z/OS® Workload Manager

How It Works
And How To Use It

INSIDE WLM **Issue No. 1**

Robert Vaupel
Senior Technical Staff Member
IBM Development Laboratory
Böblingen, Germany
Tel.: +49(0)-7031-16-4239
Email: vaupel@de.ibm.com

April 25, 2014

Contents

List of Figures

List of Tables

1. Introduction

The IBM System z® server platform is generally regarded by industry analysts and leading companies as the defining standard in enterprise computing — especially in the age of e-business. The platform's scalability, availability, security, interoperability, ability to manage mixed workloads, and its low total cost of ownership are the pillars on which enterprise-wide solutions are built. The qualities of service of System zare generally accepted as best-of-breed. Although the qualities of service of System zare generally accepted as best-of-breed, the underlying technologies that deliver those qualities are not as widely understood.

One of the strength of the System zplatform and the z/OS operating system are the ability to run multiple workloads at the same time within one z/OS operating system image or across multiple images which share the same physical processor. The function which makes this possible is dynamic workload management which is implemented in the Workload Manager component of the z/OSoperating system.

The z/OSWorkload Manager (WLM) component introduces the capability of dynamically allocating or re-distributing server resources, such as CPU, I/O, and memory across a set of workloads based on user defined goals and their resource demand within a z/OSimage. Looking over the fence of the z/OSimage the Workload Manager is able to perform this function also across multiple images of z/OS, Linux or VM operating systems sharing the System zprocessor. Another function of WLM is to assist routing components and products to dynamically direct work requests associated with a multi-system workload to run on a z/OSimage within a Parallel Sysplex that has sufficient server resources to meet customer defined goals.

This article tries to introduce the basis concepts of WLM and gives hints and tips for setting up a service definition.

2. Basic Concepts

The idea of z/OS Workload Manager is to make a contract between the installation (end user) and the operating system. The installation classifies the work running on the z/OS operating system in distinct service classes and defines goals for them which express the expectation how the work should perform. WLM uses these goal definitions to manage the work across all systems of a parallel sysplex environment. So far this technique seems very simple and in deed the goal definitions have many similarities to definitions of a Service Level Agreement which is agreed between the installation and its end users. Nevertheless it is not the same and in order to understand how the system is managed, how the system behaves and what can be used for goals we will start to look at an analogy which allows us to explain most of its concepts.

2.1. A simple Analogy

In order to understand how WLM works we will take a look at an example of daily life, we will look at the traffic in a city. Usually when we drive with a car between two locations in a city the time we need is determined by the constant speed we are allowed to go, the amount of traffic on the streets and the traffic regulations at crossings and intersects. Based on this we can already identify the time as the basic measure when going from a start to a destination point by car in the city. We can also easily understand that the amount of traffic is the determining factor of how fast we can go between these two points. While the driving speed is usually regulated and very much constant the amount of cars on the street determines how long we have to wait at intersects and crossings before we can pass them. We can immediately see that going from A to B on a Sunday morning will most likely cost much less time than during the rush hours. As a result of this we can identify the crossings, traffic lights and intersects as the points where we encounter delays on our way through the city.

A traffic management system can now use these points to manage the running traffic. We see examples of it in many of our cities. For example bus and taxi lanes allow buses and taxi to pass waiting traffic during rush hours and therefore allow them to travel faster through the city than regular passenger cars. This is a common model of prioritizing a certain type of vehicles against others in the traffic systems of a city. It allows the bus companies to meet a certain time schedule even during rush hours.

We see that it is possible to measure time needed to travel between two points in a city. We identify the points of interest where contention may occur and we found a simple but efficient method of prioritizing and managing the traffic in a city (see figure 2.1).

2.2. Managing Work in a Computer System

These thoughts can be easily ported to a computer system. First we need a technique to identify the work running in a computer system, then to measure the time it runs and to identify the contention points. The identification of work requests is something which is supported by the applications and the operating system. They tell Workload Manager when a new unit of work enters the system and when it leaves the system. Workload Manager provides constructs to separate the work into distinct classes. This is named work classification and allows an installation to deal with different types of work running on the computer system. The contention points are given when the work wants to use the resources of the computer. These are the CPUs, the I/O devices, storage but also software constructs like processes or address spaces which provide the capability to execute programs and serialization points which allow

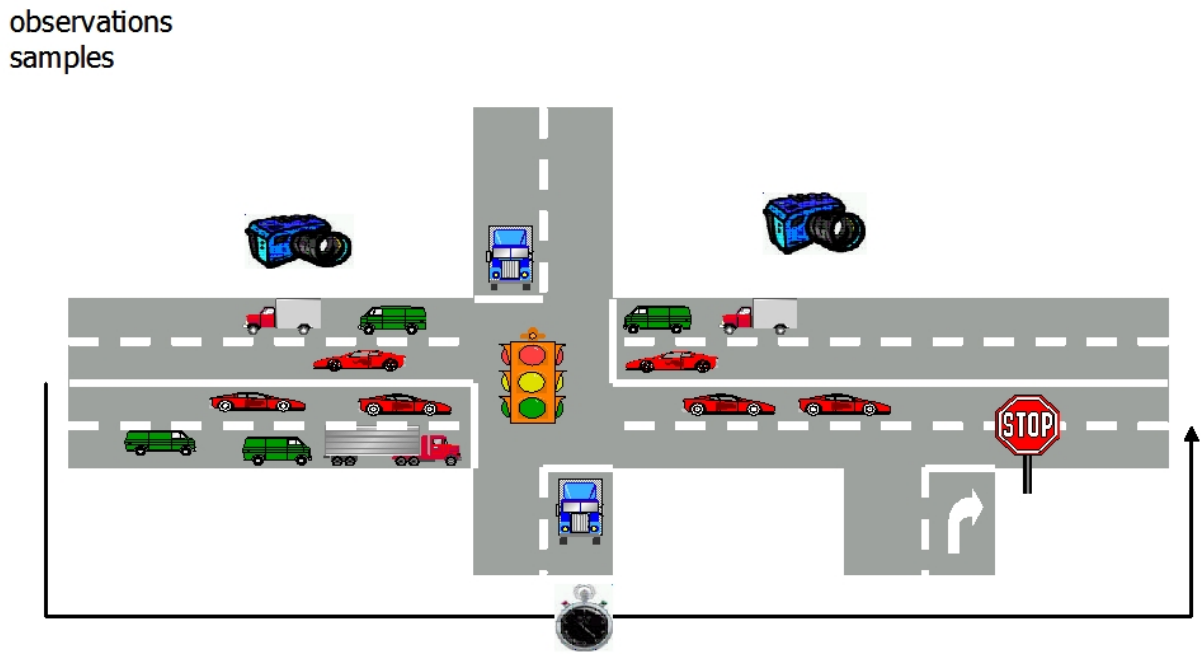

Figure 2.1.: How traffic can be managed

the programs to access resources or serialized control areas. Workload Manager observes these resources in order to identify how much work wants to use it and how much work has to wait for it.

The work classification and observing how the the work uses the resources provide the base to manage it in the computer system. The management is done based on the goals the installation defines for the work. After classifying the work into distinct classes the installation associates a goal with each class. The goal determines how much service the work in the class is able to receive. These classes of work are named service classes. Besides the goal it is also necessary to define which work is more important than other work. In case to much work wants to use the same resource of an operating system, WLM must know which one it has to give preference. This is done by defining an importance for each service class. The importance is a number from 1 to 5 with 1 being the most important work in the system and 5 the work which can suffer first when resources get constraint. There might be the desire to have work running in the system for which no concrete goal must be achieved. This work should only run when plenty of resources are available. This work is classified to service classes with a discretionary or no goal and thus tells the system that it should only run when service classes with an importance and goal don't need the resources to achieve their goals.

2.3. Service Class Goals

Now we have all basic constructs to define a policy but we have to clarify what the goals of our service classes can be. From the traffic example we see that the most natural goal is the time which is required for a car to go between two points in a city. The same is true for work running in a system. The time which the work needs between the points when it enters and leaves the system can be used as a base to manage it. Because a computer system deals with many work requests running in parallel the time of an individual request becomes less important. We are more interested in the average time the work needs to fulfill its task on the computing system. Because this is also the time the end user can observe when it waits on a response from the computing system, this time is named the response time.

We will later see that the average response time can have some problems and we can already define

a simple modification to it. Instead of using the time across all work requests of a service classes we might only be interested in a certain amount of work requests ending within a predefined time. Other work requests may run much longer but we will consider the goal as achieved when the defined amount of requests ends in the expected time period. By going back to the traffic example it means that we are satisfied when a certain percentage of the buses meet their schedules. This goal is named a percentile response time goal and defines the percentage of all work requests which must end within a predefined time limit.

Using the time as a measure to manage the work on a computing system requires that the computing system has the ability to capture it. Usually an operating system provides processes or address spaces to execute the application programs. These constructs are very often started once and live for a very long time. That means Workload Manager needs assistance from the applications to capture the response time. In deed Workload Manager provides interfaces which allow them to capture the time the work needs when it runs on z/OS. Even more WLM also supports an infrastructure which allows the applications to identify individual work requests, capture their response times and thus allow WLM to manage them in their own service classes. With respect to our traffic system it would mean that each passenger of a bus could be identified and that the buses could be directed in a way to help certain groups of passengers to meet time targets when they move through the city. While this is not possible today in the traffic example it means that on z/OS WLM is able to manage service classes of transactions which belong to the same application and which are executed by a set of server address spaces.

Finally the question arises what the system can do when it is not possible to capture the response times of the transactions. There are many reasons why this is necessary, an application might not be able to tell WLM the response times of their transactions, or there are too few and too sporadic transactions so that it doesn't make sense to manage them towards a response time goal. Also many address spaces provide services in the system which can't be easily attributed to a certain type of transactions. For those address spaces it is also necessary to understand how fast they progress in the system and it is also necessary to define a goal for them. Because the system is not able to capture the response time of this type of work it must use the information it is able to collect. This is the information about the delay and using values it observes when the work tries to use resources in the system. In the traffic example this information can be obtained by establishing cameras at all points where such delay situations may occur and constantly taking snapshots. After a certain time period these snapshots are summarized and the result helps to identify how often individual work and the service classes were delayed for a resource and how often they were able to use it. The speed in the system can now be expressed by the acceptable amount of delay for the work when it executes. The speed is named execution velocity and defined as

$$Execution\ Velocity = \frac{All\ Using\ Samples}{All\ Using\ +\ AllDelay\ Samples} \times 100$$

The result is a number between 0 and 100, 100 meaning that all measured samples are using samples and that the work didn't encounter any delays and 0 meaning that the work was completely delayed over the measurement interval. We will see that the truth always lies in the middle.

We will end this introduction with a comparison of the potential goals which can be defined for work in the system and we want to point to a difficulty when we try to compare those goals with each other. On the one side we can have a response time goal. Independently whether we use an average or percentile response time goal we can always immediately understand what this means with respect to how fast the work progresses in the system. On the other hand we have an execution velocity goal. Clearly we see that an execution velocity of 100 means no delay and therefore it means maximal speed. But what does an execution velocity goal of 30 mean? Wit respect to the traffic example we know that it is always possible to go very fast by car in a city on a Sunday morning but not during rush hours. During rush hours a fast speed also always means that the car has to wait at intersects and traffic lights. The same is true for buses and taxi. We understand that it might not be possible to cover a distance of 5 kilometers at that time in less than 30 minutes. That also means that the best we can do during rush hours implies that we have to wait very often and that a speed expressed as an execution velocity will be very low. If we want to define a reliable goal which can be achieved under all circumstances we have to understand

how the system behaves during "rush hour". Such goals may be easily achieved during other times but because there is no contention and everybody and everything can progress very fast we are not really interested in those other times. When we have two service classes of work, one for which we can measure the response time and for the other where we can't measure the response time, we need a way to compare the potential goal settings against each other. Because it is very difficult to understand how a certain execution velocity translates into a certain response time we have to capture the execution velocity of the work being managed towards response time goals simply because we need to understand what this means as an execution velocity.

2.4. The Service Definition

The result from the previous discussion and example are:

- Work needs to be identified and grouped into classes in order to manage it.
  - The identification is supported by the operating system and the applications.
  - Furthermore WLM provides interfaces which allow the application infrastructure (the middleware or subsystems) to identify work being executed so that WLM can manage it in distinct groups.
  - The end user classifies the work into distinct service classes by subsystem type. A subsystem type encompasses similar work and is usually related to an application or middleware.
  - While service classes define the runtime requirements for work, a higher grouping and differentiation construct the workload combines multiple service classes together.
- In order to manage work, the service classes need to be associated with an importance and a goal.
  - The importance tells WLM which service classes are preferred against others when the goals can't be achieved.
  - The goals tell WLM what must be achieved in order to meet end user requirements.
  - The following goals are possible for service classes
    - An average response time
    - A percentile response time, meaning a certain percentage of work requests which have to end within a certain time limit
    - An execution velocity, that is the acceptable amount of delay work should encounter when it runs in the system
    - Discretionary if no specific goal definition and no specific importance for the work in the service class exist.
  - The goals must be oriented towards end user expectations and towards what is possible in a system. WLM can't exceed the physical limitations of the system.

The end user defines the service classes and goals in a service definition. A service definition for z/OS contains one set of classification rules which separates the incoming work into distinct service classes and multiple service policies. A service policy is a named set of goals of the service classes. Each service class is associated with a base goal and by using different policies this goal can be modified. Besides these constructs a service definitions consists of:

- Report Classes which allow subdividing the service classes into entities for reporting purposes.
- Resource Groups to define limits for the minimum and maximum CPU service of work in a sysplex environment.
- Application Environments which give WLM the ability to manage server address spaces as resource for middleware applications.

- Scheduling Environments to allow applications and installations to define scheduling and affinity requirements.

All these constructs are described in detail in [11] and [12].

3. A Look Inside WLM

In the following chapter we will discuss how WLM works. Workload Manager provides the capability to give work access to resources based on demand and on their goal achievement. The underlying idea is to share the resources between the workloads so that everybody can optimally make progress in the system. The base for all decisions are the goals therefore it is important to understand how to set meaningful goals in the system.

3.1. How WLM Works

WLM constantly measures the systems, at certain intervals it combines the measurement results to a complete picture, calculates how work progresses and compares the actual achieved results with the goal definitions to find out whether adjustments are necessary. The measurement of the system state takes places every 250 milliseconds. At these points WLM captures the delay and using states of all work units in the system and resources it manages. WLM basically manages CPU, DASD I/O devices, the system storage, server address spaces and to some extent resource serialization like GRS enqueues. The states are manifold. While for CPU there is currently one using and one delay state it is possible to observe multiple possibilities why work can't use the storage, for example paging delays, shortages of the page data sets, and much more. Every 10 seconds WLM summarizes the system state information and combines it with measured data like CPU service or transaction end times. At this point WLM learns the behavior of the workloads and the system. It keeps a certain amount of historical data in storage and maintains graphs learning the relationship between dependent functions in the system.

The following describes an example for such a graph. It is possible to learn from the time work requests were delayed because they were swapped out how many address spaces should be ideally in storage. The delay per completed request while the work unit was not in storage can be measured. This is a function of the fraction of ready users which are in storage, usually named the Multi Programming Level (MPL). By plotting the MPL delay per completion over the number of ready users with MPL slots it is possible to learn at which point to few MPL slots will cause a drastic increase in MPL delay. In addition it can be learned at what point adding more MPL slots do not have a positive effect anymore on achieving the service class goals. Such a plot can be used in a decision algorithm to determine the effect of increasing or decreasing the number of ready users in storage.

There are many more examples. At this level it should only be noticed that everything WLM does is based on measured and sampled data which is combined to learn the relationship between workloads and resources in the system. After this step the achieved response times and achieved execution velocities are compared with the goal definitions. The result is the so called performance index (PI) which is the base for all further decisions in the system. The PI is defined as followed:

$$Execution\ Velocity\ Goal:\ PI\ = \frac{Execution\ Velocity\ Goal}{Achieved\ Execution\ Velocity}$$

$$Response\ Time\ Goal:\ PI\ = \frac{Achieved\ Response\ Time}{Response\ Time\ Goal}$$

The PI can be easily interpreted. If it is below 1 the goal is overachieved, if it is equal to 1 the goal is exactly met and if it exceeds 1 the goal is missed.

The next step is to visit all goal achievements and to identify service classes which do not meet their

goals as potential receivers for goal adjustments. Figure 3.1 shows the basic decision flow. All service classes are logically sorted by their importance and their goal achievement. The first receiver is the service class with the highest importance which does meet its goals and which has the worst PI. After determining the receiver WLM examines its bottlenecks. The biggest bottleneck is those for which the highest number of delays has been found. Then WLM searches one or multiple donors which also use the resource. While receivers are searched from highest to lowest importance, donors are searched by examining discretionary work first and then from lowest to highest importance.

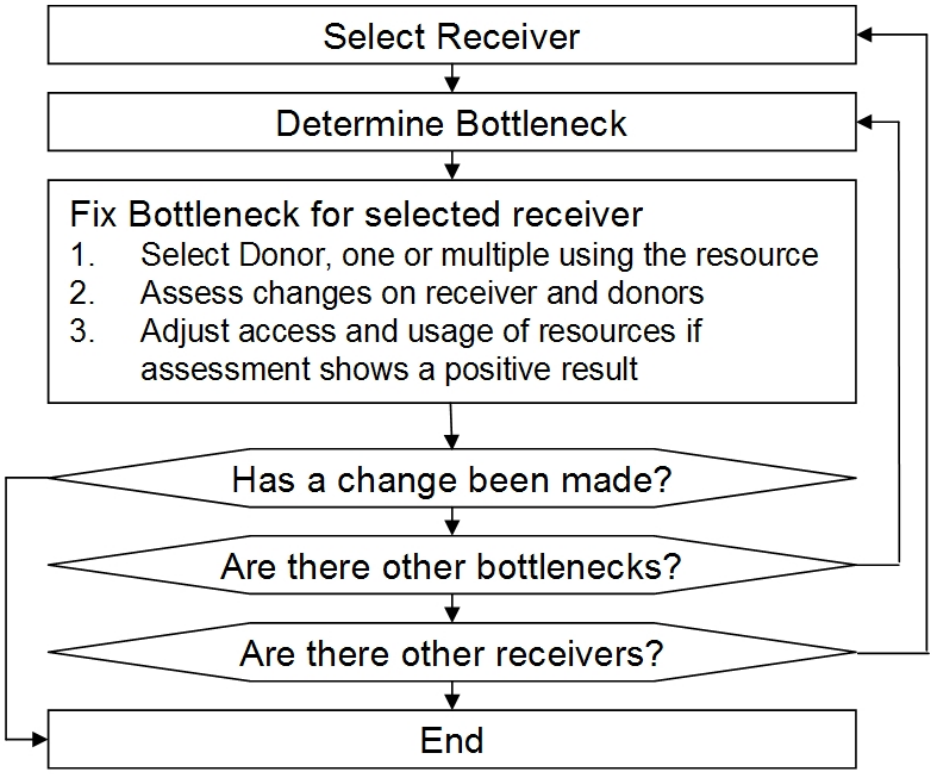

Figure 3.1.: Goal Achievement Logic

If donors are found WLM calculates and assesses the changes for the receivers and donors. At this point it uses the internally captured and learned data, for example the plots as previously described. The assessment logic calculates the improvement for the receiver and the expected degradation for the donors. If the logic concludes that the change is beneficial it executes it and the adjustment is completed for this interval.

There are many possibilities why a change is beneficial and can be made. The easiest possibility is that the receiver is an important service class and the donors have lower importance. In that case it can be safely assumed that the change is meaningful. But is also possible that a more important service class can become a donor for a lower important service class, for example if the higher important service class overachieves its goals and the projections show that an adjustment will improve the lower important work and still allows the higher important work to easily meet its goals. Very interesting are also cases if the receiver and the donor have the same importance and both do not meet their goals. In such cases WLM will try to bring the goal achievement of both closer together thus improving the overall throughput of the system.

When a change has been made, the logic completes, new data is collected and the evaluations and assessment runs after 10 seconds again. If no change was possible WLM first tries to examine other potential bottlenecks of the receiver and then continues with lower important work until one or no

change could be made.

3.2. CPU Management Example

We will now take a look at an example to demonstrate how WLM works and how WLM adjusts the access to resources based on goal settings and goal achievement. One of the most interesting algorithms is for CPU management.

Access to the CPU is controlled through Dispatch Priorities. The dispatcher maintains a queue of ready work units which is sorted in priority order. Dispatch priorities can have a value up to 255, identifying the most important work in the system. Practically only a subset of all possible values is used. The dispatch priority is assigned by Workload Manager on a per service class basis. Table 3.1 shows the range of dispatch priorities and how WLM assigns them to the work in the system.

DP Range	Description
255	SYSTEM
254	SYSSTC
253 to 249	Not used
248	Small consumer
247 to 204	Dynamically Managed Dispatch Priorities
203 to 202	Not used
201 to 192	Discretionary (Mean Time to Wait Algorithm)
191	Quiesce

Table 3.1.: CPU Dispatch Priorities

The dispatch priority of 255 is reserved to keep the system running and to provide system work preferential access to the CPU. All system work should run in the pre-defined service class SYSTEM which always gets a dispatch priority of 255. Support work and system related work can be associated with another pre-defined service class SYSSTC. Work in SYSSTC gets dispatch priority 254. The ranges from 249 to 253, and 202 to 203 are not used. The range from 204 to 247 is dynamically assigned to user defined service classes. These are all service classes with an importance of 1 to 5. One special dispatch priority is used for service classes which consume very little CPU service. Because it is difficult for WLM and not beneficial for the whole system to deal with work which only requires very little CPU, WLM gives this work a dispatch priority just above the range of dynamically managed dispatch priorities.

The range from 192 to 201 is used for discretionary work. All discretionary work is managed by a mean time to wait algorithm which simplified gives a higher dispatch priority for work using I/O and a lower dispatch priority to work demanding a lot of CPU. The last used dispatch priority is 191. It is assigned to quiesce work. That does also imply that quiesced work can run if the CPU is not busy, it also means that all other work can access the CPU before it.

The interesting range is from 204 to 247. WLM assigns these dispatch priorities to the user defined service classes which have a goal. The assignment is based on the CPU consumption, CPU demand and goal achievement of the work. In the previous chapter we saw that WLM determines through the performance index which service classes meet and which miss their goals. In the next step WLM start to look for a receiver by starting with the most important service class with the worst PI. Then it determines the resource for which an adjustment is required. For CPU that means that the service class must show a high number of CPU delays. Now WLM looks for donors. Based on the captured data WLM knows how much CPU is consumed by each service class and how much CPU is consumed by all work at the same dispatch priority. From the CPU delays and average time the work used the CPU it is possible to calculate the demand the work has for the CPU. In order to assign a new dispatch priority to a service

class, WLM logically moves the service class to higher dispatch priority and then projects the change of CPU consumption, demand and delays for all service classes at a lower and the same new dispatch priority of the work being moved up. If that's not sufficient WLM looks in the next step whether some service classes can run with a lower dispatch priority and does the same calculations. The algorithm continues until a positive result can be projected, meaning that the performance of the receiver gets better and no service classes with a higher importance is badly affected, or until WLM concludes that no change is possible.

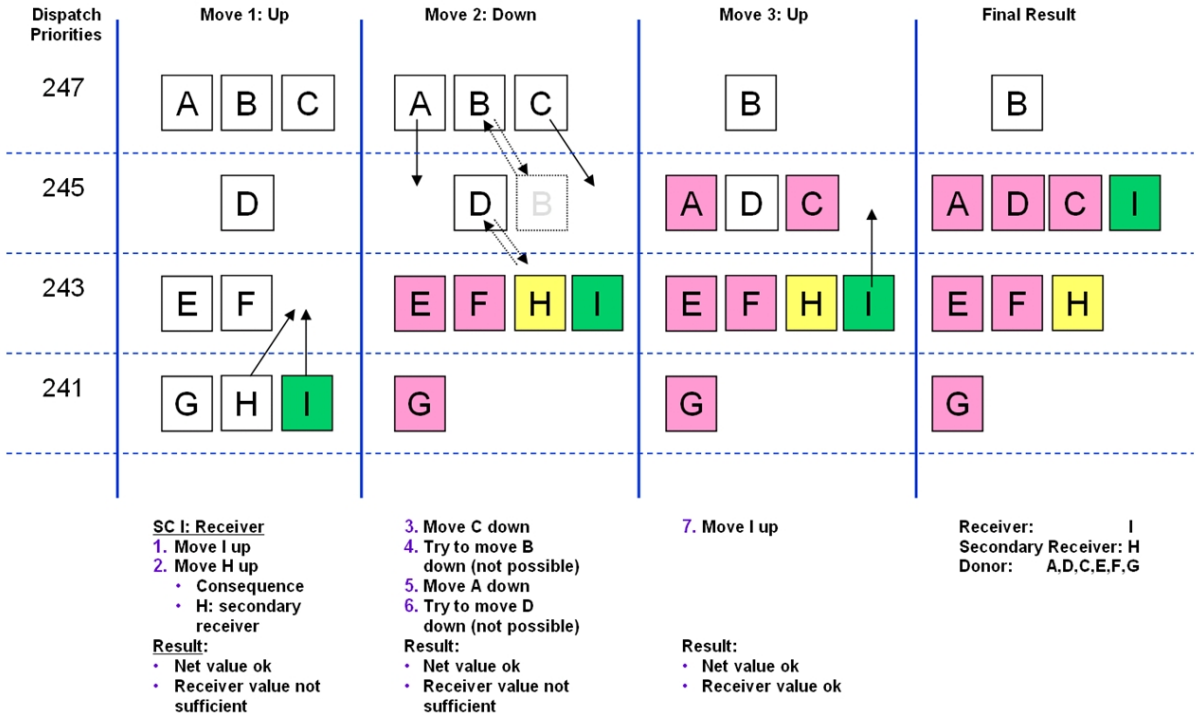

Figure 3.2.: CPU Management: Moving Service Classes

Figure 3.2 depicts this algorithm. The example assumes 9 service classes and the policy adjustment algorithm selected service class I as a potential receiver of CPU service. In the first step it moves I to higher dispatch priority. Together with I there are two other service classes G and H at the same dispatch priority. After the first move up for I, the calculations show that H also has to move up because otherwise no overall value of the move for service class I can be projected. This projection is named net value, meaning that the overall change does not harm higher important work and is still beneficial for the receiver. But this move does not show enough receiver value, meaning that the change does not help enough to improve its performance. In the next step WLM tries to move work with higher dispatch priorities to lower dispatch priorities. While the move down for C is ok, the move down for B violates the net value test. Finally, moving A down passes the net value test, but again we do not have sufficient receiver value for I. In the third step WLM again attempts to move the receiver up. In this case the move for service class I to dispatch priority 249 gives enough receiver value for I and all other classes pass the net value tests. Now WLM concludes that the change is meaningful and adjusts the dispatch priorities of the work. In case no valuable change could be projected all intermediate calculations would have been set back and no change would have occurred.

3.2.1. A Life Scenario

To demonstrate how this works on a real system we will take a look at a scenario where multiple different workloads and service classes compete for CPU resources. Table **??** shows the service classes which are used in the scenario.

Service Class	Period	Goal	Value	Imp
CICS	1	Avg. RT	0.1 sec	2
IMS	1	Avg. RT	10 sec	3
TSO	1	Avg. RT	0.1 sec	3
TSO	2	Avg. RT	1.0 sec	3
TSO	3	Avg. RT	3.0 sec	4
Batch	1	ExVel	10	5

Table 3.2.: Service Class Definitions

The most important work is CICS. This is ensured by defining the highest importance for CICS, in this case an importance of 2. Below CICS we have TSO and IMS. The first periods of TSO have the same importance as IMS but the time transactions are allowed to stay in these periods is limited so that long running TSO transactions will go to the third period which has a lower importance. The duration is defined in service units and not shown in the diagram. At the bottom of the service classes we find batch work with the lowest importance. Figure 3.3 shows the CPU service consumption of the service classes, please notice that the periods for TSO are combined for simplification purposes.

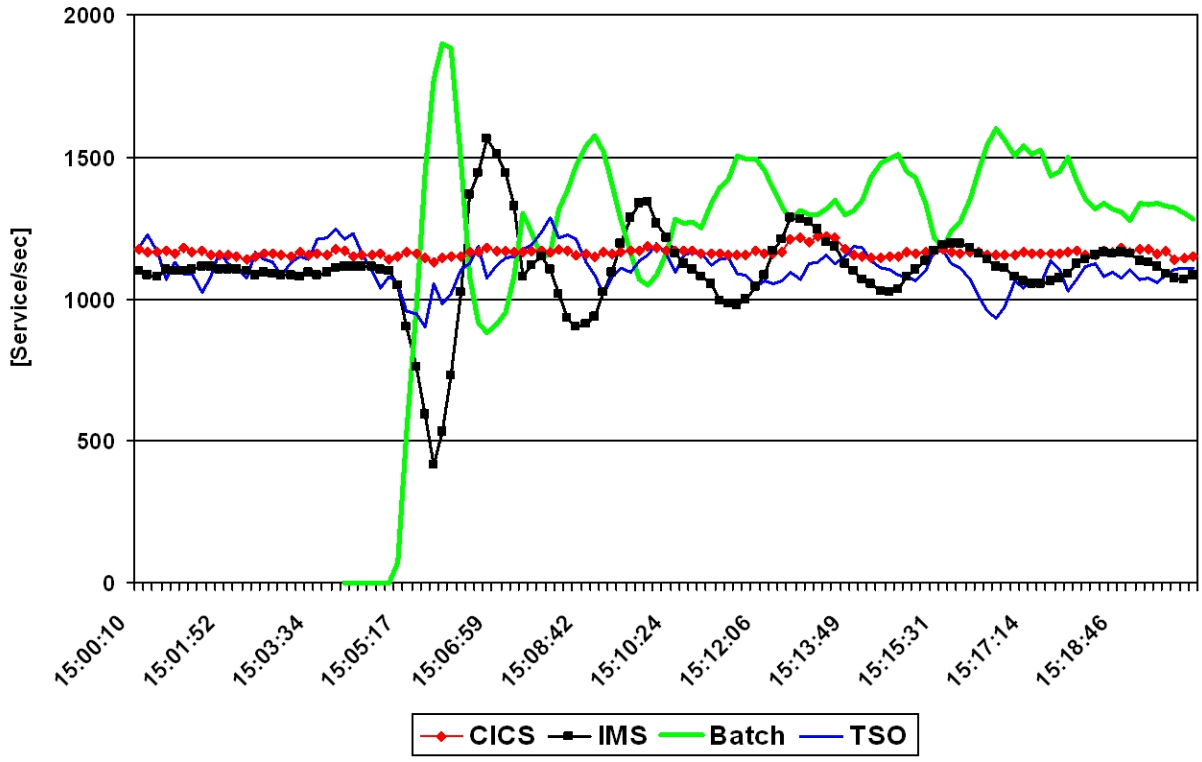

Figure 3.3.: CPU Service Consumption Example

When we examine the chart we can see that until 15:05 CPU service is only consumed by CICS, IMS and TSO. At this time, the system still has some additional capacity and the service consumption for the three service classes is very constant. Around 15:05 a large amount of batch jobs is submitted, now demanding a lot of CPU. We can observe that the high CPU demand and consumption of the batch work shows some impact on the TSO and IMS service classes while CICS doesn't seem to be affected at all. In order to better understand this behavior we will take a look at figure 3.4.

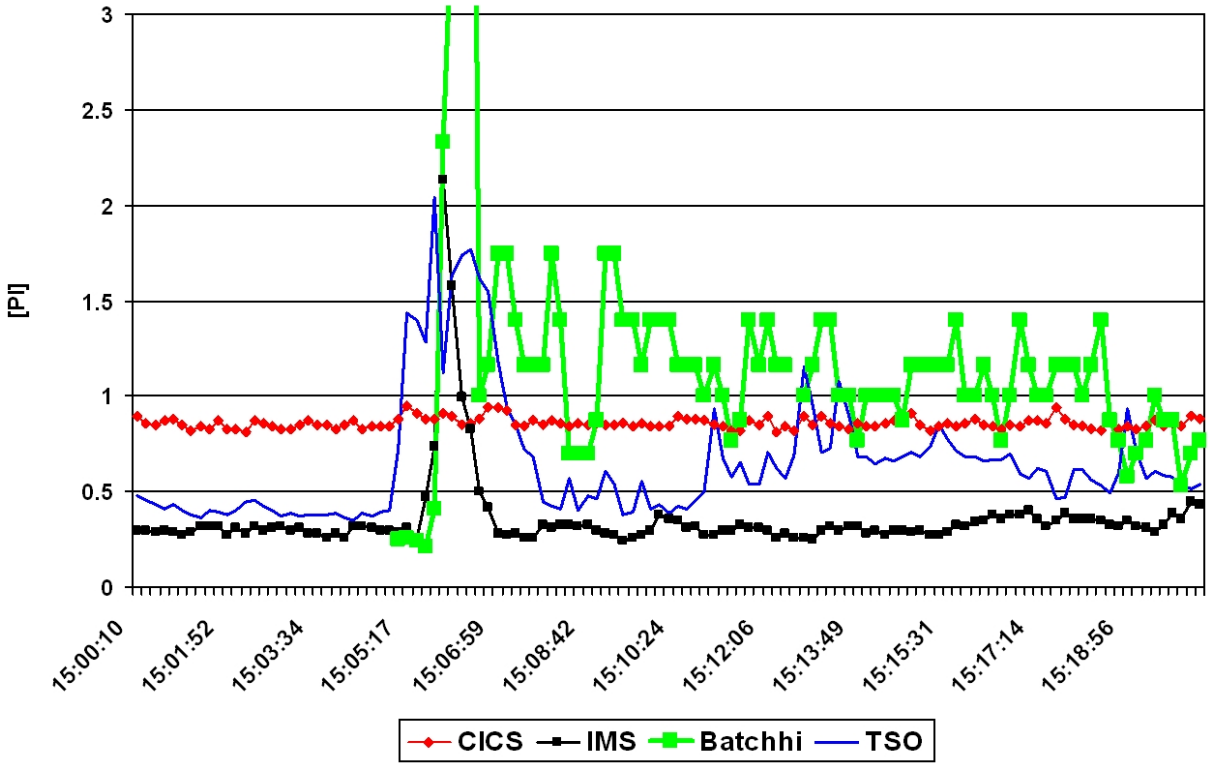

Figure 3.4.: Performance Index Example

It shows the performance index of the service classes. For the time before 15:05 we see that CICS has a performance index of around 0.9. The TSO and IMS service classes show a PI of less than 0.5. That means that CICS has a goal definition which is at the point where the system is just able to satisfy it. The goal-definitions of TSO and IMS are much more relaxed and they over achieve them easily. After the batch work has been submitted we can observe that the PIs of TSO and IMS go above 1. Obviously was the batch work able to get a higher dispatch priority at the time when the jobs were submitted. We also see that CICS is not affected; the PI stays very stable at 0.9. By looking to the right we see that the PI for IMS comes down very fast and a little bit later also the PI of the TSO service class . Later we observe that the service consumption for IMS and TSO is not as stable as it was before batch was submitted, but the performance index stays in general below 1. For batch the demand is much higher as the system capacity. So the PI stays above 1.

Let us understand what we observed in this scenario:

- We see that CICS is not affected by the batch work and that it is also not affected when the CPU demand of all work is higher than the system capacity:
  - The simple reason for this is that the goal setting was so tight and that the importance of the work was so high, that CICS always had the highest dispatch priority of the work of interest.
  - The performance index of 0.9 indicates that the capacity and speed of the system is just

sufficient to meet the goals of CICS and WLM gives CICS the highest dispatch priority to ensure that it always has the best access to the CPU.

- For IMS and TSO this is different:
  - The goal definitions are not very tight and the work can easily over achieve its goal definitions.
  - So it is possible for IMS and TSO to become a potential donor for lower important work. That was the case in the time before 15:05 where Batch only requested very little or no CPU.
- Batch work shortly impacts IMS and TSO but WLM reacts very fast and adjusts the dispatch priorities so that both workloads achieve their goals again. The whole adjustment period lasts a little bit more than 1 minute. Given that not all service classes are depicted that means that WLM needed about 6 adjustment cycles to correct the situation which developed at 15:05.

As a result we see that WLM reacts fast to unpredictable changes of the overall system conditions. The goal definitions and importance of the work define how WLM treats the work and how WLM protects it. The high important work with a tight goal definition was never affected while a low or relaxed definition allows lower important work to get sometimes better access to system resources. We now understand that the goal definition and importance of the service classes are important and that a low goal definition makes it eligible to become a donor. Nevertheless WLM still ensures that also low goals are achieved. A tight definition protects work. What we haven't seen in this scenario is the impact of the number of service classes in the system. We only listed a handful of service classes but in deed there were some more. All together there were around 10 service classes active on the system. From the previous discussion and the time which was required by WLM to correct the situation we can conclude that the number of service classes has some effect on how fast WLM can react on changes in the system. The same is true for the goal settings. While a too low goal setting may cause that a service class is preferred as a donor a too high goal definition can be counter productive as well. In that case the PI will always signal that the goals are not met and WLM will always examine the service class. This is not really beneficial for the service class as well as for the other work in the system and may also impact the time it takes for WLM to react.

3.3. Other Management Algorithms

While CPU management is the most visible part of WLM it is not the only algorithm and in fact there are many more things to handle in a computer system. One big area is storage management. There are multiple algorithms which go hand in hand to assure that the pages which are needed by the work is in the central storage of the system. It starts from working set management which manages the amount of referenced storage of address space, through block paging which is an efficiency mechanism to reduce the paging in the system to the algorithm which finally directly control the amount of storage for each service class and address space. If WLM recognizes storage constraints by observing an increase in the amount of paging it is able to define protective and restrictive storage targets for address spaces. The underlying idea is that it is better to increase the paging for a controlled amount of work and on the other hand reduce the paging for other more important work. Restrictive storage targets tell the real storage manager an upper limit of pages which can be kept in central storage for work and a protective storage target tells it that a certain amount of pages should always be available.

If the system gets more and more constraint it is possible to swap work out of storage so that the remaining work can make better use of it. These mechanisms, named the Multi Programming Level, control from a higher view point how many address spaces of each service class are allowed to be in storage at every given point in time.

For the I/O subsystem WLM supports a set of priorities for the access to DASD devices and for the channel subsystem itself. For DASD devices it is important to find out which service classes use the same set of volumes in order to assign meaningful priorities. In case the captured data tells WLM that a service class encounters many I/O delays the first step is to identify other service classes which use the same DASDs. Between those it is now possible to set I/O priorities based on the importance, goal definitions

and I/O demand of the work. Another possibility becomes available with the exploitation of Parallel Access Volumes. It provides Alias UCBs to access the same logical volume by multiple I/O requests at the same time.

WLM also manages server address spaces for applications based on the number of executed units of work. The subsystems can use a set of programming interfaces to inform WLM about incoming work. In addition WLM supports a so called application environment which tells it how to start the server address spaces. Based on the incoming work requests and goal definitions of the service classes WLM starts and stops the server address spaces and optimizes the use of system resources for these applications.

4. Managing Work on z/OS

One of the biggest strength of WLM is its ability to recognize and manage units of work being executed by important middleware applications running on z/OS.

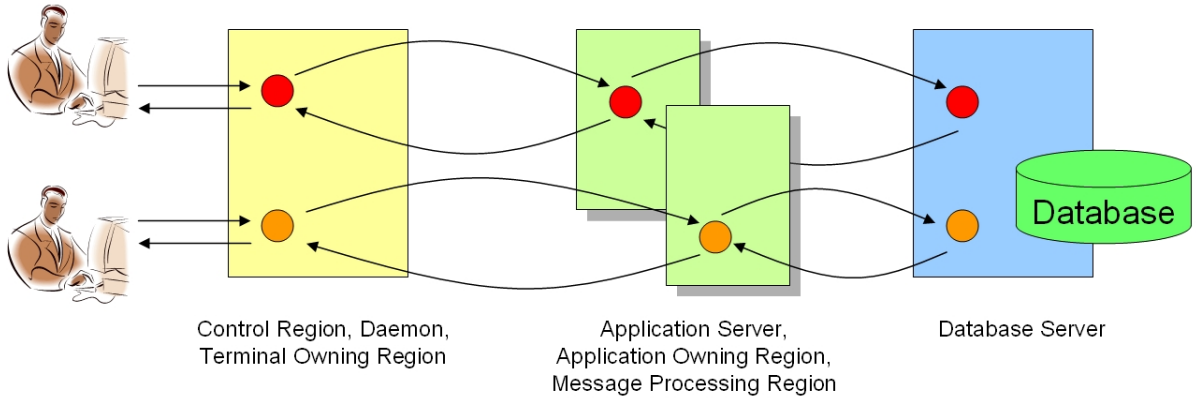

Figure 4.1.: Transaction Processing on z/OS

Figure 4.1 depicts the general structure of transaction processing on z/OS. Depending on the middleware or subsystem, one or multiple address spaces provide a connection to external clients. For CICS this is typically a Terminal Owning Region (TOR) and for Websphere for example a Control Region. External clients can be everything, for example a user sitting at a terminal or a program on a different server. The address space which receives the work request on z/OS primarily checks its authority and resource requirements and if the application can process it, the region sends it to an application processing region. For CICS these are named Application Owning Regions (AOR) and for IMS Message Processing Regions. The application region starts a program which processes the request and the program invokes other middleware, operating system services or services from other subsystems to complete the request. One typical example is that the request accesses a database, in many cases DB2 on z/OS.

What we can see from this example is that a client work request spans across multiple address spaces on z/OS. In order to manage these requests it is necessary to recognize the transaction flow and based on this to manage the address spaces and requests to meet the end user expectation. WLM developed a set of programming interfaces which allow middleware and subsystems to identify work requests to WLM, to help WLM to trace the work through the system and to identify the time the requests stay on z/OS (see [13]). Over time WLM developed various techniques to recognize work on the system and to manage it independently from or together with the address space were the programs run to process them.

4.1. Managing Address Spaces

The most basic variant is the address space. The start of an address space is known to WLM and if no more information is available about the work in the address space, WLM can manage it based on its resource consumption and demand. Address spaces are started and classified as started tasks and the only goal which is applicable for them are execution velocity goals or discretionary.

4.2. Managing Batch Work

Batch work re-uses already started address spaces. These are called initiators and they select work from the Job Entry Subsystems (JES). The beginning and end of each new batch job is signaled through a system interface to WLM. This allows WLM to recognize its duration, to provide classification rules for them and to manage the initiators based on the currently active job. Batch work can be managed through execution velocity and response time goals as well as discretionary work. Furthermore WLM is able to start the initiators on demand based on the goals for the batch service classes and the batch jobs waiting on the JES input queues. A similar technique is used for Advanced Program to Program Communication (APPC). APPC also has initiators available in the system which receive incoming work and the initiators tell WLM when a new work request begins and ends.

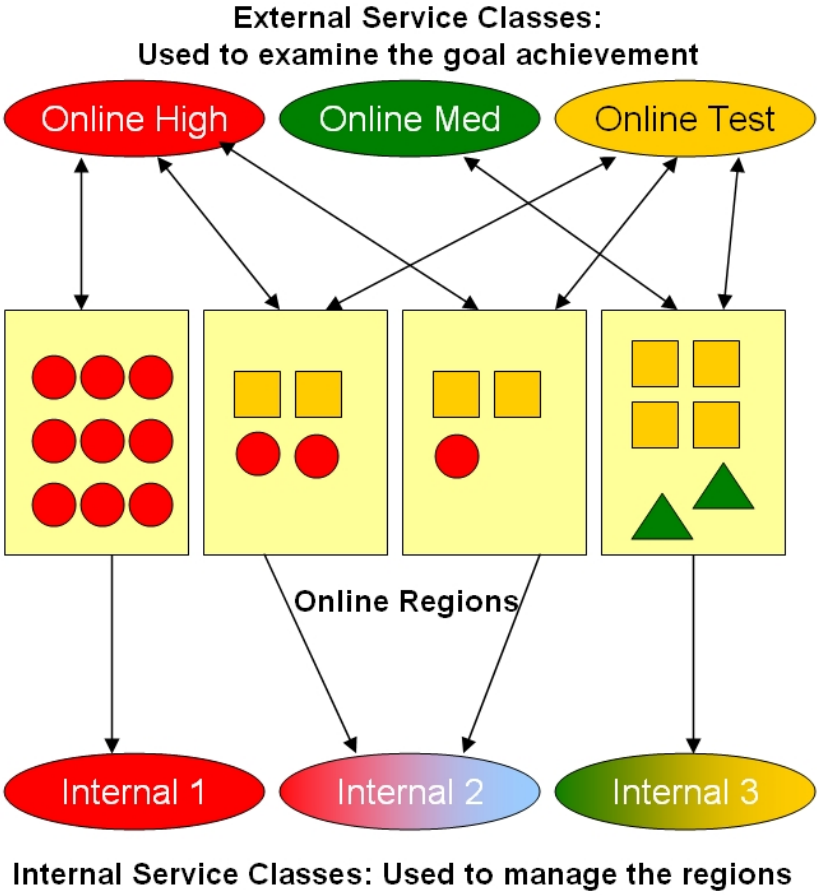

Figure 4.2.: CICS Server Topology

4.3. Managing TSO

TSO uses a different technique. After a TSO user address space has been started it waits on input from the TSO user terminal. When the user presses the Enter button a new TSO transaction starts and its end is defined when the result is returned to the terminal. The begin and end of TSO transaction is signaled to WLM so that it is able to recognize the individual transactions and to manage the TSO

address spaces based on them. Unix System Services (subsystem name is OMVS) uses the same interface to inform WLM about the beginning and the end of end user requests.

4.4. Managing CICS and IMS

CICS and IMS regions register as work managers to WLM and create control information - the performance blocks - for all work which can be executed in parallel in the region. When a work request enters the TOR or IMS Control Region it is classified and associated with a performance block. State conditions are recorded in the performance block during execution and when the request is passed from a TOR to an AOR the continuation is maintained and information is provided to WLM to understand the relationship.

WLM continuously collects data from the performance block and captures the beginning and end of all work requests to understand which region processes which type of work. The type of the work request is specified through work classification which associates the work request with a service class. WLM creates a picture of the topology including the server address spaces and the service classes of the work requests. It uses this topology to manage the server address spaces based on the goal achievement of the service classes for the work requests. Figure 4.2 depicts this topology. In this example there are three external service classes for CICS or IMS work and 4 registered server regions which process the requests. One region processes only requests for service class "Online High" (red circles). As a result one internal service class is created which is used to manage this region. Two regions process transactions depicted as red circles and turquoise rectangles. The second internal service class is used for these two server regions and the last region is associated with the third internal service class. Let's assume the external service class "Online High" misses its goals. Through the topology WLM finds out that 3 server regions process work requests of this service class which are associated with two internal service classes. Then it calculates how much each of the internal service classes contributes to the goal achievement of the external service class. Let's assume WLM found out that the server regions of the internal class 2 need help. WLM will help the server regions of this service class in order to improve the goal achievement of the external service class "Online High". When we examine this example we can see, that in this case the transactions for "Online Med" running in the server regions for class "Internal 2" also receive help even if they may not need it. This is a side effect of this kind type of transaction management. The benefit is that the installation can use response time goals for the work requests being processed by CICS and IMS. The example also demonstrates that too much granularity in defining service classes might not be helpful. A more complete description on how WLM manages CICS on z/OS can be found in [7].

4.5. Managing Websphere and DB2

A much more direct way of managing work requests which span multiple regions was introduced with enclaves. An enclave is an entity which encapsulates the execution units (TCBs and SRBs) which execute programs on behalf of the same work request. The address spaces which receive the work requests are again WLM work managers. For each new work request an enclave is created which is associated with a service class through the work classification process. The work manager either schedules a preemptible SRB into the enclave to execute the work request or passes the enclave plus execution information to a server region. The server region will pick up the request and the TCB which will execute it joins the enclave in order to get managed towards the goals of the service class for the work request. This technique has a lot of advantages. For CPU management WLM is able to directly manage the enclaves' independent from the address spaces where the execution units belong to. This allows giving different execution units of the same address space different dispatch priorities depending on the service class they belong to. In addition all execution units which execute programs on behalf of the same work request are managed together across all involved address spaces. Enclave management is exploited by DB2, Websphere and other major components and middleware applications on z/OS. Enclave management gives all functions to the end user in defining goals, monitoring the execution and observing the progress of work in the system. It is the most advanced and unique transaction management feature of its kind.

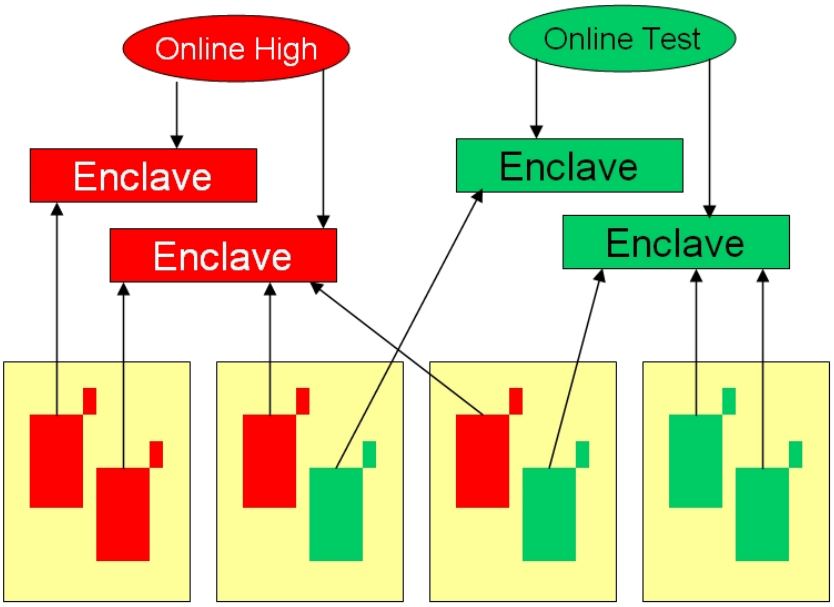

Figure 4.3.: Relationship between Execution Units and Enclaves

Figure 4.3 shows the relationship between service classes, enclaves and exceution units (TCB and SRBs). It is possible that multiple execution units join the same enclave.

5. Defining Service Classes and Goals

After discussing the internal mechanisms of WLM we will discuss what needs to be done to define a service definition and what maintenance is required to keep a service definition up to date. With setting up a service definition we will not discuss the operational steps and we will not discuss how to use the WLM Administrative Application. The basic setup and using considerations can be found in [11]. Further information on setting up a service definition is available in [12] and from the WLM homepage on the Internet [http://www.ibm.com/servers/eserver/zseries/zos/wlm/].

5.1. Setting up Service Classes

The first step which must be done is to separate work into distinct service classes. There are two competing requirements:

1. It is necessary to distinguish different work and to define service classes in a way that different groups of end users can get the service they need from the operating system.
2. In the previous discussion we saw that WLM examines every 10 seconds all service classes in order to find out which of them need help. If there are too many service classes it is possible that WLM needs too many adjustment cycles to really calculate all possibilities to help lower important work.

The result of these requirements tell us that on the one hand we want and should define service classes to distinguish all types of work and users from each other and on the other hand to restrict ourselves to help WLM to work efficiently. By following some guidelines it should be possible to satisfy both requirements:

- It is recommended to define a default service class for each major subsystem even it is not used today in the environment.
- On the other hand service classes should only be defined for work when sufficient demand exists for it in the system. The demand can be measured either by the service consumption or by the amount of ending transactions during high utilization periods of this work. Based on this it doesn't make sense to define service classes with a demand of less then 1% of measured service units at any period or less than 1 transaction ending per minute. In such cases it is most often better to combine this work with other work of the same type.
- It is not meaningful to create service classes for the same type of work with identical service objectives. For example if you define started tasks for different applications with exactly the same goals and importance you can combine them in the same service class and use report classes to differentiate them for reporting purposes.
- You should never use service classes for reporting purposes. WLM supports up to 999 report classes which should satisfy all reporting requirements.

Based on these simple rules it should be possible to always stay in a range of 25 to 30 service class periods with non-discretionary goals, even in environments with a high diversity of work on the systems. It is also no problem to define more service class periods as long as it is ensured that not all of them are active at the same point in time. 30 is not hard boundary. It might be very well possible that you run a sysplex environment with more active service classes. 30 is just a recommended number to avoid efficiency problems. It should also be noted that service classes with discretionary goals do not count because they are not subject of receiving additional resources.

5.2. Setting up Service Goals

For defining goals we will examine the pro and cons of each goal type.

5.3. Response Time Goals

WLM supports two types of response time goals: an average response time goal and a percentile response time goal. At this point we want to find out which might be better suited as a goal. The following picture shows a response time distribution captured by WLM for a service class. It must be explained that WLM always captures the distribution of ending transactions for its own decision process as well as for external examination. Externally this information is available through monitoring products like RMF. The only difference is that internally WLM uses a little bit higher granularity.

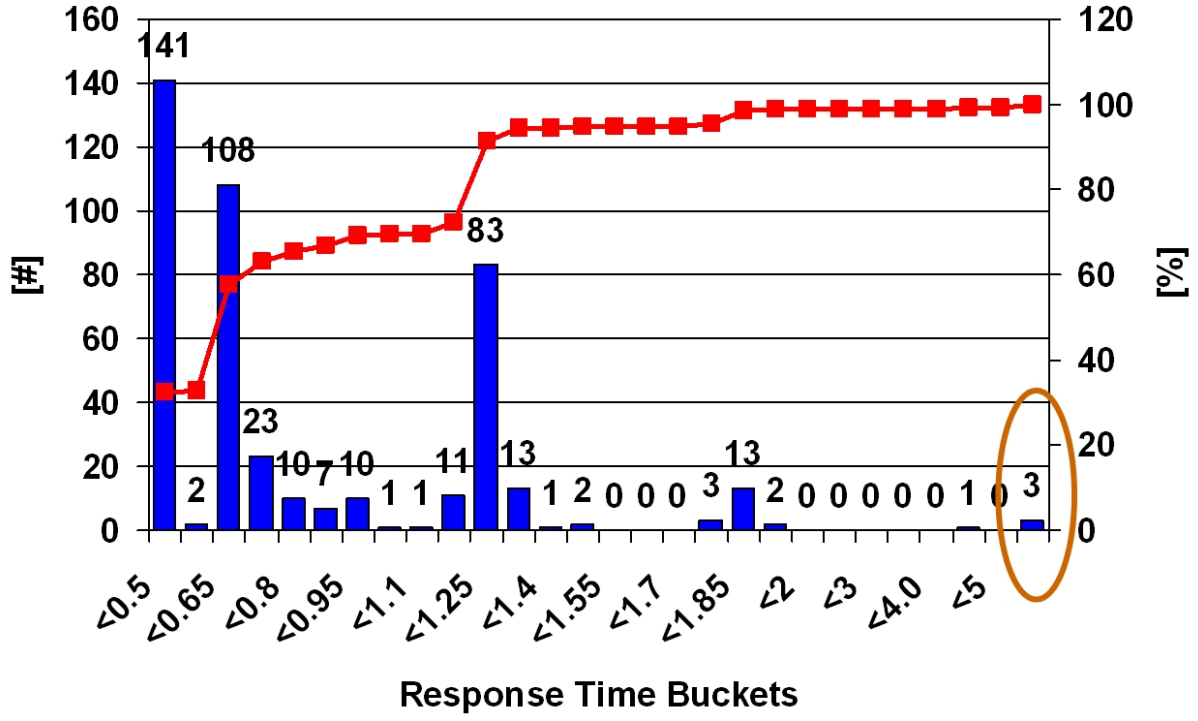

Figure 5.1.: Distribution of Ended and Running Transactions for a Service Class

Figure 5.1 shows an example where 435 transactions executed or ended within the time interval WLM uses for making decisions. By looking at the summary curve (red line), we can see that around 70% of all transactions ended within 1 second and about 90% of all transactions within 1.25 seconds. Only 10% of the transaction ran for a longer period of time or haven't completed yet. In this example the installation defined an average response time goal of 1s. Just by looking at the response time distribution one would expect that the average response time would be in the range of 1 to 1.5 seconds. In fact it was more than 10 seconds. This was because of the 3 transactions circled on the right side of the graph already executed for a very long period of time. We can immediately understand a disadvantage of an average response time goal. It can be very sensitive to long running outliers. By using a percentile response time of 90% of all transactions should end within 1s, the goal wouldn't be achieved too. But the difference is that the policy adjustment algorithm may see a better possibility to help the work. It is much more likely to

adjust the resource access for the work so that 90 transactions run a little bit faster than changing the resource access that it is possible to help some very long running transactions.

5.4. Execution Velocity Goals

The execution velocity goal defines the acceptable delay for the work when it runs through the system. The biggest difficulty for this goal is that the amount of acceptable delay is something which is inherited by the environment and can't be simply defined from an external expectation. For example let's assume we have an IMS service class with 100 Message Processing Regions running on a system with 5 processors. During times of high utilization we can expect that all regions want to execute work, which means all or many of the regions demand access to the CPU at the same time. Because all regions have the same dispatch priority they all compete on the same level. As a result we can't expect to see a too high execution velocity. A goal of 50 is probably unrealistic and we can most likely observe something in the range of 10 which would be a realistic goal. It must be noticed that during times of low utilization a much higher execution velocity can be observed. As a result from this observation we must conclude that a meaningful goal can only be observed during times of high system utilization, if possible as near as possible to 100%.

As a deviation of this example we can also assume that the achievable execution velocity decreases with the importance level of the service classes. It is more likely that during high utilization periods, service classes with a high importance will have a high dispatch priority those with a low importance will have a low dispatch priority. Consequently the low important work has to wait longer for the CPU or I/O devices and thus will have a lower execution velocity. Again it should be noticed that this can only be observed during periods of high system utilization. Otherwise the low important work will get fast enough access to the CPU so that high execution velocities are very well possible. The example with the traffic in the City and the comparison of traveling through the City on a Sunday morning versus the rush hour can be used as a good analogy.

Execution velocity goals are also very sensitive on the amount of samples WLM can capture for a service class. For example the following amount of samples taken for a work in the system result in the same Execution Velocity:

$$Execution\ Velocity = \frac{Using}{Using + Delay} = \frac{50}{50 + 100} = \frac{5}{5 + 10} = 33\%$$

If the system now captures 5 more using samples in each case the result in the example of the 150 samples will be nearly unchanged. The execution velocity will then be around 35%. In the example with the 15 state samples 5 additional using samples will change the result to a value of 50%. As a conclusion it is important for work being managed by execution velocity goals to show enough utilization in the system. This is important during times of high system utilization. During these times service classes with very sporadic and fluctuating utilization patterns can cause WLM to adjust their access to the resources even if this is not really helpful from an overall perspective. So again we come to the conclusion that it is beneficial for the system to create service classes with enough completions and a measurable utilization.

Further explanations and discussions about execution velocities can be found in [**?**].

5.5. Discretionary

A discretionary goal is disliked by many people but there is no reason for it. Discretionary work gets access to the resources if they are not needed by other work to achieve their goals. At least if the other work has a low goal definition. A low goal definition in this case means an execution velocity of less than 30 or a response time goal of more than 1 minute.

What does this exactly mean? It means that WLM tries to steal a little bit of resources from those

service classes and give it discretionary work if these service classes highly over achieve their goals. Highly over achieve means that these service classes have a PI of less than 0.7. What are the consequences? The consequence is that discretionary work can make progress in a system where it otherwise couldn't run because other work with "low" goal definitions consumes all of the resources.

There might be cases where this behavior is beneficial and other cases where this behavior is not wanted. In case it is not wanted a simple trick exists which allows to prevent discretionary work to steal CPU resources from other work with low goal definitions. It is only necessary to define a resource group without minimum and maximum value and associate all service classes which can potentially donate CPU resources to discretionary work with this resource group[1]. WLM will not "steal" any CPU resources from the work with low goal definitions.

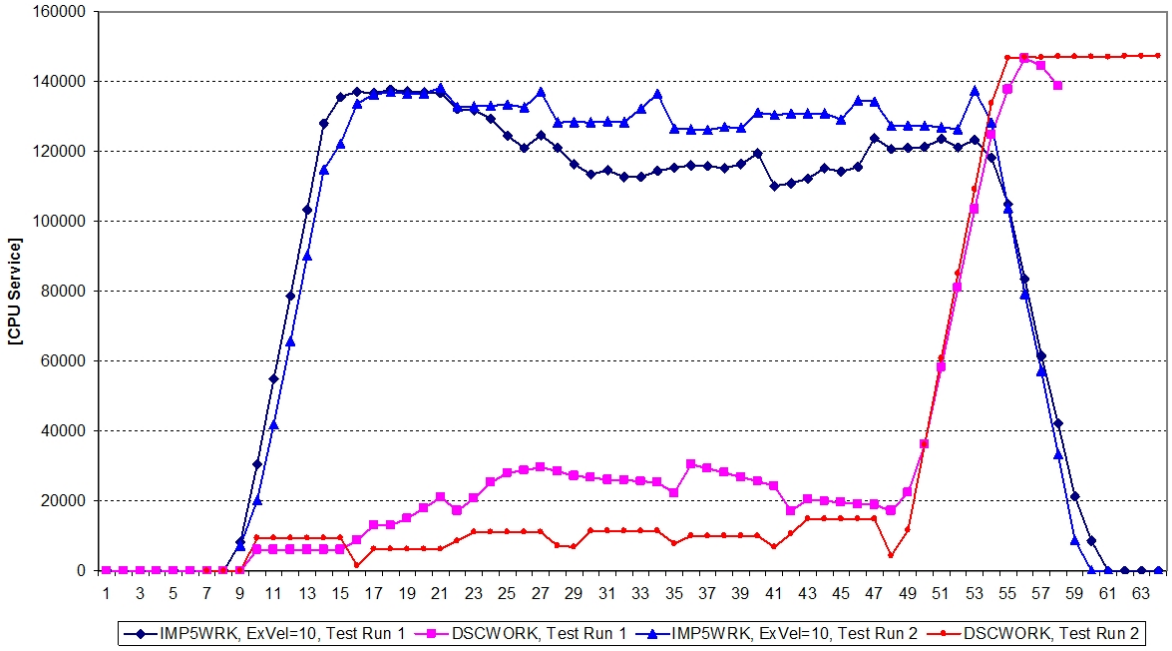

Figure 5.2.: Discretionary Goal Management

The question is also how much CPU resources will be stolen; in fact only a small amount. Figure 5.2 illustrates this behavior. It shows a service class with an execution velocity goal of 10, named IMP5WRK and a service class with discretionary goal, named DSCWORK. Test run 1 shows how much of the CPU resources are taken from IMP5WRK to allow DSCWORK to run on the system. On the right hand of the picture we can observe the demand of DSCWORK after IMP5WRK has been stopped. In test run 2 we use a resource group without minimum and maximum and associate IMP5WRK with this resource group. Now no CPU resources are taken from IMP5WRK. If we compare the amount of CPU service which is consumed by DSCWORK in both scenarios we see that less than 10% of the available CPU to IMP5WRK has been stolen in test run 1 to help DSCWORK.

As a result of this discussion we should notice:

- It is not harmful to define discretionary work.
- Discretionary work can borrow resources from work with goals as long as these resources are not needed to achieve the service class goals. This is only possible for work with low goal definitions and when they over achieve their goals. The amount of resources which is taken from this work is small and does not degrade the work with goal definitions.

[1] If a resource group is already used this is not necessary

- If this behavior is not wanted it is possible to implement a simple circumvention. This circumvention will allow discretionary work to run only when CPU resources are not needed otherwise.

5.6. Resource Groups

The idea of resource groups is to provide a mechanism to control the CPU service consumption of a set of service classes. Resource groups allow to define a minimum amount of service which should be made available to the service classes associated with them and to define a maximum which the set of service classes should not exceed. The minimum is useful to guarantee service to a set of service classes which can't be guaranteed simply by the WLM policy adjustment algorithm. A service minimum can only be guaranteed if no higher important work requires this service. But the minimum service is the last resort which is taken from the set of service classes when the system is constrained.

Since z/OS 1.8 three types of resource groups are supported:

1. Sysplex-wide resource groups which balance the resource consumption across the systems and service classes assigned to the resource group in a sysplex.
2. A resource group type with a sysplex-wide definition but a system scope. This resource group type allows defining a percentage of the partition weight as maximum and minimum limits.
3. A resource group type with a sysplex-wide definition and system scope which allows defining the percentage of logical processor capacity of the partition as maximum and minimum limits.

In the previous chapter we found a simple and useful way to exploit resource groups. There are others too but an installation should be warned that resource groups are a complicated control. They are discussed in much detail in [8] which can be obtained from the WLM team.

5.7. Protecting Work

Three options exist to protect the resource access of critical work:

1. Long term storage protection or storage critical option
2. Long term CPU protection or CPU critical option
3. Exemption from transaction response time management

We will discuss the last bullet in the next chapter and focus on storage and CPU protection for the moment. In section 3.2.1 we already saw a case where low important work could receive a higher dispatch priority than higher important work in the system. The scenario also showed that this situation was resolved within a minute but in some cases this might be too slow. The CPU Critical option has been introduced to prevent low important service classes to ever get a higher or equal dispatch priority than more important service classes. Similar storage protection can be used to prevent high important service classes to loose storage when it doesn't need it at the moment.

5.7.1. CPU Critical Option

Figure 5.3 shows the effect of CPU Critical in the system. The example shows 7 service class periods. For service class IMSPROD with an importance of 3 the CPU Critical attribute is defined. As a result all service classes with a lower importance, in this example the service classes BATHI, second period of TSO and BATLOW, will never receive a dispatch priority higher than IMSPROD.

WLM still adjusts the dispatch priorities dynamically. In this example it is assumed that WLM assigned a dispatch priority of 241 to IMSPROD. All service classes with an importance 4 and 5 can now only get a dispatch priority up to 239. The CPU Critical attribute is not defined to any other service class. That means all other importance 3, 2 and 1 service classes can compete for CPU resources with IMSPROD.

It is also still possible that WLM may reduce the dispatch priority of the other importance 1, 2 and 3 service classes below the importance 4 or 5 service classes. Nevertheless the layout and assignment of dispatch priorities becomes more static and the freedom of WLM is reduced. The advantage is that work in IMSPROD can never suffer from temporary bursts of batch work as it was shown in section 3.2.1.

Using CPU Critical can make sense for CPU sensitive work and if an installation is concerned that WLM may note react fast enough on sudden utilization changes. When it is used the most important thing is that it is not over used. That means it should not be used for too many service classes in the system.

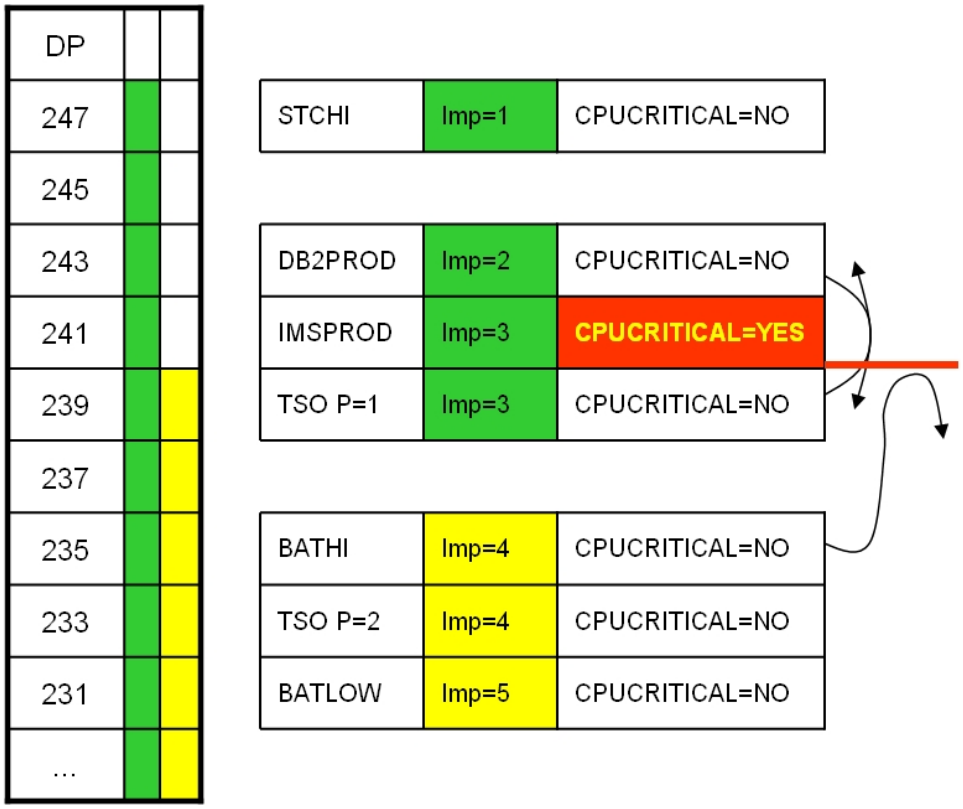

Figure 5.3.: CPU Critical

One important fact must be mentioned too. CPU Critical protects high important service classes in a way that the work of lower important service classes can't get a higher dispatch priority. That's all. There are many other decisions in the system besides the assignment of dispatch priorities. This means that CPU Critical work stills needs a meaningful goal. It should never be assumed that this option will provide preferential treatment for the protected service classes for other occasions than CPU management.

5.7.2. Storage Critical Option

Storage critical can be used to prevent the system from stealing central storage frames. Why is this necessary? During times of high workload utilization WLM protects the important work from loosing their storage frames. If work competes for storage WLM has sufficient mechanisms to ensure that paging occurs primarily for lower important work. But many on-line applications also have time periods of low utilizations for example during night time. During these times other work, for example batch work may run in the system and this work may also require a lot of central storage. Because the on-line applications obviously don't need their storage anymore, WLM allows it to page it out and gives the storage to the

batch applications. On the next day when the first users try to access the on-line applications again, their frames are no longer available and must be paged backed into central storage. This can cause page in delays for a measurable period of time and also can cause negative effects on the end user response times. In order to avoid this it is possible to protect the storage of critical address spaces and avoid that it is paged out during night shift. This is not absolutely true. As a security valve WLM allows that storage protected address spaces loose up to 25 pages or frames per hour, so a small reduction might be seen.

The following test scenario shows the effect of available storage and paging rates to work being protected by the Storage Critical attribute and work being unprotected.

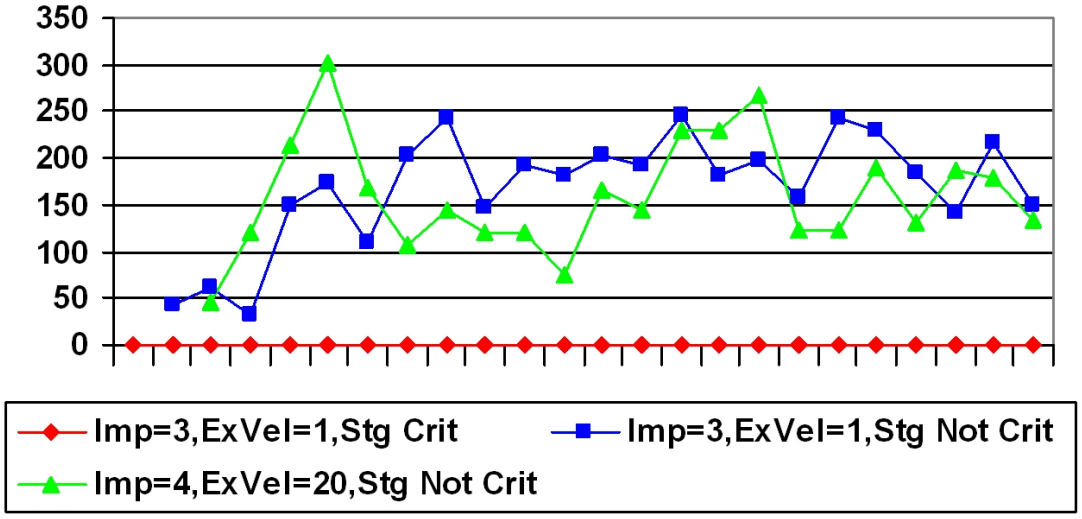

Figure 5.4.: Paging rate of service classes with and without storage critical

Figure 5.4 shows the paging rate of three service classes. In this test scenario 4 address spaces ran in each service class and all address spaces have the same storage requirements. Two service classes have an importance of 3 and a very low execution velocity goal of 1 and the third service class has an importance of 4 with an execution velocity goal of 20. One of the importance 3 service classes is storage protected the other two service classes are unprotected. We can observe that the storage protected service class doesn't show any paging while the unprotected service classes show high paging rates of up to 300 pages per second. Figure 5.5 displays the occupied storage of each service class. The storage critical service class is able to keep its 9000 pages constantly in storage while the other service classes have to share the remaining storage between each other.

What we can conclude from this example is that Storage Critical has an ever stronger effect on system performance than CPU Critical. The storage of the protected service classes is virtually taken away from the available storage for all other workloads. This is true until the system may run into storage shortages. At this point WLM does no longer grant storage protection and also steals from storage protected work. Also all higher important work can steal from lower but storage protected address spaces. But this will not happen until WLM squeeze all non protected lower important to the minimum. As a result it should be noticed that storage protection should be used with even more care than CPU protection and it should really only be used if there is an absolute necessity for it and not as a simple prevention because the installation does not trust WLM.

5.8. A Bad Example

At this point we will take a look at the service classes of an installation and find out what can be done incorrectly. Table 5.1 shows the service classes and goal definitions. We see that 19 service classes have

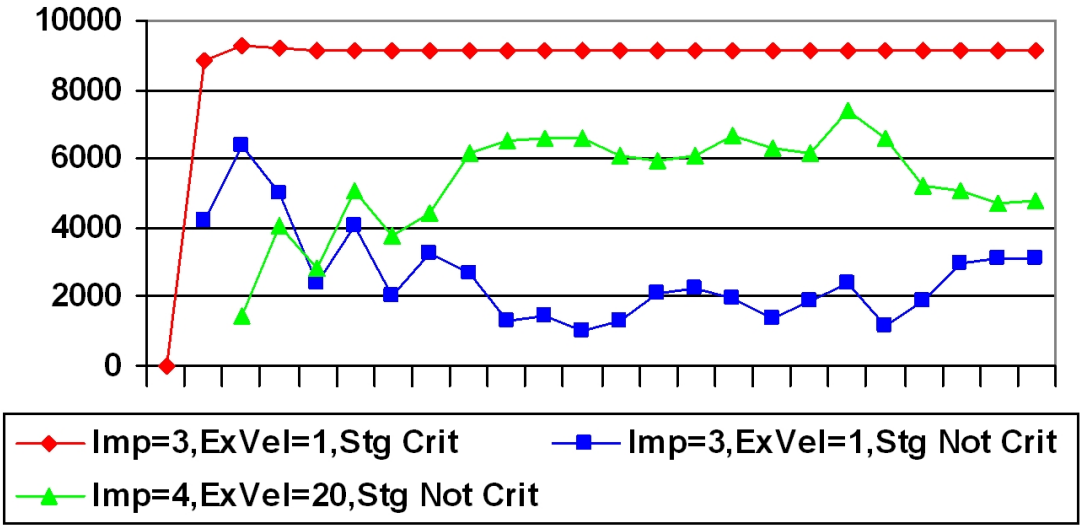

Figure 5.5.: Storage consumption of service classes with and without storage critical

been defined. These service classes use primarily 3 importance levels, 1, 2 and 3. In addition we see very high execution velocity goals and an excessive use of the CPU critical attribute.

Let's take a look at the potential reasons for this service definition layout and its affects:

- The high execution velocity goals of 80 and more can't be achieved by the work in the system. So the performance index of this work is constantly far above 1. The high execution velocity was assigned to these service classes in order to force a high dispatch priority for them.
- The high execution velocity goals for the important service classes became necessary because the dominant workloads IMSMED, STCMED and BTCHHIGH also have very high goal definitions.
- In the end the installation was not satisfied because no service classes met their goals anymore. The installation became anxious and used the CPU Critical attribute to protect the important work from all other. This resulted in a situation that nearly everything was protected.

It should be noticed that this does not automatically mean that the system doesn't run anymore. First of all it means that WLM is not able to manage the dispatch priorities for most of the work. And it means that WLM can't react efficiently on changes in the workload demand.

Now the question what can be done to leave this spiral of defining more and more aggressive goals?

1. The service classes should make better use of all available importance levels. Importance 4 is not used at all and measurement data showed that nothing was really executed in BTCHLOW. The first step is to spread the service classes across the available importance levels.
2. Measurements must be taken at high utilization times in order to find out what are realistic execution velocities for the defined service classes. This is very often not simple. Many installations run only with a utilization of less than 90% and the achieved execution velocities can be quiet high during these times. So it is necessary to take data from the end of the month, the end of a quarter or even the end of a year to find really high utilization periods. In many cases you might be surprised how low the achieved execution velocities will be during these periods.
3. The installation uses IMS and there is nothing wrong with using response time goals for IMS . Response time goals do not provide a better performance but they provide much easier data for analysis purposes. The response time can be much more easily derived and related to an end user expectation as this is possible with execution velocity goals. We will discuss an approach to set up response time goals for CICS and IMS in a following chapter.

Workload	Service Class	Imp	Goal	CPU Crit
STC	STCTOP	1	**Exvel = 95**	**YES**
TSO	TSOOPER	1	**Exvel = 90**	**YES**
STC	STCHIGH	1	**Exvel = 95**	**YES**
ONLINE	MQ	1	RT: 90% < 0.5 sec	No
ONLINE	IMSHIGH	1	*Exvel = 80*	**YES**
STC	SPECIAL	1	*Exvel = 65*	**YES**
ONLINE	WEBONL	1	RT: Avg = 0.2 sec	**YES**
ONLINE	IMSMED	2	*Exvel = 70*	**YES**
OMVS	OMVSNRM	2	*Exvel = 70*	**YES**
APPC	APPCNRM	2	*Exvel = 70*	**YES**
BATCH	BTCHSPEC	2	*Exvel = 65*	No
STC	STCMED	3	Exvel = 50	No
BATCH	BTCHTOP	3	Exvel = 40	No
STC	STCLOW	3	Exvel = 40	No
BATCH	BTCHHIGH	3	Exvel = 30	No
TSO	TSONORM	3	Exvel = 30	No
BATCH	BTCHNORM	3	Exvel = 20	No
BATCH	BTCHMED	3	Exvel = 20	No
BATCH	BTCHLOW	4	Exvel = 10	No

Table 5.1.: Sample Service Policy

4. Finally if it is sufficient to use CPU Critical for few service classes which really need protection. It is even better to use CPU Critical and realistic goals instead of such high execution velocity goals to enforce a certain order of dispatch priorities. Nevertheless the first attempt should be to re-define the service definition without the usage of CPU Critical and then set it for no more than one or two service classes

5.9. System Service Classes

We briefly discussed the system service classes SYSTEM and SYSSTC in section 3.2. All system related address spaces should be classified to service class SYSTEM and all address spaces which either support the system or the operation of it to SYSSTC. It is strongly recommended to read chapter "Using System Supplied Service Classes" in the WLM Planning manual [11] and use the "System Parameter (SPM)" qualifier for the STC subsystem to classify all system tasks to SYSTEM or SYSSTC. System tasks are those given the privileged and/or system task attribute in the IBM-supplied program properties table or in the SCHEDxx parmlib member.

Figure 5.6 shows the classification rules panel of the WLM Administrative Application.

The example shows the use of the SPM rules. In addition it is recommended to classify at least PCAUTH, TRACE and SYSBMAS to SYSSTC too. In addition this example shows that the IRLM address spaces are classified to SYSSTC. This makes a lot of sense. In general IRLM requires fast access to the CPU and does not consume too much of it. Therefore it is predestined to get classified to SYSSTC. A similar rule can be used for other work:

- If the work is very CPU sensitive, uses only very little of it (less than 5% of one CP) and does not show abrupt and high CPU demand it can be classified to SYSSTC.

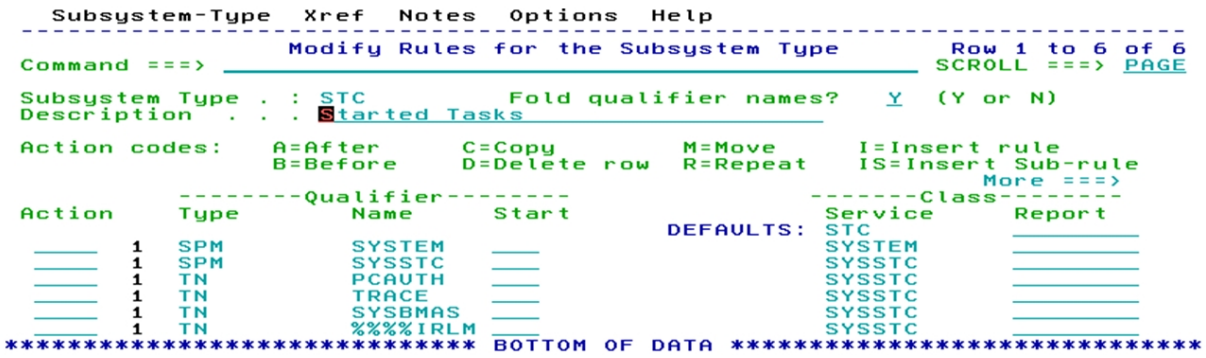

Figure 5.6.: Make best use of System service classes

This rule can be followed as long as the total amount of work in SYSSTC does not increase too much. An installation should try to keep the total CPU consumption of the SYSTEM and SYSSTC service classes below 10 to 15% of the total system capacity.

A third service class named SYSOTHER is pre-defined in the system. It has a discretionary goal and by default all work other than started tasks is assigned to it. An installation should try to classify all work so that nothing runs in SYSOTHER. If then work shows up in SYSOTHER it is an indication that something isn't classified and needs attention. The example is contained in the sample service definition IWMSSDEF which is shipped with z/OS.

5.10. Response Time Goals for CICS and IMS

At the end of this document we will discuss how response time goals for CICS and IMS can be used by an installation. Many installations still do not use them. In the past the main reason was that either all CICS, IMS work must be managed towards response time goals or towards execution velocity goals. But this is no longer true. For all z/OS releases it is possible to exempt regions from being managed towards response time goals. This is meaningful under certain conditions, for examples if a test and production environment runs on the same sysplex. For the test environment there is very often no need to have production like goals, everything what is necessary are goals to keep the running when it is present.

Another reason for not using response time goals was the difficult migration. Response time data was only really available after the goals had been enabled. There was no possibility to capture the response time while the regions were still managed towards execution velocity goals. With z/OS V1R3 a new level of report classes has been introduced. They allow now to report transaction completions and response time distributions even if the regions are still managed towards execution velocity goals.

At the end the question remains why an installation should use response time goals. Nobody should expect that the work will run faster when this feature is exploited. Most value can be seen in obtaining more and better reporting data and more easily relating the CICS and IMS transactions to the end user response times.

In order to exploit the feature the following steps can be helpful:

1. Define initially one service class for IMS and if you use CICS, a separate service class for CICS. The goals are not very important and a good guess is sufficient.
2. Define a set of report classes based on the granularity you want to obtain for your CICS and IMS work. If you have no special requirements a good starting point is to define a report class for each subsystem instance (SI) of the work managers .
3. Set up the classification rules for subsystem IMS (and/or CICS). Use the service class as default

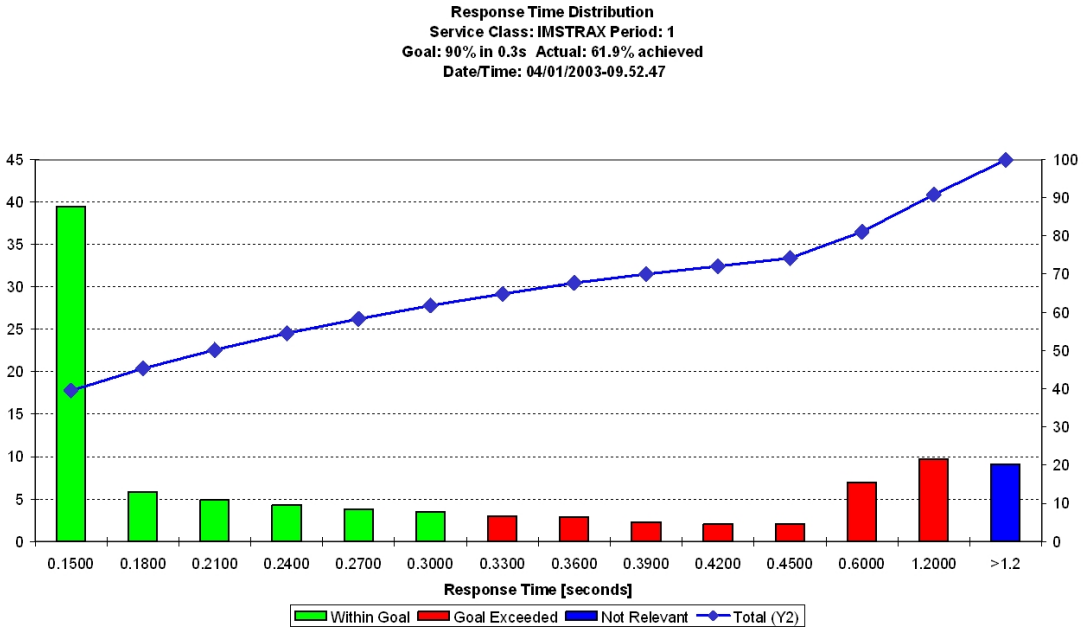

Figure 5.7.: Response Time Goals for CICS and IMS

and define one rule for each subsystem instance to associate them with report classes.

4. Modify the classification rules for the STC or JES subsystem. This depends on how you start your regions for IMS or CICS. For all regions change the attribute under column "Manage Regions using goals of" from TRANSACTION to REGION. This ensures that all regions are still managed towards the service classes with the execution velocity goals which currently exist.

5. Install and activate the new service definition and policy.

After this point nothing has changed but you are now able to obtain response time data for your CICS and IMS work. Based on the response time distribution which can be obtained from RMF reports it is now possible to adjust the initial goal of the service class, to examine the report classes and perhaps to define additional service classes and so forth. The graphic above shows a response time distribution for an IMS transaction service class. This report is available through the RMF Spreadsheet Reporter and is an example of how easy it is to examine and evaluate the data.

Once you become confident with the new data you can switch the management for the regions back to TRANSACTION. At that point the work becomes managed towards the response time goals of the IMS and CICS transaction service classes.

With APAR OA35428 for z/OS 1.11 and above another option is available to manage CICS work. This allows differentiating between TORs which function as work receiver and AORs which consume and process the work requests. Typically a work receiver only performs short processing steps for the work requests and then hands the request to a region which starts programs to process the request and which consume much more CPU. Such programming models are also exploited by DB2/DDF and Websphere and in these cases the work receiver, either the DDF region or the Websphere Control regions are differentiated from the work consumer regions. This programming model has advantages if the transactional workload is the only work executing on the system. In order to support this function also for CICS a new option has been introduced which allows exempting the CICS TORs from being managed towards response time goals but the option also ensures that the CICS environment is managed towards the response time goals. A complete description of this option can be found in [7].

6. Performance Monitors and Reporters

Before we complete our small review of WLM we will briefly discuss how an installation can use a performance monitor and reporting products to supervise and setup goals for WLM. From the previous sections we saw that we primarily looked at the using and delay samples for service classes, the response times of transactions, the performance index and the utilization of the service classes and the system. These are exactly the indicators which are most important to define goals. Unfortunately performance monitoring and reporting products provide other indicators too which sometimes are not very useful anymore to understand WLM. One of the most misused and misunderstood value is the dispatching priority. In the previous chapters we saw that WLM assigns it dynamically to service classes, based on their CPU demand, the CPU utilization and service consumption, the goal settings and goal achievement. That means it is completely unpredictable for an observer to understand exactly at what time which dispatching priority might be assigned to a service class. This is also true if CPU Critical is used and not overused in a system and especially true if response time goals have been specified for CICS and IMS transactions. In the latter case the address spaces which process the transactions are managed towards the goals of the service classes for the transactions and as we saw before the effective dispatching priority depends on the factors above as well as the mixture of the transactions being executed by these address spaces.

Bottom line: It is very important to understand that it is not useful to monitor dispatching priorities and then trying to set the goals based on such observations. The result can be very easily a service definition as described in section 5.8.

There is another difference between WLM and performance monitoring and reporting products which make it difficult to look to deep into WLM and to try to understand how the internal algorithms work. WLM collects data every 250ms and assesses the system state every 10 seconds. In fact it does even more; some algorithms run every 2 seconds and other parts on a much more frequent basis. Performance monitors and reporting products usually collect data on a 1 second basis and present the data on much longer time intervals. These monitors and products show a long term picture of a set of WLM decisions. That's absolutely sufficient for monitoring the goal achievement of work and to revisit goal settings on a periodic basis. It is also sufficient to use the goal achievement over a longer time interval, for example 15 minutes as the measure whether goals are missed or met and then take actions based on observations and trends of multiple such intervals. It is not useful to monitor each instance and to try to foresee which action WLM might take next. In case there is an indication that WLM does not function properly it is possible to collect SMF 99 records after reconsulting IBM support and send them to IBM for further analysis. Otherwise it is not recommended to collect SMF 99 data periodically.

7. Summary

At this point we will complete our short overview of the basic WLM constructs. You should get some additional information from this paper in addition to the documents listed in the Literature section. It is highly recommended that you read the WLM redbook (see [12] and the WLM Planning manual [11] before you start modifying your service definitions. For further reading the documents [2] and [3] are recommended as well as the WLM redbook [12]. Additional documentation can be found on the internet at

WLM http://www.ibm.com/servers/eserver/zseries/zos/wlm/

IRD http://www.ibm.com/servers/eserver/zseries/ird[1]

SWPRICE http://www.ibm.com/servers/eserver/zseries/swprice/[2]

WSC http://www.ibm.com/support/techdocs[3]

One important part of revisiting your goal definitions is the necessity to measure the goal achievement of your work and to analyze the workload behavior. If you do not have a tool for this purpose it is recommended to take a look at

RMF http://www.ibm.com/servers/eserver/zseries/zos/rmf/

and download the RMF Spreadsheet Reporter. It helps you to do trend and workload analysis based on RMF SMF data.

This article does not focus on any particular enhancement related to WLM. There is much more, like Intelligent Resource Director which expands WLM capabilities to the LPARs running on the same Central processing Complex, Parallel Access Volumes which were briefly mentioned and which dramatically improve the I/O performance of a z/OS system, and Batch Management which underwent a set of improvements with z/OS 1.4 and gives you the ability to manage your batch work efficiently in a sysplex environment.

[1] Intelligent Resource Director
[2] Software Pricing
[3] Washington Systems Center

A. Trademarks

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

AIX®, AS/400®, BatchPipes®, C++/MVS, CICS®, CICS/MVS®, CICSPlex®, COBOL/370, DB2®, DB2 Connect, DB2 Universal Database, DFSMS/MVS®, DFSMSdfp, DFSMSdss, DFSMShsm, DFSORT, e (logo)®, ECKD, ES/3090, ES/9000®, ES/9370, ESCON®, FICON, GDPS, Geographically Dispersed Parallel Sysplex, HACMP/6000, Hiperbatch, Hiperspace, IBM®, IBM (logo)®, IMS, IMS/ESA®, Language Environment®, Lotus®, OpenEdition®, OS/390®, Parallel Sysplex®, PR/SM, pSeries, RACF®, Redbooks, RISC System/6000®, RMF, RS/6000®, S/370, S/390®, S/390 Parallel Enterprise Server, System/360, System/370, System/390®, System z, ThinkPad®, UNIX System Services, VM/ESA®, VSE/ESA, VTAM®, WebSphere®, xSeries, z/Architecture, z/OS, z/VM, zSeries

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
- Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.
- 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.
- Red Hat, the Red Hat "Shadow Man" logo, and all Red Hat-based trademarks and logos are trademarks or registered trademarks of Red Hat, Inc., in the United States and other countries.
- SET and Secure Electronic Transaction are trademarks owned by SET Secure Electronic Transaction LLC.

B. Glossary

A

APPC *Advanced Program to Program Communication* - Protocol for program to program communication between multiple systems

ASCB *Address Space Control Block* - z/OS control block which represents a virtual storage entity tight to an end user or set of programs to execute

B

BCP *Basic Control Program* - z/OS or MVS kernel routines

C

CCW *Channel Command Word* - Defines an I/O operation (read, write, control) to be performed on an I/O device

CDS *Couple Dataset* - Dataset which contains control information to setup a parallel sysplex environment

CEC *Central Electronic Complex* - The system (processors, memory, I/O adapters), not including I/O devices

CFCC *Coupling Facility Control Code* - Operating System of the coupling facility

CHPID *Channel Path Identifier* - Identification of the channel path, typically a number

CICS *Customer Information Control Server* - A transaction monitor that runs primarily on z/OS

CISC *Complex Instruction Set Computing* - Processing architecture which contains many complex instructions which perform functions like small programs

CKD *Count Key Data* - System z disk architecture

CP *Central Processor* - Standard processor of a System z

CPU *Central Processing Unit* - see CP

CSS *Channel Subsystem* - The heart of moving data in and out of of a mainframe

CSSID *Channel Subsystem Identifier* - Number which identifies the Channel Subsystem in case multiple exist

D

DASD *Direct Access Storage Device* - A storage device which supports direct access (typically a disk)

DB2 *Database* - Relational database based on E. F. Codd's theory of relational databases

DDF *Distributed Data Facility* - Component of DB2 to exchange information with external clients

DEDB *Data Entry Database* - Fast path database for IMS

DL/I *Data Language Interface* - Database of IMS

DRDA *Distributed Relational Database Architecture* - Distributed database architecture of the open group standard

E

ECKD *Extended Count Key Data* - incorporates fixed-block and CKD architecture

EMIF *ESCON Multiple Image Facility* - Feature which allows to use ESCON channels from multiple partitions

ESA/390 *Enterprise Systems Architecture/390* - 32-bit predecessor of System z architecture

ESCON *Enterprise System Connection* - Half-duplex optical fiber serial channel

ESPIE *Extended Specify Program Interruption Exit* - Interrupt exit routine

ESTAE *Extended Specified Task Abnormal Exit* - Recovery routine for z/OS user or problem state programs

ESTI *Enhanced Self-Timed Interface* -

ETR *External Time Reference* - Device to synchronize all TOD (time-of-day) clocks in a cluster environment (Parallel Sysplex)

EXCP *Execute Channel Program* - z/OS macro to execute an I/O operation

F

FCP *Fibre Channel Protocol* - Transport protocol for transporting SCSI commands on Fibre Channel networks

FICON *Fibre Channel Connection* - Full-duplex fibre optical serial channel

FIFO *First In, First Out* - Queuing mechanism

FLIH *First Level Interrupt Handler* - Interrupt handler that gets immediate control when the interrupt occurs (where the new Program Status Word points to)

FRR *Functional Recovery Routine* - Recovery routine for z/OS system programs

G

GDPS *Global Dispersed Parallel Sysplex* - Parallel Sysplex which is spatially distributed to ensure high availability

GRS *Global Resource Serialization* - z/OS subsystem which supports global lock management

H

HCD *Hardware Configuration Dialog* - z/OS component to define I/O devices to the system

HFS *Hierarchical File System* - UNIX file system on z/OS

HMC *Hardware Management Console* - Console to access and manage hardware components of System z

HWA *Hardware Address* -

I

I/O *Input/Output* - Abbreviation for all parts which send data to and from an electronic complex

ICB *Integrated Cluster Bus* - Bus for connecting system in a parallel sysplex for short distance. The bus relies on few parts and provides very high speed and reliable connectivity

ICF *Integrated Coupling Facility* - Processor on system z which allows to run coupling facility control code

IFL *Integrated Facility for Linux* - Processor on system z which allows to execute z/VM and Linux operating systems

IML *Initial Microcode Load* - Initialization process of System z hardware. At its completion, operating systems can be booted (IPLed).

IMS *Information Management System* - A transaction monitor and database for z/OS (introduced 1968 for the Apollo space program)

IOCDS *Input/Output Configuration Data Set* - Data set which contains hardware I/O definitions (related to IODF; also should be consistent with IODF)

IODF *I/O Definition File* - Data file which contains the I/O definitions for System z (created by HCD)

IPC *Inter Process Communication* - Protocol for system processes to interact which each other

IPL *Initial Program Load* - Process to start the z/OS operating system

IRB *Interrupt Request Block* - z/OS Control Structure to start an I/O routine

IRD *Intelligent Resource Director* - A combination of multiple z technologies to enhance the autonomic capabilities of PR/SM, z/OS and the I/O subsystem

IRLM *IBM Resource Lock Manager* - Lock manager for DB2 and IMS

ISPF *Interactive System Productivity Facility* - End user interface for TSO users

J

JES *Job Entry System* - z/OS subsystems which support the execution of scheduled programs

L

LCP *Logical Processor* - Representation of a processor to the virtual system or logical partition

LCSS *Logical Channel Subsystem* - A system may use multiple logical channel subsystems (currently up to 4) to increase connectivity

LDAP *Lightweight Directory Access Protocol* - Application protocol for accessing and maintaining distributed directory information services over an IP network

LIC *Licensed Internal Code* - System z microcode or firmware

LICCC *Licensed Internal Code Configuration Control* -

Ln *Level n Cache* - L1 is closest to the processor, the highest number is used to describe memory (”storage” in System z terminology)

LPAR *Logical Partition* - Container which hosts an operating system to execute on System z virtualization layer. Up to 60 LPARs are supported

M

MBA *Memory Bus Adapter* - I/O hub chip used on z10 and earlier machines. No longer used on z196.

MCM *Multi-Chip Module* - contains processor and storage controller chips

MCSS *Multiple-logical-Channel Subsystem* - Restructuring of the physical CSS into multiple logical instances in order to enable more external devices to be attached to the CEC

MLCS *Multiple Logical Channel Subsystems* - see MCSS

MMU *Memory Management Unit* - Hardware component which handles virtual memory

MPL *Multi Programming Level* - Term which expresses the ability of workload to access system resources

MSU *Million of Service Units per Hour* - Unit to measure CPU capacity on System z

MTTW *Mean Time To Wait* - Algorithm which gives access to units of work based on their deliberate wait time

MVS *Multiple Virtual Storage* - Original name of z/OS based on the ability to support multiple applications in virtual storage

O

OLTP *Online Transaction Processing* - Umbrella term for transaction processing

OSA *Open System Adapter* - Networking adapter

P

PAV *Parallel Access Volume* - Protocol which supports parallel access to the same I/O device

PCHID *Physical Channel Identifier* - identifies the plug position of a channel adapter

PCP *Physical Processor* - see CP

PPRC *Peer to Peer Remote Copy* - A protocol to replicate a storage volume to a remote site

PR/SM *Process Resource and System Manager* - Management component of the logical partition technology of System z (alias for LPAR hypervisor)

PSW *Program Status Word* - Central register to control all program execution

PU *Processing Unit* - Physical processor

Q

QDIO *Queued Direct I/O* - memory to Memory I/O mechanism between LPARs on System z

R

RACF *Resource Access Control Facility* - z/OS subsystem which supports access control

RAS *Reliability, Availability, Serviceability* - Terminology to depict the robustness of information technology systems (originated from IBM mainframe)

RETAIN *Remote Technical Assistance Information Network* - IBM network to handle service requests for end users

REXX *Restructured Extended Executor* - Interpretive Execution Language from IBM

RISC *Reduced Instruction Set Computing* - Processing architecture which only contains elementary instructions like LOAD, STORE, and register-to-register operations

RLS *Record Level Sharing* - VSAM access method which introduces record sharing and serialization

RMF *Resource Measurement Facility* - z/OS Performance Monitor

RRMS *Resource Recovery Management Services* - z/OS component to synchronize the activities of various syncpoint managers

RSF *Remote Support Facility* - Part of HMC to report and repair hardware and firmware components

S

S/360 *IBM System/360* - Is a mainframe computer system family announced by IBM on April 7, 1964. It is the computer architecture of which System z is the current incarnation.

SAP *System Assist Processor* - System z I/O processor

SCC *Storage Controller Control* - Storage controller chip

SCD *Storage Controller Data* - Cache chip

SCE *Storage Control Element* - Controls access to main storage data by processor unit

SDWA *System Diagnostic Work Area* - Control structure to capture information in case of an abnormal program termination

SE *Support Element* - Laptop that acts as user interface to System z machine

SIE *Start Interpretive Execution* - Instruction to drive a processor in a logical partition (LPAR) or virtual machine (z/VM)

SIGP *Signal Processor* - Instruction to inform a processor about status change

SLE *Session Level Encryption* - Encryption between originator and receiver of data across all network elements

SLIH *Second Level Interrupt Handler* - Term which encompasses a set of specialized interrupt handling routines

SMF *Systems Management Facility* - z/OS component which supports performance and status logging

SMP *Symmetric Multiprocessing* - A computer system with all physical processors accessing the same storage and I/O subsystems

SRB *Service Request Block* - Control structure to execute a z/OS system program

SRM *System Resource Manager* - Component of z/OS for resource management (introduced 1974, now part of WLM)

STP *Server Time Protocol* - Follow-on to ETR

SU/sec *Service Unit per second* - Capability of a System z processor to execute instructions

SVC *Supervisor Call* - Interface to invoke a z/OS system program

Sysplex *System Complex* - A single logical system running on one or more physical systems

System z *IBM mainframe computer brand* - Current 64-bit incarnation of the S/360 architecture

T

TCB *Task Control Block* - Control Structure to execute user or problem state programs on z/OS

TSO *Time Sharing Option* - z/OS component which supports the parallel execution of multiple end users on MVS

U

UCB *Unit Control Block* - z/OS control structure which represents an I/O device

UoW *Unit of Work* - An execution unit on z/OS

USS *Unix System Services* - z/OS component which supports a full functioning UNIX environment on z/OS

V

VCPU *Virtual CPU* - see LCP

VMM *Virtual Machine Monitor* - Hypervisor or control program to run multiple virtual machines

VSAM *Virtual Storage Access Method* - A set of access methods for System z I/O devices

VTAM *Virtual Terminal Access Method* - Access method for communications devices (now part of z/OS TCPIP subsystem)

VTOC *Volume Table of Content* - Index of a DASD device

W

WLM *Workload Manager* - Central z/OS component for resource management (introduced 1995)

X

XCF *Cross System Coupling Services* - z/OS Services which support the exploitation of a z/OS sysplex

XES *Cross System Extended Services* - z/OS services which support the access to the coupling facility

XRC *Extended Remote Copy* - System z protocol for data replication

Z

z114 *zEnterprise 114* - Mid-range end model of System z processor family (2011)

z196 *zEnterprise 196* - High end model of System z processor family (2010)

zEC12 *zEnterprise EC12* - High end model of System z processor family (2012)

zAAP *System z Application Assist Processor* - System z processor to execute Java code. This processor type can only be used by z/OS and only for instrumented software like the Java Virtual Machine. A special instruction tells the dispatcher when Java execute starts and ends.

zFS *System z File System* - UNIX file system on z/OS

zIIP *System z Integrated Information Processor* - System z processor to execute code which is subject to get offloaded from regular processors. The offload capability is described by the middleware through an interface to WLM and the z/OS dispatcher. Exploiters are middleware like DB2 and TCPIP.D5

Bibliography

[1] *TSO Time Sharing Option im Betriebssystem z/OS*, Dr. Michael Teuffel, Oldenbourg, 2002, ISBN-13: 978-3486255607

[2] *Das Betriebssystem z/OS und die zSeries: Die Darstellung eines modernen Großrechnersystems*, Dr. Michael Teuffel, Robert Vaupel, Oldenbourg, 2004, ISBN-13: 978-3486275285

[3] *High Availability and Scalability of Mainframe Environments*, Robert Vaupel, KIT Scientific Publishing, 2013, ISBN-13: 978-3-7315-0022-3

[4] *In Search Of Clusters, The Ongoing Battle in Lowly Parallel Computing*, Gregory F. Pfister, Prentice Hall, 1998, ISBN 0-13-899709-8

[5] *Adaptive Algorithms for Managing A Distributed Data Processing Workload*, J. Aman, C.K. Eilert, D. Emmes, P. Yocom, D. Dillenberger, IBM Systems Journal, Vol. 36, No. 2, 1997, Seiten 242-283

[6] *MVS Performance Management (ESA/390 Edition)*, Steve Samson, J. Ranade IBM Series, Printed and bound by R.R.Donnelley and Sons Company, ISBN 0-07-054529-4, 1992

[7] *z/OS Workload Manager Managing CICS and IMS Workloads* Robert Vaupel, March 2011, http://www-03.ibm.com/systems/z/os/zos/features/wlm/WLM_Further_Info.html

[8] *Resource Groups and how they work* Dieter Wellerdiek, 2008, http://www-03.ibm.com/systems/z/os/zos/features/wlm/WLM_Further_Info.html

[9] *ABC of z/OS System Programming*, IBM Redbooks, Volume 11, SG24-6327-xx

[10] *OS/390 MVS Parallel Sysplex Capacity Planning*, IBM Redbook, SG24-4680-01, January 1998

[11] *z/OS MVS Planning: Workload Management*, z/OS Literatur, SA22-7602-xx

[12] *System's Programmer Guide to: Workload Management*, IBM Redbook, SG24-6472-xx

[13] *z/OS MVS Programming: Workload Management Services*, z/OS Literatur, SA22-7619-xx

[14] *z/OS Resource Measurement Facility: Performance Management Guide*, z/OS Literatur, SC28-1951-xx

[15] *z/OS Basic Skills Center*, http://publib.boulder.ibm.com/infocenter/zos/basics/index.jsp

Index