File System Benchmarks, Then, Now, and Tomorrow

Thomas M. Ruwart
Ciprico, Inc.
I/O Performance, Inc.
Minneapolis, Minnesota
tmruwart@ciprico.com
Ph: +1.763.551.4000

Abstract

With the growing popularity of storage area networks (SANs) and clustered, shared file systems, the file system is becoming a distinct and critical part of a system environment. Because the file system mitigates access to data on a mass storage subsystem, it has certain behavioral and functional characteristics that affect I/O *performance* from an application and/or system point of view. Measuring file system performance is significantly more complicated than that of the underlying disk subsystem because of the many types of higher-level operations that can be performed (allocations, deletions, directory searches, ...etc.). The tasks of measuring and characterizing the performance of a file system is further complicated by SANs and emerging clustering technologies that add a *distributed* aspect to the file systems themselves. Similarly, as the cluster/SAN grows in size, so does the task of performance measurement. The objective of this study is to identify some of the more significant issues involved with file system benchmarking in a highly scalable clustered environment.

Introduction

This paper discusses an approach on measuring file system performance in a large-scale, clustered supercomputer environment – more specifically the 30TeraOp supercomputer being built at Los Alamos National Laboratories (LANL) during the latter half of 2001 as part of the Accelerated Strategic Computing Initiative (ASCI). This supercomputer will have approximately 600 Terabytes of disk storage connected to a cluster of over 11,000 processors running a UNIX variant and a multi-layer distributed file system. This environment stresses the capabilities of a file system benchmark program and demonstrates what happens to file system I/O as the system size scales to very large sizes.

At the outset of this study it was necessary to examine the capabilities of the various file system benchmarks currently produced and/or used by commercial organizations. These include old favorites such as Bonnie and IOZONE as well as the SPECFS, PostMark, AIM Suite VII, LMBench, Hbench, and IOMeter[1]. It was determined quickly that none of these benchmarks had the capability to meet the requirements dictated by the architecture of the 30Terop machine. It was also apparent that none of these benchmarks could, for one reason or another, be easily modified to operate effectively in the targeted environment. Therefore, the objective of this study is to determine how to design a benchmark suite that can be used to gauge the bandwidth and transaction performance of a highly distributed, hierarchical, parallel file system and the underlying I/O and storage subsystem.

This paper summarizes the high-level design issues involved in the development of a program suite that can benchmark the storage and I/O subsystems as well as the file system hierarchy above them. This paper begins with a description of the overall hardware infrastructure of the 30TeraOp machine. This is followed by descriptions of the I/O Hierarchy and File System Performance. A short discussion of Benchmarking versus Characterization as well as the Perspective from which performance is view is presented. Other various issues that affect performance results such as Caching effects and File System Aging are briefly described in order to keep them visible. The remainder of the paper describes some of the logistics of managing workload generators, collecting performance data, and analyzing and generating reports.

Hardware Infrastructure Issues

The task of gauging the performance of a file system used to be relatively simple when the files systems were all local to a computer system. With the advent of physically shared disk subsystems however, measuring the performance of a file system is becoming significantly more complex. This is due to the fact that the number of measurement points has increased from one computer system to many computer systems and that they share access to the disk subsystem, or more importantly the *data*, at many levels.

Furthermore, they can share other parts of the disk subsystem such as controllers, paths, switches, …etc. or they can have entirely separate connections but still share the media or data.

In order to make sense of any application-level benchmark suite, the performance characteristics of the underlying hardware infrastructure must be separated from the performance characteristics of the file system. It is also critical to understand how the file system interacts with the underlying hardware infrastructure. In the case of the 30TeraOp machine, there are several pieces of the hardware architecture that need to be considered:

- Compute nodes
- System Area Network or Cluster Fabric
- I/O Nodes
- Storage Area Network or Storage Fabric
- Storage Devices

Each of these pieces has certain functional and performance limits that need to be evaluated and understood in isolation as well as when they operate as a system in their final configuration. For example, it is necessary to determine the performance characteristics of a single disk array on a single I/O node and then scale the number of disk arrays up to the configuration maximum to determine any performance anomalies that result from such scaling. A similar exercise is required for evaluating the performance scaling of the Storage Area Network as I/O nodes and disk arrays are added to its fabric. And finally, the performance of the compute nodes and the System Area Network needs to be evaluated as the number of compute-nodes accessing I/O nodes scales.

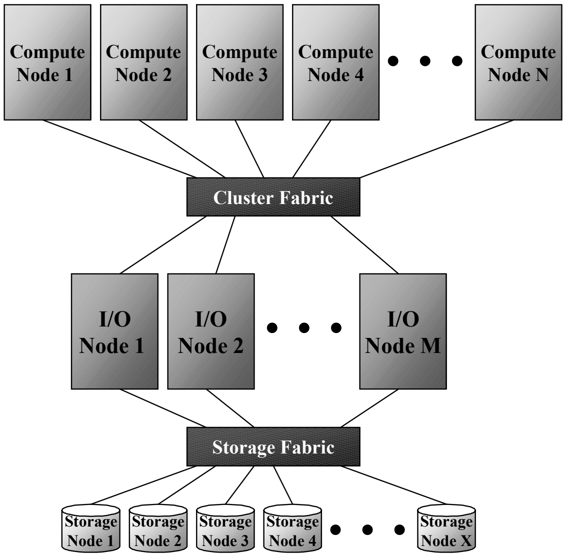


Diagram 1. The generalized architecture of the 30TeraOp super computer cluster at Los Alamos National Labs. There are N Computer Nodes tightly coupled through a cluster fabric to M I/O nodes that are connected to X storage nodes through a Storage Fabric.

The I/O Hierarchy

First, it is necessary to make a distinction between the "file system hierarchy" and the underlying I/O subsystem. In general, a file system is essentially an *application* of sorts that issues I/O requests to the underlying I/O subsystem on behalf of the user application. However, in the architecture of the 30TeraOp machine with the hierarchy of multiple file systems this distinction becomes blurred. The user application interfaces with the top-level file system that, in turn, interfaces with another file system below it instead of the storage subsystem. This "lower level" file system then interfaces with yet another file system below it and then finally to the actual storage subsystem hardware (see Diagram 1).

The file systems within this hierarchy are different from one another in functional and performance characteristics. The top-level file system is a "parallel" file system in that it understands how to have multiple threads of a single process concurrently accessing different parts of a single file. In general this works fine for a small number of threads, say less than 100. However, on the 30TeraOp machine, the number of threads trying to access a single file could be 10,000-100,000. Imagine the start-up time for 10,000 threads each issuing an open for a single file across several hundred separate compute nodes (the M to N problem). Another scenario is that of a single compute node attempting to access all the 10,000 pieces of a single file through all the I/O nodes (the 1 to N problem). A third scenario is for all the compute nodes to access data from a single I/O node (the M to 1 problem). Given the possible number of computer nodes (M) being in the hundreds to thousands and the number of I/O nodes (N) being in the tens to hundreds, the number of possible scenarios suggests that a single benchmark run will not be sufficient to describe the performance and behavior of all the file system levels in this environment. Therefore, it is necessary to not just measure the performance of the file system from the application's point of view but to *characterize* it with detailed performance measurements of all the components directly related to an I/O operation. In other words, it is necessary to measure the performance of each of the file system levels in isolation (if possible) and then as a composite of all file system levels.

File system performance

File system performance in general can be broken up into two major categories: Meta-data and User data performance. Meta-data performance relates to operations such as opening, closing, creating, deleting, space allocation, and obtaining or updating the status or attributes of a file. These operations do not involve accessing any of the actual *user* data in the file. Meta-data itself includes all the data structures required to represent a file on a storage device (i.e. inodes, directories, super blocks, file access tables, …etc.) User data performance strictly involves reading or writing the actual data within the file.

The 30TeraOp machine incorporates a file system hierarchy consisting of several levels of file systems. Generally speaking, at the top of the hierarchy, there is a parallel-type file system that runs on the compute nodes that gives a "parallel" application the ability to have multiple threads simultaneously access different parts of a single file in parallel. Below the parallel file system is a file system that presents single consistent view of all the I/O nodes as a single file space. On each of the I/O nodes there is a file system that consolidates access to the storage under the control of each of the I/O nodes.

Therefore, when an application makes a request to open, close, read, or write a file in this environment, the request is propagated through each of these file system levels before it reaches the actual storage media. This can have a significant impact on both the meta-data and user data performance. This is one aspect of the 30TeraOp file system architecture that requires such a significant effort in benchmark suite design.

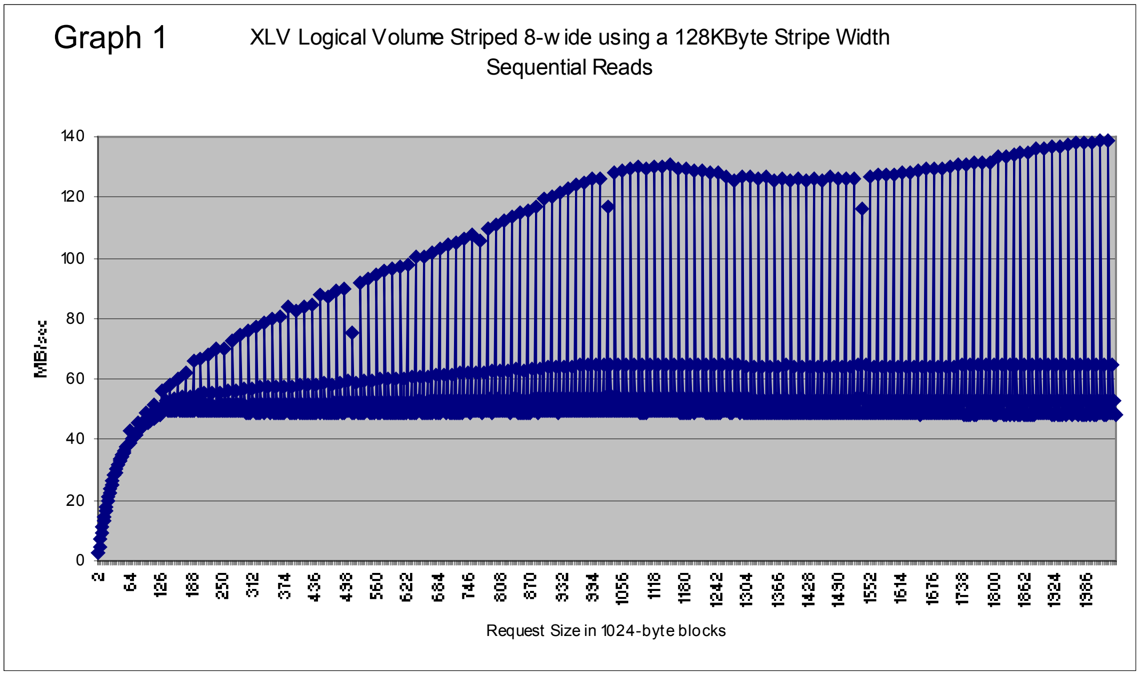
Benchmarks versus Characterization

File system benchmarks, in general, provide a single value or a limited number of values that are intended to represent the overall performance of the file system. However, this is somewhat like trying to reduce a vector to a scalar in the sense that a significant amount of important detail can and does get lost in the translation. In order to get an accurate assessment of the performance of a storage subsystem it is necessary to perform a benchmark over a range of values of specific parameters. The result of this is a performance profile or characterization as a function of some parameter.

There are many parameters that determine how a single I/O operation interacts with the computer system as a whole. It is this interaction that ultimately determines the performance of the I/O operation. Since many of these parameters are dynamic variables it is more useful to see the performance of the file system as a function of some of the more *significant* parameters. Significant parameters are those that have more of an impact on performance than others.

For example, the Request Size may have more of an impact on the bandwidth performance of the application than the disk location being accessed. This is because the "request size" that the application gives to the top-level file system to transfer data between the application memory buffer and the storage media goes through several changes and realignments before the request is received by the storage device(s). The application may, for instance, request a 16MB transfer. The file system may break this up into sixteen 1MB requests to the underlying logical device driver. The device driver may split the 1MB request into four 256KB requests each to one of four disk arrays. Finally, the four disk arrays then divide their 256KB requests into eight 32KB requests that are delivered to the disk drives within the disk array. Thus the 16MB request that the application made turned into many 32KB requests that operate both in parallel across all the disk drives as well as serially within the disks if the width of the logical device is smaller than the original 16MB request.

Because of the request size manipulation that occurs along the life cycle of an application data transfer request, the performance that the application sees depends heavily on the requested transfer size. One particular transfer size may perform well while other transfer sizes may have a dramatically different



performance level. Graph 1 demonstrates how dramatically the performance levels can change as a function of request size. This graph shows the bandwidth performance of an application reading from an 8-wide striped logical volume as a function of request sizes ranging from 2KB to 2MB in increments of 2K bytes. It is apparent that a small change in the request size results in a potentially large change in performance – at the right side of the graph the performance varies from 140MB/sec down to 45MB/sec with only a 2KB change in request size.

Perspectives

The *perspective* is the point of view from which the performance is measured. Three of the more generally accepted perspectives are:

1. Application
2. System
3. Storage Subsystem

The Application perspective is what of most of the file system benchmarks represent. From this perspective all of the underlying system services and hardware functions are hidden. This perspective includes the all the cumulative effects of other applications running at the same time as the benchmark run. This is also true for applications running on other machines that may be simultaneously accessing the storage subsystem under test. From this perspective the results of a benchmark can be skewed due to undesirable interactions from these other applications and other machines.

The Application perspective can also divide I/O operations into the two distinct categories (Meta data and User data) based on the type of higher-level operation being performed. The Application interface to the file system is generally through high-level system calls such as open, close, read, write, and create. There are also higher level system calls that perform such operations as rename, create directory, remove, and lookup a name. It is the performance of these operations that ultimately determine the overall performance that the Application sees for both meta data and user data operations.

The System perspective is viewed by running system-monitoring tools (such as *sar*, *osview, or filemon* for example) during a benchmark run. These tools provide coarse-grained real-time monitoring of the system I/O activity for such high-level operations as file reads and writes as well as the number of operations actually sent to the storage subsystem on a device-by-device basis. From this perspective it is possible to see and measure the effect of other applications that are running concurrently with the benchmark program. Furthermore, with some of the more sophisticated system monitoring tools, it is possible to monitor the activity on other systems that may be sharing access to the storage subsystem under test. However, there is still a problem with getting a complete view of all the systems on a common reference clock in order to better understand the *interaction* of all the systems with the shared storage subsystem.

The Storage Subsystem perspective is the most difficult to monitor since there are not many tools available to collect performance data from the storage subsystem. Furthermore, the storage subsystem can be split into two pieces: (1) the interconnect fabric and (2) the storage devices. Each of these pieces need to be monitored independently because they represent two areas of possible resource contention. The monitoring points for the fabric are the fabric switches since all the I/O traffic flows through these points. The storage subsystem can be monitored at the storage device controllers that connect into the fabric.

Given these three observation perspectives it is possible to sort out the affects of other parts of the system such as caching, disks, interconnect speeds, process contention, …etc.

Caching Effects

There are several levels at which caching is used to mitigate performance issues with the underlying storage layers. These include the file system buffer cache, disk array controller caches, and disk drive caches.

The file system with its buffer cache is on the top layer of the cache hierarchy. The file system buffer cache is generally some significant amount of physical system memory that is used to hold large chunks of data from files being accessed through a file system manager (i.e. UFS). For example, when an Application or benchmark program issues a read system call most file system managers will read data into the file system buffer cache and then copy the requested data into the user buffer. Similarly, for write operations, the data is copied from the user buffer into the file system buffer cache and later written to the storage media. For normal applications this is acceptable behavior but when running benchmarks it is necessary to understand when the cache is being used and when it is not. Otherwise, the results of the benchmark can be rendered meaningless.

The disk array controller and disk drives have separate caches that are not connected or controlled by the file system manager or the device drivers. It is under the control of the disk array or disk drive controller and there are many different control algorithms that determine how it is used and how effective it is. The configuration and usage modes of the cache are purely vendor-dependent and model specific. It is mentioned here because it is important to understand how the cache, if present, is being used during a benchmark run so that its effects can be taken into account when setting up the benchmark runtime parameters and/or interpreting the results.

File System Aging Effects

Over time, a file system that has had files repeatedly created and deleted can exhibit side effects related to this *aging* (fortunately one of them is not forgetfulness). A file system that ages in this way can get highly fragmented as is commonly found on FAT32 and NTFS file systems for example. On smaller file systems it is possible to "defragment" the file systems because the ratio of the size of the file system to the data rate of the underlying disk is generally quite low (on the order of 2000 for a standard 20GB disk). This means that it is possible to read and write all the data from the disk subsystem in order to rearrange it sufficiently to decrease the fragmentation.

On a very large-scale supercomputer system such as the 30TeraOp machine this is not practical because the ratio of the size of the file system to the achievable transfer rate of the underlying hardware is ten times that of the PC example. This means that it could take on the order of days to defragment the entire file system which could be a significant burden on the available resources. However, depending on the data layout and application accessing the files, this may or may not present a performance problem which makes defragmentation optional. Therefore, it is necessary to be able to test and/or monitor a file system's performance as it ages in order to determine when defragmentation is really necessary.

Workload Generator Control Mechanisms

When running multiple workload generators it is necessary to control their activities to achieve the desired behavior. For the single machine case the control mechanism is relatively simple and straight forward. In general, a single master control process passes high-level instructions and parameters to each of the workload generators on the machine, lets them run, collects aggregated results, and displays the performance information at the conclusion of the run. The same process is true for a cluster of machines but

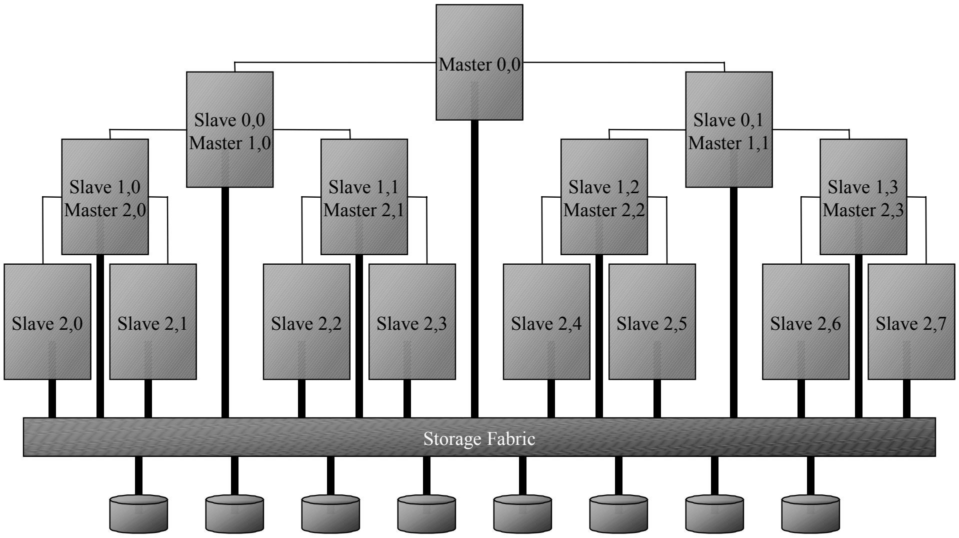


Diagram 2.

it is important to understand how the control process *scales* as the number of machines in the cluster running workload generators increases.

In the case of a large cluster, it may be necessary to manage several hundred to several thousand workload generators. The management process essentially involves starting the workload generators (and optionally terminating them) and collecting, correlating, and reporting the results at the conclusion of the run. This process can take up to several minutes for a large number of workload generators if the process is serialized. However, if the process is managed as a binary tree for example, it is possible to reduce this startup time from order N to order log(N). Diagram 2 shows how a Master manager starts a Master manager on each of two other machines which, in turn, start Master managers on each of four machines, and so on. This significantly decreases the resources needed on any single machine in the cluster to start up all the Master control processes in the cluster.

Unsynchronized/Synchronized Workload Generators

A workload generator is a program that generates I/O traffic according to a very specific set of parameters. The most important parameters that have a direct impact on the observed bandwidth or transaction performance of the individual workload generator include:

- Size of a transfer
- Number of transfers to generate
- Access pattern to use for the transfers
- Percentage of read versus write operations
- Temporal access patterns – bandwidth or transaction rate

Other parameters that can have an indirect impact on the observed performance include:

- Processor allocation
- Memory allocation parameters
- Process priority parameters
- Synchronization parameters

When running on a single isolated machine (as opposed to a cluster of machines), multiple concurrent workload generators can be synchronized with each other so as to achieve the correct overall *temporal* access patterns desired. A temporal access pattern can be described as the number of operations per unit time that are issued by the workload generator. The rate at which the operations are issued from a particular workload generator can be tied or synchronized to those issued by other workload generators. This synchronization can be done using several different methods but the use of semaphores, shared memory spaces, message-passing interfaces, and signals are some of the more common methods.

Furthermore, it is important to note that multiple concurrent workload generators on a single machine share a common *reference clock*. This allows for the cross correlation of the detailed time-stamp trace data that each of the workload generators can produce during a run. By cross-correlating these results it is possible to show how the workload generators interact.

In a multiple-machine environment the task of synchronizing multiple workload generators *across* the machines becomes more complicated. In order to be able to cross correlate time stamped trace data from workload generators on different machines it is necessary to define one machine as having the *master* reference clock to which all other machines are synchronized to. One workload generator (or workload control process) from each machine is responsible for synchronizing to the master reference clock machine and disseminating this information to all other workload generators that may be running on the same machine.

The process of determining a *global* time value is relatively simple and takes very little time for a single process. Given that there is a single machine defined to be to *master* time-server, a machine wanting to synchronize with the master machine would send it several requests for the master time clock value. Given the time values from the master clock machine and the round-trip time to actually get these values, it is possible to determine a *global* time that would be approximately the same for every other machine accessing the same master clock machine. Using a traditional 100Mbit Ethernet and standard PC hardware running 700 MHz Pentium III processors it is possible to get a global clock that is accurate to within ±100 microseconds. Given the time to processes most I/O operations to a disk subsystem are greater than 1 millisecond (or 1000 microseconds), the resolution of this global clock is sufficient for the purposes of I/O performance testing.

One minor issue that arises from the use of a single machine to act as the master reference clock is that as the number of machines in a cluster running workload generators increases, the time that it takes to get the master reference clock to all the machines increases as well. The main reason for this is that the process of determining a global time in a cluster of many machines should be serialized among those machines in order to get the most accurate value for this clock. If the time it takes to determine a global clock for a single workload generator on a traditional 100Mbit Ethernet is, for example, .25 seconds for any single machine to determine its global time, then the time it takes to synchronize all machines properly would be N times .25 seconds. As N gets large, say 1000 for example, then the total time to get the processes synchronized and started can be on the order of minutes. This time can of course, be significantly reduced by using other faster, lower latency networks if available. This is mentioned only to show one potential problem area when scaling from a single machine to a large cluster of machines.

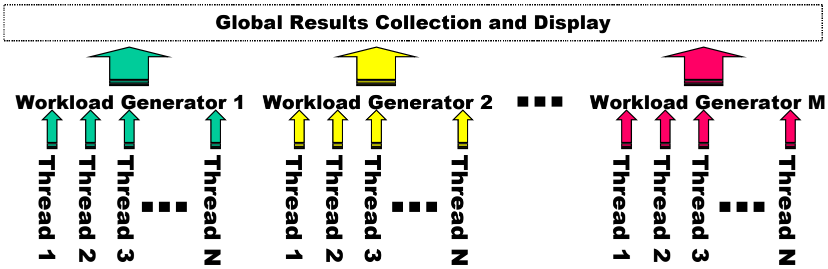
Performance Data Collection

Two issues to consider when collecting performance data are the type of data collected and when to collect it. Concerning the first issue, there are essentially two types of data to collect from a workload generator: *Aggregate* data and *Time-Stamped* data. The second issue of *when* to collect the data has to do with moving the performance data between the workload generators and after all the workload generators have quiesced so as not to inadvertently skew the results by moving data over the same networks that are being measured.

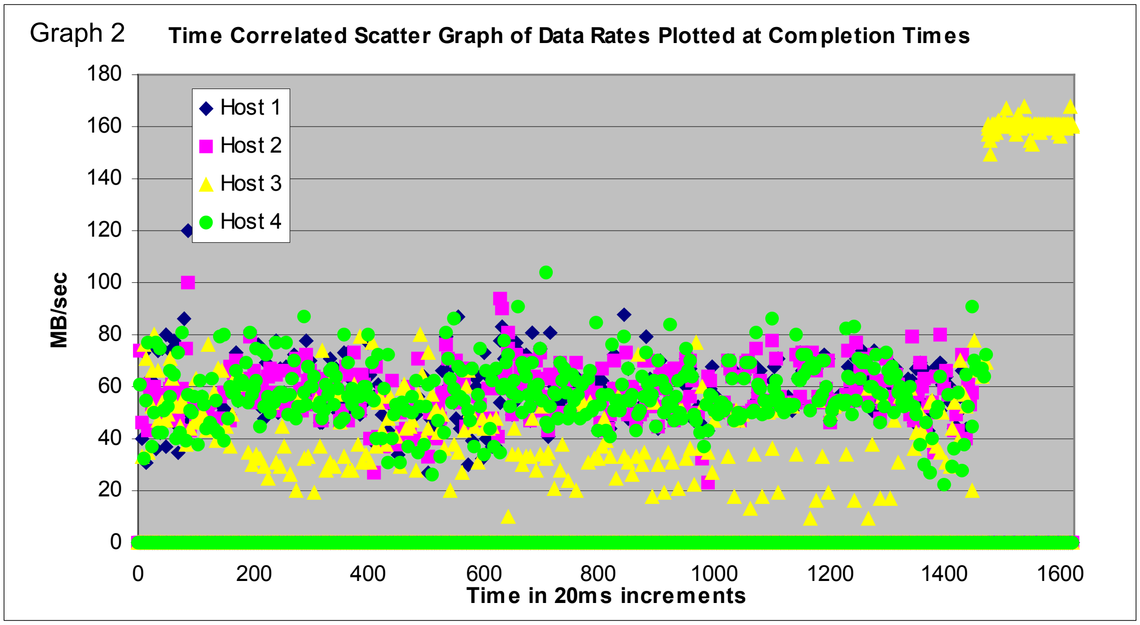
Each workload generator runs a set of workload threads, each of which can generate performance data. This performance data is collected by the workload generator and is passed back its parent or master workload generator for further processing. The *aggregate* performance data is continually passed up the hierarchy until it reaches the top of the control "tree" at which time it is analyzed and displayed. This hierarchical approach is used to efficiently handle small amounts of performance data that represents the aggregate results but not the large amounts generated by the time-stamping operations which could easily result in several megabytes of data per thread.

The *aggregate* performance comes from three different levels in the hierarchy:

- A single thread of a workload generator
- A combined average of all threads in a workload generator
- A combined average of all workload generators



The *time-stamped* data is important for visualizing the behavior of individual workload threads as well as the interaction of workload threads during a single run of concurrent workload generators. Graph 2 shows the time stamped results from four individual workload generators running simultaneously on four different machines (one thread per machine). Each dot on the graph is an individual I/O event that represents the instantaneous bandwidth performance for a specific I/O operation as a function of the time that the operation completed. Upon close observation, it can be seen that hosts 1, 2, and 4 generally maintain approximately 55MB/sec each while host 3 seems to continually lag behind at approximately 30 MB/sec until the very end (at time 1500 ms) where hosts 1, 2, and 4 stop issuing I/O operations and host 3 utilizes the entire 160MB/sec available bandwidth. Without this time-stamp data it would be difficult to see the relationship and behavior or interaction of the four host computer systems.



Run Monitoring and Report Generation

During a run it is helpful to monitor the progress of the benchmark during the run. However, the amount of data that is reported during the monitoring process as well as where it is reported should take into consideration the issues of data movement mentioned in the previous section. The monitoring process must not consume so many resources (compute, network, …etc.) that it interferes with the benchmark. Ideally, the monitor process should be run on a machine separate from the machines running the benchmark and the amount of performance data being collected by the monitor program is kept to a minimum.

Once a benchmark run is complete it is reasonable to collect performance data onto a single machine for the purposes of analysis and report generation. There are different levels or reporting that are important to the benchmarking and characterization efforts. These include:

- Summarized report of the run

- Detailed report of the run from each node
- Detailed report of trace data from each node
- Analysis tools for the detailed trace data

The Summarized and detailed reports from each run are necessary to understand the overall performance of the system and all the operating and configuration parameters used during the run. Given the amount of time-stamped data from a large run it is necessary to build an entire post-run analysis and knowledge-extraction system to aid in the interpretation of this data. At this time of this writing this capability has not been implemented.

Summary

The 30TeraOp machine is scheduled to be brought online later in 2001 with its complete I/O subsystem. The I/O architecture of this machine presents many valuable lessons in *scaling* I/O in a clustered environment. One such lesson is the process of designing and running an I/O benchmark program that is attempting to mimic the behavior of an application or a class of applications, interpreting the results, fine tuning the I/O subsystem and/or file system(s), and re-running the benchmark in order to achieve a certain level of I/O performance is very labor intensive. The lesson learned is that ideally, the applications themselves should become the I/O performance benchmarks. Because of the scale of the 30TeraOp machine, a detailed, real-time system-perspective I/O monitoring capability would be an immense help in identifying performance bottlenecks that occur for different applications because it is difficult if not impossible to see all the I/O anomalies from the benchmark itself. This real-time information could be fed back into the system in an effort to dynamically adjust whatever is necessary to improve the I/O performance of the application that is running. And after all, running the application is what counts, not the benchmark program.

References

[1] Tang, Diane, “Benchmarking Filesystems”, *Thesis*, TR-19-95 Harvard College Cambridge Mass, April 1995

[2] Smith, Keith A. et al, “File System Aging – Increasing the Relevance of File System Benchmarks”, *Proceedings*, 1997 ACM SIGMETRICS Conference, June 1997, ACM

[3] Bancroft, Martha, et al, “Functionality and Performance Evaluation of File Systems for Storage Area Networks(SAN)”, *Proceedings*, 17th IEEE Symposium on Mass Storage Systems / 8th NASA Goddard Conference on Mass Storage Systems and Technologies, April 1999, IEEE Computer Society Press