RAID Organization and Performance*

Thomas J.E. Schwarz and Walter A. Burkhard

Gemini Storage Systems Laboratory
University of California at San Diego
La Jolla, California 92093-0114

Abstract

We present and analyze a novel disk array architecture that generalizes the RAID Level V data organization while providing excellent storage utilization, response times, and fault tolerance. A key feature of our approach is that reliability groups can contain several check data disks beyond the single parity disk.

Introduction

Redundant arrays of inexpensive disks (RAIDs), introduced by Patterson, Gibson and Katz [9] and further studied by Chen, Menon and Mattson, Muntz and Lui, [2] [4] [7] [8], achieve tolerance for a single disk failure by introducing redundancy. Previous work regarding RAIDs has been concerned with the cost and run-time performance for both realistic and synthesized workloads for rather small arrays.

We introduce a generalization of the five level RAID organization that accommodates multiple failures within reliability groups while retaining its excellent storage utilization, response time, and fault recovery properties. These schemes constitute a step towards the high-availability computer systems recently advocated by Gray and Siewiorek [5].

We are concerned with exploring the performance improvements that are available within very large disk arrays. We will consider workloads featuring operations that involve a small quantity of data typical of database transactions. We are interested in considering the run-time effects of various consistency schemes and we analyze a strong synchronization scheme, a write rejection scheme and a no-synchronization scheme. Our model differs in many ways from those previously considered by Chen, Gibson, Katz and Patterson [2]. We consider very large disk arrays and assume a workload without locality in which each request manipulates a small quantity of data. We assume a heterogeneous mix of independent users. Chen et al. consider small disk arrays operating, with non-independent requests, under strong operating system control. We report general analytic results applicable to both varieties of disk arrays.

The paper is structured as follows. In Section I we introduce RAID organizations that we relate to maximum distance separable (MDS) error-correcting codes [6]. We discuss data placement and three consistency schemes. In sections 2, 3, and 4 we present our RAID response time results for fault-free operation. Some simulation results are presented as well. In section 5 we present some results regarding response times during failure recovery operations. Within section 6, we recapitulate our results and comment regarding work in the future.

1 RAID Organization

RAIDs achieve fault tolerance by introducing redundancy into the system. The RAID organization stores data systematically with parity data on an additional disk. In the most advanced organizations, the parity data is distributed evenly throughout the constituent disks. In this RAID organization, each write operation must update the message and associated parity data. The new parity data is the exclusive-or of the old parity data with the exclusive-or of the new and old message. Read operations are accommodated by directly accessing the desired message data. When a disk fails, the exclusive-or of the associated remaining disks reconstructs the desired data. Message and parity data are symmetric under this recovery operation.

Classic RAID organizations view the bits at the same relative positions in each disk as forming a codeword in an algebraic error-correcting block code. To be useful within a RAID, the code must be systematic, that is codewords must be uniformly structured with

*This study supported in part by the NCR Corporation, Dayton, Ohio and Rancho Bernardo, California and the University of California MICRO Program.

identifiable message and check bits. Then when a message bit changes only the parity disk need be updated as well.

A *reliability or parity group* within a RAID is a fault tolerant ensemble of message disks and a single parity disk. If one disk is unaccessible, its contents can be reconstructed from the contents of the other disks. The reliability groups of a RAID refine its structure and the groups need not be disjoint. We have shown that the check bit of a reliability group is (up to negation) necessarily the parity of the message disks. Accordingly additional check bits provide no new check information.

By changing the granularity of the stored data and basing the codes on bytes instead of bits, the ensemble of useful error-correcting codes is significantly enlarged to include the maximum distance separable (MDS) codes. The best known variety of these codes are the (twice extended) Reed-Solomon codes, which are widely used in more traditional applications of error-correcting codes. A reliability group now consists of m message disks with c additional check disks and provides a fault tolerant ensemble of disks. These reliability groups can withstand c failed disks without losing any data. Its entire contents can be reconstructed using the data on any combination of m message and check disks. The data on message disks are stored systematically in unencoded form. The check data can be calculated as a linear form of the message data. As with the traditional parity reliability groups, during a write operation, the difference between the previous and new contents of the message disk as well as the previous contents of the check disk suffice using simple algebraic operations determine the new check disk data.

Assume we wish to construct a reliability group that contains m message disks and $n-m$ check disks. A generator matrix G, defined below, maps m message symbols p_i to n codeword symbols c_j as follows:

$$(p_1, p_2, \ldots, p_m)G = (c_1, c_2, \ldots, c_m, c_{m+1}, \ldots, c_{n-1}, c_n).$$

Each codeword symbol c_i will be stored on a different disk. We require the resulting code be *systematic;* that is, $c_i = p_i$ for $1 \leq i \leq m$ and m message disks contain the message symbols in unencoded form. Within the reliability group, we require that any m codeword symbols suffice to determine the remaining $n-m$ symbols of the codeword. This property can be phrased in terms of solutions of m linear equations with m unknowns. Accordingly the $m \times n$ generator matrix G must have the property that any set of m columns are linearly independent. We refer to this as the *independence* property. Such generator matrices are characteristic of Reed-Solomon codes. For the remainder of this paper, we only consider RAID disk array organizations derived from $m \times n$ generator matrices satisfying the independence property with the following general form: $G =$

$$\begin{pmatrix} 1 & 0 & 0 & \ldots & 0 & \alpha_{1,m+1} & \alpha_{1,m+2} & \ldots & \alpha_{1,n} \\ 0 & 1 & 0 & \ldots & 0 & \alpha_{2,m+1} & \alpha_{2,m+2} & \ldots & \alpha_{2,n} \\ 0 & 0 & 1 & \ldots & 0 & \alpha_{3,m+1} & \alpha_{3,m+2} & \ldots & \alpha_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \ldots & 1 & \alpha_{m,m+1} & \alpha_{m,m+2} & \ldots & \alpha_{m,n} \end{pmatrix}$$

The $m \times m$ identity matrix on the left of G provides the systematic property. These matrices can be obtained through elementary matrix transformations on Vandermonde matrices. The matrices and codes are defined over the Galois field $GF(2^8)$; accordingly the code symbol size and byte size are identical. The difference $n-m$ specifies the number of failed disks tolerated concurrently within the reliability group. The ratio n/m measures the storage efficiency of the group.

The systematic and independence conditions we discussed in the previous paragraph are rather standard; the independence property is also required by Rabin [11] for his Information Dispersal Algorithm. Preparata has also noted the utility of MDS codes in this context; he also provides a different approach [10].

Within classic RAID organizations the check bit of a redundancy group is (up to negation) necessarily the parity of the message disks. Additional check disks can only provide the same check data. Our organizations have no such constraint. However there is a limitation is on the size of a reliability group which cannot exceed one plus the size of the underlying field. Our reliability groups contain at most 257 disks; reliability groups of this size or less seem to be practical for current technologies.

As a very small example suppose we desire a reliability group that contains two message disks and two check disks. Our generator matrix could be

$$G = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 2 \end{pmatrix}$$

Codewords for a file stored using this scheme are generated using contiguous message symbols (bytes) from the file. Assuming the file F contains $a_0, a_1, a_2, a_3, \cdots$ then the four disks D_1, D_2, D_3, and D_4 will contain the following values

$D_1 : a_0, a_2, a_4, a_6, \cdots$
$D_2 : a_1, a_3, a_5, a_7, \cdots$
$D_3 : a_0 + a_1, a_2 + a_3, a_4 + a_5, a_6 + a_7, \cdots$

$$D_4 : a_0 + 2 \cdot a_1, a_2 + 2 \cdot a_3, a_4 + 2 \cdot a_5, a_6 + 2 \cdot a_7, \cdots$$

We can reconstruct the contents of F from the contents of any pair of disks. Any pair of symbols at identical locations within their files constitute half a codeword and this suffices to determine the other portion of the codeword. Continuing our example, suppose we have access to the contents of D_3 and D_4. The reconstruction of F is accomplished by first constructing the inverse matrix $R_{3,4}$ for columns three and four of G

$$R_{3,4} = \frac{1}{3}\begin{pmatrix} 2 & 1 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 2 \end{pmatrix}^{-1}$$

in which we have utilized the exclusive-or addition of $GF(2^8)$. The individual reconstruction computations are for $i = 0, 1, 2, \ldots$

$$(a_{2i} + a_{2i+1}, a_{2i} + 2 \cdot a_{2i+1})R_{3,4} = (a_{2i}, a_{2i+1}).$$

Each pair of symbols from the same locations within D_3 and D_4 determines a pair of message symbols. If a RAID contains dedicated check and message disks, one or the other tend to become a bottleneck. If a disk fails, the disks in the same reliability group become bottlenecks of the RAID performance. Both problems are solved with *address hashing.* Hashing distributes both the check and message data evenly throughout the array. Hashing inside the resilience group is the distinguishing feature of Level V RAIDs while hashing throughout the disk array has been proposed and investigated by Muntz and Lui[8].

Since the RAID write operations involve more than one disk, maintaining consistency among related disk blocks becomes important. We will investigate three consistency schemes: strong synchronization, write-restart and no-synchronization. Within the strong synchronization scheme, each physical write is preceeded by the acquisition of a lock which is not released until the operation is completed. The RAID design has to prevent the possibility of deadlock within the scheme. The write-restart scheme coordinates individual physical write operations by not initiating a write operation unless each physical write operation can begin immediately. Thus a write can only proceed if all the required disks are idle. This scheme presents the small possibility of starvation [12] which we ignore here. Finally the third scheme sacrifices safety for performance. Check disks are accessed only after the information block has been read. The elapsed time between the first and the last physical write operation can be considerably longer than for the other two schemes. A system crash renders more check data inconsistent and the clean-up operation after a crash becomes more complicated. The nature of our data organization makes it possible to update check disks in any order, provided each disk write operation is executed locally exactly once.

The new data organization presented here has excellent fault tolerance characteristics; initial data availability calculations indicate with even very poor quality disks (20,000 hour mttf,) we obtain 10^6 year mean time to data loss for small arrays [1].

2 Strong Synchronization

We assume a disk array with a large number of disks and that hashing uniformly distributes the load throughout the array. The message and check disks for any reliability group can reside on any disk within the array. We also assume the service times for reads and writes are exponentially distributed. We assume the request arrivals to be Poisson distributed. This leads to reasonably conservative performance bounds on the system.

A read operation is very straightforward provided no failures have occurred. A single message disk is accessed. We discuss failure operation in Section V.

We now describe an update operation. For ease of presentation in this section we assume that logical writes involve two disks: the message and check disk. Figure 1 contains a graphic representation of a typical sequence of actions. The operation requires synchronization between the message disk and the check disk. When a request arrives, it is placed in a queue at both disks. The two disks begin servicing the operation independently each as soon as possible. We describe the operation of the disks with special notice for the two synchronization signals from the check disk to the message disk. The message disk determines the difference Δ between the old and new data blocks. The Δ value is sent to the check disk when it is ready to receive the value. The check disk must send a Δ-*ok* signal to the information disk as the indication that it can receive the Δ value. The check disk also sends the *commit-ok* signal when it has read the old check value. The *magnitude* of the difference between the times of the Δ value and the *commit-ok* signals is the so-called *synchronization time;* this time reflects waiting time for one of the two disks. Figure 1 show the message disk waiting for the check disk; the check disk could be waiting just as well.

The message disk begins servicing the operation by initiating the seek of the old information data block as well as the (parallel) transfer of the new message data block to a disk buffer. The difference Δ between the old and new message values is determined. The

Δ value is sent to the check disk if the check disk is ready to accept it, otherwise the information disk begins "waiting." When the check disk begins servicing the request, the Δ-*ok* signal is sent immediately. After the old check data block has been read, the *commit-ok* signal is send immediately. When the Δ value is received by the check disk and it has sent the *commit-ok* signal, both disks write their new data blocks. The check disk must do some simple arithmetic operations on the Δ and old check data block to determine the new check data block before writing it. The release times for information and check disks are not necessarily identical, but the expected time remaining for either is two rotational latencies.

We introduce some terminalogy for our model. The time a disk is busy servicing the request we designate as I and reference as the *internal service time.* This time excludes the time spent waiting for the other disk to react that we have referred to as the synchronization time. We refer to the total time starting with the service of a request at one disk until this disk is released the *external service time* D which is the sum of the internal service time and the synchronization time. Similarly we refer to the time from the request reception by the RAID to the release of the disks the *external response time* of the request; this includes the internal service time, the synchronization time, and the waiting in queue time and is designated R.

2.1 A Queueing Network Approximation

We obtain the response times for read and write operations as response times within a two-population-queue at a single service device. The external service times are D_r and D_w respectively. The service device has utility $U = \lambda_r D_r + \lambda_w D_w$. With probability $U^\nu(1-U)$, there are exactly ν queued clients ahead of any arriving client. The device is busy with probability $u_r \stackrel{\text{def}}{=} \lambda_r D_r$ with a reader client and with probability $u_w \stackrel{\text{def}}{=} \lambda_w D_w$ with a writer client. An unknown client will require external service time $\bar{D} = \frac{u_w}{U} D_w + \frac{u_r}{U} D_r$. Accordingly the read external response time is given by

$$\begin{aligned} R_r &= D_r + \bar{D}(1-U)\sum_{\nu=1}^{\infty} \nu U^\nu \\ &= \frac{D_r + \lambda_w D_w (D_w - D_r)}{1-U}. \end{aligned}$$

Similarly, for the write external response time we have

$$\begin{aligned} R_w &= D_w + \bar{D}(1-U)\sum_{\nu=1}^{\infty} \nu U^\nu \\ &= \frac{D_w + \lambda_r D_r (D_r - D_w)}{1-U}. \end{aligned}$$

Now we determine the external service times. Since reader operations involve only a single disk, D_r is a fraction α of the internal write service time I. We are left with the interesting question of the external write service time D_w. Our principal result within this Section is the determination of the synchronization time for the write operation with write quorum x; it is the following

$$(H_x - 1)(R_w - D_w + I)$$

where H_x is the $x^{\underline{th}}$ harmonic number [13] . Accordingly the external service times are given by

$$\begin{aligned} D_r &= \alpha I \\ D_w &= H_x I + (H_x - 1)(R_w - D_w) \end{aligned}$$

We present in Figure 2 response time results for three disk array configurations each with 100 message disks derived from these equations; the first is a single reliability group consisting of 102 disks, the second consists of five disjoint groups each with 22 disks, and the third contains ten reliability groups each with 12 disks. The solid curves designate fault-free operation response times. The The write quorum x is three for each array. The fraction of read operation ρ is 0.5.

These curves demonstrate a singularity; the associated utility is much smaller than one. This behavior is an essential feature of our model. The D_w equation is quadratic and can be solved only for small loads. In particular, if λ_r is zero, we obtain the following

$$D_w = \frac{(H_x I \lambda_w + 1) - \sqrt{(H_x I \lambda_w + 1)^2 - 4 H_x^2 I \lambda_w}}{2 \lambda_w H_x}$$

And this equation has a solution provided

$$\lambda_w \leq \lambda_{max} \stackrel{\text{def}}{=} \frac{2H_x - 1 - 2\sqrt{H_x(H_x-1)}}{H_x I}.$$

This contrasts with the load of $\lambda_w = 1/I$ if the theoretically maximum achievable throughput is obtained. As an example, suppose that write operations have a write quorum of two and I is 20 *ms.* Then $\lambda_w \leq 0.1786/20$ and as the load approaches this critical value from below, the utilization does not go to one; rather it goes to 0.42. We have no analytic results for loads exceeding this critical value. Simulation results, presented in Figure 3, model our "write-only" scenario. The service time is Poisson distributed which is a non-continuous version of an exponentially distributed service time with expectation of 20 *ms.* We plot the external response time R versus system arrival rate λ for

a RAID with 100 disks and a write quorum of two. The maximum disk load $\lambda_{max} = .1786/20 = 0.00893$; accordingly the maximum system arrival rate $\lambda_{max} = 0.00893 \times 50 = 0.4465$. The simulation results presented within Figure 3, have a very sharp knee at approximately λ_{max}. We present additional simulation results that demonstrate the same qualitative behavior. The service time used has constant plus uniform components. Increasing the write quorum x decreases the critical value here as well.

3 Write-Restart Synchronization

The disappointing performance figures of strong synchronization result due to the synchronization required among the participating disks. The response times reflect the time of the slowest participant. The write-restart synchronization scheme provides much better response time performance. Here the write operation can proceed only if all participating disks are idle. The write requests need not be executed in arrival order but the synchronization scheme itself does not provide any scheduling optimization. As we have noted there is a small probability of starvation which can be thwarted within the implementation. Once a logical operation begins, it continues without interruption to completion.

Our analysis of the response times proceeds much as within the previous section beginning with the same assumptions. The read external response time R_r is, as in section 2,

$$R_r = \frac{D_r + \lambda_w D_w (D_w - D_r)}{1 - U}.$$

The read external service time D_r is a fraction α of the write internal service time I as before. Since all disks among an operation begin together and individual write internal service times are exponentially distributed, the write external service time D_w would be $H_x I$. However we can provide a sharper estimate for D_w. For a single disk, approximately one-half of the internal service time I is spent seeking and rotating before the data is read, and the remainder of the time is spent during the full rotation of the update phase. We assume that only the initial seek and rotation times are exponentially distributed yielding $D_w = \frac{1}{2}(H_x + 1)I$. The write external response time R_w expression grows in complexity with the write quorum; it is composed of the write internal service time plus the time spent waiting for clients ahead to clear. We develop the external write response time for disk arrays having write quorum two. The probability of finding at least one disk of a pair busy is $P \stackrel{\text{def}}{=} 1 - (1-U)^2$. The expected service time $\bar{D}_2$ measuring the busy time of at least one of a pair of disks is

$$\begin{aligned} \bar{D}_2 &\stackrel{\text{def}}{=} \frac{u_r^2}{P}[D_r, D_r] + \frac{2u_r u_w}{P}[D_r, D_w] + \frac{u_w^2}{P}[D_w, D_w] \\ &+ \frac{2u_r(1 - u_r - u_w)}{P} D_r + \frac{2u_w(1 - u_r - u_w)}{P} D_w. \end{aligned}$$

where $[D_r, D_r]$, $[D_r, D_w]$, and $[D_w, D_w]$ are order statistics for the maximum of two specified service times. We can estimate $[D_r, D_r]$ as $1.3D_r$ rather than $\frac{3}{2}D_r$, $[D_w, D_r]$ as approximately D_w and $[D_w, D_w]$ as $1.05D_w$. The formula for larger write quora will be similar. Finally

$$R_w = D_w + \bar{D}_2(1 - P)\sum_{\nu=1}^{\infty} \nu P^{\nu} = D_w + \frac{\bar{D}_2 P}{1 - P}.$$

We present write-restart response time results within Figure 4 for write quorum three. This write quorum will facilitate comparison with the results for strong synchronization given within Figure 2. The fraction of read operations ρ is 0.5. The behavior is familiar. The write response time for zero load is lower than for either strong- or no-synchronization because of our improved D_w value. The write-restart scheme is operational for any utility less than one unlike the strong scheme.

4 No Synchronization

We have studied the performance of disk arrays in which the risk of inconsistencies resulting from a write operation is limited by synchronization among the participating disks. In this section we assume a different strategy. Either the risk is ignored or during each update a log of the various phases of the operation is maintained throughout the lifetime of the operation. We make the same assumptions regarding input arrival rate and service times as within section 2 and we assume that any risk aversion measures do not introduce a performance bottleneck.

A read operation accesses a single disk and a write operation accesses x disks according to the write quorum. A write operation at an message disk consists of seek and rotate, a full rotation commencing with reading the block, and a transfer when new data is stored on the disk. A write operation at a check disk consists of one disk access to read the check disk followed by a write that follows the read of the message disk.

The check disk accesses of a write operation are initiated only after the message disk has been read. In this strategy the write service time at both message and check disks D_w coincides with the internal service time I of Section II. The read service time D_r is smaller by a full rotation which we estimate as $\frac{1}{2}I$.

The read response time is, as in section 2,

$$R_r = \frac{D_r + \lambda_w D_w (D_w - D_r)}{1 - U}.$$

The write response time, with write quorum x, consists of the time to read from the message disk followed by the write response time for $x-1$ check disks. Thus

$$R_w = \left(\frac{D_r + \lambda_w D_w (D_w - D_r)}{1 - U}\right) + H_{x-1}\left(\frac{D_w + \lambda_r D_r (D_r - D_w)}{1 - U}\right).$$

We present response time results for both read and write operations within Figure 5. The fraction of read operations ρ is 0.5 again. The write quorum is three for each disk array. Once again the operations are accessable for any utility less than one. This scheme with no synchronization provides response times that are only incrementally better than the response time for the write-restart synchronization scheme.

5 Fault Recovery

We discuss the performance of various recovery schemes. Read operations involving a failed disk are replaced by m other disk reads within the reliability group. Write operations involving a failed disk depend on its role. For a failed message disk, we first obtain the contents of the message disk by reading m disks and then we write the check disks. For a failed check disk, we omit the portion of the update involving the failed disk.

Consider a reliability group with write quorum x and n disks that contains one failed disk. The total "read" load λ_r is

$$\frac{n-1}{n}\rho\lambda + \frac{m}{n}\rho\lambda + \frac{m}{n}(1-\rho)\lambda + \frac{(x-1)}{n}(1-\rho)\lambda \quad (1)$$

where the addends, from left to right, designate the load for read operations not involving the failed disk, the load incurred to accomodate the read at the failed disk, the read required for write operations involving the failed disk as the message disk and the write operation when the failed disk is a message disk. The last addend contributes a "read load" since each check disk is written without first reading it. The total "write" load λ_w is

$$\frac{n-x}{n}x(1-\rho)\lambda + \frac{(x-1)^2}{n}(1-\rho)\lambda. \quad (2)$$

The proportion of the write load not involving the failed disk is $\binom{n-1}{x}/\binom{n}{x} = \frac{n-x}{n}$; accordingly the first addend in this expression designates the total write load not involving the the failed disk and the other addend designates the write load arising when the failed disk is a check disk. We write to $x-1$ disks for the second addend.

Multiple disk failures are treated in exactly the same manner. Assuming the number of failures is less than $n-m$ we can always recover the data. We have presented response times for single failures, derived from equations 1 and 2, within Figures 2, 4 and 5.

Now we consider the use of stand-by disks; two effects change our perspective. First the reconstruction process adds $m\lambda_c$ to the total read load for reconstruction rate λ_c. Second the loads calculated above remain valid only for the portion of the stand-by disk not already reconstructed. Otherwise the normal read and write system loads apply. Let $f(t)$ designate the fraction of the blocks reconstructed on the stand-by disk at time t; then our reconstruction rate is $\frac{df}{dt}$. Now our system load formulas are as follows:

$$\overline{\lambda}_r = f(t)\rho\lambda + (1 - f(t))\lambda_r + m\lambda_c$$
$$\overline{\lambda}_w = f(t)(1-\rho)x\lambda + (1 - f(t))\lambda_w$$

In a simple scheme $\frac{df}{dt} = \lambda_c$ and $f(t) = \lambda_c t$; this recovery scheme methodically rebuilds the stand-by disk. The benefits of write redirection and read piggybacking depend on access locality. Let $Z(t)$ designate the total number of blocks accessed in the interval between time 0 and t; Z might reflect the 80-20 rule or Zipf's distribution. We obtain

$$\frac{df}{dt} = \lambda_c + (1 - \lambda_c t)\frac{dZ}{dt} \qquad f(0) = 0$$

for the reconstruction rate.

6 Conclusions

We have introduced a useful extension to the RAID Level V data organization that provides excellent fault tolerance and response times. We present the analysis results of three synchronization schemes for hashed disk arrays. Several observations are noted for storage systems designers.

Strong synchronization and increased numbers of writes per RAID update adversely effects the run-time performance of the system. However, both of these factors improve the fault-tolerance of the ensemble. The number of writes per update is the principal protection against almost simultaneous failure of a small number of disks caused by environmental (temperature extremes, earthquakes, electro-magnetic radiation) or bad batch [3] (fungicide problem etc.) phenomena.

Stand-by disks provide excellent failure tolerance against unrelated, individual disk failures. The reason lies in the small amount of time to replace a failed disk by a stand-by; it is very unlikely that another disk failure occurs during this replacement and in the flexibility of using pooled stand-bys to replace any failed disk rather than simply expanding the reliability groups with the increased number of writes per RAID update (which would generally lower the run-time performance.) An interesting research topic that remains is how to accommodate non-independent disk failures typical of essential component failure.

If the strong synchronization scheme is used, the RAID performance could be improved by preventing the build-up of requests at a single disk. This variation avoids blocking of writes at other disks. A similar approach is used by Chen[2]. Write-restart synchronization provides an excellent response times, almost as good as with no synchronization.

References

[1] Walter A. Burkhard, Kimberly C. Claffy and Thomas J.E. Schwarz. "Performance of Balanced Disk Array Schemes," *Eleventh IEEE Symposium on Mass Storage Digest*, pp. 45-50, 1991.

[2] Peter M. Chen, Garth A. Gibson, Randy H. Katz and David A. Patterson. "An Evaluation of Redundant Arrays of Disks using an Amdahl 5890," *Performance Evaluation*, pp. 74-85, 1990.

[3] Garth A. Gibson. "Reliability and Performance in Redundant Arrays of Magnetic Disks," *International Converence on Management and Performance Evaluation of Computer Systems (CMG) XX Proceedings,* Computer Measurement Group, December 1989.

[4] Garth A. Gibson, Lisa Hellerstein, Richard Karp, Randy H. Katz and David A. Patterson. "Failure Correction Techniques for Large Disk Arrays," *Proceedings of the Third International Conference on Architectural Support of Programming Languages and Operating Systems,* pp.123-132, 1989.

[5] Jim Gray and Daniel P. Siewiorek. "High-Availability Computer Systems," *COMPUTER,* pp. 39-48, 1991.

[6] Florence J. MacWilliams, Neil J.A. Sloane. *The Theory of Error-Correcting Codes,* North Holland, 1978.

[7] Jai M. Menon and Richard L. Mattson. "Performance of Disk Arrays in Transaction Processing Environments," *Proceedings of the 12th International Conference on Distributed Computing Systems,* 1992.

[8] Richard R. Muntz and John C.S. Lui. "Performance Analysis of Disk Arrays under Failure," *Proceedings of the VLDB,* pp. 162-173, 1990.

[9] David A. Patterson, Garth A. Gibson and Randy H. Katz. "A Case for Redundant Arrays of Inexpensive Disks (RAID)," *ACM SIGMOD Conference Proceedings,* pp.109-116, 1988.

[10] Franco P. Preparata. "Holographic Dispersal and Recovery of Information," *IEEE Transactions on Information Theory,* Vol. 35, pp. 1123-1124, 1989.

[11] Michael O. Rabin. "Efficient Dispersal of Information for Security, Load Balancing and Fault Tolerance," *Journal of the Association for Computing Machinery,* pp. 335-348, 1989.

[12] Daniel J. Rosenkrantz, Richard E. Stearns, Phil M. Lewis, II. "System Level Consistency Control for Distributed Database Systems," *ACM Transactions on Database Systems,* pp. 178-198, 1978.

[13] Thomas J.E. Schwarz and Walter A. Burkhard. "RAID Performance via Queueing Network Analysis," Technical report, University of California, San Diego, 1991.

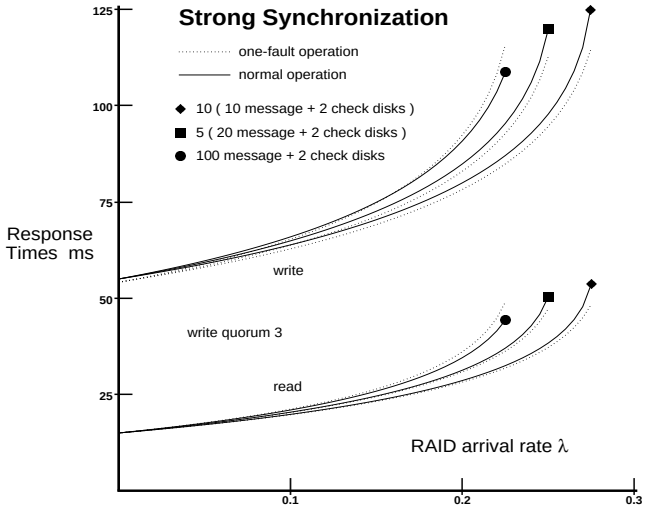

Figure 2. Strong Synchronization

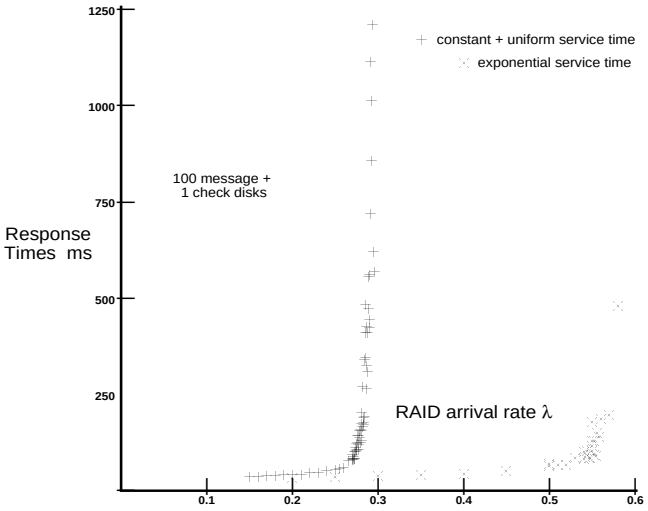

Figure 3. Simulation Results

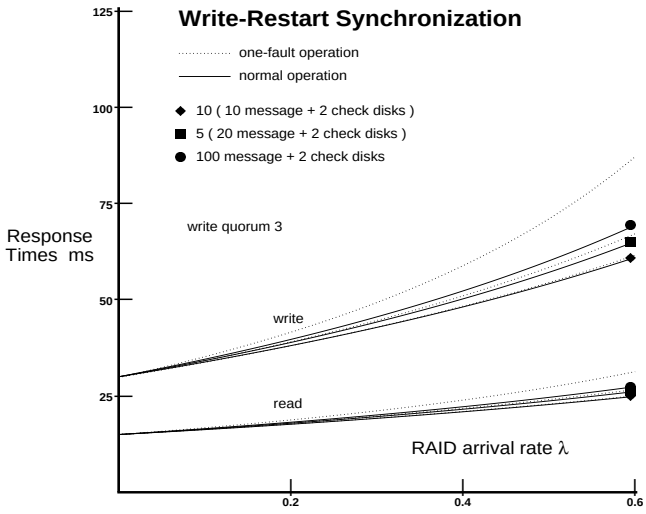

Figure 4. Write-Restart Synchronization

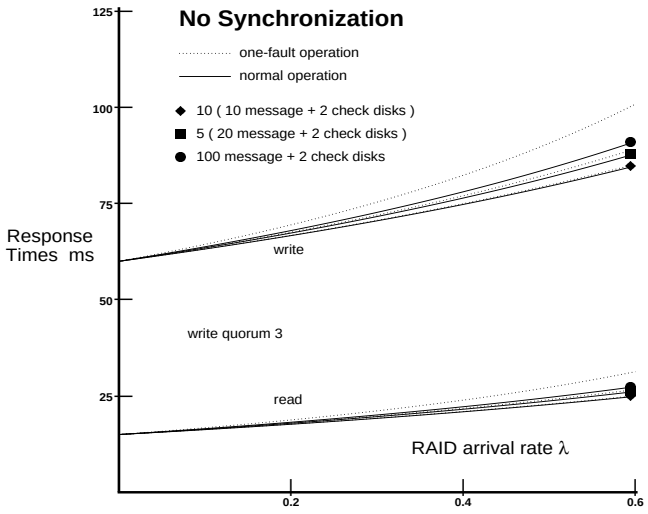

Figure 5. No Synchronization

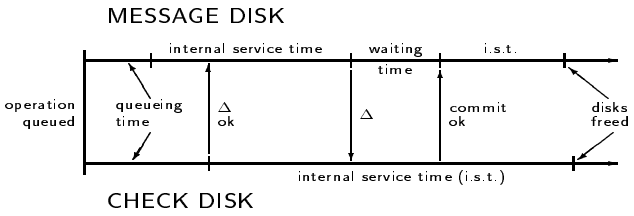

Figure 1: RAID Update Timing Example.