Two Case Studies of the Application of Dynamic Modeling Techniques in Performance Assessment and Prediction of Complex Shared Storage Architectures

Marti Bancroft, Phillip L. (Rocky) Snyder, and Mark Woodyard
Sun Microsystems, Palo Alto, California, USA
Contact: marti.bancroft@west.sun.com, (503) 537-9155 voice

Abstract

In evaluating alternatives in large supercomputer complexes, the tradeoffs between when NFS-type distribution of data and the more recent 'cluster' or 'Storage Area Network' approaches are optimal can require careful analysis. Further, many HPC (High Performance Computing) workloads are not limited to using only large files accessed with large block I/O requests. Some loads can be very mixed. File systems can fill up rapidly, resulting in performance which deviates substantially from that observed in relatively empty file systems, when most accesses are on the outer tracks and metadata accesses are most efficient. Establishing the optimal file system parameters for a group of homogeneous supercomputers attempting to exchange data is challenging enough. When heterogeneity is added to the HPC complex, as - for example - when using one of the commercially-available heterogeneous SAN software products, the task of defining a configuration to ensure the desired behavior, both under a steady-state load and during surge conditions, can become daunting.

In the work we describe in this paper, we investigated two different systems and storage architecture situations. The first was a relatively homogenous system, more than 10 supercomputers of the same vendor, same operating system level, similar or identical capabilities, which needed to share primarily large file, large block data at several different sustained rates but with very predictable performance for certain transactions under even the heaviest surges. There are, however, some smaller files in this workload. The goal was to evaluate whether a SAN, or NFS using high bandwidth local links, or some combination of both, could best provide the desired system behavior. An additional goal was to identify the various tuning options which could be used by the system administrators and application developers to ensure the performance required was preserved as the system grew or various initial conditions changed substantially. The second system evaluated was a heterogeneous system, consisting of servers and supercomputers of different capabilities and different product generations, as well as running different operating systems on hardware from different vendors. Connections between these systems ranged from high-performance local links to WANs (wide area networks). Some of the issues considered in this system related to the proximity of the dataset to the intended compute server, since the goal was optimal workload distribution as well as deadline assurance.

This paper reports on the analysis techniques used and their preliminary correlation with observed data (to the extent test configurations were available). Static analysis (using spreadsheets, assuming steady-state conditions among the participating systems), dynamic behavior modeling using a tool called Extend from Imagine That, Inc, and performance testing on a configuration subset were all used in an attempt to identify configurations that performed as required, no matter the load. This work is currently in process, the results presented here are those on which further architecture investigation will be based, and aimed at the storage aspects. In system #2, an additional focus of the research is on the development of mechanisms such as a multidimensional 'currency' and intelligent agent-bidders, which can work together to effect optimal scheduling of resources, of which the storage elements are simply one resource type. That research will be more fully reported elsewhere, only the storage aspects are discussed here. Updated test results will be presented at the conference. We will also show the dynamic models executing a workload scenario.

The requirements of the .com world and that of classic scientific HPC have an interesting intersection in the area of storage. This is surprising to some, because HPC storage is seen as being the 'classic' supercomputer I/O problem - very large files, large blocks, and a desire to transfer data to and from the user space directly rather than dividing it into small chunks to better utilize UNIX system buffering. Since time to solution drives most HPC work, having computation wait for I/O - other than to get the first portion of data upon which computation can meaningfully operate, is quite unthinkable. Thus the concepts of list I/O, asynchronous I/O, direct I/O, and raw byte-streams at rates exceeding 100 million bytes per second are almost synonymous with supercomputing. Although we consider supercomputing workloads to be more than simply streaming large files, there certainly is a .com analogy in the delivery of video-on-demand and music-on-demand.

There are already two practical side benefits of the approach we are using. One is that the dynamic modeling can also provide insight into cost-effectiveness of additions to the systems. For example, is it better to add a switch, a new set of RAID disks, or a new server? The other practical benefit is in quickly investigating behavior during failures. Configuring complex systems for availability and ensuring a minimum level of performance is maintained during a partial outage can benefit from the rapid 'what-if' type analysis that a behavioral model allows, pointing at options meriting a more detailed study. Even a comparatively simple model can shed light on system responses.

The rest of this paper is in three parts: a description of each of the analysis or test techniques, a discussion of case study #1, and a discussion of the storage aspects of case study #2.

Techniques Used:

The initial approach to define the parameters to be modeled and then tested in each case study was to use static analysis, using spreadsheets, and assuming steady-state conditions among the participating systems. The spreadsheets were constructed using high-level block diagrams of the hardware components with assumptions about their steady-state behavior in the system. These assumptions were based on either requirements for the overall system, simplified input-process-output models with an assumed linear relationship between the variables, or – sometimes – mathematical models of the individual components which predict their behavior under a range of carefully defined circumstances. While this approach may seem trivial, it is surprising how much information can be obtained during an initial feasibility study if some of the component behaviors are well known. Another advantage is that the spreadsheet approach can rule out configurations that cannot be supported and quickly focus attention on those which had a possibility of being successful. We have also used spreadsheets to assist in prediction of single-CPU performance for actual codes for one or even two future generations of product. However, this paper will focus on their use in defining overall system behavior.

If the system static-analysis spreadsheet is structured to use blocks which represent discrete hardware elements of the system – at the highest level one block would be a computer, another a storage subsystem, another a switching element, and yet another an input device – as shown in Figures 1 and 2 – this leads most directly into the dynamic or behavioral modeling tools, which also are easiest to construct if the highest level corresponds to those components. A side benefit is that they are also easiest to explain to others when displayed at that level. Since part of our task is explaining both validated and predicted system behavior to customers and our own management, this is not – to us – a benefit to be ignored.

The spreadsheet model blocks can have their behavioral data determined by either simple relationships (example: the outputs are equal to the sum of the inputs), by more complex mathematical relationships derived from testing or other analysis (examples include the application of an estimate of faster I/O rates required to ensure any transient effects are overcome), or by tables of data for discontinuous behaviors (which are, unfortunately, known to occur in systems under heavy load). The simplistic approach represents only a snapshot in time, and assumes equilibrium conditions – or a well-defined set of same – which does not resemble the likely behavior in a complex arrangement of components.

Once some configuration options are defined in this manner, either testing or dynamic models, or both can be used to observe and predict the performance of the complex over time, under varying loads and with varying assumptions about the capabilities of the components of the architecture, as well as changes in the architecture itself.

The dynamic modeling tool used for the work reported in this paper is Extend, by Imagine That Inc. of San Jose, California. This tool allows the simulation of systems which have behavior that can be characterized as a series of events (single computer disk activity, shared file system activity by multiple computers, operating system resource management, network transmission behavior, computer internal behavior (CPU execution, cache behavior, memory behavior) and the like. It also allows the simulation of

processes that may be described by equations (chemical reactions, orbital mechanics, and similar). Model blocks containing both types of simulation may be intermingled in a model, as well as blocks that describe a behavior by means of a 'program'. A large collection of libraries is available with the tool, and additional ones are available for purchase from other sources, all of which makes construction of models fairly expeditious. Validation can take as long as building a model. The compromise between validation and model construction is highly dependent on the system under study, as there is a trade between the costs of constructing a test – small to moderate in the case of determining behavior using existing computer components, to high in the case of a new airplane design – and the expense and difficulty of constructing a model of sufficient detail to accurately capture the behavior. Such modeling is extremely helpful in understanding time-dependent system behaviors.

The development of a model of a complex system is a stepwise process, as shown in Figure 3. Each of the components in Figures 1 and 2 can be regarded as a high-level block in such a model, with connections between the blocks to show the flow of events. Within each block, as the model becomes more detailed, there may be many sub-blocks, each characterizing the time-dependent behavior of an aspect of the overall construct. Linking these all together with links that – in some cases – must be characterized as well – yields a behavioral representation of the system under study to which can be posed a variety of questions.

Case study 1: server or SAN architecture?

Figures 1 and 2 show, at a high level, the architectural alternatives considered for the first system studied. The arrows show the direction of the majority of traffic, and the use of parallel lines as fill indicate multiple high-bandwidth streams. Issues which needed to be addressed by analysis, testing, and predictive models included sizing of the components, scheduling options in Solaris to ensure required behaviors on the input streams, scalability of the system, feasibility of using a shared reader/writer file system for the number of processes anticipated, optimum layout of the file system in either approach, and the impact of various failure scenarios and surge scenarios on the overall performance of either approach.

Initial analysis (static, using the spreadsheet approach) showed that the two input servers in Figure 1 were likely to be near maximum configuration and fairly heavily loaded, depending on the protocol used for the transfer of data to the downstream processors. This made the system not easily scalable and failure of a single component (e.g. one of the input servers) would effectively halve the throughput of the system. While this effect could be ameliorated by using the domain partitioning and dynamic domain reconfiguration features of Solaris on the Enterprise 10000 and follow-on Sun SMP HPC systems, an obvious alternate approach was to reconsider the data flow through the entire system. Figure 2 shows an alternative in which the input streams travel directly to a shared disk buffer, rather than passing through a server memory. In the absence of proven third-party I/O capability to the archive component, small archive servers were evaluated to provide the transfer of datasets between the input buffer and the archive media, and their retrieval on demand. The input streams must be handled in real-time, but there is effectively no flow control on them. Thus a variation of alternate 2 was to have only input and archival handled by an input server, and utilize the shared disk buffer to make input datasets available to the compute servers directly. The benefit of this variation is that a large memory can be configured in the now less loaded input/archive servers, which acts as a temporary buffer in the event of disk buffer congestion under load. This was of interest because of the need to try to keep stripe widths on the disk buffer as small as possible.

To understand why that is a concern, the authors cite their various experiences with attempting to ensure the required aggregate bandwidths in a heavily-loaded file system by using ever wider striping factors. Severalthings occur. One is that the disks used – RAID strings of sufficient depth to ensure maximum fibre channel sustainable bandwidth over most of the tracks, the fact that the rate varies on the devices due to the recording, and the nature of the RAID controllers themselves – all mean that maximum sustainable performance requires a minimum amount of data per string (or LUN, the logical addressable unit, that being the RAID controller). Previous extensive testing with similar disks showed that, given then existing system latencies, there was an 'ideal' amount of data to each controller of between 3 and 4 Megabytes (power of 2 here). This assumes that the load on each file system (or file system partition) is a mixture of random reads and writes, which is characteristic of the anticipated tasking of either of the alternatives in Figures 1 or 2. The minimum required stripe width is that necessary to sustain one full rate input event and the equivalent of a full-rate read event, plus a to-be-determined smaller archival read event. In this system that minimum was believed to be an 8-way stripe, which would give a minimum I/O size of 32 Mbytes (power of 2 again). Why? Eight of the 4 Megabyte chunks are required to perform a single read or write to the disks, if traditional striping is used.

However, there are multiple input events and corresponding multiple read events occurring simultaneously in this system. The usual solution has been to increase the stripe width to provide the required estimated peak bandwidth, with a corresponding increase in minimum I/O request size. We have tested systems as large as 24 wide data stripes using 96 MB I/O requests. While these can be tuned to give a high aggregate performance there is a practical but often overlooked consideration. The wider the stripe, the greater the performance impact of a transient delay on any one of the RAID LUNs. Still, one of the questions to be answered by the analysis, modeling, and testing in case study 1 was whether better overall performance – sustained, and under surge or failure conditions – could be obtained by having multiple, narrower stripes among which the input events were distributed, or if the best configuration was the more traditionally used very wide stripe.

Previous experience indicated that the file system(s) metadata needed to be separated from the actual data, because the nature of the transactions differed and the short metadata transactions would interfere with the very large data transfers. The file system selected for testing for both alternates supported this feature, but additional questions had to be addressed about the scaling of metadata. How much simultaneous traffic could be handled by each of the potential metadata configurations? What was the performance impact of mirroring? Was there a benefit to having a non-rotational storage device as a metadata 'cacher' (e.g. some form of solid-state disk)? If this benefit was significant, then some method to ensure no loss of metadata was required. Performance of file system recovery was another consideration. Which was faster – recovery on the narrower stripe, or the wider stripe, or was there no difference. Intuitively, one can see that the scalability of multiple narrower stripes is likely to be more modular than much wider stripes – but what about the ease of a single name space, a single file system to search, which is one attraction of the wider stripe approach?

To investigate this, we also evaluated the behavior of a segmented file system with stripe groups, with an allocation policy which caused each new allocated file to go to the next segment, in a round-robin fashion. Since a given in either alternative was that the file for an incoming input event would be preallocated (the goal to be to get as much continguous sequential space as possible), a robust implementation of such a scheme would have – for example – 3 stripe groups, each 8 fiber channels wide, and each one being considered a segment of the overall file system space. Thus the first input event would be allocated to segment one, the second to segment two, the third to segment three, the fourth back to segment one, and so on. Given the initial decision to delay the start of reading any file until the final write completed, this would have the effect of starting reads on a segment in which there would be a period of no high-bandwidth priority write events. If this approach worked reliably, then the question of stripe group size, number of stripe groups, and scaling of metadata and overall file system size would need to be studied again as the results might differ.

Yet another issue in both alternatives is the best method for handling the input events to ensure no loss of incoming data. When the path is through a server memory, the penalty of the double transaction is mitigated by the benefit (when needed) of the server memory as a rate- smoothing buffer. Solaris 8 has a reliable real time scheduling class, plus provides for user-defined classes. Additional questions to be answered relate to whether it is necessary to use the real-time class for both the input to server memory (which seems a logical choice, as that decision could be arranged so as not to interfere with other work on the input server) and for the writes to the disks. In the first alternative, the input server is required to both write to the disks with some guaranteed behavior, and read from the disks to feed work to the compute servers. If a real-time priority is used for input writes, would the read processes be effectively locked out? Is there a combination of priorities which would ensure the writes always keep up with the input, yet enough reads occur to ensure the computational timelines are met? Would using multiple narrower stripe groups ameliorate the situation? Given there are interdependent factors (scheduling class, priorities within the class, size of the memory buffers, scenario of simultaneous activities) this is yet another question to be addressed by analysis/modeling/testing.

In the case of the ingest event being written directly to disk, there is effectively no way to prioritize the behavior of the different events occurring on the same spindles. Yes, a resource lock would affect a file, for example a file would be locked to a single write during an input event, and it certainly could be possible to lock a file to a single reader. However, this would not necessarily control the activity on the disks. In such a system, and even in the event there is still an input server acting as a 'buffer' for the input events, the fundamental nature of a shared access file system is that the performance experienced by any specific host, for any individual read or write, can vary. It is clear that modeling of time-dependent behavior is critical in such an environment, because while the sustained or average rates may show ample performance

margins, the lows may be inadequate. In a switched SAN environment, using fibre channel switches, we found – as reported at the NASA Mass Storage Symposium 2000 – that a crude bandwidth allocation could be arranged by using different I/O request sizes, where multiples of the basic minimum request size could in effect get a larger portion of the total bandwidth because of the way the intervening switch handled connection management. Still, depending on this for guaranteed results – especially as a system scales – made us nervous. Still, some testing was done to parameterize this option. Another approach is to vary the host bus adapter stripe width as a method of distributing bandwidth in the shared reader/writer file system. This was another area tested, because it was not clear that the hoped-for behavior (the wider the host bus adapter stripe, the more of the effective disk bandwidth to that host) would be given, over time, to that host.

At the time of this draft, the spreadsheet is complete, the behavioral models are partially complete, and the testing is planned and equipment being gathered in Sun's HPC Benchmarking Center in Beaverton, OR, for testing during the month of December. Figure 5 depicts the test configuration (a subset selected because we believe it is the minimum required to give scalability answers). Table 1 lists the tests being performed (not all are complete as of this revision). Table 2 contrasts the performance of NFS v3 over Gigabit Ethernet and direct I/O to the same 12-wide stripe filesystem. Table 3 shows the baseline NTE (not to exceed) single channel rates before and after a necessary software patch which reduced write performance. This gave us a measure of expected write performance once the patch is no longer necessary. All remaining testing was done with the patch installed. Table 4 shows the efficiency of striping performance, in terms of the single channel rates. Table 5 shows preliminary test results for contention on the file system from the same host. Table 6 shows preliminary multithreading results. Table 7 shows preliminary results of bandwidth allocation in a segmented loop or fabric connection environment.

Case study 2: effect of dataset size, anticipated processing time, and link characteristics on decisions in a virtual computing system

In a computing system that is composed of components that are geographically distributed, another factor enters into predicting performance – the effects that the size of the dataset, its proximity to the compute server alternatives, and the performance of the intervening links can have on attaining the least time to result. An additional factor, of course, can be cost. These are in addition to the relative performance differences of candidate compute servers. To further complicate this environment, in some cases there may be multiple copies of the dataset (this occurs where there are data repositories geographically distributed for whatever reason) and the choice now becomes which dataset, which compute server, and which link combination will give the best result.

Wide area network (WAN) links can have widely variant behavior under surge or failure conditions. While a complete analysis of those was beyond the scope of this case study, we relied on the work of a colleague, Norm Eaglestone (Sun Microsystems Federal, McLean, VA) in bounding the behavior of WANs as input to the models. The primary work to be reported on in the time frame of this paper is the analysis and modeling work.

The portion of this project which will be presented at the NASA Symposium deals with the modifications necessary to portions of Sun's Grid Engine product to include these dataset and link related concerns in the scheduling and load balancing features of GRD (Global Resource Director). The work consists of analysis of the behaviors needed, use of the behavioral modeling tool Extend to evaluate alternatives which results in optimal behavior, and then to prototyping by modifying portions of GRD and (probably) adding a module to deal with dataset distribution issues as part of the scheduling decision process. Figure 4 shows, at a high level, the architecture used in the model. Although one 'cloud' is shown as 'The Net', in the actual model there are many links, ranging from high performance local (direct) connections to WANs. Some 'logical' nodes clearly have multiple of the resource types closely coupled.

Sun GridEngine (formerly GRD/Codine) was chosen for prototyping because it already has many of the constructs needed for large scale workload balancing, resource management, and job control, and is modular and extensible in the directions believed needed to deal with data placement issues in a loosely coupled, dynamic environment.

One anticipated benefit of evolving the dynamic modeling in this area over the next months is to help answer questions like: Where should secondary data archives (if any) be placed to produce the best workload balance? Which is a better investment – upgrading a computational resource close to the data source, or duplicating frequently-used data sets closer to alternate resources? What is the effect of

upgrading a network link to increase the bandwidth? Where can duplicate archives be placed to best ameliorate link outages?

The anticipated evolution of this approach is into flexible, dynamic systems, in which sources, archives, computational resources, and even links will be configured dynamically along with the more permanent resources. Sun's Java/Jini/Jiro technology has already been demonstrated in prototype dynamic configurations. Legacy archives can be 'wrapped' to allow them to participate in this dynamic environment as a single logical resource. Clearly scaling is yet another question to be addressed by the modeling effort.

Table 1: Initial Testing Matrix

Test Description	Striped	StGrp
Single Host - write - best parameters	X	X
Single Host - read - best parameters	X	X
Single Host - write and read, same file system, different files	X	X
Single Host - read and write, different file systems	X	X
Multiple Hosts - read and write, single file system, different files	X	X
Multiple Hosts - read and write, sngle file system, same file using extent-based locking	X	X
Outer track maxima	X	X
Inner track minima	X	X
Workload Mix - Direct I/O, Large Files	X	X
Workload Mix - Buffered I/O, Smaller Files	X	X
Effect of Combination Workload, Same File System	X	X
Metadata Loading - Alone	X	X
Effect of Metadata Load on Performance - Single Host, basic	X	X
Effect of Metadata Load on Performance - Multiple Hosts, basic	X	X
Effect of Metadata Load on Performance - Differing Workloads	X	X
Effect of Connection Type on Achieved Bandwidth Partitioning	X	X
Effect of Host Scheduling Class on Achieved Bandwidth Partitioning	X	X
Effect of Different Metadata Options	X	X

Table 2: NFS v3 vs 12-way direct I/O stripe

Test	NFS	Direct I/O
Read 12 GB	30 MB/s	1100 MB/s

Table 3: Effect of Patch on Single Channel Write Performance

Test	No patch	Patch
Read	92.5 MB/s	92.5 MB/s
Write	82.5 MB/s	68 MB/s

Table 4: Stripe Efficiency as % Single Channel Rates - Direct I/O

Test	6 wide (no patch)	12 wide (write patch)
Read 12 GB	95%	95%
Write 12 GB	94%	98%

Table 5: Same Host Contention - Same File System, Different 12 GB Files (12 threads)

Test	Stripe = 6	Stripe = 12
Read	506 MB/s	1106 MB/s
Write	465 MB/s	799 MB/s
Read + Write	411 MB/s aggregate	974 MB/s aggregate

Table 6: Effect of Multithreading on 6-way Striped File System - 12 GB File

# threads	Read	Write	R + W
1	-	139 MB/s	64 MB/s
3	346 MB/s	336 MB/s	147 MB/s
6	537 MB/s	444 MB/s	155 MB/s
12	515 MB/s	479 MB/s	205 MB/s
18	500 MB/s	482 MB/s	215 MB/s
24	520 MB/s	480 MB/s	227 MB/s
32	522 MB/s	480 MB/s	233 MB/s

Table 7: Bandwidth Allocation Effect of Fibre Channel Connection Type

Segmented Loop	File system bandwidth to requestors independent of block size
Fabric connections	File system bandwidth to requestors affected by block size, larger I/O requested yield greater performance

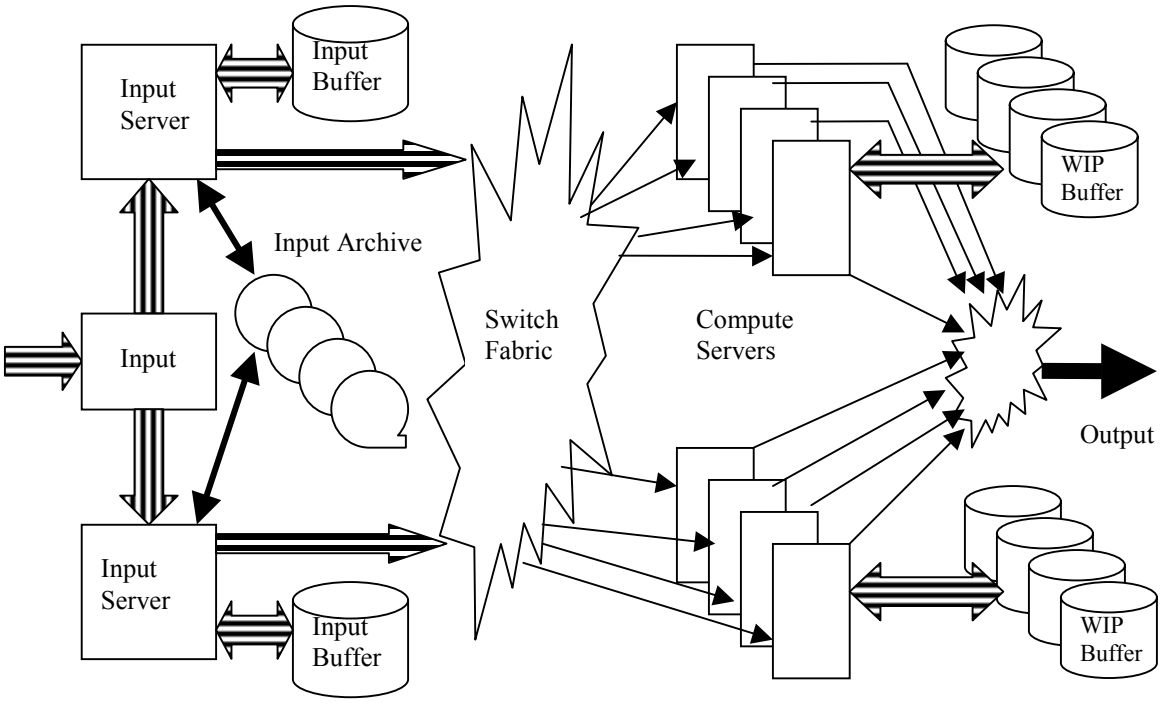

Figure 1: System Block Diagram for Static Analysis – Option A

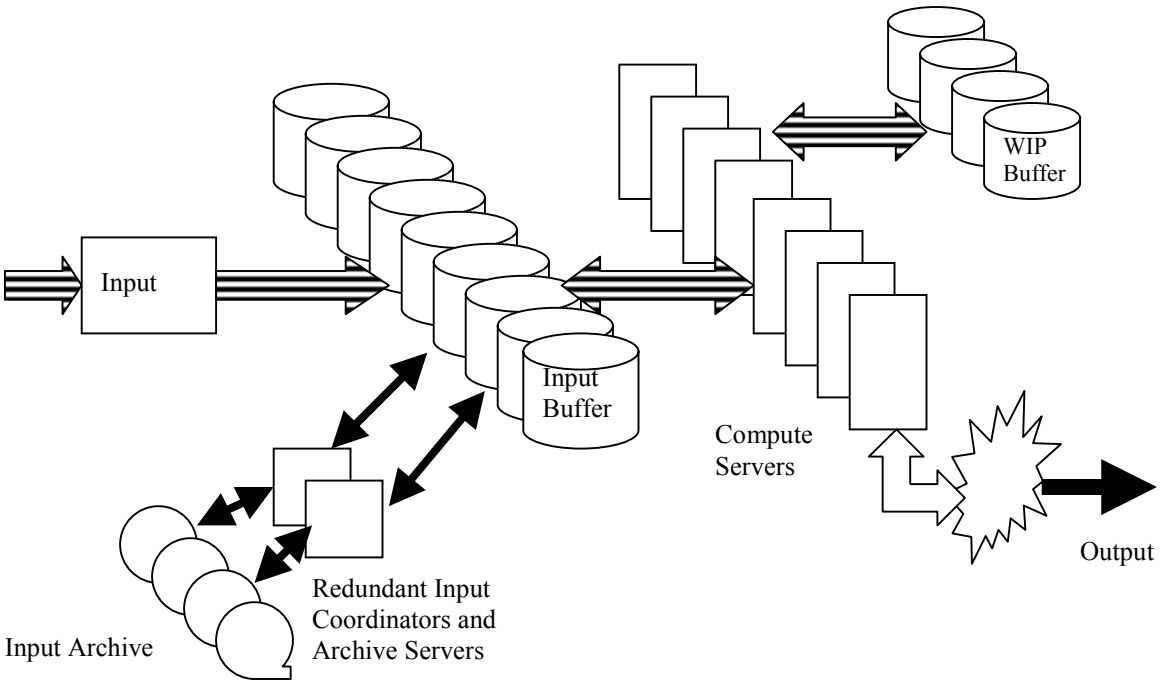

Figure 2: System Block Diagram for Static Analysis – Option B

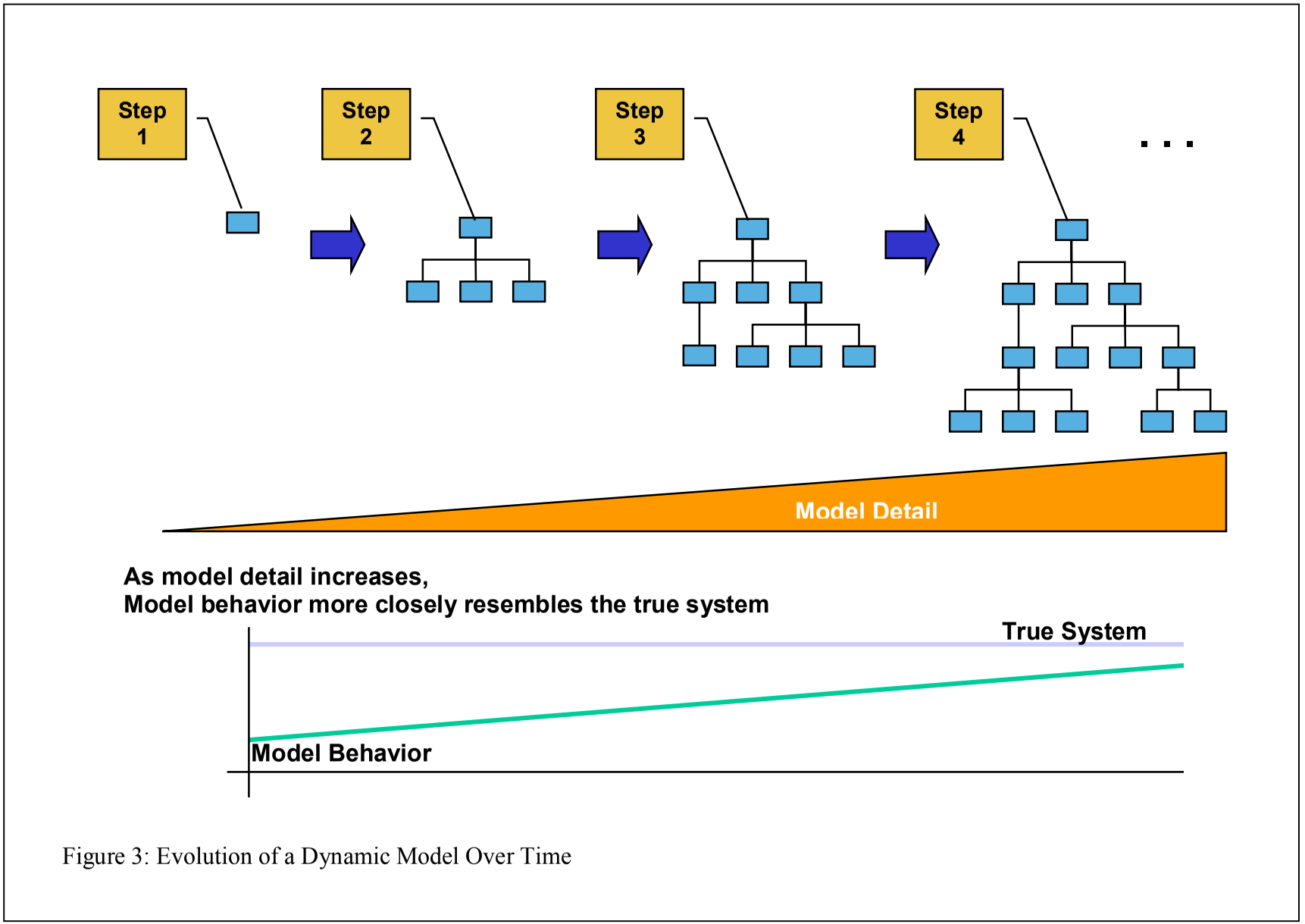

Figure 3: Evolution of a Dynamic Model Over Time

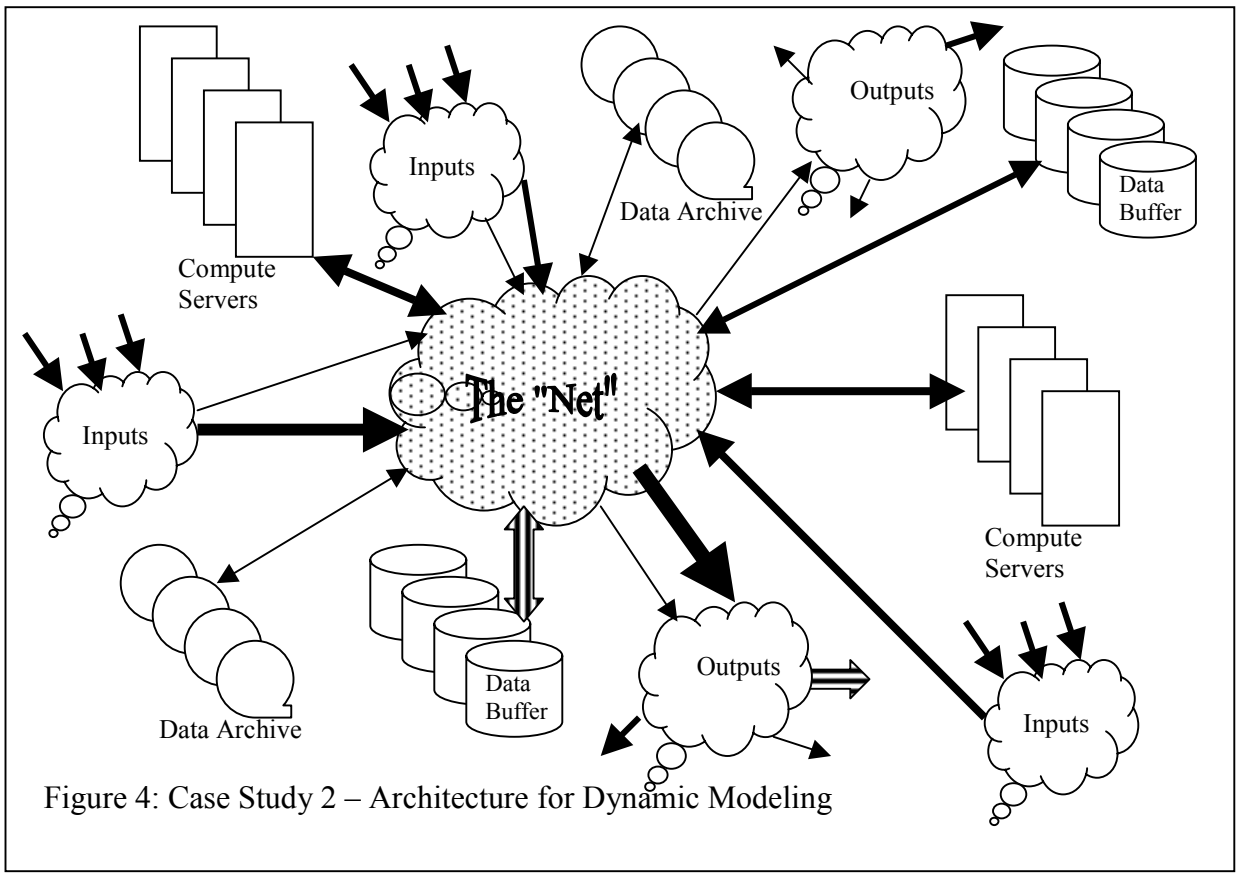

Figure 4: Case Study 2 – Architecture for Dynamic Modeling

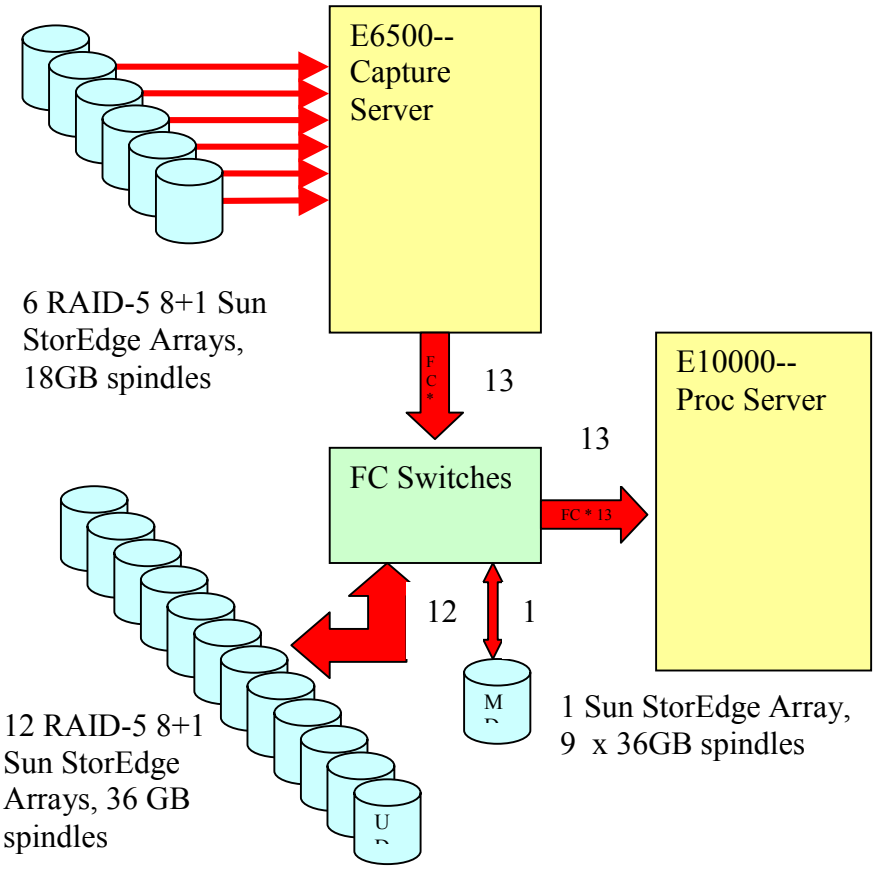

Figure 5: Initial Test Configuration

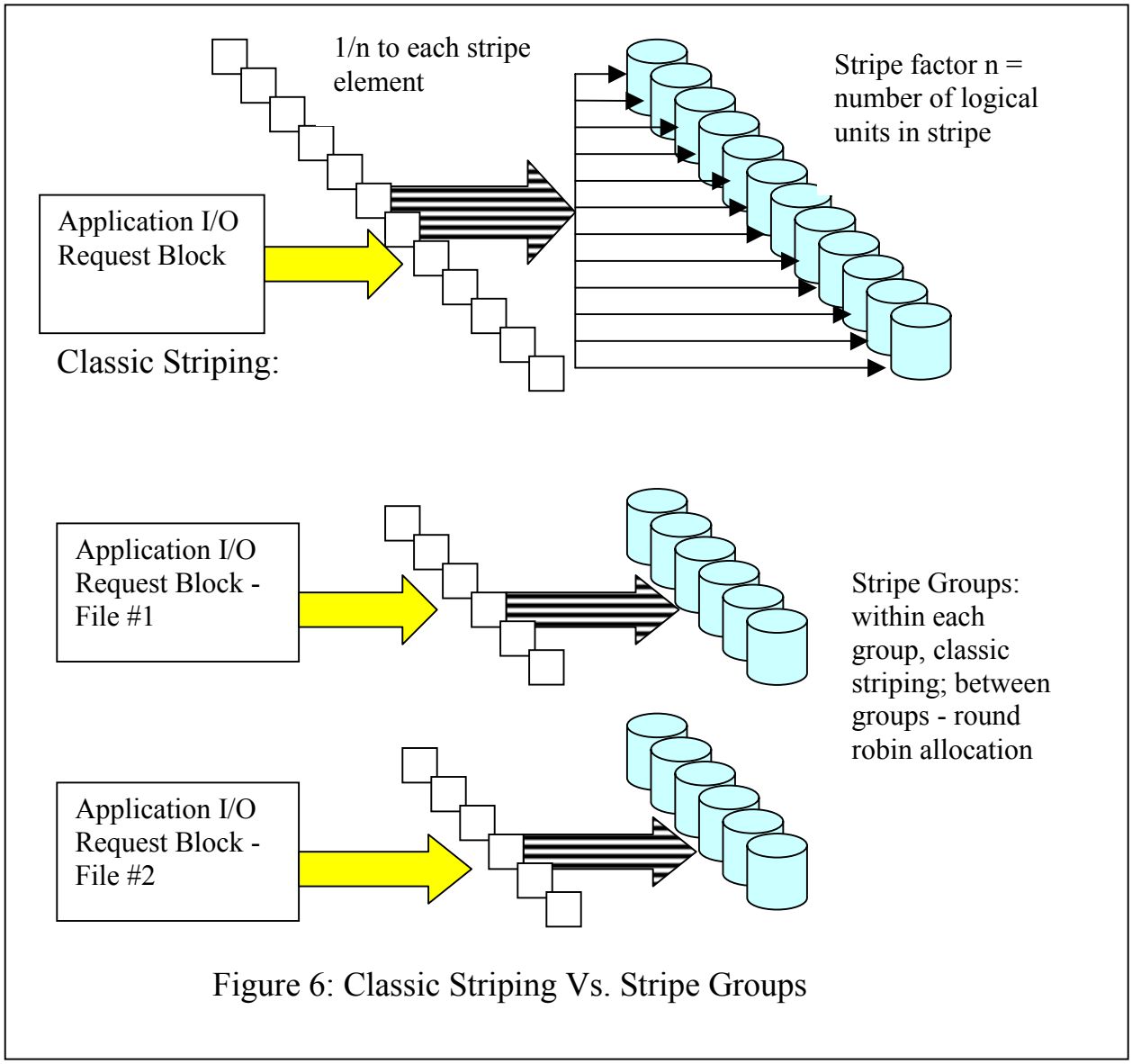

Figure 6: Classic Striping Vs. Stripe Groups