Logan: Automatic Management for Evolvable, Large-Scale, Archival Storage

Mark W. Storer, Kevin M. Greenan, Ian F. Adams
Ethan L. Miller, Darrell D.E. Long
Storage System Research Center
University of California, Santa Cruz
{mstorer, kmgreen, iadams, elm, darrell}@cs.ucsc.edu

Kaladhar Voruganti
NetApp
Kaladhar.Voruganti@netapp.com

Abstract—Archival storage systems designed to preserve scientific data, business data, and consumer data must maintain and safeguard tens to hundreds of petabytes of data on tens of thousands of media for decades. Such systems are currently designed in the same way as higher-performance, shorter-term storage systems, which have a useful lifetime but must be replaced in their entirety via a "fork-lift" upgrade. Thus, while existing solutions can provide good energy efficiency and relatively low cost, they do not adapt well to continuous improvements in technology, becoming less efficient relative to current technology as they age. In an archival storage environment, this paradigm implies an endless series of wholesale migrations and upgrades to remain efficient and up to date.

Our approach, Logan, manages node addition, removal, and failure on a distributed network of intelligent storage appliances, allowing the system to gradually evolve as device technology advances. By automatically handling most of the common administration chores—integrating new devices into the system, managing groups of devices that work together to provide redundancy, and recovering from failed devices—Logan reduces management overhead and thus cost. Logan can also improve cost and space efficiency by identifying and decommissioning outdated devices, thus reducing space and power requirements for the archival storage system.

I. INTRODUCTION

The ability to store and maintain massive quantities of data is becoming increasingly important, as scientists, businesses, and consumers are increasingly aware of the value of archival data. Scientists have long attempted to preserve data archivally, though such efforts have sometimes fallen short. In the business arena, data preservation is often mandated by law [1, 17], and data mining has proven to be a boon in shaping business strategy. For individual consumers, archival storage is being called upon to preserve sentimental and historical artifacts such as photos, movies and personal documents. Unfortunately, traditional storage systems are not designed to meet the needs of long-term, archival data [6]. Paradoxically, despite the increasing value of archival data, high cost is one of the biggest obstacles to applying traditional storage techniques to design systems to house archival data. Long-term storage systems should be inexpensive enough to allow the preservation of all data that *might* eventually prove useful.

In contrast to traditional storage systems, which are typically more concerned with scalability in performance and capacity, an archival storage system designed for long-lived data must scale over many dimensions, including time, vendors and technologies [6]. The goal therefore, is to move away from an endless series of migrations and "fork-lift" upgrades, to a continuously evolving system. To realize this goal, we are developing Pergamum, a system that consists of a distributed network of up to 10^5–10^6 energy-efficient storage devices, called *tomes*, that communicate over a commodity Ethernet backplane [26]. While these devices can operate independently, their full potential is realized when they cooperate in inter-device redundancy groups to provide data reliability and ensure data longevity. Since the storage nodes are intelligent, each device contains specialized software that acts as an abstraction layer between the system, and the device's underlying hardware. This flexibility provides the potential for an adaptable system that changes gradually with technology; while individual components may change, the overall system evolves gracefully.

Although a fully distributed system is well-suited to an evolvable design, it introduces the problem of managing the global state of a fully decentralized system. Recent work has made great strides towards efficiently aggregating data over very large networks of distributed nodes [31, 32]; however, these approaches may be insufficient in a system that could easily encompass millions of nodes. A million-node distributed system must facilitate nodes joining the system, manage placement of data and redundancy information, handle node failure, and gracefully phase out nodes as they age. All of

this must be done with no central point of failure. However, it is impractical for each node to maintain global knowledge—keeping just 10 KB per node for each of a million nodes would require 10 GB of storage per node. Moreover, keeping that information current would require far too many messages to be exchanged between nodes. Instead, an archival system must allow nodes to operate with partial knowledge of the whole system and more complete knowledge of a small part.

Archival systems become more useful as their cost decreases, making energy efficiency an important aspect of long-term storage. However, while some earlier systems have addressed energy efficiency [8, 11, 18], none have examined how opportunity costs affect a system over its lifetime. Since drive capacity, real estate values and power costs are always increasing, system efficiency must be measured against what is currently achievable, not simply what was *once* achievable. For example, most storage systems assume that drives are replaced due to failure or wholesale system upgrades, suggesting that drives may remain in use well past the point of being economically efficient. Further, proactive decommissioning could also improve system reliability; earlier work has shown that the previously-held bathtub failure model for hard drives may not be valid [24], and that even small numbers of sector failures can presage overall drive failure [5].

Because our overall goal is to reduce the cost of long-term archival storage by reducing management cost, we designed Logan to address several major challenges. First, Logan must automatically integrate new nodes into an existing system. This includes both making the system aware of the new device, and integrating the new device's storage into redundancy groups to protect the data it stores against device failure. Second, Logan must ensure the correctness and longevity of data stored redundantly across nodes in the face of changing device membership and failures. Third, Logan should strive to keep power and other ongoing costs as low as possible by monitoring a device's *usefulness*—the utility it provides compared to the resources it consumes—and decommissioning the device when it has outlived its useful lifespan.

The remainder of the paper is organized as follows. Section II places Logan within the context of existing research. Next, Section III provides an overview of our current design. Finally, in Section IV we discuss where we plan to focus our future efforts; we conclude the paper in Section V.

II. RELATED WORK

In designing Logan to meet the goals of energy-efficient, reliable, archival storage [6], we used concepts from a variety of projects. In this section, we place our work in the context of existing work by presenting an overview of relevant existing storage systems, and distinguishing them from Logan by identifying their architecture, intended workload, and cost strategy.

A. Distributed Communication

Because distributed systems are composed of loosely coupled, independent devices, system-wide communication can be challenging. An extreme approach is the pursuit of global awareness, in which a fully connected graph allows one-hop communications between any two nodes [2, 12, 14]. Unfortunately, the per node storage overhead, and proliferation of messages with these strategies make them suitable for only relatively small systems. Some systems have sought efficiency gains through the use of randomization, but even these approaches incur relatively heavy costs [7, 28].

Information management systems attempt to provide system level awareness, while still maintaining a decentralized, distributed architecture. Some, such as SDIMS and Shruti, utilize DHT algorithms as part of their foundation [31, 32]. These information management systems aggregate information about a distributed system's state, and make it available in a way that does not collect all of the information in a central point of failure. Another approach to data aggregation is seen in Astrolabe, which eschews DHTs in favor of a gossip-based approach [21]. Unfortunately, this approach focuses more on data summaries that data aggregation and can be inefficient for some workloads.

B. Storage Systems

Distributed, peer to peer architectures have been used in storage systems geared to a variety of workloads. Logan, which is designed to run on a Pergamum-style archival storage architecture [26], relaxes some performance criteria in exchange for economic efficiency, unlike many existing systems that attempt to achieve performance levels on par with traditional, centralized storage [22, 23].

Logan is designed for a fully distributed architecture with no need for specialized nodes and central repositories of information. In long-term storage scenarios, centralized points of failure can compromise data longevity. For example, Venti, which can store archival data on removable media, utilizes a centralized index [20]. Unfortunately, while this index can be rebuilt by reading in partial indices stored on media, the bottleneck of media readers makes this a prohibitively lengthy procedure as overall data size increases to hundreds of petabytes and

beyond. Similarly, the Google File System is a semi-distributed system that utilizes master nodes. However, as it was not designed for long-term archival data, it avoids the recovery problem by relaxing consistency and longevity constraints [9].

Energy efficiency is an area that many designs have explored in pursuit of cost savings. Some reports state that commonly used power supplies operate at only 65–75% efficiency, representing one of the primary culprits of excess heat production, and contributing to cooling demands that account for up to 60% of data-center energy usage [10]. The development of Massive Arrays of Idle Disks (MAIDs) generated large cost savings by leaving the majority of a system's disks spun down [8]. Further work has expanded on the idea by incorporating strategies such as data migration, the use of drives that can spin at different speeds, and power-aware redundancy techniques [16, 18, 19, 29, 33, 35].

Finally, a number of projects have sought to yield cost savings from improved management techniques [3]. IBM's Intelligent Bricks project utilizes intelligent storage appliances in liquid cooled rack designed for very high device density [30]. Unfortunately, the design intentionally forgoes accessibility of nodes as a trade off for higher density. In a long-term scenario, this implies that eventually failed nodes will consume floor space while providing no utility. Others, such as Sun's Honeycomb project, trade energy efficiency for management gains [27]; in some scenarios, as many as sixty disks could be involved with a single write.

III. Proposed Design

The system we envision is driven by a workload that exhibits read, write and delete behavior that differs from typical disk-based workloads, providing both challenges and opportunities. The workload is write-heavy, motivated by regulatory compliance and the desire to save any data that *might* be valuable at a later date. Reads, while relatively infrequent, are often part of a query or audit and thus are likely to refer to data that is temporally correlated. Similarly, data being deleted are likely to exhibit a temporal relationship since retention policies often specify a maximum data lifetime. This workload resembles traditional archival storage workloads [20, 34], adding deletion for regulatory compliance.

We are currently developing Logan, a management layer that runs atop the distributed network of inexpensive, independent storage appliances provided by Pergamum [26]. This architecture provides a number of advantages in a long-term archival scenario. First, each device acts as an abstraction layer to the underlying media, easing transitions to new technologies. Second,
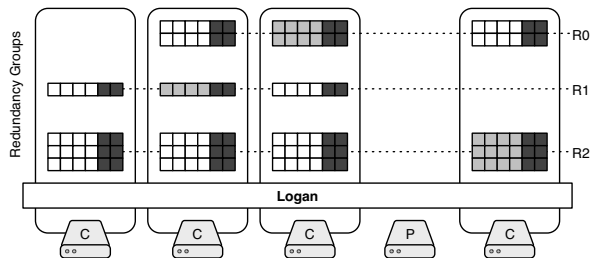

Fig. 1. Overview of Logan running a distributed network of devices. One device, *P*, is in the PENDING state, while the others, *C*, are CONTRIBUTING in redundancy groups. Data blocks (white) are protected with internal parity (dark grey) and external parity.

using a high number of low powered processors yields energy savings versus a few high powered processors (cutting processor voltage in half results in half the clock speed but one fourth the power consumption). Third, an inexpensive node can be treated as an indivisible entity; if any part of the node fails, the entire node is discarded and replaced. This reduces the management overheard associated with locating and replacing individual components.

While each device is independent and actively ensures the longevity of its own data, nodes also cooperate in *redundancy groups* to provide system wide data reliability. Data in each device is divided into fixed sized *blocks*, and blocks are grouped into fixed sized *segments*. In a large archival system, data failures will be quite common, and thus a "one size fits all" approach to reliability is excessively wasteful. Figure 1 illustrates the two-level reliability model used in Pergamum [26], the system on which Logan runs. First, each device survives media faults by utilizing block-level erasure coding over segments [4]. Second, distributed RAID techniques are used over groups of segments to survive the loss of a device [25]. Since heterogeneity is inevitable in an evolvable system, device capacity will vary between appliances. Thus, unlike a simple RAID system where all drives are the same size, a device can contribute segments to more than one redundancy group in order to utilize all of its local storage capacity.

A. Management Groups

Because global knowledge becomes increasingly infeasible as the number of nodes increases, devices are arranged into *management groups*. Each management group chooses one or more group leaders that oversees administration for a given term. At a system wide level, the management groups are arranged in a logical hypercube because this topology offers a number of benefits. First, it offers efficient communications routing. As shown in Figure 2, message routing occurs in a
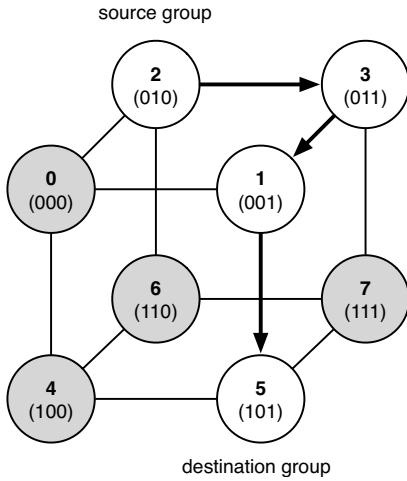
Fig. 2. Eight management groups arranged in a hypercube of dimension 3. Nodes involved in routing from group 2 to 5 shown in white. Routing is done in $O(\lg n)$ time, since each hop brings the message one bit closer to its destination.

$$C \oplus P \vee G1 = N1 = 100 < C$$
$$C \oplus P \vee G2 = N2 = 111$$

Fig. 3. When parent P (node 2), produces a child C (node 6), it provides the child with a list of its grandparents $G1$ and $G2$ (nodes 0 and 3). The child then calculates which, if any of its bitwise neighbors it needs an introduction to from its grandparent.

hierarchical fashion, with messages first routed to the destination group, and finally routed within the destination group. Second, management groups grow until they reach a certain size, and then are split into two groups. This technique for incremental growth in the number of management groups is well suited to the construction of hypercubes.

The system begins with a single management group. Nodes entering the system are added to this group until it reaches a predetermined saturation point, at which point it splits into two groups with an average of half the membership each. Membership and splitting is based on the LH* family of distributed data structures [15]. This approach offers several benefits. First, these data structures do not require a globally consistent view in order to function properly. This property is especially important because, in a system of the scale we envision, tight consistency expectations are unrealistic. Second, they allow the system to gracefully scale from a small system of just a single management group to a large system with thousands. Third, since group membership is calculated instead of statically assigned, a node can be located from its name alone.

LH* utilizes two variables, n and i, to coordinate all of operations. First, system routing utilizes these variables in the hash function used to determine which management group a node belongs to. In the event that a node is routed using an older version of n and i, the selected bucket will route the message to the correct bucket, as well as update the source with the up to date variables values. Second, n acts a token allowing
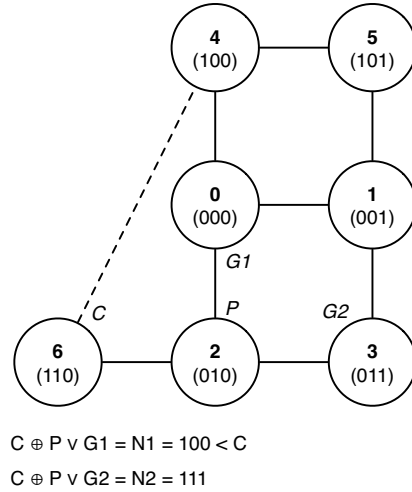
distributed splitting; when bucket n splits, it passes the token to bucket $n + 1$ mod 2^i.

When a group splits, it must establish connections with its bitwise neighbors, N, in order to preserve the logical hypercube. One such connection is from the child, C, to its parent, P. This connection is easy to make. Additionally, it must establish a connection with each existing group whose name is exactly one bit different than its own name. To perform this, the parent supplies the child with a list of its grandparents, G. For each grandparent, the child calculates $C \oplus P \vee G = N$. If $N < C$, then C asks G to make an introduction. If $N \geq C$, then that bitwise neighbor has not yet been formed and no further action is required. This process is illustrated in Figure 3.

B. Device Lifetimes

When a new node enters the system, it performs a broadcast to nearby nodes to locate other Logan devices; there is no need for the new node to broadcast to all (or even most) of the nodes in the system. The devices that respond are able to provide the new node with their view of the global system state, which in turn is used by the new node to calculate its management group. Once a new node has calculated its management group, it asks for further assistance in locating a fellow member of its group. Eventually the new node is introduced to the current group leader.

Once a node has become member of the system, it is managed through its entire lifespan, progressing through three states: PENDING, CONTRIBUTING, and EXPIRED.
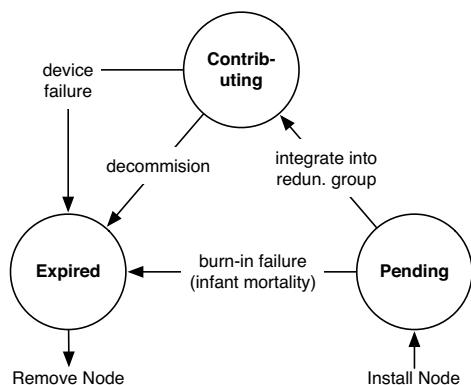
Fig. 4. Nodes in the system exist in one of three states. A functioning node that is not yet part of any redundancy groups is in the initial, PENDING state. CONTRIBUTING nodes have been integrated into redundancy groups. EXPIRED nodes have either failed or been decommissioned, and can be removed from the system.

The first state, PENDING, indicates a node that is alive on and known to the system, but is not yet a member of any redundancy groups. The second state, CONTRIBUTING, denotes a live node that is member of one or more redundancy groups. The third state, EXPIRED, indicates that a node has failed or been decommissioned. EXPIRED nodes can be physically removed from the system; if other nodes are removed, the system handles the removal as a failure. Figure 4 illustrates these three states and the transitions between them.

A newly installed node is not immediately integrated into redundancy groups, but rather is placed into the PENDING state. This design provides a number of benefits. First, this allows the node to undergo a self-check and burn-in period in order to reduce the impact of infant mortality and batch correlated failures. Second, when it is time to expand the available storage in the system, Logan is able to make smarter management decisions by utilizing the devices in the PENDING pool, as compared to an approach that immediately integrates every node as soon as it arrives.

When Logan integrates a device into into one or more redundancy groups, that node enters the CONTRIBUTING state. In this mode, nodes store data, participate in redundancy group activities, and answer read and write requests. Eventually, as the node ages and its usefulness decreases relative to newer nodes, it will increasingly become a candidate for decommissioning.

C. Management Tasks

Management groups are tasked with a number of administrative duties. The first of these, *scale out*, deals with expanding the capacity of the system. It involves the creation of redundancy groups, and the assignment of segments to those groups. The second area, *recovery*, determines where data will be recovered to when a node is lost. The final area, *maintenance*, monitors the health of the system and actively identifies nodes that are ready to be decommissioned.

Each device in Logan maintains a list of named attributes that describe that device. In addition to querying the device for values, the system can also update the values. This can be used to reflect usage effects such as the accelerated wear caused by drive spin-ups, or the effects of batch correlated failures.

In order to make good management decisions, we are exploring the use of heuristic algorithms such as simulated annealing [13]. These algorithms attempt to solve an optimization problem by utilizing heuristics to repeatedly perform minor modifications to a partial solution. To this end, these algorithms utilize three main components. First, the solution space, X is the space of all possible solutions from which the answer will be drawn. Second, the neighbor function, N, heuristically chooses a new solution that is "close" to the current solution in the solution space. Finally, a objective function, P, measures the "goodness" of a solution, and is the value that the heuristic algorithm attempts to minimize or maximize.

Management group leaders collect the attribute lists from the members in their groups in order to develop a statistical understanding about the group's devices. This information is used during the administrative functions to identify expensive devices, in terms of utility versus resources consumed, without requiring administrator input. For example, this approach can identify a group's most power-hungry device. Further, this approach can determine how power-hungry that device is in comparison to the average of the group's devices.

1) Scale Out: For scale-out operations, each management group maintains a list of its redundancy groups and the devices assigned to those groups. This list is consulted and updated based on two redundancy group operations. First, Logan can form a new redundancy group. Second, Logan can expand an existing redundancy group. The latter strategy is possible because redundancy groups have a population range. Logan does not always fully populate new redundancy groups. Rather, it creates partially populated groups that still meet the system's reliability criteria, thus allowing the system to expand capacity, even when there are insufficient devices to create an entirely new redundancy group. For example, the system might require parity groups to be of the form $n+3$ disks, where $6 \leq n \leq 13$. This would mean that a redundancy group would have a minimum

of 9 disks and a maximum of 16 disks, and be able to grow from 9 to 16 gradually over time if needed.

At the device level, each management group maintains a list of of its devices and their unassigned, or free, segments. From this pool, Logan can assign device segments to redundancy groups from two primary sources. First, Logan can utilize previously unassigned segments from a device in the CONTRIBUTING state. Second, it can utilize segments from a PENDING device. Naturally, this would cause the device to transition to the CONTRIBUTING state.

Logan monitors the system, and performs a scale out operation when it detects that available free space in a management group has dropped below a predetermined low water mark. When this occurs, the management group performs a number of scale-out initialization steps. First, it determines which redundancy group are less than fully populated. Second, it determines if the existing groups offer sufficient scale-out room, or if new redundancy groups must be created.

2) Recovery: As with any storage system, and especially a long-term archival system, failure is inevitable. Additionally, since the system must be cost efficient, it is not enough to simply recover data to the first available free space. To address this problem, Logan uses similar heuristic search techniques to determine where data should be recovered to in the event of a device failure.

An instance of solution space is a mapping of segments to redundancy groups. At each iteration of the algorithm, some subset of free segments are mapped to the segments of the failed device. The primary constraint to enforce during recovery is that each member of a redundancy group is a different device.

3) Maintenance: The goal of maintenance is to determine if there is a management group configuration that can offer better service for the same or lower resource consumption.

As in previous management group operations, the state of the system consists of a mapping of device free segments to redundancy groups. However, in the case of maintenance, the redundancy group list consists of all the existing redundancy groups. At each iteration, devices that are likely to be decommissioned based on their expected lifetime or high energy costs per segment are randomly swapped with available segments. For this operation, a valid solution enforces the constraint that a device can only be decommissioned if all of its committed segments have a replacement, and that those replacements conform to the standard redundancy group constraints.

Unlike recovery and scale-out which are performed as soon as the heuristic completes, maintenance chores can handled opportunistically. A device that has been identified for decommissioning can wait until a scrubbing event or recovery event occurs in order to defray the power costs associated with a wholesale migration of a nodes complete contents. An important factor that enables this opportunistic approach is that the optimizations that maintenance seeks to achieve are not critical to data safety. When the unit being decommissioned activates, it can check to see if the units slated to takes its place in redundancy groups are still available. If they are not, the decommissioning can be cancelled, or new replacements can be chosen.

IV. FUTURE WORK

Current effort on Logan is focused on refining the skeleton, described in the previous section. Much of this work is directed towards exploring the use of heuristic algorithms in making sound management decisions. Additionally, we are exploring the behavior of the system to help determine the correct size of management groups; too large and the group leaders are overwhelmed, too small and the resulting splitting results in unnecessary management overhead. Finally, we are examining the boot-strapping problem; while the system is designed to scale up to hundreds of thousands of nodes, it must inevitably start with one.

Further along in our research plans, we plan on examining how best to deal with large-scale disasters and network partitions. In a long-term storage system, these sorts of events are inevitable, and must be survived gracefully, and with a minimum of needless energy expenditures. Many such events, such as a failed switch causing a network partition, are benign in the sense that data may still be safe, it is simply unreachable. However, the system's reaction in such a scenario could inadvertently cause more harm than good; the system may try and immediately rebuild all data that it could not contact.

As previously discussed, large archival systems are well suited to recovery procedures that allow the response to be scaled to the size of the problem. Currently, we utilize a two level scheme of intra-device and inter-device reliability. A third level, across geographically diverse sites, would be useful in order to protect data from natural disasters or other other "act of god" failures.

The dependency list of a given device describes the nodes that contribute to the reliability of a given node's data. Put another way, if a device fails, all of the device's in the failed device's adjacency list will need to contribute data during the recovery process. Thus, the size of the dependency list could have considerable

impact on data reliability, and during recovery, energy consumption. A large redundancy group allows greater parallelization during recovery, and implies greater diversity in the redundancy group's devices. In contrast a smaller adjacency list requires less devices to spin up during recovery. Considering these and other potential tradeoffs, an understanding of how adjacency affects reliability and power consumption could allow us to tailor our optimization methods to their ideal size.

Another intersection of reliability and power can be seen in a failed devices recovery schedule. That is, the amount and ordering of parallelization that occurs during rebuild. With a fuller understanding of power use during rebuild Logan could determine not only the placement of recovered data, but also the order that recovery should proceed. This area is complicated by the affect of very transient system states. For example, device population changes much slower than the list of currently spun up devices.

V. Conclusions

While archival systems are well served by a distributed architecture, such a design introduces the management challenges of heterogeneity in an evolving and aging system. Further, as part of a comprehensive cost strategy, such a system should continuously seek ways to maximize the utility it offers for the resources it is consuming.

To this end, we are developing Logan, a management layer that runs atop, a distributed network of energy-efficient, intelligent storage appliances [26]. Nodes are arranged in redundancy groups which allows data to be recovered from a lost node. To manage redundancy groups, and to facilitate system-wide communication, Logan arranges devices into management groups. Further, Logan collects information about the nodes in each management group and uses this data to make intelligent management decisions. Logan helps control archival storage costs by automating a number of common administrative tasks, and opportunistically decommissioning of old hardware.

Acknowledgments

References

[1] "Health Information Portability and Accountability Act," 104th Congress, Oct. 1996.

[2] I. Abraham and D. Dolev, "Asynchronous resource discovery," in *Proceedings of the Twenty-Second Annual Symposium on Principles of Distributed Computing (PODC 2003)*, Boston, MA, Jul. 2003, pp. 143–150.

[3] E. Anderson, M. Hobbs, K. Keeton, S. Spence, M. Uysal, and A. Veitch, "Hippodrome: running circles around storage administration," in *Proceedings of the 2002 Conference on File and Storage Technologies (FAST)*, Monterey, CA, Jan. 2002.

[4] L. N. Bairavasundaram, G. R. Goodson, S. Pasupathy, and J. Schindler, "An analysis of latent sector errors in disk drives," in *Proceedings of the 2007 SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, Jun. 2007.

[5] L. N. Bairavasundaram, G. R. Goodson, B. Schroeder, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "An analysis of data corruption in the storage stack," in *Proceedings of the 6th USENIX Conference on File and Storage Technologies (FAST)*, Feb. 2008, pp. 223–238.

[6] M. Baker, M. Shah, D. S. H. Rosenthal, M. Roussopoulos, P. Maniatis, T. Giuli, and P. Bungale, "A fresh look at the reliability of long-term digital storage," in *Proceedings of EuroSys 2006*, Apr. 2006, pp. 221–234.

[7] B. S. Chlebus and D. R. Kowalski, "Gossiping to reach consensus," in *Proceedings of the 14th ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, Winnipeg, Manitoba, Aug. 2002, pp. 220–229.

[8] D. Colarelli and D. Grunwald, "Massive arrays of idle disks for storage archives," in *Proceedings of the 2002 ACM/IEEE Conference on Supercomputing (SC '02)*, Nov. 2002.

[9] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google file system," in *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP '03)*. Bolton Landing, NY: ACM, Oct. 2003.

[10] Green Grid Consortium, "The green grid opportunity, decreasing datacenter and other IT energy usage patterns," http://www.thegreengrid.org, The Green Grid, Feb 2007.

[11] A. Guha, "Solving the energy crisis in the data center using COPAN Systems' enhanced MAID storage platform," Copan Systems white paper, Dec. 2006.

[12] M. Harchol-Balter, T. Leighton, and D. Lewin, "Resource discovery in distributed networks," in *Proceedings of the Eighteenth ACM Symposium on Principles of Distributed Computing (PODC 1999)*, Atlanta, GA, May 1999, pp. 229–237.

[13] S. Kirkpatrick, C.D. Gelatt, Jr., and M. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[14] S. Kutten, D. Peleg, and U. Vishkin, "Deterministic resource discovery in distributed networks," in *Proceedings of the 13th ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, Crete Island, Greece, Jul. 2001, pp. 77–83.

[15] W. Litwin, M.-A. Neimat, and D. A. Schneider, "LH*—a scalable, distributed data structure," *ACM Transactions on Database Systems*, vol. 21, no. 4, pp. 480–525, 1996.

[16] D. Narayanan, A. Donnelly, and A. Rowstron, "Write off-loading: Practical power management for enterprise storage," in *Proceedings of the 6th USENIX Conference on File and Storage Technologies (FAST)*, Feb. 2008, pp. 253–267.

[17] M. G. Oxley, "(H.R.3763) Sarbanes-Oxley Act of 2002," Feb. 2002.

[18] E. Pinheiro and R. Bianchini, "Energy conservation techniques for disk array-based servers," in *Proceedings of the 18th International Conference on Supercomputing*, Jun. 2004.

[19] E. Pinheiro, R. Bianchini, and C. Dubnicki, "Exploiting redundancy to conserve energy in storage systems," in *Proceedings of the 2006 SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, Saint Malo, France, Jun. 2006.

[20] S. Quinlan and S. Dorward, "Venti: A new approach to archival storage," in *Proceedings of the 2002 Conference on File and Storage Technologies (FAST)*. Monterey, California, USA: USENIX, 2002, pp. 89–101.

[21] R. V. Renesse, K. P. Birman, and W. Vogels, "Astrolabe: A robust and scalable technology for distributed system monitoring, mangement, and data mining," *ACM Transactions on Computer Systems*, vol. 21, no. 2, pp. 164–206, May 2003.

[22] A. Rowstron and P. Druschel, "Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility," in *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)*. Banff, Canada: ACM, Oct. 2001, pp. 188–201.

[23] Y. Saito, S. Frølund, A. Veitch, A. Merchant, and S. Spence, "FAB: Building distributed enterprise disk arrays from commodity components," in *Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2004, pp. 48–58.

[24] B. Schroeder and G. A. Gibson, "Disk failures in the real world: What does an MTTF of 1,000,000 hours mean to you?" in *Proceedings of the 5th USENIX Conference on File and Storage Technologies (FAST)*, Feb. 2007, pp. 1–16.

[25] M. Stonebraker and G. A. Schloss, "Distributed RAID—a new multiple copy algorithm," in *Proceedings of the 6th International Conference on Data Engineering (ICDE '90)*, Feb. 1990, pp. 430–437.

[26] M. W. Storer, K. M. Greenan, E. L. Miller, and K. Voruganti, "Pergamum: Replacing tape with energy efficient, reliable, disk-based archival storage," in *Proceedings of the 6th USENIX Conference on File and Storage Technologies (FAST)*, Feb. 2008.

[27] Sun Microsystems, "Sun StorageTek 5800 system architecture," White paper, Dec. 2007.

[28] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world' networks," *Nature*, vol. 393, no. 6684, pp. 440–442, Jun. 1998.

[29] C. Weddle, M. Oldham, J. Qian, A.-I. A. Wang, P. Reiher, and G. Kuenning, "PARAID : A gear-shifting power-aware RAID," in *Proceedings of the 5th USENIX Conference on File and Storage Technologies (FAST)*, Feb. 2007.

[30] W. W. Wilcke, R. B. Garner, C. Fleiner, R. F. Freitas, R. A. Golding, J. S. Glider, D. R. Kenchammana-Hosekote, J. L. Hafner, K. M. Mohiuddin, K. Rao, R. A. Becker-Szendy, T. M. Wong, O. A. Zaki, M. Hernandez, K. R. Fernandez, H. Huels, H. Lenk, K. Smolin, M. Ries, C. Goettert, T. Picunko, B. J. Rubin, H. Kahn, and T. Loo, "IBM Intelligent Bricks project—petabytes and beyond," *IBM Journal of Research and Development*, vol. 50, no. 2/3, pp. 181–197, 2006.

[31] P. Yalagandula and M. Dahlin, "A scalable distributed information management system," in *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM '04)*. Portland, OR: ACM Press, Aug. 2004, pp. 379–390.

[32] P. Yalagundula and M. Dahlin, "Shruti: A self-tuning hierarchical aggregation system," in *Proceedings of the First International Conferece on Self-Adaptive and Self-Organizing Systems (SASO 2007)*, Boston, MA, Jul. 2007, pp. 141–150.

[33] X. Yao and J. Wang, "RIMAC: a novel redundancy-based hierarchical cache architecture for energy efficient, high performance storage systems," in *Proceedings of EuroSys 2006*, Oct. 2006, pp. 249–262.

[34] L. L. You, K. T. Pollack, and D. D. E. Long, "Deep Store: An archival storage system architecture," in *Proceedings of the 21st International Conference on Data Engineering (ICDE '05)*. Tokyo, Japan: IEEE, Apr. 2005.

[35] Q. Zhu, Z. Chen, L. Tan, Y. Zhou, K. Keeton, and J. Wilkes, "Hibernator: Helping disk arrays sleep through the winter," in *Proceedings of the 20th ACM Symposium on Operating Systems Principles (SOSP '05)*. Brighton, UK: ACM, Oct. 2005.