Adaptive Disk Striping for Parallel Input/Output

Huseyin Simitci Daniel A. Reed*
{simitci, reed}@cs.uiuc.edu

Department of Computer Science
University of Illinois
Urbana, Illinois

Abstract

As disk capacities continue to rise more rapidly than transfer rates, adaptive, redundant striping smoothly trades capacity for higher performance. We developed a fuzzy logic rule base for adaptive, redundant striping of files across multiple disks. This rule base is based on a queuing model of disk contention that includes file request sizes and disk hardware parameters. At low loads, the rule base stripes aggressively to minimize response time. As loads rise, it stripes less aggressively to maximize aggregate throughput.

This adaptive striping rule base is incorporated into our second generation Portable Parallel File System (PPFS II). Experimental results showed that the analytical models of disk striping are capable of accurately predicting file system behavior. Also, it is shown that, depending on the access pattern, adaptive striping can double the input/output performance compared to striping with fixed distribution parameters.

1 Introduction

As new high-performance computing systems, achieving multi-teraflops and beyond quickly emerge, the performance of storage subsystems remains an obstacle to utilizing the full power of these systems. Moreover extant parallel file systems (e.g., SGI XFS [1] or IBM GPFS [2]) cannot deliver the full hardware input/output bandwidth to these parallel applications. Even small changes in the access pattern may cause performance degradation. Emerging distributed applications with time-varying input/output demands [3, 4] will exacerbate this situation.

To support multi-teraflop applications manipulating multi-petabyte data sets, next-generation file systems will have to stripe data over thousands of secondary and tertiary storage devices [5, 6, 4]. However, we have to balance the need to decrease transfer time using striping and the need to make multiple, independent transfers. Such a complex task will require file systems that can intelligently make adaptive file distribution decisions. To truly understand the effects of various access and system parameters on the performance of the striped requests, we need to derive analytic models of disk striping. These models will also allow prediction of the input/output behavior of peta-scale machines with hundreds of thousands of disks.

Several studies of parallel file systems [7, 8] have shown the importance of matching underlying file system policies with the application's access patterns. Mismatched policies and access patterns can significantly reduce input/output performance.

These observations point to several unresolved problems in dynamic file distributions — contention between accesses inside an application and across multiple applications and tradeoffs between disk space and input/output performance. More specifically, key research questions include:

- analytical models of disk striping to study the input/output performance on very large computational systems,
- adaptive selection of data striping policies based on request patterns and system load,
- techniques for trading disk storage for bandwidth by redundantly storing multiple, striped copies of files.

*This work was supported in part by the Defense Advanced Research Projects Agency under DARPA contracts DABT63-94-C0049 (SIO Initiative), F30602-96-C-0161, and DABT63-96-C-0027 by the National Science Foundation under grants NSF CDA 94-01124 and ASC 97-20202, and by the Department of Energy under contracts DOE B-341494, W-7405-ENG-48, and 1-B-333164.

To address these issues, we are investigating the performance-directed selection of file striping distributions across storage devices and redundant storage of multiple distributions to reduce access time. This exploration builds atop our earlier work on physical and logical, input/output pattern comparisons [9, 10] and portable, parallel file systems [7]. It integrates real-time performance data, automatic access pattern classification, and fuzzy logic controls for choosing and configuring flexible policies. The foundation of the research is a prototype software library called PPFS II (Portable Parallel File System II).

In this paper, we outline an approach to adaptive disk striping that focuses on three primary research areas: development of analytical models of dynamic disk striping and redundant storage; utilization of automatic access pattern classification and real-time file system performance data; and implementation of fuzzy logic rule bases for disk striping policy selection.

The remainder of this paper is organized as follows. In §2, we present research relevant to this study. We discuss our approach to adaptive file system policies in §3. We present the queueing models of disk striping in §4. §5 contains an introduction to fuzzy logic control and discusses the adaptive striping rule base. In §6, we describe our experimental, adaptive, file system prototype. We present experimental results in §7. Finally, we conclude by presenting the directions of future work and summarizing the implications of adaptive disk striping policies in §8.

2 Related work

Disk striping optimization. One way to remedy the performance difference between computing elements and storage devices is to stripe data across several storage devices [5], effectively increasing the overall data throughput. This distribution technique is foundational to RAID systems [11], and striping file systems [12, 13]. But the effectiveness of this technique is dependent on the configuration of the storage system and the characteristics of the workloads using it.

Cormen and Kotz [14] point out that asymptotically optimal disk I/O algorithms require flexible striping parameters. They also note that requiring the input/output operations to be fully striped is equivalent to using just one disk with a block size multiplied by the number of disks. Vitter and Shriver [15] studied the optimal number of input/output operations required by several parallel algorithms. One of the results in this study reveals that the constraint of fully striped input/output increases the number of disk accesses by more than a constant factor compared to independent accesses to parallel disks.

Chen and Patterson [16] observed the importance of using an optimum striping unit, the amount of logically contiguous data on each storage device. They define *parallelism* as the number of disks serving a request, and *concurrency* as the average number of outstanding user requests in the system. Then, they show that higher levels of concurrency require lower levels of parallelism to decrease the contention for resources. Similarly, lower levels of concurrency allow more parallelism for individual requests, which decreases individual response times.

Elford and Reed [17] studied the effects of evolving disk technology on disk-array designs. Their modeling and simulation results suggest that the net effect of combined disk technology improvements is to reduce the range in which disk arrays are preferable to collections of disks that are not synchronized. Data density increases negate many of the advantages of disk arrays. However, this was predicated on many assumptions about request sizes and access patterns. In particular, it assumed that requests are distributed equally across the disks. High arrival rates, access hot spots, shared file access on parallel systems, and variable size requests all affect performance, which makes choosing an appropriate data distribution dependent on a host of interrelated factors.

In [18], Scheuermann, Weikum, and Zabback present an analytic model for striping on parallel disk systems which is very similar to the models that we discuss in §4. Our striping models differ by making more simplifying assumptions on service time distributions and considering the case of networks of servers and clients. In [18], only shared-memory multiprocessors are considered and network latencies are ignored. This work points out the importance of file specific striping tuning even in a shared-memory architecture.

Redundant storage. Several researchers have observed that disk areal densities are increasing much faster than access latency times (seek plus rotation) are decreasing [19, 6]. This fact only exacerbates the lagging disk access times. This improvement difference implies some trade-offs. Namely, one can use the extra capacity to store copies of the data files redundantly.

For performance and reliability improvements, file replication techniques such as mirroring have been extensively studied in the database community [20]. Wolfson, Jajodia, and Huang [21] proposed an adaptive, file replication scheme that migrates redundant copies of files to locations where the read-write activ-

ity is highest on a tree network. In an earlier study, Wolfson and Milo [22] showed that finding an optimal replication scheme, with minimal cost for a given read-write pattern over general network topologies, is NP complete.

Input/output characterization. There are a number of studies that present models of physical disk [23, 24, 25] and disk-array [17] access behavior. Also, the logical and physical patterns of application input/output in parallel scientific applications have been studied extensively [9, 26, 8, 27, 10, 28]. These studies have shown that parallel applications exhibit a wide variety of input/output request patterns. Insights from these studies led to new, parallel file-system application programming interface (API) standardization efforts like the SIO API [29] and the MPI-IO API [30].

Flexible parallel file systems. Almost all parallel file systems provide users with some way to customize the file system policies. For example, Intel Paragon's PFS [31] and IBM SP2's PIOFS [32] allow users to dictate certain file distribution parameters such as striping widths and striping units.

First generation PPFS (Portable Parallel File System) [7, 33, 34, 35, 36] is an input/output library, which is portable across parallel systems and workstation clusters. PPFS has a rich interface for application control of data placement and file system policies. Yet, to achieve performance gains with PPFS, as is the case in PFS and PIOFS, the application writer must understand both the application access pattern and the PPFS input/output cost model. However, several characterization studies have shown that developers often do not know their file access patterns in sufficient detail to correctly choose file policies. As a result, some studies have proposed techniques that can automatically classify access patterns [34, 35] and dynamically choose appropriate policies [33, 36].

Computational steering. Interactive application steering [37, 36] is studied extensively, particularly in the context of scientific applications and immersive visualization.

Several techniques for automated decision making have been proposed, ranging from decision tables and trees through standard control theory to fuzzy logic. In contrast to other alternatives, fuzzy logic has attributes that make it a good choice for poorly understood optimization spaces with conflicting goals [38, 39].

3 A parallel file system with adaptive striping

Clearly, it is imperative to choose the optimal number of disks to stripe across and the optimal striping unit if one wants to use parallel disks effectively. These observations led us to study analytical models of striping and to develop fuzzy rule bases for adaptive striping.

We are investigating techniques that can dynamically change the way data is striped across storage devices based on request sizes, request concurrency inside an application and across multiple applications, and file access patterns. To explore long-term trends and thousands of storage devices, we have constructed both analytic models and fuzzy logic rule bases that capture the relationship between access patterns and thousands of storage devices.

We consider policies that trade storage space for increased input/output bandwidth by generating multiple, redundant copies of data with different optimized distributions. The increasing gap between the storage capacity and access time makes this approach both viable and necessary.

Besides having higher storage requirements, redundant storage introduces other overhead. The copies must be constructed during file creation or copied offline after execution, and they must be maintained to keep data consistent. The advantage of redundant copies depends on the number of times the data will be accessed and the actual access patterns.

We are integrating adaptive striping policies into our next-generation parallel file system, PPFS II. This system will be discussed in more detail in §6. PPFS II is designed to be a testbed for adaptive, parallel file-system policies.

Parallel file data structures in PPFS II can accommodate multiple, redundant copies of a file with different data layouts. Figure 1 shows an example layout for a single parallel file with two copies. Runtime information about the file is kept as the *Dynamic Meta Data*. Data distribution information is contained in *cluster* structures. Each complete copy of the file constitutes a new *fork*. *Forks* are distributed — generally in a round-robin fashion, to the server disks. Each extent of a *fork* on a disk is called a *tine*. *Tines* are stored as the underlying operating system's native files.

The two *forks* in Figure 1 illustrate how different striping layouts can be implemented. *Fork 0* is distributed on two disks with a relatively big stripe size. *Fork 1*, on the other hand, is distributed among four disks with a smaller stripe size and with gaps between *tine* segments.

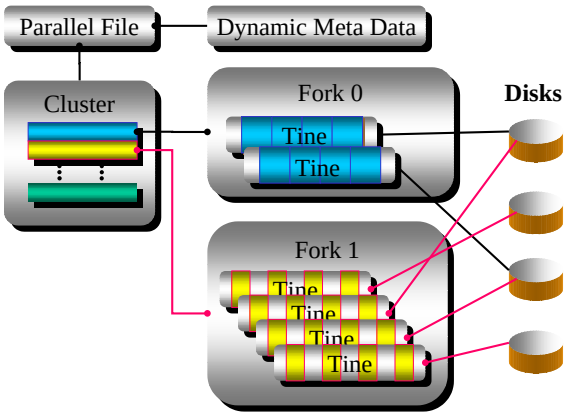


Figure 1. PPFS II file layout structure.

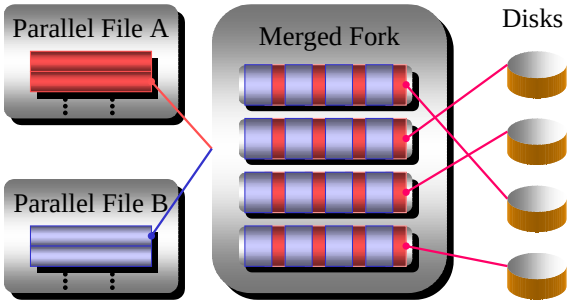


Figure 2. Merged storage of two parallel files.

This flexible layout structure allows other optimizations. If several files are accessed in an interleaved pattern by the application, they can also be stored in an interleaved fashion, as seen in Figure 2. This allows for some prefetching optimizations when a segment of one of the files is accessed. The adjacent segment of the next file will be next in sequence on the disk track, or will be cached somewhere in the storage hierarchy.

4 An analytical model of disk striping

Models of disk systems where each request is served by a single disk are well understood. They are generally modeled as M/G/1 queues [40]. However, for striping file systems where each request is served by multiple servers (fork-join queues), there do not exist any general analytical models.

In this section, we will explore analytical models of disk striping, the basis for our adaptive striping rule base. First, we will discuss the disk and network service time distribution assumptions.

4.1 Service time distributions

The notation that will be used for the remainder of the document is contained in Table 1. We assume the disk service time is the sum of seek time, rotational latency, media transfer time, controller-interface time, and serial software overhead. Also, we consider the case where the disks are distributed over a network and add a network setup time. Network transfer time will be included in the interface time. All of these service time components will be assumed to be nonoverlapping.

To maintain tractability, we assume that any rotational and seek latency is possible when a request arrives at the disk. If the time for a full rotation is c seconds, we assume the rotational latency is uniformly distributed on the interval $[0,\ c]$. Then, the expected rotational latency is $c/2$ seconds with a variance of $c^2/12$. This presumes rotational position sensing (RPS). Ruemmler and Wilkes [23] discuss the balance of accuracy and simplicity implied by this assumption.

Similarly, if the time for a full-stroke seek, a seek between farthest tracks, is f seconds, we will assume seek time is uniformly distributed on the interval $[0,\ f]$. Then, the expected seek time will be $f/2$ seconds with a variance of $f^2/12$.

For simplicity, we assume that the data density on the disk is constant and that there are t blocks on a track. We are not considering disk layouts which contain more blocks on longer, outer tracks than shorter, inner tracks. For a fixed request size, the media transfer time is a function of the disk rotation speed $1/c$ and the number of tracks accessed.

For each block transferred there will be i seconds of disk controller interface and network transmission delay. Since we are assuming that the disks can be distributed across a network, for each disk request we assume there is a network setup delay which is uniformly distributed on the interval $[0,\ k]$, with an expected value of $k/2$ seconds and a variance of $k^2/12$. Finally, since the client may dispatch the sub-requests serially, we will add an overhead of h seconds per sub-request to the service time.

4.2 Distributed striping model

In the model, it is assumed that there are m disks distributed across a network. Mean request size is l blocks. A block is a fixed amount of data on a disk, most probably representing a file-system block. And the total request arrival rate to the system is

Table 1. Modeling parameters.

Variable	Definition
c	Time for full disk rotation (s)
f	Time for full stroke seek (s)
k	Network connection setup time (s)
t	Size of disk track (blocks)
D	Request width (disks)
m	Number of available disks
l	Mean request size (blocks)
i	Network and interface delay (s/block)
h	Sub-request software overhead (s)
λ_m	Aggregate request rate (reqs/s)
λ_D	Request rate for stripe width D
R	(Sub-)request response time (s)
S	(Sub-)request service time (s)

λ_m. For tractability, we will assume Poisson arrivals. So, the interarrival times will have an exponential distribution.

We assume that each request is divided into D sub-requests, which are distributed onto D disks with a stripe size of l/D blocks. D will be referred to as the *request width*. For a given request size, the number of disks accessed by a request depends on the *striping unit*. The number of all the disks a particular file is distributed on will be denoted by the *striping width*.

In this model, multiple unrelated sub-requests might be queued on a particular disk because of overlapping requests. We assume the requests are distributed uniformly among the disks, and the request arrival rate for each disk is the total sub-request arrival rate divided by the number of disks, which is

$$\lambda_D = \frac{\lambda_m D}{m}. \quad (1)$$

Since uniform distribution assumes there are no access hot-spots, this will result in a lower-bound on the actual response time.

Each disk can be modeled as an M/G/1 queue [40]. A possible queuing network model is shown in Figure 3. Since, in this model, sub-requests are served asynchronously, first we will find the service and response time of each sub-request. Then, we will take the maximum of these sub-request response times as the response time of the main request.

The service time for each sub-request can be computed with six components,

$$S_D = \frac{f}{2} + \frac{c}{2} + \frac{k}{2} + \frac{c}{t}\left(\frac{l}{D}\right) + i\left(\frac{l}{D}\right) + hD. \quad (2)$$

The variance of the sub-request service time can be

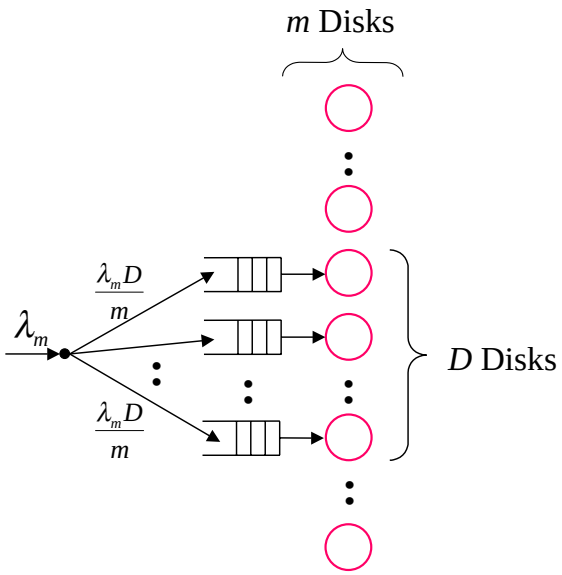


Figure 3. Distributed striping model.

computed as,

$$\sigma^2_{S_D} = \frac{(f^2 + c^2 + k^2)}{12}. \quad (3)$$

Then, for this M/G/1 queue, the queuing delay, W_D, and the response time, R_D, can be computed as

$$W_D = \frac{\lambda_D\left(\sigma^2_{S_D} + S^2_D\right)}{2(1 - \lambda_D S_D)}, \quad (4)$$

$$R_D = W_D + S_D = \frac{\lambda_D\left(\sigma^2_{S_D} + S^2_D\right)}{2(1 - \lambda_D S_D)} + S_D. \quad (5)$$

The response time of the main request is the expected value of the maximum of D sub-request response times. We will use the formula

$$R_{max_D} = \gamma(D)\, R_D \quad (6)$$

to approximate the expected maximum of D independent and identically distributed variables with a mean of R_D. The scaling factor $\gamma(D)$ depends on the probability distribution function of R_D [41, 17]. Kim and Tantawi [41] provide the following scaling functions:

$$\gamma(D) = \begin{cases} \frac{2D}{D+1}, & \text{uniform} \\ 0.5772 + ln(D), & \text{exponential} \\ 1 + \frac{\sigma_{R_D}}{R_D}\sqrt{2log(D)}, & \text{normal.} \end{cases} \quad (7)$$

Figure 4 plots this scaling function for various distribution function assumptions.

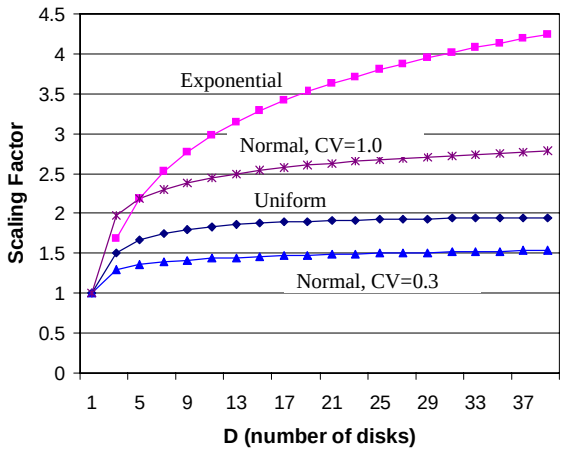


Figure 4. Scaling factors for D random variables.

If we assume a uniform distribution of response times, we will get

$$R_{max_D} = \frac{2D}{D+1} R_D. \quad (8)$$

We should note that this is an approximation. Clearly, some response time components do not satisfy this uniformity assumption. However, as will be seen in §7, experiments on our PC cluster has shown that this approximation accurately predicts the response times.

4.3 Parameter studies with the striping model

To explore the effects of each parameter on overall input/output system performance, we have plotted several performance graphs using some realistic system parameters. Unless otherwise stated, the parameters have the fixed values shown in Table 2. The values for rotational latency ($c = 8.4$ milliseconds), full stroke seek ($f = 18$ milliseconds), and blocks per track ($t = 22$) are approximated using the product specifications of the Western Digital WDE4360 Ultra SCSI hard drive, which is used in our experimental platform that will be presented in §7. Here, one block is assumed to be 4 KB. We will assume a system with 128 disks and an average request size of 1024 blocks. Network setup time and interface time will be assumed to be 30 milliseconds and 0.5 millisecond, respectively. We will assume there is an overhead of one millisecond per sub-request at the client. These values are observed during the experiments on our experimental platform. Finally, in plots where the request rate is fixed, we will assume a request rate of 30 requests per second.

Table 2. Parameter values used in plots.

Variable	Value
c	0.0084 s
f	0.018 s
k	0.030 s
t	22 blocks
m	128 disks
l	1024 blocks
i	0.0005 s
h	0.001 s
λ	30 req/s

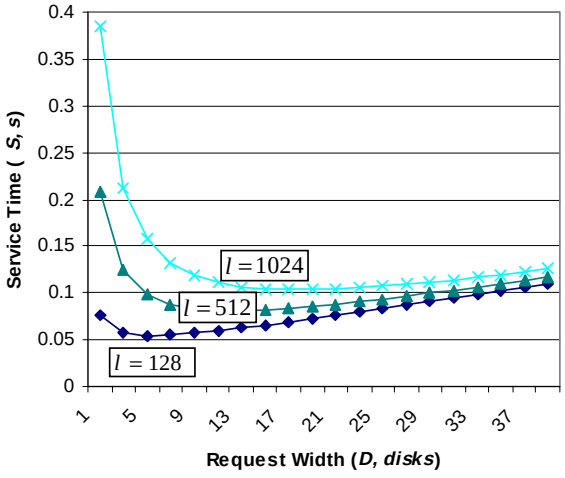


Figure 5. Service time as a function of request width.

First, we plot the service time, S, with respect to the request width in Figure 5. For this distributed striping model, the figure shows the service time of the sub-request, which will be scaled after the response time is calculated. As the number of disks used increases, the service time decreases because of the decrease in media transfer time. As expected, this decrease in service time slows down gradually. Increasing the request width beyond a certain point increases the service time because of increasing serial overhead. As seen from the figure, larger request sizes utilize extra disks better.

The response time is the sum of queuing delay and service time, and is scaled as in Equation 8. It is plotted in Figure 6 with respect to the total request arrival rate and the request width for a fixed request size of 1024 blocks. For a fixed request rate and a fixed number of total disks, increasing the request width (D), also increases the sub-request arrival rate. Since the decrease in the service time cannot compen-

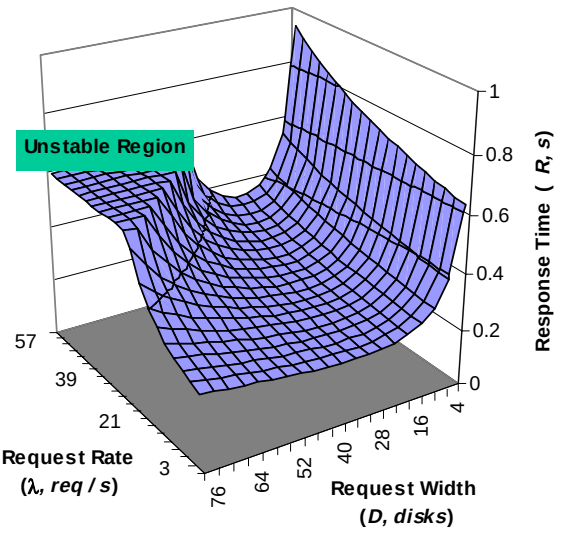


Figure 6. Request response time as a function of request rate.

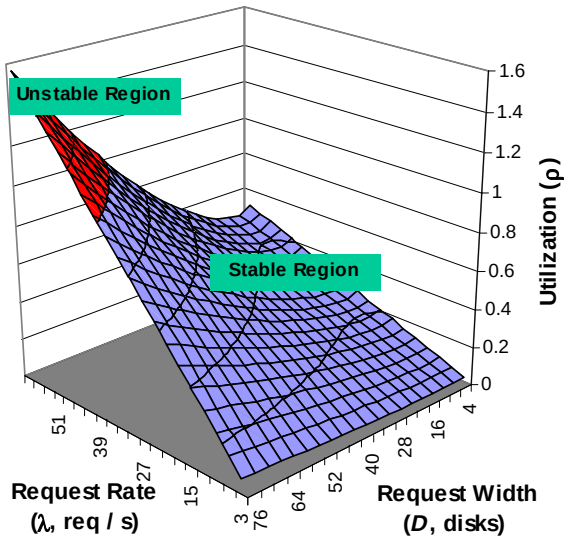


Figure 7. Disk utilization as a function of request rate.

sate for the increase in the request rate, after a certain point the response time increases very rapidly. Note that smaller request arrival rates can tolerate bigger request widths.

The area labeled "Unstable Region" in Figure 6 denotes the region where the values of request width and the request rate cause the utilization,

$$\rho \quad = \quad \lambda_D S \tag{9}$$

for an M/G/1 queue, to be greater than one. An M/G/1 queue is unstable when $\rho > 1$, and this is considered an undesirable situation. It basically means that the system is not able to keep up with the request arrival rate any more and the queues will grow indefinitely. For the same set of parameters, the utilization, or traffic intensity, is plotted in Figure 7, where the regions considered stable ($\rho < 1$) and unstable ($\rho > 1$) are labeled as such. As Figures 6 and 7suggest, disk utilization quickly becomes very high with increasing request rates.

Similar behavior is repeated if we vary the request size, which basically increases the service time, and this is illustrated in Figure 8. As Figures 6 and 8 show, the request width can and must be chosen optimally, considering the request and input/output subsystem parameters.

If we take the derivative of the Equation 8 with respect to D, and solve

$$\frac{\partial\, R_{max_D}}{\partial\, D} \quad = \quad 0 \tag{10}$$

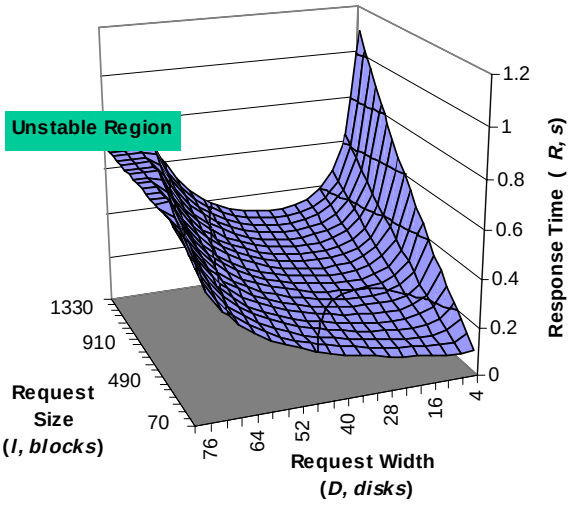


Figure 8. Request response time as a function of request size.

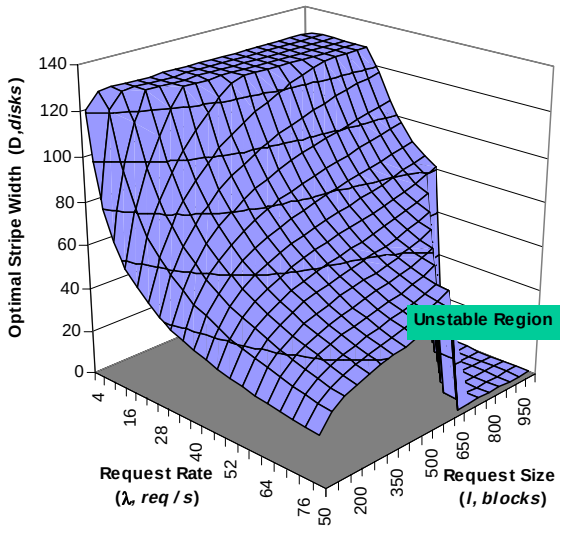


Figure 9. Optimal request width for the distributed striping model.

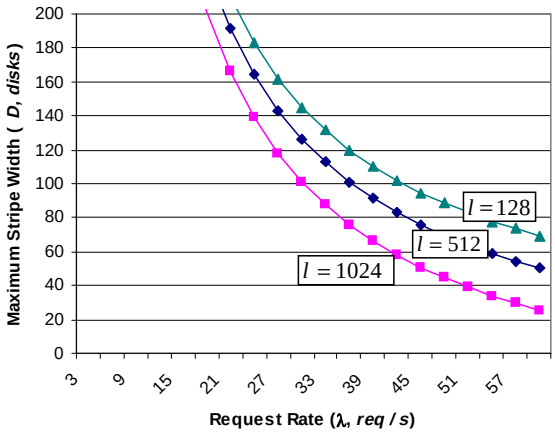


Figure 10. Maximum request width with respect to request rates.

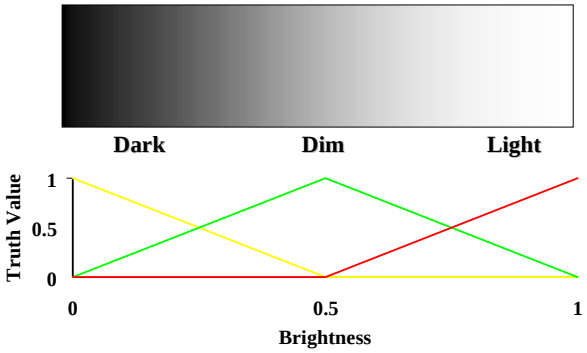


Figure 11. Fuzzy variable *Brightness*.

for D, we can find an optimal request width which minimizes the response time. Optimal request width is plotted in Figure 9 with respect to request arrival rate and mean request size. Larger request sizes allow bigger request widths, while higher request arrival rates dictate smaller request widths.

If we solve Equation 9 for D, we can compute the request width needed to obtain a utilization of ρ'. Such a solution is given by

$$D = \frac{(-cl\lambda_m - il\lambda_m t + m\rho' t)}{(c + f + k)\,\lambda_m t}. \qquad (11)$$

Figure 10 plots the request width for varying request rates when $\rho' = 1$, using Equation 11. This figure actually shows the maximum allowable request width for a given request rate and request size pair. Any request width greater than the corresponding value would cause the system to be unstable. This upper bound on the request width can be used for initialization of a Newton like optimization method to find an optimal request width.

5 Fuzzy logic adaptive control

Classical control techniques and decision tables/trees require in-depth knowledge of the control domain. They also depend on consistent parameter-space division. In contrast to these techniques, fuzzy logic control has characteristics that make it an excellent choice for problems such as input/output systems — it can handle imprecise system definition and conflicting goals. A fuzzy logic control system can use common sense knowledge about the domain by manipulating linguistically described concepts [42, 43, 44].

In fuzzy logic, the semantic properties of the system variables are represented with a collection of *fuzzy sets*. Figure 11 illustrates the fuzzy sets (concepts) corresponding to the fuzzy variable *Brightness*, together with membership functions that define the transitions between these concepts. At the right end of the spectrum, membership function LIGHT has a truth value of one. As the *Brightness* decreases, the truth value of LIGHT decreases linearly and the truth value of DIM increases. At the left end, only DARK is true and others are fully false. Definition of these fuzzy sets, together with a set of `IF - THEN` rules, constitutes the *knowledge base* of the fuzzy controller.

Such a knowledge base for the shutter speed of a

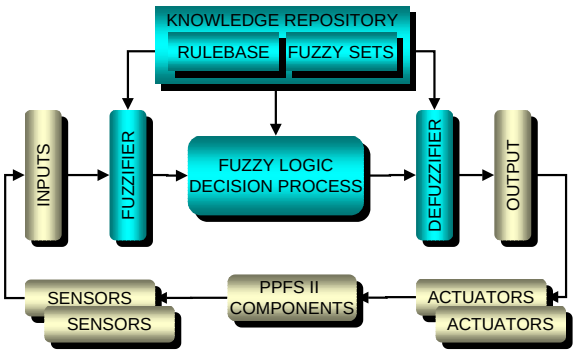


Figure 12. PPFS II fuzzy controller architecture.

camera might contain rules like:

```
IF (Brightness == LIGHT) THEN
   ShutterSpeed =  FAST

IF (Brightness == DARK) THEN
   ShutterSpeed =  SLOW
```

Fuzzy operators, counterparts of boolean operators, are used to combine input fuzzy values. For example, a fuzzy OR operator might take the maximum of the two input fuzzy sets.

In this example, first the truth values of the antecedents of each rule are computed. Then, the consequents are scaled with the resulting value. In this way, multiple rules may generate outputs for the same input. At the end, outputs of all rules with the same consequent are joined to obtain a single output.

Figure 12 illustrates the flow logic of a fuzzy controller as implemented in PPFS II. The functions of the model components are as follows:

Knowledge repository. Contains the fuzzy production rules and the definition of the fuzzy sets. This is static information in this model. A more complex and adaptive system can be obtained by providing a feedback loop from the system outputs to the knowledge repository to tune the fuzzy sets and to modify the rules.

Inputs. Inputs are gathered from the system sensors and pre-processed to serve the needs of the fuzzy logic system.

Fuzzifier. The inputs are "fuzzified." That is, the values are converted to their fuzzy representations through the information taken from the knowledge repository.

Fuzzy logic decision process. This process executes all the rules in the knowledge repository that have the fuzzified input in their premise, resulting in a new, fuzzy set representation for each output variable.

Defuzzifier. Representative scalar values are obtained from output fuzzy sets using a defuzzification method (e.g., centroid defuzzification).

Outputs. Outputs from the defuzzifier are post-processed, to transform them into usable control information, and then distributed to the actuators.

Actuators. These are software components in the system that allow dynamic control of the system components assigned to them.

Sensors. Sensors gather critical performance attributes of the system components and forward them to the fuzzy control logic.

In contrast to techniques like decision tables which cause discrete changes, fuzzy logic provides smooth transitions between policies. Its natural approach to domain definition allows quick experimentation and tuning of the control system.

5.1 Adaptive file striping rule base

Below we will describe a fuzzy system module that determines the best striping unit adaptively, considering the current state of the system. This rule base is based on the parametric studies of the analytic striping models presented in §4.

The fuzzy variables *RequestRate* and *RequestWidth*, can have the fuzzy values given in Figure 13. *RequestRate* has values INFREQUENT, OCCASIONAL, FREQUENT, and CONTINUOUS to denote the rate of request arrivals. The rate will be scaled to be in the interval $[0,1]$ to make the rules portable to various systems. We also normalize *RequestWidth* by the total number of devices, which makes *RequestWidth* to be in the interval $[0,1]$, also. Then, a *RequestWidth* of 1.0 denotes that all the devices are used. With this definition, the fuzzy values in Figure 13 can be used for the fuzzy linguistic variable *RequestWidth*. We have used only triangular and trapezoidal membership functions, because they allow fast and optimized fuzzy inferencing procedures. Other, smoother curves can be employed with a higher computing cost of manipulation.

As discussed in §4, the level of parallelism for individual requests must decrease as the system load,

```
if (RequestRate == INFREQUENT && RequestSize == TINY ){RequestWidth = SMALL;}
if (RequestRate == OCCASIONAL && RequestSize == TINY ){RequestWidth = TINY;}
if (RequestRate == FREQUENT   && RequestSize == TINY ){RequestWidth = TINY;}
if (RequestRate == CONTINUOUS && RequestSize == TINY ){RequestWidth = TINY;}

...

if (RequestRate == INFREQUENT && RequestSize == LARGE){RequestWidth = HUGE;}
if (RequestRate == OCCASIONAL && RequestSize == LARGE){RequestWidth = LARGE;}
if (RequestRate == FREQUENT   && RequestSize == LARGE){RequestWidth = SMALL;}
if (RequestRate == CONTINUOUS && RequestSize == LARGE){RequestWidth = TINY;}

...

if (NetworkPerformance == LOW) {RequestWidth = SMALL;}
if (NetworkPerformance == HIGH) {RequestWidth = LARGE;}

if (DiskPerformance == LOW) {RequestWidth = LARGE;}
if (DiskPerformance == HIGH) {RequestWidth = SMALL;}

if (FileParallelism == LOW) {StripeWidth = LARGE;}
if (FileParallelism == HIGH) {StripeWidth = SMALL;}

if (RequestRate == INFREQUENT) {FileReplicationTime = ONLINE;}
if (RequestRate == FREQUENT) {FileReplicationTime = OFFLINE;}
```

Figure 14. Fuzzy logic rule base for adaptive striping.

RequestRate in the models, increases. This establishes an inverse relation between *RequestWidth* and *RequestRate*. And, to be able to utilize multiple disks, the *RequestSize* must be big enough. These relations can be represented by the fuzzy rules given in Figure 14. The relation defined by the rule base can be used to adaptively determine the *RequestWidth*, which, in turn, dictates a striping unit with the relation

$$\text{StripingUnit} = \frac{\text{AverageRequestSize}}{\text{RequestWidth} \times \text{NumOfDisks}}. \quad (12)$$

Figure 15 depicts *RequestWidth* as a function of the *RequestSize* and *RequestRate* according to the rule base. As desired, increasing *RequestRate* dictates less striping, thus higher striping units. And along a constant *RequestRate* level, higher request sizes require higher request width, thus more parallelism. The shape of the graph is a function of the fuzzy sets, fuzzy rules and the fuzzy methods used.

One thing to note is that the decisions obtained from fuzzy controllers may be considered as suggestions. Their applicability can be further tested by the actuators responsible for the files in question. For example, the actuator can round up the striping unit to a multiple of the access granularity of the device.

6 Experimental infrastructure

As a testbed for our adaptive file system policies, we have designed and completed a prototype of a next generation portable parallel file system, PPFS II. It provides a real-time, file system adaptation environment on top of both clustered PCs and traditional parallel computers.

PPFS II uses *Autopilot* [45, 46], a closed-loop, performance measurement and adaptive control system. *Autopilot* provides lightweight software sensors to capture the performance data from system components and actuators to manipulate software behavior. Even though sensors provide qualitative data to enable adaptive decision making, earlier research [33] has shown that using qualitative classification data about the access patterns enables better optimizations. In PPFS II, this information can be obtained by user supplied hints or via automatic classification techniques.

PPFS II utilizes a set of trained, artificial, neural networks (ANNs) [47] to classify application input/output patterns. These automatic classifiers are implemented as *Autopilot* sensors which distribute the information to connected decision procedures. To maintain rule base portability, the user hints and the classification information obtained through sensors are treated as clues or suggestions. They do not directly effect system operation. Rather, decision pro-

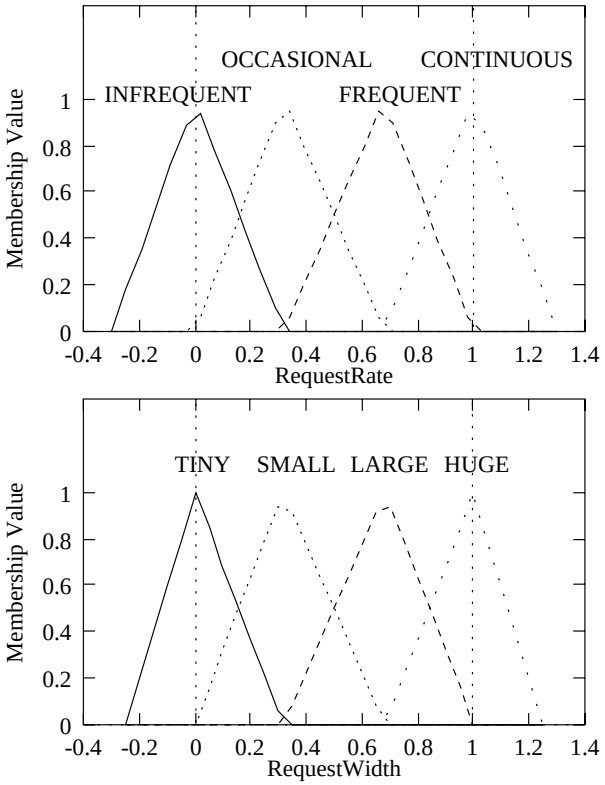


Figure 13. Fuzzy linguistic values for *RequestRate* and *RequestWidth*.

cedures interpret them to arrive at policy decisions.

PPFS II implements distributed decision procedures in the form of fuzzy logic rule bases. Based on system- and user-level, input/output resource usage information, these rule bases intelligently select file system policies. These policies can balance conflicting resource demands within and across applications and trade storage space for increased input/output bandwidth.

Figure 16 illustrates the high-level component architecture of PPFS II prototype. All file system components are connected via logical links provided Globus/Nexus [48] that allow location transparent system operation. Metadata manager keeps the component addresses and data layout information. Sensor/actuator manager is the central place for publishing and querying Autopilot sensors and actuators.

To develop and experiment with PPFS II, we have set up a cluster of PCs. The cluster consists of nine Dell Dimension XPS PCs with Pentium II 266 MHz processors and 64 MB main memories. The cluster is connected with a 100 Mbit/sec switched Fast Ethernet network which provides 3.2 Gbit/sec peak crossbar bandwidth. Each PC in the cluster contains three 4.3 GB disks connected to an Ultra Wide SCSI host

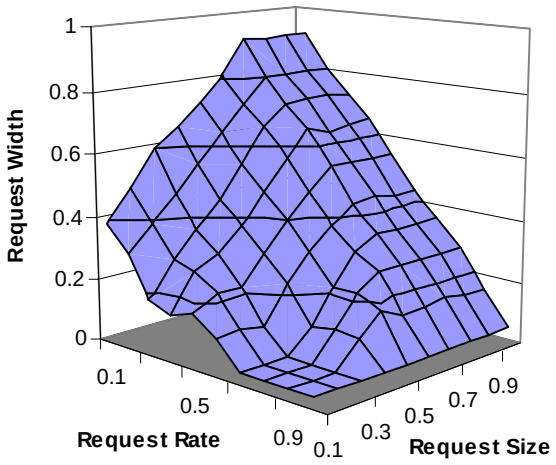


Figure 15. Optimal request width obtained from the rule base.

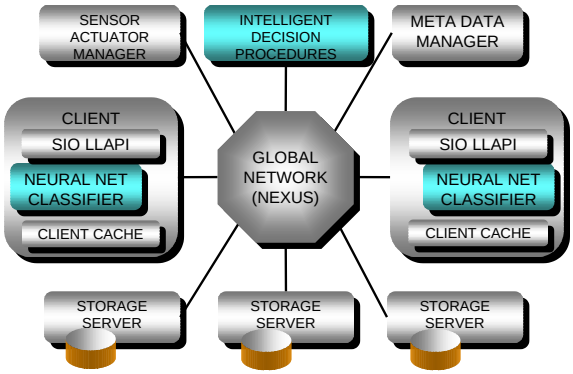


Figure 16. PPFS II distributed grid architecture.

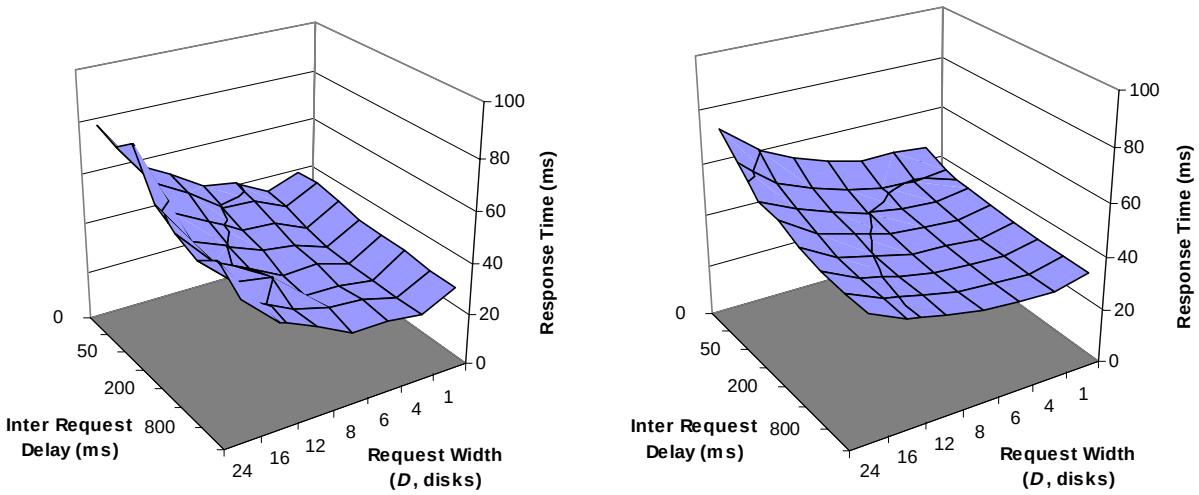
adaptor. This results in 24 storage servers in the cluster, one for each disk.

7 Experimental results

In this section, we will present preliminary results we have obtained on our experimental platform with the PPFS II prototype.

7.1 PPFS II disk striping performance

To experiment with the PPFS II striping performance, we used a suite of SIO Low-Level API [29] benchmarks. These configurable benchmarks are flexible enough to generate many of the common, parallel input/output access patterns seen in earlier



(a) Experimental

(b) Queuing model prediction

Figure 18. Response times for 64 KB read requests.

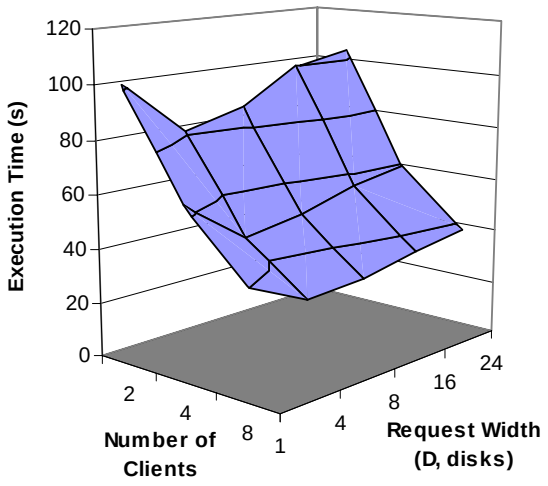


Figure 17. Total execution time to read 1 GB with 512 KB requests.

characterization studies.

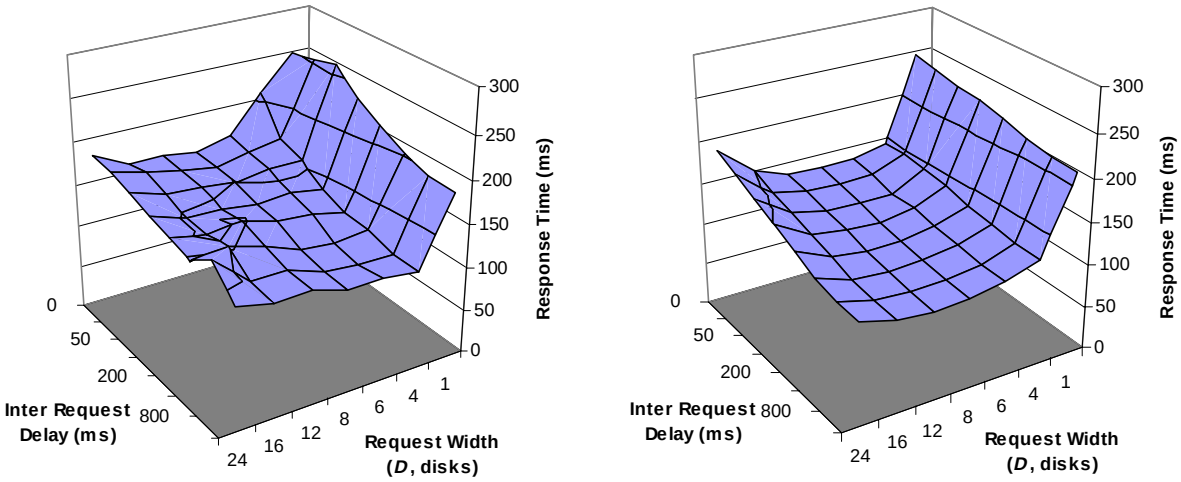
First, to assess the striping performance of the experimental platform, we ran a series of benchmarks with varying numbers of clients, which is illustrated in Figure 17. In each case, the corresponding number of clients are reading a 1 GB file collectively, with 512 KB blocks, and with random offsets at 512 KB boundaries. The figure presents the execution time with respect to the request width, the number of disks used for a particular request, and the number of clients used in the experiment. Using multiple clients reduces the execution time because of parallel accesses. The figure shows that the overall execution time is affected by the choice of the request width. For this experimental setting, a request width of four disks provides the optimal execution time.

7.2 Striping model verification experiments

To validate our analytical models of striping, we ran a series of benchmarks with varying request arrival rates and request widths. The results are presented in Figures 18 and 19, which show the effects of different request widths on the response time of individual requests. In these experiment, eight nodes are reading a one gigabyte file Requests on each client are separated in time with the inter-request delays shown in figures. Figures 18 and 19 also present the corresponding response time predictions obtained using the analytical striping model presented in §4. For these plots, the model parameter values given in Table 2 are used after adjusting the request size parameters. These figures show that the analytical striping model can predict the result of using different striping parameters. The average difference between the actual and predicted response times is 11% in Figure 18 and 13% in Figure 19.

8 Conclusions

Our preliminary experiments have shown that automatic, rule based, adaptive control of input/output



(a) Experimental

(b) Queuing model prediction

Figure 19. Response times for 512 KB read requests.

subsystem can provide significant performance improvements. Analysis of input/output behavior on our experimental platform also proved the importance of striping parameters in the face of changing system conditions. Our striping rule base is based on an analytical model, which is shown to be accurate in predicting experimental outcomes.

In the future, PPFS II will utilize Hidden Markov models (HMMs) to build a probabilistic model of the access pattern using prior execution training. This generality will allow automatic classification of arbitrary access patterns.

Finally, the closed-loop and interactive performance steering techniques developed for PPFS II will not only benefit the input/output subsystem, but they can also be applied to other subsystems that exhibit dynamic behavior in a computational grid.

Acknowledgments

PPFS II and *Autopilot* are the result of the collective work of a number of past and present researchers in the Pablo Research Group, most notably Ruth Aydt, Ryan Fox, Mario Medina, James Oly, Randy Ribler, Nancy Tran, and Guoyi Wang.

References

[1] M. Holton and R. Das, *XFS: A Next Generation Journalled 64-Bit Filesystem With Guaranteed Rate I/O*. Silicon Graphics, Inc.

[2] IBM Corp., *GPFS: A Parallel File System*, April 1998.

[3] NSF Terascale Computing Initiative, "Terascale and Petascale Computing: Digital Reality in the New Millenium." `http://rrbhpnt.asc.cise-nsf.gov/NSFReport.htm`.

[4] P. H. Smith and J. V. Rosendale, "Data and Visualization Corridors," Tech. Rep. CACR-164, CACR, CALTECH, September 1998.

[5] K. Salem and H. Garcia-Molina, "Disk Striping," in *Proceedings of the 2nd International Conference on Data Engineering*, pp. 336–342, ACM, Feb. 1986.

[6] T. Sterling, P. Messina, and P. H. Smith, *Enabling Technologies for Petaflops Computing*. MIT Press, 1995.

[7] J. V. Huber, C. L. Elford, D. A. Reed, A. A. Chien, and D. S. Blumenthal, "PPFS: A High-Performance Portable Parallel File System," in *Proceedings of the 9th ACM International Conference on Supercomputing*, pp. 385–394, July 1995.

[8] D. A. Reed, C. L. Elford, T. Madhyastha, W. H. Scullin, R. A. Aydt, and E. Smirni, "I/O, Performance Analysis, and Performance Data Immersion," in *Proceedings of MASCOTS '96*, pp. 1–12, Feb. 1996.

[9] H. Simitci and D. A. Reed, "A Comparison of Logical and Physical Parallel I/O Patterns," *International Journal of High Performance Computing Applications*, vol. 12, no. 3, pp. 364–380, 1998.

[10] E. Smirni and D. A. Reed, "Workload Characterization of Input/Output Intensive Parallel Applications," in *Proceedings of the 9th International Conference on Modelling Techniques and Tools for*

Computer Performance Evaluation, pp. 169–180, Springer-Verlag, June 1997.

[11] D. Patterson, P. Chen, G. Gibson, and R. H. Katz, "Introduction to Redundant Arrays of Inexpensive Disks (RAID)," in *Proceedings of IEEE Compcon*, pp. 112–117, Spring 1989.

[12] J. H. Hartman and J. K. Ousterhout, "The Zebra Striped Network File System," *ACM Transactions on Computer Systems*, vol. 13, pp. 274–310, Aug. 1995.

[13] P. C. Dibble, M. L. Scott, and C. S. Ellis, "Bridge: A High-Performance File System for Parallel Processors," in *Proc. 8th Int'l. Conf. on Distr. Computing Sys.*, (San Jose, CA), pp. 154–161, jun 1988.

[14] T. H. Cormen and D. Kotz, "Integrating Theory and Practice in Parallel File Systems," in *Proceedings of the 1993 DAGS/PC Symposium*, (Hanover, NH), pp. 64–74, Dartmouth Institute for Advanced Graduate Studies, June 1993.

[15] J. S. Vitter and E. A. M. Shriver, "Algorithms for parallel memory I: Two-level memories," *Algorithmica*, vol. 12, pp. 110–147, Aug.andSept. 1994.

[16] P. M. Chen and D. A. Patterson, "Maximizing Performance in a Striped Disk Array," in *Proceedings of the 17th Annual International Symposium on Computer Architecture*, pp. 322–331, 1990.

[17] C. L. Elford and D. A. Reed, "Technology Trends and Disk Array Performance," *Journal of Parallel and Distributed Computing*, vol. 46, pp. 136–147, 1997.

[18] P. Scheuermann, G. Weikum, and P. Zabback, "Data Partitioning and Load Balancing in Parallel Disk Systems," *The VLDB Journal*, vol. 7, pp. 48–66, Feb. 1998.

[19] P. Chen, E. Lee, G. Gibson, R. Katz, and D. Patterson, "RAID: High-Performance, Reliable Secondary Storage," *ACM Computing Surveys*, vol. 26, pp. 145–185, June 1994.

[20] D. Bitton and J. Gray, "Disk Shadowing," in *Proceedings of the 14th International Conference on Very Large Data Bases*, pp. 331–338, 1988.

[21] O. Wolfson, S. Jajodia, and Y. Huang, "An Adaptive Data Replication Algorithm," *ACM Transactions on Database Systems*, vol. 22, pp. 255–314, June 1997.

[22] O. Wolfson and A. Milo, "The Multicast Policy and its Relationship to Replicated Data Placement," *ACM Transactions on Database Systems*, vol. 16, pp. 181–205, Mar. 1991.

[23] C. Ruemmler and J. Wilkes, "An Introduction to Disk Drive Modeling," *Computer*, vol. 27, pp. 17–28, Mar. 1994.

[24] E. Shriver, A. Merchant, and J. Wilkes, "An Analytic Behavior Model for Disk Drives with Readahead Caches and Request Reordering," in *Proceedings of SIGMETRICS'98*, pp. 182–191, 1998.

[25] J. Wilkes, B. L. Worthington, G. R. Ganger, and Y. N. Patt, "On-line Extraction of SCSI Disk Drive Parameters," in *Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems*, pp. 146–156, May 1995.

[26] P. E. Crandall, R. A. Aydt, A. A. Chien, and D. A. Reed, "Input/Output Characteristics of Scalable Parallel Applications," in *Proceedings of Supercomputing '95*, (San Diego, CA), pp. CD–ROM, IEEE Computer Society Press, Dec. 1995.

[27] E. Smirni, R. A. Aydt, A. A. Chien, and D. A. Reed, "I/O Requirements of Scientific Applications: An Evolutionary View," in *Proceedings of the Fifth IEEE International Symposium on High-Performance Distributed Computing*, pp. 49–59, Aug. 1996.

[28] E. Smirni, C. L. Elford, and D. A. Reed, "Performance Modeling of a Parallel I/O System: An Application Driven Approach," in *Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing*, Mar. 1997.

[29] P. F. Corbett, J.-P. Prost, C. Demetriou, G. Gibson, E. Riedel, J. Zelenka, Y. Chen, E. Felten, K. Li, J. Hartman, L. Peterson, B. Bershad, A. Wolman, and R. Aydt, "Proposal for a Common Parallel File System Programming Interface Version 1.0." `http://www.cs.arizona.edu/sio/api1.0.ps`, Nov. 1996.

[30] The MPI-IO Committee, "MPI-IO: A Parallel File I/O Interface for MPI," April 1996. Version 0.5.

[31] Intel Corporation, Intel SSD, Beaverton, OR, *Paragon System User's Guide*, 1995.

[32] P. F. Corbett and D. G. Feitelson, "The Vesta Parallel File System," *ACM Transactions on Computer Systems*, vol. 14, pp. 225–264, Aug. 1996.

[33] T. M. Madhyastha, C. L. Elford, and D. A. Reed, "Optimizing Input/Output Using Adaptive File System Policies," in *Proceedings of the Fifth Goddard Conference on Mass Storage Systems and Technologies*, pp. II:493–514, sep 1996.

[34] T. M. Madhyastha and D. A. Reed, "Input/Output Access Pattern Classification Using Hidden Markov Models," in *Proceedings of the Fifth Workshop on Input/Output in Parallel and Distributed Systems*, (San Jose, CA), pp. 57–67, ACM Press, Nov. 1997.

[35] T. M. Madhyastha and D. A. Reed, "Intelligent, Adaptive File System Policy Selection," in *Proceedings of the Sixth Symposium on the Frontiers of Massively Parallel Computation*, pp. 172–179, IEEE Computer Society Press, Oct 1996.

[36] D. A. Reed, C. L. Elford, T. Madhyastha, E. Smirni, and S. L. Lamm, "The Next Frontier: Interactive and Closed Loop Performance Steering," in *Proceedings of the 1996 International Conference on Parallel Processing Workshop*, pp. 20–31, August 1996.

[37] J. Vetter and K. Schwan, "High Performance Computational Steering of Physical Simulations," in *Proc. Int'l Parallel Processing Symp.*, (Geneva), pp. 128–132, 1997.

[38] L. A. Zadeh, "Fuzzy Sets," *Information and Control*, vol. 8, pp. 338–353, June 1965.

[39] N. K. Kasabov, *Foundations of Neural Networks, Fuzzy Systems, and Knowledge Engineering*. The MIT Press, 1996.

[40] R. Jain, *The Art of Computer Systems Performance Analysis*. Wiley, 1991.

[41] M. Y. Kim and A. N. Tantawi, "Asynchronous Disk Interleaving: Approximating Access Delays," *IEEE Transactions on Computers*, vol. 40, pp. 801–810, July 1991.

[42] L. A. Zadeh, "A Fuzzy-Algorithmic Approach to the Definition of Complex or Imprecise Concepts," *International Journal of Man-Machine Studies*, vol. 8, no. 3, pp. 249–291, 1976.

[43] L. Zadeh, "Commonsense Knowledge Representation Based on Fuzzy Logic," *IEEE Computer*, vol. 16, p. 61, Oct. 1983.

[44] L. A. Zadeh, "Fuzzy Logic = Computing with Words," *IEEE Transactions on Fuzzy Systems*, vol. 4, no. 2, pp. 103–111, 1996.

[45] R. L. Ribler, J. S. Vetter, H. Simitci, and D. A. Reed, "Autopilot: Adaptive Control of Distributed Applications," in *Proceedings of HPDC7*, July 1998.

[46] D. A. Reed and R. L. Ribler, "Performance Analysis and Visualization," in *Computational Grids: State of the Art and Future Directions in High-Performance Distributed Computing*, Morgan-Kaufman, Aug. 1998.

[47] T. M. Madhyastha and D. A. Reed, "Exploiting Global Input/Output Access Pattern Classification," in *Proceedings of SC '97: High Performance Computing and Networking*, (San Jose), IEEE Computer Society Press, Nov. 1997.

[48] I. Foster and C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit," *International Journal of Supercomputing Applications and High Performance Computing*, vol. 11, pp. 115–128, Summer 1997.