MVS WORKLOAD MANAGER VELOCITY GOALS: WHAT YOU DON'T KNOW CAN HURT YOU

John Arwe

IBM Corporation
522 South Road
Poughkeepsie, NY 12601-5400

Abstract

One of the goal types introduced with the MVS Workload Manager (WLM) in MVS/ESA SP5.1.0 was execution velocity. During its brief lifetime this goal type has bemused and befuddled performance analysts and consultants alike more than any other concept introduced by WLM. This paper will examine the definition of execution velocity, the practical implications of its definition, factors which influence its applicability to managing different types of work, how to use it more safely, and the possible effects of future changes.

Trademarks and Notices

Introduction

Why are WLM execution velocity goals confusing? Why do they tend to violate Rankin's "principle of least surprise" in many people's eyes? The simple answer is velocity goals confuse because they do not measure reality; rather they are an imperfect way to map reality to an abstract number. Contrast this with other WLM goal types:

- Response time goals use elapsed time, an objectively measurable metric commonly used to understand on-line performance.
- System goals use a fixed set of controls, time-based metrics are not used to assess progress.
- Discretionary goals are a hybrid, using fixed dispatching priority controls but variable storage and MPL controls that are changed based on system throughput.

Execution velocity[1] is an abstract mathematical description with no objectively measurable metric.

Definition of Execution Velocity

WLM's execution velocity is defined as a quotient of various sampled states:

$$\frac{\textit{CPU Using}}{\textit{CPU Using} + \textit{Delays Managed by WLM}}$$

which can be restated as

$$\frac{\textit{CPU Using}}{\begin{array}{c}\textit{CPU Using} + \textit{CPU Delay} \\ + \textit{Paging Delays} + \textit{Swapping and MPL Delays}\end{array}}$$

This mathematical definition has a number of implications which need to be examined. The states are based on sampling of work by WLM at fixed intervals, so the most obvious weakness of velocity is that it relies on sampling rather than measured results. Sampling is a well-accepted compromise of precision versus overhead when precise measurement of the quantities involved would significantly affect the state of the system. Sampling is useful as long as the sampled states accurately reflect the actual behavior of the work. Certainly in MVS, while everyone would like to measure quantities like CPU queueing delay, the cost of doing so in the dispatcher would be prohibitive.

Not all state samples are included in the velocity calculation. Since I/O delays are not managed by WLM, neither I/O using nor I/O delay samples are considered in velocity. Likewise other potentially significant states such as enqueue contention, database lock contention, tape mounts, HSM recalls, and time waiting for initiation do not influence velocity and are not reported in the SMF type 72 records although some of them are reported in other SMF records. Idle state samples are separately reported in the SMF type 72 records in the `IDLE` category along with other major using/delay states. Non-idle state samples for delays not managed by WLM are collectively reported in the SMF type 72 records in the `OTHER` category. For many traditional workloads, delays such as I/O are the primary delay reasons and `OTHER` state samples can comprise the majority of the delays. It is not unusual to see 60% `OTHER` delay for second period TSO due primarily to I/O waits.

Factors Affecting Execution Velocity

All Samples are Not Created Equal

Most sampled states included in velocity calculations have fairly intuitive definitions, falling into the broad categories of CPU, storage, and swapping. An important distinction amongst the sample types is their cardinality. CPU using samples, CPU delay samples, and paging delay samples are all counted per dispatchable unit, i.e. per task or service request block (SRB). `OTHER`, `IDLE`, and swapping-related delays are counted per address space,[2] regardless of the number of ready dispatchable units in the address space.

The difference in cardinality amongst the various sample types suggests a number of effects on calcu-

[1] Execution velocity is abbreviated to velocity hereafter.

[2] "Address space" should be read to mean "address space or enclave" wherever CPU or paging issues are concerned.

lated velocity and on the overall behavior of the WLM policy adjustment process by which resource allocations are changed to meet the objectives of the active service policy while optimizing system throughput.

First, and frequently overlooked: a single address space, in a single pass by the state sampler, can accumulate **more than one** state sample. This is different from releases before MVS/ESA SP 5, where the RMF state sampler recorded one state sample per address space per sampling interval. It means among other things that, because it takes into account multitasking, the samples are more likely to accurately reflect the state of the system. An unfortunate side effect is that there is no way to correlate the number of samples and the transaction response time once multitasking occurs.

Second, WLM will tend to fix problems for swapped-in work before adding more work to the multiprogramming set for a given service class period regardless of the goal type used. Swapped-in work has the opportunity to record one CPU/paging sample per dispatchable unit, whereas swapped-out work can record only one state sample per address space. Hence CPU and paging samples will tend to dominate over swap and MPL delay samples when WLM policy adjustment uses the number of delay samples to select the resource which a service class period most needs. If a service class period consists primarily of non-swappable work, there are no swapping or MPL delay samples so CPU and storage are the only resources managed by WLM which can reduce delays for the service class period.

Third, the observed velocities of multitasking address spaces can be sensitive to the number of CPUs in use by MVS. At equal CPU utilizations, systems with different numbers of CPUs running heavy multitaskers like DB2 will have very different CPU queue lengths. Probability theory states that with a CPU utilization of 90%, at any given instant on a machine with one CPU there is a 90% chance that all CPUs are busy (and hence a 90% chance that newly ready work will be queued). At the same utilization on an otherwise identical machine with five CPUs, the chance that all CPUs are busy is $(.90)^5$ or 59% if the state of each CPU is assumed to be independent. The table below shows some observed values running the DB2 Large System Performance Evaluation (LSPR) workload on the same processor family with the DB2 address spaces[3] running in service class SYSSTC.

Table 1. n-way affect on DB2 velocity

Processor Model 9021-	711	952
Number of CPUs	1	5
DB2 achieved velocity	13.3	74.9
SYSSTC achieved velocity	17.4	74.7
CPU Busy %	89.75	88.60

Amount of Work Contributing Samples

Remembering again that velocities are calculated for service class periods, the number of sampled address spaces in a service class period with a velocity goal will influence the number of state samples collected per sampling interval, and they may also influence the values of the samples themselves. Periods with few work units will contribute correspondingly few state samples per sampling interval, and WLM will use more historical data to achieve a statistically significant sample. More importantly with respect to velocity goals in particular though, is that periods with a large number of work units compared to the number of CPUs will have lower achievable velocities due to CPU queueing. This is very similar to the DB2 scenario discussed earlier, with multiple address spaces replacing the multitasking in the previous discussion. A velocity goal of 50 simply will not be achievable with 100 active IMS Message Processing Regions (MPRs) competing for the CPU. Typical achieved velocities with this many active MPRs are between 0 and 10.

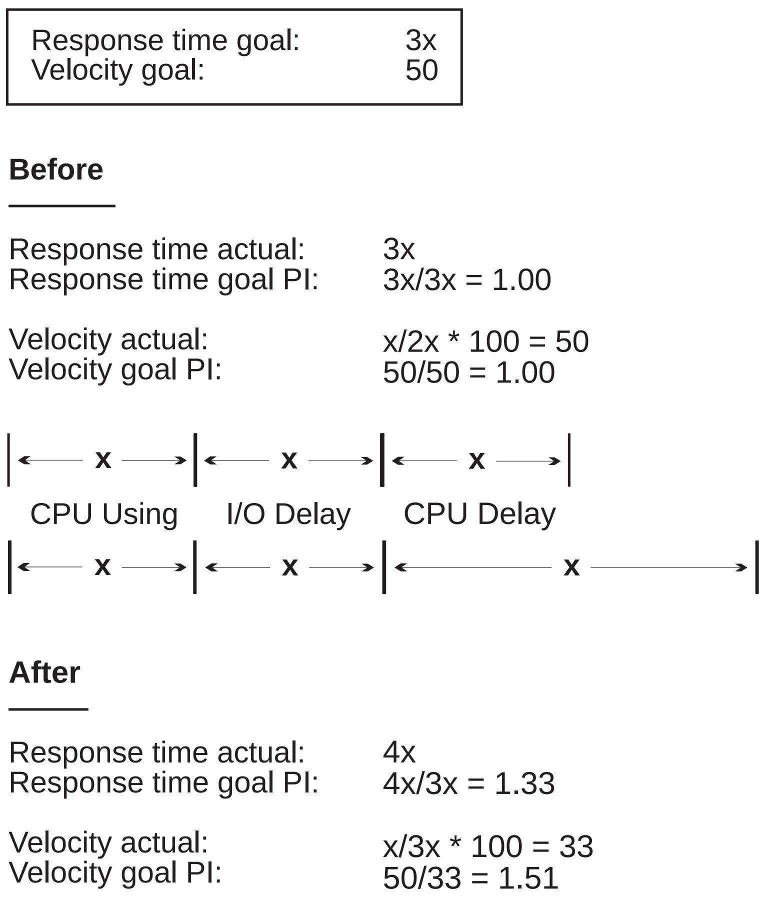
Unmanaged Delays are Ignored

The exclusion of IDLE samples from velocity is fairly obvious: if an address space or enclave is not doing anything, it is neither using nor delayed. WLM cannot make work become less idle since the duration of the idle time is dependent upon factors outside the system.

[3] In many places individual address spaces are used as examples because they provide a common frame of reference, but this does **not** imply that WLM calculates actual velocities for each address space. WLM calculates velocities for service class periods with velocity goals only.

Note that other WLM goal types do not intend to consider idle time either: transaction response times are intended to ignore idle time, and system goals do not consider elapsed time at all. Transactional work whose response time does include idle time, such as CICS conversational transactions and CATIA transactions, have proved difficult to manage for this very reason. Percentile response time goals are a partial remedy, in that they allow the response times which include large amounts of idle time to be ignored.

There is a pragmatic reason for the exclusion of OTHER samples from velocity: OTHER samples consist partially of IDLE states that WLM's state sampler does not or cannot detect. Contrast this with response time goals however, where the elapsed time that corresponds to OTHER samples *is included* in the response time. Thus OTHER delays do affect WLM's decision-making (specifically the actual vs. goal comparison) for response time goals but the same delays do not affect velocity goals. In resource-constrained environments, service class periods with large OTHER time that are managed using velocity goals will exhibit performance indices (PIs) that are more variable than other goal types since the work is effectively being managed on a subset of its total delays. The performance index is a comparison of the goal for a service class to its actual achieved response time or velocity. Any change in the subset of delays used to calculate velocity will have a disproportionate effect on periods with velocity goals. For example, imagine a period just meeting its response time goal and whose time is equally divided among CPU using, CPU delay, and I/O delay. Doubling the CPU delay would cause it to miss a response time goal by 33%, while it would miss a velocity goal by 50%, as shown in Figure 1 on page 5. Another consequence of this treatment is that as WLM is enhanced to manage additional delays and resources, the corresponding state samples will move from OTHER to the new delay states causing observed velocities to change and thereby changing resource allocation decisions. This migration aspect should not be overlooked when deciding amongst the various goal types. Like storage isolation controls in the past, on-going management of velocity goals is to be expected. Other goal types should be much less susceptible to this phenomenon.



Response time goal:	3x
Velocity goal:	50

Before

Response time actual:	3x
Response time goal PI:	3x/3x = 1.00
Velocity actual:	x/2x * 100 = 50
Velocity goal PI:	50/50 = 1.00

After

Response time actual:	4x
Response time goal PI:	4x/3x = 1.33
Velocity actual:	x/3x * 100 = 33
Velocity goal PI:	50/33 = 1.51

Figure 1. Effects of Delay Changes on Performance Indices

Why are there Always CPU Delay Samples?

People monitoring MVS performance have often asked why MVS/ESA SP 5 almost always shows CPU delay samples, even for their most important work. The answer is *reduced preemption* and it has been around for years. The change in MVS/ESA SP 5 CPU sampling from single state to multistate has made delays due to reduced preemption more visible than in the past.

Simply put, MVS usually avoids interrupting executing work to perform a work search in the instant after new work becomes ready to execute. Rather it relies on the normal behavior of work to release the CPU voluntarily, and backs this up with timed preemptions. In either case, a work search is eventually done and if the newly-ready work is of high enough dispatching priority it is run, otherwise it remains queued. This mechanism avoids a fruitless work search when the newly-ready work is of equal or lower dispatching priority than all currently executing work. When the reverse is true and the newly-ready work has a higher dispatching priority than currently executing work, deferring the interrupt often allows the executing work to progress to a voluntary wait. Allowing executing work to give up control voluntarily saves a work search and context switch that otherwise would have been required to redispatch the interrupted work after the higher priority work had run, making more effective use of the processor cache at the cost of delaying the higher priority work for a short time. For a more detailed treatment of reduced preemption and a comparison to its predecessor, see [Pierce].

When the WLM state sampler finds a ready dispatchable unit not running on any CPU, it records a CPU delay sample. If the delayed dispatchable unit has a dispatching priority higher than currently executing work, it is being delayed by reduced preemption. Thus the presence of a small number of CPU delay samples is part of normal MVS behavior;

the sampler does not distinguish between delays due to reduced preemption and those due to ordinary CPU queueing. Since some CPU delay samples are to be expected for all work, this means that achieved velocities will rarely reach 100%. When 100% is reached, it is more likely that the sampler did not observe work waiting than it is that the work actually did not enter a wait.

The effects described above will also vary with the mean time to wait of the work being sampled. If the dispatching priority is held constant, work with a low mean time to wait is more likely to be delayed by reduced preemption because it enters the CPU queue more often. Work with a high mean time to wait will monopolize the CPU within MVS's limits, and reduced preemption will not delay executing work.

Since it becomes difficult to achieve velocities consistenly above 90 due to reduced preemption, velocity goals in that range should be used sparingly to avoid forcing WLM to attempt to address delays which in reality cannot be removed. Such goals tend to be achievable however with complex DB2 queries, scientific batch, and very CPU-intensive work which generally exhibits a relatively long mean time to wait.

Inherently Variable Work

Observed velocities for some work will vary considerably more than for other work. Why is this? Most often it is due to the dispatching behavior of the work involved, involving either a low mean time to wait as discussed above and/or infrequent periods where the work is ready to execute. In the definition of velocity the idle state and unmanaged delays are ignored. Work such as `VTAM` or `IRLM` that is often idle compared with a production `CICS` region generates fewer state samples used in the velocity calculation per sampling interval, forcing older historical data to be used in building a statistically representative profile of the work. Let us use `VTAM` as an example to see why.

In goal mode WLM will sample the state of each address space and enclave every quarter second. Assuming a 15-minute RMF interval and that `VTAM` is in the `IDLE` or `OTHER` states 98% of the time, a total of 72 samples will be statistically available for the velocity calculation.

$$\frac{15\ \textit{Minutes}}{\textit{Interval}} \times \frac{60\ \textit{Seconds}}{\textit{Minute}} \times \frac{4\ \textit{Samples}}{\textit{Second}} = \frac{3600\ \textit{Samples}}{\textit{Interval}}$$

$$.02 \times \frac{3600\ \textit{Samples}}{\textit{Interval}} = \frac{72\ \textit{Samples Used in Velocity}}{\textit{Interval}}$$

Since this work has a low mean time to wait and does not consume much CPU per dispatch, the state sampler is much more likely to find the work waiting for the CPU than executing. Assuming 100 microseconds of CPU is consumed per dispatch and that the average CPU queue time is 1.9 milliseconds, 95% of the time when `VTAM` has ready work it would be seen as delayed for CPU by the state sampler.

$$.95 \times \frac{72\ \textit{Velocity Samples}}{\textit{Interval}} = \frac{68.4\ \textit{CPU Delay Samples}}{\textit{Interval}}$$

If 68.4 is taken as 68, the calculated velocity will be 5.555:

$$\frac{4\ \textit{CPU Using Samples}}{4\ \textit{CPU Using Samples} + 68\ \textit{CPU Delay Samples}} \times 100$$

which after truncation will be 5.0. If 68.4 is taken as 69, the calculated velocity after truncation will be 4.1666:

$$\frac{3\ \textit{CPU Using Samples}}{3\ \textit{CPU Using Samples} + 69\ \textit{CPU Delay Samples}} \times 100$$

which after truncation will be 4.0. Notice that a single state sample's difference skews the result 1.3%. Add in the difference between the velocity reported by RMF over an n-minute interval and the value computed by WLM each 10 seconds and considerable variation is observed. The graph below was built using data from SMF type 72 records and SMF type 99 records running a TPNS TSO workload. VTAM was classified to a service class with a velocity goal of 70, it was the only address space in the service class, and was at the highest importance running work. The corresponding achieved velocity from the SMF type 72 record was 9.0 which yields a performance index of

$$\frac{70.0\ \textit{Goal}}{9.0\ \textit{Actual}} = 7.77$$

after truncation. Note that the performance index in the SMF type 99 records is scaled by 100; this scaling factor has been removed in the graph but not in the tables which follow.

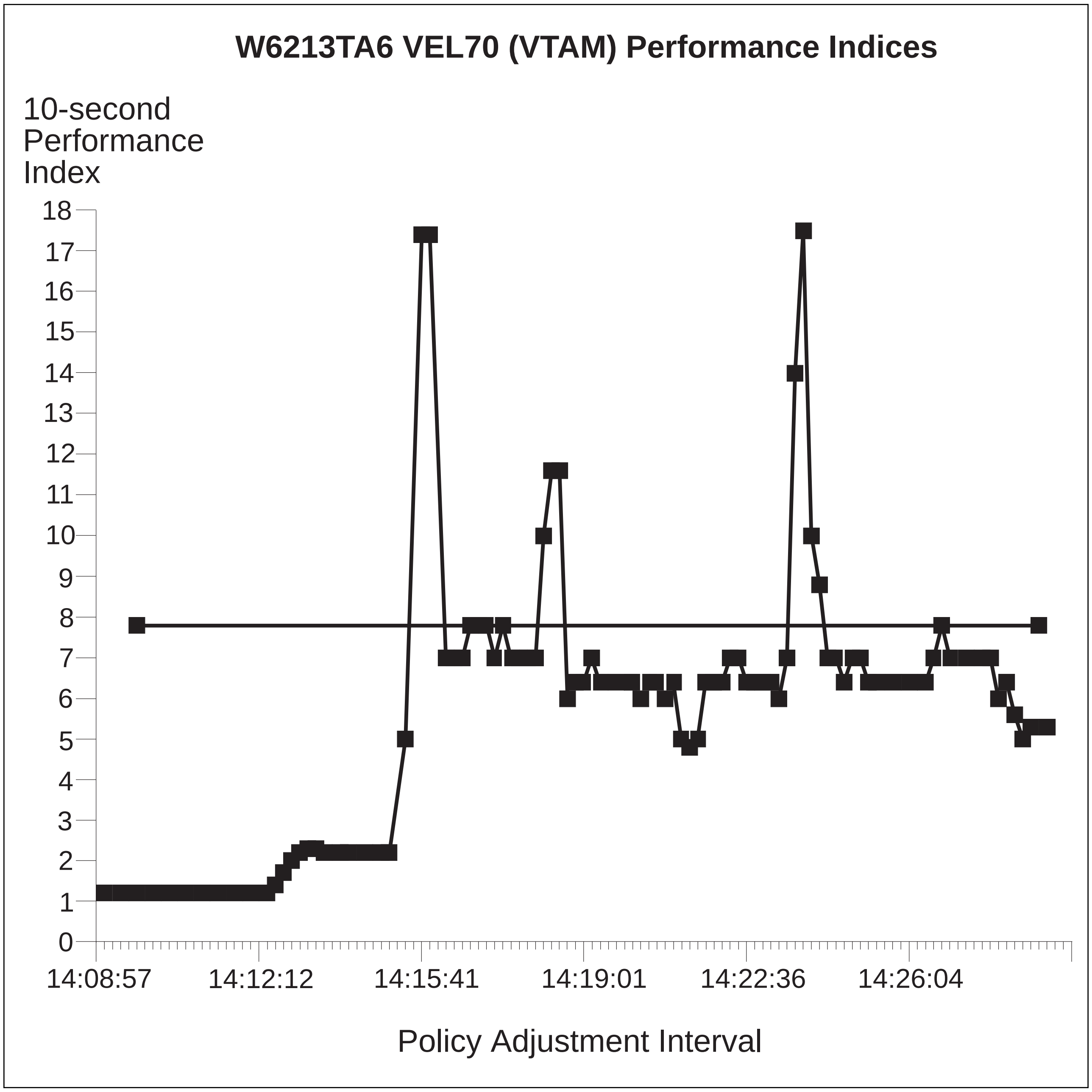


Figure 2. SMF99 Performance Index of VTAM Over Time. Constrast the variability of the actual 10-second performance indices calculated by WLM with the RMF average of 7.77 for the measurement.

```
      TS   PI      CPUUD    AUXDELAY    MPLSWAP    IDLE    OTHER

14:12:12  125     311          1          0         0        29
14:12:22  125     310          1          0         0        29
14:12:32  137     320         37          0         0        29
14:12:42  179     319        151          0         0        29
14:12:53  205     325        229          0         0        29
14:13:03  218     325        269          0         0        29
14:13:13  241     325        327          0         0        29
14:13:23  241     328        326          0         0        29
14:13:34  233     342        314          0         0        29
14:13:44  233     334        308          0         0        29
14:13:54  233     334        307          0         0        29
14:14:04  233     340        309          0         0        29
14:14:14  233     363        308          0         0        29
14:14:25  233     350        308          0         0        29
14:14:35  233     350        308          0         0        29
14:14:45  233     350        310          0         0        29
14:14:55  233     358        315          0         0        29
14:15:06  233     356        312          0         0        29
14:15:16  233     357        314          0         0        29
14:15:26  500     148        402          0         0         0
14:15:41 1750      49        519          0         0         0
14:15:47 1750      49        546          0         0         0
14:15:57  700     148        650          0         0         0
14:16:07  700     148        641          0         0         0
14:16:17  700     148        610          0         0         0
14:16:28  777     148        686          0         0         0
14:16:38  777     150        729          0         0         0
14:16:48  777     149        738          0         0         0
14:16:59  700     171        755          0         0         0
14:17:09  777     160        734          0         0         0
14:17:19  700     164        744          0         0         0
14:17:29  700     170        739          0         0         0
14:17:39  700     174        745          0         0         0
14:17:50  700     172        760          0         0         0
14:18:00  700      91        486          0         0         0
```

Figure 3. SMF99 Delay/Using Sample Data for VTAM. Do not be alarmed at the large number of auxiliary storage delay samples for VTAM. The measurement used was designed to recreate the conditions described by OW20486 which involved heavy paging to auxiliary storage. The PI column is the performance index recorded in the SMF type 99 records at the end of each 10-second policy adjustment interval and is shown in Figure 2. This data demonstrates that RMF interval averages are averages, and do not necessarily reflect the data SRM is acting upon each policy adjustment interval. The RMF interval performance index of 9.0 is quite different from the performance indices computed by SRM which range from 1.25 to 17.5.

```
      TS  CPUUD    AUXDELAY    MPLSWAP    IDLE   OTHER     CPUU10  CPUD10

 9:11:51   122         0          0       8136     1          0         0
 9:12:01   122         0          0       8175     1          0         0
 9:12:12   122         0          0       8217     1          0         0
 9:12:22   122         0          0       8258     1          0         0
 9:12:32   122         0          0       8299     1          0         0
 9:12:42   122         0          0       8340     1          0         0
 9:12:53   126         0          0       8379     1          1         1
 9:13:03   126         0          0       8419     1          1         0
 9:13:13   125         0          0       8461     1          0         0
 9:13:23   125         0          0       8502     1          0         0
 9:13:33   125         0          0       8543     1          0         0
 9:13:44   125         0          0       8584     1          0         0
 9:13:54   127         0          0       8623     1          1         0
 9:14:04   126         0          0       8665     1          0         0
 9:14:14   126         0          0       8706     1          0         0
 9:14:25   104         0          0       4862     0          0         0
 9:14:35   104         0          0       4903     0          0         0
 9:14:45   104         0          0       4944     0          0         0
 9:14:55   104         0          0       4985     0          0         0
 9:15:06   104         0          0       5026     0          0         0
 9:15:16   106         0          0       5065     0          1         0  ←
 9:15:26   107         0          0       5105     0          0         1
 9:15:36   106         0          0       5147     0          0         0
 9:15:47   106         0          0       5186     0          0         0
 9:15:57   110         0          0       5226     0          1         1
 9:16:07   110         0          0       5194     0          1         0
                                                   0          0         0
```

Figure 4. Typical State Sample Profile of VTAM. This data is from a less constrained measurement of a TSO workload, where paging to auxiliary storage is not occuring. This sample profile is more typical for VTAM on normal systems. The CPUU10 and CPUD10 are the CPU using and delay samples collected over the preceding policy adjustment interval. Note that historical data dominates the performance index when few samples used in the velocity calculation are collected per policy adjustment interval. For example, at 09:15:16 105 of the 106 samples used to calculate velocity are historical data. This is more likely to happen when few address spaces are grouped in a service class period.

```
--------------------- TS=14:08:57 -------------------------------

SRVCLASS    PERIOD       PI    WHATGOAL    GOALVAL    IMP    BDP

TSOCLASS       1         36    Short RT       300      2     247
TSOCLASS       2         15    Short RT      1500      2     247
TSOCLASS       3         82    Short RT      5000      3     247
TSOEVEN        1         40    Short RT       300      2     251
TSOEVEN        2        211    Short RT      2500      2     251
TSOEVEN        3         96    Short RT      5000      2     249
TSOODD         1         40    Short RT       300      2     251
TSOODD         2        180    Short RT      2500      2     251
TSOODD         3         98    Short RT      5000      2     251
VEL70          1        125    Velocity        70      2     249  ← VTAM ran here
VEL80          1        307    Velocity        80      1     253
VEL90          1         93    Velocity        90      2     253
$SRMBEST       1          0    System           0      0     255
$SRMGOOD       1          0    System           0      0     254
$SRMDI00       1         81    Disc             0      6     192

--------------------- TS=14:16:48 --------------------------------

SRVCLASS    PERIOD       PI    WHATGOAL    GOALVAL    IMP    BDP

TSOCLASS       1        413    Short RT       300      2     249
TSOCLASS       2        182    Short RT      1500      2     245
TSOCLASS       3       1200    Short RT      5000      3     237
TSOEVEN        1       2700    Short RT       300      2     247
TSOEVEN        2        141    Short RT      2500      2     243
TSOEVEN        3        206    Short RT      5000      2     239
TSOODD         1       1800    Short RT       300      2     245
TSOODD         2        134    Short RT      2500      2     241
TSOODD         3        163    Short RT      5000      2     241
VEL70          1        777    Velocity        70      2     253
VEL80          1        320    Velocity        80      1     253
VEL90          1         93    Velocity        90      2     253
V50TOOL        1        294    Velocity        50      2     253
$SRMBEST       1          0    System           0      0     255
$SRMGOOD       1          0    System           0      0     254
$SRMDI00       1         81    Disc             0      6     192
```

Figure 5. SMF99 Subtype 2 Record Goal-Related Data. SMF type 99 subtype 2 period records are only written for service class periods which demanded resources over the previous policy adjustment interval, thus this extract shows all goals relevant for the measurement. The BDP column is the base dispatching priority of the service class period. VTAM's dispatching priority started out to be lower than most system programmers would like: below 5 of 9 TSO periods, and competing with a 6th.

At 14:16:48 VTAM was moved above all of the TSO periods and it remained there for the duration of the measurement, but the presence of historical data from the preceding period of light TSO load helped to delay WLM's response to its woes for 8 minutes. Had VTAM been classified or defaulted to service class `SYSSTC` its dispatching priority would have been above that of TSO from the start and would have remained there.

Using Velocity to Statically Rank Work

For years system programmers have fought dispatching priority wars for server address spaces: VTAM above JES2, CICS TOR (terminal owning region) above CICS AORs (application owning regions), et cetera. Different people came to different conclusions about how to juggle these address spaces so that when one loops the others are not starved for CPU. What does this have to do with velocity goals? Those once bitten by having things in the wrong order tend to latch onto velocity goals as a way to continue ranking their server address spaces. **This does not work.** The relationship between two different velocity goals implies nothing deterministic about their relative dispatching priorities.

Some of the "A's dispatching priority must be above B's dispatching priority" rules originated from a loop in one while both were at the same dispatching priority. Others came from conflicts between workloads whose CPU demand conflicted at certain times, and still others from true client-server relationships where enough client work exists to degrade or starve the server if the dispatching priorities are out of order. The first case has seen some changes in MVS/ESA SP 5. Prior to MVS/ESA SP 5 a looping address space could prevent other address spaces *at the same dispatching priority* from being dispatched as well as those with lower dispatching priorities. The only solutions were to rank their dispatching priorities or use time slicing. In MVS/ESA SP 5 the dispatcher was changed to implement "fair share" access to the CPU within a dispatching priority which greatly reduces this effect, thus allowing the previously incompatible address spaces to share the same dispatching priority without fear of one looping address space totally locking out its priority-mates. The need for this type of static ranking has lessened, but the old rules remain in many minds.

When deciding on velocity goals for "loved ones", it is better to not even bother putting velocity goals on trusted started tasks which consume relatively little CPU and are idle much of the time. VTAM and IRLM are prime examples of such started tasks; put them in SYSSTC rather than trying to manage them to velocity goals. Doing so assures them of receiving the excellent CPU access that they require and eliminates the problem of setting velocity goals for them. Because they are idle most of the time the observed velocities for these address spaces tend to vary greatly even with constant controls. Having WLM attempt to manage important and highly variable work to a predictable velocity is not practical.

Velocity Extremes

Combining the definition of velocity with Murphy's Law, how far can WLM go? What is the illogical conclusion of a velocity goal? Since only the subset of samples where work is not in the IDLE or OTHER states affects velocity, the danger points are 0% and 100% ready. Work that is never ready is not consuming enough resources to worry about, so we are left with the 100% point to worry about. Jobs or transactions that become 100% ready are known by many names: soakers, loopers, and other more colorful variants. "Looper" probably conveys the meaning best: 100% CPU appetite. In terms of state samples, a looper is always in either the CPU using or CPU delay states. In what ratio are the two states? Its very definition says that the velocity *is* the desired ratio, in this case CPU using to CPU appetite (using plus delay). In other words, a velocity goal of 50 could be read to mean "try to run this work at least 50% of the time it wants to run." For production subsystems this may be exactly what is desired, but few system administrators would want last period TSO to have such generous access to the CPU. One can imagine the implications of this for a looping transaction. Especially for service classes containing ad hoc work, this difference between what may be achievable currently, perhaps using excess resources, and what WLM should manage to may be very real. If relatively unimportant work is over-achieving due to excess resource availability, there is no reason to use the achieved actual as a goal; a lower velocity or importance on such a goal allows WLM to degrade such work when the resources are needed by other work.

Using Velocity Safely

If velocity goals are so sensitive to various factors, why use them? Quite simply, response time goals and discretionary goals are not enough. Since the real world does have things like subsystem regions which do not use WLM's transaction management interfaces and even with WLM-exploiting subsystems some subsystem work is not done on behalf of a specific transaction, a way to manage non-transactional work is needed that does not rely on response times. Execution velocity was invented to fill this need and there are ways to use it without falling into traps.

Use SYSTEM and SYSSTC

The first thing to do is to figure out which work should just be put into service classes `SYSTEM` and `SYSSTC` and do so. `VTAM`, `JESx`, `IRLM`, and most system address spaces should fall into one of the service classes above. Trustworthy production server regions (CICS TORs, IMSCTL) are also candidates for `SYSSTC` as long as they do not present the threat of starving other work by monopolizing the CPU. No velocity goals need to be provided for work which is managed using these service classes.

Combine and Conquer

Due to the variability of velocities, micromanagement of many different service class periods with only slightly different velocity goals is not practical. Many sites find it useful to think in terms of high/medium/low velocities rather than concentrating on the numbers. A set of three such service classes with one period each should cover most work with velocity goals, perhaps using low=10, medium=30, high=70. This is nothing magic about these numbers, they merely provide guidelines for relative magnitude. If production regions consistently achieve a velocity of 80 or 90, either scale the high velocity goal to meet the observed value or add a one-period service class with the observed actual as the goal specifically for the production regions. The rule to remember is to take good care of loved ones, and consider undercutting the achieved velocities as measured in WLM compatibility mode for less important work to allow WLM some freedom during demand spikes in the production regions.

Hedge Your Bets

If untrustworthy work, e.g. potential loopers such as test regions or ad hoc batch, is given a velocity goal then use either goal importance or a resource group maximum to minimize potential damage to competing work. Lowering the importance of the velocity goal will prevent loopers from impacting higher importance work, although it could still cause CPU starvation for work with discretionary goals. Use a resource group maximum if lowering the importance is not possible or if discretionary work needs some protection.

Use Peak Demand to Measure Achieved Velocities

Do not base velocity goals on observed off-peak values: remember that the goal will be managed to *all of the time*, and peak times are the ones with most downside potential. The best source of data is *your* workload, running in WLM compatibility mode during a peak period. For the important work, use the achieved compatibility mode velocity from a peak period or the highest from a sample of peak periods. For less important work, use a value *below* the achieved compatibility mode velocity so WLM has room to work when tradeoffs are necessary.

Conclusion

There are many different factors which affect velocity goals, what is achievable, and what WLM can effectively manage to. These factors include:

1. The inherent inaccuracy of sampling
2. The cardinality of the different sample types
3. The treatment of `OTHER` state samples
4. The effects of reduced preemption
5. The dispatching behavior of often-idle server address spaces
6. Misconceptions about the relationship between velocity and dispatching priority
7. The implications of the definition of velocity when applied to looping work
8. Migration implications as new delay types are moved from `OTHER` to new managed delay samples

These factors do not preclude the use of velocity goals. Rather, they underscore the idea that velocity goals should be used with care when they are appropriate, and should be qualified by an importance or resource group maximum to reduce the impact on competing work.

References

1. [LSPR] "IBM Large System Performance Reference," SC28-1187
2. [Pierce] "Dispatching Management in MVS - TCBs to Enclaves," Pierce, Bernard R., CMG95

Acknowledgements

The author wishes to thank the following people for their constructive review and suggestions for this paper:

Ed Berkel
Steve Grabarits
Peter Enrico
Greg Dyck
Peter Yocom
Steve Kinder
Catherine Eilert
Wayne Morschhauser