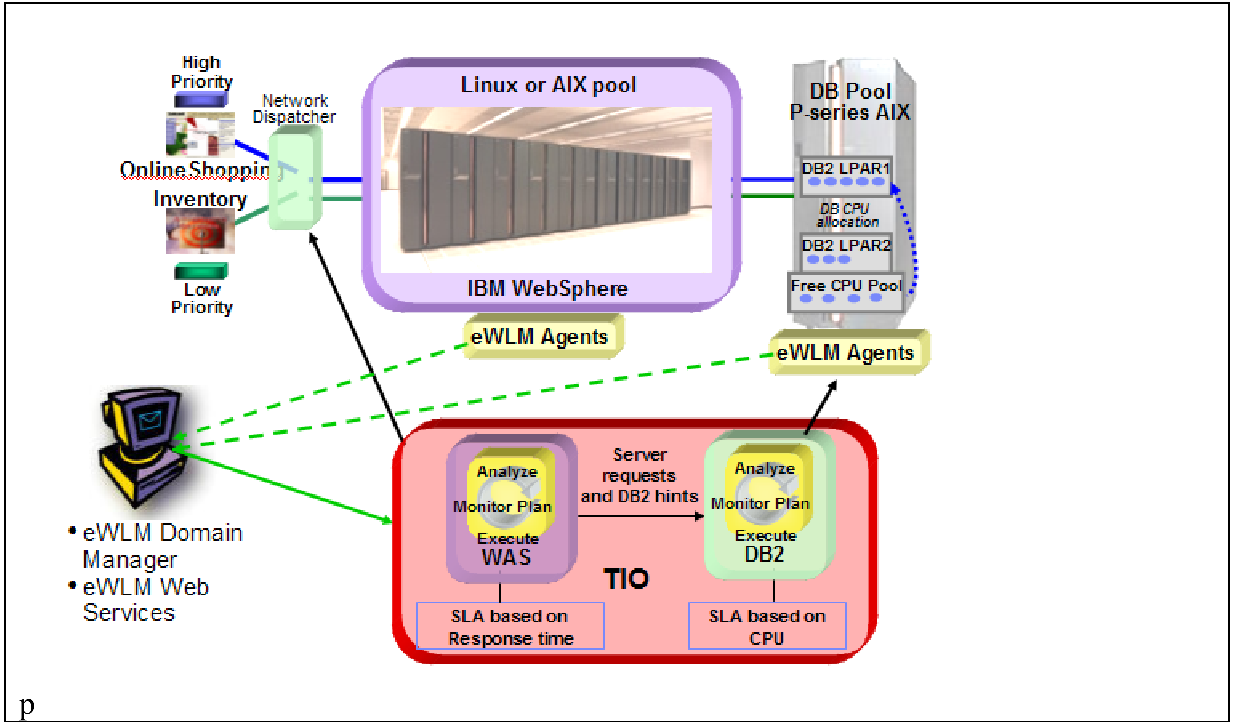


Figure 1. Components of WADO and EWLM proof of concept

The major software products and hardware platforms of this PoC environment were:

- IBM Enterprise Workload Manager (EWLM) v1.0 on Microsoft Windows 2000 for Domain Manager and on AIX® v5.2 (maintenance level 3 or later) for Managed Servers
- IBM WebSphere Application Server v5.1 with fixpack 1 or later on AIX 5.2 (maintenance level 3 or later)
- IBM WebSphere Network Deployment Manager v5.1 with fixpack 1 or later on AIX 5.2 (maintenance level 3 or later)
- IBM WebSphere Network Deployment Edge Component v5.1 on AIX 5.2 (maintenance level 3 or later)
- IBM DB2 v8.2 on AIX 5.2 (maintenance level 3 or later) pSeries® (LPAR configurable)
- IBM Tivoli Intelligent ThinkDynamic Orchestrator 2.1 on Microsoft Windows 2000
- IBM Orchestration for WebSphere and DB2 v2.1 on AIX

To simulate a business environment, there are two Java™ 2 Platform Enterprise Edition (J2EE) applications deployed. One is PlantsByWebSphere, shipped by WebSphere, that simulates an online shopping application (OnlineShopping) designated with higher priority. The second is a database intensive in-house application that simulates an inventory application (Inventory) designated with lower priority.

The major components in WebSphere and DB2 Orchestration are the IBM Tivoli Intelligent ThinkDynamic Orchestrator (ITITO), WebSphere autonomic control loop, DB2 autonomic control loop, and WAS-DB2 orchestration.

IBM Tivoli Intelligent ThinkDynamic Orchestrator

ITITO is IBM's data center automation and provisioning solution. It provides prebuilt workflows and workflow enablement that allow customers to execute their company's best policies, practices, and procedures consistently. These workflows are typically more powerful than simple scripts, and are modular, reusable, and extensible.

By monitoring the application environments in an enterprise's data center, ITITO can sense potential performance degradation and determine if action is needed. By using its capacity management and resource reservation capabilities, ITITO can also predict when resources will be needed and when these resources will become available. When ITITO determines an action is needed, it starts the appropriate provisioning workflows on demand to help match the IT resources with the applications' current load. In this manner, the data center helps to achieve the best possible business-aligned service delivery. In WADO, ITITO is used as both the database autonomic control loop and also as the overall data center manager that facilitates the communication between the autonomic control loops.

WebSphere autonomic control loop

WebSphere autonomic control loop is an on demand technology developed by the HiPODS, IBM Research, and WebSphere Development teams using the constructs of the autonomic control loop and open Web services standards to balance workloads and allocate resources on demand. The WebSphere autonomic control loop allows customers to run multiple J2EE applications in the same WebSphere infrastructure in a virtualized fashion. As the traffic of a higher priority workload increases and the SLA objective cannot be met, the system drains the lower priority workloads to other servers in the local or remote cluster to make additional servers available for high priority transactions. When all the servers in the WebSphere server pool are highly utilized serving high priority applications, WebSphere's autonomic control loop communicates with ITITO to provision additional servers from the data center server pool. As the traffic of the higher priority workloads decreases, the system reschedules the lower priority workloads on to the servers where they were previously running. This minimizes the effort to manually administer the applications when it is deployed.

DB2 autonomic control loop

The DB2 autonomic control loop is implemented by using the default ITITO control loop that manages server clusters. Logical partitioning (LPAR) CPUs are modeled as servers in the ITITO data center model, so a machine capable of being partitioned is modeled as a pool of CPU "servers." Each LPAR is seen as a server cluster, and the DB2 autonomic control loop balances CPUs instead of servers among these LPAR "clusters" based on CPU utilization of each LPAR. In addition, a specialized LPAR monitoring driver collects CPU metrics from each LPAR. Leveraging the dynamic logical partitioning (DLPAR) capability provided by IBM pSeries and AIX, new workflows add or remove CPUs to the dynamic LPARs in a manner that is transparent to the application. Notably, DB2 Version 8 also includes the function to make itself DLPAR aware, so that databases hosted by DB2 can benefit immediately from the on-the-fly addition of new resources with zero down time, and the database manager is allowed to take appropriate actions before underlying resources are removed.

WAS-DB2 orchestration

With WAS-DB2 orchestration, when the WebSphere autonomic control loop adds or removes resources to WebSphere workloads, it generates a resource hint for each database instance on which that workload depends. The hints reflect an awareness of the capacity relationship between the WebSphere workload and the database. They are propagated to the DB2 autonomic control loop through Web services. The DB2 autonomic control loop processes these events and is able to compensate proactively.

For the PoC, we integrated EWLM into the existing WADO environment and enhanced the autonomic control loops to use an SLA based on AVGRESP.

IBM Enterprise Workload Manager (EWLM)

EWLM consists of these components:

- Domain manager: The central point of control for a domain. The domain manager coordinates the activation of policies on the managed servers and the collection of performance data.
- Control center: The EWLM user interface that allows you to manage and monitor the performance of servers in an EWLM management domain. The control center resides in the same machine as the domain manager.
- Managed servers: Servers whose work requests are monitored by EWLM. Managed servers send their performance data to the domain manager.
- Domain policy: Policy that specifies performance goals for work processed in the domain. Only one policy at a time can be deployed to the domain and only one service policy at a time can be active. Domain policy contains:

    - Applications where a work request originates (entry application).
    - Platform for which you want to define a process class (optional).
    - Transaction class rules used to identify work requests initiated by an application in transaction classes.
    - Process class rules used to identify work requests initiated by a platform in process classes (optional).
    - Performance goals in service classes that correspond to your business objectives or business partners' service level agreements. Four types of performance goals are supported: average response time, percentile response time, velocity, and discretional. A velocity goal is a measure of how much progress work is making over time. A discretional goal is based on best effort.
    - The service class for work requests to use when classified by a transaction or process class.
    - Service policies that contain service classes with varying performance goals.

In a domain managed by EWLM, the applications and servers can be monitored. To monitor a server, the server must be installed with the EWLM support and be joined to a EWLM domain manager. To monitor an application, the application needs to be instrumented using Application Response Measurement (ARM) 4.0. IBM currently provides ARM-instrumentation support on the following middleware applications:

- DB2 Universal Database, Version 8.2
- WebSphere Application Server, Version 5.1.1

- Web serving plug-ins

Our PoC enabled DB2 and WebSphere Application Server.

When a work request arrives at an entry application, it is classified to a service class. Then the application calls the arm_start_transaction() API to indicate the starting of a transaction. The ARM instrumentation constructs a correlator with the classification information, which will be passed to the secondary application or to other managed servers that process the work request. The movement of the work request from one application or server to another is considered a hop. When a hop occurs a new correlator is constructed. All of the correlators are associated with the work request to track the performance of the work request until it completes. When the work request is completed, the domain manager consolidates all of the performance data that it received and includes this performance data in the detail reports, monitors, and topologies.

The transaction on the edge is considered to be at "Hop 0", the transaction executing at the second tier is considered to be at "Hop 1". The subtransactions running in the following tiers will be considered to be executing for the transaction class determined at hop 0.

Figure 2 describes the transaction flow with application resource measurement (ARM):

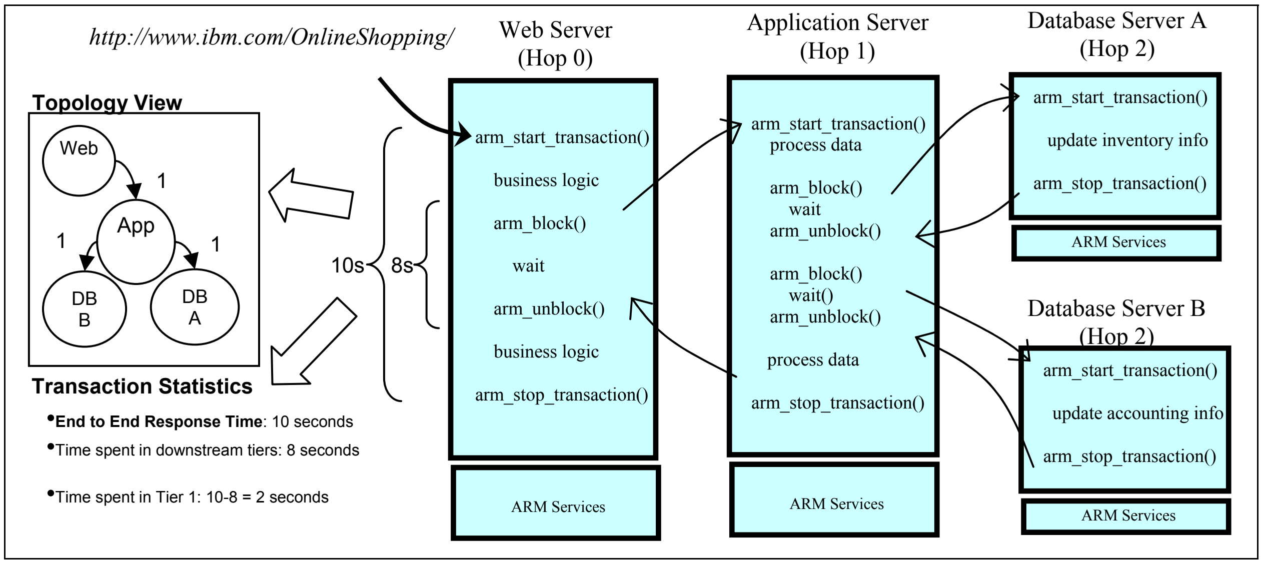


Figure 2. Transaction flow with application resource measurement (ARM)

After EWLM receives the individual transaction or process performance statistics, it aggregates the statistics by grouping a large quantity of data under an aggregation point. Examples of large quantities of data are response time statistics for several thousand transactions and resource statistics for several hundred processes. Examples of aggregation point are process class, transaction class, service class, application environment, application environment instance, and hop. The performance statistics available that are being aggregated include:

1. Transaction response time statistics
2. Resource utilization statistics for transaction, process and managed server.
3. Response time distribution
4. Application and server topology

Figure 3 and Figure 4 show the graphical user interface provided by the EWLM control center.

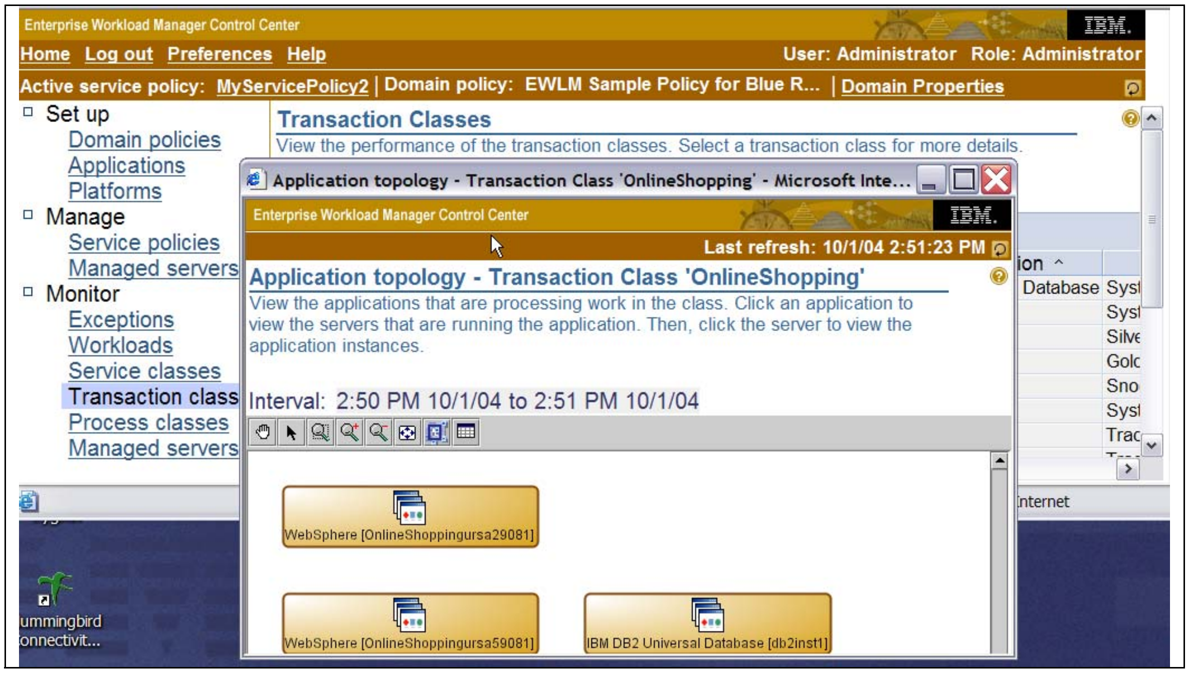
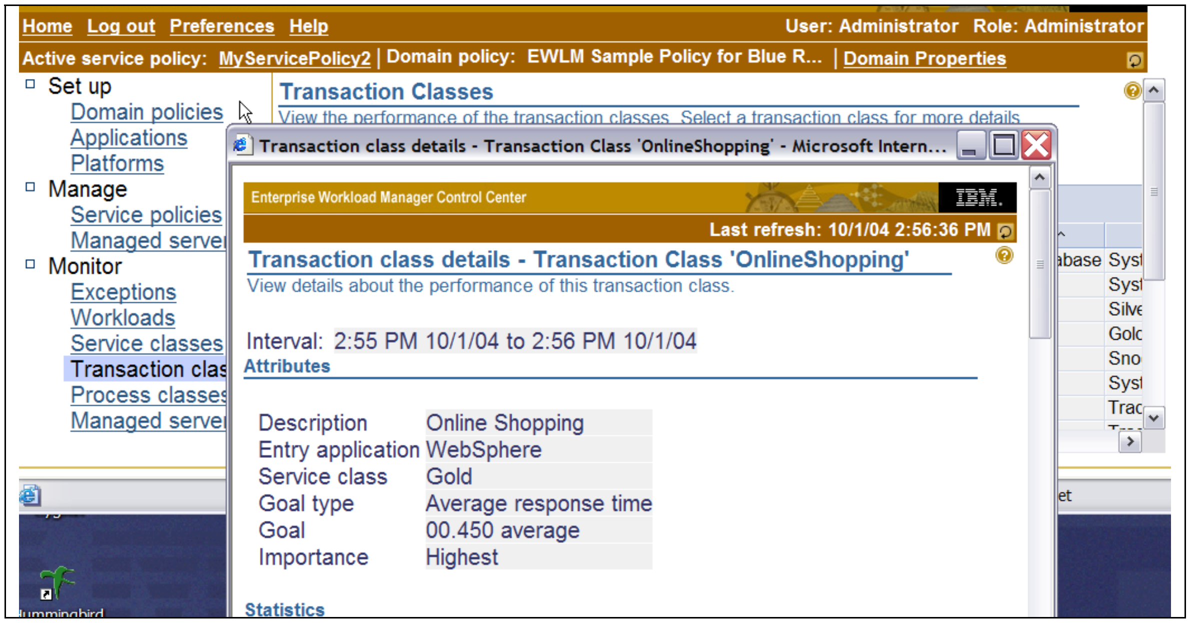

Enterprise Workload Manager Control Center
Home Log out Preferences Help
User: Administrator Role: Administrator
Active service policy: MyServicePolicy2 | Domain policy: EWLM Sample Policy for Blue R... | Domain Properties
Set up
Domain policies
Applications
Platforms
Manage
Service policies
Managed servers
Monitor
Exceptions
Workloads
Service classes
Transaction class
Process classes
Managed servers
Transaction Classes
View the performance of the transaction classes. Select a transaction class for more details.
Application topology - Transaction Class 'OnlineShopping' - Microsoft Inte...
Last refresh: 10/1/04 2:51:23 PM
Application topology - Transaction Class 'OnlineShopping'
View the applications that are processing work in the class. Click an application to view the servers that are running the application. Then, click the server to view the application instances.
Interval: 2:50 PM 10/1/04 to 2:51 PM 10/1/04
WebSphere [OnlineShoppingursa29081]
WebSphere [OnlineShoppingursa59081]
IBM DB2 Universal Database [db2inst1]

Figure 3. Application topology view of EWLM Control Center


Home Log out Preferences Help
User: Administrator Role: Administrator
Active service policy: MyServicePolicy2 | Domain policy: EWLM Sample Policy for Blue R... | Domain Properties
Set up
Domain policies
Applications
Platforms
Manage
Service policies
Managed servers
Monitor
Exceptions
Workloads
Service classes
Transaction class
Process classes
Managed servers
Transaction Classes
Transaction class details - Transaction Class 'OnlineShopping' - Microsoft Intern...
Enterprise Workload Manager Control Center
Last refresh: 10/1/04 2:56:36 PM
Statistics

Transaction class details - Transaction Class 'OnlineShopping'

View details about the performance of this transaction class.

Interval: 2:55 PM 10/1/04 to 2:56 PM 10/1/04

Attributes

Description	Online Shopping
Entry application	WebSphere
Service class	Gold
Goal type	Average response time
Goal	00.450 average
Importance	Highest

Figure 4. Transaction class detail view of EWLM Control Center

The end-to-end performance data can also be retrieved through EWLM System Management JMX API support on Domain Manager.

With this PoC, we developed a Web service that resides on the EWLM domain manager machine to invoke the JMX APIs to retrieve the transaction response time statistics. The autonomic controls loops in WADO can in turn retrieve these performance data using this Web service.

Implementation

This section summarizes the steps we followed to implement this solution and the lessons we learned:

- Setting up the EWLM environment in WADO
- Defining and deploying the domain policy
- Enabling AMR instrumentation on the applications
- Creating the JMX client Web service
- Configuring JMX client security
- Creating the WADO data provider Web service

Setting up the EWLM environment in WADO

Using the existing WADO environment, we added a Windows machine to host the EWLM domain manager. For the existing WebSphere application server and DB2 server machines, we installed the EWLM managed servers component and added them to the EWLM domain manager. To join these machines to the domain, we enabled the EWLM ARM service, then started the EWLM managed server on each machine. When the managed server is started, it communicates with the designated domain manager and becomes part of the management domain.

One thing to look out for is that the managed server component can't be started without the following three file sets installed on AIX: bos.rte 5.2.0.30, bos.net.tcp.client 5.2.0.30 and bos.net.ewlm.rte.

Defining and deploying the domain policy

Not to complicate the implementation and yet to be able to prove the technology, we took the sample domain policy and added two service classes, Gold and Silver, which are average response time performance type goals and two transaction classes that have application URI as the classification rule. Requests on OnlineShopping will be classified to the OnlineShopping transaction class and requests on Inventory will be classified to the Inventory transaction class. Because OnlineShopping has higher priority, it is assigned with the Gold service class. This implies that under higher load conditions, servers will be taken away from processing Inventory traffic and made available to process OnlineShopping traffic.

In an actual implementation, it's very important to understand the behavior of the business application to define the right transaction class and determine its service requirement.

Enabling application resource measurement (ARM)

Next, we enabled ARM instrumentation for the WebSphere applications through the WebSphere Administration Console:

- Enable PMI Request Metrics and ARM
- For each WebSphere Application Server, add the following Custom Properties to the Application Server Java Virtual Machine

Property	**Value**
ArmTransactionFactory	com.ibm.wlm.arm40SDK.transaction. Arm40TransactionFactory
com.ibm.websphere.pmi.reqmetrics.ARMIMPL	arm4
java.library.path	EWLMMS_LIBPATH
ws.ext.dirs	EWLMMS_HOME/classes/Arm
com.ibm.websphere.pmi.reqmetrics.loggingEnabled (optional)	False
com.sun.tools.javac.main.largebranch (optional)	True
com.ibm.websphere.pmi.reqmetrics.PassCorrelatorToDB	True The correlator is passed only when this property is set to True.

For more information, see the EWLM section of the eServer™ Info Center.

To enable DB2 instrumentation, we:

- Used DB2 Universal JDBC type 4 driver in WebSphere Application Server for our data sources so that the ARM correlator will be passed from WebSphere to DB2

- Set the following variables, using db2set, in the DB2 instance that we want EWLM to monitor:
  DB2LIBPATH=/usr/lib
  DB2_ARM_ENABLED=yes

Creating the JMX client Web service

With server machines and applications ready with EWLM, the next step was to write the code for retrieving the EWLM response time performance data and feeding them into the WADO autonomic control loop.

Because there is a restriction on the EWLM JMX API support that requires the JMX client to exist on the same EWLM domain manager machine, we developed a Web service (referred to later as data retriever), which was to be deployed on the WebSphere application server that hosts the EWLM control center. This Web service invokes the JMX APIs to retrieve from the EWLM domain manager the transaction class response time performance data. We used wlmCollectData(), which returns the total mode data. The total mode data is the statistics accumulated since the domain policy was activated; total mode data is reset after every activation.

The data for the transaction class is returned in a tree-like hierarchy described by Figure 5.

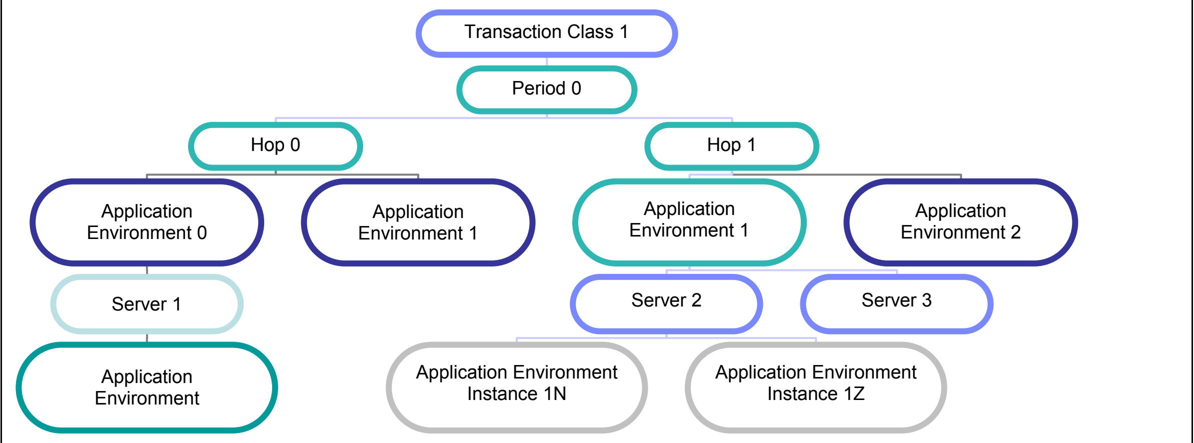

Transaction Class 1
Period 0
Hop 0
Hop 1
Application Environment 0
Application Environment 1
Application Environment 1
Application Environment 2
Server 1
Server 2
Server 3
Application Environment
Application Environment Instance 1N
Application Environment Instance 1Z

Figure 5. Hierarchy for transaction class data (see table below for node descriptions)

Node	Description
Transaction class	Aggregated transaction response time and resource utilization statistics for end-to-end transactions that are associated with the transaction class.
Period	In EWLM 1.0, there is only a single period under a service class. This is currently the same as transaction class.
Hop	Aggregated transaction response time and resource utilization statistics for transactions that were classified to a specific transaction class at the edge and executed at a specific hop.
Application environment	Aggregated transaction response time and resource utilization statistics for transactions that were executed under a specified application environment, classified to a specific transaction class at the edge and executed at a specific hop.
Server	Aggregated transaction response time and resource utilization statistics for transactions that were ran on a specific managed server, executed under a

	specific application environment, classified to a specific transaction class at the edge and executed at a specific hop.
Application environment instance	Aggregated transaction response time and resource utilization statistics for transactions that were executed under a specific application environment Instance, ran on a specific managed server, executed under a specific application environment, classified to a specific transaction class at the edge and executed at a specific hop.

With our two J2EE applications, we have the corresponding OnlineShopping and Inventory transaction classes. For each transaction class, we have WebSphere as hop 0 and DB2 as hop 1. The Web service code loops through this structure and returns the total successful transaction time and transaction counts for the WebSphere tier and DB2 tier respectively. The time does not include the time spent on the subsequent tier (*total successful transaction time* subtracted with *total blocked time*).

Configuring JMX client security

Before we deployed this Web service, a security configuration for using JMX API was needed. Security for the EWLM systems management APIs is implemented using authentication for clients when the jmxConnect API is invoked and by an appropriate JAAS security policy. When a client invokes the jmxConnect API, the client needs to pass an appName and passPhrase. This appName and passPhrase pair is checked against a credentials database (EWLMcred.db, which is located in the Interfaces subdirectory of the EWLM working directory) that the EWLM JMX agent maintains. If the appName and passPhrase pair is defined in the credentials database, then a jmxConnector is created and associated with that jmxConnector is an authenticated JAAS principal. Principals that need to have access to the systems management APIs need a grant entry defined in the EWLM JAAS security policy (JMXUSER.POLICY located in the Interfaces subdirectory of the EWLM working directory. EWLM provides these Java command to update the credentials file: OTPGenerator databaseURL [add|delete|update] -U username [-A 'otp-md5'|'otp-sha1'] [-S seed] [-s sequence] [-p passphrase].

Because this Web service resides on the same machine as the domain manager, we configured this security requirement on the domain manager machine accordingly and deployed the Web service on the WebSphere Application Server that hosts the EWLM control center.

Creating the WADO data provider Web service

We then created for the WebSphere autonomic control loop a new data provider Web service that uses AVGRSP as the SLA measurement. The data provider of this SLA type calls, at the interval specified in the SLA, the data retriever Web service described above to retrieve the transaction response time data. The returned data is the total aggregated response time for the transaction class since the domain policy was last activated. To calculate the average response time for the last time interval, last ten seconds for example, it requires one set of data on the current time and another on the previous ten seconds. We use the transaction response time difference and transaction count difference to compute the average response time for this time period. The computed average response time data is fed back to the control loop. The control loop balances the resource accordingly by adding or removing servers and assigns the added resources to the higher priority application. This Web service was deployed as part of the WebSphere autonomic control loop.

Lastly, we specified an AVGRSP-based SLA for the OnlineShopping application. The WebSphere autonomic control loop will now monitor the OnlineShopping workload based on the average response time.

The system is now set for running scenarios.

Lessons learned

During testing, we observed the following about the response time data:

- The block time on hop 0 (WebSphere) is less than the response time on hop 1 (DB2).
- A misleading 0 average response time computed from the data provider Web service.

These observations are explained below.

For EWLM 1.0, each managed server sends the performance statistics to the domain manager every ten seconds (approximately). The exact time that this happens varies from managed server to managed server, depending on factors such as when the process is started and the amount of time that the managed server spent on initialization.

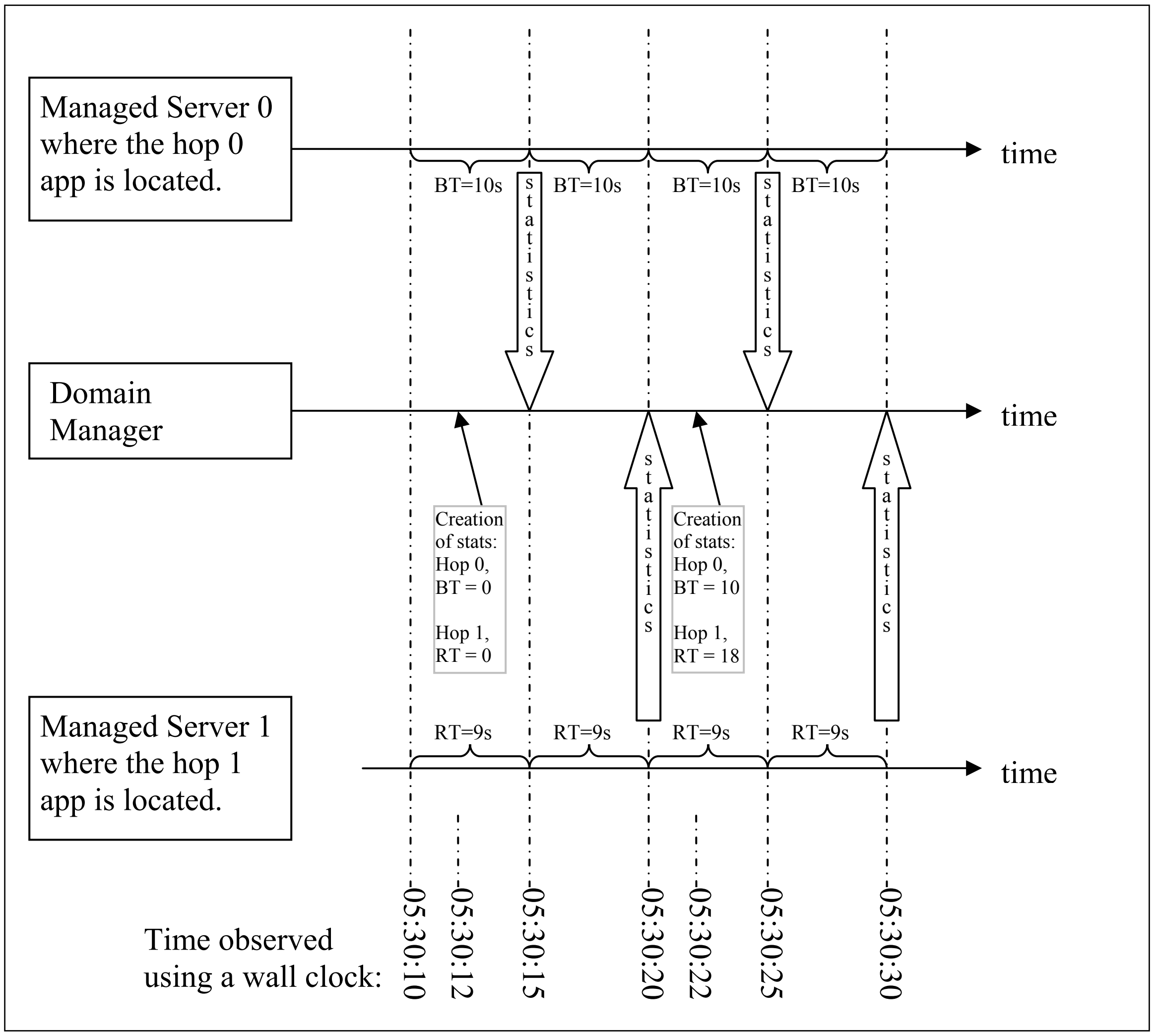
The domain manager creates the reporting statistics every ten seconds (approximately), which happens asynchronously to the arrival time of the managed server statistics.

Therefore, for short intervals the block time returned from wlmCollectDatat may not be a good source to deduce the time spent on the network between hops.

Also our data provider Web service in the WADO autonomic control loop calls the data retriever Web service at the interval specified in the SLA, which was set to five seconds, but as mentioned above, the domain manager creates the reporting statistics every ten seconds (approximately), we therefore may possibly get the same old data on the second invocation and wrongly deduce that the average response time is 0.

Hence, we changed our data provider code to ensure a longer interval so that EWLM will have created new reporting statistics for each invocation.

Figure 6 is a graphic explanation of the test results.


Managed Server 0 where the hop 0 app is located.
time
BT=10s
BT=10s
BT=10s
BT=10s
statistics
statistics
Domain Manager
time
Creation of stats: Hop 0, BT = 0 Hop 1, RT = 0
statistics
Creation of stats: Hop 0, BT = 10 Hop 1, RT = 18
statistics
Managed Server 1 where the hop 1 app is located.
RT=9s
RT=9s
RT=9s
RT=9s
time
Time observed using a wall clock:
05:30:10
05:30:12
05:30:15
05:30:20
05:30:22
05:30:25
05:30:30

Note: The second statistics has a BT(Hop 0) < RT(Hop 1).

Figure 6. Graphic explanation for test results

Scenarios

This section describes two scenarios that simulate the retail business environment. Based on the retailer's business goal, OnlineShopping demands quicker response to the browsing and buying requests. Therefore it has a shorter average response time performance goal and higher priority for server resources. Inventory simulates an internal business application and can accommodate a longer average response time performance goal and lower priority for server resources. Each application has an associated database on its own database server. Each database server runs on its own LPAR and both LPARs share a free CPU pool.

The example screen show in Figure 7 has four windows. The first, seen on the bottom half of the screen, is the WebSphere Administrative Console. It shows the different application servers:

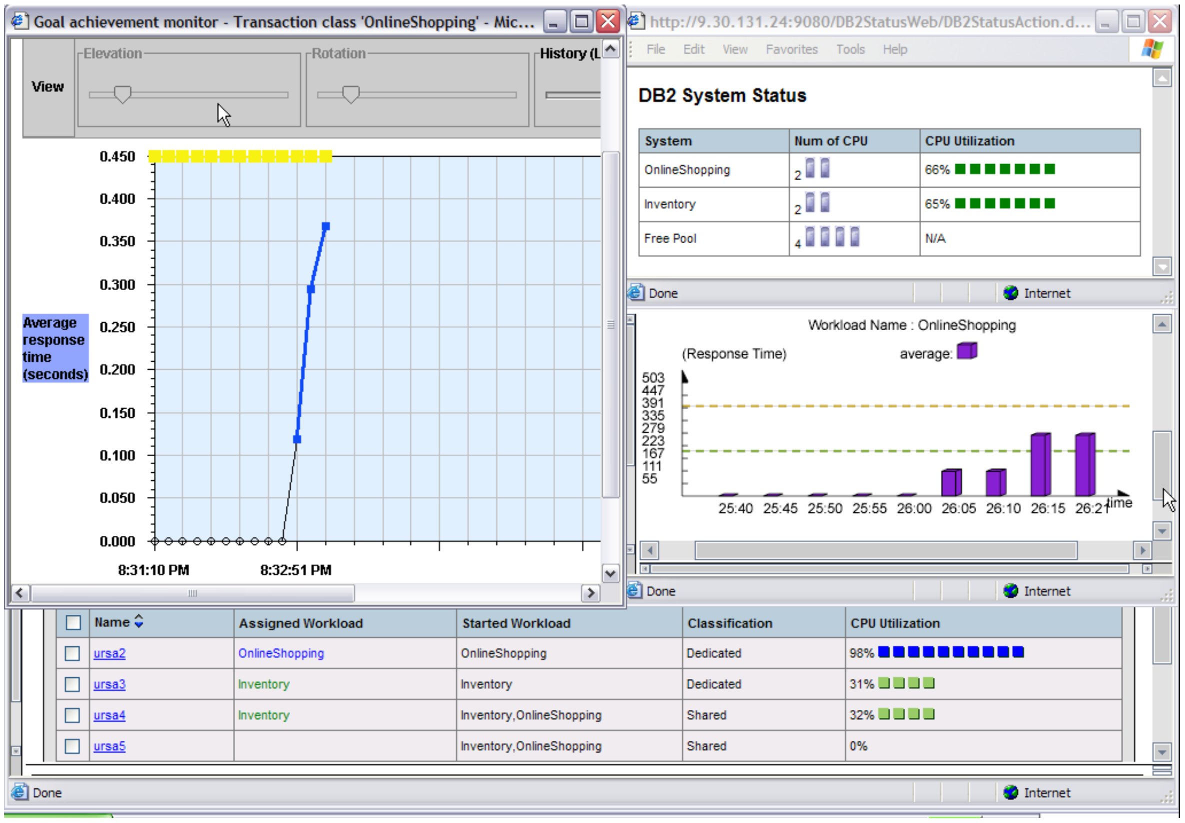
- The first server is dedicated to the OnlineShopping.
- The second server is dedicated to Inventory.
- The last two servers are to be shared to run either application. One of the servers is currently running Inventory.

The second window, in the upper left corner, is the EWLM control center showing end-to-end response time chart for transaction monitor. The time unit is seconds.

The third window, in the middle right corner, is also the WebSphere administrative console showing the average response time monitoring for OnlineShopping. The time unit is milliseconds. The two lines are the minimum and maximum service level agreement for OnlineShopping. When the average response time for the application (transaction) exceeds the maximum, more servers are allocated to accommodate the demand.

The fourth window, in the upper right corner, is the DB2 status window:

- The first row shows the first LPAR, which has the database for OnlineShopping
- The second row show the second LAPR, which has the database for Inventory
- The third row is a free pool of CPUs that can be added on demand to the first or second LPAR



DB2 System Status

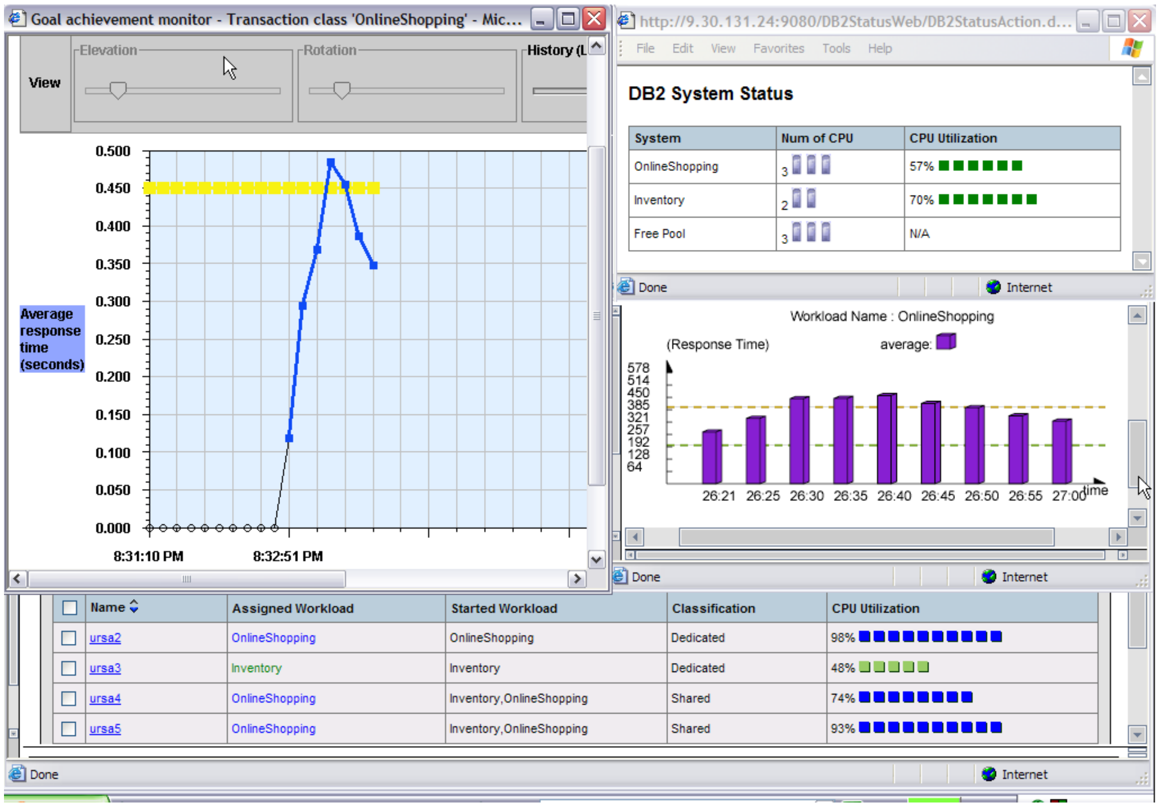
System	Num of CPU	CPU Utilization
OnlineShopping	2	66%
Inventory	2	65%
Free Pool	4	N/A

☐	Name	Assigned Workload	Started Workload	Classification	CPU Utilization
☐	ursa2	OnlineShopping	OnlineShopping	Dedicated	98%
☐	ursa3	Inventory	Inventory	Dedicated	31%
☐	ursa4	Inventory	Inventory,OnlineShopping	Shared	32%
☐	ursa5		Inventory,OnlineShopping	Shared	0%

Figure 7. Sample solution screen showing WebSphere Administrative Console, EWLM Control Center, and DB2 system status

Scenario 1

Figure 8 depicts scenario 1 where the SLA for the OnlineShopping application is violated when the OnlineShopping traffic increases. The autonomic control loop for WebSphere detects the violation and allocates additional servers to OnlineShopping from the shared pool and from the Inventory application. While it adds servers, the control loop also proactively notifies the DB2 autonomic function in ITITO to anticipate additional workload. To prepare for the increased workload, a workflow script is executed to allocate additional CPUs from the free pool to the OnlineShopping LPAR.



DB2 System Status

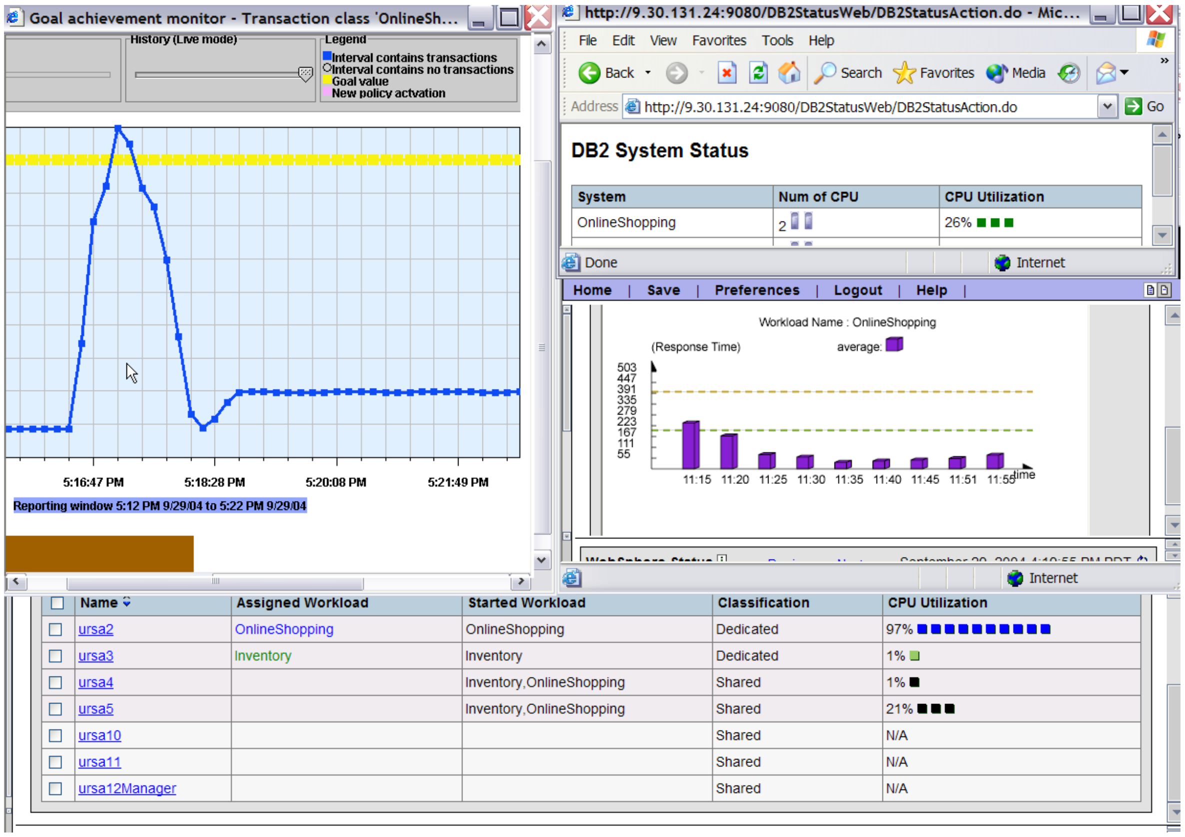
System	Num of CPU	CPU Utilization
OnlineShopping	3	57% 🟩🟩🟩🟩🟩🟩
Inventory	2	70% 🟩🟩🟩🟩🟩🟩🟩
Free Pool	3	N/A

	Name	Assigned Workload	Started Workload	Classification	CPU Utilization
☐	ursa2	OnlineShopping	OnlineShopping	Dedicated	98% 🟦🟦🟦🟦🟦🟦🟦🟦🟦🟦
☐	ursa3	Inventory	Inventory	Dedicated	48% 🟩🟩🟩🟩🟩
☐	ursa4	OnlineShopping	Inventory,OnlineShopping	Shared	74% 🟦🟦🟦🟦🟦🟦🟦🟦
☐	ursa5	OnlineShopping	Inventory,OnlineShopping	Shared	93% 🟦🟦🟦🟦🟦🟦🟦🟦🟦🟦

Figure 8. Autonomic resource balancing occurs when the OnlineShopping application begins to violate its maximum service level agreement

Scenario 2

Figure 9 depicts scenario 2 showing that when the traffic of OnlineShopping decreases, its SLA is violated against the minimum average response time. The autonomic control loop for WebSphere detects the violation and deallocates servers from the OnlineShopping application and returns them to the shared pool. Again, while it deallocates servers, it also proactively notifies the DB2 autonomic function in ITITO to anticipate reduced workload. To prepare for the decreased workload, a workflow script is executed to deallocate CPUs from OnlineShopping and return them to the free pool.


Goal achievement monitor - Transaction class 'OnlineSh...
History (Live mode)
Legend
Interval contains transactions
Interval contains no transactions
Goal value
New policy actvation
5:16:47 PM
5:18:28 PM
5:20:08 PM
5:21:49 PM
http://9.30.131.24:9080/DB2StatusWeb/DB2StatusAction.do - Mic...
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://9.30.131.24:9080/DB2StatusWeb/DB2StatusAction.do Go
Done
Internet
Home | Save | Preferences | Logout | Help |
Workload Name : OnlineShopping
(Response Time)
average:
503 447 391 335 279 223 167 111 55
11:15 11:20 11:25 11:30 11:35 11:40 11:45 11:51 11:55
time
Internet

Reporting window 5:12 PM 9/29/04 to 5:22 PM 9/29/04

DB2 System Status

System	Num of CPU	CPU Utilization
OnlineShopping	2	26%

Name	Assigned Workload	Started Workload	Classification	CPU Utilization
ursa2	OnlineShopping	OnlineShopping	Dedicated	97%
ursa3	Inventory	Inventory	Dedicated	1%
ursa4		Inventory,OnlineShopping	Shared	1%
ursa5		Inventory,OnlineShopping	Shared	21%
ursa10			Shared	N/A
ursa11			Shared	N/A
ursa12Manager			Shared	N/A

Figure 9. Autonomic resource balancing occurs when the OnlineShopping application begins to violate its maximum service level agreement

Conclusion

This PoC demonstrated that EWLM can be easily integrated into the existing WADO environment. With the orchestration benefit that WADO already offers in the area of autonomic resource balancing and cost reduction, integration of EWLM adds the additional value of using transaction response time as another measurement metric for setting the service level agreement to take the advantage of the resource orchestration.

This PoC utilized a small subset of the functions that EWLM provides. Besides the transaction response time data, it also collects data such as processor utilization, server delay, and transaction rate, which are aggregated in different levels of granularity and are useful in understanding the application and server topology along with their resource utilization. EWLM offers customer a powerful facility for end-to-end performance management.

References

- The WebSphere Application Server Performance Web site provides a centralized access to many helpful performance reports, tools, and downloads. See www.ibm.com/software/webservers/appserv/performance.html

- See all the IBM High Performance On Demand Solutions white papers at www.ibm.com/websphere/developer/zones/hvws

- Learn about the IBM Tivoli Intelligent ThinkDynamic Orchestrator at www.ibm.com/software/tivoli/products/intell-orch/

- See all the IBM Tivoli products at www.ibm.com/software/tivoli/products/

- IBM Redbooks have additional WebSphere performance information at www.ibm.com/redbooks.nsf/portals/Websphere

- For centralized access to the technical information about software solutions for the IBM eServer platforms, see the IBM eServer Information Center at www.ibm.com/eserver/v1r1/en_US/index.htm?info/icmain.htm

- For additional EWLM information see the IBM Redbook *IBM Enterprise Workload Manager*, SG24-6350 at www.ibm.com/abstracts/sg246350.html?Open

Acknowledgements

We acknowledge the major contributors from the HiPODS and EWLM teams to this project and this paper: Jeff Aman, Patrick Chan, Willy Chiu, Donna Dillenberger, Rekha Garapati, Susan Holic, Larry Hsiung, Chin Huang, Rahul Jain, YuanHui Liu, Peggy McKeon, Geoffrey Lubold, Ed Ng, Daryl Scott, Jayashree Subrahmonia, Peter Tan, Vien Tran, Matt Wang, Eric Ye, Peter Yocom, Lei Zhang

Notices

Trademarks

<u>Special notice</u>