Paper 291-28

A Case Study of the Tools, Techniques, and High Level Model Used to Tune AIX Version 5L for The SAS System

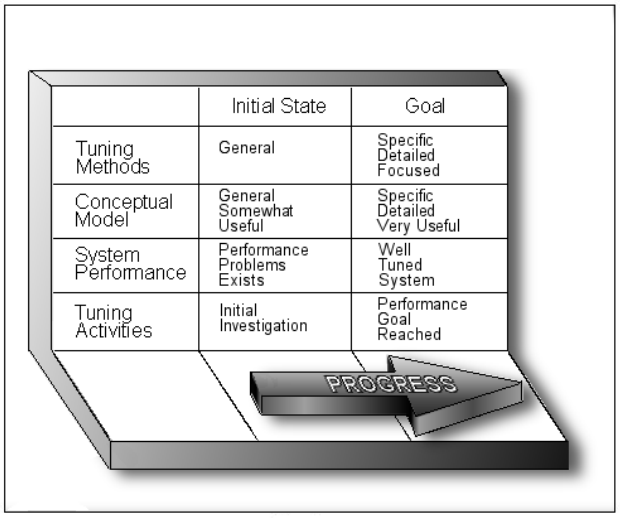
Frank Bartucca & Torre DeVito, The IBM International Competency Center at SAS Institute, Cary, NC

ABSTRACT

This paper presents a case study of the efforts undertaken at the SAS/IBM Corporate Technology Center to optimize the performance of the SAS System on AIX Version 5L. A high-level model is presented as the overall organizing framework. Performance measurement and analysis techniques are examined in the context of a data-driven approach. The utility and effectiveness of various performance tools is evaluated.

INTRODUCTION

The purpose of this paper is to provide performance tuning information to AIX system administrators responsible for optimizing the performance of SAS software on IBM pSeries Servers. The information presented is collected from the investigation and solution of several performance problems reported by SAS/IBM customers and by SAS or IBM employees engaged in performance testing of SAS software running on IBM pSeries Servers.



	Initial State	Goal
Tuning Methods	General	Specific Detailed Focused
Conceptual Model	General Somewhat Useful	Specific Detailed Very Useful
System Performance	Performance Problems Exists	Well Tuned System
Tuning Activities	Initial Investigation	Performance Goal Reached

- Figure 1 -

USE A CONCEPTUAL MODEL

It is difficult to grasp the complexity of a large computer system without a conceptual model of the system. Even if this model is not written down on paper, captured in a spreadsheet or otherwise represented, system administrators do have such a model in mind as they work on performance tuning. When performance problems are solved quickly, the system administrator usually has developed a refined model of system performance and specific methods that allow rapid convergence to a solution.

As shown in Figure 1, the process of resolving performance problems involves progressing toward the performance goal on the right. The collection and analysis of performance data drives progress forward. As more data is gathered, the system performance model is refined and becomes more useful in providing the values for tuning the system and resolving the performance problem. In addition to a tuned system, other products of this effort are more refined system performance models and methods. Continual refinement and expansion of the scope of the models and methods can occur if information is preserved and reviewed at the starting point of subsequent performance tuning efforts.

SYSTEM GESTALT: CONSIDER THE WHOLE

Figure 2 is a simple high-level picture that considers the whole system. It is the model used at the very start of performance tuning efforts mostly because it represents the whole system. Precedence is given to the whole system view to avoid making pre-judgments that could prematurely exclude parts of the system from the investigation.

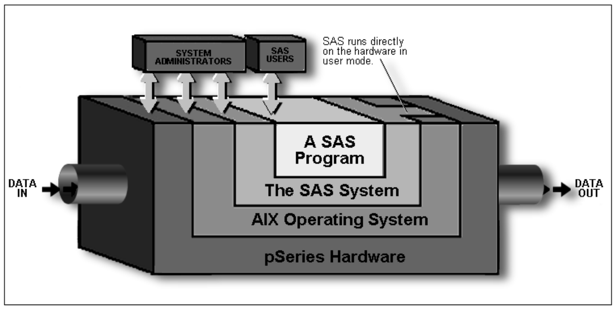
The behavior of a large computer installation depends not only on the hardware and software but also on the system administrators and the workload presented by the user community. There may be other applications running but our focus will be on system installations mainly serving a community of SAS users running SAS programs as the workload.

The SAS System layer in Figure 2 can be viewed as having its own internal kernel that is used to provide services to the rest of the SAS system. This approach allows SAS to function on multiple platforms and still maintain a consistent environment for the SAS user. The SAS kernel handles requests for service from the rest of SAS. Some requests can be handled completely within the SAS kernel. Other requests require the SAS kernel to construct and send one or more requests to the host system it is running on. In this case the host system is AIX.

Both the SAS System and AIX run directly on the underlying IBM pSeries hardware. The SAS System runs on the CPUs in USER mode and accesses memory in VIRTUAL mode. AIX can run on the CPUs in USER mode or SYSTEM mode, can access memory in VIRTUAL mode or REAL mode and can access peripherals in I/O space. On pSeries systems running in LPAR mode, there is also a Hypervisor layer between the operating system (OS) and the hardware where the Hypervisor firmware runs in HYPERVISOR mode. This paper will not delve into the Hypervisor Layer, since the OS basically hides it.

THE PERCEPTION OF PERFORMANCE

The majority of questions about the performance of a production computer system come in the form of questions about response-time and throughput. Response time is the elapsed time



- Figure 2 -

between when a request is submitted and when the response from that request is returned. Throughput is a measure of the number of workload operations that can be accomplished per unit of time. The conclusion that the response-time or throughput of a computer system is good or bad is often arrived at subjectively unless a framework of benchmarks is available for comparison and evaluation.

Most users on a computer system have opinions about its performance. Unfortunately, those opinions are often based on oversimplified ideas about the dynamics of program execution. Uninformed intuition can lead to expensive and inaccurate guesses about the capacity of the system and the solutions to the perceived performance problems.

As an example an IBM account team was working with a customers who had outgrown their PC based environment and replaced a room full of small servers with a p690. The customer needed to get more performance out of the p690 than they were seeing and decided that the system they installed was too small. They wanted to install more memory, more CPUs and switch to using Gigabit Ethernet. This recommendation was proposed as a solution by some of the new users of the p690 at the customer site. The account team ran a perfpmr (suite of performance measurements) and found that the applications that were showing poor performance were constantly creating, accessing and deleting temporary files on a server over the internal network. After removing restrictions, which were needed in the PC environment to limit memory consumption, the application's performance improved dramatically. One job that had to run overnight on the PC server farm and had been taking two hours to complete on the p690 now ran in 9 minutes.

HOW'S MY DRIVING?

The approach used during this study is that of distilling performance questions down to questions about how efficiently each layer in the model, shown in Figure 2, is driving the layer below it and being driven by the layer above it. In an ideal computer system each layer would "know" how to translate requests from the driving layers above into the speediest and most efficient operations possible. Also, requests sent to lower layers would be constructed to drive the lower layer in the most efficient way possible.

TUNING PARAMETERS

Since there are no ideal computer systems, the need for tuning arises. The designers of hardware, firmware, system software, databases, scientific and business applications, development environments, etc. usually provide parameters that can be used to modify the behavior of their component without direct modification of the component itself. Some of these parameters can usually be classified as *performance tuning parameters.*

WHY IS COMPUTER PERFORMANCE COMPLEX?

There was a time when a programmer could read a program, calculate the sum of the instruction times, and confidently predict how long it would take the computer to run that program. Thousands of programmers and engineers have spent the last 30 years making such straightforward calculations impossible, or at least meaningless.

Today's computers are more powerful than their predecessors, not just because they have far shorter cycle times, but because of innumerable hardware and software architectural inventions. Computers have gained capacity by becoming more complex as well as becoming quicker.

The complexity of modern computers and their operating systems is matched by the complexity of the environments in which they operate. In addition to running individual programs, today's computer deals with varying numbers of unpredictably timed interrupts from I/O and communications devices. To the extent that the computer's HW and SW designers' ideas were based on an assumption of a single program running in a standalone machine, modern computers may be partly defeated by the randomness of the real world. To the extent that those ideas were intended to deal with randomness, they may win back some of the loss. The wins and losses change from program to program and from moment to moment.

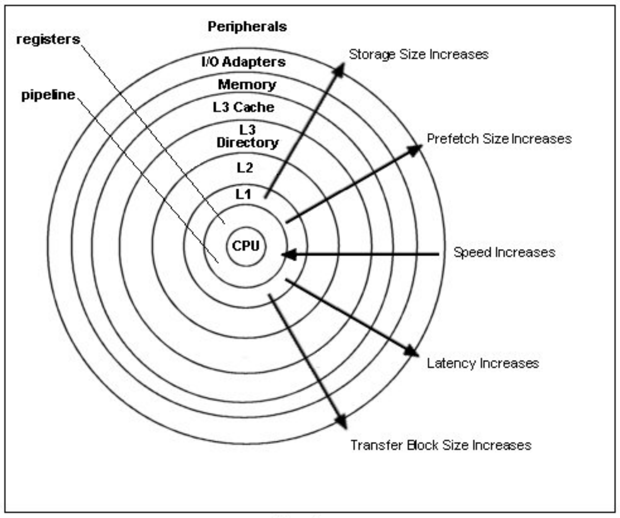
The result of all these hardware wins and losses is the overall performance of the system. Specific measurements of response time and throughput are only meaningful in the context of the sequence of demands of a given workload. If the demands mesh well with the system's hardware and software architectures, then the workload can be said to run with good performance on the system. It may not be accurate, however, to make the general statement that the system has good performance.

THE HARDWARE LAYER

The hardware layer of the system model is expanded in Figure 3. This figure is a generalized diagram of the hardware storage hierarchy of the Power4 based IBM pSeries systems used in this study. The concentric rings surrounding the CPU illustrate the performance properties of the various storage elements in the system. Storage element speed increases, latency decreases and storage capacity decreases as data is moved closer to the CPU.

The opposite is true as data is moved out to the outer rings: storage element speed decreases, latency increases and storage capacity increases. To compensate for the slower speed and longer latencies of the outer rings, wider busses are used to transfer data toward the CPU. As an example, these wider busses allow larger pSeries servers to have a peak main memory bandwidth of 204.8 GB/s. Also, specialized data prefetch hardware exists to start the movement of data toward the CPU ahead of time when sequential data access patterns are detected.

In Figure 3 the L3 directory ring and all rings enclosed by it are



- Figure 3 -

implemented on the Power4 chip. Power4 chips are available in speeds from 1.0 GHz to 1.45 GHz and each chip can support one or two active CPU cores. Each CPU core is a speculative superscalar out-of-order execution design and can issue up to 8 instructions each cycle with a sustain completion rate of 5 instructions per cycle. This design allows more than 200 instructions to be "in flight" at any given time.

Registers in the CPU are specialized storage elements. The CPU can directly operate on the contents of its registers. Data needed by the executing instructions must be moved into registers and the data that need to be saved are moved out toward memory.

Data destined for or coming from other layers of the storage hierarchy incur the access latency required to traverse intermediate layers. Access latencies increase rapidly the further away data are from the CPU. The Power4 CPUs have 244 registers and two 128-entry Effective-to-Real Address Translation (ERAT) tables. Both the registers and translation tables have zero latency. Access latency to non-cached, non-prefetched data in main memory is about 350 cycles. The best-case latency to access data that are cached on a fast PCI adapter using programmed I/O is at least 1,000 cycles. The time required to access data on a disk drive that requires head movement approximates many milliseconds or many millions of CPU cycles.

STRATEGIES FOR EFFICIENT HARDWARE DRIVING

Layers that drive the hardware layer need to follow these guidelines to maximize hardware performance:

- Data that the CPU needs to move into its registers should be found in the lowest-latency cache possible as often as possible.
- Prefetch data to minimize latency.
- Move as much data as possible with as few operations as possible.
- Once data is in a buffer do as much work on it as possible before moving it again.
- Keep the availability of fast memory high by flushing data that no longer needs to be in fast memory.
- Maximize cache hits, minimize thrashing.
- Exploit hardware parallelism.

PRINCIPLES OF PERFORMANCE TUNING

These are the foundation principles of the tuning approach used in this study:

- Continuously monitor performance
- Start with a big picture
- Set goals and determine the scope
- Create a controlled test environment
- Implement modifications one at a time
- Visualize and review data
- Know when to stop

CONTINUOUSLY MONITOR PERFORMANCE

Just as people continuously monitor their health by having regular checkups, the case can be made for continuous performance monitoring of a large computer installation. Continuous monitoring can:

- Detect emerging problems before they have an adverse effect
- Detect problems that exist but are not being reported by the user community
- Capture data when a problem occurs for the first time

Continuous monitoring involves:

- Obtaining performance data from the system at regular intervals
- Archiving the data as an historical record that can be searched
- Displaying summaries of the information for the system administrator
- Detecting events that require more detailed data collection

The AIX tools evaluated as useful for continuous performance monitoring are:

- iostat
- netstat
- sar
- vmstat
- PDT

START WITH A BIG PICTURE

As mentioned above, one should monitor performance over time, not just when problems or complaints occur. This will establish a baseline for usage patterns, and indicate performance trends in the system.

Create an overall conceptual model of the system. The model should include representations of bandwidth, latency, capacity, data transfer width, and speed. This will help to pinpoint where “hot-spots” may exist, and establish if these “hot-spots” are clustered within a specific area or if they are system-wide.

SET GOALS AND DETERMINE THE SCOPE

Set a performance goal to reach. This goal should be realistic, in other words it should be attainable, measurable, and appreciable. The scope of the tuning project should be checked periodically to make sure that is has not grown too large or become too narrowly focused.

CREATE A CONTROLLED TEST ENVIRONMENT

The first step in creating a test environment is to define the workload. Next the test applications must be designed to cover the area of concern. Sufficient controls must be placed over the execution environment to obtain high quality data.

Automating the tests may be useful. This may help simulate multi-user environments, or enable testing to occur during off-peak times when other users will not be affected.

Once the tests have been designed test the test to ensure that results are reliable and repeatable. It would also be helpful to determine the maximum performance that could conceivably be obtained.

IMPLEMENT MODIFICATIONS ONE AT A TIME

Whether working within a test environment or on a production machine, one should make a single modification at a time. If more than one parameter is modified, gains and losses may cancel one another out, and performance improvements or degradations could be masked.

It is also important to make sure that modifications can be “backed off”. One should be able to go back to the original settings if performance is not enhanced, or worse, if performance actually deteriorates.

VISUALIZE AND REVIEW DATA

Creating graphs of the data and taking other means to visualize the results will reveal trends, draw attention to problems and bring differences to light. The old adage that “a picture paints a thousand words” is certainly true here.

KNOW WHEN TO STOP

Repeat the modification and monitoring process as necessary. As the cycle of adjustment and observation proceeds look for the point of diminishing returns; the point at which removing a bottleneck in one place does not *expose* a bottleneck in another area, but in fact *creates* one. It is important to tune for the general case.

PERFORMANCE DATA IN THE SAS LOG AND HOW TO READ IT

The FULLSTIMER option needs to be turned on to capture the most detailed performance information in the SAS log. The FULLSTIMER sends detailed information on user CPU and system CPU times, as well as statistics on memory use to the SAS log. Performance is monitored for the entire SAS program, as well as for each PROC or DATA STEP. The FULLSTIMER is off by default, but can be turned on within a SAS program with the following statement:

```
   OPTIONS FULLSTIMER;
```

It can also be turned on from the command line, using the following command:

```
   $<SAS_INSTALL_DIR>/sas –fullstimer myprog.sas
```

Once the FULLSTIMER option is turned on, entries like the following will be recorded:

```
   NOTE: The SAS System used:
   real time 1:16.76
   user cpu time 1:12.35
   system cpu time 2.45 seconds
   Memory 1887k
   Page Faults 0
   Page Reclaims 1537
   Page Swaps 0
   Voluntary Context Switches 823
   Involuntary Context Switches 7512
```

These entries have the following definitions:

- **real time:** Elapsed wall-clock time
- **user cpu time:** The total amount of CPU time spent running in user mode
- **system cpu time:** The total amount of CPU time spent in system mode running on behalf of the process
- **Memory:** The maximum amount, in kilobytes, of memory used
- **Page Faults:** The number of page faults that resulted in real I/O requests. Also known as major page faults
- **Page Reclaims:** The number of page faults serviced without any I/O activity. I/O activity is avoided by reclaiming a page frame from the list of pages awaiting reallocation. Also known as minor page faults
- **Page Swaps:** The number of times a process was swapped out of main memory.
- **Voluntary Context Switches:** The number of times SAS made a system call that could not return immediately. These system calls are I/O requests or wait-for-a-resource type of requests.
- **Involuntary Context Switches:** The number of times SAS completely used a 10 ms time slice plus the number of times SAS was preempted by a higher priority thread.

TUNING FOR LARGE DATA SETS

The following information is from a performance investigation focused on:

- SAS option settings that determine the characteristics of the I/O requests to AIX
- AIX settings and configurations of the virtual memory manager (VMM), disk, paging and I/O subsystems on an 8-way IBM pSeries p690 partition with 7.5 GBytes of main memory.

There may be times when a large pSeries server or LPAR is dedicated to a single user, to run SAS. This provides the opportunity to tune the server or LPAR for specific use. As an example, even with data sets of several GB in size some SAS jobs can be made to run completely within main memory. This can be done by changing various parameters such as disabling the filesystem write-behind algorithm, mounting filesystems with the "nointegrity" mount option, using RAM filesystems, pre-loading shared libraries and so on.

Even without this tuning, much of the file I/O will be cached in main memory if there is enough available memory to 1) cache all of the files created and accessed by SAS and 2) keep all running programs and their data sections in memory.

In most cases, however, a community of users will simultaneously run jobs that compete for the resources of a large server or one or several users will have a smaller server or LPAR dedicated to them. This means that most concerns about SAS performance will involve I/O performance.

AIX aggressively caches disk blocks in memory and will use 80% or more of real memory as a file cache by default. If this cache becomes full the page replacement algorithm will start to run and filesystem performance will reach a plateau.

SUGGESTIONS FOR TUNING FILE I/O FOR LARGE FILES

- Make sure that BUFSIZE is equal to or evenly divisible into the transfer size (single disk) or stripe size (disk array) of the disk subsystem. Currently "BUFSIZE=64K" seems to work well.
- Use the JFS mount option "nointegrity" for filesystems that will only hold temporary data or that have data that can easily be reloaded or recreated. This will prevent system time and disk-I/O being spent on synchronous writes to the "jfslog". Note that if there is a power failure or the system crashes then the integrity of the filesystem's meta-data cannot be guaranteed, as it is when a "jfslog" is used.
- Use the JFS mount option "rbr" (release behind read) for filesystems with large files (they are large if they would occupy a significant portion of the file cache) that are read once during a job and do not need to be cached in the file cache. Once the requesting program is given the file data the VMM storage for the file is released from the file cache. Without this option the file data would continue to occupy memory with no benefit of reuse.
- Use the JFS mount option "rbw" (release behind write) for filesystems with large files that are written sequentially and require very little, if any, additional access during the rest of the job. Once the VMM passes the file data to the disk-I/O-subsystem the VMM storage for the file is released from the file cache. Without this option the file data would continue to occupy memory with no benefit of reuse.
- Use striped Logical Volumes.
- Use a high performance disk subsystem.
    - Use an ESS for top performance. If ESS-class storage is not available then...
        - If data can be reloaded or recreated in case of hard disk failure then use Raid-0 SSA arrays for high performance. Raid-0 has 100% storage efficiency.
        - Use Raid-5 SSA arrays to obtain medium performance if availability is more important. Raid-5 has 80% storage efficiency (for a 4+P array).
        - Use Raid-10 SSA arrays to obtain high performance and availability. Raid-10 has 50% storage efficiency.
- Tune Virtual Memory Manager (VMM) and filesystem parameters
    - minperm
    - maxperm
    - maxperm
    - maxpgahead

- minfree
- maxfree
- numclust
- numfsbufs
- hd_pbuf_cnt
- sync_release_ilock
- maxclient
- j2_nBufferPer
- j2_minPageReadAhead
- j2_maxPageReadAhead

USEFUL PERFORMANCE TOOLS

- **filemon** - Monitors the performance of the file system, and reports the I/O activity on behalf of logical files, virtual memory segments, logical volumes, and physical volumes.
- **topas** - The topas command reports selected statistics about the activity on the local system.
- **vmtune** - Displays or changes operational parameters of the Virtual Memory Manager and other AIX components.
- **vmstat** - The vmstat command reports statistics about kernel threads, virtual memory, disks, traps and CPU activity. Reports generated by the vmstat command can be used to balance system load activity. These system-wide statistics (among all processors) are calculated as averages for values expressed as percentages, and as sums otherwise.
- **iostat** - Reports Central Processing Unit (CPU) statistics and input/output statistics for the entire system, adapters, tty devices, disks and CD-ROMs.

CONCLUSION

Meeting the performance targets of a large computer system is a complex task but tuning the system is only part of the battle. Maintaining high performance levels requires a constant base level of system monitoring. The variability of workloads on some systems requires frequent tuning, while other systems need only occasionally tuning. In either event, a tuning approach driven by performance data allows continual refinement of performance models and methods. Conceptual models of system performance help the system administrator organize and compartmentalize the performance tuning task while still maintaining a system-wide view. This approach avoids the pitfall of tuning based on the individual user's perception of system throughput and responsiveness at the expense of system-wide efficiency.

REFERENCES

AIX 5L Version 5.2
Performance Management Guide
Fifth Edition (October 2002)
© Copyright International Business Machines Corporation 1997, 2002. All rights reserved.

IBM Certification Study Guide
AIX Performance and System Tuning
Second Edition (December 2002)
SG24-6184-01
© Copyright International Business Machines Corporation 2002. All rights reserved.

AIX 5L Version 5.2
Performance Tools Guide and Reference
First Edition (October 2002)
© Copyright International Business Machines Corporation 2002. All rights reserved.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Frank Bartucca and Torre DeVito
IBM ICC At SAS Institute
Building R
Cary, NC 27511
919-531-1420
ibmsas@us.ibm.com
http://www.ibm.com