This page intentionally left blank
Starting Out ... a High-Level Look at Things

Imagine you’re zManager and you want to manage workloads that jump across platforms. What information would you like access to?

- Insight into the hypervisors to see state of the physical resource utilization across virtual servers
- Some interface into the virtual servers operating systems to be able to see and manage them
- A way to identify and group work into a collection meaningful to the human administrators
- A way to quantify the relative priorities between work that has been grouped into workloads
- A way to adjust resources to meet defined goals about relative priorities

We start out this unit by asking you to imagine you’re zManager and you’ve been given responsibility for managing workloads across different virtual servers on the zEnterprise system. What types of information do you imagine you would need?

The chart suggests a few things that would help:

- The ability to gather performance information from the hypervisors so the some knowledge of what virtual servers are operating and what consumption of the real resources are taking place at the level of each virtual server
- Some awareness of what’s taking place within the virtual servers would be helpful. This would include information about the operating system and what that OS sees as its resource utilization
- You would need some way to understand what constitutes a workload and how to identify work as being associated with one workload or another.
- You would need some way to understand what the relative priorities were between defined workloads on the system. Everything can’t be “high priority” because then nothing is; and if everything is of equal priority then the value of differential workload management is diminished. So the ability to discern what those relative workload priorities are would be important.
- If you had the responsibility of trying to manage the system so the defined priorities were met, you would need some way to make adjustments to meet those goals. Reporting on the attainment of goals is one thing, managing to those goals is the next step.

As you might well imagine, we will now take a tour through the zManager’s “Platform Performance Management” function to show you how those pieces of information are provided to zManager so it can go about managing to the performance goals.
Insight into Hypervisor Activity

zEnterprise manages the hypervisors so it can have microcode probes to “see” what’s going on in the various hypervisors in the system:

This is why zEnterprise imposes some requirements on System z levels and why zManager controls the loading of hypervisors ... so it can “see” into the state of the virtualization taking place across the defined virtual servers

One of the reasons why zManager imposes some restrictions on what hypervisors may participate in the ensemble is because there’s a great deal of information exchange going on between zManager and the hypervisors across the internal management network (the INMN). Therefore, zManager has to be certain it can (a) communicate with the hypervisor, which involves understanding and using the interface of the hypervisor, and (b) in some cases influencing the activity of the hypervisor to manage it.

At a minimum zManager is capable of receiving performance information from the hypervisors so it can accumulate and report on activity. All the supported hypervisors – PR/SM, z/VM and PowerVM all do at least this level of reporting.
Insight into Virtual Servers

This requires a probe into the virtual server operating system. That is called “GPMP” – Guest Platform Management Provider:

GPMP is a lightweight management probe into the OS of the virtual server. It’s not strictly required, but it does provide some benefits as we’ll see.

You’ll enabled GPMP on your AIX virtual servers in the upcoming labs
We enabled GPMP on z/OS before the workshop.

While the hypervisor is capable of seeing what CPU and memory is being used by the virtual servers it is supporting, the hypervisor can not necessarily see up into the operating system that is running there. To provide the additional insight into the OS in the virtual server a lightweight management probe called “Guest Platform Management Provider” is supplied. This provides additional information to zManager about the activity in the virtual server itself.

We’ll explore GPMP more in a bit. For now it’s worth noting that while GPMP is not strictly required – many of the zManager management functions can work without it – it does play an important role in zManager’s ability to see end-to-end response time statistics (ARM, which we’ll cover in a bit as well). It also plays an important role in integrating the performance goal definition information from zManager into zWLM running on z/OS.

In the upcoming hands-on lab you’ll have the opportunity to enable GPMP in your AIX virtual servers. It’s a relatively simple task as you’ll see. As part of the lab exercise GPMP on z/OS will also be used.
We enabled that on z/OS ahead of time since we have one z/OS LPAR that is part of this workshop.
Identifying, Classifying and Managing Work

Different work often has different performance priorities. To differentiate work it’s important to first identify it … then you can assign performance goals

In any given system of virtual servers there’s a lot of work taking place. Before zManager can begin to manage work to any defined goals, it must first have some way of understanding how to logically group virtual servers into associated workloads, and how to identify that a given piece of work that starts up or enters the system is part of a defined workload.

This is a fairly large and potentially slippery topic if you don’t have some background in workload management. So when we get to that section in this unit we’ll cover how “workloads” are defined, how you indicate the relative priorities between activities, and how you help zManager identify work and assign it to your workload and priority definitions.

We’ll offer a very cursory peak at this point, then go into more detail later:

- A “workload” is a collection of virtual servers logically grouped into a definition that zManager will consider running related work.

- A “service class” is a definition that provides zManager some understanding of the priority you wish to assign work that runs under that service class.

- “Classification rules” tell zManager how to understand which service class to assign work to. When the work is assigned to the service class, then zManager may begin to understand how to manage it to the defined goal.

z/OS is a considered a “virtual server” to zManager and as such a z/OS LPAR may be part of a broader workload definition comprised of other z/OS LPARs as well as non-z/OS virtual servers. Therefore it’s important we have a way to tell WLM in z/OS about the priorities of the work in a workload its participating in. How this is done will be explained later as well.
Reporting on Relative Goal Attainment

One element of this is simply reporting on how a group of work is performing against the goals you’ve defined:

<table>
<thead>
<tr>
<th>Work</th>
<th>Priority</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABC</td>
<td>Highest</td>
<td>Not Meeting Goals</td>
</tr>
<tr>
<td>DEF</td>
<td>Medium</td>
<td>Meeting Goals</td>
</tr>
<tr>
<td>XYZ</td>
<td>Low</td>
<td>Exceeding Goals</td>
</tr>
</tbody>
</table>

Knowing this brings you one step closer to having better control over the environment.

With this information in hand it would allow you to at a minimum take a look at resource allocations (CPU, memory) and make manual adjustments.

Reporting on performance against defined goals is the foundation of zManager’s Platform Performance Management functionality.

Many more details to come ...

If the zEnterprise was simple a collection of virtual servers on blades with no other management functionality, you would be able to see basic performance information as reported by the hypervisors. But that would not necessarily provide you with a good sense of how groupings of work important to you are performing relative to how you need them to perform.

This forms the starting point for our discussion of Platform Performance Management. zManager provides a mechanism for defining work you consider to be part of a group meaningful to the business. It then allows you to assign that work a relative priority of how important that work is to you and the business. With that in place, zManager may then collect performance data across the zEnterprise and report on how your workloads are performing against your defined goals.

**Note:** We have a lot of detail to go through before all the mechanics of this make better sense. For now we’re establishing an important conceptual foundation for those details.

If zManager were to stop right here this would be meaningful information. For it would provide you insight into how resources are allocated across your virtual servers. A workload with a high priority that is not meeting its goals is a warning sign. A low priority set of work that is exceeding its goals tells you perhaps the virtual servers doing that work have more resources than they actually need. So at a minimum this reporting knowledge would provide you information you could use to analyze further and make manual resource adjustments within your zEnterprise.

But zManager doesn’t stop here. zManager has the ability to make those adjustments for you automatically, depending on the hypervisor. That’s next.
Making Adjustments to Meet Performance Goals

This brings us to the heart of the matter – zManager’s ability to detect when goals are not met and making adjustments to attempt to meet goals.

I am aware of the environment, and I am aware of your defined workloads and goals. I will adjust to try to meet your goals.

zManager can shift CPU between virtual servers* within a hypervisor to meet the workload goals defined. We explore how some of this accomplished as we proceed through this unit.

This picture is obviously simplified … there’s a great deal of sophisticated monitoring, data collection and analysis that’s going on behind the scenes.

Once zManager has an awareness of how work is logically organized and what the priorities are, it may be to manage to those goals by shifting resources. This is done between virtual servers on a hypervisor. To accomplish this certain things need to be in place (“Processor Management” enabled at the hypervisor and virtual server level, and virtual server CPU defined as “shared”).

Bear in mind that zManager will not likely be managing a single workload, but many. So there’ll be a great deal of competition for the system resources. zManager is designed to understand and balance it all using algorithms based on knowledge of system workload management learned over the years by IBM.

The point here is that while zManager does indeed report, it also manages. And to manage the workloads implies re-allocating resources where zManager has been given the ability to do so.
Tracking Overall Response Time

This brings another piece to the table – ARM. ARM provides a way to correlate response times across different platforms:

Multi-tiered workloads traverse multiple virtual servers, and processing time and network latency factors into the overall response time seen end-to-end. How would zManager have any idea what makes up a flow and what response times are seen at each hop?

The answer is ARM – Application Response Measurement … an open standard mechanism for recording and correlating the time spent in each hop of a distributed workload. It accomplishes this by passing correlator tokens along with the payload of traffic that flows from hop to hop in a tiered topology. zManager is capable of understanding this information and managing based on that information.

ARM, like GPMP, is not strictly required. It is, however, potentially helpful.

More on ARM coming up.
Role of the “Automate” Suite

Brief reminder of the two suites and where they fit in this discussion:

“Manage” Suite

- Provides the majority* of the function represented by this graphic pie chart
- With Manage you can monitor resource utilization at the hypervisor and virtual server level, but you can’t automate it with goal-oriented management.

“Automate” or “Advanced Management”

- Optional suite of function that provides the goal-oriented performance management we’ll cover in this unit
- Advanced Management covers System x blades, Automate all else.
- No charge feature code (FC0020) for z/OS, but carries a charge per connection for power blades (FC0045) and IFLs (FC0052)

Going forward in this unit we assume both Manage and Automate is present on the zEnterprise system

Now we get back to our pie chart showing the six categories of management activity and the two “suites” we spoke of earlier – “Manage” and “Automate.”

To accomplish the kind of automatic and dynamic goal management requires that you have the “Automate” suite licensed to the elements of your zEnterprise system. It’s a no-charge feature to z/OS but carries a charge for Power blades.

While it is possible to do some level of monitoring with just the “Manage” suite, what we’ll discuss in this unit requires “Automate” … so going forward we assume that is present.

* There is a piece of the “Energy” function that requires “Automate” – Static Power Savings
GPMP
Understanding the role of the Guest Platform Management Provider

Now let’s explore GPMP …
**GPMP and What it Provides**

GPMP provides zManager a better view up into the virtual server:

If you use ARM, GPMP is what feeds the ARM information to zManager so it can report on response times and draw the workload topology diagram. We’ll see examples of those reports later in this unit.

If you wish to define workloads and classify work, GPMP provides additional classification criteria beyond simply virtual server name. More on workloads and classification coming up.

If you wish to associate zManager service class names with z/OS WLM service classes, you require ARM and GPMP on the virtual server and z/OS. More on how zManager intersects with zWLM later in this unit.

So … not strictly required, but it does provide benefits at a relatively low cost in processing overhead.

As noted earlier, GPMP is a light-weight management agent that runs inside the operating system of the virtual servers in a zEnterprise system. It feeds information from the OS-level to zManager over the INMN internal management network.

Strictly speaking, GPMP is not required. You can do dynamic workload management without it. But keep in mind the following things when considering whether to enable GPMP or not:

- The ARM function we briefly spoke of a few charts back is dependent on GPMP to forward its response time information along. zManager uses this information to create the Virtual Server “topology” report and “hops” report we’ll see at the end of this unit. Again, not strictly required, but those reports and ARM itself do provide some value. So if you’re interested in ARM, you require GPMP as well.

- Without GPMP the classification of work is based on the virtual server name only. But with GPMP you have several more options available to use when classifying work, such as the OS name, the OS type and the host name of the OS in the virtual server.

- The ability to propagate zManager service class information into WLM on z/OS so it may associate the work with a WLM service class is dependent on GPMP and ARM being present not only on the distributed virtual servers but z/OS as well. Work will still take place without this, and WLM on z/OS will still classify work without this, but the integration of zManager-to-WLM service classes will not be nearly as tight as it is with ARM and GPMP.

So, as we prepare to dig into GPMP keep in mind that it is helpful, but not required. Ultimately the decision to use or not use is yours.
GPMP and Eligible Virtual Server Types

GPMP is software that is enabled in the operating system of the virtual server managed by zManager. Eligible virtual server operating systems include:

- Red Hat Enterprise Linux (RHEL) 6, 5.5, 5.4, or 5.3
- Novell SUSE Linux Enterprise Server (SLES) 11 or 10
- z/OS V1R12, V1R11, or V1R10 with PTFs
- AIX 6.1 TL5 or AIX V5.3 TL12
- z/VM is itself a hypervisor as well as an operating system. It does not itself run GPMP. However it does participate to a degree in zManager functions. z/VM 6.1 with PTFs is required

In this workshop we focus on z/OS and AIX

We’ll show you how it is enabled on z/OS and AIX

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

GPMP is set of Java code that provides the agent functionality. It is available on the operating systems shown on the chart above.
GPMP Enablement on AIX
Some of this you did in the first lab, some you'll do in the upcoming lab ...

Enable GPMP in Virtual Server

Enable IPv6 in Virtual Server
GPMP uses INMN to feed zManager. INMN is IPv6.

Then in AIX:
• Mount virtual media to virtual server
• SMITTY to enable eWLM services
• Use `rpm` to install AIX GPMP package
• Start GPMP and set to autostart next boot

In summary a UNIX install of a small Java software package that will act as a management interface for zManager into the virtual server operating system

In the upcoming lab you’ll get a chance to enable GPMP on your AIX virtual servers. There are two parts to this:
• Indicating GPMP support on the virtual server details at time of creation. Perhaps you recall us calling this aspect out in the lab instructions.
• Then when AIX is up and operational doing a few relatively simple tasks to make GPMP operational.
Which AIX Virtual Servers?
It is not necessary to GPMP-enable every virtual server. In the lab we have you enable two of the four AIX virtual servers:

- We want GPMP on the WAS and IHS servers because we will be defining workloads that include those virtual servers. More on workloads coming up.

GPMP is required to feed ARM* information, which is required to associate the distributed priorities with zWLM goals. But it is not strictly required for some zManager functions. zManager still has a view into the hypervisor and can see CPU usage between virtual servers.

Earlier you created four virtual servers, but in the upcoming lab we’re going to have you enable GPMP on only two of them. Why?

Because for two of the four virtual servers in our lab setup GPMP provides no real net-benefit. Those two servers are the JMeter driver server and the Co:Z “donor” server.

- The “driver” server has dedicated CPU and is intended to run the driver software to simulate the users to the system. zManager will not be managing this virtual server in any meaningful way – we won’t have zManager add to or subtract from its CPU allocation since it’s dedicated CPU. And because we want these simulated users to appear as if coming from the outside world, this JMeter virtual server will not be considered part of a workload. So, no real reason to enable GPMP on it.

- The Co:Z server, or “donor” server, will serve to give up CPU in the upcoming lab to the HTTP server and WAS server as part of zManager’s management of the overall workload priorities. We could have enabled GPMP, but it’s not required … zManager can still take CPU from the donor server and give to the receiving servers without GPMP enabled.

Why did we choose to enable GPMP on the HTTP virtual server and the WAS virtual server? Because we wished to incorporate ARM into the mix, and ARM requires GPMP to act as the transport agent for ARM information to zManager.

Why did we wish to include ARM? Because that’s what allowed us to integrate the zManager workload with z/OS WLM classification to service classes there.
GPMP on z/OS -- Overview

Gives zManager insight into the z/OS operating system as well as enabling collaboration between zManager workload service classes and zWLM.

You do some setup work to enable GPMP to operate on z/OS LPAR:

- Some RACF work, creates some logging directories

z/OS R12 and above provides some additional fields in WLM panels for GPMP:

- If before z/OS 1.12 then no automatic start of GPMP on z/OS ...
  - F WLM,GPMP,START

WLM starts GPMP using supplied JCL proc:

- If before z/OS 1.12 then F WLM,GPMP,START

GPMP runs as an address space.

Validate with WLM display:

- D WLM,SYSTEMS,GPMP

As we noted, we enabled GPMP on z/OS prior to the workshop because there’s only one z/OS LPAR in play for this workshop and having multiple teams try to enable the same function at the same time is problematic. So we went ahead and enabled it ourselves in prep for the workshop.

The process of enabling GPMP on z/OS involves a small handful of system programmers tasks:

- Some customization of supplied JCL jobs that create RACF profiles and some USS file systems, as well as the customization of a start proc for GPMP
- If z/OS R12 or above the definition of automatic start within WLM itself

When all is done you’ll see GPMP running as a single address space with the default jobname of HVEMCA.
Let's turn our attention to ARM …
Very High Level Overview of ARM

ARM stands for “Application Response Measurement” and it is a mechanism for tracking response time across multiple servers in a distributed workload:

Open standard -- www.opengroup.org/tech/management/arm/

Provides a way to track and correlate time spent in each “hop” of a distributed workload that spans servers

Requires application or middleware to actively exploit the ARM interface

Information in correlators may be used by tools (zManager) to report on application response time as well as paint a picture of the workflow topology

ARM is a mechanism intended to create correlator tokens so information about work that passes from server to server may be part of the exchanges … added to and recorded along the way.

It’s an open standard … you may investigate further at the URL above if you wish. This is not some zManager-only or IBM-only thing.

ARM came about by the recognition that more and more workloads in today’s world were comprised of processes on disparate servers separated by network. Understanding the end-to-end response time of such a system is difficult without a very close understanding of the pieces of the puzzle that made up the workload. So clever people came up with a way for the middleware and application level to call a function to pass along time spent at each “hop” in the process. This is done in something called a “correlator token.”

Now truth is, unless you’re a middleware developer interested in coding to the ARM APIs you probably don’t need to worry about much more detail than this. The key is that many vendors are enabling their middleware to understand and use ARM. IBM is one such vendor. Many key elements of IBM’s middleware portfolio is “ARM-enabled,” including WebSphere Application Server and DB2 Universal Database, including that which runs on z/OS.

zManager is fed information about ARM metrics through the GPMP agent on each hop. With that zManager is able to better understand the topology of your workload and better understand the time spent at each hop and overall. zManager prepares two reports based on ARM data. We’ll see them next.
Is ARM Required?

Strictly speaking, no … but it provides zManager with information about overall transaction performance, and it allows association to zWLM service class.

**Virtual Server Topology Report**

Shows virtual servers participating in an end-to-end ARM-enabled transaction.

**Hops Report**

Shows the hop information along with successful transactions and average response time at each hop.

**zWLM Service Class**

Allows you to associate a zManager workload service class to a defined zWLM service class so coordinated performance management is possible. More on this coming up.

Just like GPMP, ARM itself is not strictly speaking required … depending on what you’re looking to do. With ARM enabled zManager is able to discern and portray the server topology of a workload. It can see what virtual server is the first touched in a workload, what the intermediary servers are and who the last server is. Further, it can accumulate statistics on the various hops – the number of requests, and the processing time at each step.

Further, with ARM enabled then the zManager service class name can be passed into WLM on z/OS so WLM can associate the work with a service class of your creation that aligns with the priority defined in the zManager server class. This is simply one more way to more tightly integrate workload priorities across disparate systems.
IBM ARM-Enabled Middleware

Here’s a quick summary of ARM-enabled middleware:

- **IBM WebSphere Application Server**
  - All platforms ... V6, V7 and V8

- **HTTP Server via WAS Plugin Function**
  - IHS, IIS, Domino, iPlanet

- **IBM DB2 Universal Database**
  - Including on z/OS

In the upcoming lab you’ll turn on ARM for IHS and WAS:
- IHS – modify XML file
- WAS – radio button in Admin Console

Here’s a survey of ARM-enabled middleware from IBM. Not coincidentally you’ll be using all these components in the upcoming lab.

- The HTTP Server is capable of hosting a piece of code within it called the “WAS Plugin.” This “Plugin” is code that intercepts inbound HTTP requests and determines which backend WebSphere Application Server is hosting the requested application. For the purpose of ARM enablement, the Plugin is what’s ARM-enabled. It simply runs inside the HTTP Server.

- WebSphere Application Server is also ARM enabled. Typically WAS serves as the second “hop” in a three-tier application design.

- Finally, DB2 is ARM-enabled, including DB2 on z/OS.

The act of enabling ARM on the components is relatively simple:

- For the Plugin running in HTTP, this is as simple as modifying the `plugin-cfg.xml` file, which you’ll do in the upcoming lab.

- For WAS it’s as simple as drilling into the Administrative Console and selecting a radio button.

- For DB2 on z/OS, which we performed ahead of time, the instructions are well documented in the “Ensemble Performance Management Guide” (GC27-2607) under the heading “Configuring ARM for DB2 on z/OS.”
Workloads and Classification
Helping zManager understand your priorities

Now we’re ready to discuss the broader and potentially more slippery topic of workloads and workload classification.
The zManager “Workload”

A collection of virtual servers with a defined set of priorities. zManager attempts to adjust resources to meet defined priorities:

- Workload: WORKAB
  - Virtual Servers A and B
  - Service classes with classification rules

- Workload: WORKCD
  - Virtual Servers C and D
  - Service classes with classification rules

- Workload: WORKEF
  - Virtual Servers E and F
  - Service classes with classification rules

- Workload: WORKG
  - Virtual Servers G
  - Service classes with classification rules

Preliminary points … then we go on to more details:

- The objective is to give zManager some idea of how you’ve organized work on the system
- Multiple workload definitions may exist … and virtual server overlapping is permitted
- Each workload carries with it service classes which defines the priority of the workload
- Each service class carries with it a classification rule to associate work with a workload

The first key concept to get on the table is the zManager “workload.” A workload is a collection of virtual servers that together comprise a set of work you deem as related to one another. And to this workload you define a set of priorities.

Image a scenario with two blade servers, each with four virtual servers defined. Your first workload consists of virtual server A and B combined to form workload “WORKAB.”

Associated with that is a service class which defines the relative priority and rules that help zManager determine what work is associated with this workload. We’ll cover those next. For now just accept that something called “service classes” and “classification rules” exist.

Image virtual server C and D and associated with workload WORKCD. E and F combine to form WORKEF and G is by itself in workload WORKG.

If we stop right there we see that we’ve logically organized the work in servers A and B into a collection called WORKAB, C and D into a logical collection called WORKCD, etc. This is where things get a little slippery – multiple workload definitions are permitted (we demonstrate that here), and overlapping of virtual server assignments to workloads is also permitted (we do not demonstrate that here to keep things simple for now).

Next we have to define what “service classes” are and how rules are used to assign work to the service classes, which reside in the defined workloads.
Classification Rules and Service Classes
This is what associates work with a workload and provides zManager and understanding of what work running where has relative priority.

If Virtual Server = A or B
- Name: MyService1
- Performance Goal:
  - Velocity
  - Fast
- Business Importance: High
- Classification Rule
  - If Virtual Server = A or B

If A, B then
- Velocity, Fast, Importance=High

If B, C then
- Velocity, Medium, Importance=Medium

If E, F then
- Velocity, Highest, Importance=Highest

If G then
- Discretionary

zManager weighs all this as its watching hypervisors and virtual server CPU utilization.

“Service Classes” and “Classification Rules” need to be discussed in tandem.

A “Service Class” is a definition of the relative priority work is to be given. For zManager the performance goal is based on a concept of “velocity” which we explain on the next chart. The values for velocity include five steps from “fastest” to “slowest.” As the names indicate, a performance goal of “velocity=fastest” implies it must have better performance than “velocity=slowest.”

Business importance is another means of weighing one priority versus another. The value for that is “Highest” to “Lowest.” We’ll explain “Business Importance” on the next chart as well. For now, consider Velocity+Importance to be how zManager weight relative priority against other definitions.

A service class definition is not worth much unless we can get work assigned to it. That’s what the “classification rule” does – it tells zManager how to associate work it sees with a defined service class. The easiest starting rule to consider is based on the virtual server. In the chart above -- If a piece of work is initiated in virtual server A or B, then consider it part of service class MyService1 and assign it a velocity goal of Fast.

Here’s where things get a bit more complex … there will likely be multiple workloads with multiple service classes and classification goals … and zManager has to keep all that in its head while it manages goals. So zManager is watching over it all … it has all the defined workloads and service classes and classification rules loaded up in its memory nad it’s watching over all those definitions plus watching over all the virtual server performance metrics.

And it’s balancing all the relative priorities against all the actual performance metrics it’s seeing.

And it’s attempting to shift resource usage to accommodate all those different workloads and priorities. Lots of complex math and algorithms going on under the covers. It’s not magic, it’s just sophisticated computer code watching and balancing it all.
Pause … Explain Two Key Concepts

We’re just showed terminology related to defining performance priorities. We need to explain those:

“Velocity” as a performance goal

A relative measure of performance for a virtual server. At a high level it’s a measure of how quickly work gets access to system resources and completes relative to other work on the system.

Options are: Fastest, Fast, Moderate, Slow, Slowest

- **Fastest** = work should get access to system resources quickly so it can complete
- **Slowest** = work may wait if other higher priority work is consuming the system resources

“Business Importance”

This provides a zManager with a sense for relative priorities when the performance goals themselves aren’t enough.

This value is combined with performance goal as part of zManager’s broader algorithm for determining performance results.

Options are: Highest, High, Medium, Low, Lowest

We threw two terms on the table in the previous chart that we need to pause and explain – “velocity” and “business importance.” For those who are familiar with WLM on z/OS these concepts will not be new ... many of the z/OS WLM terms and concepts are in play on the zManager.

“Velocity” is a term used to define a relative measure of access to system resources so work can begin execution and complete. It’s a form of measure of time spent waiting for access to resource. Work waiting for resources is not executing ... it is waiting. Work executing is doing just that – executing ... and it’ll execute as fast as the work is capable given access to the resources. So the key is not how fast it executes with full access to the resources ... the key is how much time is spent waiting to get access to the resources.

A velocity of “fastest” means the work should not wait for access ... if at all possible it should be given priority to access so it can begin execution and complete. Work with a velocity goal of “slowest” has been deemed to be of lesser priority so it can wait, if needed, for resource access. If work with “fastest” is competing with “slowest,” then “fastest” will, on average, be considered higher priority for access to resources.

“Business Importance” is a way to indicate to zManager how to further differentiate priorities when the velocity goal is not enough. Five degrees of “importance” are possible – Highest to Lowest.

Combined, velocity=fastest-to-slowest and importance=highest-to-lowest, this provides 25 degrees of relative importance by which zManager can weigh work against resources in an attempt to balance it all and meet the goals defined.
## Service Class Goals and Business Importance

Performance Goals and Business Importance form a very granular grid of relative priority you define for any given service class:

<table>
<thead>
<tr>
<th>Velocity Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fastest</td>
</tr>
<tr>
<td>Fast</td>
</tr>
<tr>
<td>Moderate</td>
</tr>
<tr>
<td>Slow</td>
</tr>
<tr>
<td>Slowest</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Business Importance</th>
<th>Service Class A</th>
<th>Service Class B</th>
<th>Service Class C</th>
<th>Service Class D</th>
<th>Service Class E</th>
<th>Service Class F</th>
<th>Service Class G</th>
<th>Service Class H</th>
<th>Service Class Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>Highest</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>Z</td>
</tr>
<tr>
<td>High</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Medium</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Low</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lowest</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A performance goal of “Discretionary” falls below even “slowest”

zManager watches all the work and weighs all these priorities as it works to manage the overall resources to meet your goals.

This chart is meant as a very conceptual display of how the velocity x importance definitions can be combined to form a fairly granular grid of relative priorities.

The two extremes are Fastest/Highest (Service Class A) and Slowest/Lowest (Service Class Z).

From there we can expand the 5x5 matrix to include all the combinations of Velocity and Importance and show how service classes might fall within the matrix.

For example “B” would have relative priority over “D.” “B” over “C”. What about “D” and “E” … well, that’s where zManager algorithms come into play. On the chart those two are a bit of a toss-up. But to zManager, which is just a computer program, there are algorithms at work to determine the relative priority.

One performance goal we’ve not yet mentioned is “discretionary.” That falls below the lowest combination of velocity and business importance. That does not mean it’ll never get executed. It just means that relatively speaking zManager will consider it lower than even “Slowest/Lowest.”

Once again … it’s important to understand that zManager is weighing potentially hundreds of service class priority ratings against the system resource utilization it sees in an attempt to balance the goals to the available resources. Simple algebra? Hardly. But pure magic? No.
Performance Index (PI)

For every service class zManager reports on how it sees actual performance against the defined goals for the service class:

- \( PI > 1 \) Service class not meeting velocity goal
- \( PI = 1 \) Service class meeting velocity goal
- \( PI < 1 \) Service class overachieving velocity goal

Number computed relative to all other service classes active on the zEnterprise

Example of Service Classes Performance Index Report

- Starts out not meeting its performance goal
- Over time zManager makes adjustments to help service class meet its performance goal
- In this example zManager was able to help service class achieve performance goal
- \( PI = 1 \), or the goal

You will see this in action in the upcoming lab

zManager provides a single number to express the degree to which any given service class is meeting the goals defined for that service class. The number is called the “Performance Index” (PI) and it’s a relative measure of accomplishment against defined goals vs. all the other service classes in play on the system. A PI value greater than 1 means the goals are not being met; 1 means the goals are being met; less than 1 means the goals are being exceeded.

A single service class operating by itself on a set of server resources will likely exceed goals. Lots of service classes overcommitted on a server box will likely result in the PI number exceeding 1.

If we cycle back to the concept of zManager managing system resources to meet goals, we start to see what takes place – if zManager sees that a service class is not meeting its goals, and it sees that other virtual servers have service classes of lesser priority, then it can shift CPU from one virtual server to another in an attempt to balance the resources to priorities. Over time it can get the PI number to drop closer to 1, which is “meeting goals.” That’s what you’ll do in the upcoming lab.

Will it do this every time? Maybe not … depending on the system resources and your defined goals. If you define everything as velocity=fastest and importance=highest then zManager doesn’t have much room to work. But if you do a fair job of balancing the definitions to true business needs, then zManager has a chance to balance the system resources to your goals.
Performance Policies
Provide a way to dynamically shift a group of service classes in and out of effect within a defined workload:

From the HMC you may dynamically activate a defined performance policy
Provides a way to change performance management profiles easily and dynamically

Now we layer another component on this … “Performance Policies.” Consider “Performance Policies” to be like buckets in which service classes are held.

The key is this – for any given workload only one performance policy can be active. Others may be defined but inactive.

The purpose of this is to allow you to define different sets of service classes for different business conditions you might see … weekday vs. weekend or day vs. night. This then allows you to have definitions that can be toggled –dynamically – when you need to put in place a different set of service classes. This avoids the re-coding of service class priorities. You simply code the different policies and activate them as needed.
Preliminary Summary of Workloads

Let’s step back and collect all the concepts into a single summary picture:

- **Virtual servers** are added to the workload definition.
- The workload has a default performance policy with a default service class.
- Your performance policy with one or more service classes, each with a classification rule along with a default service class for unclassified work.
- Multiple performance policies permitted.
- Multiple workload definitions active concurrently.

Here’s the summary picture … a picture we deliberately avoided showing at the start of this discussion. The message is:

- A workload definition starts by having some number of virtual servers associated with it.
- Every workload has a default performance policy and default service class associated with it. This is where work would go if this performance policy is active and the work is not otherwise classified.
- Your performance policy will have between one and many service classes associated with, including a default service policy. The desire is work is classified to one of your service classes using the defined classification rules. But if not, the default service class is there to “catch” work otherwise not classified.
- Multiple performance policy definitions are permitted … but remember, only one may be active at a time.
- Multiple workload definitions – each with performance policies and service classes may be active at a time.
Overview of Classifying Work in z/OS WLM

Some details to follow …

zManager Workload

non-z/OS Virtual Server

non-z/OS Virtual Server

ARM

GPMP

zManager can shift resources between virtual servers within the hypervisor
If Processor Management enabled and CPU shared. Based on zManager’s knowledge of all workloads and performance goals.

z/OS Virtual Server

Workload Manager (WLM)

Classification

GPMP

ARM

Enabled on LPAR

Enabled Target Middleware

zManager will not shift resources within LPAR; that’s WLM’s role
But by associating that work with a WLM service class of your choice you can enable closer coordination of priorities between zManager and zWLM

Performance Monitoring and Reporting
Including z/OS virtual servers

Let’s turn our attention to the topic of propagating priority definitions (zManager service class) into z/OS and WLM so the management there may correlate with more closely with what’s going on in the rest of the zEnterprise.

It’s important at this point to remind you that WLM on z/OS is a special thing … it has control over allocating access to system resources within the OS. That is not true of zManager and other operating systems. All zManager can do is re-allocate resources between virtual servers. But it can not reach up into an AIX virtual server (for example) and shift resource from Process X to Process Y. The operating system doesn’t permit that, and zManager can only do what it can do within the limits of the operating systems above the hypervisor level.

So the key here is not having zManager tell PR/SM to allocate more CPU to the LPAR (it can’t do that). Rather, the key here is having information from zManager passed into WLM so that WLM services classes of comparable priority may be invoked. Does that guarantee precise priority management alignment? Not at all. But what it does do is provide you the administrator the ability to define WLM goals so they align more closely to what you have defined on zManager.

Think about it … if the zManager service class is Velocity=Fastest and Importance=Highest, wouldn’t it make sense to make sure any work in z/OS carry a comparable priority? Why burden the overall workload with a z/OS component slowed down with a WLM definition that grants the work something like discretionary? It makes no sense. Keep the priorities aligned for best overall alignment of workload priorities. And that’s done by having WLM service class goals that integrate with passed-in zManager service classes, and having the priority definitions of each properly aligned.
Associating zManager SC to WLM SC

Requires GPMP+ARM on both sides as well as EWLM subsystem classification rule to connect zManager service class name to WLM service class name.

If GPMP and ARM are enabled on the last hop before entry to z/OS, and DB2 for z/OS is ARM enabled with GPMP enabled for the LPAR … then the WLM EWLM subsystem classification rules can be applied to match the zManager service class name on the ARM token with a WLM service class.

The key is the ARM token coming in carrying the zManager service class name destined for a target on z/OS that is ARM enabled. Absent that, other WLM classification rules may apply, but not the EWLM classification rules.

Once inside the EWLM subsystem classification rules then the zManager service class name can be matched against a type of “ESC” to map the inbound request to a z/OS WLM service class name.

In the upcoming lab each team will define their zManager service class to a value of ZMGR##SC, where # is your team number. We have already prepared the EWLM subsystem definitions to include those values. We then map those values to z/OS WLM service class values of PPM# where # is your team number. You’ll see this mapping taking effect when we have you look at the SDSF Enclave information where you’ll see your team’s WLM service class in effect for the enclaves created by DB2 DDF.

The bottom part of this screen shows how to handle the case where the service class name coming in from zManager is longer than the traditional 8-character values found in z/OS. zManager allows names up to 64 characters mixed case. The best we can handle in z/OS WLM is 24 characters. To accommodate the longer values we can create tiered definitions as shown, with the longer name continuing on the next line as shown.
RMF Monitor III and SDSF Enclave Report

Some examples to show you that once classified the work shows up like any other classified work:

<table>
<thead>
<tr>
<th>Name</th>
<th>T</th>
<th>Goal Act</th>
<th>Exec Vel</th>
<th>Response Time</th>
<th>Perf</th>
<th>Ended</th>
<th>WAIT</th>
<th>EXECUT</th>
<th>ACTUAL</th>
<th>Indx</th>
<th>Rate</th>
<th>Time</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB_WKL</td>
<td>W</td>
<td>0.0</td>
<td>0.0</td>
<td>N/A</td>
<td>3.105</td>
<td>0.000</td>
<td>0.002</td>
<td>0.002</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DDFDEF</td>
<td>S</td>
<td>50</td>
<td>0.0</td>
<td>N/A</td>
<td>3.105</td>
<td>0.000</td>
<td>0.002</td>
<td>0.002</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PMHGHST</td>
<td>L</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.043</td>
<td>0.000</td>
<td>0.014</td>
<td>0.014</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYSID</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Showing service class PPMGHST activity based on propagation of zManager service class into zWLM

In the lab we'll drive you into the SDSF enclave display to see your service classes shown up in the display. That will be validation that your zManager service class is in fact getting into WLM and being classified to the service classes in WLM that we set up ahead of time.

The top part of this picture shows a hypothetical example where the RMF III Monitor can be used to see the activity in your service class. Earlier we showed an example service class of PPMGHST, and here we show that service class showing up in RMF Monitor III with activity. If you drove into RMF on the lab system you'd see something similar for your team's predefined service class of PPMGRT#, where # is your team number.
Final turn, heading for home … let’s now talk about zManager reports.
“Workloads” Under “Ensemble Management”

Provides a handy entry-point to the workload-related reports:

We’ll start by focusing on the entry point to reports related to workloads. This can be found on the HMC under “Ensemble Management.” That will display a list of all the workloads defined to the ensemble with some very sumary information supplied about each.

If you select the radio button next to any given workload there’ll appear a little twisty that gives you access to a context menu. Under “Monitor” will be another menu with four reports – Service Classes, Virtual Servers, Workload Resource Adjustments and Workloads. We’ll focus on these reports.
A Roadmap to the Location of Reports

The following is a reference showing the access path you would use to get at the reports we’ll show you:

- Service Class Resource Adjustments Report
- Hops Report
- Virtual Server Topology Report
- Performance Index (PI) Report for Service Class
- Hypervisor Report
- Adjustments Report
- CPU Utilization for Hypervisor
- Adjustments Report for selected workload
- Virtual Servers Report for selected workload
- Service Classes Report for selected workload
- CPU Utilization Distribution for workload

Some reports are accessible through multiple paths

With time and a little practice the location and content of these reports becomes second nature

This chart is meant as a reference for where various reports are located under the context menu. What makes this just a touch confusing is that some reports are available through multiple paths down the menu tree. But as with most things, time and practice begins to familiarize you with where things are located. You should soon become more comfortable with where certain reports and certain information is located.
Note about Granularity and Retention of Data
There are limitations to the granularity and retention of historical ...

- The 36 hour rolling window of retention

  - Most Recent 1 Hour
  - Previous Hours
  - Past
  - Future

zManager maintains 36 hours of recorded performance data

- The most recent 1 hour of data is maintained with 1 minute interval granularity

- The 35 hours prior to the most recent hour are maintained with 15 minute interval granularity

As we get into the reports and start showing data it’s important to understand the limits of the data that is retained by zManager. zManager maintains 36 hours of historical data. As that 36 hour window progresses forward, data on the backend is discarded. Further, the most recent one hour time period is maintained in one minute increments, but the back 35 hours are maintained in 15 minute increment detail.

By the way, how much of this 36 hour window you display in your reports is configurable. You’ll see that next.
Workloads Report

Provides a high-level view of all workloads and how they’re doing:

- All the workloads
- Selected Workload
- Showing the time interval for the information displayed. This is configurable
- Actions against selected workload. These take you to the workload-specific reports
- Shows the highest PI for any service class in the workload
- Most reports have a twisty here that expands to show other reports in the lower half of the page display for the workload selected

On that four-report sub-menu is an option called “Workloads” report. This is something different from the “Workloads” that’s under “Ensemble Management” on the main HMC panel. This “workload report” provides the same list of workloads but with a degree more information.

One change you see is that on this report each workload is displayed with the service class within the workload that is experiencing the highest PI at that time. A workload may contain multiple service classes, but this is only showing the one with the highest PI.

Note the little twisty at the bottom of the chart. This is a common tool used in all the reports to provide a place to display more detailed reports for the selected object in the upper portion of the screen. On this report the object would be a given workload. In other reports the select object might be a service class, or some other detail of a defined workload.

At the top of the report you see where you may configure the amount of data used for the generation of the report. In this example the past 15 minutes is being used. But the highlighted “Modify” link allows you to change this value. You may select up to the full 36 hours of retained information. Be aware, however, that depending on what’s been taking place on your system there may be no data for your workload or service class. zManager will alert you to this. For example, when you first create a workload and it’s only a few minutes old, zManager won’t allow you to select 36 hours of history because that history simply doesn’t exist for your workload.

Finally, we draw your attention to the “Select Action” pulldown menu at the top. This processes an action against the selected object on the panel. What you’ll find in this pulldown changes depending on the report and object selected. On this “Workloads” report the “Select Action” pulldown gives you access to three other reports, which we’ll explore in the upcoming charts.
One of the reports found under the twisty at the bottom of the “Workloads” report is the CPU utilization seen virtual servers in the selected workload. (The other report is the Performance Index graph, which we’ll show you next.)

The CPU utilization chart may appear a little odd at first, but upon reflection it’s providing some interesting information. Recall that a workload is comprised of one to many virtual servers. Each virtual server at any given moment is experiencing a certain percentage of CPU utilization as seen by the hypervisor. This chart is showing you how many virtual servers fall into ranges of CPU utilization … 0 to 10%, 10 to 20%, etc. In this example three virtual servers make up the workload. One falls into the 0 to 10% utilization range; one in the 10 to 20% utilization range; and the final virtual server falls into the 30 to 40% utilization range.

If we had more virtual servers in our example workload and the number of servers falling into the various ranges differed, this chart would look more like the bar graph histogram we might expect. The picture looks a little funny simply because we have one serve falling into different ranges.
Workloads Report – Performance Index

Here’s the Performance Index report for the selected workload:

Earlier we spoke of the “Performance Index” (PI) and what it means … PI>1 means the service class’s goals are not being met; PI=1 means the goals are being met; and PI<1 means the goals are being exceeded.

This chart is showing the PI value across time for the service classes in the workload for which there is data. In this example only one service class had work classified to it, so only on PI line is displayed. This workload had three service classes defined to it, but again … only one had work being assigned. The assignment of work to a service class is based on the classification rules that exist within the service class.

In the upcoming lab the objective is to observe this graph for your workload showing a downward slope towards a PI=1 as zManager adjusts resources to meet your goals. This chart shows a service class that is not meeting its goals except for a short period of time.
Virtual Servers Report

This provides a snapshot of the virtual servers that are associated with a workload.

The Virtual Servers report provides a snapshot of the state for the virtual servers that are present in the selected workload. There are two ways to access this report – one is from the pulldown menu and the other is from the sub-menu under “Monitor.” In either case the information that is shown is the same … the virtual servers in the workload and some performance information about each.

The columns that display on this report are configurable, as they are on other reports as well. The little icon highlighted is what you may use to configure which columns display and which are hidden.

To the right notice that this report is displaying information about the CPU utilization up in the operating system of the virtual server. If this information is present it means GPMP is enabled for the virtual server. That’s how zManager would get information about the OS.

Once again, notice the twisty down at the bottom left. This will open up the lower portion of the report and show more detailed information for the virtual server that’s selected. Under the virtual servers report the charts found under this twisty show a line graph of the CPU utilization for the selected virtual server over the time interval you have configured. (We don’t show that graph in this presentation … it’s a standard line graph with the X-axis representing time and the Y-axis representing CPU utilization.)
Workload Resource Adjustments Report

Shows successful and failed attempts to adjust CPU resources between virtual servers under the control of a hypervisor:

For this to work, “Processor Management” must be enabled for the hypervisor and the virtual servers, and virtual servers must be defined as “Shared” CPU (not dedicated).

Donor to Receiver CPU transfer by zManager

You can see the incremental transfer of CPU from donor to receiver

This is how zManager attempts to manage the PI to the goal

The “Workload Adjustments Report” tells the very interesting story of what adjustments zManager has made to virtual server CPU allocations in an attempt to meet the goals of the workload.

Note: zManager will only make adjustments when (a) “Processor Management” is enabled for both the hypervisor and the virtual server, (b) the CPU for a virtual server is “shared” rather than dedicated, and (c) there’s a need to make adjustments. You may find this report without any information … and the most likely cause for that will be that no adjustments have been made in the time interval configured for the report.

The report has two halves – the top half is “successful” adjustments and the bottom half is for “failed” adjustments. This example is showing a list of successful adjustments with no failed.

This notion of adjusting CPU is based on the idea that you have a “donor” virtual server that can give up CPU, and a “receiver” virtual server that needs it. Earlier we spoke of zManager keeping all the workloads and service classes in its head as it attempted to balance everything to the defined goals … well, here’s where the balancing takes place. In the chart above you can see that zManager is moving small increments of CPU from one virtual server to several others. In the upcoming lab the “donor” will be the server ending in the letter “c” … and the receiving virtual servers will be the ones doing all the work … the HTTP and the WAS servers. The objective in the upcoming lab is to see these adjustments being made so that the PI line for your workload shows a steady movement down to the PI=1 goal.
Virtual Server Topology and Hops Reports

Relies on ARM+GPMP to understand and portray the topology of the flow within the workload across the associated virtual servers:

We’ll show you two more reports then let you get to lab. If you drill down to the Service Classes report you’ll get a list of the service classes that make up the workload. From that list you can select a service class and then from the pulldown menu you’ll see the “Virtual Server Topology” report and the “Hops” report. Both are shown on this chart.

The Topology report is a graphical representation of the virtual servers that make up the work as reported by GPMP and ARM. The ARM token reports the hop number as it flows through, and zManager can use that to understand the sequential relationship of virtual servers in the work flowing through. If you don’t have ARM and GPMP enabled, this information won’t be present.

In the picture above the first server is the HTTP Server with the WAS plugin, the middle server is the WAS server itself and the final server is the z/OS LPAR running DB2 for z/OS. The number of requests seen for each hop. In the picture above the number of requests from the HTTP Server to WAS is relatively low (39882) while the number from WAS to DB2 is very high (4463634). Why the difference? The “DayTrader” application we’re using as a sample makes a lot of database calls for each request from the user. So 39,882 user requests turns into nearly 4.5 million calls up to DB2.

The Hops report is showing the same kind of information but in a tabular format. The response times seen for each hop are shown as well. This information, as well as information in other reports, can be exported in CSV format for further analysis.
Let's summarize and then take a look at the lab you'll do next.
Summary

**GPMP and ARM**
- GPMP is a lightweight performance agent that runs in the OS
- ARM is a means of tracking end-to-end response time
- Neither is strictly required though certain things are lost if not there

**zManager Workloads**
- A means of logically associating virtual servers into a work group
- Service classes define the relative priority
- Classification rules assign work to the service class
- Performance policies contain the service classes
- Workloads contain the performance policies
- zManager then reports performance and manages to goals
- Propagation of zManager service class into z/OS WLM possible

**Reports**
- Accessible through HMC
- Reports we explored were related to the workload
- Information about performance attainment, virtual server CPU, etc.

A summary of the material presented in this unit.
Hands-on Lab Review

The upcoming lab will have you build on the virtual server creation work you did in the first lab. First you'll enable GPMP on the HTTP virtual server and the WAS virtual server. Then we'll have you ARM-enable WAS and the HTTP Server plugin. We'll have you install the DayTrader application into WAS.

Then you'll define the zManager workload and set the performance goals for the workload. Then you'll fire up the driver software and watch as zManager works to manage the virtual servers to the goals you've defined.

End of Unit