Rebuild Strategies for Redundant Disk Arrays

Gang Fu, Alexander Thomasian*, Chunqi Han, and Spencer Ng†
Computer Science Department
New Jersey Institute of Technology -NJIT
Newark, NJ 07102, USA

Abstract

RAID5 performance is critical while rebuild is in progress, since in addition to the increased load to recreate lost data on demand, there is interference caused by rebuild requests. We report on simulation results, which show that processing user requests at a higher, rather than the same priority as rebuild requests, results in a lower response time for user requests, as well as reduced rebuild time. Several other parameters related to rebuild processing are also explored.

1 Introduction

RAID5 with rotated parity is a popular design, which tolerates single disk failure and balances disk loads via *striping*. Striping allocates successive segments of files called *stripe units* on the $N-1$ disks in an array of N disks, with one stripe unit dedicated to parity, so that a capacity equal to the capacity of one disk is dedicated to parity. In this case the *parity group size* G is equal to N. The parity blocks are kept up-to-date as data is updated and this is especially costly when small randomly placed data blocks are updated, hence the *small write penalty*.

If a single disk fails, a block of data on that disk is recreated by exclusive-ORing (XORing) the corresponding blocks on the $N-1$ surviving disks. Each surviving disks needs to process the load due to fork-join requests to recreate lost data blocks besides its own load, so that the load on surviving disks is increased, e.g., at worst doubled when all requests are reads. *Clustered RAID* solves this problem by selecting a parity group size $G < N$, so that the load increase is proportional to the *declustering ratio:* $(G-1)/(N-1)$ [3]. *Balanced Incomplete Block Designs – BIBD* [1, 4] and *random permutation layout* [2] are two approaches to balance disk loads from the viewpoint of parity updates.

The rebuild process is a systematic reconstruction of the contents of the failed disk, which is started immediately after a disk fails, provided a hot spare is available. Of interest is the time to complete the rebuild $T_{rebuild}(u)$ and the response time of user requests versus time: $R(t), 0 < t < T_{rebuild}(u)$. The utilization at all disks, which is equal to u before disk failure occurs, is specified explicitly, since it has a first order effect on rebuild time.

A distinction is made between *stripe-oriented* or *rebuild-unit(RU)-oriented* and *disk-oriented* rebuild in [1]. In the former case the reconstruction proceeds one RU at a time, so that the reconstruction of the next RU is started after the previous one is reconstructed and even written to disk. Disk-oriented rebuild reads RUs from all surviving disks asynchronously, so that the number of RUs read from surviving disks and held in a buffer in the disk array controller can vary. It is shown in [1] that disk-oriented rebuild outperforms stripe-oriented rebuild, therefore the stripe-oriented rebuild policy will not be considered further in this study.

Rebuild requests can be processed at the same priority as user requests, which is the case with the *permanent customer model – PCM* [2], while [1, 6] process rebuild requests when the disk is idle according to the well-known *vacationing server model – VSM* in queueing theory: an idle server (resp. disk) takes successive vacations (resp. reads successive RUs), but returns from vacation (resp. stops reading RUs) when a user request arrives at the disk. In effect rebuild requests are processed at a lower priority than user requests. In PCM a new RU is introduced at the tail of the request queue, as soon as the processing of the previous RU is completed, and hence the name of the model.

The few studies dealing with rebuild processing [1, 2, 6] leave several questions unanswered, such

*The first three authors are partially supported by NSF through Grant 0105485 in Computer Systems Architecture.

†Hitachi Global Storage Technologies, San Jose Research Center, San Jose, CA.

as the relative performance of VSM versus PCM, the effect of disk zoning, etc., but not all issues are addressed here due to space limitations. We extended our RAID5 simulator to simulate rebuild processing. This simulator utilizes a detailed simulator of single disks, which can handle different disk drives whose characteristics are available at: http://www.pdl.cmu.edu/Dixtrac/index.html. The reason for adopting simulation rather than an analytic solution method is because of the approximations required for analysis, which would have required validation by simulation anyway. Simulation results are given in the next section, which is followed by conclusions.

2 Experimental Results

The parameters of the simulation are as follows. We utilize IBM 18ES 9 GByte, 7200 RPM disk drives. We assume an OLTP workload generating requests to small (4KB) randomly placed blocks over the data blocks of the $N = 19$ disks in the array. Track alignment ensures that all 4 KB accesses are carried out efficiently, since they will not span track boundaries. Accesses to randomly placed disk blocks introduce a high overhead, i.e., 11.54 ms per request with FCFS scheduling, less than 1% of which is data transfer time. We assume that the ratio of reads to writes is R:W=1:0, since reads introduce a heavier performance degradation upon disk failure. While we experimented with different disk utilizations, only results for $u = 0.45$, which results in a 90% disk utilization are reported here. We assume zero-latency read and write capability, which has a significant impact on rebuild time.

The parameter space to be investigated includes: (i) VSM versus PCM. (ii) The impact of buffer size per disk B specified as number of tracks. (iii) the size of the RU (rebuild unit) is a multiple of tracks and $RU = 1$ is the default value. (iv) the effect of preempting rebuild requests. (v) The effect of piggybacking, also considered in [1]. (vi) The effect of read-redirection and controlling the fraction of reads redirected. In fact due to its beneficial effect read-redirection is postulated in all other cases reported here [1, 5]. (vii) The effect of the number of disks on rebuild time. (viii) A first order approximation for rebuild time. Due to space limitations item (iv) is not investigated.

2.1 VSM versus PCM

The response time of user requests $R(t)$ and the completion percentage $c(t)$, which is the fraction of tracks already rebuilt, versus time (t) for both VSM and PCM are shown in Figure 1. A disk failed and rebuild was started at time $t = 135$ sec. The graphs shown are averages over ten runs, but even more runs are required to obtain a tighter confidence interval, say at 95% confidence level. The following observations can be made:

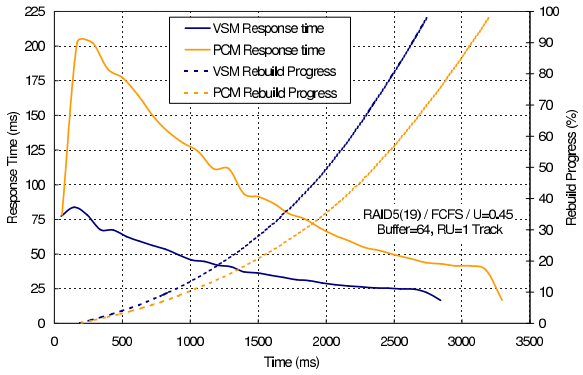

Figure 1: Performance Comparison of VSM and PCM

(i) Disk utilizations are effectivy 100% utilized in both cases, but since RU reads are processed at a lower (nonpreemptive) priority in VSM user requests are only affected by the mean residual service time for RU reads, which is close to half a disk rotation at lower disk utilzations. PCM yields a higher $R(t)$ than VSM since it reads RUs at the same priority as user requests. For each $\bar{n}$ user requests processed (on the average) by a disk, the disk processes $\bar{m}$ consecutive rebuild requests, so that the arrival rate is increased by a factor of $1 + \bar{m}/\bar{n}$. Furthermore, rebuild requests, in spite of zero latency reads, have a longer service time than user requests. (ii) The rebuild time in PCM is higher because during the rebuild period, the disk utilizations due to user requests is approximately the same, but the disk "idleness" is utilized more efficiently by VSM than PCM. The reading of consecutive RUs is started in VSM only when the disk is idle, so that if the reading of an RU is completed before a user request arrives, the reading of the next RU can be carried out without incurring seek time. Uninterrupted processing of rebuild requests is less likely with PCM, since by the time the request to read an RU is being served, it is not likely that the disk queue is empty. This intuition can be ascertained by comparing the mean number of consecutive RU reads in the two cases.

2.2 The impact of buffer size

Figures 2 and 3 show $R(t)$ and $T_{rebuild}$ versus buffer size for VSM and PCM, respectively. We can reduce buffer space requirements by XORing the available blocks right away, but this would introduce con-

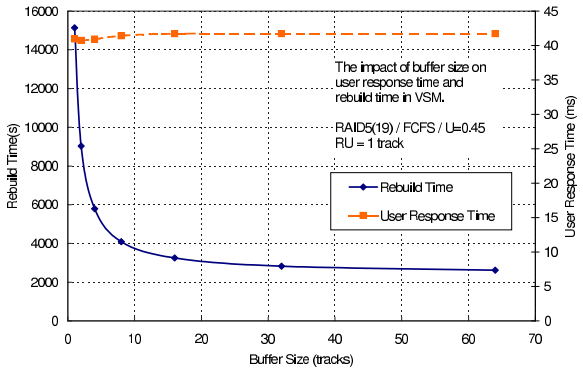

Figure 2: The impact of buffer size in VSM

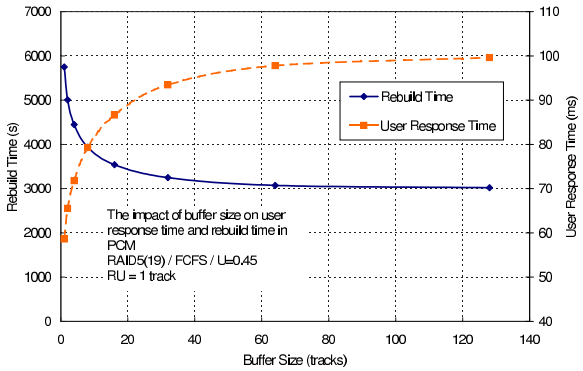

Figure 3: The impact of buffer size in PCM

straints on hardware requirements. Two observations can be made: (i) With disk-oriented rebuild a larger buffer size leads to shorter rebuild times in both VSM and PCM. This is because due to temporary load imbalance, rebuild processing with disk-oriented rebuild is suspended when the buffer is filled. A shared or semi-shared buffer requires further investigation. (ii) The impact of buffer size on $R(t)$ for VSM is very small, but it is significant in PCM. In VSM, the rebuild requests are processed at a lower priority, so the $R(t)$ is only affected by the time to read an RU. In PCM a small buffer size limits the rate of RU reads, i.e., no new RU reads are inserted into the disk queue when the buffer is full, but as B is increased, RU reads are introduced at a rate determined only by u.

2.3 The impact of rebuild unit size

Figures 4 and 5 shows $R(t)$ and $T_{rebuild}$ versus the RU size $= 1, \ldots 16$ tracks. The following observations can be made: (i) The larger RU size leads to higher response times in VSM and PCM. In VSM the mean residual time to read an RU is added to the mean waiting time caused by the contention among user requests [6].(ii) Larger RU sizes lead to shorter rebuild times in VSM and PCM, because the rebuild time per RU is reduced, since the cost of one seek is prorated over the reading of multiple RUs.

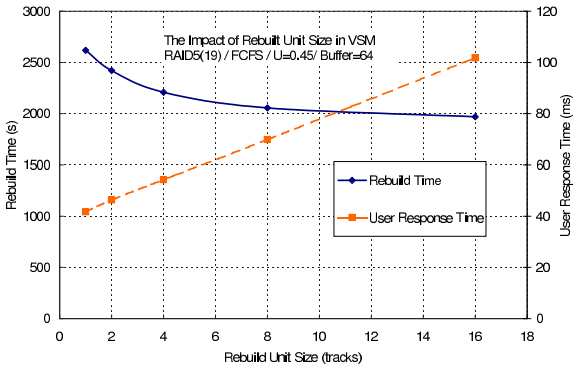

Figure 4: The impact of rebuild unit size in VSM

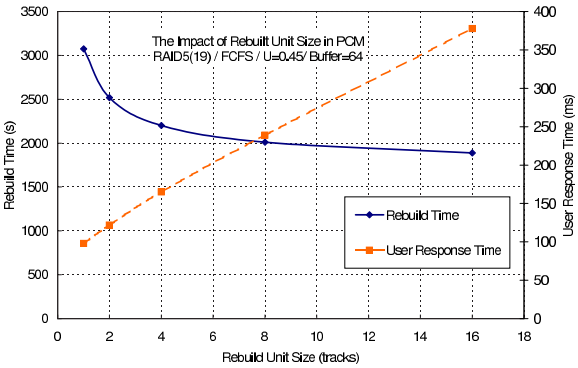

Figure 5: The impact of rebuild unit size in PCM

2.4 The effect of piggybacking

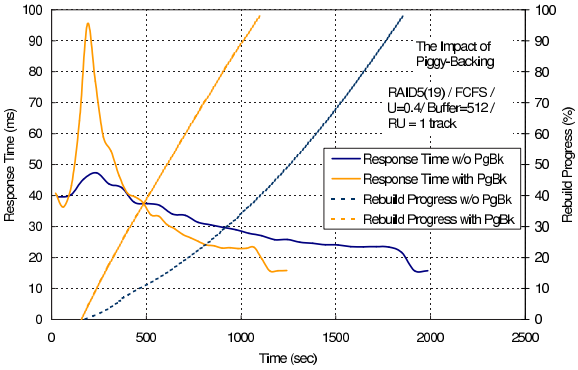

Figure 6: The effect of piggybacking

Figure 6 shows $R(t)$ and $c(t)$ versus t in VSM with and without piggybacking. Piggybacking is done at the RU level, rather than at the level of user accesses, e.g., 4 KB blocks, which has been shown to be ineffectual [1]. In other words, we extend the reconstruction of a single block into a full track. Piggybacking shortens rebuild time, but initially the disk utilization will exceed $2u$ in our case, since track reads have a higher mean service time than the reading of 4 KB blocks, which are carried out on behalf of fork-join requests (this is the reason why $u = 0.4$ in this case). We can control the initial increase in $R(t)$ by controlling the fraction of piggybacked user requests.

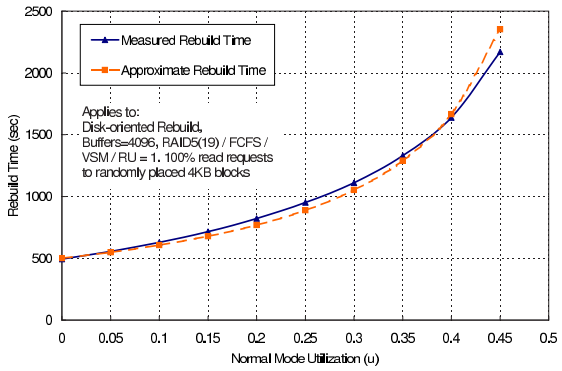

Figure 7: The effect of disk utilization on rebuild time

2.5 The impact of read redirection

Read-redirection shortens the rebuild time (by a factor of three with 19 disks, u=0.45, and B=64) and also reduces $R(t)$ as gradually more tracks from the failed disk are rebuilt. Since we assumed there are no write requests, we get the maximum improvement in performance with read redirection, but there will be less improvement in response time when all requests are updates. In "unclustered" RAID5 the utilization of the spare disk remains below the other disks and there is no need to control the fraction of read requests being redirected, as in [3].

2.6 The effect of the number of the disks

Table 1 gives the rebuild time using VSM versus the number of disks (N) with various number of buffers. The disk utilization is $u = 0.45$. It is observed that even at high disk utilization, rebuild time with VSM is not affected very much by N. An additional observation is that due to the prevailing symmetry, that the load at all disks is balanced, rebuild time is determined by the reading time of any of the disks and that the writing of the spare disk follows shortly thereafter. When the buffer size dedicated to the rebuild process is small, rebuild time is more sensitive to N.

Number of disks		9	19	29	39
Rebuild Time (sec)	B=16384	1604.2	1622.8	1631.9	1632.0
	B=64	2586.3	2618.9	2701.4	2704.5
	B=16	3033.8	3246.0	3434.7	3526.5

Table 1: The effect of number of disks on rebuild time

2.7 The effect of disk utilization on rebuild time

It follows from Figure 7 that rebuild time increases sharply with disk utilization. This is especially so with an infinite source model, where the request arrival rate is not affected by increased disk response time. Rebuild time can be reduced by not processing low priority applications.

We obtain a first order approximation for rebuild time with given disk characteristics by postulating that the rebuild time $T_{rebuild}(u)$ is simply a function of the disk utilization u, so that $T_{rebuild}(u) = T_{rebuild}(0)/(1-\alpha u), \alpha u < 1$, where $T_{rebuild}(0)$ is the time to read an idle disk, and α is a measure of the average increase in disk utilization during rebuild.

With read redirection and R:W=1:0 the utilization of surviving disks initially doubles, but then it drops gradually as disk utilization reaches u, so that $\alpha \approx 1.5$ is an initial guess. On the other hand the rebuild time is slow when rebuild starts, so that more time is spent in the initial stages of rebuild. Curve fitting shows that $\alpha \approx 1.75$ yields an acceptable estimate of rebuild time, but there is a sensitivity to R:W ratio, and other factors, which are being explored.

3 Conclusions

It is interesting to note that VSM outperforms PCM on both counts, user response times and rebuild time. The latter is counterintuitive since PCM rebuilds more aggressively. We are also investigating the effect of preempting rebuild requests, utilizing multiple rebuild regions, and allowing multiple user requests before rebuild processing in VSM is stopped. Finally, we are interested in rebuild processing in RAID6 and EVENODD, especially when operating with two disk failures.

References

[1] M. C. Holland, G. A. Gibson, and D. P. Siewiorek. "Architectures and algorithms for on-line failure recovery in redundant disk arrays", *Distributed and Parallel Databases 11*(3): 295-335 (July 1994).

[2] A. Merchant and P. S. Yu. "Analytic modeling of clustered RAID with mapping based on nearly random permutation", *IEEE Trans. Computers 45*(3): 367-373 (1996).

[3] R. R. Muntz and J. C. S. Lui. "Performance analysis of disk arrays under failure", *Proc. 16th Int'l Conf. VLDB*, 1990, pp. 162–173.

[4] S. W. Ng and R. L. Mattson. "Uniform parity distribution in disk arrays with multiple failures", *IEEE Trans. Computers 43*(4): 501-506 (1994).

[5] A. Thomasian and J. Menon. "Performance analysis of RAID5 disk arrays with a vacationing server model for rebuild mode operation", *Proc. 10th Int'l Conf. Data Eng. - ICDE*, 1994, pp. 111-119.

[6] A. Thomasian and J. Menon. "RAID5 performance with distributed sparing", *IEEE Trans. Parallel and Distributed Systems 8*(6): 640-657 (1997).