Measurements Over Distributed High Performance Computing And Storage Systems

Elizabeth Williams
Supercomputing Research Center
17100 Science Drive
Bowie, Maryland 20715-4300

Tom Myers
Department of Defense
9800 Savage Road
Ft. Meade, Maryland 20755-6000

1.0 Introduction

The rapid pace of technological change and the move toward "open systems" is making the process of acquiring systems much more complex. Traditionally, functional and performance requirements have been carefully described for systems to be acquired and the systems usually have come from a single vendor. The process worked as long as the requirements remained nearly static and systems changed slowly over their life time. There generally has been no need for a requirement to provide measurements and performance monitoring to see that requirements were met over the long term. Measurements that were available were often left over from development.

In the future the requirements for many systems are expected to change more quickly, and parts of the systems, acquired from multiple vendors, will evolve to meet those changing needs. There is a desire to ask for life-time measurements of systems in request for proposals (RFPs) when systems are being acquired. Thus, there is a need for measurements and performance monitoring as an integral part of the system to ensure that requirements are met over the long term after acceptance.

This paper gives a strawman proposal for a framework for presenting a common set of metrics for supercomputers, workstations, file servers, mass storage systems, and the networks that interconnect them. Production control and database systems are also included. Though other applications and third party software systems are not addressed, it is important to measure them as well.

The capability to integrate measurements from all these components from different vendors, and from the third party software systems has been recognized and there are efforts to standardize a framework to do this. The measurement activity falls into the domain of management standards. Standards work is ongoing for Open Systems Interconnection (OSI) systems management; AT&T, Digital and Hewlett-Packard are developing management systems based on this architecture even though it is not finished. Other efforts include the Storage System Management Sub-committee of the Mass Storage System Working Group and the UNIX International Performance Management

Working Group [1]. In addition, there are the Open Systems Foundation's Distributed Management Environment and the Object Management Group. A paper comparing the OSI systems management model and the Object Management Group model has been written [2]. Though most of the standards effort has been on the mechanisms for gathering and reporting measurements, we expect to cooperate with these standards making efforts. The work reported here is ongoing.

The IBM world has had a capability for measurement for various IBM systems since the 1970's and different vendors have been able to develop tools for analyzing and viewing these measurements. Since IBM was the only vendor, the user groups were able to lobby IBM for the kinds of measurements needed. However, in the UNIX world of multiple vendors, a common set of measurements will not be as easy to get.

In this paper we distinguish between metric and measurement. A **measurement** is a quantity that is directly measured while a **metric** is a quantity that can be derived from a set of measurements. Our focus is on using low level vendor specific measurements to support a set of higher level metrics that are common across a variety of vendors. The set of measurements to support the common metrics should in general be the minimum that is provided. Most systems should also make available measurements of unique aspects of the system that are not covered by the common set. For example, measurements on vectorization and hit ratios for memory hierarchies may not be in the common set of metrics but such measurements are desired.

2.0 Uses for Measurements

Measurements of systems are, of course, useful in many other ways than just to support system acquisition. They can be used to support day-to-day operations, management decisions and planning, and performance monitoring. The following are seven types of uses we have identified:

(1) distributed computing system scheduling,

(2) fire-fighting - solve immediate problems to provide acceptable response time and resource allocation to all processes,

(3) tuning systems for current workloads,

(4) capacity planning,

(5) allocating resources,

(6) looking for trends and characterizing workloads,

(7) verifying system strategies are working or assumptions about workloads are valid, e.g. locality of reference,

(8) validating accounting reports.

In analyzing how measurements are used, the following three points are very important. (1) For fire-fighting and tuning, a systems administrator must be able to **link** a particular "event" to a set of user commands.The systems administrator should be able to know when a resource is responding slowly and which process is causing the problem. We stress that it is important to be able to link particular events of interest back to user commands though we know that it is sometimes difficult. (2) Process as well as system-wide measurements are needed. (3) Accurate time stamps or

other timing information is necessary so that various independent measurements can be correlated with each other as a system is observed over time.

3.0 Measurement Collection Techniques

It is also understood that taking measurements and collecting them cause overhead and may in extreme cases affect the performance of the systems measured; this is not specifically addressed in this paper. However, data can be collected at various levels of detail depending on how much overhead is involved. The most complete level of measurement is a **log** or **trace** of each transaction or event. The next level of measurement is a set of counters that produce a histogram, which is an approximation to the distribution, of the metric of interest. The least detailed level of measurement is a simple counter from which the average, variance, maximum and minimum of the metric of interest can be derived. The level of measurement for any component depends on the overhead associated with the workload. When possible, the ability to selectively choose a different measurement level allows users of a system to manage how much overhead is given to measurement activities. Another way of managing the overhead associated with measuring a system is to sample a measurement at some interval that is frequent enough to observe interesting behavior but with reduced overhead. The sampling rate should be adjustable.

For measurements to be useful, they must be well documented It must be clear exactly what is being measured. The documentation should specify how much overhead is involved, what technique is being used to generate the measurement, and if there are user selectable parameters such as a sample rate or an enable/disable switch. Information about how a system is configured must either be statically defined or recorded along with a set of measurements.

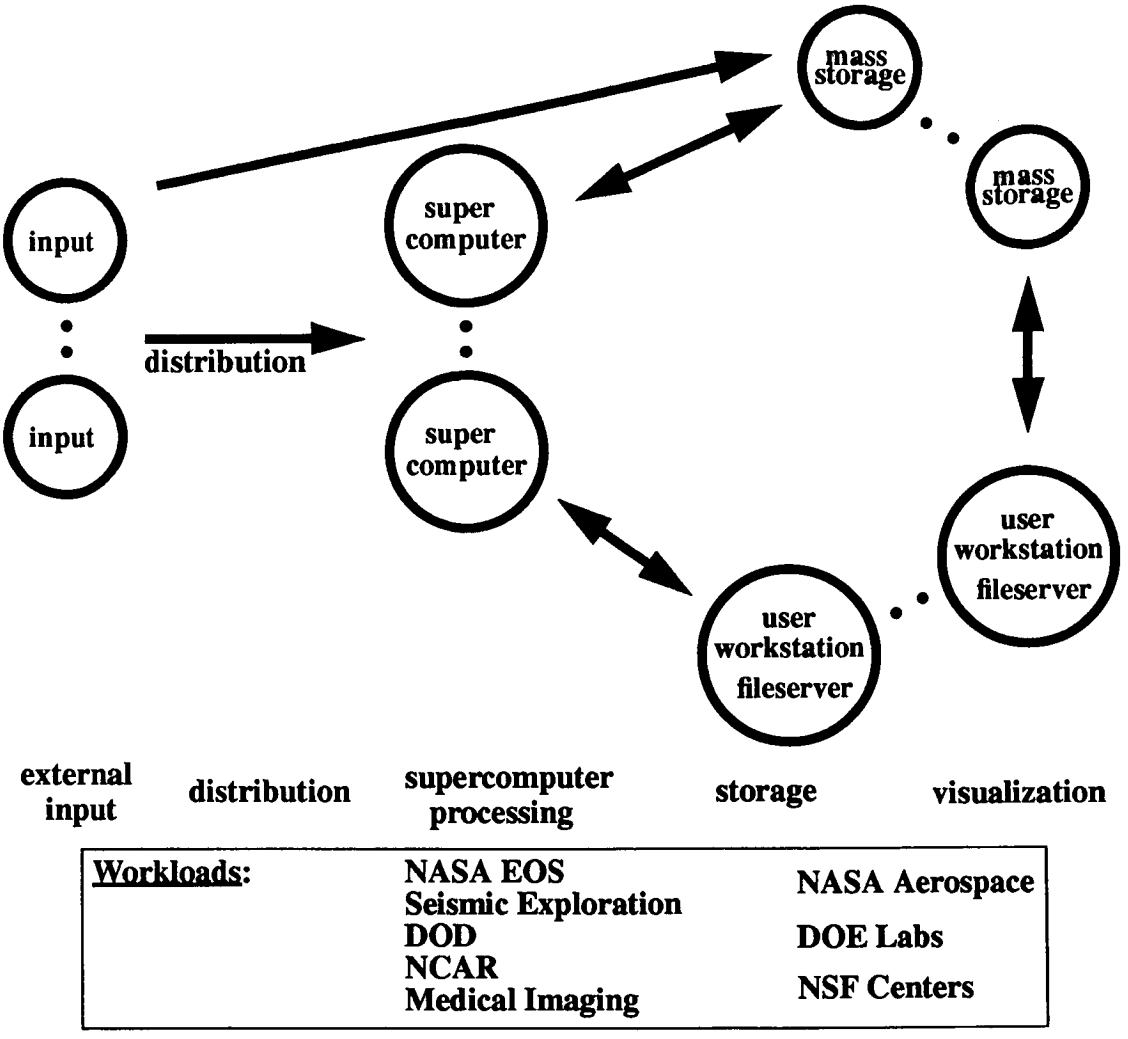

Figure 1: Model of Network Computing System

4.0 Model of Distributed High Performance Computing Systems

In Figure 1 we present a model of a distributed high performance computing system. The model identifies the five highest level functions of **external input** sources to indicate the collection of data for processing in the system, **distribution** for the network among components, **supercomputer processing** for high performance computing, **storage** for distributed mass storage, and **visualization** for user support processing. The distributed characteristics of this model are not depicted specifically but one can think of NASA's EOS system as the basis for this model. The other high performance computing systems listed at the bottom of the figure will all have similar models.

The five model functions are made up of various hardware and software system components. The hardware system components include supercomputers, mainframes, workstations, mass storage devices, file servers, networks, input machines and other network devices such as disk arrays. The

software system components include operating systems (**OS**) (includes file system and protocols), mass storage systems (**MSS**), database systems (**DBMS**), production control systems, third party software and user applications. Below the system component level are lower level building blocks to measure. These are the hardware building blocks such as CPU, memory, memory interconnection, disk, tape, terminal I/O, recorder/drive, robotics box, channel/controller, network interface, router and external I/O. The software building blocks are dependent on the particular system software component. For an operating system there are process management (scheduler/queues, context switches), I/O system (buffers, cache, queues), memory management (allocation, swapping, queues, paging, caches), file system, protocols, interprocess communications and other operating system services. For a mass storage system there is each module in the Mass Storage Reference Model (**MSRM**). For an application there are user defined modules, operating system components and various hardware building blocks used by the application. For a database there are indexes, tables, stored procedures, logs, locks, transactions and users. The software building blocks are not yet completely identified.

Figure 2 illustrates this three level hierarchy of metrics. The abstract metrics at the base of the pyramid are a list of generic metrics that are used at all three levels. The eye represents the need to have comprehensive and uniform observations at all levels.

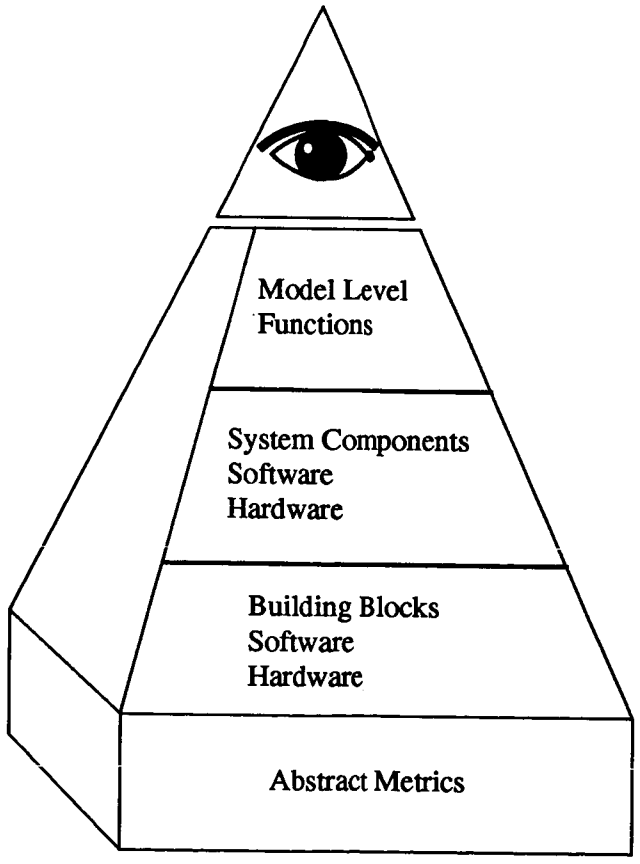

Figure 2. Hierarchical Levels

In Figure 3, the levels are expanded to show how higher level objects are composed of lower level objects. Three model functions are connected to system components and some of the system components are connected to lists of building blocks. Space does not permit a full expansion of all functions and components in this figure.

The common metrics for objects at the lowest level are derived from vendor measurements. And the metrics for higher level objects are derived from combinations of metrics for lower level objects. Thus we expect that the derivation of the lowest level metrics will be vendor specific but that higher level metrics will be vendor independent. We have not dealt with the issue of how to attach derivations to the metrics. We have not evaluated the difficulty for vendors to implement measurements to support the framework, nor have we tried to gauge the relative importance of the various metrics.

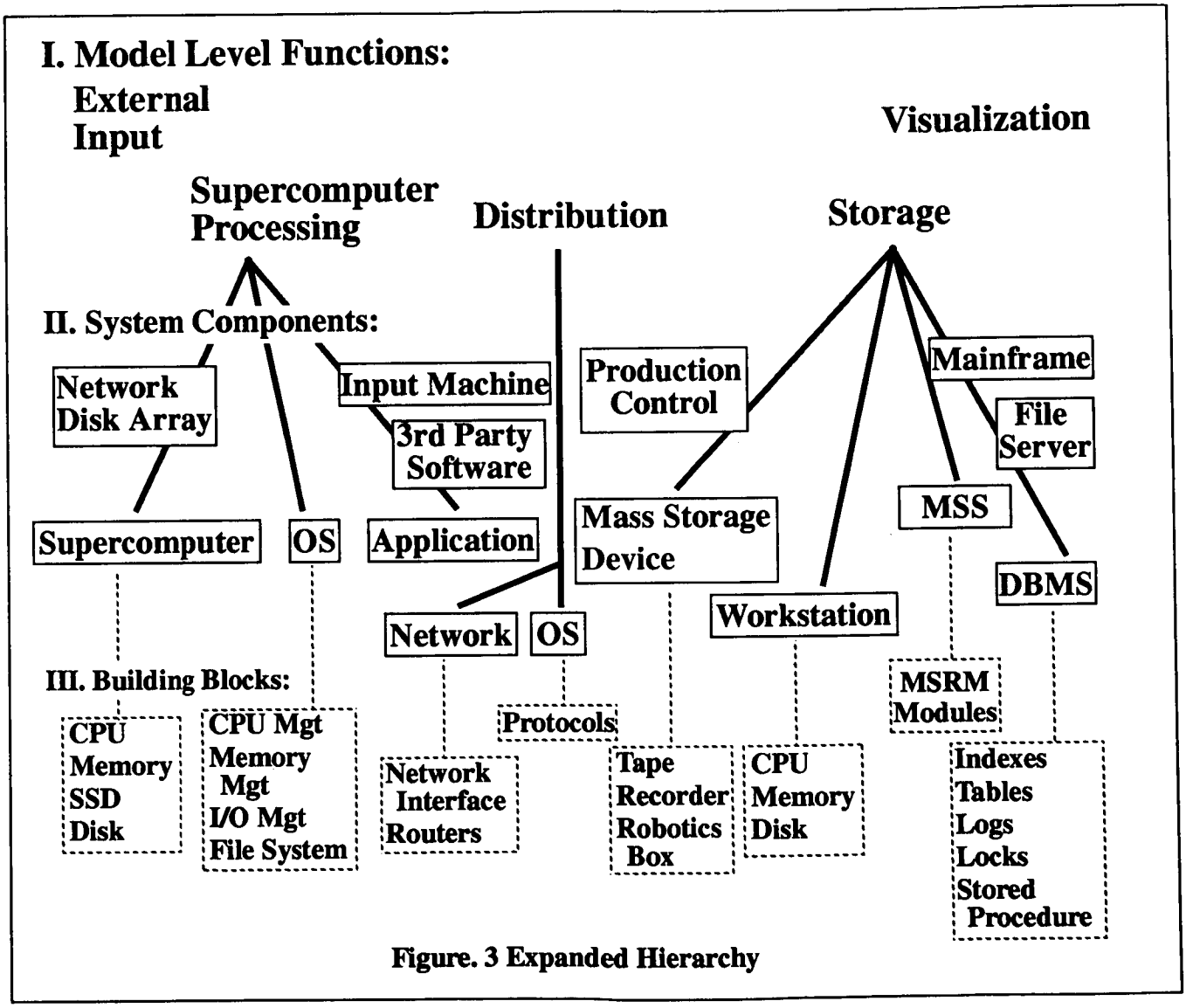

Figure. 3 Expanded Hierarchy

The use of common metrics is intended to provide vendors and other system developers a framework that can be used to design measurements as an integral part of the system that they deliver.

Too often measurements are used only to verify that a system is operating correctly and are insufficient for understanding the performance of the system especially when it is interconnected as a component of a larger system.

5.0 Abstract Metrics

The following list of abstract metrics are used to observe any **object** in the hierarchy by specifying an instance of the metric for the object:

1 Utilization, Capacity, Idle

2 Throughput

3 Response Time[1], Delay, Expansion Factor[2]

4 Waiting Time

5 Service Time

   a. Bitfile Size, Packet Size, Computation Requirement

   b. Speed of Device

6 Queue Length

7 Number of Jobs, Bitfiles, Packets

8 Routing, Branching Probabilities for Jobs Paths, Reuse, Age

9 Hit Ratios, Effectiveness of Strategies
(data migration, locality of reference)

10 Error Rates

All of these metrics are commonly used except for 8 and 9. Branching probabilities are useful for modeling systems.

At the bottom of the hierarchy the specific metric for each object is given in terms of characteristics of the object, such as mips and mflops for CPU throughput metrics. In addition at higher levels, users will want to specify metrics in terms of the workload of the system, e.g. satellite images processed per second through all model level functions for the NASA EOS.

6.0 Metric Tables

The following pages contain tables of metrics for the objects within the hierarchy. The left hand column in each table has the list of abstract metrics. The other columns have instances of the corresponding metric for the object at the top of the column. The tables are generally sparse since this

---

1. Response Time = Service Time + Wait Time + Other Time

2. Expansion Factor: wall clock time in shared system / wall clock time in dedicated system, which often can be approximated by wall clock time in shared system / CPU time

work is still ongoing and we invite help in completing the tables, adding more objects to the hierarchy and adding more abstract metrics.

abstract metric	Processing and Storage	Processing and Input	Input and Storage	Input, Processing, Storage
utilization, capacity, idle				
throughput	Mops per bit stored Mops per bitfile stored	Mops per bit input	Input bits per bit stored	Bitfiles processed through all functions per second
response time, delay, expansion factor				Response time through all functional components
waiting time				
service time: job size, device speed				
queue length				
number of jobs				
routing paths, reuse, age, branching probabilities				Bitfile routes through all functional components
hit ratios, effectiveness of strategies				
error rates				

Table 1: Model Level - Across Functions

Table 1 has metrics for the overall system where the objects being observed are combinations of model level functions. At this level, the metric instances are suggestions since they will depend on what the system does and will be defined by the users of the system.

Tables 2, 3 and 4 have the metrics for storage and some of its lower level components and building blocks. Tables 5 through 9 have the metrics for supercomputer processing and some of its lower level objects. Table 10 has the metrics for distribution (networks) and some of its lower level objects

In conclusion, we have presented a strawman proposal for a framework for presenting a common set of metrics across many systems and we have listed some of the metrics. This work is ongoing and we invite participation from users, vendors and system developers.

abstract metric	storage	mass storage device	mass storage reference model
utilization, capacity, idle	% space used # B[a], # O, # M total by class[b] or storage device	% space used % fragmentation	# bitfiles by class bitfiles/media bits/bitfile by class
throughput	{B O M}/sec access[c] by class or storage device	bits/sec accessed media/sec accessed	bitfiles/sec accessed
response time, delay, expansion factor	{B O M} response time by class or storage device or overall	{B M} response time by class	Bitfile response time by class
waiting time	*[d]		*
service time: job size, device speed	*		*
queue length			length at various model modules
number of jobs			
routing paths, reuse, age, branching probabilities	# accesses vs. storage device {B O M} vs. age vs. # accesses	#media vs.#accesses #media vs. # age #media vs. age vs. # accesses	# bitfile vs. # accesses # bitfile vs. # age # bitfile vs. age vs. # accesses
hit ratios, effectiveness of strategies			migration policy metric hit ratios
error rates	BER overall, by device failure by device	BER, failures	

Table 2: Storage - System ComponentS

a. B= Bits; O = Bitfiles; M = Media
b. class = {media type, bitfile size, access type, user, user process, user defined}
c. accesses = reads, writes, deletes
d. Asterisk implies that the metric is the obvious one in this context.

System Components not included in this configuration:

1. workstation or mainframe for controlling mass storage device
2. database for meta-data about the stored bitfiles

abstract metric	tape	recorder, tape drive	disk arm/ platters	robot
utilization, capacity, idle	% space used/tape % free tapes	% time reading % time writing % time scanning % idle	% time reading % time writing % time seeking % free space or fragmented (int/ext) for platters	% time in use
throughput		bits read/sec bits write/sec mounts/sec	bits read/sec bits write/sec seeks/sec	# requests/sec
response time[a], delay, expansion factor		includes (positioning) start, stop, scan, read/write delays	includes read/ write, seek, rotation delays	includes start, stop (positioning) delays
waiting time				
service time: job size, device speed				
queue length				
number of jobs				
routing paths, reuse, age, branching probabilities	# tapes vs. # accesses[b] # tapes vs. age # tapes vs. age vs. # accesses			
hit ratios, effectiveness of strategies			arm movement distance/seek	distance/request
error rates	BER each tape BER for all tapes	failures/time int.	failures/int. BER for platters	robot failures/int.

Table 3: Storage - Building Blocks - Hardware

a. response time = service time + waiting time + other factors associated with using resource
b. accesses = reads, writes, deletes

abstract metric	physical volume repository	bit file mover	storage server	bit file server
utilization, capacity, idle	% time in use			
throughput	bitfiles/sec accessed	bitfiles/sec accessed	requests/sec	requests/sec
response time, delay, expansion factor	*[a]	*	*	*
waiting time	*	*	*	*
service time: job size, device speed	*	*	*	*
queue length	*	*	*	*
number of jobs		*	*	*
routing paths, reuse, age, branching probabilities			# {B O} vs. age vs. size vs. accesses	
hit ratios, effectiveness of strategies			migration/caching policy	
error rates				

Table 4: Storage - Building Blocks - Software

a. Asterisk implies that the metric is the obvious one in this context.

abstract metric	Supercomputer Processing	Supercomputer	Operating System	Application CPU, mem, IO
utilization, capacity, idle	% to users % to system % to idle	% to users % to system % to idle	% to system % holding on locks	% to application % to system vectorization speedup
throughput		mops, mips mflops	processes/ sec system calls/sec interrupts/sec - all by class	mflops particles/sec
response time, delay, expansion factor			response time for all processes expansion factor for all processes	response time for application
waiting time			waiting time for all processes	
service time: job size, device speed			CPU burst time vs. memory size	CPU time memory size logical reads, writes
queue length				
number of jobs				
routing paths, reuse, age, branching probabilities			process path probabilities for I/O devices	
hit ratios, effectiveness of strategies				page hit ratio swaps system calls
error rates				

Table 5: Supercomputer Processing - System Components

abstract metric	CPU	Memory	SSD[a]	disk arm/ platters
utilization, capacity, idle	% time issuing inst % time holding issue % time vect or para % vector {ops,inst} vector length	% time issuing read or write (% free space or fragmented)	% time issuing read or write (% free space or fragmented)	% time reading % time writing % time seeking % free space or fragmented (int/ ext) for platters
throughput	ops/inst mops, mips mflops	{Bytes, Words} read/s, write/s by type	reads/sec writes/sec	bits read/sec bits write/sec seeks/sec
response time, delay, expansion factor				
waiting time	% time waiting on functional units	waiting time/ref hold issue/ref contention/ref		
service time: job size, device speed	hardware specified	hardware specified	hardware specified	includes read/ write, seek, rotation delays
queue length				
number of jobs				
routing paths, reuse, age, branching probabilities	instruction mix: % {ops, inst} by instr class			
hit ratios, effectiveness of strategies	instr cache hit ratio memory cache hit ratio	page hit ratio	device cache hit ratio	arm movement distance/seek
error rates				failures/interval BER for platters

Table 6: Supercomputer - Building Blocks - Hardware

a. SSD = solid state device

abstract metric	Channel/Controller	Terminal I/O
utilization, capacity, idle	% time busy	
throughput	bits/sec by device[a] channel ops/sec	characters/sec
response time, delay, expansion factor		
waiting time		
service time: job size, device speed		
queue length		
number of jobs		
routing paths, reuse, age, branching probabilities	bits vs. device	
hit ratios, effectiveness of strategies		
error rates		

Table 7: Supercomputer - Building Blocks - Hardware

a. device = {SSD, disk}

abstract metric	CPU Management	Memory Management	I/O System
utilization, capacity, idle	% time to user % time to idle % time to system	% space used % space fragmented	% buffer space used
throughput	context switches/sec by class processes/sec	allocations per sec swaps per second by memory size pages per sec	logical & physical read/ write per sec by bits, device
response time, delay, expansion factor			
waiting time	WT	WT	WT
service time: job size, device speed	CPU burst time per process	memory size by process memory residency time	time for service by logical, physical I/O
queue length	QL of CPU queue(s)	QL of Memory queue(s)	QL of device queues
number of jobs		# jobs in memory	
routing paths, reuse, age, branching probabilities			
hit ratios, effectiveness of strategies			I/O buffer hit ratio by read, write
error rates			

Table 8: Operating System - Building Blocks - Software

abstract metric	File System	Interprocess Communication	Other OS Services
utilization, capacity, idle	% used on each I/O device		
throughput	operations/s by class		
response time, delay, expansion factor			
waiting time			
service time: job size, device speed			
queue length			
number of jobs			
routing paths, reuse, age, branching probabilities			
hit ratios, effective- ness of strategies			
error rates			

Table 9: Operating System - Building Blocks - Software

abstract metric	Distribution	Networks	Operating System: Protocols	Routers/ Network Interfaces
utilization, capacity, idle		% time used bits/packet	bits/object packets/object	
throughput	bits/s bits/s vs. path objects/s objects/s vs. path by class[a]	bits/s packets/s by class	bits/s pkt/s objects/s by class	
response time, delay, expansion factor	by class and object size	by class and object size	by class and object size	
waiting time	by class and object size	by class and object size	by class and object size	
service time: job size, device speed	by class and object size	by class and object size	by class and object size	
queue length		send/receive queues	send/receive queues	
number of jobs				
routing paths, reuse, age, branching probabilities	relative use of paths			
hit ratios, effectiveness of strategies		collisions/packet		
error rates	BER, failures	retrans/sec	timeouts failures	

Table 10: Distribution - System Components

a. class = {protocol used, path, user, process, send/receive}

7.0 References

[1] Leon Traister and Terry Flynn, "A Measurement Architecture for Unix-Based Systems", CMG Transactions, Winter, 1991, pp. 69-77.

[2] Peggy Quinn and George Preoteasa, "Reconciling Object Models for Systems and Network Management", Technical Report, UNIX System Laboratories, Inc.