The Age-Old Question of How to Balance Batch and Interactive

Barry Sharp
Boeing Shared Services Group - Technical Services
High Performance Computing - Engineering Operating Systems
The Boeing Company, P.O. Box 3707 MC 7J-04, Seattle, WA 98124-2207, USA
barry.sharp@boeing.com
Copyright (c) 1998, The Boeing Company, all rights reserved

Abstract

The UNICOS system is designed with both batch and interactive workload requirements in mind. However, in practice, the vanilla UNICOS kernel memory scheduler struggles to adequately balance these two distinct workloads, even with its wealth of scheduler tuning parameters. This paper presents a simple modification to the kernel-level memory scheduler that works harmoniously with the vanilla system to simplify the process of balancing the two workloads.

1.0 Introduction

The UNICOS software allows a site to tailor the system to schedule work activities so that desired throughput and/or performance goals can be met. The tailoring can be performed at system startup or dynamically during production to handle changing conditions or simply done at preset times of the day, such as daytime versus nighttime. The tailoring instructions are transmitted to the OS kernel software using the ***nschedv*** command. UNICOS also provides a Job Category feature that allows control over where process memory images are placed on the swap device (SWAPDEV). This SWAPDEV placement strategy is tightly coupled with the work scheduling. It is important to note that work scheduling is done in two phases. Firstly, new work, batch and/or interactive is scheduled into the system and, secondly this new work starts to be managed by the kernel memory scheduling which then eventually competes for CPU time on a priority basis. NQS is used to schedule new batch work into the system for execution and will immediately compete with the interactive workload. The interactive work is initiated by services such as ***inetd***, ***ftpd***, and ***rshd*** and normally is short-lived (but not always so). After new work has been scheduled for execution the kernel memory scheduler is the primary facility that governs all work throughput and/or performance.

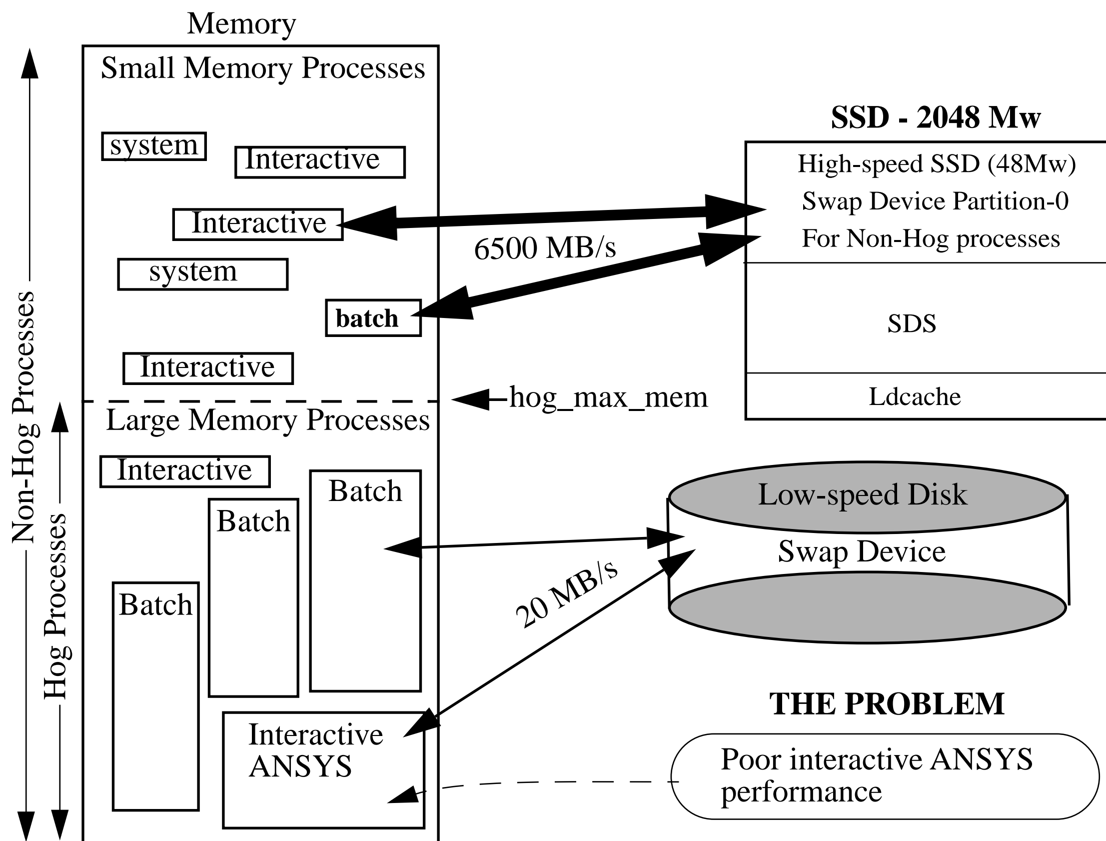
The kernel memory scheduling is the main focus of attention in this paper. The kernel CPU scheduling is not discussed and was not seen as a problem with respect to balancing the batch and interactive workloads. The simple matter was that a process cannot gain access to the CPU without first being in memory!

Balancing the customer requirements for batch and interactive work on a single UNICOS 7.0 system was manageable for few years, but then proved to be very difficult when our engineers began an aggressive schedule in using large memory X-Window-based interactive ANSYS codes for creating and analyzing 3-D solid models. A major part of their decision making process relied on quick turn-around. This situation drove us to spend time to improve our understanding on how the kernel arbitrated the memory and CPU resources between batch and interactive work. With this understanding, and after unsuccessfully using ***nschedv*** to obtain the desired balance between these two work loads, a simple modification to the kernel memory scheduling was designed. The remainder of this paper presents this design.

2.0 Initial attempt to balance the work loads

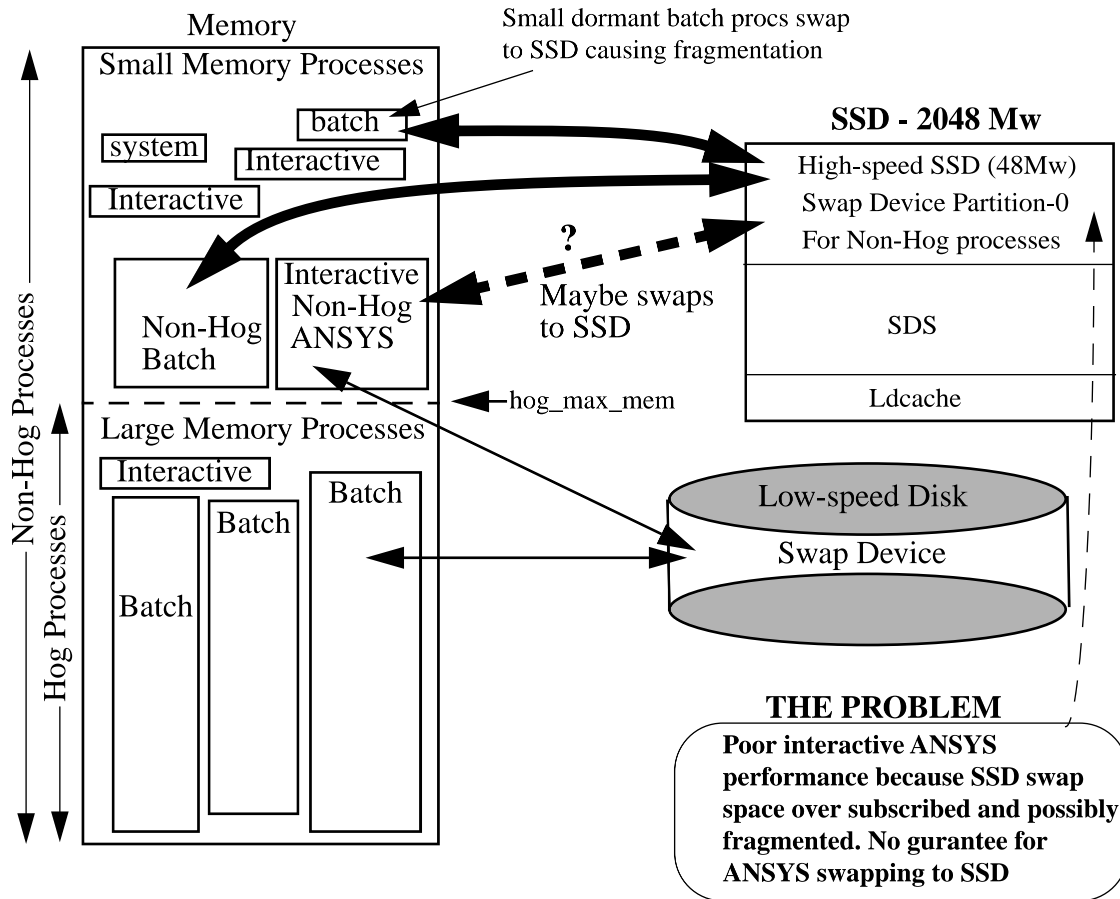
As mentioned above, the initial attempt to resolve the problem was done by adjusting the ***nschedv*** parameters. The parameters were set to cause the large memory ANSYS codes to swap to the high-speed swap partition on the SSD by making it a Non-Hog process.

Figure 1 - Original Memory Scheduling and Swap Space Strategy



This was achieved by simply changing the ***nschedv -h memhog*** value to be just slightly larger than the size of the ANSYS memory size. This caused the ANSYS program to be classified as a Non-Hog process. The configuration we had prior to this time for memory scheduling and swap space strategies is given in Figure 1. In this figure, the ANSYS program is a Hog process. Figure 2 depicts how Figure 1 changed when the above ***nschedv*** was implemented to make the ANSYS program a Non-Hog.

Figure 2 - Memory Scheduling and Swap Space Strategy



Swapping the large Non-Hog ANSYS processes to the SSD provided only a marginal improvements. Speculation as to why, promoted the idea that other Non-hog batch processes were also swapping to the SSD and residing there for long periods of time in a dormant state while their active children were performing all the real work. If this was correct, then this aspect would mean the SSD swap space for Non-Hog interactive processes, such as the ANSYS code, would be or could be denied. They would instead be swapped to the slow-speed disk devices. The swap device (the SSD partition included), can at times, become annoyingly fragmented which inhibits the swap placement strategy from working successfully.

The above speculation was proven correct once we had the ***swapinfo*** command (see below) written and running. The ***swapinfo*** utility proved to be enormously helpful during

the development cycle. In fact, even today, most analysts and QA testing employ this utility for improved visibility on the overall process memory allocations and other miscellaneous items it provides.

Having failed to achieve the right balance between batch and interactive using ***nschedv*** we were convinced the answer lay in having more precise control over interactive and batch processes contending for memory and swap space. This control mechanism would also need to influence where these type processes get swapped on SWAPDEV.

3.0 Functional Requirements

We required a memory scheduling design that would allow precise control over how memory is allocated based on whether a process is associated with NQS (batch) or interactive. With this in mind the following functional requirements were arrived at.

- Provide one or more new memory scheduling algorithms which must be controlled using the standard ***nschedv*** command. The initial implementation was to consider the following three scheduling policies:

    a. Allow memory to be *precisely* partitioned between interactive and batch work

    b. Allow *small* batch work to encroach on interactive memory partition space

    c. Force *large* interactive work to contend for space in the batch memory partition

- Re-configure physical devices used for swapping to improve swap i/o bandwidth
- Configure SWAPDEV partitioning to assist with the new memory scheduling goals
- As space in the SSD swap partition is view as being critical for interactive performance the Cray released software should be modified to avoid inappropriate use of the SSD swap partition. An example of this would be to avoid having SSD used when suspending large memory batch jobs
- Provide software to monitor the system's Memory scheduling, Job category assignments, and swap placement strategy

4.0 Discussion

The focus of all memory scheduling by the kernel is in the sched.c module. This module is concerned solely with the arbitration of memory when processes require memory that is not immediately available. It also manages the Hog memory allocations and Job categorizations. When ***nschedv -i 1*** has been defined all processes are assigned a Job Category. This categorization affects their initial or preferred swap device partition placement. For example, Table 1 and Table 2 indicate preferred process swap image placement when SWAPDEV is partitioned into 2 and 3 segments respectively.

TABLE 1. Swap Placement for 2 swap partitions

Job Category 1	Job Category 2 & Job Category 3
Swap Partition-1	Swap Partition-2
System processes	Batch Non-Hog processes
Super user Non-Hog processes	Interactive Hog processes
Real Time processes	Super User Hog processes[a]
Shared Text processes	Overflow from Category 1
Interactive Non-Hog processes	Hog processes

a. A super user process CANNOT be marked as a CPU-Hog

TABLE 2. Swap Placement for 3 swap partitions

Job Category 1	Job Category 2	Job Category 3
Swap Partition-1	Swap Partition-2	Swap Partition-3
System processes	Batch Non-Hog processes	Hog processes
Super user Non-Hog processes	Interactive Hog processes	Overflow from Category 2
Real Time processes	Super User Hog processes	
Shared Text processes	Overflow from Category 1	
Interactive Non-Hog processes		

In addition to Job Categories for swap placement, processes are also assigned a process category by sched.c based on their interactive or batch origination. This category affects how quickly a processes can swap out or in.

The following difficulties were quickly determined during initial observations on how the system performs swap placements.

- Determining the configured swap partition sizes is somewhat difficult (use /etc/crash)
- Determining which swap partition a process resides in
- Determining whether swap placement overflow has occurred
- Determining whether system has correctly categorized a process because:

    a) it can change rapidly as it changes between Hog and Non-Hog

    b) it can change rapidly when critical kernel resources obtained/released

    c) system may be slow in assigning a category if memory under-subscribed

Some additional notes are also worth mentioning here. The kernel memory scheduling is performed in a strict order.

First, processes such as indicated below are considered for memory allocations

1. The swapper process (pid=0) can NEVER swapped
2. The init process (pid=1) CAN be swapped (we however have modified init to be locked into memory)
3. Idle processes (pids=2,3,4,....NCPU+1) can NEVER be swapped
4. A real-time process
5. A process that is holding critical locks such as ldcache locks
6. A runnable process in a kernel thread with inodes locked
7. Super user processes (i.e., pc->pc_uid==0 && pc->pc_suid==0) cannot be a CPU-hog

Secondly, process such as those indicated below are considered for memory allocations

1. 1st - Interactive processes if ***nschedv -i 1*** defined and processes that are not system processes nor belong to a batch NQS session
2. 2nd - Processes belonging to a batch NQS session

5.0 Kernel Modifications

Memory scheduling is driven by swapin activities. The kernel memory scheduler sched.c evaluates processes on the SWAPDEV device and determines which processes should be swapped out and which processes should be swapped in. What swaps in and what swaps out is determined by process priority, Job category, and Hog status. As mentioned in the Discussion section, Job category can change frequently depending on certain system resources a process may be holding, such as inode or ldcache locks. In general, the lower the process Job category, the higher its priority for memory becomes. Job categories range from 0 to 9. There is a direct relationship between Job category and the process type (kernel, real-time, root, Non-Hog, and Hog processes) and whether it is interactive (the system code refers to these as being non-batch) or batch originated. The Job categories are contained in the proc entry structure elements pc_swcat_in and pc_swcat_out. Most of the time these two fields are identical. However, they have been observed to be different and probably depends on what the kernel decides needs doing to expedite some action on behalf of the process (could be that the process is holding a critical resource which should be released as quickly as possible). The pc_swcat_in and pc_swcat_out values are used by the kernel in deciding where the preferred place on SWADEV the process should be swapped. As the kernel will at times manipulate these two values it is possible to see a process swapped to a place that it is really not meant to be. However, this situation does not last for much more than a few seconds.

A batch process is defined by the S_BATCH flag in the process session table structure element s_flag.

The sched.c module was modified to accommodate the requirements for improved memory scheduling. Other ancillary modules that also required changes were fork.c, grow.c, lock.c, slp.c, and text.c (with UNICOS 8 clock.c was also modified). All of these kernel modules come into ‘play’ when a process grows or shrinks its memory size such as for

exec(2), fork(2), and sbreak(2). The modifications made to these modules was in direct support to honor a new -S option made to the ***nschedv*** command (see External Modifications). All local code was very carefully integrated into these modules to not disturb any of the vanilla code (no vanilla code was deleted at all). This technique has proven to be of great benefit when re-applying this feature during major system release upgrades. Except for UNICOS 7 to 8 when the kernel was heavily modified with multi-threading locks, the effort to apply the local code has taken just a few hours.

The header file schedv.h was also modified to include some new ‘Behavior modification flags’ in the memory scheduler structure element sv_flags.

One other unanticipated kernel modification was found to be necessary during several attempts to configure the swap device partitions. This was in the module swap.c. The mod in swap.c turns off the round-robinning between partitions 2 and above if the number of partitions is exactly 4. This was done because it was required to have one large high-performance swap partition for batch jobs and be able to add more partitions if additional swap space was needed. These extra partitions would presumably be lower performing and would be used only as a last resort. This code is not in use now since 5 partitions are configured, and partitions 2, 3, and 4 are exactly the same size and have identical i/o performance (see ***swapinfo*** screen output below). This swap partition configuration allows the release code to round-robin between partitions 2, 3, and 4.

6.0 External Modifications

The /etc/nschedv command was examined and a new option (actually what was chosen was the defunct option S which was already coded into /etc/nschedv) was decided on as follows.

/etc/nschedv -S option

where

0 - Disables local memory scheduling and reverts to Cray’s default

1 - Selects local memory scheduling policy #1

2 - Selects local memory scheduling policy #2

3 - Selects local memory scheduling policy #3

Memory scheduling policy #1

All batch processes are classified as Hogs and all interactive processes are classified as Non-Hogs, regardless of memory and/or CPU usage. The /etc/nschedv -H hog_max_mem can therefore be used to control the total amount of memory that can be used by all batch processes at any given time. The /etc/nschedv -h memhog -c cpuhog options have no meaning when -S 1 is used.

Memory scheduling policy #2

Only batch processes are classified as Hogs per the /etc/nschedv -h memhog -c cpuhog options. All interactive processes will be classified as Non-Hogs regardless of their memory and/or cpu usage. The /etc/nschedv -H hog_max_mem can therefore be used to control the total amount of memory that can be used by all batch Hog processes at any given time.

Memory scheduling policy #3

All batch processes are classified as Hogs and all interactive processes are classified per the /etc/nschedv -h memhog -c cpuhog options. The /etc/nschedv -H hog_max_mem can therefore be used to control the total amount of memory that can be used by all batch and interactive Hog process at any given time.

7.0 Memory Scheduling and Swap Placement Validation

UNICOS does not provide any practical way for observing process Hog status, Hog memory usage, process Job Categories, Interactive and Batch memory usage, or swap space usage and process image swap placement. The only facility that provides some information is /etc/crash and, maybe /usr/bin/sar (at a stretch). The author has observed /etc/crash reporting erroneous data when memory is not over-subscribed.

To facilitate validation and monitoring of the various activities mentioned above, a program called ***swapinfo*** was created. This program employed ‘curses’ to manage its screen format and output and used highlighting (reverse video) to identify problems as they occurred. Optionally, problem reporting can be captured and written to a time-stamped logfile for post analysis.

The basic operation of ***swapinfo*** is to periodically gather various pieces of information from the kernel tables and then scans the process table to arrive at a picture of what processes are loaded in memory and those that are swapped. The information is then displayed or updated on the screen.

The swapinfo command was given several command line options as follows:

swapinfo [-a] [-c 0|1] [-l logfile] [-o 1] [-p passes] [-r refresh] [-v 1|2|3|4]

All of these options can be selected dynamically during ***swapinfo*** execution. A typical screen output using a verbosity level 2 (-v 2) is given below. Note the ‘*’ in the table showing swap partition usage indicates where the round-robin swap partition rotor pointer is. In the example provided the ‘*’ rotates among partitions 2, 3, and 4. The small partition 1 is there just to make the right partition be partition 2, the one where the round-robinning starts. It is of no consequence if the processes assigned to partition 1 have their own partition.

```
UNICOS: Swapper Usage      13:50:01                 Pollrate: 3 Passes: INDEF
BCS Memory Scheduling Mode-1 ..     [-h 2048  -c 0 -H 889791 -i 1 MPX Swapper]
Total swap space being used .......          0.11%     (   5.88 Mw)
Maximum swap space used so far ....          0.11%     (   5.88 Mw @ 13:50:01)
Top ten largest swap holes in Mws .       1715.53, 1676.91, 1594.92, 120.69, 116.26,
                                              38.70,    11.67,     4.80,    4.34,    2.69,
Smallest largest swap hole seen ...        3513408    (1715.53 Mw)
In-core Interactive memory demand..          116704  (  56.98 Mw, Max   56.98 Mw)
Swapped Interactive memory demand..            6880  (   3.36 Mw, Max    3.36 Mw)
In-core Batch       memory demand..          173920  (  84.92 Mw, Max   84.92 Mw)
Swapped Batch       memory demand..            1280  (   0.63 Mw, Max    0.63 Mw)
Number of swapped SDS images.......               0  (Max seen   2)
Total swapped SDS space............               0  (   0.00 Mw, Max seen= 228.13 Mw)
Current largest swapped SDS image..               0  (   0.00 Mw, Max seen= 700.00 Mw)
Machine's allocatable user memory .          96.78%  ( 456.48 Mw)
In-core user memory being used ....          290624  ( 141.91 Mw)
Total user memory subscription ....          31.83%  ( 145.97 Mw, Max 145.97 Mw @ 13:50:01)
Bat/Int hog status errors found ...            0/  0 This pass,      0/     0 Accumulative
Total Hog memory subscription ....           19.69%  (  85.55 Mw, Max  85.55 Mw @ 13:50:01)
Computed    incore Hog mem inuse ..          173920  (  84.92 Mw, 58 procs)
Kernel says incore Hog mem inuse ..          173920  (  84.92 Mw)
Max memory for hogs (hog_max_mem) .          889791  ( 434.47 Mw)
Swapped Hog memory being used .....            1280  (   0.63 Mw, 8 procs)
Pid and size of largest process ...           24545  (  34.47 Mw) Memory resident
NonHog Inter In-Core/Swapped Mem ..          116704  (  56.98 Mw)/   3200 (   1.56 Mw)
NonHog Batch In-Core/Swapped Mem ..               0  (   0.00 Mw)/      0 (   0.00 Mw)
Hog Inter In-Core/Swapped Mem .....               0  (   0.00 Mw)/      0 (   0.00 Mw)
Hog Batch In-Core/Swapped Mem .....          173920  (  84.92 Mw)/   1280 (   0.63 Mw)
Swap partition allocation errors ..               1  This pass,      1 Accumulative
Multitasked sessions .............                2  In-Memory,      0 Swapped
Number of plocked processes .......               7  (   2.05 Mw)
Number of frozen processes ........               0  (   0.00 Mw)
Number of hog_tied processes ......               0  (   0.00 Mw)
Number of times hog proc swaps ....               0
Number of in-core/swapped procs ...             396  (max 396) /      30 (max 30)
```

Swap Part-#	Starting Block	Part Size Blocks	Times Used	Blocks Used	Percent Used	# Procs Swapped	# TxtEnt Swapped
part-0	0	312576	11764	14176	4.54	23	19
part-1	312576	784	0	0	0.00	0	0
part-2*	313360	3513888	16	320	0.01	2	0
part-3	3827248	3513888	16	480	0.01	3	0
part-4	7341136	3513888	16	320	0.01	2	0

```
Swap Partition maximums seen
Part#          0                   1                   2                   3                   4
Max:   4.54% @13:50:01   0.00% @13:50:01   0.01% @13:50:01   0.01% @13:50:01   0.01% @13:50:01
```

Job Cat	Swap Space	Job Cat	Swap Space
0	6848	1	0
2	0	3	0
4	192	5	0
6	0	7	0
8	0	9	1120

Work	In-Core Job Categories						
Type	Kern	Real	Root	Non-Hog	Hog	Total	Max
Inter	0	7	185	146	0	338	338
Batch	0	0	0	0	58	58	58

Work	Swapped Job Categories						
Type	Kern	Real	Root	Non-Hog	Hog	Total	Max
Inter	22	0	1	0	0	23	23
Batch	0	0	0	0	7	7	7

8.0 Unanticipated Bonuses

The simple control mechanism provided by the /etc/nschedv -S 1 command allowed memory to be logically partitioned between interactive and batch workloads. By monitoring the accumulative interactive memory demands the /etc/nschedv -H hog_max_mem command can be used to periodically change the partitioning (within certain site restricted limits) so that interactive performance can be maintained. The most important aspect for good interactive performance is to ensure there is sufficient memory for interactive processes.

This has proven to be a real bonus as the interactive load, once being a day time occurrence during week days, is now quite a variable. This interactive work load can at times extend well into the night time and even is ongoing during weekends when engineers are working to tight schedules. Rather than constantly asking the engineering community for their interactive forecasts, the system is simply left to adapt itself automatically to the interactive load as it comes and goes.

This technique has now been used for about 4 years and has relieved us having to spend time and effort in addressing customer concerns about interactive performance. The progression in partitioning memory between interactive and batch over the past few years is shown in Figure 3. As can be seen, the interactive memory demand causes the hog_max_mem value to vary from a define maximum to defined minimum threshold. The minimum threshold is set to not allow interactive to consume too much memory and the maximum threshold is set to allow space for system daemons etc to have room for memory at all times.

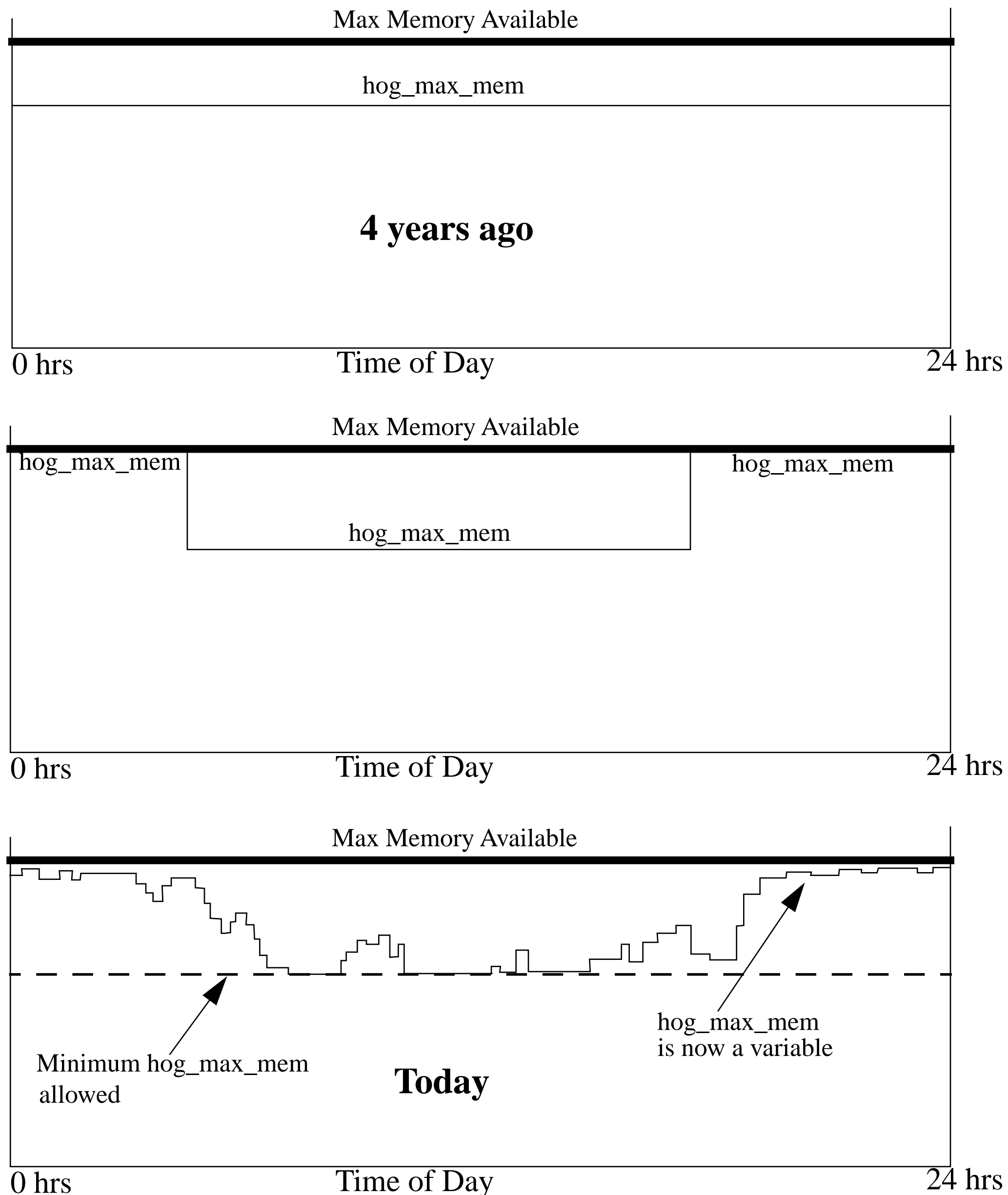


Figure 3 - Progression of how memory partitioning occurred

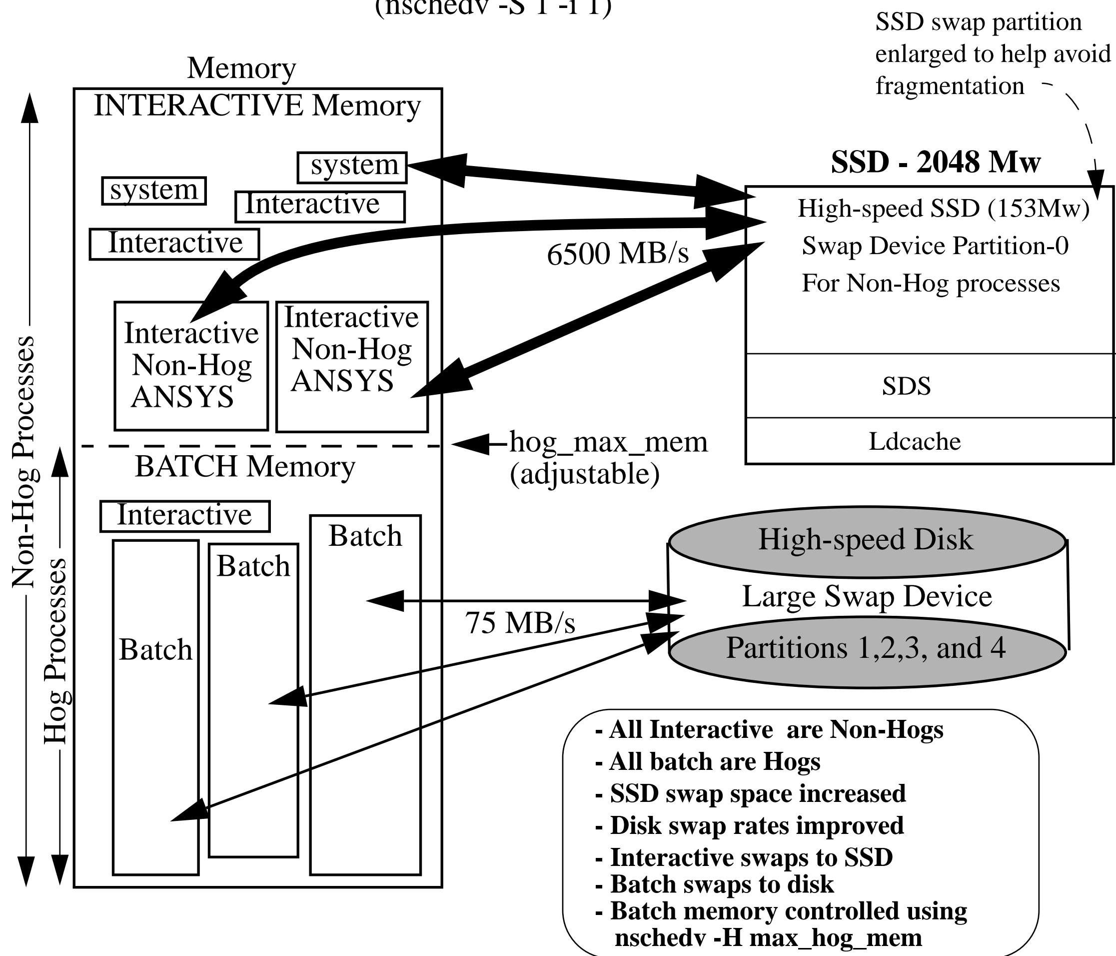
The ***swapinfo*** command reports the top ten largest ‘holes’ in SWAPDEV. This has been useful information that has helped us understand why some large SDS jobs could not be preempted. When an SDS job is preempted by URM its SDS image is copied to the SWAPDEV and its SDS arena released. For a large SDS arena to be copied successfully there must be a contiguous section on SWAPDEV (i.e., a ‘hole’ large enough must exist) to hold the job’s SDS arena image. Armed with the ***swapinfo*** display it is then just a matter

of time observing the behavior and making an informed decision to reconfigure and/or upgrade the SWAPDEV hardware.

9.0 Summary

Once the mechanics of kernel memory scheduling were understood it took just a small amount of time and coding in seven kernel modules and one header file to introduce a suitable local memory scheduling policy for solving the problem. The external design allowed the policy to be easily implemented using the ***nschedv -S #*** command. Figure 4 depicts the configuration employed today using ***nschedv -S 1 -i 1.***

Figure 4 - Current Scheduling and Swap Space Strategy
(nschedv -S 1 -i 1)



The actual implementation proved to be a lot easier than was first thought, and the local code integration effort has been of no real consequence during UNICOS major release upgrades.

Interestingly enough, even though three local memory scheduling policies were designed, policy 2 and 3 have not ever been employed during production. Our QA testing always

ensures they work though, just in case we have second thoughts on the matter. Policy 1, as it happens, was all that it took to resolve our balancing problem.

The ***swapinfo*** program has shown itself to be an admirable addition to our toolkit. It provides invaluable insight on the aspects of kernel memory scheduling, swap space management, swap space allocations, and possible mis-behavior.

By monitoring the interactive memory demand the logical memory partitioning for interactive and batch work loads can be made dynamic quite easily. This technique allows the system to automatically adapt itself to the interactive demands.

10.0 Acknowledgments

Credit must be given to a past CRI employee, Henry Newman, for giving a very insightful CUG ‘92 workshop session on the UNICOS kernel Memory Scheduling Strategies using ***nschedv*** and the Job category features. The workshop handouts provided the author with an excellent reference point from which the seed for the enhancement presented here came from.

Also, credit must go to three very worthy Boeing employees, Bill Matson, Mark Lutz, and Dave Atler. To Bill, for integrating the local kernel mods into several new UNICOS major releases without one word of complaint, to Mark for diligently working the kernel swap.c problem related to the round-robinning swap image placements, and to Dave for asking me tough questions on how this or that worked or wanting explanations for some strange anomaly he saw during his QAing.