Disk Array Models in Minerva

Arif Merchant, Guillermo A. Alvarez
Computer Systems and Technology Laboratory
HP Laboratories Palo Alto
HPL-2001-118
May 7th , 2001*

E-mail: {arif, galvarez} @hpl.hp.com

storage systems, disk arrays, analytical models

Enterprise storage systems typically depend on disk arrays to satisfy their capacity and availability needs. To design and maintain storage systems that efficiently satisfy evolving requirements, it is critical to be able to evaluate configuration alternatives without having to physically implement them. Because of the large number of candidate configurations that need to be evaluated in real-life situations, simulation models are excessively slow for that task. In this paper, we describe analytical throughput models for RAID 1/0 and RAID 5 storage in the Hewlett-Packard FC-30 disk array. We validate our models against the real array, and report the relative errors in the models' predictions. Our models have a mean error of 5.4% and a maximum error of 19%, for the set of validations workloads we used.

Disk array models in Minerva

Arif Merchant[*], Guillermo A. Alvarez[*]

Abstract

Enterprise storage systems typically depend on disk arrays to satisfy their capacity and availability needs. To design and maintain storage systems that efficiently satisfy evolving requirements, it is critical to be able to evaluate configuration alternatives without having to physically implement them. Because of the large number of candidate configurations that need to be evaluated in real-life situations, simulation models are excessively slow for that task. In this paper, we describe analytical throughput models for RAID 1/0 and RAID 5 storage in the Hewlett-Packard FC-30 disk array. We validate our models against the real array, and report the relative errors in the models' predictions. Our models have a mean error of 5.4% and a maximum error of 19%, for the set of validations workloads we used.

1 Introduction

Enterprise-scale computer installations are extremely difficult to design and configure. They can contain tens to hundreds of host computers, connected by a Storage Area Network (SAN) such as FibreChannel [3] or Gigabit Ethernet [1], to tens to hundreds of storage devices, with up to tens of thousands of disks and logical volumes. Total capacities in the tens of terabytes are becoming common. The complexity of configuring the storage system is compounded as individual storage devices such as disk arrays usually have many parameter settings of their own. System administrators are ill-suited to deal with this complexity, because of the sheer size of the solution space and also because of the subtle interactions between the consequences of configuration decisions.

The MINERVA tool [2] solves this problem by designing and configurating storage systems without human assistance. It has been shown to generate, in at most a few hours of computer time, system designs that compare advantageously to the ones generated by humans by hand. An automated system can explore the space of storage configurations and the complex interactions between parts of the workload and storage devices much more thoroughly than a person can. A typical run of MINERVA examines several millions of partially-built candidate configurations, while searching for the one with minimum cost that satisfies the quality of service requirements of client applications.

In order to converge to good solutions in an acceptable amount of computation time, an automated system needs to be able to obtain fast answers to "what if" questions. According to the performance predicted for each candidate configuration being considered, the system discards it or continues to build upon it. Physically building the system and measuring its performance is not an option, because of cost and of the number of candidate configurations being considered. Models based on simulation require big development efforts, and cannot generate answers in a few milliseconds' time. Because of this, we concentrated our modeling efforts on analytical models. Previously existing analytical models for disk arrays [7, 9] are not applicable to real, commercial arrays; their prediction errors are quite significant, because the simplifying assumptions made to render the models tractable fail to capture many aspects that impact real-word performance.

In this paper, we present analytical models for RAID 1/0 and RAID 5 storage in a commercial disk array. The models take as input a declarative, attribute-based specification of the workload being run on the array, and generate as output estimations of the total, steady-state throughput that the array is going to deliver for that workload. We validate our models against the real array, and report the relative errors in the models' predictions. Our models are substantially more accurate than pre-existing analytical array models: they have a mean error of 5.4% and a maximum error of 19%, for the set of validations workloads we used.

[*]{arif,galvarez}@hpl.hp.com, Hewlett-Packard Laboratories, 1U-13, 1501 Page Mill Rd., Palo Alto, CA 94304, USA

The remainder of this paper is organized as follows. We describe the array and layouts being modeled and the throughput models in Section 2. We then describe our validation experiments and show the results in Section 3. Section 4 concludes the paper.

2 The models

Our models target a mid-range array containing one or more array controllers, an interconnection network that connects the controllers with the disks, and a set of disks. The array can be configured with any mix of RAID 5 and RAID 1/0 LUs.

2.1 Workload model

The workload for a LU is modelled as a set of correlated ON -OFF streams each with some spatial locality. Each stream S_i is alternately OFF for a period of mean length $\mathsf{off\_time}(S_i)$ and ON for a period of mean length $\mathsf{on\_time}(S_i)$. While ON , the stream is a Poisson arrival process with a mean request rate $\mathsf{request\_rate}(S_i)$ and a mean request size $\mathsf{request\_size}(S_i)$. A request is a read with probability $\mathsf{read\_probability}(S_i)$ and a write otherwise. Requests from a stream arrive in *runs* — sets of spatially sequential requests. The mean number of consecutive sequential requests for S_i is $\mathsf{run\_count}(S_i)$.

We also assume that the ON -OFF structure of the streams is correlated. When stream S_i switches from OFF to ON, the probability that stream S_j is already ON is p_{ij}.

The values of the parameters for each stream are provided as a part of the workload specification. They can be measured from an I/O trace of the workload.

In the model descriptions below, we present the equations for the case when all the streams are ON continuously; we will indicate how to modify these equations to handle the case of *phased* (correlated) streams in Section 2.6.

2.2 Controller and interconnection network model

We model the restriction due to the controller and interconnection network through a pair of constraints: a *bandwidth* constraint and a *throughput* constraint. The bandwidth constraint models a limit on the number of bytes that can be passed per second through the controllers and the interconnection network.

$$\sum_i \mathsf{request\_rate}(S_i)\mathsf{request\_size}(S_i) \leq \mathsf{max\_controller\_bandwidth}. \tag{1}$$

The throughput constraint models a limit on the number of requests per second that can be passed through the controllers and the interconnection network.

$$\sum_i \mathsf{request\_rate}(S_i) \leq \mathsf{max\_controller\_throughput}. \tag{2}$$

We assume that the controller cache does not affect the maximum throughput of the array; this is a reasonable assumption when the cache size is small compared to the overall amount of storage in the array, since the effects of temporal locality are likely to be small.

2.3 Disk model

The service time of a request at a disk is the sum of positioning and transfer times. We assume that the first request in a sequential run incurs a positioning time equal to the mean positioning time for a random request on the disk, while the remaining requests in the run incur no positioning time at all. Based on this, the mean disk service time for a stream with a mean request size of $\mathsf{request\_size}$ and run count at the disk of $\mathsf{disk\_run\_count}$ is $\mathsf{mean\_position\_time}/\mathsf{disk\_run\_count} + \mathsf{request\_size}/\mathsf{transfer\_rate}$, where $\mathsf{mean_position_time}$and $\mathsf{transfer_rate}$are disk-specific parameters.

The above service time is modified by the presence of multiple streams, since requests from a different stream may intervene between requests in a sequential run, thus breaking it into multiple smaller runs. Denote the request rate of stream S_i arriving at a single, given disk as $\mathsf{disk\_request\_rate}(S_i)$. Assuming that the run lengths are geometrically distributed with the given mean and the total request rate from all streams is $\Lambda = \sum_j \mathsf{disk\_request\_rate}(S_j)$, it can be easily shown that the run length of the broken, smaller runs is

$$\mathsf{effective\_run\_count}(S_i) = \left(\frac{\mathsf{disk\_request\_rate}(S_i)}{\Lambda} \left(1 - \frac{1}{\mathsf{disk\_run\_count}(S_i)} \right) \right)^{-1}.$$

The utilization of the disk can then be calculated as

$$\mathsf{disk\_utilization} = \sum_i \mathsf{disk\_request\_rate}(S_i) \left(\frac{\mathsf{mean\_position\_time}}{\mathsf{effective\_run\_count}(S_i)} + \frac{\mathsf{request\_size}(S_i)}{\mathsf{transfer\_rate}} \right). \quad (3)$$

The above equations assume that each stream is continuously ON . The effects of ON -OFF phasing and correlation between streams can further be included in this model; how to do this is indicated in [4].

2.4 RAID 1/0 model

Consider a RAID 1/0 LU with $\mathsf{LU\_size}$ disks and $\mathsf{number\_streams}$ request streams; clearly $\mathsf{LU\_size}$ must be even. We treat RAID 1 as a special case of the RAID 1/0 model with $\mathsf{LU\_size} = 2$. We assume that each read request goes to one disk (one of the two that contains the data, with equal probability), and approximate that requests are not split across disks. Write requests go to both disks containing the data; again we assume that the requests are not split across disks. The requests are assumed to go to all mirrored pairs in the LU with equal probability. Thus, the load on all disks in the LU is identical.

For any given workload (combination of request streams), we say that the workload is feasible if all the disks in the LU have a utilization less than one. Since the load on all the disks is identical, it suffices to check the utilization of any disk. The maximum throughput for the workload is found by increasing the request rates of all the streams proportionately until the utilization of the disk becomes equal to one.

The utilization of the disk is computed using the above disk model, in which the request rates of the streams at the disks are computed as

$$\mathsf{disk\_request\_rate}(S_i) = \frac{\mathsf{request\_rate}(S_i)(\mathsf{read\_probability}(S_i) + 2(1 - \mathsf{read\_probability}(S_i)))}{\mathsf{LU\_size}}.$$

This is based on the fact that each write request generates two corresponding disk writes, while a read request generates only one.

2.5 RAID 5 model

The RAID 5 model in Minerva is more sophisticated than the RAID 1/0 model, mainly because it is newer and incorporates a better understanding of the array architecture. As in the case of the RAID 1/0 model, the utilization of the LU is found by assuming that all disks are evenly loaded, and computing the utilization of a single disk. The computation of the request rate a stream presents to the disk, however, is quite different.

We assume that each sequential run consists entirely of read, or entirely of writes. For the purpose of modelling, we treat the entire run as a single request, based on the assumption that the array controller optimizes disk accesses by coalescing a run of requests going to the disk into one request.

Data write requests generate both disk writes and reads (for computing the parity), whereas data read requests generate disk reads only.

The rate at which read runs arrive at the array from stream S_i is

$$\mathsf{read\_run\_rate}(S_i) = \frac{\mathsf{request\_rate}(S_i)\mathsf{read\_probability}(S_i)}{\mathsf{run\_count}(S_i)}.$$

The mean number of stripe units touched by the read run is disks touched by the read run is

$$1 + \frac{\mathsf{run\_count}(S_i)\mathsf{request\_size}(S_i)}{\mathsf{stripe\_unit\_size}}.$$

Therefore the number of disks touched can be approximated as

$$\mathsf{disks\_read\_per\_read\_run}(S_i) = \min\left(\mathsf{LU\_size}, 1 + \frac{\mathsf{run\_count}(S_i)\mathsf{request\_size}(S_i)}{\mathsf{stripe\_unit\_size}}\right).$$

We can now approximate the amount of data read from a single disk during the run as the length of the entire run divided by the number of disks involved.

$$\mathsf{disk\_read\_size\_from\_reads}(S_i) = \frac{\mathsf{run\_count}(S_i)\mathsf{request\_size}(S_i)}{\mathsf{disks\_read\_per\_read\_run}(S_i)}.$$

Also, the disk read rate per disk due to data reads from stream S_i is

$$\mathsf{disk\_read\_rate\_from\_reads}(S_i) = \frac{\mathsf{read\_run\_rate}(S_i)\mathsf{disks\_read\_per\_read\_run}(S_i)}{\mathsf{LU\_size}}.$$

The rate at which write runs arrive at the array from S_i is

$$\mathsf{write\_run\_rate}(S_i) = \frac{\mathsf{request\_rate}(S_i)(1 - \mathsf{read\_probability}(S_i))}{\mathsf{run\_count}(S_i)}.$$

If the write run size is smaller than a stripe unit, it generates two reads and two writes of the same size in order to write the data and to compute and write the new parity. If the write run is larger, the amount of data actually written depends on how the data aligns with the stripe boundary. The mean amount of data written can be approximated as

$$\begin{aligned}&\mathsf{data\_written\_per\_write\_run}(S_i)\\ &= \begin{cases} 2\mathsf{run\_count}(S_i)\mathsf{request\_size}(S_i) & \text{if } \mathsf{run\_count}(S_i)\mathsf{request\_size}(S_i) < \mathsf{stripe\_unit\_size} \\ \frac{((\mathsf{LU\_size}-2)\mathsf{stripe\_unit\_size}+\mathsf{LU\_size}\mathsf{run\_count}(S_i)\mathsf{request\_size}(S_i))}{(\mathsf{LU\_size}-1.0)} & \text{otherwise.}\end{cases}\end{aligned}$$

As in the case of read runs, the mean number of disks touched by writes due to a write run is

$$\mathsf{disks\_written\_per\_write\_run}(S_i) = \min(\mathsf{LU\_size}, 1 + \mathsf{data\_written\_per\_write\_run}(S_i)/\mathsf{stripe\_unit\_size}).$$

Since each write run generates $\mathsf{disks\_written\_per\_write\_run}(S_i)$ disk reads distributed over the $\mathsf{LU\_size}$ disks in the LU, the disk write rate due to S_i is

$$\mathsf{disk\_write\_rate}(S_i) = \frac{\mathsf{disks\_written\_per\_write\_run}(S_i)\mathsf{write\_run\_rate}(S_i)}{\mathsf{LU\_size}} \tag{4}$$

The average amount of data written per disk can be approximated as

$$\mathsf{disk\_write\_size}(S_i) = \frac{\mathsf{data\_written\_per\_write\_run}(S_i)}{(1 + \mathsf{data\_written\_per\_write\_run}(S_i)/\mathsf{stripe\_unit\_size})}. \tag{5}$$

The number of disk reads generated by a write run depends upon whether the run is smaller than a stripe unit, as described above, whether it is smaller than $(\mathsf{LU\_size} - 2)\mathsf{stripe\_unit\_size}/2$, in which case it depends on the alignment of the run with the stripe units, or whether it is larger, in which case the *large write* optimization can be used. In the latter case, the parity data is generated by reading the portion of a stripe which is *not* being written, instead of reading the old value of the data being modified. The resulting expression for the amount of data read is

$$\mathsf{data\_read\_per\_write\_run}(S_i)$$
$$= \begin{cases} 2\mathsf{run\_count}(S_i)\mathsf{request\_size}(S_i) \\ \qquad\qquad \text{if } \mathsf{run\_count}(S_i)\mathsf{request\_size}(S_i) < \mathsf{stripe\_unit\_size} \\ \frac{((\mathsf{LU\_size}-2)\mathsf{stripe\_unit\_size}+\mathsf{LU\_sizerun\_count}(S_i)\mathsf{request\_size}(S_i))}{(\mathsf{LU\_size}-1)} \\ \qquad\qquad \text{if } \mathsf{run\_count}(S_i)\mathsf{request\_size}(S_i) < (\mathsf{LU\_size}-2)/2\mathsf{stripe\_unit\_size} \\ \frac{(\mathsf{LU\_size}^2-4)\mathsf{stripe\_unit\_size}}{(2(\mathsf{LU\_size}-1))} \\ \qquad\qquad \text{otherwise.} \end{cases}$$

The number of disks touched by reads per write run can be approximated as

$$\mathsf{disks\_read\_per\_write\_run}(S_i) = \min(1 + \mathsf{data\_read\_per\_write\_run}(S_i)/\mathsf{stripe\_unit\_size}, \mathsf{LU\_size}).$$

The mean size of the reads generated due to writes is

$$\mathsf{disk\_read\_size\_from\_writes}(S_i) = \mathsf{data\_read\_per\_write\_run}(S_i)/\mathsf{disks\_read\_per\_write\_run}(S_i)$$

and the disk read rate generated by writes is

$$\frac{\mathsf{disk\_read\_rate\_from\_writes}(S_i) = \mathsf{disks\_read\_per\_write\_run}(S_i)\mathsf{write\_run\_rate}(S_i)}{\mathsf{LU\_size}.}$$

The overall disk read rate is, therefore,

$$\mathsf{disk\_read\_rate}(S_i) = \mathsf{disk\_read\_rate\_from\_reads}(S_i) + \mathsf{disk\_read\_rate\_from\_writes}(S_i). \qquad (6)$$

The overall mean disk read size is

$$\mathsf{disk\_read\_size}(S_i) = \qquad (7)$$
$$\frac{\mathsf{disk\_read\_size\_from\_reads}(S_i)\mathsf{disk\_read\_rate\_from\_reads}(S_i) + \mathsf{disk\_read\_size\_from\_writes}(S_i)\mathsf{disk\_read\_rate\_from\_writes}(S_i)}{\mathsf{disk\_read\_rate\_from\_reads}(S_i) + \mathsf{disk\_read\_rate\_from\_writes}(S_i)}.$$

By substituting these disk read and write rates and sizes in the disk model, we can determine if any disk is saturated.

2.6 ON -OFF phasing

It has been shown in [4] that, in order to verify that a performance constraint C holds at all times for a workload consisting of a set of ON -OFF streams S_1, S_2, …, S_n, it suffices to show that C holds in a short interval after S_i switches from OFF to ON , for each $i = 1, 2, \ldots, n$. During this short interval, we can assume that there are no transitions between the ON and OFF states.

Since we know that, at the instant S_i switches from OFF to ON , S_j is ON with probability p_{ij}, and if it is ON , the request rate is $\mathsf{request\_rate}(S_j)$, we approximate the request rate of S_j after S_i comes ON as $p_{ij}\mathsf{request\_rate}(S_j)$. We verify each constraint by using these request rates in the formulae of the sections above, for each possible value of i.

2.7 Calibration factors

There are many factors that affect the performance of an array which we do not model, such as the caching policies, the manner in which disk requests which are contiguous on the disk are coalesced and the re-ordering due to disk scheduling policies. In most cases, the details of these policies are not divulged by the manufacturer, nor is there a simple way to deduce them. As such, we do not expect that our model would fit any real array exactly. In order to improve the fit of the model to a given array, we use *calibration factors*. The request rates $\mathsf{request\_rate}(S_i)$ in the formulae above are replaced with $\mathsf{request\_rate}(S_i)\mathsf{calibration}(\mathsf{request\_size}(S_i), \mathsf{LU\_size}, \mathsf{LU\_type})$ where $\mathsf{calibration}(\cdot)$

Storage device	30-disk FC-30
Arrival time process	AFAP (as fast as possible)
Number of streams	At least 4 per disk
Number of LUNs	1
Disks / LUN	2 (for RAID 1) 4, 8 (for RAID 1/0) 4, 8 (for RAID 5)
Stripe unit size	64KB
Request size	8KB, 32KB, 64KB
Run count	1, 4, 8, 32
Read fraction	0, 1/4, 1/2, 3/4, 1

Table 1: *Workloads used to validate the analytic LUN models. Baseline values are underlined. The number of streams per disk was selected beforehand to nearly maximize throughput, and kept constant through all experiments.*

is a quadratic function of the request size and the number of disks in the LUN for each type of LUN (RAID 5, RAID 1 or RAID 1/0). The coefficients of this quadratic function are found by measuring maximum throughputs for a range of workloads with different sizes over LUNs of different sizes, and fitting the calibration function to minimize the mean squared error in the model over those points.

3 Model validation

In order to validate the models' predictions for a real disk array, we used the HP SureStore Model 30/FC High Availability (FC-30) disk array [5] as a case study.

An FC-30 supports two front-end controllers and up to 30 disks, which can be configured into up to 8 LUNs. Each LUN uses a RAID 1 (mirroring), RAID 1/0 (striped mirroring), or RAID 5 (striping with left-symmetric rotated parity) layout. For convenience, we will also use the term RAID 1/0 for 2-disk RAID 1 LUNs. Our FC-30 has 30 disks of 4 GB each, two controllers and 60 MB of cache RAM, which was split 20:40 between read-cache and write cache; sniffing (background scanning of the disks for errors) was disabled. Both ports of the FC-30 array were connected by a single 1 Gb/s FibreChannel arbitrated loop [3] to a four-processor HP9000 K410 server with 1 GB of main memory running HP-UX 11.0. We used a synthetic workload generator generating many concurrent requests to simulate the effect of multiple applications running on the server. Accurate I/O timings were obtained from the HP-UX in-kernel tracing facility [8, 6].

We started by calibrating a generic array model using read-only or write-only access patterns with uniformly-dispersed ("random") requests of various sizes. We then validated the models by measuring the maximum request rates supported by the FC-30 for a richer set of micro-benchmark workloads, and comparing the measurements against the models' predictions. The workloads used in the validation experiment varied three parameters: request size, read fraction and run count. They consist of a baseline workload (32KB requests, 50% reads and, run count = 1), plus a series of workloads in which one of the parameters was varied at a time. In addition, we applied the workloads to several different LUNs: 2-, 4-, and 8-disk RAID 1/0 LUNs, and 4- and 8-disk RAID 5 LUNs. The validation tests consisted of the cross-product of all workloads and all LUNs listed in Table 1—i.e. each workload was run against each LUN. The streams in this experiment access the LUN synchronously, issuing a new request as soon as the previous one has completed.

Figure 1 shows a subset of the results from the validation studies (in particular, for run count = 1). The FC-30 serviced between 70 and 360 requests/s during these experiments. On average, the models are moderately accurate but slightly pessimistic, with a mean error of $+5.4\%$, and a range of $[-11\%, +19\%]$.

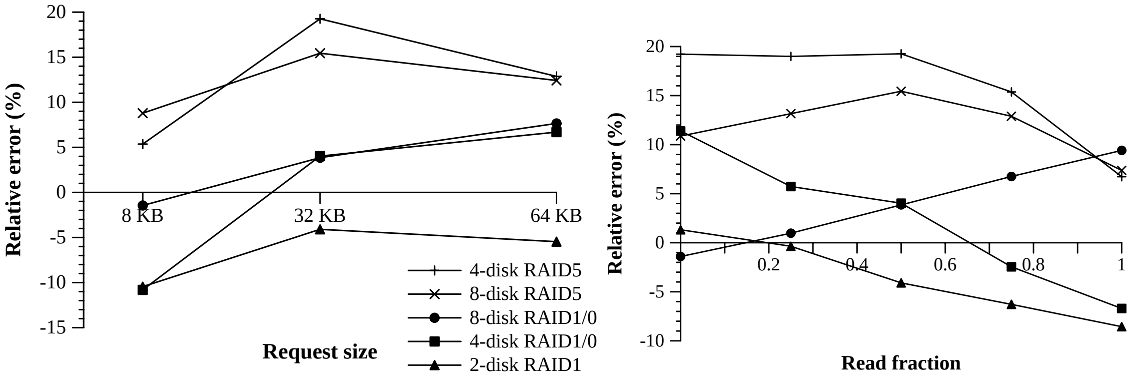

Figure 1: *Relative error in the model's throughput predictions, computed as (measured–predicted)/measured, when request sizes and read fractions deviate from the baseline values. Points above the x axis represent pessimistic model predictions.*

4 Conclusions

Because of all the simplifying assumptions made in previously published analytical models, they are quite inaccurate when compared with commercial devices. Furthermore, most works in the literature present a model for a single array layout; it can be difficult to use models from different sources because they rely on different, incompatible sets of assumptions and system models.

We have presented an analytical throughput model for disk arrays and shown that it has less than 20% error for the FC-30 disk array. The model covers both RAID 5 and RAID 1/0 layouts and accounts for request rate, request size, sequential run count, the effects of interleaving between different request streams, correlation between streams and ON -OFF phasing of the request streams.

In future work, we would like to build a more detailed model which will not require extensive calibration.

References

[1] 3Com Corporation. *Gigabit ethernet comes of age*, June 1996. Technology white paper.

[2] G.A. Alvarez, E. Borowsky, S. Go, T.H. Romer, R. Becker-Szendy, R. Golding, A. Merchant, M. Spasojevic, A. Veitch, and J. Wilkes. Minerva: An automated resource provisioning tool for large-scale storage systems. Technical report, November 2000. Submitted for Publication.

[3] ANSI. *Fibre Channel Arbitrated Loop*, April 1996. Standard X3.272-1996.

[4] E. Borowsky, R. Golding, P. Jacobson, A. Merchant, L. Schreier, M. Spasojevic, and J. Wilkes. Capacity planning with phased workloads. In *Proc. of First Intl. Workshop on Software and Performance*, pages 199–207, October 1998.

[5] Hewlett-Packard Company. *Model 30/FC High Availability Disk Array—User's Guide*, August 1998. Pub. No. A3661-90001.

[6] G. H. Kuenning. Kitrace — precise interactive measurement of operating systems kernels. *Software Practice and Experience*, 25(1):1–21, January 1995.

[7] E. K. Lee and R. H. Katz. An analytic performance model of disk arrays. In *Proc. SIGMETRICS*, pages 98–109, May 1993.

[8] C. Ruemmler and J. Wilkes. Unix disk access patterns. In *Proc. Winter USENIX*, pages 405–420, January 1993.

[9] A. Thomasian and J. Menon. RAID5 performance with distributed sparing. Technical Report RC 9878, IBM Almaden Research Center, August 1994.