Mass Storage System Performance Prediction Using a Trace-Driven Simulator

Bill Anderson
National Center for Atmospheric Research (NCAR)[*]
Boulder, CO 80305
andersnb@ucar.edu

Abstract

Performance prediction of Mass Storage Systems can be difficult because of the complexity of the systems and the interdependence of the components. This difficulty can lead to over or under provisioned systems and can limit the ability to identify software algorithms (such as cache management or request ordering algorithms) that scale well and yield high performance. Moreover, as Mass Storage Systems scale to a petabyte and beyond, the ability to predict performance could become increasingly important since errors in capacity planning could also grow.

This paper discusses a trace-driven discrete event simulator that we have developed to aid us in ranking design and configuration alternatives. The simulator reads in a configuration file, ingests a workload and estimates multiple metrics, including average user response times, cache hit ratios and device utilization. Simulated components include tape drives, disk systems and software components.

The simulator has been used to help us determine the size of a disk cache that is used to offload reads from tapes; we found that, for files with sizes under 50 MB, a cache size of around 8 TB can provide a read hit ratio of approximately 67%. The simulator has also been used to estimate the number of STK 9940B tape drives needed to replace our 9840A and 9940A drives. Based on validation runs, we have found that metrics predicted by the simulator are within approximately 20% of the actual values.

1. Introduction

Mass Storage Systems are complex with many interdependent components and configuring and tuning them can be challenging[1]. NCAR's Mass Storage System (MSS) currently holds 2 PB of total data in a little over 24 million bitfiles. The MSS processes over 1.2 million user reads and writes each month plus a similar number of reads and writes for internal data migration. Given the complexity of the workload and of the system itself, we have found that predicting the performance of the system in response to configuration changes can be difficult. For example, while we can measure the current read hit ratio of a disk cache, it can be hard to estimate if a larger disk cache will lead to a commensurate increase in the hit ratio or if a smaller cache would provide the same hit ratio. Another example would be to estimate how average user response time (calculated as the time from when a user initiates a file transfer to when the transfer is completed) varies as a function of number of tape drives in the system. Would more tape drives improve user response time significantly?

One way to determine the effect of a configuration change is to make the change and then track the system performance. However, that approach can be expensive with an operational MSS, may bring no substantial improvement to the system and can require months of tracking the system performance to assess what effect the change had. Also, the workload might change during that time and it might be difficult to know if the resulting changes in performance were due to the system configuration changes or to workload changes.

Analytic models (e.g., those based on queueing theory) are another option for estimating the effect of configuration changes on performance. Most of these models require many simplifications and assumptions about the workload and system behavior, however, limiting their accuracy[2].

To aid us in ranking design and configuration alternatives, we developed a trace-driven discrete event performance simulator of our MSS. A simulator allows us to model the MSS system more accurately than analytic approaches and provides a framework to experiment with different configurations without having to procure or test hardware and make possibly disruptive changes to our production system. However,

[*] The National Center for Atmospheric Research is sponsored by the National Science Foundation and operated by the University Corporation for Atmospheric Research.

all approaches to performance analysis and modeling have their weaknesses. While simulation modeling offers the ability to model the system more accurately, it also usually requires more time to develop a simulation than to develop an analytic model. Our simulation required approximately 1.5 person years to develop. The investment appears to be commensurate with the benefits, however.

We have used the simulator to help us in sizing a disk cache to offload reads from tape and choosing disk cache management policies. The simulator predicted that a read hit ratio in the range of 65-68% was attainable with a modest sized cache; beyond that, greater cache sizes resulted in small gains in the read hit ratio. Based in part on those simulator results, we chose a cache configuration that the simulator predicted would result in a read hit ratio of 67%. After the cache was put into production, the actual average hit ratio from February 2004 to July 2004 has been 65.6%, although it does fluctuate from month to month as the workload fluctuates.

The next section describes some related work and Section 3 provides more details about the simulator. Section 4 describes some results from the disk cache study, section 5 describes some results from a tape drive and disk cache study, section 6 describes some limitations of the simulator and our conclusion and summary are presented in section 7.

2. Related Work

Simulations and analytic models have been used to improve and understand the performance of computer systems and subsystems for decades. Detailed simulators have been developed for disk subsystems (e.g., the *Pantheon* disk subsystem simulator [3] and the *DiskSim* disk subsystem simulator [4]) and for tape libraries (e.g., [5]). In addition to subsystem-specific work, simulation and analytic models of Mass Storage Systems have also been developed (e.g., [6] and [7]).

Our work builds on this past work and extends it in a couple of ways. First, our simulator incorporates all the major hardware and software components of our MSS rather than focusing on a subsystem. Simulators that focus on a specific subsystem and that do not model the other subsystems may not be able to estimate the net effect of system changes on global metrics such as mean user response time. Achieving high performance in one subsystem may not result in high performance for the system as a whole since another component could be the bottleneck.

Second, our work extends the previous simulation and analytic models of Mass Storage Systems that we are aware of by increasing the accuracy of the results. For example, the workload of our simulator is based on traces of actual user activity that capture the detailed workload patterns more accurately than analytic models can and than previous simulators that we are aware of have. Also, our simulator models the behavior of the major software components in detail. The software used to control Mass Storage Systems can be complex with many algorithms that affect the performance of the system. By modeling these components in detail, we can also estimate their effect on performance.

3. Simulator Description

This section gives a high-level overview of the type of simulator and its development environment. It then provides a synopsis of the NCAR MSS, how the NCAR MSS was simulated and how the simulator was validated.

3.1. Simulator Type and Development Environment

The simulator we developed is a discrete-event simulator, which means the primary state variables of the simulator (such as number of jobs in the system) take on discrete values. The workload used to drive simulators is usually either internally generated by the simulator using random number generators or based on traces. Traces are usually derived from an actual workload on a system. Since we had traces of all user read and write events in our MSS logs, we used a trace-driven approach. The traces of the user reads and writes over time periods of a month or more are ingested by the simulator and used as the workload for the simulated system. For a good overview of simulation modeling, see [2] and its references.

There are numerous simulation languages and packages available both commercially and freely. For certain domains, such as telephone call centers and factory assembly lines, there are packages that can be easily tailored to model those systems and that would save substantial time over developing a simulation from scratch using a general purpose language. However, we were not able to find an equivalent package for Mass Storage Systems. After researching several languages and packages, we decided to use Java along with a package called JavaSim[8]. This combination resulted in a nice environment for creating a detailed simulation of our system using a contemporary, generally available language.

The development environment is similar to the one provided by the SIMULA language [9], but is a more readily available development environment. The simulation approach that this environment provides is generally referred to in simulation terminology as a *process-oriented* approach (in contrast to an *event-oriented* approach).

All components of the simulator are Java objects. Additionally, each component of the simulator that performs time dependent operations is also a Java thread (a runnable or active object) and the simulator management software controls the activity of the threads. For example, a tape silo takes time to transfer a tape from a cell to a drive, so each silo is modeled using a thread. In contrast, a component that does not have operations that require simulated time to perform is also an object, but not a thread. For example, there are Java classes to represent tape media. A medium has attributes such as capacity and type; however, the medium component itself does not simulate the passage of time. Below is an example that provides an overview of how the passage of time is simulated using threads.

Consider a single silo that receives requests to mount and dismount tapes as shown in Figure 1.

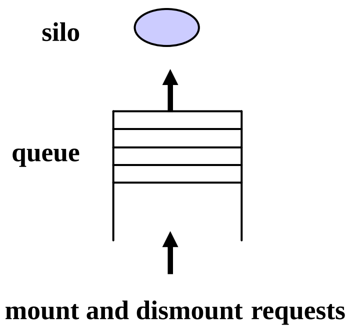

Figure 1. Logical View of a Silo

Assume the silo is idle. At that point the silo thread is in a suspended state. Then, a mount request arrives in the silo queue. The silo thread is resumed at that point, detects that it has a mount request, starts processing it and then estimates how long it would take for a real silo to mount the tape. In our current version of the simulator, it obtains this estimate by looking in a table of constant values. Suppose the estimate is 4.5 seconds. The silo thread then makes a function call to the simulator management software telling it to suspend it and then to resume it after 4.5 simulated seconds have elapsed. The simulator management software suspends the silo thread at that point. When 4.5 seconds pass on the simulator clock, the simulator management software resumes the silo thread. The silo thread starts running and sees that it has finished mounting the tape. It updates internal data structures as necessary and then requests that it be suspended until the next mount or dismount request arrives. The passage of time is simulated for the silo by suspending and resuming the thread. The passage of time is simulated for the other components in the same way. Note that while the program has many threads, only one thread is running at a time. The threads are used as a way to maintain state and to simulate the passage of time for components and not for parallel processing.

3.2. Simulating the NCAR MSS

The NCAR MSS uses an *rcp* (remote copy) approach where users transfer complete files from the MSS to client filesystems and from client filesystems to the MSS; it roughly follows the IEEE MSS Reference Model [10]. Figure 2 is a high level logical diagram of the NCAR MSS.

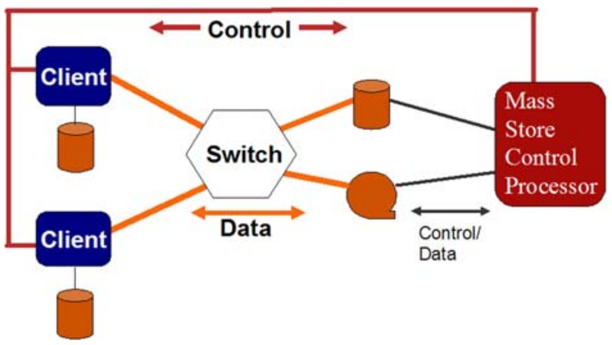

Figure 2. Logical Diagram of NCAR MSS

The key hardware components are the client systems, the high-speed data paths and switches, the storage devices (disk arrays, tape drives), tape media, the tape libraries and the Mass Store Control Processor (which runs some of the software that manages the MSS). This hardware is managed and controlled by numerous software components.

The first step in modeling a component (hardware or software) was to develop a conceptual model by identifying the important operations for that component and determining how any time delays associated with the operations would be modeled. Once a conceptual model was built, the model for that component was implemented in the simulator software. The important operations that a component performs were identified from multiple sources, including other NCAR group members for in-house developed software components and vendors for hardware devices. Fortunately, many operations often can be ignored (e.g., error recovery from infrequently occurring errors that are known to have a minimal impact on system performance). We modeled our components with the least amount of detail that we thought would be consistent with the desired accuracy, knowing that we could add more detail later if we needed to improve the accuracy.

Once the key operations of a component were identified and conceptually modeled, the time delays associated with the operations were modeled. Examples of delays include the time to mount a tape and the time to position a tape for reading. We modeled delays in two different ways, depending on the operation being simulated.

The first approach was a deterministic approach that was used where we could accurately calculate the delay for an operation at any given point in time based on the state of the system or when the delay value is known to have a small effect on performance. In general, this calculation could be a function of any of the simulator state variables. In practice, for our system, this function is usually a constant value since the time to perform some operations is actually constant or nearly constant. For example, the time to load a particular type of tape on a particular type of tape drive is very close to a constant value that can be measured or obtained from a vendor.

The second approach was a probabilistic approach. Some delays may be too difficult to determine accurately at a given point in time and may have a relatively large effect on performance. In those cases, a probability distribution can be useful. For example, the time to position a tape varies quite a bit and can be a relatively long time. If one modeled the position of files on a tape, the position time could be calculated deterministically. However, we did not want to model the system at that low level of detail. Instead, we use a probability distribution to model those delays. Other delays in the system are also modeled using probability distributions. The probability distributions that we use at this point are empirically derived from system logs. We do not use the analytic distributions (e.g., exponential) since our empirically derived ones appeared to capture the behavior more accurately. Below is a description of how the major delay parameters in the simulator components are modeled.

Software Delays in executing software (for example, updating a database entry) were modeled using constant values. When simulating our current actual system, the constant values were obtained from measurements.

Tape library Delays for tape mounts, dismounts and pass-throughs (to other libraries) were modeled using constant values. When simulating our current actual system, the values were obtained from the vendor.

Tape drive Delays for tape loads and unloads were modeled using constant values and, when simulating our current actual system, the values used were obtained from the vendor and measurements. Tape read positioning and data transfer delays were modeled using probability distributions. When simulating our current actual system, the distributions were derived from measurements and log records. However, for older tape drives where log records did not contain positioning or transfer time information and for other positioning times, constant average values were used.

Disk subsystem Delays for data transfer were modeled using probability distributions that were created from measurements and log records.

Job arrivals Delays between user job arrivals were calculated from the traces of user read and write events.

Delay parameters for a simulator run are specified in a primitive ASCII configuration file along with other parameters for a run such as the number of tape drives of each type to use, size of disk caches, and internal migration parameters. The configuration file makes it fairly easy to simulate different configurations. However, to simulate an entirely different Mass Storage System, we would need to replace the code in the simulator that simulates the behavior of the NCAR MSS software components with code that simulates the behavior of the other MSS software components. The workload for the simulator, as stated earlier, is derived from the actual MSS logs. The simulator is run on an IBM Power4 host and takes about 8 wall clock hours and 2 GB of memory to simulate 1 month's worth of MSS activity. Figure 3 shows a high level view of how the simulator operates.

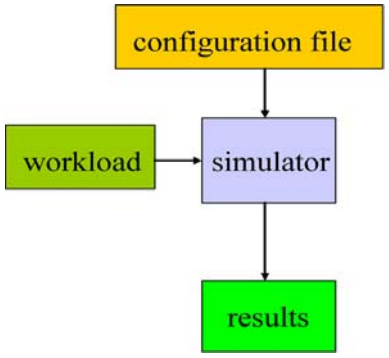

Figure 3. Simulator Operation

3.3. Simulator Validation

Validation is an important part of the simulator project. By validation, we mean checking that the simulator predictions closely match what would be observed with a real system. All components of the simulator were individually validated and the simulator as a whole was also validated. Dozens of validation runs were performed for simplified cases where exact answers are known. For example, we created simple workloads that contained only a few data transfer requests where we could calculate by hand the exact response times, device utilizations, etc. and compare the calculated values with the values generated by the simulator.

Validation runs were also performed for cases where the simulator was configured like the real MSS during some time period (e.g., same number of tape drives, same disk cache parameters, etc.). We used the trace of user read and write events from that same time period (usually a month long time period) to drive the simulator and we compared the results from the simulator with the actual results from the MSS for that time period. Based on the comparison, we would add more detail to one or more components to improve the accuracy of the simulator. Validation is definitely an on-going process and would be a part of any study that uses the simulator.

While our initial goal was to have enough detail so that all the metrics predicted by the simulator would be within 10% of the actual values, we found that it would require too much detail (and time) to reach that level of accuracy so we currently only attain an accuracy of 20%. Table 1 shows some validation results comparing simulator predicted values with the actual values for the month of July 2004. The workload used to drive the simulator was from the actual workload in June and July of 2004. So that the cache would be in steady state when the simulator statistics were collected, the simulator was driven with the two-month workload June-July. Once the June workload had been ingested, the statistics were reset so that they cover the one month July time period.

Table 1: Validation Results for July 2004

Metric	Actual value	Simulator predicted value	Relative error (%)
Disk cache read hit ratio (%)	66	66	0.0
Number of tape mounts	160,984	162,291	0.8
Mean overall user response time (s)	97	81	16.5
Mean write user response time (s)	105	84	20.0
Mean read user response time (s)	85	77	9.4

4. Disk Cache Study

One of the ways we used the simulator was to help us estimate the read hit ratios we would obtain with an expanded disk cache. Initially we had a small 180 GB disk cache and we allowed files with sizes up to 15 MB in the cache. One of the purposes of the cache is to handle reads that would otherwise have to come from tapes. The cache can be especially beneficial for reads of small files since the effective transfer rate (file size divided by the total time to read the file) of a small file from tape can be very low due to the time it takes to mount, load and position a tape.

When a user writes a file to the MSS, if the file's size is less than the maximum size allowed in the cache and its class of service parameters match the cache criteria (e.g., the file is not identified as a backup file that is unlikely to be read), it is written directly to disk. Files with sizes bigger than the maximum size or with class of service values that do not match the cache criteria are written directly to tape. Files are aged off of the cache to tape based on an LRU (least recently used) algorithm during a migration cycle that usually takes place once a day. Files that do not have a copy on disk, but that are read frequently enough and that match the size and class of service criteria will be copied back to the cache from tape so that subsequent reads will come from disk.

We used the simulator to study the possible benefits of expanding the size of the disk cache and the maximum size of files allowed in the cache. We also used the simulator to look at cache management algorithms. Read hit ratio was the primary metric we used in the study. Figure 4 is a plot of the predicted hit ratio from the simulator for different sized caches. The maximum allowed file size for the caches in the plot is 50 MB and the time period simulated was 01Aug03 through 31Oct03.

Figure 4 shows that the simulator predicts diminishing returns and that much of the read hit ratio benefit can be obtained with smaller cache sizes. Based in part on these simulator predictions, we chose an initial cache size of approximately 8 TB which the simulator predicted would provide a 67% hit ratio. The real cache was phased in over several months and was in steady state (i.e., the cache had filled up) by February 2004. Figure 5 shows the actual hit ratio for the cache

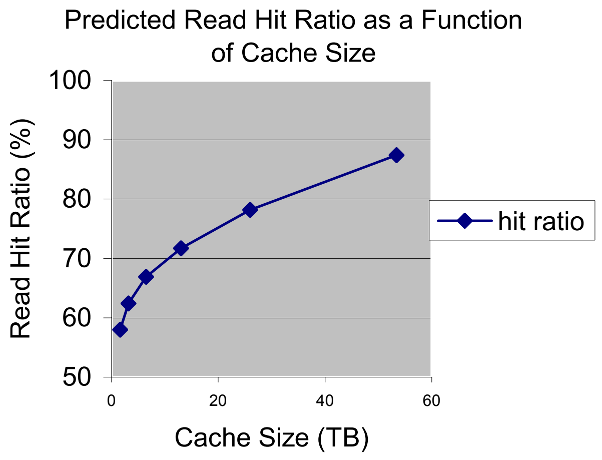

Figure 4. Predicted Hit Ratio

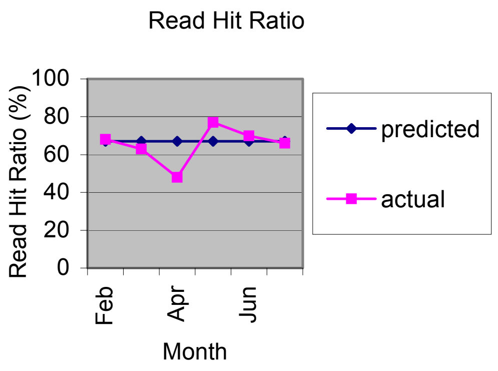

Figure 5. Actual and Predicted Hit Ratio

from February 2004 through July 2004 along with the line showing the predicted hit ratio of 67%. Data beyond July 2004 is not shown because the cache configuration changed after that point so the comparison between predicted and actual results is not valid beyond that month.

As part of this study, we also evaluated different cache management algorithms. As mentioned earlier, files that do not have a copy in the disk cache, but that are read frequently enough and match the size and class of service criteria will be migrated (copied) back to the cache. For example, if a small file was originally in the cache but was aged off due to inactivity and then later becomes active again (i.e., is reread frequently), it can be beneficial to copy the file back to the disk cache so that future reads come from disk rather than from tape. We looked at how to decide when files that are not in the disk cache but are read back should be copied to the cache. We tried copying each file that met the cache criteria to disk immediately following the first user read of the file. We also tried copying each file to disk immediately following the 2nd consecutive user read of the file within some time period (e.g., 24 hours or 1 week). We found that copying files to disk after the first read did improve the cache hit ratio moderately, but the vast majority of the files copied back were never read again before being aged off, so copying them to disk would be unnecessary overhead. The small improvement in the hit ratio did not justify the large overhead needed to achieve it. We found that copying a file to the cache after it is read twice within 24 hours seemed to be a good approach.

5. Tape Drive and Disk Cache Study

In addition to the disk cache study, we have also used the simulator to help us estimate mean user response time as function of both the number of STK 9940B tape drives and the disk cache configuration. The purpose of this study was to obtain some quantitative estimates of performance as a function of those parameters and to obtain some insight into the performance tradeoffs of disk and tape. The first part of this section provides more background on the configuration and behavior of the NCAR MSS, the next part describes the experiments that were conducted, and the final part presents the results and conclusions from the runs.

5.1. MSS Configuration and Behavior

At the start of these studies, our actual MSS consisted of an ~8 TB RAID disk cache that accepted files with sizes up to 50 MB, 38 STK 9940-A tape drives, 14 STK 9840-A tape drives, 5 STK 9310 Powderhorn libraries and a number of GigE and Hippi data paths. These tape drives and tape media were distributed among the 5 libraries. For this study, we kept the number of 9840-A and 9940-A tape drives constant for all runs and also kept the library and the data path configurations constant across all runs. We varied the configuration of the disk cache (size of the cache and maximum file size allowed in the cache) and the number of 9940-B tape drives.

As mentioned in section 4, when a user writes a file to the MSS, if the file attributes meet certain criteria (its size is below the maximum size allowed in the disk cache and its class of service attributes match the cache criteria), it is written directly to the disk cache. Files with attributes that do not match the criteria are written to tape. Files that are initially written to the disk cache are copied to tape approximately a day after they are written to disk and are removed from disk based on an

LRU (Least Recently Used) algorithm. This behavior was kept the same for all simulator runs for this study.

When a file is read by a user from the MSS, if the file resides on the disk cache, it is transferred from there; otherwise, the file is transferred from tape. If a file is not in the cache, is read twice within 24 hours and its attributes match the cache criteria, the system copies the file from tape to disk so that future reads can be serviced by the disk cache. This behavior was also kept the same for all simulated scenarios.

One attribute a user can specify for a file is the "reliability" (which can have the value "normal" or "economy"). If the user sets the reliability to "normal", two copies of the file are created when it is written to tape, which occurs when the user first writes the file (if the file is written initially to tape) or when the system creates a tape copy (if the file is written initially to disk). If the file is initially written to tape and two copies are to be created, the user write command does not return until both copies have been created whereas for a user write that goes to disk, the user only has to wait for a single disk copy to be created.

A disk cache can potentially help our MSS performance in several ways. First, reads from the cache are often faster than reads from tape due to the large positioning time for tapes. Second, writes to the disk cache can be faster than writes to tape if the disk subsystem aggregate performance is greater than the aggregate performance of the tape drives and also because, for files that a user has specified should be stored with "normal" reliability, the user only has to wait for a single copy to be written when the file goes to disk while they have to wait for two copies to be written when the file goes directly to tape. However, files written to disk are eventually written to tape (except for those that are rewritten or deleted before the tape copy is created) so the tape drives must be able to keep up with the writes to the disk cache or writes to the disk cache will block when the cache becomes full.

5.2. Description of Experiments

The simulation runs for this study used three month workloads derived from the actual workload of the system. The first two months were used to initialize the system and "warm-up" the disk cache so that it was in steady state. After the first two months, the statistics were reset and collected over the final month. Two different three month time periods (May2004-July2004 and Sep2004-Nov2004) were used so that we could see the effects of different workloads on the results.

The primary metric used to evaluate the different configurations was mean user response time. By response time, we mean the time from when the command to transfer a file to or from the MSS is initiated to when the command is complete and exits. We chose this metric since, of the set of performance metrics, our users are probably most aware of response time. Also, only the response time for user-initiated transfers is included in our metric. The time for system-initiated transfers is not included in the metric, but they are simulated so they indirectly affect the response time for user transfers.

As mentioned above, the parameters that were varied for the runs were the number of 9940-B tape drives and the disk cache configuration. All other system parameters were kept constant. Each 9940-B had a maximum transfer rate of 30.0 MB/sec, load and unload times of 18.0 seconds, and position times that were based on probability distribution or constant averages. These values were kept the same for all runs; only the number of 9940-B drives was changed. Also, all 9940-B drives were in a single silo and most, but not all 9940-B media were in that same silo. The disk cache was assumed to have an aggregate transfer rate of 180.0 MB/second and that was kept constant for all runs; only the size of the cache and the maximum size of files allowed in the cache were varied. Table 2 below summarizes these configuration parameters.

Table 2: 9940-B and Disk Cache Parameters

Parameter	Value
9940-B load time	18.0 seconds
9940-B unload time	18.0 seconds
9940-B position times	Probability distributions and constant average values
9940-B max data transfer rate	30.0 MB/second
Disk cache aggregate data transfer rate	180.0 MB/second

The number of 9940-B tape drives that were tried was 20, 30, and 40. All writes to tape during the simulated time period were written using the 9940-B drives. The 9940-A and 9840-A drives were used for reads of data written prior to the start of the simulated workload. Three different disk cache configurations were tried:

- A 7.8 TB cache with a maximum file size limit of 50 MB
- A 36.1 TB cache with a maximum file size limit of 500 MB
- A 62.6 TB cache with no maximum file size limit

The disk cache configurations were chosen using the following steps. First, we chose the three maximum cached file sizes to try for the experiments (50 MB, 500 MB, and no limit). For each of those maximum file sizes, we ran some simulations using different sized caches to estimate read hit ratio as a function of cache size. Those values were plotted and the cache size that resulted in a good hit ratio and that was a reasonable size (the "knee" of the curve) was chosen as the cache size to use with that maximum file size limit.

5.3. Results and Discussion

Figures 6 and 7 are plots of mean user response time estimates (with standard deviations plotted as error bars) as a function of the number of 9940-B tape drives and the disk cache configurations for the May2004-July2004 and Sep2004-Nov2004 workloads, respectively. The simulations using both workloads suggest that 20 tape drives is too few since the response time would probably be too high, although caching larger files does improve the response time for the 20 tape drive case. For 30 tape drives, there appears to be a benefit to caching files up to about 500 MB, but not much benefit in caching files bigger than that. There seems to be little benefit to using 40 drives compared with 30 drives, at least for these workloads.

In general, the data suggest that having a balance of tape drives and disk cache is better than having a very large amount of one or the other. The plots also illustrate the non-linear behavior of response time. If the system were under configured, response time would potentially be much too large; if the system were over configured, there may be little performance benefit to having the additional hardware.

While we investigated the effect of different workloads on performance (May2004-July2004 and Sep2004-Nov2004), we did not investigate the effects of workload changes that might be induced by the system configuration changes, but we do discuss it a little more in Section 6. We also did not study the economic costs of the various scenarios, which likely would be an important consideration. It would also be interesting to experiment with different parameter values (constants and those derived from probability distributions) to estimate the effect those would have on the results.

6. Limitations

While the simulator has been a useful tool for us, it has some important limitations. One limitation is that the simulator cannot predict the future MSS workload.

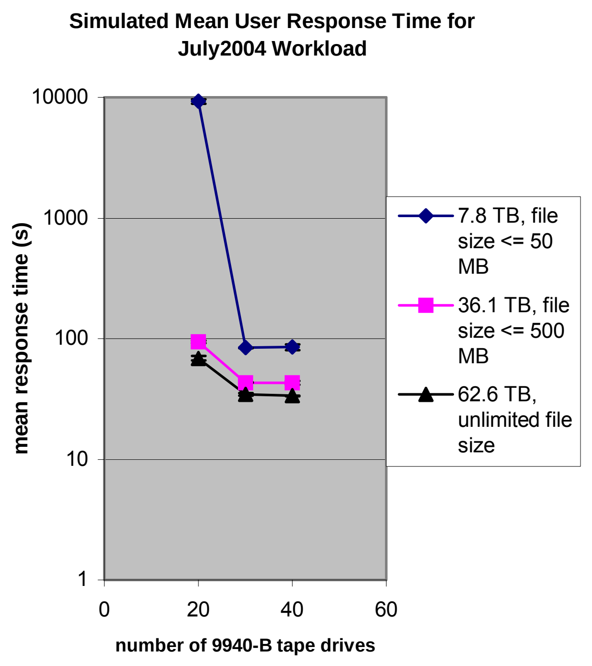

Figure 6. Estimated Response Time for July 2004 Workload

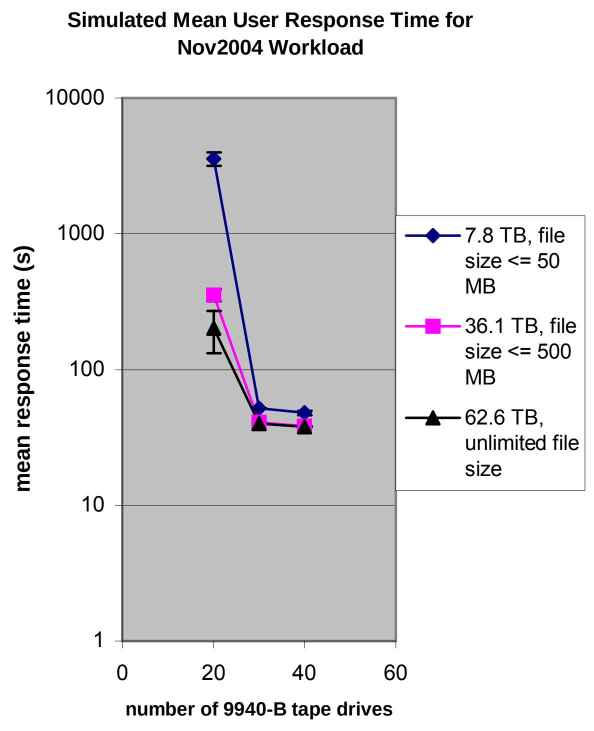

Figure 7. Estimated Response Time for Nov. 2004 Workload

Given the workload, it can predict the behavior of the system, but it cannot predict the workload itself. This limitation is important because the workload can change due to the addition of new supercomputers or due to changes in user behavior.

Changes to the workload can also be induced by MSS configuration changes. For example, adding additional hardware to improve the system performance could result in users submitting more requests (since the system is faster). However, all approaches to configuring and managing the MSS are limited in the same way. Whether we are doing simulations or back-of-the-envelope calculations or using some other approach, they are all limited by the inability to really know what the future workload will be. We do have the ability to create and ingest synthetic workloads in the simulator so if we can estimate what a future workload will be, we can use that to drive the simulator. Still, such workloads would only be estimates so we have to keep this limitation in mind.

An approach that one group has taken when simulating I/O subsystems is to build an emulator that can run applications that drive the simulator [11]. If one configuration of the system being simulated runs faster than another, the workload would automatically be scaled. However, given the large number of user applications that independently access our MSS, such an approach would not be feasible for us.

In addition to the errors due to workload uncertainty, our metrics are only averages and, based on validation runs, are only within 20% of the actual values. Also, while all the major components of the MSS have been taken into account, there are some components that have not been included in the simulator, and leaving them out diminishes the validity of our results.

7. Summary

We have developed a trace-driven performance simulator of our MSS to aid us in ranking design and configuration alternatives. Validation runs have shown that the simulator is able to predict average metrics within 20% of the true values. The simulator reads in a configuration file and ingests a workload derived from the actual historical workload of our MSS and generates statistics. The simulator has aided us in enhancing our disk cache and we have also used it to help estimate user response time as a function of number of tape drives and disk cache size. We found that, for files under 50 MB, an 8 TB cache provided a read hit ratio of a little over 60%. We also found that caching files above 500 MB may not be worth doing.

References

[1] Reagan Moore, Joseph Lopez, Charles Lofton, Wayne Schroeder, George Kremenek. Configuring and Tuning Archival Storage Systems. In *Proceedings of the 16th IEEE Mass Storage Systems Symposium*, pages 158-168, 1999.

[2] Raj Jain, *The Art of Computer Systems Performance Analysis.* John Wiley and Sons, Inc., 1991.

[3] John Wilkes, The Pantheon storage-system simulator. *Technical Report HPL—SSP—95—14.* Storage Systems Program, Hewlett-Packard Laboratories, Palo Alto, CA, December 1995.

[4] G. Ganger, B. Worthington, and Y. Patt. The DiskSim Simulation Environment Version 1.0 Reference Manual. *Technical Report CSE-TR-358-98*, Department of Electrical Engineering and Computer Science, University of Michigan, February 1998.

[5] Ann L. Drapeau and Randy H. Katz. Striping in Large Tape Libraries. In *Proceedings of the 1993 ACM/IEEE confererence on Supercomputing*, pages 378-387.

[6] J. J Bedet, L. Bodden, A. Dwyer, PC Hariharan, J. Berbert, B. Kobler, P. Pease. Simulation of a Data Archive and Distribution System at GSFC. In *Proceedings of the GSFC Mass Storage Systems and Technologies Conference*, 1993.

[7] Odysseas I. Pentakalos, Daniel A. Menasce, Milt Halem, Yelena Yesha. An Approximate Performance Model of a Unitree Mass Storage System. In *Proceedings of the 14th IEEE Mass Storage Systems Symposium*, pages 210-224, 1995.

[8] The University of Newcastle upon Tyne, UK. *javasim.ncl.ac.uk.*

[9] *SIMULA Standard.* Swedish Standard SS 63 61 14, SIS, Box 3295, Stockholm, Sweden, (1987).

[10] Sam Coleman and Steve Miller. Mass Storage Systems Reference Model: Version 4. *NASA GSFC Conference on Mass Storage Systems and Technologies*, Volume 1, 1992.

[11] John Linwood Griffin, Jiri Schindler, Steven W. Schlosser, John S. Bucy, and Gregory R. Ganger. Timing-accurate Storage Emulation. In *Proceedings of the Conference on File and Storage Technologies (FAST).* January 2004, Monterey, CA.

Acknowledgements

Thanks to Gene Harano, Mark Love, John Merrill, Craig Ruff, Erich Thanhardt and George Williams (my MSS colleagues at NCAR) for sharing their knowledge of the internals of various components of the NCAR MSS and to my managers, Gene and John, for giving me time to work on this project. Thanks also to my shepherd, Jean Bedet, and to the anonymous reviewers of this paper.