Microsoft Research Technical Report MSR-TR-2010-52, May 2010

Does erasure coding have a role to play in my data center?

Zhe Zhang
Oak Ridge National Laboratory

Amey Deshpande, Xiaosong Ma
North Carolina State University

Eno Thereska, Dushyanth Narayanan
Microsoft Research Cambridge

Abstract

Today replication has become the *de facto* standard for storing data within and across data centers that process data-intensive workloads. Erasure coding (a form of software RAID), although heavily researched and theoretically more space efficient than replication, has complex tradeoffs which are not well-understood by practitioners. Today's data centers have diverse foreground and background data-intensive workloads, and getting these tradeoffs right is becoming increasingly important. Through a series of realistic data center deployment scenarios and workload characteristics, coupled with the implementation of a prototype Hadoop library with erasure coding functionalities, we revisit traditional metrics (performance and dollar cost), present new tradeoffs (power proportionality and complexity) and make recommendations on directions worth researching.

1 Introduction

Today's modern data centers have "foreground" user-facing workloads and "background" data intensive analytical jobs that dig into the massive datasets to answer customer's inquiries or discover valuable insights. These data-intensive jobs often use *shipping code to data* frameworks, such as MapReduce [7] and Dryad [15]. We borrow the term Data-Intensive Scalable Computing (DISC [3]) workloads to describe the combination of user-facing and analytical workloads.

To provide better reliability and availability, a certain level of data redundancy is often needed. Many of the data centers today use N-way data replication. An alternative, erasure coding, has always been available and much research has gone into it [1, 13, 17, 18, 19, 20, 22, 24, 25]. Erasure coding provides potential storage and network savings to replication. For example, an m-of-n erasure coding scheme encodes unit data into n *fragments* of size $\frac{1}{m}$ such that any m of them reconstructs the original data. While 3-way replication and 3-of-5 erasure coding both tolerate 2 faults, the former requires $3\times$ storage consumption, while the latter only requires $1.6\times$. So, while tolerating the same number of faults, less data is written over the network onto storage.

So when is it appropriate to use erasure coding in practice? There are more complex tradeoffs involved than the above simple ones. The gains of erasure coding tend to be a fraction of its space saving depending on the ratio of storage cost to total system cost for modern data centers. The diversity of workloads also affects the benefits of erasure coding. *E.g.*, the network savings of erasure coding are clear when the workload is write-dominated. DISC workloads, composed of client-facing foreground and analytical background workloads, are both read and write dominated, however. Erasure coding pays prices for its savings too. It causes slower recovery from failures across data centers. It is also perceived as "complex" by developers in industry, though this factor has not been systematically evaluated.

With a goal of encouraging discussions on the merits of erasure coding for DISC workloads, this paper makes several contributions. We present realistic system deployment scenarios (e.g., mega data centers and container-based data centers) and workload characteristics. We analyze several real foreground workloads from online services, as well as background analytical workloads. For the latter, we have implemented a prototype Hadoop library with erasure coding functionalities, and this paper reports on tradeoffs involved, including code size and state dependency [5] complexity metrics.

2 Environments and metrics

Unlike traditional enterprise environments, DISC systems and many data centers today leverage commodity components, which are typically distributed and less reliable than customized hardware. Common fault tolerance solutions adopted by commercial storage servers, such as hardware RAID [17] or storage-area networks (SANs), are often considered too expensive and not incrementally scalable. DISC environments provide reliability and availability mainly through software-based redundancy. For example, replication is used by the Google File System [9] and Hadoop Distributed File System [14].

Storage is a key component in DISC environments, however, it is not the only one. Storage is often co-located with computation and networking resources. Analytical jobs have a *shipping code to data* requirement for efficiency. It must be possible to start analytical processes on the same servers the data resides.

An implementation in Hadoop: Erasure coding has workload-independent/dependent tradeoffs. To evaluate them we have extended HDFS with erasure coding[1]. Erasure coding is done at the block granularity (64MB in HDFS), meaning that an HDFS client does not erasure code within a single block, but waits to receive m blocks before calculating and writing $n - m$ parity blocks. The m original data blocks are still sent out as soon as they are ready, but a copy of each block is kept in memory to calculate parity data. We assume an environment with battery-backed memory to handle failures when m blocks are being accumulated. Our system falls back to replication when it can not accumulate m blocks within 30 seconds. We use the "$cauchy\_good$" algorithm in the Jerasure library [18] for erasure calculations.

Our implementation is the first one in Hadoop that performs erasure coding online, on the critical path. Another recent implementation, DiskReduce [8], first performs replication, then erasure codes data in the background. In contrast, our system erasure codes data chunks when they are created, therefore also reducing the network and disk bandwidth usage.

Metrics: In evaluating the impact of erasure coding, we discuss both traditional and new metrics. The former include performance, recovery time and infrastructure costs (in $). To these metrics we add a complexity and an energy metric. In evaluating complexity we consider both the code size and the NetComplex metric introduced by Chun *et al.* [5]. The energy metric has two components. The first is regarding the amount of energy consumed per task or per GB of stored data. The second is a function that relates energy consumption to the load seen by the system. In particular, we assume the system is employing "offloading" of requests as described in our previous work [16] to enable power proportionality.

3 Workload-independent tradeoffs

Dollar savings: Table 1 shows the fraction of storage cost for a mega-data center (Mega DC [12]) and a Condo-based machine room [6]. The cost of a "wimpy" server (FAWN [23]) is shown as a reference. A main takeaway from this table is that the cost of raw storage space is relatively small, unless the whole server is a storage node (i.e., 50-90% of the server cost is storage-related). Using a 3-of-5 erasure coding instead of the (pervasively used) 3-way replication is expected to save in the range of (8.4-14-25%) of the total cost for the Mega DC and (12-20-37%) for the Condo-based one respectively. For a cluster based on FAWN nodes, the savings are around 18%.

	# servers	Cost/year	Storage cost
Mega DC [12]	54k	$81M	14-23-41%
Condo [6]	48	$71K	20-34-61%
FAWN [23]	1	$383	30%

Table 1: Storage cost in several sample systems. For Mega DC and Condo, a 15 year amortization cost for building and 3 year amortization cost for server is assumed. According to [6], a server in Condo or Mega DC costs $3000. We use three disk cost points to estimate the storage cost: 30%, 50% and 90% of the server cost ($900, $1500, $2700 per server spent on disks).

Cost during large-scale recovery: A main penalty paid by erasure coding is decreased performance during recovery (when compared to replication). This might be a serious concern when erasure coding is considered as the redundancy mechanism across data centers, or across availability domains within a large data center. Let's consider a power outage (e.g., the one that took down parts of Amazon's EC2 recently [2]) that fails one availability domain. For both replication and erasure coding, data reconstruction is needed to restore the redundancy level. Whereas with replication clients can simply switch to using another availability domain, with erasure coding m domains might need to be contacted for every read.

What about complexity?: The number one response from developers when asked about using erasure coding is that "it is too complex". Quantifying complexity is subjective. We provide two quantitative arguments here. The first focuses on the notion of "state dependencies", as first defined by Chun *et al.* [5]. This metric captures the number of nodes with state that must be contacted before an algorithm can complete. Intuitively, the larger the number of such nodes, the more things can go wrong, stalling progress and making debugging harder.

When data is written to the system, with either scheme, each hosting server is receiving a "single input, 1-hop unicast" from the data source. This scenario has a complexity of $1 + t$, where t refers to the complexity associated with the network path relaying the message, and is solely determined by the network topology of the system. During data recovery (and reads), however, erasure coding has higher complexity because more servers need to be involved. Identical to the "single input, 1-of-m paths" and "single input, k-of-m paths" scenarios ([5]) respectively, the complexity of recovery is $1 + m \cdot t$ under m-of-n erasure coding, and $1 + t$ under replication.

The second metric is based on code size. In our implementation of erasure coding in Hadoop, $\sim 1,800$ lines of Java code is added or modified, out of a total of $\sim 33,500$

[1]Available at http://research.csc.ncsu.edu/palm/hadoop-ec.htm

lines in HDFS. $\sim 1,500$ lines are on the HDFS client (originally $\sim 2,800$ lines), while the other ~ 300 lines are for bookkeeping of parity data. In addition, the Jerasure package is used for the calculation of parity data, involving another $\sim 1,700$ lines of C code (the entire library contains $\sim 7,500$ lines). We want to stress that this code can be written once and packaged in a library, and is unlikely to require continuous maintenance (most functions perform well-studied mathematical calculations).

Summary: Table 2 depicts the main workload-independent tradeoffs when erasure coding is used with a fixed n. The size of the arrows needs to be normalized by the fraction of storage subsystem cost, latency and complexity to the overall system.

$m \Uparrow$	storage cost $\Downarrow$	latency $\Uparrow$	complexity $\Uparrow$

Table 2: Main workload-independent tradeoffs.

4 Workload-dependent tradeoffs

Reads and writes: Clearly erasure coding favors write-intensive workloads as mentioned in Section 2. Table 3 shows properties of IO traces (underneath the buffer cache) for several foreground applications running at Microsoft's data centers. It also shows three background analytical jobs. We briefly describe these workloads below.

Hotmail: 48-hour email workload, containing both data and metadata accesses. Email contents might be continuously mined for spam detection and ad placement.

BLOB-DB (Windows Live Blob Metadata Server): These metadata lookup servers hold user account mappings for various blob storage services such as online photos, videos, social networking, etc., over a period of 24 hours.

MSN-DB (MSN Content Aggregation Database Server): This 24-hour trace is taken from a content publishing system for an online portal front page and is updated by mainly editorial tools and feed management systems via web services. Most of the stored data is unstructured in nature, consisting of either raw content or links to content on other partner sites.

Analytical jobs: We run three analytical jobs under Hadoop, including two typical benchmarks on synthetic data (*sort* and *wordcount*) and one real bioinformatics application (*CloudBurst* [21]). The *sort* job has almost the same amount of input, intermediate and output data. Assuming the input data is (already) written to the system by foreground jobs, the read:write ratio for just the processing stages is 0.67. The *wordcount* job, which counts the occurrence of each word in a document, has much less output data than input and intermediate data, and the read:write ratio is 5.04. *CloudBurst* is a DNA sequencing application, which returns a set of entries from a reference genome dataset according to user specified matching conditions. A typical *CloudBurst* run produces only a small amount of output data, but intermediate data size could be much larger than input. The overall read:write ratio is 0.74.

Workload	R:W ratio	TB
Hotmail	0.85	66
MSN-DB	1.99	0.4
Blob-DB	1.67	0.26
Analytical job - sort	0.67	N/A
Analytical job - wordcount	5.04	N/A
Analytical job - CloudBurst	0.74	N/A

Table 3: Read:write ratio for foreground and background workloads, along with data size in TBs, identified using the largest logical-block number (LBN) seen in the trace. The higher the R:W ratio, the smaller the benefits of using erasure coding.

Table 3 shows that our servers are provisioned as much for reads as they are for writes. Thus, the network savings from using erasure coding need to be appropriately scaled by the R:W ratio.

Network topology and replication policies: The network topology and replication policies matter when examining the benefits of erasure coding. Consider an environment where each node is both a client and a storage server. In that case, a node producing or collecting a chunk of data might be one of its hosting nodes, meaning that m-of-n erasure coding needs to send $\frac{n-1}{m}$ chunks to remote nodes for each produced chunk, while N-way replication sends $N-1$. In this case, 3-of-5 erasure coding, for example, would use 33% less network bandwidth than 3-way replication rather than 45% as in a topology where storage clients and servers run on disjoint nodes.

Data replication policies also have an impact in determining the network bottleneck. For example, if the client sends N replicas to N nodes, it is likely that the client's NIC will be a bottleneck (e.g., a 1Gbps NIC will only have a "goodput" of close to 30MB/s for 3-way replication and close to 60MB/s for the 3-of-5 scheme). But, when we used Hadoop's chain replication policy for the analytical jobs, the client NIC was not a bottleneck anymore (however, switches could become a bottleneck depending on network activity from other clients).

Of course, more data is transmitted if the input, intermediate, and output files of an analytical job are protected with replication rather than erasure coding when they are generated. However, the exact end-to-end effects on throughput and latency might not simply scale linearly with amount of data sent. For concreteness, to stage a 6GB file into an HDFS of 8 data servers, it took 246 seconds with no redundancy, 393 seconds with 3-way replication, and 389 seconds with 3-of-5 erasure coding. The replication policy, network topology, and interactions with other client's data all influence end-to-end performance.

A closer look at analytical jobs: We examine the impact of encoding schemes in different phases of the three analytical jobs mentioned in Section 4. The *sort* and *wordcount* jobs process 6GB of data while *CloudBurst* uses a 4GB input reference file. Our testbed is a cluster of 8 machines interconnected with 1Gbps Ethernet. The execution times of the three jobs with different encoding schemes are shown in Figure 4, decomposed into *map*, *shuffle* and *reduce* phases. All experiments are performed 3 times, with negligible variance.

Sort's *map* and *reduce* phases are write-intensive. 5-of-7 erasure coding has an improvement of 51% over 3-way replication in *reduce* run time, the most significant win of erasure coding out of all phases. The *shuffle* phase mainly partitions the intermediate data on mappers and merges the received partitions into sorted order before reducers run. The performance of this stage is somewhat degraded by the bandwidth consumption of replicating intermediate data. Overall, 5-of-7 erasure coding reduces 35% of run time over 3-way replication, while 3-of-5 performs slightly worse than 5-of-7.

In the *wordcount* job, however, the output volume is less than $\frac{1}{20}$ of input. As a result, encoding schemes have much smaller impact on the overall performance. Difference in run time among all schemes is less than 5%.

Similar to *wordcount*, *CloudBurst* also outputs much less data than input. However, the amount of intermediate data is relatively large. Therefore, erasure coding wins over replication by saving the network and disk bandwidth in *map* and *shuffle* stages, the latter of which dominates the total run time. The overall improvement of 5-of-7 encoding over 3-way replication is 29%. 3-of-5 achieves a similar enhancement.

One may suspect that erasure coding hurts data locality for MapReduce tasks – for each data chunk, having multiple distributed replicas reduces remote accesses with Hadoop's locality-aware task scheduling. However, we have found that even without any replication, the great majority of blocks are accessed locally. Compared with no replication, 2-way replication improves the percentage of local accesses from 78% to 91% (*sort*), from 88% to 95% (*wordcount*), and from 82% to 93% (*CloudBurst*). 3-way replication hardly provides any additional improvement. Therefore, the negative impact of employing erasure coding on data locality is quite limited.

Power efficiency and proportionality: When the servers are fully utilized, the amount of Joules/task will clearly depend on the properties of the task (e.g., write- vs. read-dominated). This metric is largely correlated with performance and the same tradeoffs as discussed above apply. To verify this, we measured the entire job's energy consumption on 8 nodes using power meters attached to each machine. During the run time of the three analytical jobs we have observed that different encoding schemes (including replication) have similar Watt levels. Their total energy consumptions are therefore approximately proportional to the execution time.

When the workload intensity decreases (many foreground workloads have diurnal patterns) we have advocated turning off servers [16] for better power proportionality. The number of "gears" for a general m-of-n scheme is $n - m + 1$. For example, 3-way replication has three gears (with 1, 2, or all 3 servers ON); 3-of-5 has three gears too (with 3, 4 and 5 servers ON). The mechanism for turning off servers has been discussed previously [16] and is beyond the scope of this paper. It is interesting to note that, once the data center has been (over-)provisioned for replication, more machines can be turned OFF than when erasure-coding is used.

Summary: The main workload-dependent tradeoffs, shown in Table 4, are complex. Workload characteristics (e.g., read:write ratio, access size, time spent in different of MapReduce stages, etc.), network topology, and policies all matter to gauge the wins from erasure coding. The size of the arrows for the performance metrics needs to be normalized by the fraction of requests that benefit (read:write ratio). "?" stands for "it depends".

$n \Uparrow$	net.bw. $\Uparrow$	write latency/thrpt. ?	gears$\Uparrow$
$m \Uparrow$	–	read latency ?	gears$\Downarrow$

Table 4: Main workload-dependent tradeoffs.

5 Related work

The inspiration for using erasure coding to archive data comes from the classic work on RAID [17] and later AutoRAID [25]. This latter work valued savings from such codes because storage space is a significant fraction of the cost of a disk array. This holds true for a SAN too, and more generally for a system whose only role is to storage data durably [4]. Our work is orthogonal to the above. We examine tradeoffs when using erasure codes for non-archival workloads in data center environments, where computation, storage and networking are co-located (i.e., no disk arrays or SANs). In such environments, commodity servers are being used. Unlike traditional enterprise, where expensive RAID enclosures and SCSI disks might be pervasive [4], in our environments the trend points to the FAWN [23] approach (eventually).

The tradeoffs of erasure coding and replication have been

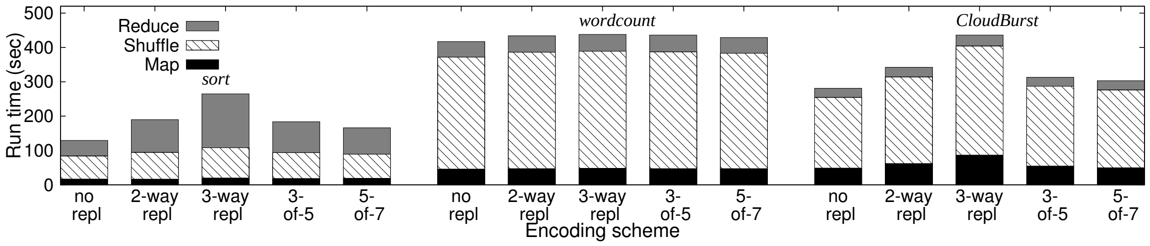

Figure 1: Execution time of analytical jobs

studied, but mostly from the performance perspective. For example, the performance tradeoffs are known [18, 22, 24] and so is recovery time [17, 19, 24, 25]. We go beyond single metric evaluations and provide a higher-level discussion based on total system cost (hardware, design complexity, and energy consumption) for the new paradigm of data-intensive services.

The mechanisms for building distributed systems that use erasure coding have been studied in academia [1, 8, 13] and industry [20]. Work to make erasure coding more power efficient is ongoing [11]. Our understanding of erasure codes has benefited from these approaches.

6 Conclusions and directions

This paper aims to give a comprehensive overview of virtues of erasure coding vs. replication for modern data centers and encourage discussions around the topic. Our analysis and experimental results show that the storage cost advantage of erasure coding over replication ranges from 8.4% to 37%, depending on the data center setup. Its performance advantage is highly workload-dependent and varies from $<$5% to $>$50%, to which the energy advantage is highly correlated. Depending on the system and workload scale, such gains may or may not justify the higher data recovery cost and software complexity.

One interesting topic for future research is to make data redundancy related recommendations for a given system setup and workload set, considering the many axis of tradeoffs. It would also be interesting to analyze the role of erasure coding on geo-distributed data. It is likely that certain applications will require the data to be stored across continents/countries, in multiple data centers. Would it make sense to replicate the data within and across data centers? What will the cost of such pervasive replication be? Would the use of erasure codes make sense within or across the data centers?

7 Acknowledgments

We thank Heshan Lin for helping us set up the Hadoop experimental environment and obtain the CloudBurst workload. This research is sponsored by Microsoft Research Cambridge, together with NSF grants CNS-0546301 and CNS-0915861. In addition, the work is supported by Xiaosong Ma's joint appointment between NCSU and ORNL.

References

[1] M. Abd-El-Malek, W. C. II, C. Cranor, G. Ganger, J. Hendricks, A. Klosterman, M. Mesnier, M. Prasad, B. Salmon, R. Sambasivan, S. Sinnamohideen, J. Strunk, E. Thereska, M. Wachs, and J. Wylie. Ursa Minor: versatile cluster-based storage. In *Proceedings of the 4th USENIX Conference on File and Storage Technologies (FAST '05)*, pages 59 – 72, 2005.

[2] Lightning knocks out Amazon's compute cloud. http://java.sys-con.com/node/998582.

[3] R. Bryant. Data-intensive supercomputing: The case for DISC. Technical Report CMU-CS-07-128, 2007.

[4] F. Chang, M. Ji, S.-T. Leung, J. MacCormick, S. Perl, and L. Zhang. Myriad: Cost-effective disaster tolerance. In *FAST '02: Proceedings of the 1st USENIX Conference on File and Storage Technologies*, page 8, Berkeley, CA, USA, 2002.

[5] B.-G. Chun, S. Ratnasamy, and E. Kohler. Netcomplex: a complexity metric for networked system designs. In *NSDI'08: Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, pages 393–406, Berkeley, CA, USA, 2008. USENIX Association.

[6] K. Church, A. Greenberg, and J. Hamilton. On delivering embarrassingly distributed cloud services. In *HOTNETS'08: Proceedings of the 7th ACM Workshop on hot topics in networks*, 2008.

[7] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. In *OSDI'04: Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation*, pages 10–10, Berkeley, CA, USA, 2004. USENIX Association.

[8] B. Fan, W. Tantisiriroj, L. Xiao, and G. Gibson. Diskreduce: Raid for data-intensive scalable computing. In *Proceedings of the 4th Petascale Data Storage Workshop in Supercomputing '09*, 2009.

[9] S. Ghemawat, H. Gobioff, and S. Leung. The Google file system. In *Proceedings of the 19th Symposium on Operating Systems Principles*, 2003.

[10] K. Greenan, D. Long, E. L. Miller, T. Schwarz, and J. Wylie. A spin-up saved is energy earned: Achieving power-efficient, erasure-coded storage. In *HotDep '08: Proceedings of the 4th conference on Hot Topics in System Dependability*, Berkeley, CA, USA, 2008. USENIX Association.

[11] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel. The cost of a cloud: Research problems in data center networks. In *ACM SIGCOMM Computer Communication Review*. ACM, 2009.

[12] A. Haeberlen, A. Mislove, and P. Druschel. Glacier: highly durable, decentralized storage despite massive correlated failures.

In *NSDI'05: Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation*, pages 143–158, Berkeley, CA, USA, 2005. USENIX Association.

[13] Hadoop distributed file system. http://hadoop.apache.org/core/docs/current/hdfs_design.html.

[14] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In *EuroSys '07: Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, pages 59–72, New York, NY, USA, 2007. ACM.

[15] D. Narayanan, A. Donnelly, and A. Rowstron. Write off-loading: practical power management for enterprise storage. In *FAST'08: Proceedings of the 6th USENIX Conference on File and Storage Technologies*, pages 1–15, Berkeley, CA, USA, 2008.

[16] D. A. Patterson, G. Gibson, and R. H. Katz. A case for redundant arrays of inexpensive disks (RAID). *SIGMOD Rec.*, 17(3):109–116, 1988.

[17] J. S. Plank, J. Luo, C. D. Schuman, L. Xu, and Z. Wilcox-O'Hearn. A performance evaluation and examination of open-source erasure coding libraries for storage. In *Proceedings of the 7th USENIX Conference on File and Storage Technologies (FAST)*, pages 253–265, 2009.

[18] R. Rodrigues and B. Liskov. High availability in dhts: Erasure coding vs. replication. In *Proceedings of the 4th International Workshop on Peer-to-Peer Systems*, 2005.

[19] Y. Saito, S. Frølund, A. Veitch, A. Merchant, and S. Spence. Fab: building distributed enterprise disk arrays from commodity components. In *Architectural Support for Programming Languages and Operating Systems*, pages 48–58. ACM, 2004.

[20] M. Schatz. Cloudburst: highly sensitive read mapping with mapreduce. *Bioinformatics*, 2009.

[21] E. Thereska, M. Abd-El-Malek, J. J. Wylie, D. Narayanan, and G. R. Ganger. Informed data distribution selection in a self-predicting storage system. In *International conference on autonomic computing*, pages 187–198, 2006.

[22] V. Vasudevan, J. Franklin, D. Andersen, A. Phanishayee, L. Tan, M. Kaminsky, and I. Moraru. FAWNdamentally power-efficient clusters. In *HOTOS'09: Proceedings of the 12th USENIX workshop on Hot topics in operating systems*, Berkeley, CA, USA, 2009. USENIX Association.

[23] H. Weatherspoon and J. Kubiatowicz. Erasure coding vs. replication: A quantitative comparison. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems*, 2002.

[24] J. Wilkes, R. Golding, C. Staelin, and T. Sullivan. The HP AutoRAID hierarchical storage system. *ACM Transactions on Computer Systems*, 14(1):108–136, February 1996.