WebSphere Application Server z/OS and WLM Interactions

David Follis
WebSphere Application Server z/OS Development
IBM Corporation

Thursday, August 5, 2010: 1:30 PM-2:30 PM
Trademarks

The following are trademarks of the International Business Machines Corporation in the United States and/or other countries.

CICS®
DB2®
GDPS®
Geographically Dispersed Parallel Sysplex
HiperSockets
IBM®
IBM eServer
IBM logo®
IMS
On Demand Business logo

* Registered trademarks of IBM Corporation

The following are trademarks or registered trademarks of other companies.

Java and all Java-related trademarks and logos are trademarks of Sun Microsystems, Inc., in the United States and other countries.
UNIX is a registered trademark of The Open Group in the United States and other countries.
Microsoft, Windows and Windows NT are registered trademarks of Microsoft Corporation.
SET and Secure Electronic Transaction are trademarks owned by SET Secure Electronic Transaction LLC.

* All other products may be trademarks or registered trademarks of their respective companies.

Notes:

Performance is in Internal Throughput Rate (ITR) ratio based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput improvements equivalent to the performance ratios stated here.

IBM hardware products are manufactured from new parts, or new and serviceable used parts. Regardless, our warranty terms apply.

All customer examples cited or described in this presentation are presented as illustrations of the manner in which some customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics will vary depending on individual customer configurations and conditions.

This publication was produced in the United States. IBM may not offer the products, services or features discussed in this document in other countries, and the information may be subject to change without notice. Consult your local IBM business contact for information on the product or services available in your area.

All statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.

Information about non-IBM products is obtained from the manufacturers of those products or their published announcements. IBM has not tested those products and cannot confirm the performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Prices subject to change without notice. Contact your IBM representative or Business Partner for the most current pricing in your geography.
Disclaimer

The information contained in this documentation is provided for informational purposes only. While efforts were many to verify the completeness and accuracy of the information contained in this document, it is provided “as is” without warranty of any kind, express or implied.

This information is based on IBM’s current product plans and strategy, which are subject to change without notice. IBM will not be responsible for any damages arising out of the use of, or otherwise related to, this documentation or any other documentation.

Nothing contained in this documentation is intended to, nor shall have the effect of, creating any warranties or representations from IBM (or its suppliers or licensors), or altering the terms and conditions of the applicable license agreement governing the use of the IBM software.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput improvements equivalent to the performance ratios stated here.

All customer examples cited or described in this presentation are presented as illustrations of the manner in which some customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics will vary depending on individual customer configurations and conditions.
# WebSphere Application Server Sessions

<table>
<thead>
<tr>
<th>Room</th>
<th>Day</th>
<th>Time</th>
<th>Title</th>
<th>Speaker</th>
</tr>
</thead>
<tbody>
<tr>
<td>312</td>
<td>Monday</td>
<td>12:15</td>
<td>Lab</td>
<td>Stephen</td>
</tr>
<tr>
<td>203</td>
<td>Monday</td>
<td>4:30</td>
<td>WebSphere: What's New?</td>
<td>Follis</td>
</tr>
<tr>
<td>203</td>
<td>Wednesday</td>
<td>9:30</td>
<td>WebSphere 101</td>
<td>Houde/Stephen</td>
</tr>
<tr>
<td>201</td>
<td>Wednesday</td>
<td>1:30</td>
<td>Introduction to IBM Support Assistant (ISA)</td>
<td>Hutchinson</td>
</tr>
<tr>
<td>200</td>
<td>Wednesday</td>
<td>3:00</td>
<td>WebSphere Process Manager and Business Process Manager Configuration</td>
<td>Hutchinson</td>
</tr>
<tr>
<td>200</td>
<td>Wednesday</td>
<td>4:30</td>
<td>OSGi/JPA/Batch Feature Packs</td>
<td>Follis / Bagwell</td>
</tr>
<tr>
<td>203</td>
<td>Wednesday</td>
<td>6:00</td>
<td>WebSphere for z/OS: I'm no longer a dummy but...</td>
<td>Bagwell</td>
</tr>
<tr>
<td>310</td>
<td>Thursday</td>
<td>8:00</td>
<td>WOLA Application Designs</td>
<td>Bagwell</td>
</tr>
<tr>
<td>310</td>
<td>Thursday</td>
<td>9:30</td>
<td>Security Architecture: How does WebSphere Play?</td>
<td>O'Donnell</td>
</tr>
<tr>
<td>310</td>
<td>Thursday</td>
<td>11:00</td>
<td>WAS on z/OS High Availability Considerations</td>
<td>Bagwell</td>
</tr>
<tr>
<td>200</td>
<td>Thursday</td>
<td>12:15</td>
<td>Staged Application Development in a WebSphere ND Cluster</td>
<td>Loos</td>
</tr>
<tr>
<td>310</td>
<td>Thursday</td>
<td>1:30</td>
<td>WAS on z/OS and WLM Interactions</td>
<td>Follis</td>
</tr>
</tbody>
</table>
Agenda

- What are we talking about?
- Defining terms
- The basic flow
- How does WLM pick a servant?
- WLM-less queueing
- What about async beans?
- Hints about classification based on XML file
- How monitoring mechanisms work
What are we talking about?

Setting the stage and establishing baseline concepts
The CR / SR Structure ... One More Time

It's worth starting with a review of the essential heart of this:

---

**Controller Region**
- JVM
- Native Code

**Servant Region**
- JVM
- Native Code

**zWLM**
- Manages starting of SRs
- Manages stopping of SRs
- Requests queued to zWLM, then to SR

**AppServer**
- CR
- SR

**CR: WAS “Plumbing” Code**
- Native and Java
- No application code
- TCP listeners reside here
- Queues requests to WLM

**SR: Application Infrastructure**
- Maintains app JVM runtime
- May support one or more applications
- Connectivity to data from SR
- Min/Max controllable by admin

---

**General Properties**
- Multiple Instances Enabled
- Minimum Number of Instances: 2
- Maximum Number of Instances: 4

Default: min=1, max=1
What is "Workload Management" on z/OS?

It is controlled access to system resources coordinated by a function that keeps watch over all the elements of the system:

- z/OS Functions
- Middleware Functions
- Online Programs
- User Sessions
- Batch Programs

Access to the System Resources

Controls access based on defined goals

zWLM

Keeps track of what's being requested

Keeps track of how much is being used

There is a tight integration between the System z hardware, the z/OS operating system with WLM having an exclusive view of it all.
What About "WLM" on Distributed WAS?

The term "Workload Management" is used, but it's a different thing:

Exchanging some system metrics between WAS servers (that is, a middleware layer function)

Unlike other operating systems, z/OS is designed to only run on System z hardware ... very tight integration from HW up through OS.
Defining Some WLM Terms
Service Classes, Reporting Classes, Enclaves and Goals
Key Starting Concepts

To set the stage for the terminology that follows ...

Work seeking system resources

- Work Request
- Work Request
- Work Request
- Work Request
- Work Request
- Work Request
- Work Request
- Work Request
- Work Request
- Work Request

Work of differing importance and priority

In order for WLM to manage resources to goals, we must get the work organized into categories based on your goals.
The WLM Service Class

The "service class" is at the heart of this ... it's the container into which categorized work is placed.

WLM Service Classes

Classification (discussion coming up)

You may have more than four service classes ... the definition of these is up to you.

WLM "Goals" defined to each service class

From this it can work to manage the system resources to the goals. More detail upcoming ...
The WLM Report Class

The "report class" is a variation on the "service class" ... WLM uses it to **report** on activity, but **not to manage** resources.

Report Class
Ex: "Work related to WAS servers in cell ABCell"

Report Class
Ex: "Work related to CICS region XYZ"

Report Class
Ex: "Work related to transaction DEF"

Provides useful detail on things like CPU usage, zAAP usage and many other system statistics.

Generally speaking -- you'll have a handful of service classes and a lot more reporting classes ... based on your needs:

- Service Classes -- enough to reasonably categorize work priorities
- Reporting Classes -- based on the granularity of your reporting needs
Classification Rules

The next step is to get work associated with a service class and a reporting class. This is done with classification rules:

Classification Types
(in WLM panels)

- CB
- CICS
- DB2
- DDF
- IMS
- JES
- OMVS
- STC
- (others)

Translation: any started task that begins with "DF" will be assigned to the service class OPS_HIGH and the reporting class DFCELL. OPS_HIGH might have a goal of "Velocity 70%"...goals are next...

Standard WLM stuff ... we started with STC because it may be the easiest to understand for those not familiar with WLM processing.
Goals and Importance -- Defined in Service Class

Goals tell WLM what to strive for in terms of service; Importance is used to determine relative importance when resources tight

**Goals**

- **Velocity**
  - How fast work should be done without being delayed
  - Number 1 to 99

- **Response Time**
  - Percentage of work completed within a specified period of time
  - Example: 95% within 1 second

- **Discretionary**
  - WLM services when other priorities not competing for resources

**Importance**

1 = Most important
2
3
4
5 = Least important

Importance indicates how important it is to you that the service goal be met.
Importance applies only if the service goal is not being met.

- Started tasks and batch programs
- Online transactional work
- Work that's okay to push aside if resources are needed
The WLM "Enclave"

An "enclave" is a way to identify and manage individual pieces of work within the many parts of a running z/OS system.

Key points from this chart

- An "enclave" is simply a way for WLM to understand priorities at a work unit level
- WAS does this automatically ... if you do no other configuration it'll still do this with default values
The Basic Flow

From work into the server through the response back
What Work Gets a WLM Enclave?

There's a lot of work that goes on inside WAS z/OS. How much of it involves WLM enclaves? "Inbound Requests":

- Standard to JSP or Servlet, or SOAP for web services, or even the SIP protocol
- Call from another server in the cell, or from outside the cell
- Management flows, from within this CR or from Node Agent

Environment Variable:
```
wlm_classification_file = /mydir/myclass.xml
```

- HTTP
- IIOP
- JMX

Controller
- Enclave created, service class assigned

WLM Work Queues

Servants
- Enclave created, service class assigned

Optimized Local Adapters
- From external address space such as CICS or batch

Messaging Engine
- Traditional GET application or Message Driven Bean (MDB)

WebSphere MQ

Asynchronous Beans
- Yeah, a unique case from the others
Assigning a Service Class to the Enclave

This is for the work request ... earlier we saw how the CR was classified using the STC type. Now we look at the CB type ...

Subsystem Type CB - WebSphere z/OS CN and TC Classifications

Classification:

- Default service class is CBDEFLT
- Default report class is RWASDEF

<table>
<thead>
<tr>
<th>Qualifier</th>
<th>Qualifier</th>
<th>Starting</th>
<th>Service</th>
<th>Report</th>
</tr>
</thead>
<tbody>
<tr>
<td># type</td>
<td>name</td>
<td>position</td>
<td>Class</td>
<td>Class</td>
</tr>
<tr>
<td>1 CN</td>
<td>DFDMGR*</td>
<td>1</td>
<td>CBCLASS</td>
<td>DFDMGR</td>
</tr>
<tr>
<td>1 CN</td>
<td>DFSR01*</td>
<td>2</td>
<td>CBCLASS</td>
<td>DFSR01</td>
</tr>
<tr>
<td>2 TC</td>
<td>DFTRAN1</td>
<td>3</td>
<td>DFTRAN1</td>
<td>DFSR01T</td>
</tr>
<tr>
<td>2 TC</td>
<td>DFTRAN2</td>
<td></td>
<td>DFTRAN2</td>
<td>DFSR01T</td>
</tr>
<tr>
<td>1 TC</td>
<td>DFTRAN3</td>
<td>4</td>
<td>DFTRAN3</td>
<td>DFTRAN3</td>
</tr>
</tbody>
</table>

Enclaves created in WAS CR are classified by rules in CB subsystem type:

1. CN of DFDMGR* matches the Deployment Manager. Work there goes to CBCLASS.
2. Work in DFSR01* cluster without a transaction classification gets CBCLASS as well.
3. Work in DFSR01* cluster with TC of DFTRAN1 or DFTRAN2 get service classes as shown
4. Work that matches the TC of DFTRAN3 regardless of WAS CN gets service class DFTRAN3
5. Anything that doesn't match any specific rules gets the default service class of CBDEFLT
Enclave Propagation

We get to why all this enclave classification stuff is done -- so that WLM can manage the threads inside the servant regions.

1. If you don't want the enclave propagated into these target servers you may turn it off with the protocol_iiop_local_propagate_wlm_enclave = false environment variable.

2. What about CICS? CICS does its own classification so propagation from WAS to CICS not possible. But enclave propagation to DB2 over a JDBC T2 driver very possible, and the benefit is a single reporting "container" for resources consumed associated with the enclave.
How Does WLM Pick a Servant?

Hint: it's not random 😊
A More Precise Picture of the CR / SR Structure

Typically we draw only one WLM work queue between the CR and the SR. But in truth there are multiple:

Queues for each service class being handled by this application server ... but work without specific SR affinity

WLM Work Queues

Queues for work that must go to a specific servant -- "affinity"

Two questions come to mind:
1. If affinity, what creates the affinity?
2. If no affinity, then which servant gets the work?

Each appserver has its own set of such work queues
Affinity to a Specific Client:

Here's a brief overview of the flow creating affinity, then what happens for requests after that:

**Initial Request**

1. Works comes into CR and is classified as described earlier.
2. No affinity yet exists, so WLM places work on the work queue for that service class.
3. WLM indicates which servant should take the work. 
   We cover this in detail next.
4. Application creates an affinity, such as creating an HTTPSession object.
5. Response goes back with affinity key, which the CR keeps track of.

**Follow-on Requests**

1. Works comes into CR and is classified as described earlier. Affinity exists, so CR alerts WLM to that affinity.
2. WLM now puts the work on the specific work queue for that servant.
3. The servant takes the work off its queue.
4. Response goes back with affinity key; CR knows to maintain affinity.
Key Concept: Servants "Bound" to Service Class

Once a servant region has done work for a particular service class, WLM "binds" that servant to service class queue:

1. Works comes into CR and is classified as described earlier.
2. A WLM work queue for that service class is created.
3. A servant is chosen (next chart) ... enclave dispatched to a worker thread in that servant.
4. WLM now sees that servant as "bound" (or "associated") with that servant class.

Work for that service class will now go to that servant. Other service classes sent to other servants.

The key is how work gets allocated in the first place ... that's next.
Choosing a Servant -- One Service Class

Imagine a multi-servant application server (ex: MIN=3, MAX=3) where all the work coming is gets assigned to the same WLM service class.

**First Available**
- Enclave dispatched to first available thread
- That servant "bound" to service class (prev. chart)
- Work continues to that servant until threads occupied, then spills over to next servant
- If no threads immediately available, WLM places on service class work queue

**Round Robin**
- `wlm_stateful_session_placement_on = 1`
- WLM assumes every dispatch will create an affinity
- Seeks to balance affinities across servants bound to that service class.
Choosing a Servant -- Multiple Service Classes

Now imagine a multi-servant application server where the work gets assigned to multiple WLM service classes:

- Classification as discussed before
- Initial servant selection as discussed previous chart ... either "first available" or "round robin"
- Servants "bound" to service classes as discussed earlier
- Make sure number of servants equal or greater than service classes serviced

- It's important to understand how work is being classified -- you can "waste" a servant if a classification takes place you weren't anticipating (usually default service class is the problem)
How Threads are Managed in a Servant

It depends ...

Enclave Threads
- Work dispatched to servant from CR with an associated WLM enclave
- WLM manages the thread to the service class of the enclave
- Recall that servants are bound to a service class and generally serve only enclaves of that service class, but exception cases do exist

Non-Enclave Threads
- These are threads doing things like GC and other work
- These are managed according to the service class to which the servant region is bound

Special case -- "single servant mode"

Single Servant Mode
- WLM will mix different service classes into servant and manage each thread according to its servant region

Multi-Servant, MIN/MAX=1
- WLM will bind a servant to first service class that come in; other service classes will sit on the queue and eventually time out
Reporting CPU Usage

Where CPU is reported depends on whether or not it's an enclave thread, and if it was an asynch bean.

CPU for enclaves attributed to the Controller -- it created the enclave. This true despite fact the enclave is dispatched and run on a servant thread.

And ... if enclave propegated into DB2 over T2, then that CPU also attributed to the controller region where the enclave created.

For asynch beans ... it depends 😊

More on asynch beans in a bit.
WLM-less Queueing

WAS takes over some of the work from WLM
### Overview of WLM-less Queueing

It's based on the `server_use_wlm_to_queue_work` variable:

<table>
<thead>
<tr>
<th>If variable = 1 (default)</th>
<th>If variable = 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Uses WLM work queues</td>
<td>• WAS uses its own queues</td>
</tr>
<tr>
<td>• WLM controls dispatching to the servant region</td>
<td>• WAS controls dispatching to the servant region</td>
</tr>
<tr>
<td>• What we've discussed up to this point is how it works</td>
<td>• Three routing options: <strong>Discussed next page</strong></td>
</tr>
<tr>
<td>• Generally preferred for stateless workloads</td>
<td>• Generally preferred for stateful workloads</td>
</tr>
<tr>
<td>• Well suited for:</td>
<td>• Well suited for:</td>
</tr>
<tr>
<td>• Stateless +</td>
<td>• Stateful +</td>
</tr>
<tr>
<td>• multi-servant +</td>
<td>• multi-servant +</td>
</tr>
<tr>
<td>• multiple service class goals</td>
<td>• All requests have same service goal</td>
</tr>
</tbody>
</table>

InfoCenter for this and other custom properties, search: `urun_rproperty_custproperties`
Hot Thread, Round Robin and Hot Robin

These are the three routing options when that variable is set to have WAS control the routing.

Yet another customer property:

server_work_distribution_algorithm = 0 | 1 | 2

0 Hot Thread
- First available thread in the servant sequence list
- If no available threads, then onto the global queue and next idle thread (any servant) takes it

1 Round Robin
- Try to dispatch to next servant in the list
- If no idle thread, then place on dedicated queue

2 Hot Robin (7.0.0.7 and above)
- Try to dispatch to next servant in the list
- If no thread, then go to next servant in the list
- If still no threads, then place on global queue
- First available thread takes it
What About Asynch Beans?

They march to a different drummer ...
High-Level Overview of Asynch Beans

Here's a schematic diagram of how the CR / SR structure looks when asynchronous beans are introduced:

1. Classified work is dispatched to the servant per the methods already discussed. The servant thread joins the created enclave.
2. At some point the application requests of the work manager that an asynch bean be started.
3. At some point the asynch bean is started. It receives a thread out of the thread pool maintained by the work manager.
4. The original work completes and returns -- the asynch bean may or may not yet be launched; if launched it may or may not be complete.

What about this? How is it classified? What enclave does it join?
Asynch Beans -- Three Scenarios

*Much* depends on *how* the work manager is called:

**If** `isDaemon=true` **passed in on** `startWork` **API, then ...**
- Asynchronous bean considered a very long running process ... potentially forever
- A new thread is created rather than pulling from the work manager thread pool
- A new enclave is created with classification based on "Daemon transaction class" defined under *Resources ⇒ Asynchronous Beans ⇒ Work managers* in the Admin Console
- If no Daemon transaction class defined, then `ASYNCDMN` is used

**If** `WorkWithExecutionContext` **specified on** `startWork` **API, then ...**
- The work manager calls a WLM API and gets the classification attributes for the original work request
- A new enclave is created with the same classification attributes as the original request

**If** execution context **not** set on `startWork` **API, then ...**
- The work manager registers with WLM as a "user of the original work request enclave"
- That allows for the original work request to complete but the enclave to stay in existence
- The asynchronous bean operates under the classification attributes of the original work request enclave

**If** asynch bean scheduled from non-enclave threads, then ...  
- There is no original enclave to work with
- A new enclave is created with classification based on "Default transaction class" defined under *Resources ⇒ Asynchronous Beans ⇒ Work managers* in the Admin Console
- If no Default transaction class defined, then `ASYNCBN` is used
Using the Classification XML File

InfoCenter, search on `rrun_wlm_tclass_sample` for a sample
How it Works

The file supplies a set of criteria to match requests to transaction class names, which then match with rules in the CB subsystem type.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Classification SYSTEM "Classification.dtd" 
<Classification schema_version="1.0">
  :  
  <InboundClassification type="iiop" ... >
    (classification information)
  </InboundClassification>
  <InboundClassification type="http" ... >
    (classification information)
  </InboundClassification>
  <InboundClassification type="sip" ... >
    (classification information)
  </InboundClassification>
  <InboundClassification type="mdb" ... >
    (classification information)
  </InboundClassification>
  <InboundClassification type="sib" ... >
    (classification information)
  </InboundClassification>
  :  
</Classification>
```

Scope to cell or node
server scope for classification deprecated

From that we get goals and importance based on specific transactions based on criteria in the classification XML file.
Some Hints

The file supplies a set of criteria to match requests to transaction class names, which then match with rules in the CB subsystem type

**IIOP**
If you classify at the method level, use the mangled method name. You can find that in the generated stub or tie.

**HTTP**
URI is commonly used, and wildcarding is allowed. Match on host and port also possible.

**SIP**
There's nothing in a SIP request to match on, so the classification is somewhat binary ... "if SIP, then transaction name is ..."

**MDB**
For "Plan A" MDBs (persistent durable queues received from MQ via the controller's message listener port) you can classify under the MDB type.
For "Plan B" MDBs (listener in the servant) the classification falls under "internal"

**SIB**
Type "jmsra" applies to MDBs which use the default message provider
Type "destinationmediation" applies to mediations defined on the SIBus

**Internal Work**
There's work that WAS itself needs to do. This is where it's classified (along with MDB Plan B)

**Optimized Local Adapters**
Handled in a special way. Go to the InfoCenter and search on tdat_olawlm
How's My Work Being Classified?

Some hints and tips on determining classification results
Some Available Tools

• **WLMQUE**
  
  A TSO-based tool that displays each application environment and information about the servant regions associated with it. Download the tool and documentation at:
  
  `ibm.com/servers/eserver/series/zeries/zos/wlm/tools/wlmque.html`

• **RMF**
  
  • IBM's tool to report on activity on z/OS. There are others....

• **SMF 120.9**
  
  • The WebSphere SMF record contains an abundance of information about what requests are run
  
  • This includes the data used with the XML file to classify the request
  
  • Also which servant region the request was dispatched in and whether it was dispatched with affinity

• **SMF 120.9 browser with plugin**
  
  • There is a sample plugin provided with the Java browser that can generate a sample classification XML file based on the work you are running
Common Problems

Some things to watch out for
Common Problems We've Seen

• Work not classified as expected
  • This can result in requests stuck in the queue and other problems. Use the tools on the previous chart to see what's up.

• Enclave propagation causes an unexpected service class
  • A server may have enough servants for all the service classes you expect, but an enclave propagated from another server might a different service class

• WLM round robin behaves oddly
  • Remember WLM is balancing affinities, not just round-robin
  • The balancing is among servants bound to the same service class – an unexpected service class can prevent WLM from using all the servants
Common Problems We've Seen

- Only one servant with multiple service classes
  - Setting min=max=1 instead of single-servant prevents WLM from scheduling different service classes into your only servant, leaving requests stuck in the queue

- Defaulting to discretionary
  - Unexpected work or mis-classification can result in a default of discretionary which usually runs very very slowly.