Effective Use of MVS Workload Manager Controls

Ed Berkel and Peter Enrico

IBM Corporation
522 South Road

Poughkeepsie, NY 12601-5400

1.0 Introduction

MVS/ESA SP 5.1 simplifies the definition, control, and reporting of the performance requirements for MVS workloads. It provides a direct way to specify performance goals for work.

To reduce the complexity of managing system resources, MVS workload management provides goal-oriented dynamic resource management. Using the MVS Workload Manager, WLM, an installation defines performance goals for CICS, IMS, JES, APPC/MVS, TSO/E, Distributed DB2, and OpenEdition MVS work on the basis of business importance. One objective of workload management is to use these goals for dynamically adjusting access to processor and storage resources. This paper discusses general recommendations and guidelines for setting goals for the different types of work installations have running on their MVS/ESA systems.

2.0 Overview of WLM external controls

An installation can gather MVS work together into a new grouping called a **service class**. Users can think of a service class as being similar to the current construct of performance group. Like performance groups, service classes have periods. However, this is where the similarity stops. Performance group periods are assigned various resource-based controls for such resources as processor, storage, and MPL. These static controls, as specified in the installation's IEAIPSxx and IEAOPTxx members of PARMLIB, can be thought of as all the 'knobs' and 'dials' that allowed an installation to control how and to whom certain resources are allocated.

A primary goal of the MVS Workload Manager is to simplify the management of MVS systems by providing externals to reflect an installation's expectation for work being processed in the system. To allow this, the MVS Workload Manager enables the installation to explicitly state to MVS the service objective, or goal, towards which the work should be managed. These goals are assigned to the periods of each service class.

2.1 Goal types

The MVS Workload Manager supports four different goal types. As stated previously, goals are assigned to service class periods. Each goal type has a unique meaning and implication if used. The four goal types are as follows:

- Response time goal
  - Average response time goal
  - Percentile response time goal
- Discretionary
- Velocity
- System Goal

2.1.1 Response Time Goal

A response time goal is the simplest goal to understand. This goal type can be based on either the average response time of ended transactions, or a response time target for a given percentage of completions (eg, 80% of CICS banking transactions complete within 1 second).

The primary difference between an average response time goal and a percentile response time goal is that an average response time goal is HEAVILY influenced by 'outliers'. That is, a single transaction which has gone amiss can make an average terrible. For example, assume 99 transactions all end in 1 second, but one transaction runs for 2 minutes. The average is over 3.5 seconds, even though 99% completed in 1 second. When considering using an average response time goal, decide if it is ok for WLM to manage the work based on the worst behaving transactions.

There are 2 reasons for using an average response time goal, if only on an interim basis. First, service level objectives (or Service Level Agreements) may already state an average response time

objective. Much of this is rooted in the fact that most performance monitors today report average response time for ended transactions for a given grouping of work. Thus, average response time objectives can already be easily tracked and understood.

The other reason to specify an average response time goal is as a 'starting point' that will eventually lead to an appropriate percentile response time goal. Since outliers occur in almost all measurements, a percentile response time goal is probably always better to use rather than an average response time goal. The difficulty with this rule of thumb is that if an installation does not have a service objective that it is just passing along to the MVS Workload Manager, then the installation might not have enough information about the distribution of completions to be able to make an intelligent choice for the percentile value. In this case, the installation can start with a goal using the average response time currently being achieved by the workload. This average response time is currently reported by the installation's RMF performance monitor. Once the installation implements MVS Workload Manager goal mode, the RMF performance monitor will show distributions of response times around the average response time goal. The installation can then use this response time distribution to determine whether the workload has a normal distribution, or some anomalies, and therefore decide if a percentile value is more reasonable at 70% or 85% or even 95%.

2.1.2 Sometimes response time goals are not appropriate

Whenever there is an opportunity to use a response time goal, that should be the choice. However, an installation must consider that a response time goal is not appropriate for all types of work. There are 2 primary considerations that must be taken into account:

- The frequency of completions
- Variability of queue time

The work being assigned a response time goal must have sufficient completions. As a rule of thumb, expect at least 10 completions in 20 minutes for a response time goal to be effective.

There is a cross-over as time increases where one would not expect to see lots of completions. For example, most installations have a lot of short running transactions that end in a 20 minute period of time with each one having a completed response time of only 1 second. Most OLTP and interactive TSO transactions fit this description. These same installations have a smaller number of other transactions where each transaction takes over an hour to complete. Examples of these transactions would be CICS, IMS, or DB2 regions, started tasks, and long running batch jobs.

The first workload type with frequent short completions would be very suitable for a response time goal. The MVS Workload Manager would be able to base its decisions to 'give to' or 'take from' transactions in progress based on what was achieved by the transactions that completed recently. The second workload type would have a less statistically valid number of completions, making it difficult or even impossible for the MVS Workload Manager to make the same type of projections.

The second consideration is that the MVS Workload Manager includes queue time when managing towards a response time goal. That is, the response time for a completed batch job includes both the time the transaction was queued waiting for an initiator, and the time the transaction was actually executing. Thus, a response time goal is not appropriate when the work being assigned the response time goal has a variable and lengthy queue time. For example, if a batch job is submitted in the morning, held all day, and released at night, the MVS Workload Manager considers the transaction's completed response time to include the time held all day as well as the time the job spent executing at night. The installation should not assign a response time goal for this type of job or job class.

Instead, an installation probably wants the job or class held for a while, then executed at a certain speed once it becomes eligible to run. This concept will be discussed in the section 2.1.5 Velocity Goals.

2.1.3 Short response time goals versus long response time goals.

It should be noted that WLM attempts to achieve response time goals differently based on whether or not the response time goal is short (20 seconds or less) or long (greater than 20 seconds).

When a service class period has a short response time goal, WLM assumes these transactions will not be around very long. That means there will not be much time to sample these transactions to decide how to handle them on an individual basis. No individual storage access controls or policies are even contemplated. Instead, newly arriving transactions are controlled by period-wide central storage and expanded storage policies.

When a service class period has a goal other than a short response time goal, the Workload Manager assumes each transaction will be around for a while. Therefore it looks at all the address spaces in the period to see whether protective or restrictive storage targets are needed for them, and then will also decide for each address space how to handle its access to expanded storage.

2.1.4 Discretionary Goals

A discretionary goal means that an installation wants MVS to run the work when there are resources left over after running all the other work with non-discretionary goals.

It is very likely that every customer is already familiar with discretionary work. It is the work that usually runs in the lowest Mean time to wait group of the IPS in a domain whose MPL level fluctuates based on available capacity. What customers have been doing for years is telling SRM "put this collection of work together and run it as you see fit". This is exactly the collection of work an installation migrating to MVS Workload Manager goal mode would want to assign a discretionary goal.

SRM will continue to manage this discretionary work according to the mean time to wait algorithm. That is, work which is CPU intensive will be assigned a lower dispatching priority than I/O intensive work. This is still valuable for increasing throughput.

In addition, jobs with a discretionary goal are still candidates for individual storage control via Working Set Management.

2.1.5 Velocity Goals

Certainly there is MVS work which is not discretionary, yet it cannot be given a response time goal due to the infrequent number of completions. To address this there is a need for a goal that basically states "When this work is ready, be sure it runs without delays", or "When that work is ready, keep it plodding along to ensure it will eventually finish".

The third goal type, velocity, supports both of these needs, as well as gradations in between. Velocity is a measure of the acceptable processor and storage delays while work is capable of running.

It should be noted that the delays considered in the velocity calculation are only those delays that WLM has some control over. Specifically, I/O delays at a control unit or device are not part of the velocity calculated for work. Mount delays and operator delays are also not part of the WLM velocity.

2.1.6 When velocity goals are appropriate

A velocity goal is appropriate for a particular type of work when a response time goal and a discretionary goal are not. When SRM does not see any completed or in-flight transactions in a 20 minute interval, it has no "real time" completion data to use to project whether current transactions will meet the response time objectives. This decreases the effectiveness of a response time goal. What kind of decision would an installation want SRM to make about allocating processor and storage access? A velocity goal tells SRM the extent to which delays are acceptable.

Similarly, if there are only a few completions in 20 minutes, the response time data collected by SRM could become skewed by the 'outliers' described in the discussion of average response time above. When a group of work does not have at least 10 transactions completing within a 15-20 minute interval, installations are able to tell SRM how to allocate resources by using a velocity goal.

It should be noted that velocity does not correspond to the old dispatching priority control. It is not guaranteed that a service class period with a high velocity goal will necessarily have a higher dispatching priority than another period with a lower velocity goal. For that reason, installations should

not expect a significant difference in work execution by making minor adjustments to velocity goals, as might be expected by making small changes to dispatching priority in the IPS.

2.1.7 System Goals

The MVS Workload Manager can handle some work by default, without requiring a customer to bother setting externally specified goals. These 'system' goals simply provide static ways for MVS to treat certain recognized types of work. There are 3 predefined service classes that are managed according to these system goals. These service class names cannot be explicitly specified in any classification rules, but are instead service classes to be assigned in the absence of rules.

SYSTEM

When selected address spaces are created, they are assigned constantly the highest dispatching priority (255) and are excluded from storage isolation controls. These include MASTER, SMF, CONSOLE, CATALOG, GRS, RASP, XCFAS, SMXC, IOSAS, DUMPSRV, ANTMAIN, JESXCF, ALLOCAS, IXGLOGR and WLM. It is best not to assign a service class to these high dispatching priority address spaces, but to allow them to be managed within the SYSTEM service class.

SYSSTC

This service class is for all started tasks not otherwise associated with a service class. Effectively exploiting this service class is described in the section of this paper entitled 'Setting Goals for Started Tasks and System Spaces'.

Address spaces managed in SYSSTC service class are given a dispatching priority of 253. Remember that the MVS dispatcher allows multiple address spaces to share the same priority without the fear of the one which started first locking out all others. An advantage of putting selected address spaces in SYSSTC is that SRM will not have to spend time analyzing their state samples and comparing them to a goal. This is especially valuable since most installations probably do not have much of a goal for certain critical address spaces other than "be sure to run these when they are ready". SYSSTC is probably appropriate for JES, VTAM, etc.

A disadvantage of putting a started task into SYSSTC is that without a goal, storage isolation will not be invoked for the started task unless cross memory page faults in the space are impacting other work with goals.

SYSOTHER

This service class is intended as a 'catcher' for all address spaces other than started tasks that an installation has not bothered to classify. It is assigned a discretionary goal.

2.2 Importance

When multiple goals are defined, it is necessary to have a way to prioritize which of those goals are really critical, and which are only wishful thinking. MVS supports this through an **importance** value associated with a goal. Each goal can be rated as very important to the business (1), down to a goal that is desirable but can be sacrificed readily (5). The absolute value specified is less meaningful than the relative value of one importance compared to that of other goals.

2.2.1 Importances have several purposes:

1. They identify the critical goals to WLM. WLM attempts to satisfy all importance '1' goals before going after the goals at importance '2', then '3' etc.
2. They help prevent a user from getting into trouble. A user can set goals that are too aggressive. The importance allows WLM to make trade-offs that protect the really critical work.
3. They allow WLM to react to changing capacity. Reacting to an outage at many installations today involves the cancellation of some work, re-prioritization of other work, and reallocation of the remaining resources. WLM will use the importance of the goals to decide immediately which of the remaining work can donate resources needed by the work with higher importance goals.

If scarce resources are preventing work from achieving goals, WLM will not just select one type of work to pick on. It will try to achieve the goals of higher importance by degrading equally the work whose goals have lower importance.

It should be noted that importance does not correspond to the control of dispatch priority. It is not guaranteed that a service class period with high importance will necessarily have a higher dispatching priority than another period with a lower importance. Importance describes the significance of meeting a goal; it says nothing about how easy or difficult that goal may be to achieve. For example, it might be very important to an installation that a job which runs for many hours continue to execute occasionally throughout the day (Velocity may be only 5%, but importance 1). SRM may find that a low dispatching priority can satisfy that goal.

2.2.2 Resource consumption controls

In addition to the 4 goal types above, one further type of control is available with WLM. That is the ability to tell MVS to explicitly control the CPU access for a given collection of work. This can be stated as either a maximum or a minimum amount of CPU resource per second, which should be made available to all the work combined into a **Resource Group**.

The units of capacity are the same units that have been familiar for years as the SRM constant for various processors. Therefore it is very easy to tell MVS: "Don't let the service units consumed by this workload exceed the rate that could be captured using half of a model 9672-R11". A customer may actually be running that work across 3 LPAR images on 2 separate CECs, neither of which are 9672s!

3.0 The Heart of Workload Manager

This section gives an overview of the central decision-making part of the workload management function. It describes how MVS adjusts the CPU and storage access of work based on customer goals. The MVS component responsible for gathering the data and making these decisions is the System Resource Manager (SRM).

3.1 Address space sampling

Performance monitors frequently use state sampling as a way of determining where work spends its time. With MVS/ESA V5, SRM will periodically sample the state of every dispatchable unit. These samples are accumulated for each address space, and then further accumulated into the service class period associated with the address space. The states that SRM cares about are those states which reflect usage of, or delay for, a resource that SRM can allocate. Those resources are the processor (CPU) and storage. A list of the states SRM samples include:

- Using the CPU
- Waiting for access to the CPU
- Waiting for a page fault (separate states are sampled depending on the type of page fault).
- Waiting for a swap-in to be started
- Waiting for a swap-in to complete

The collection of state samples are used in several ways. First the velocity achieved by an address space or service class period comes directly from these samples. The velocity goal briefly mentioned earlier is simply a percentage of:

The number of samples where work is using the CPU
divided by
The number of samples where work wished it could use the CPU

Besides allowing a direct comparison to a velocity goal, SRM uses the sampling to determine where work is spending its time. This information allows SRM to determine what resource is the primary bottleneck for the work and allows SRM to assess the impact of some possible action it might take.

3.2 Maintaining enough history

It would be foolish to make drastic tuning decisions based on a single sample. So SRM keeps enough recent history to have a clear picture of delays. When a service class period has a velocity goal, the amount of history that SRM needs to keep is determined by the number of address spaces active in that period. If only a single address space is in the period, SRM will keep several minutes worth of samples. With more address spaces, the history could easily reflect the recent 20 seconds or so. In addition to gathering data via sampling, SRM maintains information on the response time achieved by work completing in each service class period. Again, SRM needs to avoid making a drastic tuning decision based on just a few completions when managing to a response time goal. Therefore, SRM maintains some history on the response times for recently completed transactions. That history may extend as far back as 20 minutes if necessary. But when there are a significant number of completed transactions, SRM can use much more recent data for making its tuning decisions.

3.3 Performance Index

Thus far, this paper has described that SRM has address space samples and response time data, and it can look at that information over a variable time period depending on the number of address spaces or the frequency of completions. In addition, SRM projects the expected response time of work currently 'in-flight'. Comparing the actual or projected accomplishments for some piece of work to its goal is very straightforward. Frequently, SRM looks at every service class period to compare the actual to the goal.

Since there are several types of goals, SRM needs some way to compare how well or how poorly one service class period is doing compared to other work. That comparison is possible through the use of a **performance Index**.

The performance index is a calculated value reflecting how well the work in each service class is meeting its defined goal over a time interval. A performance index of 1.0 indicates the service class period is exactly meeting its goal. A performance index greater than 1 indicates the service class period is missing its goal, whereas a PI less than 1.0 indicates the service class period is beating its goal. The PI thus makes it possible to compare a period having a velocity goal with a period having an average response time goal with a period having a percentile response time goal, and figure out which is farthest away from the goal (has the largest PI).

3.4 Plots

In addition to just knowing whether a service class is meeting its goal or not, SRM must assemble some 'insight' before deciding to change resource allocations. This is accomplished with exactly the same tool used by every mathematician or analyst studying trends --- a plot showing relationships between an independent variable, usually on 'x' axis, and a dependent variable, usually 'y' axis. SRM assembles many plots. One example for swappable work in a service class period is: given a Multiprogramming level (x), what is the maximum number of users ready to run (y). SRM uses this plot to predict the impact on number of ready users when assessing a change to an MPL target. Another plot for swappable work in a service class period is: given a percentage (x) of ready users who are actually swapped in, how many milliseconds (y) of swap-in delay occur for the average transaction. This plot shows how response time may be improved by increasing MPL slots or the expected impact to response time by reducing MPL slots.

3.5 Choosing and helping a receiver

On a timed basis, SRM looks for the service class period of highest importance that is missing its goal. That period is called the receiver. Using the samples of delays, SRM figures out what resource might be able to help that receiver. When the resource is identified, SRM looks for other work that can donate the resource. By analyzing the sampled state information and plots and historical data for both the donor and receiver, SRM determines whether the donation is a good idea or not.

The above paragraph is a very simplistic statement of the approach SRM takes. SRM does not wait for a service class period to miss its goal before taking action. It can select a receiver even when the PI is less than 1. And when looking for a resource that could help the receiver, SRM attacks delays according to the order of most expected benefit.

3.5.1 Handling servers

Suppose the receiver that is identified is a service class of CICS banking transactions (BANKING). Before SRM can know how to help that service class, SRM has to identify all the address spaces which are involved in handling those banking transactions. CICS/ESA 4.1 will identify its regions by using new programming interfaces introduced in MVS/ESA SP V5.

All TOR(s) and AOR(s) and FOR(s) at the CICS/ESA V4.1 level are identified. SRM creates internal service classes on each MVS image to define the topology of the regions which are all responsible for those BANKING transactions. Then the address space samples collected for those regions are used to determine which delays are being experienced by the regions, and to assess how SRM can help them.

3.5.2 Managing towards resource group constraints

The above sections (describing how SRM calculates a PI and chooses a receiver based on goal importance and gathered data) intentionally ignored resource groups. However, resource group minimums are honored first.

Dictating the processor cycles available to work could be in direct conflict with the setting of goals for that same or other work. If some work is running in a resource group with a minimum constraint specified, and that work is demanding CPU capacity, the SRM attempts to provide that work with the specified minimum capacity even at the expense of the goals of other work.

Before choosing a receiver as described in the earlier section, SRM will help any resource groups that are not meeting their specified minimum service units per second, as long as some work in that resource group is either missing its own goal, or has a discretionary goal.

Likewise, if a given resource group has exceeded its maximum capacity allowed, the MVS Workload Manager will 'cap' that work at the expense of the work meeting its goals. This could occur even if the work was just selected as the receiver most in need of help.

4.0 General Recommendations for Setting Goals

4.0.1 What to use as a basis for goals?

In preparing an initial set of goals, the performance analyst can exploit several different sources. The following three inputs will be discussed:

1. Current IPS/ICS setup
2. Service level objectives based on the business needs.
3. Historical Data

4.0.2 Current IPS/ICS Setup

An installation could use its current IPS/ICS configuration as a basis for its MVS Workload Manager service class configuration and as a basis for its Workload Manager classification rules. For example, most installations already have a multi-period performance group set up for the production TSO work. Each period is defined with a particular duration. These installations should find it very easy to set up a service class to mirror this type of TSO performance group. Other ways the IPS/ICS may help are:

Service definition coefficients

A service unit is a measure of resources consumed by an address space. Each installation can tailor, or weight, the different components of service by specifying weighting factors. Those weighting factors are the service definition coefficients (CPU, IOC, MSO, and SRB). These coefficients are specified in the IPS, and are needed in a WLM service definition as well.

An installation could just continue using the same coefficients. That might be wise if there are any billing tools used in the installation based on service units, and if those tools cannot be changed. Note that the tools may need to be changed for goal mode anyway, since presumably they are sensitive to Performance Group Number. If so, it is recommended that corresponding report classes rather than service classes be defined for these tools to be utilized in goal mode. Define service classes primarily for SRM management purposes, and report classes primarily for reporting purposes.

If there are no restrictions on the coefficients, this could be a good time to change them. The MSO coefficient has generally been carried at the default value of 3.00 This was appropriate when storage was a very rare resource. But for most installations today, the storage resource should not contribute so extensively to the total service. For several years it has been recommended to change MSO to either 0.000 or to 0.0001 It is recommended that this be done during a system's migration to goal mode.

Another change to consider now is to reset the CPU and SRB coefficients to 1.0 That way the definition of service units per second for resource groups will be directly in the terms of the CPU and SRB service units (since multiplying by a coefficient of 1 obviously would not change the resulting product).

Many installations use CPU=10.0, IOC=5.0, SRB=10.0 today. Each type of service would contribute the same proportion or weight if these three coefficients were changed to CPU=1.0, IOC=0.5, SRB=1.0 -- of course assuming the MSO coefficient is set to 0.

Durations
A service class goal is valid for a specified duration, just as performance group periods can specify durations. Since the duration is in terms of service units consumed, obviously if the coefficients of service units are changed, there must be a similar resulting change in the durations specified for multi-period service classes. Durations are probably only used in PGNs for TSO and batch work. Those will be the most likely types of service classes to have durations as well. Realize that a change to the meaning of a service unit can significantly alter the meaning of "TSO trivial" unless the corresponding duration is adjusted as well.

Insight from RMF reports
Understanding what is running in each domain and each performance group will allow the installation's RMF reports to help while preparing the WLM policy. The velocity experienced by a domain and a PGN is reported in RMF Version 5. That will be very valuable in selecting the velocity for a group of started tasks or long running batch work.

4.0.3 Service Level Objectives
Using the current IPS/ICS setup, as previously discussed, is very limited however. For instance, studying the IPS/ICS may not give any insight into CICS response times, and certainly will not provide the velocity goals for started tasks, nor IMS response times.

The installation's existing service objectives provide a good starting point for the goals to be defined to WLM. Possibly some batch classes are already documented as requiring a given turnaround time. Probably interactive work is already documented a requiring a specified response time.

There are three items to remember when looking at service level objectives as a source of information for the WLM policy:

1. WLM's response time goals describe system response time, without including network delays at the start or the end of the transaction.
2. The objectives document is probably very concise or simple, specifying just a few goals. Contrast that with the extensive number of PGNs contained in the installation's current IPS member. The WLM policies should emulate the service level objective in having a few goals, but can mirror the IPS by using report classes where extra reporting data is needed. This shift of emphasis from a large number of PGNs to a small number of service class goals is one reason no tool can, in most cases, generate an effective sample WLM policy from the IPS.

Installations should expect to approach WLM with a different mind-set than they approach the IPS.

3. A service objective frequently defines the worst-case. That is, a customer's service agreement may state that turnaround for batch class A is 30 minutes while today the installation is delivering a 5 minute turnaround. It would probably be unwise to just pass the 30 minute service objective to WLM because end users may become irritated by such a drastic change from their expectations. Instead, use the service objectives as guidelines for setting goals, and place more emphasis on historical data.

4.0.4 Historical Data
RMF Workload Activity reports will provide a good source of information on the average response times currently being achieved for each TSO period. If an installation expects to have the same percentage of TSO transactions complete in each period in goal mode than as were completing in each period in compatibility mode then that historical data will be a good starting point for the response time goal for each of the TSO service class periods.

The RMF V5 compatibility mode Workload Activity report will also be a good source of the velocity achieved by PGNs and domains. That will provide the insight into the velocity goal to specify for service classes for started tasks and long-running batch work.

In goal mode, a batch transaction is a batch job. Most installations already have historical data on average batch elapsed time. Realize that if an installation uses PERFORM= on the JCL of individual job steps, and supports that via OPGN specifications in the ICS, then the RMF Workload Activity reports from compatibility mode should not be used to determine average response times for PGNs of batch work. That is because each step with a unique PERFORM value will constitute an individual transaction in compatibility mode, so the historical RMF data is no longer in terms of batch jobs.

Possibly an installation already has average response time data for interactive OLTP transactions via CICS Performance Monitor or IMSPARS. These could be the source of average response time goals for the WLM service definition. In addition, "MVS/ESA Planning: Workload Management" - IBM GC28-1493 documents a way to use RMF in compatibility mode to find that same data for IMS or CICS transactions.

4.0.5 Understand the definition of a transaction when setting a response time goal
This statement should be fairly obvious. One cannot specify what goal to achieve for a particular type of work if one does not know the definition for a transaction for that type of work.

It should be noted that many customers care about end user response time at a terminal. But there is nothing the base control program can do to affect network delays. Nor is there any way to synchronize the clocks for the intelligent terminals and controllers with the sysplex timer. Distributing CPU cycles and storage resources based on unsynchronized clocks is a very bad idea. Therefore, the goals specified in WLM's policies for interactive work are host response times.

TSO Transactions
A TSO transaction for WLM's policy is identical to a TSO transaction as reported via RMF for the past 20 years. The installation still has a choice via IEAOPTxx whether a CLIST should be counted as a single transaction or a collection of individual transactions.

JES Transactions
A batch job is one transaction. It starts when the job is submitted (when the JES reader processes the job), and completes when the initiator finishes executing the job. That means it does include the time queued by JES waiting for an initiator, but it does not include output processing.

CICS Transactions
A CICS transaction begins when the initial CICS region receives a message, generally from VTAM, and ends when that region returns the result to VTAM. If the transaction is routed to another CICS region (AOR, FOR, etc) the time spent processing by those other regions accumulates for the original transaction.

This transaction information is passed to WLM beginning with CICS/ESA V4.1.

IMS Transactions
An IMS transaction begins when the Control Region receives a message from VTAM, and ends when the control region passes the response back to the network.

A customer has the option to allow inserted programs (program to program switch) to be considered as new transactions, or to just be considered a continuation of an existing transaction.

IMS transaction information is passed to WLM beginning with IMS/ESA V5.1.

Distributed DB2 Transactions
A goal can be specified for a distributed DB2 transaction. This is a request arriving remotely across the network via Distributed Relational Database Architecture. Once again, the start time is the arrival time from the network and the transaction ends at the commit point when DB2 completes the distributed request.

This transaction information is passed to WLM beginning with DB2 V4.1

4.0.6 Do not try to force certain resource allocation conditions to occur.
Avoid repetitive experimentation with velocity goals in the hope of forcing a specific dispatching priority, or dispatching priority order. MVS Version 5 has changed the dispatcher significantly to provide equal access for work at the same dispatching priority. Previous concepts of what work can or cannot share the same priority may no longer be valid.

Also, the MVS Workload Manager makes resource allocation decisions dynamically throughout the day. It recognizes workload peaks and valleys on much smaller intervals of time than installations are used to controlling. Therefore, the MVS Workload Manager may determine that a given workload could manage with a lower dispatching priority during one time period of the day than the rest of the day and still meet the goal of the work.

Using RMF Monitor II, one will be able to see the dispatching priority and storage isolation targets possibly change during the day. Remember that the absolute value of the dispatching priority at any given time is not relevant. Only the relationship of one priority to others is relevant. SRM will dynamically adjust that relationship by continually assessing what resource is needed by work of highest importance that is missing goals, and see who can donate some of that resource. If the resource that is needed or is donated is CPU, the relative dispatching priorities will change.

4.0.7 Do not set unrealistic goals.
One could consider the job of the MVS Workload Manager as 'to work towards goals' rather than 'to meet all goals'. Just because one is able to specify a particular goal does not mean that the resources are available to meet that goal. The MVS Workload Manager is continually monitoring how well goals are being met. It will attempt to achieve higher importance goals before lower importance goals. If a goal is set too aggressively the MVS Workload Manager may repetitively try to help the work associated with the goal, only to determine that nothing more can be done for it (or that stealing the resources it needs from some other work is not a good trade, based on the other work's goal and importance). In such cases SRM will proceed to help other work, but cycles have been wasted trying to help work that will never meet its goal.

4.0.8 Keep it simple. Do not define 'too many' service classes.
One requirement IBM has heard repeatedly is for a simple way for accounts to define goals to MVS. This is what WLM is trying to do. It is unlikely an installation will have hundreds of different business goals. Therefore, if service classes are defined only as needed by different types of work, that will allow the definitions to MVS to be simple.

Besides the argument for simplicity, there is a good technical reason to keep the number of service classes down. SRM is deciding how to allocate system resources based on sampling. As more work is combined into a service class, there are more entities contributing samples. So there is no need to

go far back into time to obtain a statistically significant set of samples. This means SRM can make better decisions more quickly, and therefore be more responsive, rather than depending on sampled data from several minutes ago.

For example, many started tasks spend over 80% of their time in an intentional timer wait. So less than 20% of the samples from those address spaces would give any insight for SRM into how to manage the dispatching priority or storage access of the work.

It may take over 10 minutes to gather sufficient samples from a single address space that was active only 20% of the time. However, if 20 similar started tasks are combined in the same service class, SRM would have a sufficient number of samples within 30 seconds to be able to decide whether an adjustment in resources is deemed appropriate. This allows SRM to be much more responsive.

What should be a typical number of service classes? Most customers will probably use around 25 service classes. This allows 4 or 5 for started task service classes, a handful for different batch classes, and plenty for the various interactive work being run by IMS or CICS or TSO or APPC, etc.

4.0.9 If something truly has no business goal, then assign it a discretionary goal.
All work with a discretionary goal will compete for processor access behind all other work with a velocity or response time goal.

Therefore, a discretionary goal will work best if a significant percentage of the processor cycles are available to work with discretionary goals. This is due to the behavior of the mean time to wait processor algorithm which will keep I/O intensive work at a higher priority than CPU intensive work.

4.0.10 Avoid having any work classified to internal service class SYSOTHER.
SYSOTHER is one of the 3 pre-defined service classes. It is assigned to certain work which a customer does not even care enough about to classify. For example, if a WLM policy does not have any rules to assign a service class to APPC transactions, when WLM sees one it will be associated with SYSOTHER and therefore have a discretionary goal.

There is no real problem with having work fall into the SYSOTHER service class. It is just an indication that the work was probably not discussed and prioritized along with the other work that was classified.

SYSOTHER will be assigned for unclassified work from the following subsystem types: APPC, Distributed DB2, JES, OPEN/MVS and TSO/E. Started tasks are not associated with SYSOTHER. See the section entitled 'Assigning Service Classes to Started Tasks and System Spaces.

If classification rules do not exist for CICS or IMS transactions (called CICS and IMS subsystem types in the WLM ISPF application), the transactions will not be managed towards the SYSOTHER discretionary goal. Instead, the serving address spaces will continue to be managed according to the goal specified for them as batch jobs or started tasks. See the discussion of servers later in this paper.[1]

4.0.11 Compile service definition on paper first
Defining goals to MVS Workload Manager is very easy using the ISPF application. But it is highly recommended that an installation have a clear definition of what they want to tell WLM, before starting the ISPF application. There are 3 good reasons for this recommendation:

---

[1] Note there is a very brief period of time during a mode switch or during a policy activation where SRM and WLM control blocks are still being created. If SMF writes type 30 records during this time when the control blocks are not present, it will record the address space as being associated with service class SYSOTHER. So even if an installation fully classifies all work, that installation might see an occasional job associated with SYSOTHER in type 30 records.

1. The importance to assign to the goals of different service classes will probably involve discussion with different organizations or functions within the installation. Those discussions and the resulting agreement on the ranking of work are essential to a good service definition. See reference 1 for an example worksheet that might help installations in these discussions. Without a visual comparison of one goal to the next, the discussions probably could not take place.
2. Naming conventions should be established before starting to define any information to WLM. With a little advance thought, the installation will probably choose more meaningful names by following some conventions, than would be chosen on the spur of the moment in the ISPF application.
3. WLM's ISPF application tries to catch mistakes at the time they are made. For example, if service class BATCH_A is supposed to be associated with workload TEST and with resource group DEPT87, the ISPF application requires the workload and resource group to be defined before they can be referenced. The relationships can be seen very easily on paper. Then entering them into the ISPF application is a trivial data-entry task.

4.0.12 Do not mix transactions and address spaces in a service cl ass

WLM allows goals to be specified for CICS or IMS transactions, as well as for work running in TSO or APPC address spaces, batch jobs, and started tasks. Since the CICS and IMS regions are run as batch jobs or started tasks, they will of course be associated with a service class. Do not assign the same service class to these regions as is assigned to the transactions that the regions serve.

Similarly, with DB2 V4.1, it is possible to assign goals to distributed DB2 transactions. Isolate that work into its own service class, rather than mixing it with any address space work.

The primary reason for this recommendation is that the sampling data, plots, and projections SRM assembles for each service class period can become very distorted if work of such diverse structure is combined in a service class.

5.0 Setting Goals for Interactive TSO

Deciding what goals to set for interactive TSO work will probably be the easiest of all the different work types. This is because most customers already have some sort of service level objectives set for this type of work as well as lots of historical data outlining what this type of work is already achieving in compatibility mode.

5.0.1 Interactive work should have a response time goal.

This is pretty obvious since 'interactive' implies there are real live users at the other end of the terminals waiting for a quick response when they hit enter. These transactions are short and quick, and enough are completing to allow SRM to collect a reasonable statistical sample set to base decisions on.

If a velocity goal is assigned to interactive TSO, it causes SRM to control the swap protect time on an individual address space basis rather than on a period-wide basis. That is less efficient. In addition, in cases when the TSO transactions do take a while to complete, SRM will look at the address spaces with a velocity goal to decide what expanded storage access to give to their demand paging, VIO paging, and hiperspace paging. This might involve monitoring address spaces for working set management control even though the work might still end quickly. Once again, this is less efficient than running interactive work with a response time goal.

5.0.2 It is probably best not to give interactive work a discretionary goal.

A discretionary goal implies the installation does not mind if SRM chooses to not run the work for a while. In such cases SRM may swap the discretionary address space out for long periods of time if needed. When a user is sitting at a terminal, this may be exactly what the installation intends if that user is exceeding the amount of service the installation feels is reasonable for a TSO user. This might be the case for the final 1% of all the TSO transactions. So it depends what percent of the TSO work makes it to the last service class period, and once there how the installation wants those TSO users treated.

5.0.3 A HOTTSO class may be helpful

Installations may choose to create a special TSO service class with a more aggressive goal and with an importance level higher than that of other TSO users. Any installation with a special system programmers' TSO performance group today could continue this approach with a special TSO service class.

6.0 Setting Goals for Batch

Deciding what goals to set for batch work will be very dependant on what type of batch work is being processed. Batch work has good uses for each of the three goal types: response time; velocity; and discretionary.

6.0.1 A Response Time Goal can be used for Batch

A response time goal is most suitable for short, homogeneous batch jobs. Today an installation may have one or more job classes to ensure fast turnaround time for this type of batch work. To meet the objectives for this batch work running in these job classes one would need to have enough initiators started to those classes, so that most jobs do not have long queue times waiting to be selected by an initiator. Using a response time goal, either average or percentile, to help ensure this fast turnaround time is very appropriate in this case whether the batch is test or production.

When setting a response time goal for batch work it is important to note that a batch job is one transaction. The transaction starts when the job is submitted (ie. when the JES reader processes the job), and completes when the initiator finishes executing the job. That means it does include the time queued by JES waiting for an initiator, but it does not include output processing. Therefore, the turnaround goal specified should include the sum of the queue time and execution time. Therefore a response time goal should be used for batch only when sufficient initiators exist for the associated job class such that most jobs do not experience lengthy queue time, and where there is a steady flow of completions. A batch response time goal would be less effective if (1) the goal is long, or (2) there is a low rate of job completions in the service class. An average of at least 10 batch jobs completions within a 20 minute period of time would constitute a steady flow of completions of jobs within a service class.

6.0.2 A Velocity Goal can be used for Batch

A velocity goal is appropriate for long-running production or test batch jobs, and for any IMS or CICS regions which an installation runs as batch jobs. The assumption for these recommendations is that there will be so few completions that a response time goal is inappropriate. Of course, the velocity goal specified for an IMS or CICS batch job only applies for the time when the region is not telling WLM about any transactions it is currently serving. It is therefore a good goal for CICS or IMS regions that are not yet upgraded to the release level supporting MVS Workload Manager.

A velocity goal could also be very appropriate for certain jobs or job classes which are held for a long time. When those jobs are released, the installation may require them to be processed quickly. Since the queue time might be unpredictable, a response time goal is inappropriate. But the velocity goal will tell SRM how to run the work once it has been released.

6.0.3 A Discretionary goal can be used for Batch

If an installation can not use a velocity goal for some batch work (or response time goal based on the caveats discussed earlier), that batch should be given a discretionary goal. Critical path production jobs should probably have a low velocity goal rather than discretionary, unless the site has plenty of capacity.

6.0.4 Multiple periods are still possible for batch jobs.

Some installations currently use more than one period in the performance groups for batch jobs, and some installations use just one period. There are advantages and disadvantages with each approach, with implications on the initiator structure. An installation currently using multiple period PGNs for batch can certainly continue that in goal mode.

Most likely in the IPS, the dispatching priority decreased in each period. In the WLM policy, the first period might contain a response time goal. Succeeding periods could have longer response time goals, possibly with decreased importance, with either a velocity or a discretionary goal for the last period.

6.0.5 A HOTBATCH class may be helpful.
Installations may choose to create a special service class with an aggressive goal for the case when some work "must run now". No classification rules need to refer to that service class, but operators could still use the RESET operator command to assign running work to that HOTBATCH service class.

Since the RESET causes SRM to start a new transaction for the address space, any JES queue time that may have existed is now ignored. This means a response time goal could be used for the HOTBATCH class. But if an installation has sufficient completions in the HOTBATCH service class, this probably warrants a discussion with operations! Therefore, a velocity goal is probably best for this service class.

6.0.6 A SWAPOUT class is not needed.
Many installations today create a special "swap out" PGN, that causes existing work to stop executing. There is no need to create a special service class like that, because the RESET operator command has been extended with a QUIESCE option. If a swappable address space is reset with that option, it will be swapped out. If a non-swappable address space is reset that way, it will remain in storage, but as the last item to be dispatched.

7.0 Guidelines for OLTP Workloads

A primary feature of the MVS Workload Manager is the ability to specify goals for on-line transactions, like CICS and IMS/TM user transactions. Prior to the MVS Workload Manager, customers influenced the response time for these transactions by controlling the resources provided to the transaction and resource managers. So if a performance manager wanted to achieve a better response time for a certain subset of the on-line transactions, the manager had to figure out which regions processed the transactions of interest. Then next task was to analyze the resource usage by those regions and work at improving this resource usage in hopes of achieving a better response time for the transactions of interest. This level of indirectness often made it difficult to tune on-line transaction processing workloads and often resulted in an inefficient use of system resources.

Examples of such address spaces are CICS regions, IMS control regions and message processing regions, and DB2 address spaces.

With the advent of the MVS Workload Manager, one can now specify response time goals for the on-line transactions. The MVS Workload Manager will manage the transaction and resource managers to work towards meeting the goals of the transactions they serve.

This feature is not without its caveats. The MVS Workload Manager is only able to manage these transaction and resource managers if they exploit specific MVS workload management services provided in MVS/ESA SP 5.1. As of the date of this paper, the following products are exploiting these services.

- CICS/ESA V4.1 and higher
- IMS/ESA TM V5.1 and higher
- IMS/ESA DB V5.1 and higher
- DB2 V4.1 and higher (for distributed DB2 transactions)

7.1 Setting Goals for Transaction and Resource Managers

7.1.1 Assign a velocity goal for OLTP regions.

Since the OLTP regions are long-running batch jobs or started tasks, they should be assigned a velocity goal. When setting a velocity goal for OLTP regions, it is best to set a goal that will enable the regions to run sufficiently so the transactions they serve will meet the service level objectives.

As stated above, the way the MVS Workload Manager manages transaction and resource managers is dependant on whether their releases exploit the new workload manager services provided in SP 5.1. Even when regions from exploiting products start up, they get classified as batch jobs or started tasks, and are assigned a service class. Once classified, like all other address spaces, these regions are then managed by the MVS Workload Manager to meet their specified velocity goal. During this time period they are regarded by the Workload Manager as 'non-servers'.

However, once a region from an exploiting product starts processing transactions, WLM will manage the region according to the goals of the transactions it serves. During this time period the region is regarded by the Workload Manager as a 'server'.

From this summary one can see that the goal that is assigned to a region will only sometimes have an impact on the transaction response time achieved by its workload. Regions that are not exploiting the MVS workload management services must be given a sufficient goal to ensure that the transactions they are processing achieve their response time objectives. Regions that are exploiting the MVS workload management services must be given a sufficient goal to ensure timely initialization during start-up, and proper treatment when the region is not processing any transactions for an elongated period of time.

It is unlikely an installation will always know which regions are exploiting the WLM services. The installation probably does not want to change the goals for work based on whether or not these service are being exploited. Thus, it is best to assign a velocity goal that will enable the regions to run sufficiently so that the transactions they serve will meet the service level objectives.

7.2 Setting Goals for On-Line Transaction Workloads

7.2.1 No goal can be set for OLTP transactions whose servers do not support the MVS WLM services.

Before even attempting to set a goal for OLTP transactions an installation must first determine if the transaction and resource managers processing these transactions exploit the workload management services. These services allow SRM to be aware of transaction starts and completions, as well as the response time being achieved. All of those are necessary to compare to a goal, and to influence a decision to change resource allocation. Earlier levels of the subsystems can of course be run, but that is accomplished with a velocity goal for the address spaces, rather than a goal for the interactive work running in the regions.

7.2.2 Start with simple classification rules

Start with just a few service classes for the OLTP work. For example, define a service class for production CICS transactions and another for test CICS transactions. Since CICS/ESA V4.1 does not alter its own internal dispatching queue based on the goals, and SRM still controls the CICS regions at an address space level, it does not make sense to spend a lot of time classifying different CICS transactions to unique service classes with differing goals.

There is one implication of this simple approach that is not immediately obvious. RMF breaks down the response time of CICS or IMS transactions so the installation can see where those subsystems believe delays are occurring. As an installation gathers very dissimilar work together into a single service class, that response time breakdown data will not be meaningful.

Extending the simple rule stated above (separating test from production) is straightforward. There may be a specific set of CICS transaction names, or a specific collection of IMS transaction classes

that are especially critical to the installation. It may be appropriate to create a service class just for them to ensure they receive the proper WLM goal.

Here is an example of a few classification rules for an installation's CICS transactions.

```
Subsystem Type . : CICS
Description  . . . Classification Rules for all CICS trans.

          -------Qualifier-------------            -------Class--------
Action    Type       Name     Start                Service     Report
                                          DEFAULTS: CICSTEST    ________
 ____  1  SI         CIP*____ ___                  CICSPROD    ________
 ____  2    TN       AB*_____ ___                  BANKING     ________
 ____  2    TN       XYZ_____ ___                  BANKING     ________
```

In this example, the first classification rule to be checked for every CICS transaction is the VTAM Application id of the receiving region (the Subsystem Instance). Assuming the installation has a naming convention of CIPxxxx for production regions, and CITxxx for test regions, the test transactions will fail this rule and be assigned the default service class. But when a production transaction arrives, the real service class to be assigned will be determined after seeing if the transaction name either begins with the characters AB, or is transaction XYZ.

7.2.3 Use a response time goal for OLTP transactions.
A response time goal is the only kind that WLM supports for IMS or CICS transactions. The goal could be either an average or a percentile response time goal.

An average response time may be the only historical information available when first choosing a goal. But since there are many examples of very long running CICS transactions, (dynamic program link, conversational transactions, etc.) it would probably behoove most installations to implement percentile goals soon after activating goal mode.

Caution --- Some installations might use RPGNs today to see average response times for CICS. Be very careful. If any CICS transactions are routed, the current RMF data could very easily contain some double counting. As such, the average response time is reported as less than actually seen by the TOR. Installations who have NOT accounted for this today in the IEAICSxx member should not rely on current RMF data for setting an average CICS response time goal. "MVS/ESA Planning: Workload Management" - IBM GC28-1493 explains how to create a simple WLM policy that will allow the compatibility mode RMF data to provide average response times using the same definition of a transaction as goal mode will use.

7.2.4 Use report classes for OLTP regions, if needed
Installations may still want to track resources used by individual CICS or IMS regions. Type 30 SMF records will continue to be available for that. But if an installation created a special PGN for each region to use RMF's type 72 SMF records, it can continue with that approach. Avoid creating service classes for each of these regions. Instead, assign a report class per region. All the resource usage data that has been accumulated for each PGN will be provided by RMF for each service class and report class.

8.0 Assigning Service Classes to Started Tasks and System Spaces

Deciding how to handle started tasks is probably the most difficult part of preparing an installation's WLM service definition. There are several approaches, each with its own advantages and disadvantages. They are summarized as follows:

1. Collect started tasks into a small number of similar groups.
2. Classify started tasks to individual Service class.
3. Do not even bother classifying any started tasks.
4. A combination of the previous approaches.

The first option is the best one and it is strongly recommended that installations choose this option for controlling their started tasks while in MVS Workload Manager goal mode. The other options are discussed for completeness.

8.0.1 Option 1. Collect STC into a small number of similar groups.
It is probably very easy to list really important started tasks, and also to list medium-importance started tasks, and then accept that the rest are not worth listing separately.

Creating 3 or 4 such classification groups of all started tasks is a very legitimate approach. Probably the installation would want the bottom group to just have a discretionary goal. Then it could assign increasing velocity goals to the other groups, based on the installation's compatibility mode RMF reports. (To obtain the velocity from the RMF reports, the installation might change its IEAICSxx member to assign a unique report PGN using this same classification group structure).

Using the ISPF WLM application, the installation's STC classification rules can simply refer to a named group of started tasks (transaction name group), or it could list tasks individually if the installation intends to use report classes to obtain extra RMF data for any individual started task.

The highest group probably contains tasks like JES and VTAM. If an installation prefers to let SRM just handle this group of tasks "well", then it should consider letting these run in the SYSSTC service class. For example, suppose a transaction name group is called HI_STC, and it contains VTAM, JES, RMF, etc. It could also contain the SYSTEM address spaces listed earlier, or it is ok to not even bother including them. Next, suppose the installation created a group called MED_STC as well, containing the group of less-favored started tasks.

The following example of an ISPF WLM classification panel for STC shows how the high started tasks are allowed to run in the SYSSTC service class:

```
Subsystem Type . : STC
Description  . . . Classification Rules for Started Tasks

          -------Qualifier-------------             -------Class--------
Action    Type       Name     Start                 Service     Report
                                          DEFAULTS: ________    _________
 ____  1  TNG        HI_STC__ ___                   ________    _________
 ____  1  TNG        MED_STC_ ___                   VEL35___    _________
 ____  1  TN         *        ___                   DISCRETN    _________
```

The default service class is left blank. This is an indication that the installation wishes some started tasks to be handled with the SYSTEM or SYSSTC service classes described under the topic "System Goals". The first rule assigns no service class to transaction name group "HI_STC", so that group inherits the blank default. In other words, the started tasks in group HI_STC run in the SYSSTC service class, or if they are the special system address spaces, they will be run in the SYSTEM service class.

The second rule assigns the started tasks identified in the MED_STC transaction name group into a service class called VEL35. And the last rule says everything else is classified to a service class called DISCRETN which has a discretionary goal. SRM recognizes that this last rule should not apply to the standard system address spaces (like GRS, DUMPSRV, MASTER, WLM, etc) and will continue to run them in the appropriate SYSTEM service class.

Though not done in this example, the installation could choose to classify any of the system address spaces explicitly by name into a service class just like any other started task. This means the classified address space would be treated according to the goal specified in that service class. That would involve less favorable treatment than just leaving the space in the SYSTEM service class, so this is not recommended.

(As an aside, DUMPSRV is handled differently than the other SYSTEM spaces. Its access to expanded storage will be managed via a space available policy, rather than least recently used).

Again, an installation can define a classification rule for any of the system started tasks or address spaces by classifying the address space with an explicit classification rule. That is, the classification rule must specify more than just a complete wild-card or 8-character mask.

In summary, this first approach was to collect the installation's started tasks into 3 or 4 groups, letting the more critical tasks run with default treatment, and using different velocity or discretionary goals for the other groups.

This is an important point. For many years there has been tremendous energy spent trying to achieve the optimal ordering of a dispatching priority queue. That was very important when all systems were uni-processors. This effort is unnecessary now because of significant multiprocessing and changes in the dispatcher to ensure work at a given priority does not monopolize the CPU compared to other work at the same priority. Therefore more work can be clustered together and given a single dispatch priority without worrying about potential lockouts. This is a cornerstone of the success of the goal mode algorithms.

8.0.2 Option 2. Classify STC to individual Service class.
That brings us to another approach for handling started tasks. With this approach, each major started task is put into its own service class, similar to the way many accounts today have each STC in its own Performance Group. Thus an installation could have service classes with names like VTAM, JES, LLA, VLF, etc.

One drawback is that this is more complicated for the installation to control. It would not be uncommon for the system programmers to continually spend effort measuring and adjusting lots of different velocity goals, and trying to determine what impact, if any, the system will incur when changing a service class's goal from a velocity 43 to a velocity 42 goal. (ps -- What a waste of time!)

A second drawback to this approach is that each additional service class uses more storage to hold accumulated data. This is true for SRM, WLM, RMF, and SMF type 72 and type 99 records.

Another drawback to this approach is that SRM will be spending time trying to adjust these many service classes to achieve whatever velocity goal is assigned to them. SRM will help only one service class each time it assesses how well the sysplex is meeting the installation's specified goals. If many service classes exist for individual started tasks, SRM could spend several intervals trying to handle individual started tasks rather than addressing a problem facing the on-line or interactive work.

Finally, as mentioned earlier, with only 1 address space in a service class, SRM will depend on a longer time interval for obtaining the number of samples that can justify changing resource allocations. This means SRM will not be as responsive to fluctuating circumstances as it could be if more address spaces were combined in a single service class.

8.0.3 Option 3. Do not even bother classifying any started tasks.
Suppose an installation's WLM service definition does not have any classification rules for started tasks. SRM will recognize certain system address spaces and put them into the pre-defined SYSTEM service class. And RMF will provide its usual data for that service class just the same as for any service class defined.

All other started tasks will be put into the SYSSTC service class, and kept at a high dispatching priority. This is nice for JES and VTAM, etc. If any given started task is recognized as serving on-line work with a goal (CICS or IMS transactions), then that address space would be managed as necessary based on the OLTP goal.

This approach has the advantage of simplicity. But not all started tasks are well-behaved. So having them all in SYSSTC could allow a CPU-intensive, or unstudied, started task to use a large amount of processor cycles. If the processor is lightly loaded, or in a 6-way, 8-way, or 10-way MP, this might still be desirable because that one task will not affect the ability of the remaining processors to attack the important work with goals.

This option is not seriously recommended for a regular production environment, since many started tasks cannot be trusted. But it demonstrates that in certain environments, it is not necessary to bother with velocity goals & classification for started tasks. A TSO, Batch, DB2 production sysplex has been running in goal mode for several months, supporting hundreds of users, using this approach of not classifying started tasks.

Of course, the installation could take this simple approach and still classify a few "horrible" started tasks to a discretionary goal, leaving the rest to be handled by SRM's defaults.

Finally, if using this approach, an installation could still gather the traditional RMF data for any given started task by writing a classification rule that just assigns a report class, not a service class, to that started task.

8.0.4 Option 4. A combination of the previous approaches.

It is of course possible to combine the above approaches, so that many tasks are allowed to run in the default SYSTEM or SYSSTC service classes, many others are combined into a group with a given goal, yet a selected number of address spaces are explicitly classified to a specific service class.

9.0 Setting Goals for Distributed DB2 Work

Prior to MVS/ESA SP 5.2 and DB2 Version 4 Distributed Data Facility, DDF, all the DDF transactions ran within the DDF address space at the same dispatch priority. If the DDF address space was assigned a high dispatch priority to help benefit the OLTP DDF transactions, this high priority also benefited the low importance 'batch-oriented' type DDF transactions. In essence, these low importance transactions got 'a free ride'. If the DDF address space was assigned a low dispatch priority to limit the CPU consumption of the low importance 'batch-oriented' DDF transactions, the low priority also limited the CPU consumption of the OLTP DDF transactions.

DB2 V4 DDF takes advantage of a new function in MVS/ESA SP 5.2 called Enclaves. Enclaves enable MVS to manage and report on DDF transactions by providing the DDF transactions an anchor to which they could be managed. This could be thought of as being similar to the way an address space is an anchor for other types of transactions (ie. TSO, batch, etc.). These enclaves allow the DDF transactions to be managed separately from the DDF address space. This support also allows these transactions to be managed separately from one another.

Enclaves allow the WLM to manage individual DDF transactions by allowing DDF threads to be assigned to service classes. Thus, DDF transactions of differing characteristics can now be associated with different workload manager goals. Deciding what goals to assign to enclaves for DDF transactions will be very dependant on what type of DDF work is being processed.

9.0.1 All three goal types are appropriate for DDF transactions

As with most other types of work, all goal types are appropriate for DDF transactions.

9.0.2 Assign DDF transactions to multi-period service classes

One of the largest benefits of enclaves and being able to assign goals for DDF transactions is that DDF transactions can now transition through multi-period service classes. Any query has the potential to scan thousands of rows and merge multiple tables. This may mean that the same singular select type query run on two different occasions may require more or less CPU service depending upon what rows and tables the query went against. By assigning these DDF transactions to multi-period service classes an installation has more control how the query will be managed.

An installation my choose to lower the importance of a query the longer it runs, or set a response time objective for short DDF transactions that do not consume over a certain amount of service. This would also allow an installation to solve the problem described earlier regarding OLTP DDF transactions vs the low importance 'batch-oriented' DDF transactions. These two types of transactions can not only now be managed away from the DDF address space, but they can now also be managed separately from one another.

Determining the duration to set to DDF transaction service class periods can be tricky. Installations have been able to figure out over time how to set durations for TSO period to get '80% complete in period 1' via trial and error. A similar sort of discovery will have to be done for DDF transactions. These duration values will vary from installation to installation depending on if an installation's DDF queries are small or large, and whether they are a critical part of the installation's workload.

10.0 Setting Goals for APPC Work

MVS Workload Manager recognizes APPC work which has been scheduled by IBM's APPC scheduler (ASCH). Therefore, an installation classifies this work in the WLM application using rules under subsystem type ASCH.

If an installation uses a different APPC scheduler, consult that product's documentation for information on classifying its transactions.

10.0.1 Treat the ASCH address space as an system started task

Follow the guidelines in this paper to allow the ASCH address space to be assigned the SYSSTC service class. That is, it should be part of the transaction name group HI_STC.

This will ensure that the scheduler can quickly process requests for new APPC transaction programs.

10.0.2 Use a single period, velocity goal for APPC transactions

An APPC transaction begins when the ASCH address space schedules a transaction program. That program is then started by an APPC initiator address space. This concept is very similar to the JES subsystem scheduling a batch job. The APPC transaction ends when the program returns to the initiator. This is similar to the completion of a batch job. Just as there are different profiles for batch jobs, there are different types of APPC transaction programs.

If an installation is sure that all APPC transaction programs process a single network request and then complete, the service class for these transactions could contain multiple periods with response time goals. But many APPC transaction programs are not structured so simply. Many keep a permanent, or lengthy, network connection and repeatedly process individual conversations across that network.

Therefore the transaction program continually consumes more and more service. If the APPC service class contained multiple periods, the transaction would fall to the last period and remain there. Assuming the last period's goal were less desirable than first period's goal, users connected to that APPC transaction program would experience decreasing performance.

These long running transaction programs may stay connected to the network for an indeterminate time period. Therefore a response time goal is inappropriate and a velocity goal should be used.[2]

In summary, unless an installation is sure that an APPC transaction program is started and ends for each network interaction, it is best to assign APPC transactions to single period service classes with velocity goals.

11.0 Setting Goals for OpenEdition/MVS Work

OpenEdition/MVS (OMVS) is a new environment for many MVS customers. Controlling this work through MVS Workload Manager is not difficult, but it has implications across three subsystem types --- STC, OMVS, and TSO.

---

[2] An APPC transaction program could choose to tell MVS Workload Manager about it's individual transactions. Then an installation could establish response time goals, and even multiple period controls, for the transactions according to that product's conventions.

11.0.1 Treat the OMVS Kernel and OMVS processes as all other started tasks

The basic recommendation for setting goals for the OMVS Kernel and OMVS daemon processes is to follow the guidelines this paper has outlined for assigning service classes to started tasks. The OMVS Kernel should be treated as a high priority started task and should be classified to the SYSSTC system service class. The OMVS daemons should be treated the same as other medium started tasks. OMVS daemons are long-lived processes that run to perform continuous or periodic system-wide functions such as network control. A goal such as the one recommended for the MED_STC service class discussed in the started task section of this paper should be very adequate for these OMVS daemon processes.

11.0.2 Use multiple periods and response time goals for OMVS forked children

Some of the OpenEdition/MVS work runs within an APPC initiator. These transactions are called OMVS forked children. The footnote in the APPC section of this paper mentioned that an APPC transaction program can tell MVS Workload Manager about it's transactions. OMVS does this, so an installation can establish goals for these transactions. An installation classifies this work under subsystem type OMVS.

Many, but not all, OMVS transactions use few resources. This is similar to the normal experience with TSO transactions. Therefore, a structure of a service class with multiple periods is appropriate for OMVS work, to reflect a different goal for CPU intensive work compared to interactive work. The first period should have a response time goal. This is consistent with other places in this paper recommending that interactive work be given response time goals.

The duration to be associated with first period will need to be determined iteratively, such that a predetermined percentage of transactions are considered trivial and complete in first period.

The last period should contain a velocity goal to describe the installation's view of long running and/or CPU intensive OMVS forked children. A low velocity goal should be given to this period, with an importance to compare this goal with others in the system.

An installation could choose to insert another period with a response time goal between the two discussed above. This would only be valuable if the installation has a very significant amount of OMVS forked children and wants more distinction between trivial and complex transactions.

11.0.3 Revisit goals for first period TSO

A TSO user signals the intent to submit OpenEdition work via the OMVS command. The ensuing OpenEdition work may run in the TSO address space (in the case of built-in commands) or may run in an OMVS APPC initiator (in the case of forked children). All the built-in commands are processed the same as any other TSO work, according to the goals associated with TSO's service class.

Each command which results in a forked child originates as a TSO transaction, so that command is initially associated with a goal according to TSO user's service class. When the forked child begins to execute in the OMVS APPC initiator, a new transaction begins and is managed per the prior discussion, according to the OMVS service class goals. The original TSO transaction is suspended and remains, most likely, in first period and no longer accumulating service, but gathering a longer and longer response time.

Some of the OMVS transactions initiated via TSO can run for a long time, resulting in high average response times for TSO first period. Therefore, when the OpenEdition interactive environment is established through TSO, the installation should revisit the goals associated with first period TSO to ensure that a percentile response time goal is used rather than an average response time goal.

When OpenEdition work is initiated directly into OMVS, via rlogin rather than from a TSO command, there is of course no TSO transaction created initially.

12.0 How to use Workloads

For MVS Workload Manager, a **workload** is nothing more than a collection of service classes that an installation would like grouped together for an RMF report. Most installations today freely use the term 'workload' to refer to a collection of their work -- whether that is Production, Test, and End_User; or Banking, Shipping, and Inventory.

In the first example, Production might include some OLTP work, some TSO work, and some batch jobs. That is, very different types of work that collectively are the Production workload. Since the types of work are so different, they would require different goals. So this customer would create a service class for the OLTP work, a service class for the TSO work, and a service class for the batch jobs, and have all three service classes combined into the production workload. RMF will provide reports for each service class and summarize information for the workload.

In the second example, it is possible that the workloads are just different CICS transactions targeted to separate CICS regions. In this case, the customer should probably create unique service classes to define goals for the Banking, Shipping, and Inventory work. This will allow RMF to report on the units that are of interest to the account. But the installation will associate those service classes with some new "OLTP" workload.

In summary, the only purpose of a workload for WLM is to allow RMF to automatically summarize transaction data and resource usage data for related service classes. RMF will provide reports alphabetically by service class, within a workload.

13.0 How to use Report Classes

RMF will generate type 72 records for every service class and for each workload. A **report class** can be used to either aggregate data from multiple service classes, or to receive a report on some part of a service class.

There is no benefit in assigning a report class to mirror the exact contents of a service class, since RMF already provides data for each service class. Similarly, there is no benefit in aggregating work into a report class exactly identical with the aggregation into a workload. But suppose a customer has a Production workload of 3 batch service classes, a TSO service class, and an OLTP service class. The 3 batch service classes could all be aggregated into a BATCH report class.

Another use of a report class is to isolate some part of a service class.

For example, suppose a customer wants a unique type 72 record for each CICS region. Since with CICS/ESA V4.1, each region will be managed to meet the goals of the transactions it is running, it makes no sense to create individual service classes for each of these regions. Instead the customer should classify the regions to unique report classes, since the only objective is to obtain selective reporting.

14.0 Considerations for Using Resource Groups

A resource group is a named collection of work for which an installation wants to control CPU access. An installation can guarantee work will have access to a given amount of CPU cycles per second, or restrain the work from using more than a given amount. As stated earlier, constraining processor cycles available to work could be in direct conflict with the setting of service objectives for the same work. Since resource groups introduce these conflicts, it is expected that most customers will not use resource groups. However, there are several environments that are appropriate for their use.

14.0.1 Resource group maximums can override goals.

Assume a certain department or organization has purchased an amount of processing capacity. The installation will want to deliver that amount but no more to them, while still setting goals for the

transactions originating from that group. The work could all be associated with the same service class, or could be several service classes combined into the same resource group.

The following table summarizes the experience of associating a resource group with a collection of TSO users all in the same service class. The service class had 3 periods, as follows:

```
.
| Period 1:  0.4 seconds avg response time, importance 2  |
| Period 2:  1.0 second  avg response time, importance 3  |
| Period 3:  5   seconds avg response time, importance 4  |
|                                                         |
```

In three separate measurements, the resource group was constrained to a maximum of 2000, 1500, or 1000 service units per second.

Table 2. Resource Group maximums limit CPU consumption per second

TSO Period	CPU SU/sec	PI	CPU SU/sec	PI	CPU SU/sec	PI
1	883.2	0.3	728.0	0.3	505.4	0.4
2	307.6	0.6	245.4	0.7	159.1	1.0
3	586.4	0.4	517.7	12.2	298.2	33.0
Total	1777.2	1491.1	963.7			
Group Max	2000	1500	1000			

As the table shows, periods 1 and 2 continued to maintain their goals even with a very constrained resource group maximum, while third period suffered because its goal was the least important.

14.0.2 Resource group constraints can apply to OLTP regions.
For years installations have known that service units are accumulated into the RMF records or SMF type 30 record for OLTP regions, regardless of the type of transactions running in those regions. This is still true with MVS Workload Manager, even though the installation can now specify response time goals for the transactions. Since the CPU service units are accumulating to the regions, it is possible to assign a resource group minimum or maximum capacity to a collection of CICS or IMS regions.

This could be an effective way to constrain the CPU usage of those subsystems to a predefined level of capacity. Possibly that would be valuable for a test subsystem.

14.0.3 Resource group maximums cannot be used as a replacement for RTO.
Resource group maximums cannot be used as a replacement for the old Response Time Objective function in the IPS. Each addresses a different environment. RTO was used to induce some swap-in delay when a small amount of work had excess processor capacity available to it. A resource group maximum is used to induce processor delay when some work has greater demand than the maximum specified. Enough work must exist to use the maximum allowed before capping due to resource group maximums will be experienced.

14.0.4 Resource group maximum is different than RESET QUIESCE.
A resource group controls access to the CPU. If a group specifies a maximum of 1 and work in the group wants to use the CPU, SRM could make every task in the group non-dispatchable over 98% of the time. But this would not force an automatic swap-out. The RESET command with the QUIESCE option described earlier is a way to force a specific swappable address space to be swapped out, or to move a non-swappable address space to the bottom of the dispatching queue. That non-swappable space could continue to use any CPU resource not used by other work.

14.0.5 Resource group minimums can allocate equal access for discretionary work

Suppose an installation has three departments or organizations that individually could run enough discretionary work to monopolize the processor. The installation may want to ensure that each department has "its fair share", or at least some occasional access to the processor.

If all the work were run just as discretionary, it is conceivable that for some period of time, only the work from department "X" was running. The installation could have the three departments' work associated with three separate resource groups, each specifying a minimum service rate, and SRM would ensure that if possible each would have its minimum appetite satisfied.

14.0.6 Resource group maximums can artificially lower processor capacity

IBM's parallel sysplex has addressed a long-standing customer concern over the significant jump in CPU capacity following an upgrade. For those installations still wishing to reserve some of the upgrade's capacity, a resource group could be created for a CPU intensive piece of work that is guaranteed to soak up a given amount of service units per second.

SRM will then have an artificially limited amount of processor capacity remaining to use in trying to achieve the goals for other work.

15.0 Summary

This paper has described the MVS Workload Manager function introduced with MVS/ESA SP V5, and has described some of the changes within MVS to support simple statements of customer goals for work. In addition, the paper has made many recommendations for customers to consider before planning a migration to goal mode. What remains is the question most frequently received at all presentations about WLM:

15.0.1 Can WLM do as good a job as a system programmer?

This depends heavily on how much time an installation is able to invest in tuning its SRM parameters and how fast the installation can react in the event of a performance problem. With Workload Manager, the MVS operating system will react immediately when goals are being missed, and work aggressively on the most important ones.

15.1 References

1. "MVS/ESA Planning: Workload Management" - IBM GC28-1493
2. "Preemptible SRBs: Understanding, Exploiting, and Managing Them" - John Arwe, IBM, CMG '95 Proceedings

15.2 Acknowledgements

The authors wish to thank the following people for their constructive review and suggestions for this paper: John Arwe, Cathy Eilert, Steve Grabarits, Cheryl Watson.