Appears in *SIGOPS Operating Systems Review*, Vol. 43, No. 4, December 2009, pp. 92-105

The Case for RAMClouds: Scalable High-Performance Storage Entirely in DRAM

John Ousterhout, Parag Agrawal, David Erickson, Christos Kozyrakis, Jacob Leverich, David Mazières, Subhasish Mitra, Aravind Narayanan, Guru Parulkar, Mendel Rosenblum, Stephen M. Rumble, Eric Stratmann, and Ryan Stutsman

Department of Computer Science
Stanford University

Abstract

Disk-oriented approaches to online storage are becoming increasingly problematic: they do not scale gracefully to meet the needs of large-scale Web applications, and improvements in disk capacity have far outstripped improvements in access latency and bandwidth. This paper argues for a new approach to datacenter storage called RAMCloud, where information is kept entirely in DRAM and large-scale systems are created by aggregating the main memories of thousands of commodity servers. We believe that RAMClouds can provide durable and available storage with 100-1000x the throughput of disk-based systems and 100-1000x lower access latency. The combination of low latency and large scale will enable a new breed of data-intensive applications.

1 Introduction

For four decades magnetic disks have provided the primary means of storing online information in computer systems. Over that period disk technology has undergone dramatic improvements, and it has been harnessed by a variety of higher-level storage systems such as file systems and relational databases. However, the performance of disk has not improved as rapidly as its capacity, and developers are finding it increasingly difficult to scale disk-based systems to meet the needs of large-scale Web applications. Many people have proposed new approaches to disk-based storage as a solution to this problem; others have suggested replacing disks with flash memory devices. In contrast, we believe that the solution is to shift the primary locus of online data from disk to random access memory, with disk relegated to a backup/archival role.

In this paper we argue that a new class of storage called RAMCloud will provide the storage substrate for many future applications. A RAMCloud stores all of its information in the main memories of commodity servers, using hundreds or thousands of such servers to create a large-scale storage system. Because all data is in DRAM at all times, a RAMCloud can provide 100-1000x lower latency than disk-based systems and 100-1000x greater throughput. Although the individual memories are volatile, a RAMCloud can use replication and backup techniques to provide data durability and availability equivalent to disk-based systems.

We believe that RAMClouds will fundamentally change the storage landscape in three ways. First, they will simplify the development of large-scale Web applications by eliminating many of the scalability issues that sap developer productivity today. Second, their extremely low latency will enable richer query models that enable a new class of data-intensive applications. Third, RAMClouds will provide the scalable storage substrate needed for "cloud computing" and other datacenter applications [3]: a RAMCloud can support a single large application or numerous smaller applications, and allow small applications to grow rapidly into large ones without additional complexity for the developer.

The rest of this paper is divided into five parts. Section 2 describes the RAMCloud concept and the scale and performance that we believe are achievable in a RAMCloud system. Section 3 offers two motivations for RAMClouds, one from the standpoint of applications and one from the standpoint of the underlying storage technologies; it also discusses the benefits of extremely low latency and compares RAMClouds with two alternative approaches (caching and flash memory). In order to build a practical RAMCloud numerous research issues will need to be addressed; Section 4 introduces a few of these issues. Section 5 discusses the disadvantages of RAMClouds, such as high cost/bit and high energy usage. Finally, Section 6 summarizes related work.

2 RAMCloud Overview

RAMClouds are most likely to be used in datacenters containing large numbers of servers divided roughly into two categories: application servers, which implement application logic such as generating Web pages or enforcing business rules, and storage servers, which

provide longer-term shared storage for the application servers. Traditionally the storage has consisted of files or relational databases, but in recent years a variety of new storage mechanisms have been developed to improve scalability, such as Bigtable [4] and memcached [16]. Each datacenter typically supports numerous applications, ranging from small ones using only a fraction of an application server to large-scale applications with thousands of dedicated application and storage servers.

RAMCloud represents a new way of organizing storage servers in such a system. There are two key attributes that differentiate a RAMCloud from other storage systems. First, all information is kept in DRAM at all times. A RAMCloud is not a cache like memcached [16] and data is not stored on an I/O device, as with flash memory: DRAM is the permanent home for data. Disk is used only for backup. Second, a RAMCloud must scale automatically to support thousands of storage servers; applications see a single storage system, independent of the actual number of storage servers.

Information stored in a RAMCloud must be as durable as if it were stored on disk. For example, failure of a single storage server must not result in data loss or more than a few seconds of unavailability. Section 4.2 discusses techniques for achieving this level of durability and availability.

Keeping all data in DRAM will allow RAMClouds to achieve performance levels 100-1000x better than current disk-based storage systems:

- It should be possible to achieve access latencies of 5-10 microseconds, measured end-to-end for a process running in an application server to read a few hundred bytes of data from a single record in a single storage server in the same datacenter. In comparison, disk-based systems offer access times over the network ranging from 5-10ms (if disk I/O is required) down to several hundred microseconds (for data cached in memory).
- A single multi-core storage server should be able to service at least 1,000,000 small requests per second. In comparison, a disk-based system running on a comparable machine with a few disks can service 1000-10000 requests per second, depending on cache hit rates.

These goals represent what we believe is possible, but achieving them is by no means guaranteed; Section 4 discusses some of the obstacles that will have to be overcome.

Table 1 summarizes a RAMCloud configuration that is feasible today. This configuration assumes 64GB of DRAM on each server, which is the largest amount that is cost-effective today (memory prices rise dramatically for larger memory sizes). With 1000 servers the configuration offers 64TB of storage at $60/GB. With additional servers it should be possible to build RAMClouds with capacities as large as 500TB today. Within 5-10 years, assuming continued improvements in DRAM technology, it will be possible to build RAMClouds with capacities of 1-10 Petabytes at a cost less than $5/GB.

3 Motivation

3.1 Application scalability

The motivation for RAMClouds comes from two sources: applications and technology. From the standpoint of applications, relational databases have been the storage system of choice for several decades but they do not scale to the level required by today's large-scale applications. Virtually every popular Web application has found that a single relational database cannot meet its throughput requirements. As the site grows it must undergo a series of massive revisions, each one introducing *ad hoc* techniques to scale its storage system, such as partitioning data among multiple databases. These techniques work for a while, but scalability issues return when the site reaches a new level of scale or a new feature is introduced, requiring yet more special-purpose techniques.

For example, as of August 2009 the storage system for Facebook includes 4000 MySQL servers. Distribution of data across the instances and consistency between the instances are handled explicitly by Facebook application code [13]. Even so, the database servers are incapable of meeting Facebook's throughput requirements by themselves, so Facebook also employs 2000 memcached servers, which cache recently used query results in key-value stores kept in main memory. Unfortunately, consistency between the memcached and MySQL servers must be managed by application soft-

# servers	1000
Capacity/server	64 GB
Total capacity	64 TB
Total server cost	$4M
Cost/GB	$60
Total throughput	10^9 ops/sec

Table 1. An example RAMCloud configuration using currently available commodity server technology. Total server cost is based on list prices and does not include networking infrastructure or racks.

ware (e.g., cached values must be flushed explicitly when the database is updated), which adds to application complexity.

Numerous new storage systems have appeared in recent years to address the scalability problems with relational databases; examples include Bigtable [4], Dynamo [8], and PNUTS [6] (see [23] for additional examples). Unfortunately, each of these is specialized in some way, giving up some of the benefits of a traditional database in return for higher performance in certain domains. Furthermore, most of the alternatives are still limited in some way by disk performance. One of the motivations for RAMCloud is to provide a general-purpose storage system that scales far beyond existing systems, so application developers do not have to resort to *ad hoc* techniques.

3.2 Technology trends

The second motivation for RAMClouds comes from disk technology evolution (see Table 2). Disk capacity has increased more than 10000-fold over the last 25 years and seems likely to continue increasing in the future. Unfortunately, though, the access rate to information on disk has improved much more slowly: the transfer rate for large blocks has improved "only" 50-fold, and seek time and rotational latency have only improved by a factor of two.

As a result of this uneven evolution the role of disks must inevitably become more archival: it simply isn't possible to access information on disk very frequently. Table 2 illustrates this in two ways. First, it computes the capacity/bandwidth ratio: if the disk is filled with blocks of a particular size, how often can each block be accessed, assuming random accesses? In the mid-1980s 1-Kbyte records could be accessed on average about every 10 minutes; with today's disks each record can only be accessed about 6 times per year on average, and this rate will drop with each future improvement in disk capacity. Larger blocks allow more frequent accesses, but even in the best case data on disk can only be accessed $1/300^{th}$ as frequently as 25 years ago.

If data is accessed in large blocks the situation is not quite as dire: today's disks can support about one access every 1.5 hours to each block. However, the definition of "large block" is changing. In the mid-1980s blocks of 400 Kbytes achieved 90% of the maximum transfer rate; today, blocks must contain at least 10 Mbytes to achieve 90% of the maximum transfer rate, and large blocks will need to be even larger in the future. Video data meets this threshold, as do data for bulk-processing applications such as MapReduce [7], but most forms of on-line data do not; even photos and songs are too small to use the full bandwidth of today's disks.

A second way of understanding disk trends is to apply Jim Gray's Rule [12]: if portions of the disk are left unused then the remaining space can be accessed more frequently. As the desired access rate to each record increases, the disk utilization must decrease, which increases the cost per usable bit; eventually a crossover point is reached where the cost/bit of disk is no better than DRAM. The crossover point has increased by a factor of 360x over the last 25 years, meaning that data on disk must be used less and less frequently.

	Mid-1980s	**2009**	**Improvement**
Disk capacity	30 MB	500 GB	16667x
Maximum transfer rate	2 MB/s	100 MB/s	50x
Latency (seek + rotate)	20 ms	10 ms	2x
Capacity/bandwidth (large blocks)	15 s	5000 s	333x *worse*
Capacity/bandwidth (1KB blocks)	600 s	58 days	8333x *worse*
Jim Gray's Rule [11] (1KB blocks)	5 min.	30 hours	360x *worse*

Table 2. A comparison of disk technology today versus 25 years ago, based on typical personal computer disks. Capacity/bandwidth measures how long it takes to read the entire disk, assuming accesses in random order to blocks of a particular size; this metric also indicates how frequently each block can be accessed on average (assuming the disk is full). For large blocks (>10 Mbytes today) capacity/bandwidth is limited by the disk transfer rate; for small blocks it is limited by latency. The last line assumes that disk utilization is reduced to allow more frequent accesses to a smaller number of records; it uses the approach of Gray and Putzolu [11] to calculate the access rate at which memory becomes cheaper than disk. For example, with today's technologies, if a 1KB record is accessed at least once every 30 hours, it is not only faster to store it in memory than on disk, but also cheaper (to enable this access rate only 2% of the disk space can be utilized).

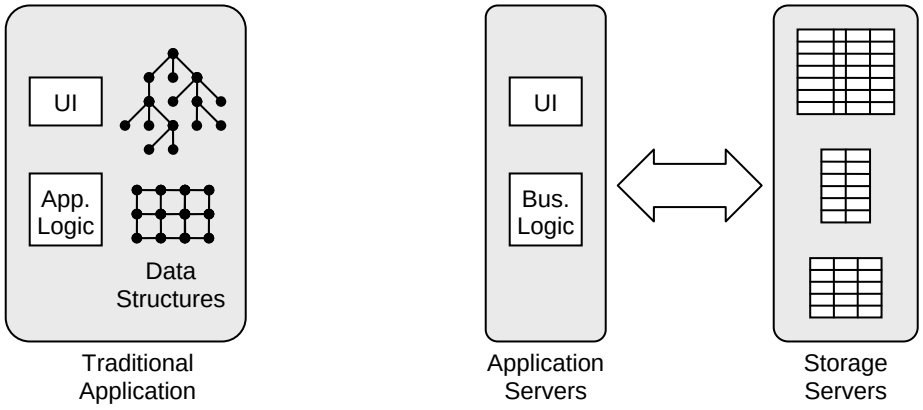

Figure 1. In a traditional application (a) the application's data structures reside in memory on the same machine containing the application logic and user interface code. In a scalable Web application (b) the data is stored on separate servers from the application's user interface and business logic. Because of the high latency of data access, it is expensive for a Web application to perform iterative explorations of the data; typically the application server must fetch all of the data needed for a Web page in a small number of bulk requests. Lower latency will enable more complex data explorations.

3.3 Caching

Historically, caching has been viewed as the answer to problems with disk latency: if most accesses are made to a small subset of the disk blocks, high performance can be achieved by keeping the most frequently accessed blocks in DRAM. In the ideal case a system with caching can offer DRAM-like performance with disk-like cost.

However, the trends in Table 2 are diluting the benefits of caching by requiring a larger and larger fraction of data to be kept in DRAM. Furthermore, some new Web applications such as Facebook appear to have little or no locality, due to complex linkages between data (e.g., friendships in Facebook). As of August 2009 about 25% of all the online data for Facebook is kept in main memory on memcached servers at any given point in time, providing a hit rate of 96.5%. When additional caches on the database servers are counted, the total amount of memory used by the storage system equals approximately 75% of the total size of the data (excluding images). Thus a RAMCloud would only increase memory usage for Facebook by about one third.

In addition, the 1000x gap in access time between DRAM and disk means that a cache must have exceptionally high hit rates to avoid significant performance penalties: even a 1% miss ratio for a DRAM cache costs a factor of 10x in performance. A caching approach makes the deceptive suggestion that "a few cache misses are OK" and lures programmers into configurations where system performance is poor.

For these reasons we believe that caches in the future will have to be so large that they will provide little cost benefit while still introducing significant performance risk. RAMClouds may cost slightly more than caching systems, but they will provide guaranteed performance independent of access patterns or locality.

3.4 Does latency matter?

The most unique aspect of RAMClouds relative to other storage systems is the potential for extraordinarily low latency. On the one hand, 5-10 µs latency may seem unnecessary, and it is hard to point to Web applications that require this level of latency today. However, we believe that low latency is one of the most important features of RAMClouds.

There may not be many Web applications that require 5-10 µs latency today, but this is because there are no storage systems that can support such a requirement. In contrast, traditional applications expect and get latency significantly less than 5-10 µs (see Figure 1). In these applications the data for the application resides in main-memory data structures co-located with the application's front end. Such data can be explored with very low latency by the application. In Web applications the data is stored on separate storage servers. Because of high data latency, Web applications typically cannot afford to make complex unpredictable explorations of their data, and this constrains the functionality they can provide. If Web applications are to replace traditional applications, as has been widely predicted, then they will need access to data with latency much closer to what traditional applications enjoy.

Today's Web applications are already using data in complex ways and struggling with latency issues. For example, when Facebook receives an HTTP request for a Web page, the application server makes an average of 130 internal requests (inside the Facebook site) as part of generating the HTML for the page [13]. These requests must be issued sequentially, since later requests depend on results produced by earlier requests. The cumulative latency of the internal requests is one of the limiting factors in overall response time to users, so considerable developer effort is expended to minimize the number and size of requests. Amazon has reported similar results, with 100-200 requests to generate HTML for each page [8].

High latency data access has also caused problems for database applications. Queries that do not match the layout of data on disk, and hence require numerous seeks, can be intolerably slow. Iterative searches such as tree walks, where the *n*th data item cannot be identified until the *n-1*th item has been examined, result in too many high latency queries to be practical. In recent years numerous specialized database architectures have appeared, such as column stores, stream processing engines, and array stores. Each of these reorganizes data on disk in order to reduce latency for a particular style of query; each produces order-of-magnitude speedups for a particular application area, but none is general-purpose. Stonebraker et al. have predicted the end of the "one size fits all" database because there is no disk layout that is efficient for all applications [23]. However, with sufficiently low latency none of these specialized approaches are needed. RAMClouds offer the hope of a new "one size fits all" where performance is independent of data placement and a rich variety of queries becomes efficient.

By providing random access with very low latency to very large datasets, RAMClouds will not only simplify the development of existing applications, but they will also enable new applications that access large amounts of data more intensively than has ever been possible. One example is massive multiplayer games involving interactions between thousands of players. More generally, RAMClouds will be well-suited to algorithms that must traverse large irregular graph structures, where the access patterns are so unpredictable that the full latency of the storage system must be incurred for each item retrieved. RAMClouds will allow a single sequential agent to operate much more quickly on such a structure, and they will also permit thousands of agents to operate concurrently on shared graphs.

Low latency also offers other benefits, such as enabling a higher level of consistency; we will describe these benefits when discussing RAMCloud implementation issues in Section 4.

3.5 Use flash memory instead of DRAM?

Flash memory offers another alternative with lower latency than disk. Today it is most commonly used in devices such as cameras and media players, but it is receiving increasing attention for general-purpose online storage [1]. A RAMCloud could be constructed with flash memory as the primary storage technology instead of DRAM ("FlashCloud"), and this would be cheaper and consume less energy than a DRAM-based approach. Nonetheless, we believe a DRAM-based implementation is more attractive because it offers higher performance.

The primary advantage of DRAM over flash memory is latency. Flash devices have read latencies as low as 20-50 µs, but they are typically packaged as I/O devices, which adds additional latency for device drivers and interrupt handlers. Write latencies for flash devices are 200 µs or more. Overall, a RAMCloud is likely to have latency 5-10x lower than a FlashCloud, which will increase the benefits discussed in Section 3.4. Also, from a research standpoint RAMCloud encourages a more aggressive attack on latency in the rest of the system; with FlashCloud the device latency will dominate, removing the incentive to improve latency of other system components.

RAMClouds also provide higher throughput than FlashClouds, which can make them attractive even in situations where latency isn't important. Figure 2, reproduced from Andersen et al. [1], generalizes Jim Gray's Rule. It indicates which of disk, flash, and DRAM is cheapest for a given system, given its re-

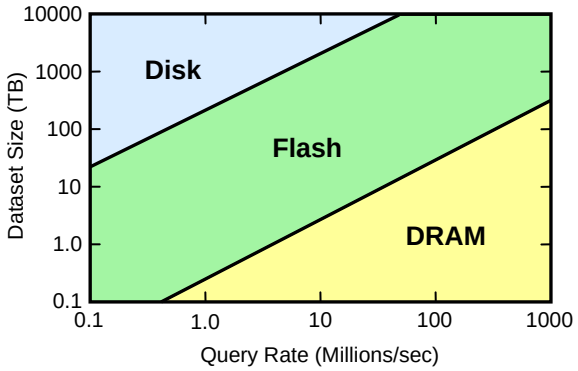

Figure 2. This figure (reproduced from Andersen et al. [1]) indicates which storage technology has the lowest total cost of ownership over 3 years, including server cost and energy usage, given the required dataset size and query rate for an application (assumes random access workloads).

quirements in terms of dataset size and operations/sec. For high query rates and smaller dataset sizes DRAM is cheapest; for low query rates and large datasets disk is cheapest; and flash is cheapest in the middle ground.

Interestingly, the dividing lines in Figure 2 are all shifting upwards with time, which will increase the coverage of RAMClouds in the future. To understand this effect, consider the dividing line between flash and DRAM. At this boundary the cost of flash is limited by cost/query/sec, while the cost of DRAM is limited by cost/bit. Thus the boundary moves upwards as the cost/bit of DRAM improves; it moves to the right as the cost/query/sec. of flash improves. For all three storage technologies cost/bit is improving much more rapidly than cost/query/sec, so all of the boundaries are moving upwards.

In the future it is possible that the latency of flash memory may improve to match DRAM; in addition, there are several other emerging memory technologies such as phase-change memory that may ultimately prove to be better than DRAM for storage. Even if this happens, many of the implementation techniques developed for RAMClouds will carry over to other random access memory technologies. If RAMClouds can be implemented successfully they will take memory volatility out of the equation, allowing storage technology to be selected based on performance, cost, and energy consumption.

3.6 RAMCloud applicability today

RAMClouds are already practical for a variety of applications today:

- As of August 2009 all of the non-image data for Facebook occupies about 260TB [13], which is probably near the upper limit of practicality for RAMCloud today.
- Table 3 estimates that customer data for a large-scale online retailer or airline would fit comfortably in a small fraction of the RAMCloud configuration in Table 1. These applications are unlikely to be the first to use RAMClouds, but they illustrate that many applications have historically been considered "large" can be accommodated easily by a RAMCloud.
- The cost of DRAM today is roughly the same as the cost of disk 10 years ago ($10-30/Gbyte), so any data that could be stored cost-effectively on disk then can be stored cost-effectively in DRAM today.

It is probably not yet practical to use RAMClouds for large-scale storage of media such as videos, photos, and songs (and these objects make better use of disks because of their large size). However, RAMClouds are practical for almost all other online data today, and future improvements in DRAM technology will probably make RAMClouds attractive for media within a few years.

4 Research Issues

There are numerous challenging issues that must be addressed before a practical RAMCloud system can be constructed. This section describes several of those issues, along with some possible solutions. Some of these issues, such as data model and concurrency control, are common to all distributed storage systems, so there may be solutions devised by other projects that will also work for RAMClouds; others, such as low latency RPC and data durability, may require a unique approach for RAMClouds.

4.1 Low latency RPC

Although latencies less than 10µs have been achieved in specialized networks such as Infiniband and Myrinet, most existing datacenters use networking infrastructure based on Ethernet/IP/TCP, with typical round-trip times for remote procedure calls of 300-500 µs. We believe it is possible to reduce this to 5-10 µs, but doing so will require innovations at several levels. The greatest obstacle today is latency in the network switches. A large datacenter is likely to have a three-tier switching structure, so each packet traverses five switches as it moves up through the switching hierarchy and back down to its destination. Typical switches today introduce delays

Online Retailer		**Airline Reservations**	
Revenues/year:	$16B	Flights/day:	4000
Average order size	$40	Passengers/flight:	150
Orders/year	400M	Passenger-flights/year:	220M
Data/order	1000 - 10000 bytes	Data/passenger-flight:	1000 - 10000 bytes
Order data/year:	400GB - 4.0TB	Passenger data/year:	220GB - 2.2 TB
RAMCloud cost:	$24K-240K	RAMCloud cost:	$13K-130K

Table 3. Estimates of the total storage capacity needed for one year's customer data of a hypothetical online retailer and a hypothetical airline. In each case the total requirements are no more than a few terabytes, which would fit in a modest-sized RAMCloud. The last line estimates the purchase cost for RAMCloud servers, using the data from Table 1.

of 10's of microseconds each, for a total switching latency of 100 µs or more in each direction. Newer 10GB switches such as the Arista 7100S [13] use cut-through routing and claim latencies of less than 1 µs, but this will still produce a total network delay of nearly 5 µs in each direction. Additional improvements in switching latency will be required to achieve RPC times less than 10 µs.

In order to achieve 5-10 µs latency and 1 million requests/second/server, RAMCloud servers will need to process each request in 1 µs or less; this will require significant reductions in software overheads. General-purpose operating systems introduce high overheads for interrupt processing, network protocol stacks, and context switching. In RAMCloud servers it may make sense to use a special-purpose software architecture where one core is dedicated to polling the network interface(s) in order to eliminate interrupts and context switches. The network core can perform basic packet processing and then hand off requests to other cores that carry out the requests. RAMCloud servers will make good use of multi-core architectures, since different requests can be handled in parallel on different cores.

On application server machines virtualization is becoming more and more popular as a mechanism for managing applications. As a result, incoming packets must first pass through a virtual machine monitor and then a guest operating system before reaching the application. Achieving low latency will require these overheads to be reduced, perhaps by passing packets directly from the virtual machine monitor to the application. Another approach is to use network interfaces that can be mapped directly into an application's address space so that applications can send and receive packets without the involvement of either the operating system or the virtual machine monitor.

In order to achieve the most efficient network communication, it may be necessary to modify the TCP protocol or use a different reliable-delivery protocol based on UDP:

- TCP retransmission timeouts are typically hundreds of milliseconds or more; this is necessary to manage congestion in long-haul networks, but it interferes with efficient operation inside a datacenter: even a low rate of packet loss will significantly degrade average latency.
- The flow-oriented nature of TCP offers little advantage for RAMClouds, since individual requests will be relatively small. Flows introduce additional state and complexity in packet processing without providing much benefit in an environment like RAMCloud.
- A custom protocol can use an acknowledgment scheme optimized for the request-response nature of RAMCloud communication, resulting in fewer acknowledgment packets than TCP.

Although 5-10 µs RPC latency seems like the best that can be achieved in the near future, lower latency would be even better. It is an open question whether further improvements are possible; for example, is it possible eventually to achieve RPC latency within a datacenter of 1 µs? At this level, even speed-of-light delays will be significant.

In addition to low latency, RAMClouds will also require high bandwidth from the networking infrastructure. Most current datacenter networks are oversubscribed by 10x or more at the upper levels, meaning that the system cannot support random communication patterns at the full bandwidth of the lowest-level links. It is not clear that there will be much locality in the communication patterns for RAMCloud servers, so bisection bandwidth may need to be increased in the upper levels of datacenter networks.

4.2 Durability and availability

For RAMClouds to be widely used they must offer a high level of durability and availability (at least as good as today's disk-based systems). At a minimum this means that a crash of a single server must not cause data to be lost or impact system availability for more than a few seconds. RAMClouds must also offer reasonable protection against power outages, and they may need to include cross-datacenter replication. We assume that any guarantees about data durability must take effect at the time a storage server responds to a write request.

One approach to durability is to replicate each object in the memories of several server machines and update all of the replicas before responding to write requests. At least 3 copies of each object will probably be needed to assure an adequate level of durability and availability, so this approach would triple the cost of memory, which is the dominant factor in overall system cost. It would also increase energy usage significantly. The cost of main-memory replication can be reduced by using coding techniques such as parity striping [17], but this makes crash recovery considerably more expensive. In any case, replication in memory would depend on a reliable power source, so power outages do not take down all of the replicas simultaneously. For these reasons, we believe RAMClouds will probably use a technology other than DRAM for backup copies of data.

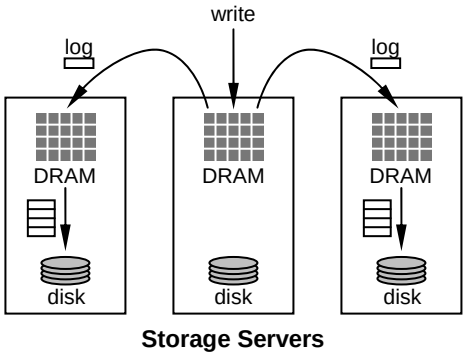

Figure 3. In the buffered logging approach to durability, a single copy of each object is stored in DRAM. When an object is modified, the changes are logged to 2 or more other servers. Initially the log entries are stored in the DRAM of the backup servers; they are transferred to disk asynchronously in batches in order to utilize the full disk bandwidth.

An alternative approach to durability is to keep a single copy of each object in DRAM, but have each server back up new data on its local disk as part of every write operation. However, this approach is undesirable because it ties write latency to disk latency and still leaves data unavailable if the server crashes. At the very least, data must be replicated on multiple server machines.

Figure 3 outlines an alternative called *buffered logging* that uses both disk and memory for backup. In this approach a single copy of each object is stored in DRAM of a primary server and copies are kept on the disks of two or more backup servers; each server acts as both primary and backup. However, the disk copies are not updated synchronously during write operations. Instead, the primary server updates its DRAM and forwards log entries to the backup servers, where they are stored temporarily in DRAM. The write operation returns as soon as the log entries have been written to DRAM in the backups. Each backup server collects log entries into batches that can be written efficiently to a log on disk. Once log entries have been written to disk they can be removed from the backup's DRAM.

If a server crashes in the buffered logging approach, the data in its DRAM can be reconstructed on any of the backup servers by processing the disk logs. However, two optimizations will be required in order to recover quickly enough to avoid service disruption. First, the disk logs must be truncated to reduce the amount of data that must be read during recovery. One approach is to create occasional checkpoints, after which the log can be cleared. Another approach is to use log cleaning techniques like those developed for log-structured file systems [21], where portions of the log with stale information are occasionally rewritten to squeeze out the stale data and reduce the log size.

Even with log truncation, the total amount of data to read will be at least as large as the size of the crashed server's DRAM; at current disk transfer speeds it will take ten minutes or more to read this much data from a single disk. The second optimization is to divide the DRAM of each primary server into hundreds of shards, with each shard assigned to different backup servers. After a crash, one backup server for each shard reads its (smaller) log in parallel; each backup server acts as a temporary primary for its shard until a full copy of the lost server's DRAM can be reconstructed elsewhere in the RAMCloud. With this approach it should be possible to resume operation (using the backup servers) within a few seconds of a crash.

Buffered logging allows both reads and writes to proceed at DRAM speeds while still providing durability and availability. However, the total system throughput for writes will be limited by the disk bandwidth available for writing log entries. Assuming typical updates of a few hundred bytes, a log-cleaning approach for truncation, and a single disk per server with 100 MB/sec throughput, we estimate that each server can support around 50,000 updates/second (vs. 1M reads/sec). It may make sense to use flash memory instead of disk for backup copies, if flash memory can be packaged in a way that supports higher write throughput than disk. For the highest update rates the only solution is to keep replicas in DRAM.

Power failures can be handled in one of three ways in a RAMCloud:

- Servers continue operating long enough after receiving warning of an impending power outage to flush data to disk. With buffered logging a few seconds is sufficient to flush buffered log entries. With replication in DRAM at least 10 minutes will be required.
- Applications tolerate the loss of unwritten log data in an unexpected power failure (this option exists only for the buffered logging approach).
- All data must be committed to stable storage as part of each write operation; this will degrade write latency and system throughput.

As previously mentioned, some applications may require data to be replicated in multiple datacenters. A multi-datacenter approach offers two advantages: it can continue operation even if a single datacenter becomes completely unavailable, and it can also reduce the latency of communication with geographically distributed users as well as long-haul network bandwidth. A RAMCloud can still provide high performance for reads in this environment, but write performance will

be significantly degraded. The bandwidth between datacenters will limit aggregate write throughput, and applications will have to accept higher latency for write operations (10's or 100's of milliseconds) in order to update remote datacenters synchronously. An alternative is to allow write operations to return before remote datacenters have been updated (meaning some "committed" data could be lost if the originating datacenter crashes). In this case, if the links between datacenters have enough bandwidth, then it may be possible to achieve the same write performance in a multi-datacenter RAMCloud as a single-datacenter implementation.

If a RAMCloud system is to ensure data integrity, it must be able to detect all significant forms of corruption; the use of DRAM for storage complicates this in several ways. First, bit error rates for DRAM are relatively high [22] so ECC memory will be necessary to avoid undetected errors. Even so, there are several other ways that DRAM can become corrupted, such as errors in peripheral logic, software bugs that make stray writes to memory, and software or hardware errors related to DMA devices. Without special attention, such corruptions will not be detected. Thus RAMClouds will probably need to augment stored objects with checksums that can be verified when objects are read. It may also make sense to scan stored data periodically to detect latent errors.

4.3 Data model

Another issue for RAMClouds is the data model for the system. There are many possible choices, and existing systems have explored almost every imaginable combination of features. The data model includes three overall aspects. First, it defines the nature of the basic objects stored in the system. In some systems the basic objects are just variable-length "blobs" of bytes where the storage system knows nothing about their internal structure. At the other extreme, the basic objects can have specific structure enforced by the system; for example, a relational database requires each record to consist of a fixed number of fields with prespecified types such as integer, string, or date. In either case we expect most objects to be small (perhaps a few hundred bytes), containing information equivalent to a record in a database or an object in a programming language such as C++ or Java.

The second aspect of a data model is its support for aggregation: how are basic objects organized into higher-level structures? In the simplest case, such as key-value stores, the storage system provides no aggregation: the only operations are reads and writes of basic objects. Most systems provide some sort of aggregation; for example, in a relational database the rows are organized into tables and tables can be augmented with indices to accelerate queries.

The third aspect of a data model is its mechanisms for naming and indexing: when retrieving or modifying basic objects, how are the objects named? In the simplest case, such as a key-value store, each object has a single unique identifier that must be used in all references to the object. At the other extreme, relational databases use content-based addressing: rows are selected by specifying the values of one or more fields in the row. Any field or combination of fields can serve as a "name" for the row when looking it up, and a powerful query language can be used to combine the rows from different tables in a single query.

The highly-structured relational data model has been the dominant one for storage systems over the last four decades, and it provides extraordinary power and convenience. However, it does not appear practical to scale the relational structures over large numbers of servers; we know of no RDBMS that includes thousands of server machines (or even hundreds) in a single unified system. In order to achieve scalability, some of the relational features, such as large-scale joins and complex transactions, will probably have to be sacrificed (most large-scale Web applications have already made these sacrifices).

At the opposite extreme, unstructured models such as key-value stores appear highly scalable, but they may not be rich enough to support a variety of applications conveniently. For example, most applications need tables or some other form of aggregation, plus indexing; these operations may be difficult or expensive for an application to implement without support from the underlying storage system.

One approach that seems promising to us is an intermediate one where servers do not impose structure on data but do support aggregation and indexing. In this approach the basic objects are blobs of bytes. The storage servers never interpret the contents of the objects, so different applications can structure their objects differently; some applications may use relational-style structures, while others may use more loosely-structured data such as JSON or XML. The servers do provide a table mechanism for grouping objects; among other things, tables make it easier to share a RAMCloud among multiple applications (access controls can be specified for tables, and administrative operations such as deleting all of the data for a defunct application become simpler).

In this approach objects can be named either using unique identifiers or using indices. The servers allow any number of indices to be associated with each table. Clients must manage indices with explicit requests ("associate key *k* with object *o* in index *i*", or "delete the association between key *k* and object *o* in index *i*"), but can piggyback index updates with object updates; the servers will guarantee that either all or none of the updates complete. This allows well-formed applications to maintain consistency between tables and their indices without requiring servers to understand the structure of objects. The complexity of managing indices can be hidden in client-side libraries (see Section 4.7).

4.4 Distribution and scaling

A RAMCloud must provide the appearance of a single unified storage system even though it is actually implemented with thousands of server machines. The distribution of the system should not be reflected in the APIs it provides to application developers, and it should be possible to reconfigure the system (e.g. by adding or removing servers) without impacting running applications or involving developers. This represents one of the most important advantages of RAMCloud over most existing storage systems, where distribution and scaling must be managed explicitly by developers.

The primary issue in the distribution and scaling of the system is data placement. When a new data object is created it must be assigned to one or more servers for storage. Later on, the object may need to be moved between servers in order to optimize the performance of the system or to accommodate the entry and exit of servers. Given the scale and complexity of a RAMCloud system, the decisions about data placement and movement will need to be made automatically, without human involvement.

For example, assume that a RAMCloud provides a table mechanism for grouping related objects. For small tables it is most efficient to store the entire table on a single server. This allows multiple objects to be retrieved from the table with a single request to a single server, whereas distributing the table may require multiple requests to different servers. The single-server approach also minimizes the amount of configuration information that must be managed for the table. If the system supports indices, it will be most efficient to keep each index on the same server as the table for the index, so that an indexed lookup can be performed with a single RPC.

However, some tables will eventually become too large or too hot for a single server, in which case they will need to be partitioned among multiple servers. It may make sense to scatter the table's data as randomly as possible in order to balance load among the servers, or it may be preferable to partition the table based on an index, so that adjacent objects in the index are co-located on the same server and can be manipulated with a single RPC. It may also make sense to partition indices along with their tables, so that index entries are located on the same servers as the corresponding table objects.

Because of the high throughput of its servers, a RAMCloud is unlikely to need data replication for performance reasons; replication will be needed only for data durability and availability. If a server becomes overloaded then some of its tables can be moved to other servers; if a single table overloads the server then the table can be partitioned to spread the load. Partitioning is already needed to handle large tables, so it makes sense to use this mechanism for load balancing as well. Replication for performance will only be needed if the access rate to a single object exceeds the 1M ops/second throughput of a single server; this situation may be so rare that it can be handled with special techniques in the affected applications.

Given the size of a RAMCloud system, its configuration is likely to change frequently, both because of changes in the underlying hardware (server crashes, new server additions, etc.) and because of changes in the applications. Thus it will be important for RAMClouds to move data between servers efficiently and transparently. It must be possible to carry out data migration while applications are running, without impacting their performance or availability. The logging approach used for data durability makes data movement relatively straightforward: data can be copied at leisure from source to destination while continuing to serve requests at the source; then the update log can be replayed on the destination to incorporate any changes made during the copy; finally a short synchronized operation between source and destination can be used to apply any final updates and transfer ownership.

4.5 Concurrency, transactions, and consistency

One of the most challenging issues for RAMCloud is how to handle interactions between requests that are being serviced simultaneously. For example, can an application server group a series of updates into a transactional unit so that other application servers either see all of the updates or none of them? The ACID guarantees provided by relational database systems handle concurrency in a clean and powerful way that

simplifies the construction of applications [19]. However, the ACID properties scale poorly: no one has yet constructed a system with ACID properties at the scale we envision for RAMClouds. Many Web applications do not need full ACID behavior and do not wish to pay for it.

Because of this, recent storage systems have explored a variety of alternative approaches that give up some of the ACID properties in order to improve scalability. For example, Bigtable [4] does not support transactions involving more than a single row and Dynamo [8] does not guarantee immediate and consistent updates of replicas. These restrictions improve some system properties, such as scalability or availability, but they also complicate life for developers and cause confusing behavior for users. For example, in a system that doesn't guarantee consistency of replicas, if a user reads a value that he or she wrote recently ("read after write") it is possible for the read to return the old value.

One potential benefit of RAMCloud's extremely low latency is that it may enable a higher level of consistency than other systems of comparable scale. ACID properties become expensive to implement when there are many transactions executing concurrently, since this increases the likelihood of conflicts. Concurrency is determined by (a) the amount of time each transaction spans in the system and (b) the overall rate at which transactions enter the system. Longer execution times for transactions increase the degree of concurrency, as does greater system throughput. By reducing latency, RAMClouds reduce the length of time each transaction stays in the system, which reduces aborted transactions (for optimistic concurrency control) and lock wait times (for pessimistic concurrency control). As a result, a low latency system can scale to much higher overall throughput before the cost of ACID becomes prohibitive.

However, the potential for stronger consistency will be harder to realize for applications that require replication across datacenters. In these applications inter-datacenter wire delays will force high write latencies, which will increase the degree of concurrency and make stronger consistency more expensive.

In addition, as discussed in Section 4.4, the high throughput of a RAMCloud makes data replication unnecessary. This eliminates the consistency issues and overhead associated with replication, and it creates a simpler programming model for application developers.

4.6 Multi-tenancy

A single large RAMCloud system must support shared access by hundreds of applications of varying sizes. A small application should get all of the performance and durability advantages provided by RAMCloud at a cost proportional to its memory usage. For example, it should be possible to provide a few gigabytes of storage to an application for only a few hundred dollars per year, since this represents just a small fraction of one server. If an application suddenly becomes popular it should be able to scale quickly within its RAMCloud to achieve high performance levels on short notice. Because of its scale, a RAMCloud can efficiently amortize the cost of spare capacity over many applications.

In order to support multiple tenants, RAMClouds will need to provide access control and security mechanisms reliable enough to allow mutually antagonistic applications to cohabitate in the same RAMCloud cluster. RAMClouds may also need to provide mechanisms for performance isolation, so that one application with a very high workload does not degrade performance of other applications. Performance isolation is a difficult challenge, and could require partitioning of both data and network bandwidth in the system.

4.7 Server-client functional distribution

A RAMCloud system is distributed in two different ways: application code runs on separate machines from storage servers, and the stored data is spread across thousands of servers. We have already discussed the distribution between storage servers; we now turn to the distribution between applications and storage servers, which raises interesting questions about where various functions in the system should be implemented.

Applications will access RAMCloud storage using a library package resident on the application servers. The library might be as simple as a thin wrapper layer on top of the RAMCloud RPC calls, but it will probably implement more significant functionality. For example, the library is likely to include a cache of configuration information so that RPCs can be directed immediately to the correct server for the desired data. There are several other functions whose implementation probably belongs in the client library as well. For example, suppose the storage system implements a query that retrieves several objects from a table and returns them in sorted order. The objects themselves may reside on multiple servers, so it is most efficient for the client library to collect partial results from each of the servers and combine and sort them locally (otherwise all of the entries would have to be collected on a single server

machine for sorting and then retransmitted to the client, resulting in extra network traffic). If the object model provided by storage servers consists of blobs of bytes, the client library is a natural place to implement a particular data model within objects, as well as higher level queries and support for indexing as described in Section 4.3. Different client libraries can implement different data models and query languages.

It is also possible that some system management functions could be implemented in the client library, such as orchestrating recovery after server crashes. However, this would complicate the system's recovery model, since an ill-timed client crash might impact the system's recovery after a server crash. It probably makes sense for the storage servers to be self-contained, so that they manage themselves and can meet their service level agreements without depending on any particular behavior of client machines.

Certain functions, such as access control, cannot be implemented safely on the clients. There is no way for a server to tell whether a particular request came from the client library or from the application itself; since the application isn't trusted to make access control decisions, the client library cannot be trusted either.

It may also make sense to migrate functionality from the clients to the servers. For example, in bulk-processing applications such as MapReduce [7] it may be attractive for an application to upload code to the servers so that data can be processed immediately on the servers without shipping it over the network. This approach makes sense for simple operations that reduce network traffic, such as counting records with particular attributes. Of course, uploading code to the servers will introduce additional security issues; the servers must ensure that malicious code cannot damage the storage system. If a RAMCloud can return data over the network with very high efficiency, then there will be less need for code uploading.

4.8 Self-management

For RAMClouds to be successful they must manage themselves automatically. With thousands of servers, each using hundreds of its peers for a sharded backup scheme, the overall system complexity will be far too great for a human operator to understand and manage. In addition, a single RAMCloud may support hundreds of applications with competing needs; it will need to monitor its own performance and adjust its configuration automatically as application behavior changes. Fortunately, building a new storage system from scratch provides an opportunity to get this essential functionality done right, thereby reducing operational cost compared to current storage systems.

Instrumentation will be a key element in RAMCloud systems. The servers will need to gather significant amounts of data to drive system management decisions, and data from different servers will need to be integrated to form a cohesive overall picture of system behavior. Furthermore, data gathering must not significantly degrade the latency or throughput of the system.

5 RAMCloud Disadvantages

The most obvious drawbacks of RAMClouds are high cost per bit and high energy usage per bit. For both of these metrics RAMCloud storage will be 50-100x worse than a pure disk-based system and 5-10x worse than a storage system based on flash memory (see [1] for sample configurations and metrics). A RAMCloud system will also require more floor space in a datacenter than a system based on disk or flash memory. Thus, if an application needs to store a large amount of data inexpensively and has a relatively low access rate, RAMCloud is not the best solution.

However, RAMClouds become much more attractive for applications with high throughput requirements. When measured in terms of cost per operation or energy per operation, RAMClouds are 100-1000x *more* efficient than disk-based systems and 5-10x more efficient than systems based on flash memory. Thus for systems with high throughput requirements a RAMCloud can provide not just high performance but also energy efficiency. It may also be possible to reduce RAMCloud energy usage by taking advantage of the low-power mode offered by DRAM chips, particularly during periods of low activity.

In addition to these disadvantages, some of RAMCloud's advantages will be lost for applications that require data replication across datacenters. In such environments the latency of updates will be dominated by speed-of-light delays between datacenters, so RAMClouds will have little or no latency advantage. In addition, cross-datacenter replication makes it harder for RAMClouds to achieve stronger consistency as described in Section 4.5. However, RAMClouds can still offer exceptionally low latency for reads even with cross-datacenter replication.

6 Related Work

The role of DRAM in storage systems has been steadily increasing over a period of several decades and many of the RAMCloud ideas have been explored in other systems. For example, in the mid-1980s there were

numerous research experiments with databases stored entirely in main memory [9, 11]; however, main-memory databases were not widely adopted, perhaps because of their limited capacities. The latency benefits of optimizing a storage system around DRAM have also been demonstrated in projects such as Rio Vista [15].

In recent years there has been a surge in the use of DRAM, driven by the performance requirements of large-scale Web applications. For example, both Google and Yahoo! store their search indices entirely in DRAM. Memcached [16] provides a general-purpose key-value store entirely in DRAM, and it is widely used to offload back-end database systems (however, memcached makes no durability guarantees so it must be used as a cache). The Bigtable storage system allows entire column families to be loaded into memory, where they can be read without any disk accesses [4]. Bigtable has also explored many of the issues in federating large numbers of storage servers.

The limitations of disk storage have been noted by many. For example, Jim Gray has predicted the migration of data from disk to random access memory [14], and the H-store project has reintroduced the notion of main-memory databases as a solution to database performance problems [20]. Several projects, such as [5] and [10], have explored mechanisms for very low latency RPC communication.

7 Conclusion

In the future, both technology trends and application requirements will dictate that a larger and larger fraction of online data be kept in DRAM. In this paper we have argued that the best long-term solution for many applications may be a radical approach where all data is kept in DRAM all the time. The two most important aspects of RAMClouds are (a) their extremely low latency and (b) their ability to aggregate the resources of large numbers of commodity servers. Together, these allow RAMClouds to scale to meet the needs of the largest Web applications. In addition, low latency also enables richer query models, which will simplify application development and enable new kinds of applications. Finally, the federated approach makes RAMClouds an attractive substrate for cloud computing environments that require a flexible and scalable storage system.

Numerous challenging issues must be addressed before a practical RAMCloud can be constructed. At Stanford University we are initiating a new research project to build a RAMCloud system. Over the next few years we hope to answer some of the research questions about how to build efficient and reliable RAMClouds, as well as to observe the impact of RAMClouds on application development.

8 Acknowledgments

The following people made helpful comments on drafts of this paper, which improved both the presentation in the paper and our thinking about RAMClouds: David Andersen, Michael Armbrust, Jeff Dean, Robert Johnson, Jim Larus, David Patterson, Jeff Rothschild, and Vijay Vasudevan.

9 References

[1] Andersen, D., Franklin, J., Kaminsky, M., et al., "FAWN: A Fast Array of Wimpy Nodes", *Proc. 22nd Symposium on Operating Systems Principles*, 2009, to appear.

[2] *Arista Networks 7100 Series Switches*, http://www.aristanetworks.com/en/7100Series.

[3] Armbrust, M., Fox, A., Griffith, R., et al., *Above the Clouds: A Berkeley View of Cloud Computing*, Technical Report UCB/EECS-2009-28, Electrical Engineering and Computer Sciences, U.C. Berkeley, February 10, 2009, http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.pdf.

[4] Chang, F., Dean, J, Ghemawat, S., et al., "Bigtable: A Distributed Storage System for Structured Data", *ACM Transactions on Computer Systems*, Vol. 26, No. 2, 2008, pp. 4:1 - 4:26.

[5] Chun, B., Mainwaring, A., and Culler, D., "Virtual Network Transport Protocols for Myrinet," *IEEE Micro*, Vol. 18, No. 1 (January 1998), pp. 53-63.

[6] Cooper, B., Ramakrishnan, R., Srivastava, U., et. al., "PNUTS: Yahoo!'s Hosted Data Serving Platform," *VLDB '08, Proc. VLDB Endowment*, Vol. 1, No. 2, (2008), pp. 1277-1288.

[7] Dean, J., and Ghemawat, S., "MapReduce: Simplified Data Processing on Large Clusters," *Proc. 6th USENIX Symposium on Operating Systems Design and Implementation*, 2004, pp. 137-150.

[8] DeCandia, G., Hastorun, D., Jampani, M., et al., "Dynamo: Amazon's Highly Available Key-value Store", *Proc. 21st ACM Symposium on Operating Systems Principles*, October 2007, pp. 205-220.

[9] DeWitt, D., Katz, R., Olken, F., et al., "Implementation Techniques for Made Memory Database Systems," *Proc. SIGMOD 1984*, pp. 1-8.

[10] Dittia, Z., *Integrated Hardware/Software Design of a High-Performance Network Interface*, Ph.D. dissertation, Washington University in St. Louis, 2001.

[11] Garcia-Molina, H., and Salem, K., "Main Memory Database Systems: An Overview," *IEEE Transac-*

tions on Knowledge and Data Engineering, Vol. 4, No. 6, December 1992, pp. 509-516.

[12] Gray, J., and Putzolu, G.F., "The Five-minute Rule for Trading Memory for Disc Accesses, and the 10 Byte Rule for Trading Memory for CPU Time," *Proc. SIGMOD 1987*, June 1987, pp. 395-398.

[13] Johnson, R., and Rothschild, J., personal communications, March 24, 2009 and August 20, 2009.

[14] Kallman, R., Kimura, H., Natkins, J., et al., "H-store: a High-Performance, Distributed Main Memory Transaction Processing System," VLDB '08, Proc. VLDB Endowment, Vol. 1, No. 2, (2008), pp. 1496-1499.

[15] Lowell, D., and Chen, P., "Free Transactions With Rio Vista," *16th ACM Symposium on Operating Systems Principles*, October, 1997, pp. 92-101.

[16] *memcached: a distributed memory object caching system*, http://www.danga.com/memcached/.

[17] Patterson, D., Gibson, G., and Katz, R., "A Case for Redundant Arrays of Inexpensive Disks," *Proc. SIGMOD 1988*, June 1988, pp. 109-116.

[18] Ramakrishnan, R., and Gehrke, J., *Database Management Systems*, Third Edition, McGraw-Hill, 2003.

[19] Reuter, A., and Haerder, T., "Principles of Transaction-Oriented Database Recovery," *ACM Computing Surveys*, Vol. 15, No. 4, December 1983, pp. 287-317.

[20] Robbins, S., RAM is the new disk…, http://www.infoq.com/news/2008/06/ram-is-disk.

[21] Rosenblum, M. and Ousterhout, J., "The Design and Implementation of a Log-Structured File System," *ACM Transactions on Computer Systems*, Vol. 10, No. 1, February 1992, pp. 26-52.

[22] Schroeder, B., Pinheiro, E., and Weber, W-D., "DRAM Errors in the Wild: A Large-Scale Field Study," *SIGMETRICS/Performance'09*, pp. 193-204.

[23] Stonebraker, M., Madden, S., Abadi, D., et al., "The End of an Architectural Era (It's Time for a Complete Rewrite)", *Proc. VLDB '07*, pp. 1150-1160.