UPGRAID: Usage-base striPe replicatinG RAID

Jorge Guerra, Joseph Naps†, Raju Rangaswami, Luis Useche, Ellen Wagner‡
jguerra@cs.fiu.edu naps@wisc.edu raju@cs.fiu.edu luis@cs.fiu.edu ewagner09@wooster.edu
†*Univeristy of Wisconsin - Madison*
‡*College of Wooster*
Florida International University

***Abstract*— Redundant Array of Inexpensive Disks (RAID) is the standard for large scale, fault-tolerant storage solutions. The various RAID levels combine space and/or performance optimization to meet the needs of the user. In the paper we introduce UPGRAID, which can be seen as a new RAID level built upon RAID5. The idea behind UPGRAID is that, by allocating a small amount of disk space for RAID stripe replication, substantial gains can be seen in both reducing response time during normal operation and reducing overall reconstruction time when a drive fails. This decrease is important as reconstruction is the most vulnerable time for a RAID5 system because another disk failure would render the array useless. Here we cover UPGRAID's design and implementation.**

I. Introduction

Redundant, failure-tolerant disk systems come in two basic forms: those that optimize for performance (e.g. RAID1), and those that optimize for space utilization (e.g. RAID5). Past research has argued for a hybrid system that would allow for the benefits of both performance and space optimization [1], [2]. In this study we begin the development of UPGRAID (Usage-based striPe replicatinG RAID) as a new RAID configuration based on RAID5. When compared to a RAID5 system, UPGRAID aims to reduce the average response time for a majority of cases while also not significantly degrading response time in a small fraction of cases. UPGRAID also looks to speed up the reconstruction process, thereby improving availability, which is favored as the "key metric" in server systems [3].

UPGRAID operates by adding stripe-level replication for frequently accessed data on top of the basic RAID5 system. Each drive contains two partitions: a larger partition that is part of the RAID5 array and a smaller partition that is used for storing the replicas of popular stripes. It then dynamically identifies and replicates popular stripes onto a drive other than the stripes' corresponding data or parity drive.

During normal operation UPGRAID read requests can be sent to the drive that is under the least load at that time. For write operations to popular stripes UPGRAID needs to update the replica stripe in addition to the standard RAID5 data and parity updates. However, the application can be potentially notified of I/O completion as soon as the replica stripe is written to, thus eliminating the need to wait for the potentially lengthy RAID5 write update. The downside is the additional bandwidth needed to update the replica stripes. To counter this write overhead, UPGRAID must dynamically control the number of stripes that have been replicated at any given time.

During reconstruction reads to a popular stripe that has not been reconstructed are substantially faster. Unlike RAID5, a popular stripe need not be computed from parity information since the replica can be used in the original's place. For writes to popular stripes updating the replica is all that is needed. The reconstruction phase itself is also made faster due to the fact that replicas can be used to rebuild popular stripes of the failed drive.

II. Approach

At its core UPGRAID is a RAID5 system. The modifications during normal operation can be broken down into four main cases: read replication, write replication, read indirection, and write indirection. The details of these four cases will now be discussed in more detail.

A. Read Replication

When a read request is received to an unmapped stripe...

1) UPGRAID determines if the stripe is eligible for replication.
2) If the stripe is eligible, a read request to the entire stripe is generated.
3) Once that read request completes, a write is generated and put into a queue to await being sent to an UPGRAID partition.

B. Write Replication

Write replication begins in a similar manner to that of read replication but has some added complications...

1) UPGRAID determines if the stripe is eligible for replication.
2) If the stripe is eligible a read request to the entire stripe is generated.
3) At this point there are sixteen pages (in the page of a sixty-four KB stripe) with the data from the original stripe as seen in Figure 1.
4) The data from the original write must now be overlaid on top of the data read from the stripe to preserve the modifications from the write as in Figure 2.
5) The modified write is sent to a queue to await submission to the proper UPGRAID partition.

Fig. 1. The data pages following a read to the RAID5 partition

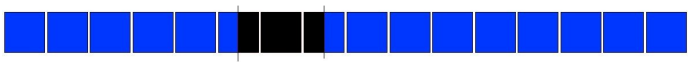

Fig. 2. The data pages with the original write overlaid on top

C. Read Indirection

When UPGRAID receives a read request to a mapped stripe the following occurs...

1) UPGRAID determines if the request should be sent to the RAID5 partition or UPGRAID partition by looking at the head position of each drive. This drive that has the smallest distance to move is chosen to fulfill the request.
2) The request is then sent to the appropriate disk and the application proceeds upon completion of that read request.

D. Write Indirection

Write replication is the simplest case...

1) The write request to the RAID5 partition is cloned.
2) This cloned request gets sent to the appropriate location on the UPGRAID partition at the same offset into the stripe as the original write, thereby preserving the mirroring property between the two stripes.

III. Popularity

A. Ranking

The popularity of the individual stripes combines both frequency of access and recency of access to calculate an overall popularity rating for the stripe. The system works by taking the number of accesses to a stripe per unit time and calculating an access graph like that in Figure 3, where the X-axis represents the passage of time and the Y-axis represents the number of accesses to a stripe during that unit of time.

Once the access curve has been defined the ranking is determined by feeding the values at the various units of time into an exponential function depicted in Figure 4. This exponential function allows for both frequency and recency to be taken into account. The growth properties of the exponential function insure that a greater weight is put on recency of access as opposed to frequency.

B. Bottom-K

In addition to keeping the ranks of all of the stripes in the UPGRAID partitions, the popularity system must also be able to keep track of the lowest ranked (bottom-k) stripes on each disk for the purpose of efficiently removing stripes from the UPGRAID partitions when new, more popular, stripes are replicated. UPGRAID maintains a matrix of these bottom-k

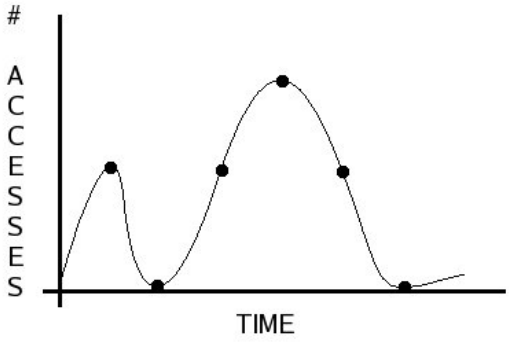

Fig. 3. Example access graph, the points represent the number of accesses per unit time

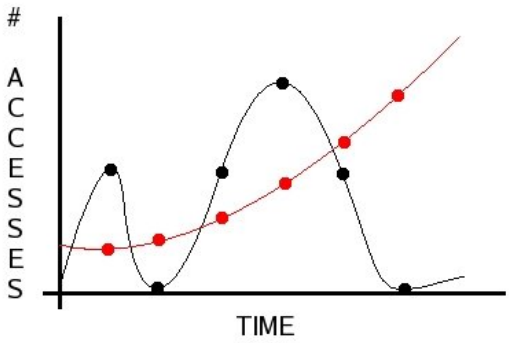

Fig. 4. The same graph with the exponential curve that defines its popularity

stripes in order to facilitate this process. By doing so, finding a suitable stripe for removal becomes a quick and efficient process.

IV. Implementation Issues

A. Indirection Map

For the purpose of indirection a stripe map is kept that allows UPGRAID to know where replicated RAID5 stripes are stored on the UPGRAID partitions. In addition to this data, a reverse map is kept for the purpose of removing stripes from the UPGRAID partitions. A write counter is also maintained so the system knows how many writes are pending to a stripe in the UPGRAID partition.

B. Replication Overhead

One issue is the overhead that is associated with replication of stripes. Even if the request was to a small portion of the stripe, the entire stripe must be read from the RAID5 partition and written to the UPGRAID partition. This overhead must be managed by the UPGRAID popularity system. This is done by still considering frequency, albeit to a lesser extent than recency. This would allow a stripe that has its use interrupted to not immediately be removed from the extended partition.

The other factor that requires experimentation is determining the threshold popularity to qualify for replication. Presently this threshold is a function of the maximum stripe ranking of the least popular (bottom-k) stripes.

C. Size of Extended Partition

At the present time ten percent of the individual drive space for the UPGRAID partitions is being used. A value that is too small would be prone to the replication overhead described above because stripes would be replicated too often. A partition that is too large would waste space on the disk and leave replicated stripes in the UPGRAID partitions that really are not that popular anymore. Determination of this optimal size requires additional experimentation with a fully stable system.

V. Related Work

Two works that took a similar vein to UPGRAID were Hot-Mirroring [1] and AutoRAID [2]. Both approaches followed the common idea of using a small portion of the drive space as a scratch pad to improve overall array function. All three approaches share the same basic disk partition schema.

A. Hot-Mirroring

Hot mirroring shares the same two-partition layout as UPGRAID. One of these partitions is the "RAID5 Cold Area" and the other is the "Mirrored Hot Area". The popular data is replicated to the hot area while the cold area retains basic RAID5 functionality. The main differences between UPGRAID and Hot-Mirroring are as follows...

- Determination of Popular Blocks - The popularity technique used by Hot-Mirroring is called "Hot-Block Clustering" [4]. The basic idea is that it takes advantage of access locality when deciding which data should be replicated. It assumes that all blocks of normal write accesses are hot and replicates them into the mirrored hot area. One problem is that, if a system containsa large amount of read-only data that is read on a regular basis, then this would not be moved into the hot area, posing a serious problem for such systems. By taking into account both reads and writes and replicating at the granularity of a stripe, UPGRAID is able to take advantage of both read-heavy and write-heavy loads while also taking advantage of the access locality of a stripe.
- Removal of Unpopular Block From Replication Area - Hot-Mirroring uses a defined migration phase in which to move blocks from the replication area to the RAID5 area. This approach decreases observed overhead as the migration does not take place during a user request but at the same time runs the risk of removing too much data. Such a case would be if data that is deemed cold but then is access shortly after its miration to the RAID5 partition. UPGRAID dynamically removes stripes from the replication area when a new stripe reaches a large enough popularity. Experimentation will reveal if this dynamic approach improves the overall system performance.
- Determining when Data is to be Removed - Hot-Mirroring's method for determining when data should be removed from the hot zone is based on the time since the last access to this data. While this is a simple metric, it does not take into account the possibility of data being accessed at regular, albeit larger, intervals. This creates the possibility of data that is momentarily popular taking precedence over data that is accessed constantly throughout the life of the system. By using recency and frequency of access UPGRAID guards against this kind of problem.

B. AutoRAID

AutoRAID also shares the same basic partition strategy that is employed in Hot-Mirroring and UPGRAID. The differences between AutoRAID and UPGRAID include...

- Defined Migration Phase - Like Hot-Mirroring AutoRAID uses a migration phase in which to move cold blocks from the replication area to the RAID5 area. This is done to account for sudden write bursts so there is space on the replication partitions for the new data. UPGRAID does this migration dynamically as new stripes are replicated.
- Focus on Writes - AutoRAID also focuses on write operations when looking at which data is to be replicated. This does not benefit read-heavy unless the same data is also write-active.

VI. Future Work

As of now the following tasks must be completed...

1) While the replication and indirection code has been written, it still needs to be tested under large synchronous and asynchronous loads.
2) The read heuristic mentioned in step (1) of read indirection has yet to be completed. The Linux RAID1 code does an operation similar to this. More investigation into their techniques would most likely produce a good approach to this problem.
3) The popularity code must be tested to ensure it has been properly incorporated into the UPGRAID system.

A. Reconstruction

Reconstruction is a large part of the overall UPGRAID system that has yet to be looked into. We explored two basic strategies...

1) One approach was the Disk-Oriented Approach (DOR) [5]. This basic idea of DOR is that a thread is created for each disk in the RAID array in addition to one master thread to control the reconstruction process. The master thread is responsible for request information from the individual disk threads while the disk threads

are tasked with retrieving the request information from their disk. This would most likely be a good starting point as it is a simpler algorithm than the second option.
2) The second approach was the popularity-based algorithm PRO [6]. PRO seems like a more attractive option as it fits into the overall popularity idea that UPGRAID is built upon. Here the reconstructed disk is separated into "hot zones", and then these zones are reconstructed based on their real time access popularity. At first glance it seems like this approach could tie in nicely to UPGRAID's popularity management to greatly speed up reconstruction.

VII. Conclusion

Since a stable version was not ready by the end of the summer session, the only conclusions that can be drawn at this point are along the lines of a proof of concept. The code that has been written thus far shows that everything that we want to do is possible within the Linux kernel, which is a good sign. With a little more polishing and testing, some hard experimental data could be generated and would give an idea of the impact of UPGRAID on a real system.

References

[1] K. Mogi and M. Kitsuregawa, "Hot Mirroring: A Method of Hiding Parity Update Penalty and Degradation During Rebuilds For RAID 5," *In Proceedings of the ACM SIGMOD International Conference On Management Of Data*, 1996.

[2] J. Wilkes, R. Golding, C. Staelin, and T. Sullivan, "The HP AutoRAID Hierarchical Storage System," *In Proceedings of the Symposium on Operating System Principles*, 1995.

[3] J. Hennessy, "The Future of Systems Research," *IEEE Computer*, vol. 32, no. 8, pp. 27–33, August 1999.

[4] K. Mogi and M. Kitsuregawa, "Hot Block Clustering for Disk Arrays with Dynamic Striping - exploitation of access locality and its performance analysis," *In Proceedings of the 21st VLDB Conference*, pp. 90–99, 1995.

[5] M. Holland, G. A. Gibson, and D. P. Siewiorek, "Fast, On-Line Failure Recovery in Redundant Disk Arrays," *In Proceedings of the International Symposium on Fault-Tolerant Computing*, pp. 422–443, 1993.

[6] L. Tian, D. Feng, H. Jiang, K. Zhou, L. Zeng, J. Chen, Z. Wang, and Z. Song, "PRO: A Popularity-based Multi-threaded Reconstruction Optimization for RAID-Structured Storage Systems," *FAST '07: 5th USENIX Conference on File and Storage Technologies*, pp. 277–290, 2007.