Proceedings of the 1996 Conference on Measurement and Modeling of Computer Systems (SIGMETRICS).,
May 1996, Vol. 24 No. 1, pp. 268-269.

RAIDframe: rapid prototyping for disk arrays

[1]William V. Courtright II, [2]Garth Gibson, [1]Mark Holland, [2]Jim Zelenka

[1]Department of Electrical and Computer Engineering,
[2]School of Computer Science,
Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA
{william.courtright, garth.gibson, mark.holland, jim.zelenka}@cmu.edu
http://www.cs.cmu.edu/Web/Groups/PDL/

In recent years, researchers have introduced a multitude of redundant disk array architectures. Unfortunately, using the traditional manual firmware-design approach employed by storage system designers, implementing control software for these redundant disk array architectures has led to long product-development times, yielded uncertain product reliability, and inhibited designers from exploring the benefits of new architectures. What is needed is array prototyping technology that makes it easier to experiment with design changes while also making it easier to ensure correct functioning of these changes.

We introduce RAIDframe, a framework for rapid prototyping and evaluation of redundant disk arrays. Using a graphical programming abstraction and a mechanized execution strategy, we are able to quickly construct working prototypes which can immediately be evaluated each of three environments: a device driver running against real disks, a user process running against real disks, or an event-driven simulator. This paper describes the basic structure of RAIDframe as well as our experiences with it.

1. Graphical Programming Abstraction

Composing multi-step storage operations as a list of primitive operations is a well established approach to storage control [Brown72]. While concurrency can be introduced into this abstraction by splitting the array controller into multiple processors each executing a linear sequence of primitives [Cao94], we advocate sequencing primitives with directed acyclic graphs (DAGs). We believe DAGs are a better representation because only the architectural dependencies between primitives are specified. In our experience, graphs like those of Figure 1 are precisely the tools needed to explain a new array architecture to a colleague. Furthermore, a structured representation enables mechanized execution, which we exploit to automate the effect of a device error on in-progress storage operations (graphs) [Courtright94].

Figure 1 illustrates a pair of DAGs used in our RAID level 5 implementation. The nodes represent primitive operations and the arcs represent dependences (control or data) which constrain execution. The primitives in this illustration are: H (header), T (terminator), R (disk read), W (disk write), and XOR (bit-wise exclusive or). The subscripts d and p denote the type of information, data or parity, being operated on. The “header” and “terminator” nodes have no function beyond enabling systematic initiation and termination of a graph.

2. Mechanized Execution

With a graph-based approach to representing array operation control, RAIDframe’s automatic-error-handling strategy, imple-

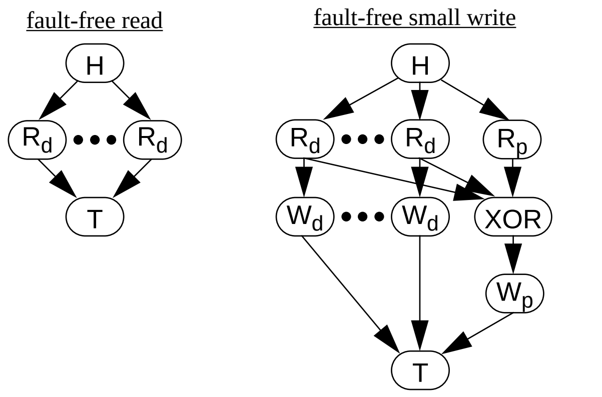

Figure 1 - RAID Level 5 Operations Represented as DAGs

mented in its architecture-independent DAG interpreter, either completes (as coded) graphs that are in-flight when an error occurs, *rolling forward*, or backs up through an in-progress graph to an earlier point, *rolling back*, where an alternative graph is more appropriate given the new state of the array. Similar to abort handling in transactional databases, this *roll-away* error handling works by identifying those nodes in a DAG which commit data to disk and by specifying the direction of recovery based on when errors occur in relation to this commit point. If an error occurs before any data has been committed, then the system rolls back, releasing resources, and retries the operation with a more appropriate graph. If an error occurs after data has been committed to disk, the system rolls forward through the remainder of the graph giving later requests the impression that this graph completed instantaneously before the error. It has been our experience that graph commit points can be specified so that roll-back is inexpensive (does not induce additional device work in the preparation for or the execution of roll-back) and roll-forward does not need to execute any device operation not already coded in the in-flight graph. It is our goal to demonstrate that an array’s peak throughput and average operation response time are not penalized by roll-away error handling.

3. Implementing an Array

The basic characteristics which differentiate redundant disk array architectures are mapping, encoding, and caching. These characteristics have been isolated (modularized) in RAIDframe, allowing them to be modified orthogonally. This modular approach, in conjunction with the use of DAGs and automated error recovery, greatly simplifies the process of implementing a new array architecture. RAIDframe provides libraries of mapping and encoding routines, nodes, graphs, and disk queueing policies which architects may either draw upon or enhance when creating new arrays.

At the time of this writing, we have completed implementation of RAID levels 0, 1, 4, 5, 6, as well as parity declustering [Holland94], distributed sparing [Holland94], interleaved declus-

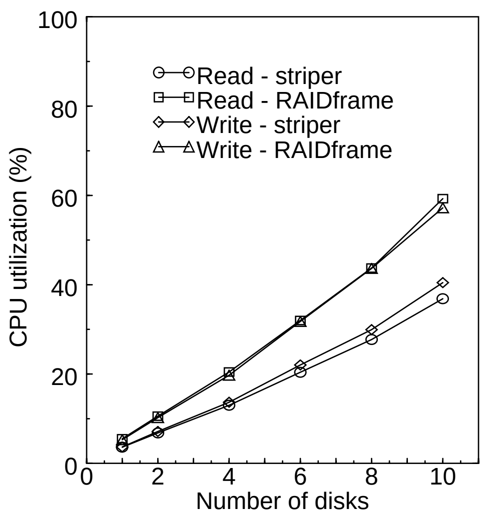

Figure 2 - RAIDframe CPU Utilization

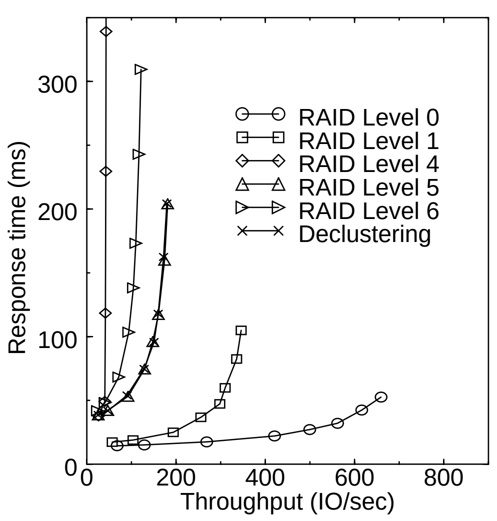

Figure 3 - Random 4K Writes, 10-Disk Array

tering [Teradata85], and chained declustering [Hsiao90]. We have consistently observed code reuse of more than 91% with an average reuse of 95%.

4. Accuracy and Performance Measurements

Using 10 HP-2247 drives connected to a 150 MHz DEC Alpha running OSF/1, we evaluated the performance of array architectures implemented in RAIDframe. First, comparing the performance of RAIDframe's RAID level 0, instantiated as a device driver, to a hand-crafted UNIX disk-striping driver, we found disk access throughput and response time distributions to be statistically identical. Of course RAIDframe's decomposition into operation graphs and a graph interpretation engine leads to a higher CPU utilization as Figure 2 illustrates for disk-saturating workloads composed of random 4 KB accesses.

With confidence that RAIDframe's basic engine can match the I/O performance of a hand-crafted disk-striping driver, we compared the I/O performance of different array architectures implemented in RAIDframe. Figure 3 presents average response time as a function of achieved throughput for a range of these architectures in a 10-disk array with a workload of random 4 KB writes. This figure displays the performance differences documented in the literature on RAID systems [Chen90, Holland94].

5. Conclusions and Future Work

RAIDframe represents redundant disk array architectures as simple programs in a graphical language that can be interpreted without sacrificing I/O performance. With this representation for an architecture, the effect of a device error on in-progress graphs is automated in RAIDframe.

Although functional as a simulator, user-level disk controller and kernel device driver, RAIDframe's extensibility and correctness during extension are under evaluation. We are developing an extensible disk cache to supports user-specified triggering mechanisms. Further information is available in a technical report [Gibson95] or on-line via our web page http://www.cs.cmu.edu/Web/Groups/PDL/.

Acknowledgments

The authors thank Khalil Amiri, Claudson Bornstein, Robby Findler, LeAnn Neal Reilly, Daniel Stodolsky, Alex Wetmore, and Rachad Youssef for their contributions to RAIDframe. This work was supported in part by Data General, DEC, H-P, IBM, Seagate, Storage Technology, and Symbios Logic. CMU's Data Storage Systems Center also provided funding from the National Science Foundation under grant number ECD-8907068. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of sponsoring companies or the U.S. government.

References

[Brown72] D. Brown, R. Gibson, and C. Thorn, "Channel and Direct Access Device Architecture," *IBM Systems Journal*, 11(3) (1972) 186-199.

[Cao94] P. Cao, S.B. Lim, S. Venkataraman, and J. Wilkes, "The TickerTAIP parallel RAID architecture," *Proc. of the 20th Ann. Int. Symp.on Computer Architecture*, (1993) 52-63.

[Chen90] P. Chen, et. al., "An Evaluation of Redundant Arrays of Disks using an Amdahl 5890," *Proc. of the Conf. on Measurement and Modeling of Computer Systems*, (1990) 74-85.

[Courtright, 1994] W.V. Courtright and G. A. Gibson, "Backward Error Recovery in Redundant Disk Arrays," *Proc. of the 20th Int. Conf. for the Resource Management and Performance Evaluation of Enterprise Computing Systems Computer Measurement Group (CMG94)*, (1994) 63-74.

[Gibson95] G. Gibson, W. Courtright II, M. Holland, and J. Zelenka, "RAIDframe: Rapid prototyping for disk arrays," Computer Science Technical Report CMU-CS-95-200, Carnegie Mellon University (1995)

[Holland94] M. Holland, "On-Line Data Reconstruction in Redundant Disk Arrays," Ph.D thesis, Computer Science Technical Report CMU-CS-94-164, Carnegie Mellon University, (1994).

[Hsiao90] Hsiao, H. and DeWitt, D. "Chained declustering: a new availability strategy for multiprocessor database machines." *Proc. of the International Data Engineering Conf.* (1990)

[Teradata85] Teradata Corporation. *DBC/1012 Database Computer System Manual (Release 1.3)*. c10-0001-01 Teradata Corporation (1985).