RAMCloud Storage Technology

Abdulmalik Alluhidan

GW ID: G38221820

**1 Introduction**

For a long period of time, magnetic disks have not supplied the needed performance of saving online data in computer systems. The need of large-scale Web applications is increasing gradually. In times of dealing with large-scale Web applications, it is hard sometimes to avoid application crashes or data loss. Over the years, the technology of disks has advanced impressively in terms of storage size. However, the disk performance has not had that achievement comparing to the improvement of its capacity [1].

Developing the performance of disk technology to meet the required level of working with large Web applications appears to be hard for researchers. To solve this issue, some developers believed that they need to use a whole new method for disk storage. Others thought of using flash-memory instead of disks, which appeared to be much better in terms of performance. It provides another substitution with less latency than disk. Recently, it is frequently used in devices like cameras and music players, but it is getting growing attention for general-purpose online storage [7]. However, large web applications usually run big datasets with a huge amount of data that neither flash nor disk can fully deal with. Therefore, the importance of DRAM in storage has been growing greatly since the beginning of computer systems, especially these years, where it is becoming the main storage system for many web applications [2].

As the scale of web applications is increasing, a lot of their long-range data are being gradually saved in DRAM. Starting in the 1970s, a cache of buffers was placed by UNIX, an operating system, in order to advance their file system level. In the early 2000s, most of the popular search engines were able to save all of their web index in DRAM. A few years after that, several websites like Facebook, Wikipedia, and YouTube have started using caching methods like memcached, a general-purpose distributed memory caching system. “In 2009, Facebook used a total of 150 TB of DRAM in memcached and other caches for a database containing 200 TB of disk storage [6]” (see Figure 1) [2], [3].

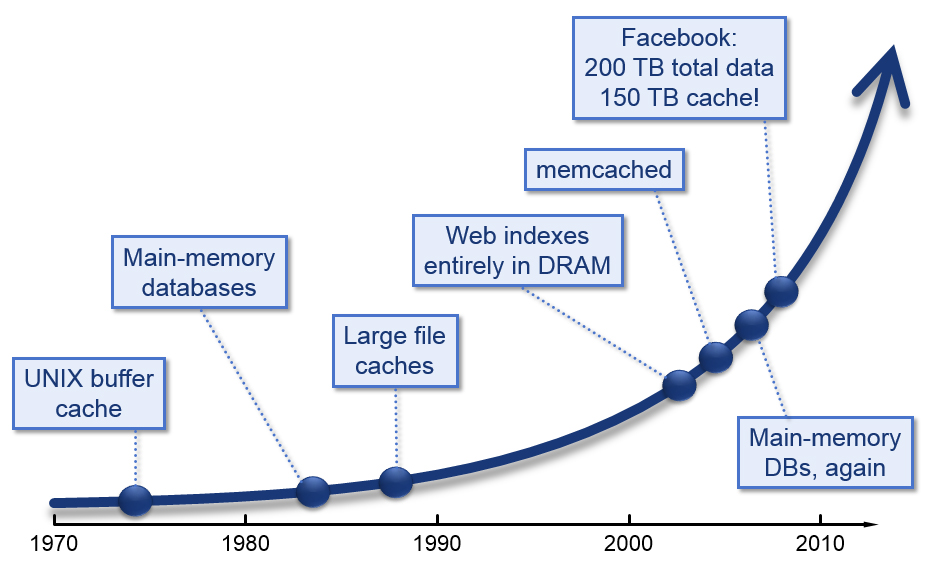


Figure 1. DRAM in Storage Systems. Adapted from [5].

Despite the fact that DRAM’s need is growing, using its full potential is still hard for researchers. Currently, there are two issues with the way DRAM is used as a storage system. First, it is sort of unprofessional for developers to consume a large size of DRAM in their application. That leads to have a minor backing storage in order to maintain a good performance for the application. Therefore, this requires more coding work to adjust the connection between the data in DRAM and the backing store. The second issue is the loss of performance that could occur. Most developers use only a small part of DRAM’s potential in their applications. That is due to the fact that DRAM commonly used as a cash of other systems, and the cash misses cost a lot. Also, developers still need to adjust the relationship between DRAM and backing store in this case as well [2], [4].

In recent years, only a few services like Redis and Cassandra, which are similar systems to RAMCloud, have started giving access to the DRAM's data. Nevertheless, these systems' accomplishments still do not meet the full ability of what DRAM storage can achieve. Therefore, a new storage approach called RAMCloud appeared in 2009 to work on this issue [2].

**2 RAMCloud**

RAMCloud is a collaborative work that still going on at the Department of Computer Science at Stanford University of several researchers lead by Professor John Ousterhout. It is a new approach in the area of high speed storage systems that aims to maximize DRAM performance by keeping all the data in DRAM at all times. RAMCloud is an open software and research project that shares many characteristics with some recent services like Redis, which is currently used in Twitter, but much faster in terms of performance.

**2.1 RAMCloud Basics**

RAMCloud aims to change the idea of storing in three different methods. First, it proposes to make the growth of large web applications easier for developers. That can be done by removing several problems that stop development. Second, its low latency will exceedingly allow bigger query types that allow a new category of large applications. Third, a suitable substitution will be provided by RAMCloud for some applications like “Cloud Computing” and others [1].

To achieve maximum performance, RAMCloud is focusing on several basic concepts. To begin with, RAMCloud is not considered to be a cash such as memcash, so there is no such thing like cash miss; basically, that means all bytes stay at DRAM from beginning to end. Also, although all the information is saved in DRAM, RAMCloud has some methods to ensure that all data is durable and available as it is in systems that are based upon disks. In addition, it has been designed to work with large-scale data centers that contain thousands of servers. Also, as of now RAMCloud can save hundreds of TB and could possibly reach petabytes in the future [1].

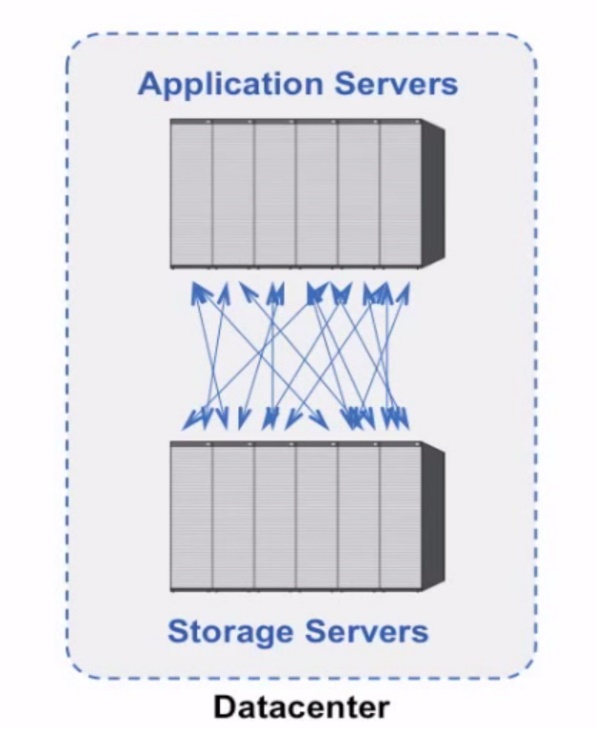
****

Figure 2. RAMCloud Overview. Adapted from [5].

In recent data centers, all devices are usually divided into two groups, which are application servers and storage servers; the application servers are responsible for running application logic like creating Web pages, while storage servers deliver extended storeroom that application servers can share. The idea of RAMCloud is to maximize DRAM storage, with taking cost under consideration (up to 64 GB today), in each server. Then, create a software, which is RAMCloud, to combine all these servers together and offers a single system that gives a lot of consistency (see Figure 2). No less than 1,000,000 small demands per second have to be checked by each multi-core storage server. RAMCloud has to achieve the lowest possible latency in order to get the full performance of DRAM. The target is to give 5-10 μs to access the data from anywhere in the data center [1], [4].

RAMCloud capitalizes on fast networks, like10Gb Ethernet, to give fast performance. Applications sharing RAMCloud the exact datacenter have the ability of getting objects in about 5μs, and that is 1000 x quicker than the performance of disk system. In RAMCloud, data is mechanically copied on minor storage like flash memory or disk. Therefore, when servers face any sort of crashing, the data is still safe. RAMCloud has a major role of bringing back lost data from the DRAM logs to a new server at an excellent speed. Because it does not maintain several copies of data online, it gives high access to users by returning from crashes very fast (1–2 seconds) [1].

By using RAMCloud, large-size applications with a huge amount of data can be run completely. Also, it enables applications to work with a large amount of data immediately in order to give fast reactions to the orders that have been placed by the user. In the same time of assisting big applications, RAMCloud is still giving a satisfied coherence. Therefore, it is believed that RAMCloud eventually will be the main storage for information in clouds situations like Amazon's AWS. RAMCloud was not designed as a research, it is a new storage system that could be use by future applications [5].

**2.2 The Importance of Latency in RAMCloud**

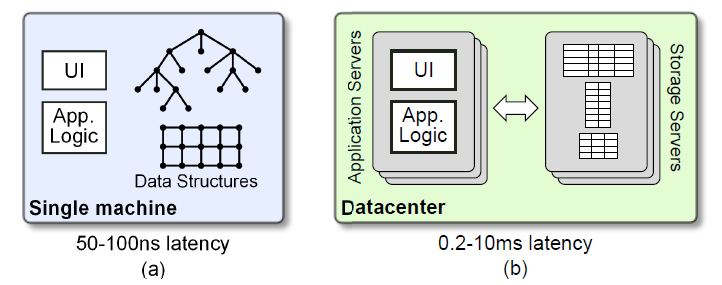
****

Figure 3. Does Latency Matter? Adapted from [2]

“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; the latency for an application to access its data is determined by the last-level cache miss time (50-100 ns). In a scalable Web application (b) the data is stored on separate servers from the application logic and user interface code; the latency for an application to access data over the network ranges from 200-300 μs (if data is cached in the storage server’s DRAM) to10 ms or more (if data is on disk)” (see Figure 3) [2, p4].

One of the best characteristics of RAMCloud comparing to different storage methods is the possibility for extremely low latency. Today, it is difficult to find web applications that need a low latency scale of 5-10 μs. However, that is due to the fact that there is no storage systems that could backup such a demand.

On the other side, it is expected for classic applications to have latency extremely less than 5-10 μs. In this case, the application data stay in main-memory sharing location with the applications front end. The application can travel large amount of data with a low latency. In terms of Web applications, different storage servers save the data. Due to a large data latency, Web applications mainly cannot stand to create complicated unexpected investigations of their data, and this limits the range of operations they can offer. Also, it has been quite expected that Web applications are to substitute classic applications. In this case they will require latency that very close to what classic applications have [1].

Data is already being used in complicated methods by recent Web applications and getting struggled with latency problems. For instance, when an HTTP demand for a Web page is established by websites like Facebook, an average of 130 local demands is made by the application server as a piece of creating the HTML for the page [6]. Because future requests relay on outcomes generated by previous requests, these requests must be created consecutively. The accumulative latency of the local requests is one of the elements that limits the overall reaction period to users. To save local requests from growing, applications running must be limited and large developer potential is produced in methods to conceal latency [1].

A number of issues has also been caused for database applications by access of high latency data. Queries that do not match the layout of data on disk, and hence require numerous seeks, can be extremely slow. Repetitive queries like tree walks, where the data part of *n*th cannot be recognized till the *n-1*th part has been tested, outcome in many high latency requests to be functional. In today's world, many expert database designs have seen the light, like array stores, column stores, and stream processing engines. These designs restructure disk data to minimize latency for a specific type of inquiries a specific application area is accelerated by each order-of-magnitude, but no one of them is a general-purpose. [1].

By offering a low latency of random access to extremely big datasets, RAMClouds will not make the growth of present applications easier only, but they also will allow a new scale of applications that can access a huge size of information more intensely than before. For instance, large multiuser games including communications between thousands of users. In general, RAMClouds will fit algorithms that should cross big random chart structures, where the access models are so unexpected that the entire performance of the storage system's latency must be acquired for each part restored. A single consecutive factor will be allowed by RAMClouds to execute faster on such a frame. Also, thousands of factors will be permitted to execute simultaneously on shared charts. [1].

**2.3 RAMCloud Data Model**

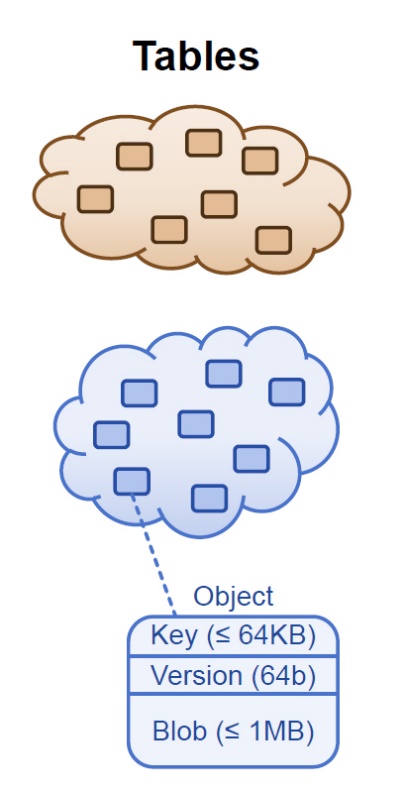
****

Figure 4. RAMCloud Cluster Architecture. Adapted from [5]

"The current data model in RAMCloud is a simple key-value store. RAMCloud supports any number of tables, each of which contains any number of objects. An object consists of a 64-bit identifier, a variable-length byte array (up to 1 MB), and a 64-bit version number (see Figure 4). RAMCloud provides a simple set of operations for creating and deleting tables and for reading, writing, and deleting objects within a table. Objects are addressed with their identifiers and are read and written in their entirety. There is no built-in support for atomic updates to multiple objects, but RAMCloud does provide a conditional update (“replace the contents of object O in table T only if its current version number is V ”), which can be used to implement more complex transactions in application software. In the future we plan to experiment with more powerful features such as indexes, mini-transactions [7], and support for large graphs [3]."

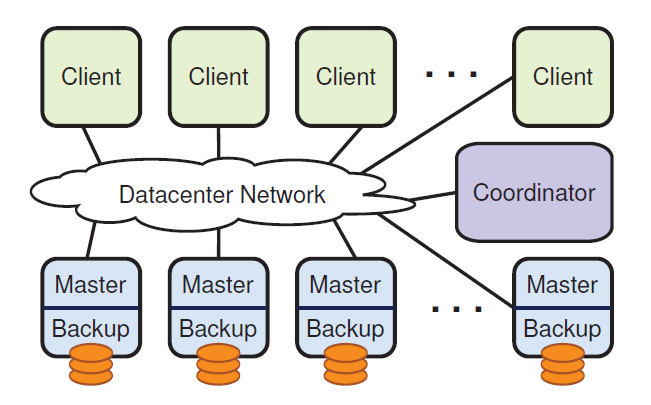
**2.4 RAMCloud System Structure**

Figure 5. RAMCloud cluster architecture. Adapted from [3]

As it is presented in Figure 5, each one of the storing servers has a master and a backup. The server pool and tablet are controlled by a main coordinator. Client applications execute on different devices and approach RAMCloud using a client library that create remote execution calls. Many storing servers are consisted in a RAMCloud body, each one of them has two parts: a master, which controls RAMCloud objects in its DRAM and facilities client demands, and a backup, which saves extra duplicates of objects from other masters via its disk or flash memory. Each RAMCloud group also consists of one illustrious server called the coordinator. The coordinator controls outline information like the network addresses of the storing servers and the sites of objects; it is not engaged in most client demands [3].

The coordinator asks objects to store servers in units of *tablets*: Objects are asked by the coordinator to store servers in parts of tablets: sequential key domain within one table.

The entire minor tables are saved in their on one storing server; big tables are divided across many servers. The tablet configuration cannot be controlled by client applications. However, a locality can be accomplished by the client applications by taking benefit of the point that minor tables (and close keys in big tables) are saved jointly on one server [3].

The charting is saved between tablets and storage servers by the coordinator. The client library of RAMCloud keeps a cache of this data, fetching the charting for each table the first time it is controlled. Storing demands can be placed to the appropriate storing server by clients immediately with no need for the coordinator to be engaged. If a client’s cached group information becomes old due to a tablet moving, the client library found this when it issues a demand to a server that no longer has the tablet. At this point, it glows the old data from its cache and fetch updated data from the coordinator.

**2.5 Managing Replicas in RAMCloud**

The need to offer durability and availability is the main point that decide the structure inside a RAMCloud storage server. When these demands missed, a master would contains little more than a hash table, which a structure that can map keys to values, that maps from (table recognizer, object recognizer) get together to objects in DRAM. The major difficulty is giving durability and availability without having to sacrifice performance or gradually growing system price [3].

Replicating each one of the objects in the storages of some servers is one likely way to availability. However, with a stander replication factor of three, both the price and energy use of the system would be tripled by using this approach. In fact, each one of the servers is already completely loaded, so adding more storage would also demand adding additional networking and servers. The major-memory replication fees can be less by using coding methods like parity striping [8], so this makes crash recovery greatly more costly. Moreover, in case of power outage, DRAM-based replicas are still susceptible [3].

Instead of doing that, only one copy of each object in DRAM is maintained by RAMCloud, with additional copies on minor storing like disk or flash. By using that approach, replication would be almost free in the matter of energy use and cost (the DRAM will control both of these factors in times of dealing with main copies) but it brings a couple of problems. Firstly, the regular performance of the system might be effected by the use of slower storage for backup. Secondly, since the information will have to be rebuilt from minor storing, this method could consequence in long-term periods of unavailability or poor quality performance epically after server crashes. The next section gives an idea of how RAMCloud found solutions for this issue [3].

**2.6 Log-Structured Storage**

A logging approach is used by RAMCloud to controls object data. This was primarily driven by the will to get backup information transferred to flash or disk in the most efficient possible way. However, it also supplies an effective memory control technique, allows quick recovery, and has a basic way for execution [3].

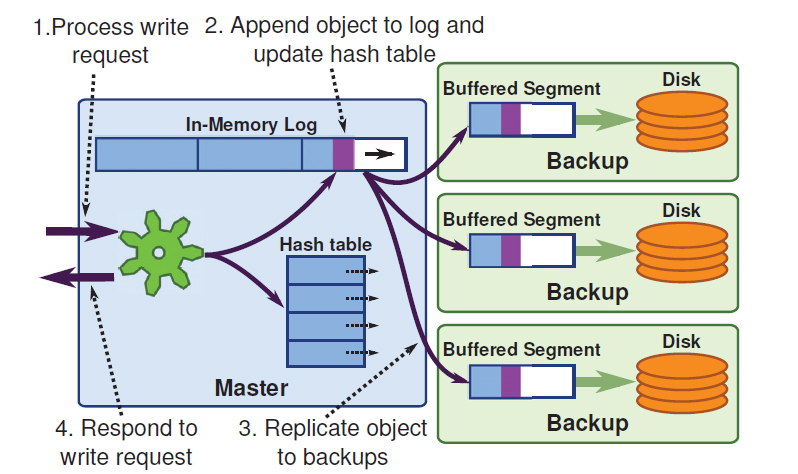


Figure 6. Buffered logging. Adapted from [3]

When a write request arrives at a master server, it creates a copy of its local memory (a copy of record). Then, it sends an operation log entry to some number of backups, which stores object information in their memory. After that, it returns immediately without writing anything to disk. Once the backup have all returned, the master returns back to clients and informs them that the data have been found and they are safe. However, the data still not in the disk yet. The way it gets to disk is that the backups collect a whole bunch of data and write it to disk in one big operation. In this way, disk can be used with its full bandwidth and not limited by disk seek time. Once that done, the backups can discard the copy from memory (see Figure 6). More importantly, Backups need to use a supplementary source of power to guarantee that objects can be written to settled storage after a power outage [3], [4].

RAMCloud controls its logs by using methods similar to the methods in log-structured file systems. 8 MB sections are separated by each master’s log. A large size of unused capacity within each section is kept by the master, which increases as objects are removed or overwritten. It improves lost space by implementing a log cleaner sometimes; one or more sections are chosen by the cleaner to clean, reads the live records from the sections and rewrites them at the top of the log, then removes the cleaned sections as well as their backup copies. Sections are also the part of buffering and I/O on backups; the large section size allows effective I/O for disk and flash [3].

The log-structured method is not only used by RAMCloud for backup room, but for data in DRAM as well: the master’s memory is built as a set of log segments similar to those saved

on backups. This enables masters to control both memory data and backup data using one single technique. “The log provides an efficient memory management mechanism, with the cleaner

implementing a form of generational garbage collection. In order to support random access to objects in memory, each master keeps a hash table that maps from table identifier, object identifier pairs to the current version of an object in a segment. The hash table is used both to look up objects during storage operations and to determine whether a particular object version is the current one during cleaning (for example, if there is no hash table entry for a particular object in a segment being cleaned, it means the object has been deleted) [3]” [3].

**3. Recovery**

In today’s world, there are two major problems that mainly face cash recovery. The first one is that backup sometimes loses its power, which means losing information saved in its memory, and hence it needs to ensure that buffered in memory is just as durable as those in disks. Therefore, backup needs to has some continues solutions so when the power outage occurs, it will at least have enough time to flush all of its buffered data out to disk. There are many different ways to do that. One of the most interesting ways is the memory modules, which are starting to appear in the last few years, that look and behave just like regular DRAM memories except that they have also a little flash chip and a cap capacitor with enough power. When there is a shortage in the main power, the DIMM by itself can flush its contents to flash memory, and hence it can be recovered from there when the system reboots. Another way is what Google uses, they use per-server battery backups in their data centers. Therefore, machines will have enough power to last for few minutes once the main power goes off [4].

The Second problem is the server crashes. When a server crashes, the best way to recover it is to reply the entire log for that server so that users can reinstruct the hash table. While that is happening, users unfortunately cannot service any requests. The way this issue is solved in RAMCloud is to make the crash recovery extremely fast to the extent that makes users will not even notice the crash, and applications will continue their operations normally. Considering the fact that there is no disk that can read 64 GB in 1or 2 seconds, or a network that can transmit that, the RAMCloud mechanism uses the scale of the system to do it in parallel. In fact, there are three different approaches to do that, and they are introduced in the coming sections [4].

The first approach, which is not an ideal, does not use a scale at all. In this method, each master picks three backups and keeps a copy of its entire log in each one of the backups. When the master crashes, backups select a recovery master to take control of that. Then, the data has to go out of the disks on the backups, and transmitted over the network to the recovery master; which replied the logs, reinstruct the hash table, and then resume the operation. The issue with

this approach is the fact that it only has three disks to read 64 GB, which takes a long time

(3-4 minutes). Therefore, the ideal way to improve the recovery process is to use more disks and backups, which introduced in the next approach [4].

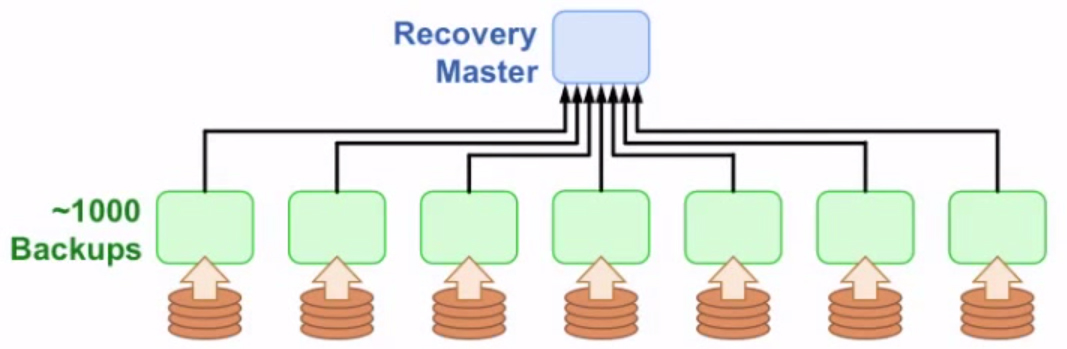


Figure 7. The Second Recovery Approach. Adapted from [5]

The second approach is that each master divides its log to a group of segments. Each segment gets backed up on a different set of backups chosen randomly, and every master will have data on every backup in the cluster. When a master crashes in this method, all the machines in the cluster can be used for recovery. For example, if there are 1000 backups, they can all in parallel read small chunks of the master's data in less than 1 second (see Figure7). Therefore, this method is faster and doesn't require waiting for a long time for the disk. However, this only apply on memories of the 1000 disks and not the memory of the master. The solution for that is exceeding the number of the recovery masters which is the third approach [4].

The third approach, which is used in RAMCloud, is to have more recovery masters. If there are a 100 recovery masters and every one of them recovers one hundred copies of the lost information. Then, it is possible that the data get transferred in about half a second if we apply a network speed of 10 GB/second. When a crash happens, the coordinator divides each master's data into a set of partitions. When it is time to recover, master’s data get divided and the coordinator assigns different tablets, ranges of keys within tables, to other masters. Every backup

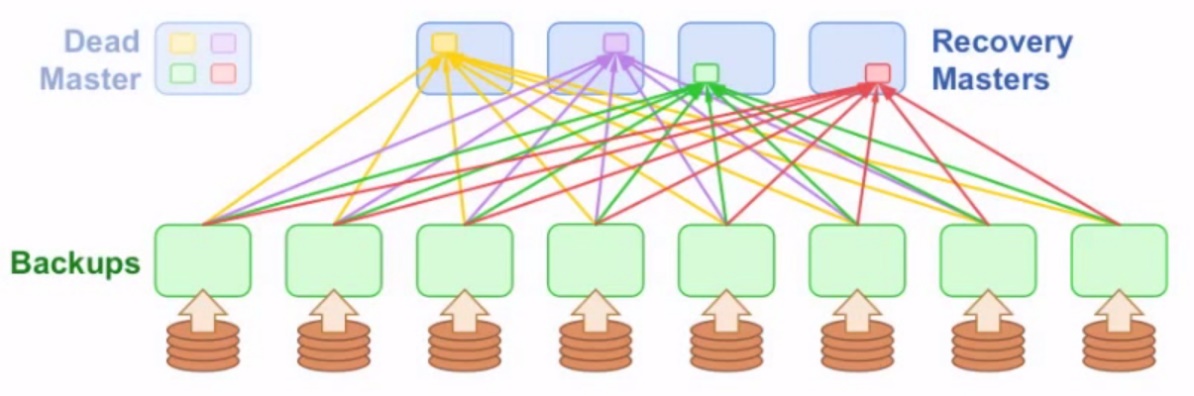


Figure 8. The Third Recovery Approach. Adapted from [5]

divides its log data among recovery masters. A massive parallel pipeline is rapidly used after that to do recovery, where each backup reads data from its disk and divides that data to a number of paces, one piece for each recovery master; then, it get shipped to that master, which replies only the portion of the log for certain objects. As long as the data for each object goes to the same master, we can reply that and find what is the most recent version and reinstruct the hash table. Then, the system will normally resume its operation (see Figure8) [4].

**4. Redis vs. RAMCloud**

Redis is one of the most popular services that shares many features with RAMCloud. It is in fact a key value store that maintains all information in memory. This section is for comparing Redis to RAMCloud from different aspects [9], [10].

**4.1 Data Model**

In terms of data model, the characteristics that Redis provides is more than those in RAMCloud. Professor Ousterhou stated several cases of these features. “For example, in addition to the simple get-set semantics provided by RAMCloud, Redis also provides the following operations:

* Atomic increment (RAMCloud provides this as well).
* Treat values as sets: add to set, remove from set, union, etc.
* Treat values as sorted sets: each element in a set has a "score", which can be used to order the elements.
* Treat values as lists: push, pop, index, range, etc.
* Treat values as bit strings (e.g. count "on" bits).
* Treat values as hash tables.
* Transactions involving multiple operations.
* Publish-subscribe
* Expiration times for objects” [9], [10].

**4.2 Performance**

100K-1M operations/second is the output that Redis seems to have, which is the same scale of a RAMCloud server. Redis seems to have a latency of about 200 μs using Ethernet speed of 1 GB. The latency in RAMCloud is faster, about 5us for reads and 15us for writes. However, this speed is applicable only with faster networks. With a speed of 1 GB Ethernet, it is still better than Redis but less than its double speed [9], [10].

**4.3 Scalability**

On one hand, Redis seems to have been designed to work with a single server. When the systems achieves beyond a single server, the amount of data should be divided among servers. Also, Readis supports range partitioning as well as hash partitioning. The way partitioning managed is by having clients pick the right server. This can be done by sending all demands to a proxy that broadcasts to the right server, or by forwarding demands to a server that has been randomly chosen, which broadcasts them to the right server with extra latency in both cases. However, there are some Redis operations that do not work in a divided world, multi-object transactions is an example of that. In general, it seems that scaling leads to several issues with performance as well as the data model in Redis [5].

On the other hand, RAMCloud was created and advanced to work on the level of thousands of servers. All operations of RAMCloud can work on a high level without getting its performance effected. Scaling in RAMCloud is clear, the size of the system is not important for clients to be known. In one way or another, RAMCloud gets improved as it gets bigger. For instance, as the cluster size grows, crash recovery from a failed server gets quicker [9], [10].

**4.4 Durability**

In general, the data model that Redis provides is comparatively unconvincing. Although snapshotting and processing logging are both provided, data are getting lost usually after a crash happens, even when logging is enabled. The only method to avoid data loss is by using the most typical flushing mode; a good accomplishment can be offered using this way. Also, Redis provides master-slave replication, but they are not simultaneous. Therefore, the loss of data is possible during crashes. In Redis, scaling presents organizational issues: the data on each backup server must be handled individually [10].

In case of RAMCloud, a better durability is offered. RAMCloud offers automatic replication, crash recovery, and fail-over. Data is copied and durable before functioning returns, and without any performance issues happening. So, the durability is assured. Also, no organizational problems for scalability in RAMCloud [10].

**5 Research issues**

Today, although RAMCloud 1.0 has been released and the first PhD student discussing this topic has successfully graduated, there are still several issues that have not been solved yet. The integration of scale and latency generates many absorbing research difficulties. Until recently, the RAMCloud team at Stanford University found solutions for several of them. To begin with, ensuring data durability without losing reads and writes latencies. At a least this means that a single server’s cash must not cause data to be missed or influence system availability for more than a few seconds. Also, making advantage of the system scale to recover very fast after crashes. Lastly, handling storage in DRAM [5].

However, a large number of other issues remain and have not been addressed. One of which is offering effective features such as minor indexes and multiple-object transactions without losing the latency or the system scalability [5].

**6 Conclusion**

Researching about RAMCloud was a great experience. It is totally a new project that started recently, which makes it quite challenging to research due to lack of resources. Its low latency and ability to coordinate many servers allow RAMCloud to achieve the required performance for a large-scale of Web applications that could not exist in the past. RAMCloud’s team at Stanford University is basically trying to construct a storage system that delivers the fastest possible access to the biggest possible datasets. Thus, it uses DRAM as the main place for data, and it integrates the prime memories of a large number of servers to help large-scale datasets. It is predicted in the future that cashing will have to be large enough so that they will offer a small cost advantage while still giving large performance danger. However, RAMCloud may cost a little more than caching systems, but they will offer an assured performance ultimately [1], [2].

**7. References**

[1] Ousterhout, J. et al. (2011, July). The Case for RAM Clouds: Scalable High-Performance Storage Entirely in DRAM. *Communications of the ACM*, Vol.43, No.7.

[2] Ousterhout, J. et al (2014, October). The RAMCloud Storage System. Retrieved from https://ramcloud.stanford.edu**.**

[3] Ongaro, D. et al (2011, October). Fast crash recovery in RAMCloud. *SOSP '11 Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principle*.

[4] LinkedIn Tech Talks. (2011, October 11). “John Ousterhout - RAMCloud: Scalable High-Performance Storage Entirely in DRAM” [Video file]. Retrieved from https://www.youtube.com/watch?v=lcUvU3b5co8.

[5] RAMCloud Retrieved from RAMCloud Project Page https://ramcloud.atlassian.net/wiki/display/RAM/RAMCloud.

[6] R. Johnson and J. Rothschild. *Personal Communications*, March 24 and August 20, 2009.

[7] M. K. Aguilera, A. Merchant, M. Shah, A. Veitch, and C. Karamanolis. Sinfonia: A new

paradigm for building scalable distributed systems. *ACM Trans. Comput. Syst.*, 27:5:1–5:48,

November 2009.

[8] D. A. Patterson, G. Gibson, and R. H. Katz. A case for redundant arrays of inexpensive disks. In Proceedings of the 1988 ACM SIGMOD international conference on management of data.

[9] Redis. Retrieved from Redis Official Page

http://redis.io.

[10] Ousterhout, J, "Redis vs. RAMCloud". Retrieved March, 2015 Available: https://ramcloud.atlassian.net/wiki/display/RAM/Redis+vs.+RAMCloud