Characterizing Performance and Energy-efficiency of The RAMCloud Storage System

Yacine Taleb*, Shadi Ibrahim*, Gabriel Antoniu* and Toni Cortes†
*Inria Rennes Bretagne-Atlantique, Rennes, France
†Barcelona Supercomputing Center, Barcelona, Spain
†Universitat Politècnica de Catalunya, Barcelona, Spain
Email: *first.last@inria.fr, †first.last@bsc.es

***Abstract*—Most large popular web applications, like Facebook and Twitter, have been relying on large amounts of in-memory storage to cache data and offer a low response time. As the main memory capacity of clusters and clouds increases, it becomes possible to keep most of the data in the main memory. This motivates the introduction of in-memory storage systems. While prior work has focused on how to exploit the low-latency of in-memory access at scale, there is very little visibility into the energy-efficiency of in-memory storage systems. Even though it is known that main memory is a fundamental energy bottleneck in computing systems (i.e., DRAM consumes up to 40% of a server's power). In this paper, by the means of experimental evaluation, we have studied the performance and energy-efficiency of RAMCloud — a well-known in-memory storage system. We reveal that although RAMCloud is scalable for read-only applications, it exhibits non-proportional power consumption. We also find that the current replication scheme implemented in RAMCloud limits the performance and results in high energy consumption. Surprisingly, we show that replication can also play a negative role in crash-recovery.**

***Keywords*-In-memory storage, RAMCloud, Performance evaluation, Energy efficiency**

I. Introduction

Nowadays, large scale Web applications, such as Facebook, are accessed by billions of users [1]. These applications must keep low response times even under highly concurrent accesses. To do so, they strongly rely on main memory storage through caching [1]–[3].

The increasing size of main memories has lead to the advent of new types of storage systems. These systems propose to keep all data in distributed main memories [4]–[6]. In addition to leveraging DRAM speed, they can offer: *low-latency* by relying on high speed networks (e.g., Infiniband) [4], [5], [7]; *durability* by replicating data synchronously to DRAM or asynchronously to disk [4]–[6], [8], [9]; *strong consistency* and transactional support [4], [6], [10]; and *scalability* [4], [7]. RAMCloud [4] is a system providing the aforementioned features. It is now used in various fields: analytics [11], or used as a low-latency storage for SDNs [12], [13], or for scientific workflows [14].

While energy-efficiency has long been an important goal in storage systems, e.g., disk-based storage [15]–[17] and flash-based storage [18], prior literature has not investigated enough the energy-efficiency of in-memory storage systems. Some studies reported that DRAM-based main memories consume from 25% up to 40% of a server's total power consumption [19].

Given the increasing concerns about datacenter energy consumption which will cost $13B to the American businesses by 2020 [20] and the prevalence of in-memory storage systems, this paper aims to provide a clearer understanding of the main factors impacting performance and energy-efficiency of in-memory storage systems. We carry out an experimental study on the Grid'5000 [21] testbed, and use the RAMCloud in-memory key-value store. We design and run various scenarios that help us to identify the main performance bottlenecks and sources of energy-inefficiency. More precisely we answer the following questions:

- *What is the energy footprint of peak performance of RAMCloud?* We reveal issues of power non-proportionality that appear with read-only applications.
- *What is the performance and energy footprint of read-update workloads?* We focus on these workloads (mixed reads and updates) as they are prevalent in large scale Web applications [3].
- *How does RAMCloud's replication scheme impact performance and energy?* Though it is supposed to be transparent in RAMCloud, we find that replication can be a major performance and energy bottleneck.
- *What is the overhead of RAMCloud's crash-recovery in terms of availability and energy consumption?* In addition we study how replication affect these metrics. Surprisingly, we show that replication can play a negative role in crash-recovery.
- *What can be improved in RAMCloud, and for in-memory storage?* We discuss possible improvements for RAMCloud and derive some optimizations that could be applied in in-memory storage systems.

It is important to note that, although RAMCloud is used as an experimental platform in this work, our target is more general. Our findings can serve as a basis to understand the behavior of other in-memory storage systems sharing the same features and provide guidelines to design energy-efficient in-memory storage systems.

II. Background

A. A representative system: RAMCloud

Ideally, the main attributes that in-memory storage systems should provide are performance, durability, availability, scalability, efficient memory usage, and energy efficiency. Most of today's systems target performance and memory

efficiency [7], [22], [23]. Durability and availability are also important as they free up application developers from having to backup in-memory data in secondary storage and handle the synchronization between the two levels of storage. On the other hand, scalability is vital, especially with today's high-demand Web applications. They are accessed by millions of clients in parallel. Therefore, large scale clustered storage became a natural choice to cope with such high demand [1].

In contrast with most of the recent in-memory storage systems, RAMCloud main claims are performance (low-latency), durability, scalability, and memory efficiency. The other closest system to provide all these features to be found in the literature is FaRM [5]. Unfortunately it is neither open-source nor publicly available.

B. The RAMCloud Storage System

Architecture A RAMCloud's cluster consists of three entities: a coordinator maintaining meta-data information about storage servers, backup servers, and data location; a set of storage servers that expose their DRAM as storage space; and backups that store replicas of data in their DRAM temporarily and spill it to disk asynchronously. Usually, storage servers and backups are collocated within a same physical machine [8].

Data management Data in RAMCloud is stored in a set of tables. Each table can span multiple storage servers. A server uses an append-only log-structured memory to store its data and a hash-table to index it. The log-structured memory of each server is divided into 8MB segments. A server stores data in an append-only fashion. Thus, to free unused space a **cleaning mechanism** is triggered whenever a server reaches a certain memory utilization threshold. The cleaner copies a segment's live data into the free space (still available in DRAM) and removes the old segment.

Data durability and availability Durability and availability are guaranteed by replicating data to remote disks. More precisely, whenever a storage server receives a write request, it appends the object into its latest free segment, and forwards a replication request to the backup servers randomly chosen for that segment. The server waits for acknowledgements from all backups to answer a client's update/insert request. Backup servers will keep a copy of this segment in DRAM until it fills. Only then, they will flush the segment to disk and remove it from DRAM.

Fault-tolerance For each new segment, a random backup in the cluster is chosen in order to have as many machines performing the crash-recovery as possible. At run time each server will compute a *will* where it specifies how its data will be partitioned if it crashes. If a crashed server is detected then each server will replay the segments that were assigned to it according to the crashed server's will. As the segments are written to a server's memory, they are replicated to new backups. At the end of the recovery the segments are cleaned from old backups.

III. Experimental Settings

A. Metrics

We took as main metrics the throughput of the system (i.e., the number of requests served per second) and the power consumption of nodes running RAMCloud. We used as well the energy efficiency metric [17], i.e., $Number\_of\_requests\_served/joule$.

B. Platform

The experiments were performed on the Grid'5000 [21] testbed. The Grid'5000 platform provides researchers with an infrastructure for large-scale experiments. It includes 9 geographical sites spread across French territory and one located in Luxembourg.

We used Nancy's site nodes to carry out our experiments. More specifically, the nodes we used have 1 CPU Intel Xeon X3440, 4 cores/CPU, 16GB RAM, and a 298GB HDD. Additionally each node includes a Infiniband-20G and a Gigabit Ethernet cards. We have chosen these nodes as they offer capability to monitor power consumption: 40 of these nodes are equipped with Power Distribution Units (PDUs), which allow to retrieve power consumption through an SNMP request. Each PDU is mapped to a single machine, allowing fine grain power retrieval. We run a script on each machine which queries the power consumption value from its corresponding PDU every second. We start the script right before running the benchmark and stop it after all clients finish.

RAMCloud's configuration Throughout all our experiments we make sure to reserve the whole cluster, which consists of 131 nodes, to avoid any interference with other users of the platform. We dedicate the 40 nodes equipped with PDUs to run RAMCloud's cluster, i.e., master and backup services. One node is used to run the coordinator service. The remaining 90 nodes are used as clients. We have fixed the memory used by a RAMCloud server to 10GB and the available disk space to 80GB. We have fixed the memory and disk to much larger sizes than the workloads used as we explain later in Section V.

We configured RAMCloud with the option *ServerSpan* equal to the number of servers. As RAMCloud does not have a smart data distribution strategy, this option is used to manually decide how many servers each table will span. Data is then distributed uniformly.

Throughout all our experiments, we used RAMCloud's Infiniband transport only. The impact of the network on performance and energy efficiency has been studied in [24].

C. Benchmark

We used the industry standard Yahoo! Cloud Serving Benchmark (YCSB) benchmarking framework [25]. YCSB is an open and extensible framework that allows to mimic real-world workloads such as large scale web applications, to benchmark key-value store systems. YCSB supports a large number of databases and key-value stores, which is convenient for comparing different systems.

To run a workload, one needs to fill the data-store first. It is possible to specify the request distribution, in our case we use uniform distribution. Executing the workload consists of running clients with a given workload specification. We used three basic workloads provided by YCSB: *Workload A* which is an update-heavy workload (50% reads, 50% updates),

Workload B which is a read-heavy workload (95% reads, 5% updates), and *Workload C* which is a read-only workload. This combination of workloads has already been used in other systems evaluations [7], [9], [22]. Other workloads could be used to assess different features of the system, e.g., scans to evaluate the indexing mechanism, however, we leave it as future work.

When assessing RAMCloud's peak performance as we do in Section IV, we use a workload of 5M records of 1KB, and 10M requests per client. Running the benchmark consists of launching simultaneously one instance of a YCSB client on each client node. With 30 clients it corresponds to 300M requests which represents 286GB of data requested per run. In some runs, e.g., when running 30 clients with a single RAMCloud node, the execution time reaches in average 4300 seconds, which we believe outputs enough and representative results to achieve our goals.

For the rest of the experiments (Sections V and VI, where we use update-workloads, we pre-load 100K records of 1KB in the cluster. Each client issues 100K requests, which corresponds to the total number of records. Having each client generate 100K requests results in having 1M requests with 10 clients for example, and 9M requests with 90 clients, which corresponds to 8.58GB of data requested per run. With workload A, it corresponds as well to 4.3GB of data inserted per run. Therefore, we avoid saturating the main memory (and disk when using replication) of servers and trigger the cleaning mechanism.

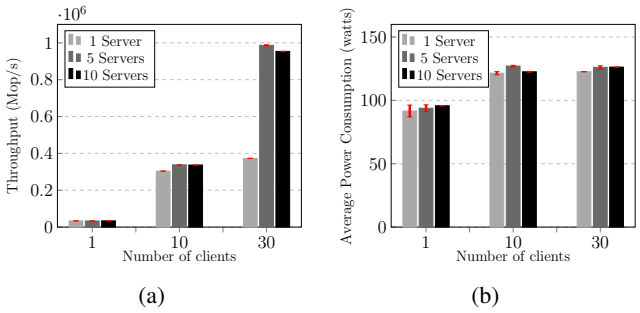
In our figures, each value is an average of 5 runs with the corresponding error bars. When changing the cluster configuration (i.e., number of RAMCloud servers), we remove all the data stored on servers as well as in backups, then we restart all RAMCloud servers and backups in the cluster to avoid any interference with prior experiments. Overall, we made roughly 3000 runs with a total run time of approximately 1000 hours.

IV. The Energy Footprint of Peak Performance

In this section we present our experiments on understanding of the maximum achievable performance. It is important since it gives us a landmark that can help us compare with our next experiments and identify potential performance bottlenecks. Furthermore, given that baseline we can compute energy-efficiency as well.

A. Methodology

To allow RAMCloud to deliver its maximum performance, we configured our experiments as follows: (1) Disabling replication to avoid any communication between RAMCloud servers. In that way we have server-client communication only. (2) We use read-only workloads to avoid write-write race condition, and more importantly to prevent appending data to memory and triggering the cleaning mechanism of RAMCloud. (3) We use Infiniband network. (4) We make sure that the dataset is always smaller than memory capacity. (5) We distribute data uniformly (i.e., at RAMCloud cluster level) and requests (i.e., at the clients level) over the cluster to distribute work equally among servers and avoid hotspots.



(a) (b)

Fig. 1: The aggregated throughput (a) and average power consumption per server (b) as a factor of the cluster size

(6) We use a single client per machine to avoid any network interference at the NIC level of a machine or any CPU contention between client processes within a same machine.

We varied the RAMCloud cluster size from a single server to 5 and 10 servers. We did as well vary the clients number from one to 10 and 30 clients. We used the workload consisting of 5M objects and 10M read-only request per client.

The maximum throughput per server: Figure 1a shows the total aggregated throughput for this scenario. In one node experiment, the system scale up to 10 clients until it reaches its limit at 30 clients for a throughput of 372Kreq/s (this is inline with the results presented in [26]). Increasing the number of servers to 5 improves the total throughput for 10 clients and 30, achieving linear scalability. However, increasing the number of servers from 5 to 10 does not bring any improvement to the aggregated throughput achieved, which is due to the clients' limited rate at this point.

The corresponding power consumption: Figure 1b shows the average power consumption. With one server and a single client the average power is 92W. It increases to 93W and 95W for 5 and 10 servers respectively. Even with smaller amount of work for 5 and 10 servers we can see that the power consumption remains at the same level. The same behaviour happens when increasing the load. For 10 clients the average power is between 122W and 127W. More surprisingly for 30 clients it stays at the same levels, i.e., 122W and 127W. This suggests that the servers are used at the same level (i.e., CPU) under different loads.

Non-proportional power consumption?: To confirm or invalidate this statement we looked into CPU usage of individual servers when running the workload. Table I displays the minimum and maximum CPU usage values. First we notice that the difference between the minimum and maximum values is not bigger than 2% for all scenarios. What is striking is to see that the difference between CPU usage of one node and 5 and 10 nodes is very small. When digging into details we discovered that RAMCloud hogs one core per machine for its polling mechanism, i.e., polling new requests from the NIC and delivering them to the application to achieve low-latency. Since we are using 4-core machines, this ends up using 25% of CPU all times, even without any clients.

Surprisingly, increasing the RAMCloud cluster size from 1 to 5 servers increases the aggregated throughput by 10% only

Clients \ Servers	1	5	10
	average	min — max	min — max
0	25	25 — 25	25 — 25
1	49,81	49,65 — 49,78	49,61 — 49,91
2	74,16	72,12 — 72,72	62,60 — 63,85
3	79,66	74,03 — 74,41	72,18 — 73,27
4	89,80	77,81 — 78,66	74,27 — 75,27
5	94,34	84,90 — 85,98	75,87 — 77,02
10	98,35	96,93 — 97,40	91,89 — 93,06
30	99,26	96,77 — 97,19	94,86 — 95,98

TABLE I: The minimum and maximum of the average CPU usages (in percentage) of individual nodes when running read-only workload on different RAMCloud cluster sizes and with different number of clients

while keeping the same CPU usage per node. We think this is an overhead to handle all concurrent incoming requests. By looking at the scenario of a single node, if we look carefully at the cases of 10 clients and 30 clients we can see a difference of 23% in throughput for a 1% difference in CPU usage. This suggests that this issue relates to the threads' handling. First, each additional running server has a dedicated polling-thread, which in our case translates to 25% of additional CPU usage. Moreover, not all threads are effectively utilized. For instance, the aggregated throughput and the average CPU usage per node are very similar for 1, 5, and 10 servers when servicing 10 clients. To mitigate the energy non-proportionality issue, one can think of carefully tuning the number of nodes in a cluster to meet the needs of the workloads.

Figure 2 shows the energy-efficiency: As expected the energy efficiency is at its highest with a single server and with the largest number of clients. With 5 servers, the energy-efficiency can barely reach half of the one of a single server. Further increasing the RAMCloud cluster size to 30 decreases the energy efficiency by a factor of 7.6x compared to the one of a single server.

Finding 1: RAMCloud is scalable in throughput for read-only applications. However, we find that it can have non-proportional power consumption, i.e., the system can deliver different values of throughput for the same power consumption. The reason is that servers reach their maximum CPU usage before reaching peak performance. Energy-proportionality can be achieved by adapting the number of servers according to their peak performance.

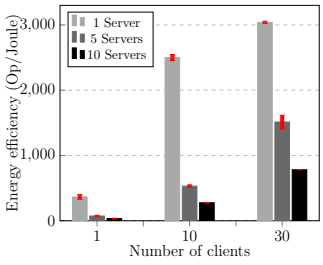


Fig. 2: The energy efficiency of different RAMCloud cluster sizes when running different number of clients.

Clients \ Workload	A	B	C
	avg — err	**avg** — err	**avg** — err
10	**98K** — 4K	**236K** — 11K	**236K** — 18K
20	**106K** — 11K	**454K** — 11K	**482K** — 31K
30	**64K** — 4K	**622K** — 20K	**753K** — 16K
60	**63K** — 2K	**816K** — 26K	**1433K** — 38K
90	**64K** — 2K	**844K** — 19K	**2004K** — 31K

TABLE II: Total aggregated throughput (Kop/s) of 10 servers when running different with different numbers of clients. We used the three YCSB by default workloads A, B,and C

V. The Energy Footprint with Read-Update Workloads

As demonstrated in [3] today's workloads are read-dominated workloads with a GET/SET ratio of 30:1. Hence, the questions we would like to answer are: How good is RAMCloud in terms of performance when running realistic workloads? Does it scale as well as it does for read-only workloads? How energy efficient is it when running realistic workloads? What are the architectural choices that can be improved?

A. Methodology

We used the same methodology as in IV-A except that we changed the workloads to have both read-update (95% reads, 5% updates) and update-heavy (50% reads, 50% updates) workloads. We recall that replication is disabled. It is important to note that we use different number of servers and clients compared to the previous experiment since the goal is different. Thereby, we do not compare the following scenario with the previous one. We use from 10 to 40 servers and from 10 to 90 client nodes. For space's sake we do not show all scenarios, but they all converge towards the same conclusions we give.

Comparing the performance with the three workloads: Table II shows the aggregated throughput for 10 servers when running the three workloads A, B, and C. For read-only workload the throughput increases linearly, reaching up to 2Mop/s for 90 clients. For read-heavy workload the throughput increases linearly until 30 clients where it reaches 622Kop/s. It increases in a much slower pace up to 844Kop/s for 90 clients. The worst case is for update-heavy workload that surprisingly reaches its best throughput of 106Kop/s when running 20 clients, then performance starts declining down to 64Kop/s for 90 clients. At 90 clients, the throughput with workload C is 31x more than the one with heavy-update workload.

To have a better view on this phenomena, Figure 3 shows the ratio of the throughput when taking 10 clients as a baseline. We can see clearly that read-only applications have a perfect scalability, while read-heavy collapses between 30 and 60 clients. With heavy-update workload, throughput does not increase at all and degrades when increasing the number of clients.

Interestingly, we can see what impact do update operations have on performance. Since RAMCloud organizes its memory in a log-structured fashion writes should not have that impact on performance, because every update/insert operation results

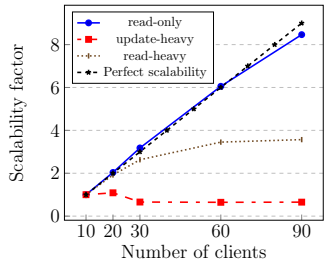
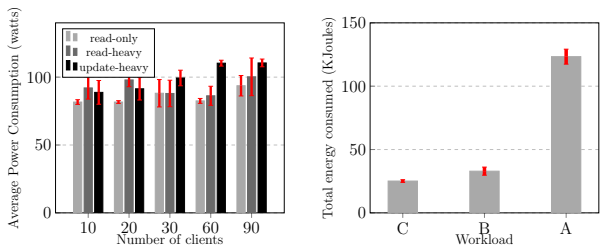


Fig. 3: Scalability of 10 RAMCloud servers in terms of throughput when varying the clients number. The "perfect" line refers to the expected throughput when increasing the number of clients. We take the baseline as the throughput achieved by 10 clients.

in appending new data to the memory log and updating the hash-table. However, bringing up that much concurrency e.g., 90 clients leads to a lot of contention within a single server, and therefore servers will queue most of the incoming requests. This suggests that there is a poor thread handling at the server level when requests are queued.

More updates means more power per node?: The energy impact is straightforward when looking at figure 4a that represents the average power consumption of the RAMCloud cluster. The power consumption per server when executing read-only workloads stays at the same level, i.e., 82Watts up to 60 clients, after that it jumps to 93Watts. We can observe the same pattern for read-heavy except that it stands at a higher power consumption than read-only, which is around 92Watts, then it goes up to 100Watts for 90 clients. Power consumption of the heavy-update workload is the highest, though it starts with 10 and 20 clients around 90Watts, it continues to grow with the number of clients to reach 110Watts for 90 clients which is the highest value for all experiments.

To complement these results we show Figure 4b which displays the total energy consumed when running the three workloads with 90 clients. The total energy consumed is computed by multiplying the power consumption, of every server, collected every second by the execution time. We can see that read-heavy has 28% percent more energy consumption than read-only. The more surprising fact is the difference between heavy-update and read-only which is 492% more energy consumed for the heavy-update workload.



(a) (b)

Fig. 4: (a) represents the average power consumption (Watts) per node of 20 RAMCloud servers as a function of the number of clients. (b) represents the total energy consumed by the same cluster for 90 clients as a function of the workload.

While it is expected that update operations take more time and processing than read operations, it is noteworthy to recall that RAMCloud relies on an append-only log-structured memory. By design updates are equivalent to inserts, therefore the additional overhead should mostly come from updating the in-memory hash-table and appending new data to the log. Consequently, we expected the gap between reads and updates to be very low as we have disabled replication in this particular case. We suspect the main cause being the thread management at the server level. If all threads of a server are busy then upcoming requests will be queued, generating delays. Moreover, the number of servicing threads impacts performance as well. This number can depend on the hardware type (e.g., number of cores). Worse, it depends also on the workload type, e.g., in figure 3 read-only scale perfectly while only update workloads experience performance degradation. For instance, performance will decrease under write-heavy workloads with more threads due to useless context switches, since all work could be done by a single thread. Whereas it is the opposite for read-only applications. Identifying this number empirically can bring substantial performance improvements. This issue was confirmed by RAMCloud developers. [1]

Finding 2: We find that RAMCloud loses up to 57% in throughput with YCSB's read-heavy workload compared to read-only. With heavy-update workloads, the performance degradation can reach up to 97% at high concurrency (with replication disabled in both cases). Furthermore, we found that heavy-update workloads lead to a higher average power consumption per node, which can lead to 4.92x more total energy consumed compared to read-only workloads. We find this issue is tightly related to the number of threads servicing requests. Finding the optimal number can improve performance and energy-efficiency, however, this is not trivial since it depends on the hardware and workload type.

VI. Replication's Impact on Performance and Energy-Efficiency

RAMCloud uses replication to provide durability and availability. There are two main techniques used to replicate data: *Primary-Backup* or *Symmetric* [9]. Since the per bit cost of memory is very high, primary-backup replication (PBR) is usually preferred to symmetric replication, especially when all data is kept in DRAM. RAMCloud uses PBR and keeps a single replica in memory to serve requests and pushes eventually replicas to disk as described in Section II-B. Thus, the main concern about replication is at what extent it impacts the performance and energy consumption of the system and eventually look for possible improvements.

A. Methodology

We measure the transparency of the replication scheme by stressing it with an update-heavy workload (50% reads, 50%

[1] https://ramcloud.atlassian.net/wiki/display/RAM/Nanoscheduling

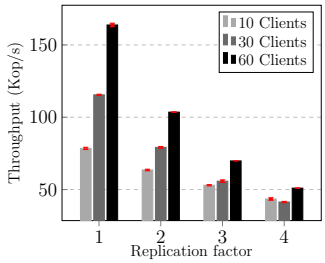
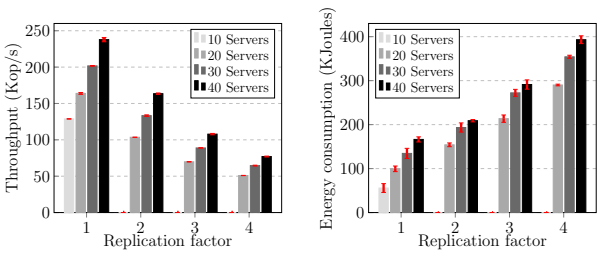


Fig. 5: The total aggregated throughput of 20 RAMCloud servers as a factor of the replication factor.



(a) (b)

Fig. 6: (a) The total aggregated throughput as a function of the number of servers. The number of clients is fixed to 60. (b) The total energy consumption of different RAMCloud servers numbers as a function of the replication factor when running heavy-update with 60 clients.

updates). This workload was preferred to update-only (100% updates) workload since the latter is far from what in-memory storage systems and Web storage systems were designed for [27]. We use the same parameters as in IV-A, except the replication factor that we vary from 1 to 4. We change the number of RAMCloud servers from 10 to 40 and the clients from 10 to 60 nodes.

Replication impact on throughput: In Figure 5 we plot the total aggregated throughput when running heavy-update with different replication factors for 20 RAMCloud servers. When running with 10 clients we can see a clear decrease in throughput whenever the replication factor is increased, e.g., from replication factor 1 to 4 the throughput drops from 78Kop/s to 43Kop/s which corresponds to a 45% decrease. Further increasing the number of clients to 30 and 60 leads to an increase of throughput for replication factor 1 and 2 which means that RAMCloud's cluster capacity has not been reached yet. Ultimately, for 30 and 60 clients setting the replication factor to 4 leads to a throughput of 41Kop/s and 50Kops, respectively, which means that we saturated RAMCloud's cluster capacity.

While the impact of replication on performance might seem substantial, since the normal replication factor at which the system should run with is at least 3 or 4, the explanation lies on the replication scheme itself. In RAMCloud for each update request, a server will generate as many requests as its replication factor, e.g., if the replication factor is set to 4, upon receiving a request a server will generate 4 replication requests to other servers that have available backup service up and running. For every replication request a server sends, it has to wait for the acknowledgements from the backups before answering the client that issued the original request. This is crucial for providing strong consistency guarantees. Indeed, suppose a server can answer a client's request as soon as it has processed it internally and sent the replication requests without waiting for acknowledgements, then in case of a failure of that specific server, the RAMCloud cluster can end up with different values of the same data.

We plot in Figure 6a the aggregated throughput when running heavy-update at fixed rate of 60 clients. We varied the number of servers as well as the replication factor. It is interesting to see how enlarging the number of servers can reduce the load, e.g., when having replication factor set to 1 the throughput can be increased from 128Kop/s to 237Kop/s when scaling up the cluster from 10 to 40 servers. We remark the same behaviour for higher replication factors, though the throughput is lower when increasing the replication factor. It is noteworthy that the figure does not include values for 10 servers when going beyond replication factor 2. The reason is that the experiments were always crashing despite the number of runs because of excessive timeouts in answering requests.

Replication impact on energy consumption: To give a general view on the power consumption under different replication factors we plot figure 7 that shows the average power consumption per node of a cluster of 40 servers when running 60 clients for different replication factors. As expected the lowest average power is achieved with a replication factor of 1 is the lowest with an average power of 103Watts per server. Increasing the replication factor leads to an increase up to 115Watts per server for a replication factor of 4.

The rise in the power consumption per server results in a substantial increase in the total energy consumption of the RAMCloud cluster. Figure 6b illustrates the total energy consumption when fixing the number of clients to 60. Firstly, it is interesting to look at the difference in the total energy consumption when increasing the replication factor. For instance, with 20 RAMCloud servers and replication factor set to 1 the total energy consumed is 81K Joules. It rises up to 285K Joules which corresponds to an increase of 351%. For 40 servers the extra energy consumed reaches 345% whenever tuning the replication factor from 1 to 4. Secondly, it is noteworthy to compare the energy consumption when resizing the number of servers. As an example, with a replication

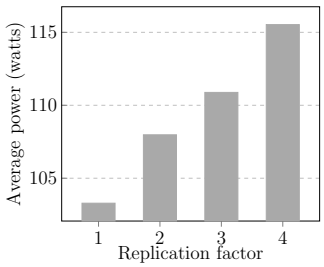


Fig. 7: The average power consumption per node of a cluster of 40 servers when fixing the number of clients to 60.

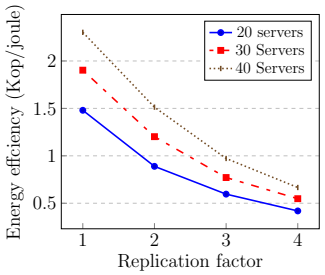
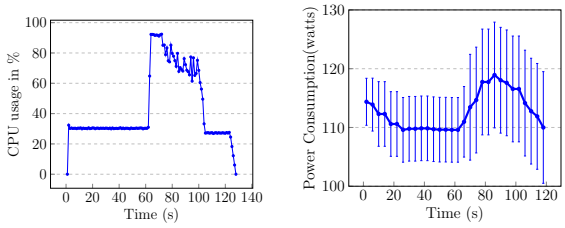


Fig. 8: The energy efficiency of different configurations as a function of the replication factor when running heavy-update with 60 clients



(a) (b)

Fig. 9: The average CPU usage (a) and power consumption (b) of 10 (idle) servers before, during, and after crash-recovery. At 60 seconds a random server is killed.

factor of 1, the difference between the energy consumed by 20 and 30 servers is 16%. The increase in the energy consumed when scaling the cluster from 20 to 40 servers is 28%. When increasing the replication factor these ratios tend to decrease. More specifically, when the replication factor is fixed to 2, the difference in the total energy consumed by 20 and 30 servers is 10%, but this difference reaches 17% only when comparing 20 and 40 servers.

Finding 3: Increasing the replication factor in RAMCloud causes a huge performance degradation and more energy consumption. For instance, changing the replication factor from 1 to 4 leads to up to 68% throughput degradation for update-heavy workloads while it leads to in 3.5x additional total energy consumption. This overhead is due to the CPU contention between replication requests and normal requests at the server level. Waiting for acknowledgements from backups increases the overhead.

Changing the cluster's size. Figure 8 illustrates the energy efficiency when fixing the number of clients to 60. In a very surprising way, we can see that having more servers leads to better energy efficiency. As an illustration, when the replication factor is set to 1 the energy efficiency of 20 servers equals 1500 while for 30 servers it is 1900, finally, for 40 server it reaches 2300. The ratio tends to decrease whenever the replication factor is increasing.

It is noteworthy that the relative differences in the energy-efficiency between different number of servers shrinks when increasing the replication factor. The explanation resides in Figure 6a where we can see that the relative differences in throughput decreases, therefore the fraction throughput/power goes down.

In contrast with the results from Section IV where the best energy efficiency for read-only workloads was achieved with the lowest number of servers, with heavy-update and replication enabled, it appears that provisioning more servers not only achieves better performance, but also leads to a better energy efficiency.

Finding 4: Surprisingly, in contrast with Finding 1, increasing the RAMCloud cluster size results in a better energy efficiency with update-heavy workloads when replication is active. Therefore, the type of workload should be strongly considered when scaling up/down the number of servers to achieve better energy-efficiency.

VII. Crash-recovery

Data availability is as critical as performance during normal operations since there is only a single primary replica in RAMCloud. Studying RAMCloud's crash recovery can help in understanding the factors that need to be considered when deploying a production system, e.g., replication factor, size of data per server, etc. Therefore, our main concern in this section is to answer the following questions: What is the overhead of crash-recovery? What are the main factors impacting this mechanism?

A. Methodology

Two important metrics we consider are recovery time and energy consumption. We perform the assessment by inserting data into a cluster and then killing a randomly picked server (after 60 seconds) to trigger crash-recovery, with different number of servers. First, we assess the overhead of a crash-recovery, and then we vary the replication factor to observe the impact on the recovery-time and the corresponding energy consumption.

The overhead of crash-recovery. For our first scenario we setup a RAMCloud cluster of 10 servers. We then inserted 10M records which corresponds to 9.7GB of data uniformly split across the servers, i.e., each server receives 1M records. It is noteworthy that we have set the replication factor to 4 which corresponds to the normal replication factor used in production systems [28].

Figure 9a shows the average CPU usage of the cluster. As we already explained in section II, RAMCloud monopolizes one core for its polling mechanism, which results in our case in a 25% CPU usage even when the system is idle. When the crash occurs the CPU usage jumps to 92%, then gradually decreases. This is mainly due to loading data from disks and replaying it at the same time. Figure 9b shows the overall average power consumption per node for the same

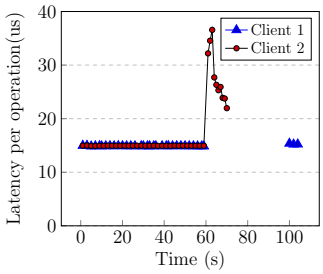


Fig. 10: The latency per operation before, during, and after crash recovery for two client running concurrently. Client 1 requests exclusively the set of data that is on the server that will be killed. Client 2 requests the rest of the data.

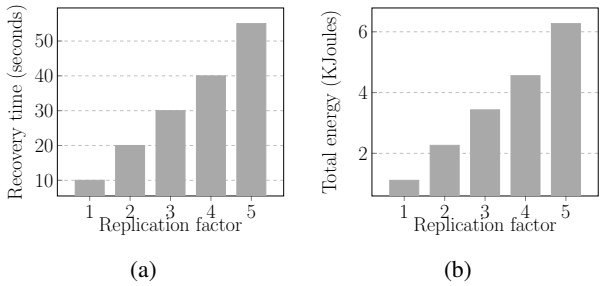


Fig. 11: (a) Recovery time as a function of the replication factor. (b) Average total energy consumption of a single node during crash recovery as a factor of the replication factor. The size of data to recover corresponds to 1.085GB in both cases.

scenario. The jump in resource usage translates in a power consumption of 119W. Since the power measured every second is an average, the increase in power consumption is not as sudden as the CPU usage.

The overhead of crash-recovery on normal operations. Figure 10 shows the average latency of two clients requesting data in parallel in the same configuration as the previous scenario. However, we chose manually a server to kill and make sure one of the clients always requests the subset of data held by that server. It is interesting first to notice that after the crash happens, the client which requests lost data is blocked for the whole duration of crash recovery, i.e., 40 seconds. Second, it is important to point out the jump in latency for the client which requests live data, i.e., from 15us to 35us just after the crash recovery starts. The average increase is between 1.4X and 2.4X in latency during crash recovery. This can be explained if we consider that in figure 9a crash recovery alone causes up to 92% CPU usage. Consequently, normal operation are impacted during crash recovery.

Finding 5: As expected, crash recovery induces CPU (in addition to network and disk) overhead and up to 8% additional power consumption per node. We find that during crash recovery: (1) lost data is unavailable, causing latencies equal to the recovery time (e.g., 40 seconds for replication factor 4); (2) The performance of normal operations can have up to 2.4X additional latency in average due to the overhead of crash recovery.

Replication impact on crash-recovery. Random replication in RAMCloud intends to scatter data across the cluster to have as much resources as possible involved in recovery.

We have setup an experiment with 9 nodes and inserted 10M records which corresponds to 9.765GB of data. Since we are inserting data uniformly in the RAMCloud cluster, each server has up to 1.085GB of data. Same as previous experiment, we choose a random node after 1 min, then we kill RAMCloud process on that node.

Figure 11a shows the recovery time taken to reconstruct data of the crashed node. Surprisingly, increasing the replication factor increases the recovery time. With a replication factor of 1, only 10 seconds are needed to reconstruct data, this number almost grows linearly with the replication factor up to replication factor 5, where the time needed to reconstruct the same amount of data takes 55 seconds.

In order to shed the light on these results it is important to recall in more details how crash recovery works in RAMCloud. When a server is suspected to be crashed, the coordinator will check whether that server truly crashed. If it happens to be the case, the coordinator will schedule a recovery, after checking that the data held by that server is available on backups. Recovery happens on servers selected a-priori. After these steps, the recovery starts, first by reading lost data from backups disks, then replaying that data as in normal insert operations, i.e., inserting in DRAM, replicating it to backup replicas, waiting for acknowledgement and so on. We argue that the performance degradation shown in Finding 3 similarly impacts crash recovery. Since data is re-inserted in the same fashion, increasing the replication factor generates additional overhead and concurrency, leading to higher recovery time.

Impact of disk contention. While investigating the factors impacting the crash recovery performance we find that disk contention can have an impact. For instance figure 12 corresponds to the total aggregated disk activity of the servers during crash recovery. We can see that right after the crash, there is a small increase in the read activity corresponding to reading backup data. Shortly after, there is a huge peak in write activity corresponding to the re-replication of the lost data. Read and write activities continue in parallel until the end of crash recovery.

We think that at small scale this issue is exacerbated. Since the probability of disk-interference between the backup performing a recovery, i.e., reading, and a server replaying data, i.e., writing, is high.

Finding 6: Counterintuitively, increasing the replication factor badly impacts the performance and the energy consumption of crash-recovery in RAMCloud. The replication factor should be carefully tuned according to the cluster size in order to avoid the phenomena discovered in Finding 3 since during a crash-recovery data is replayed.

In Figure 11b we report the total energy consumed by a single node during crash recovery within the precedent experiment. As expected, the energy consumed grows when increasing the replication factor. The energy increases almost linearly with the replication factor. It is noteworthy that the average power consumption of a node is comprised between 114 and 117 watts during recovery. Thus, the main factor leading to increased energy consumption is the additional time took to replay the lost data.

VIII. Related work

Many research efforts have been dedicated to improve the performance of in-memory storage systems. However, very little visibility exists on their energy-efficiency although there have been a lot of works on improving the energy-efficiency of disk-based storage systems.

In-memory storage systems More and more companies are adopting in-memory storage systems. *Facebook* leveraged *Memcached* to build a distributed in-memory key-value store. Twitter used *Redis* and scaled it to 105TB of RAM. Alternatively, a lot of work has been proposed pushing the limits of performance of storage systems to a next level. As an example, *MemC3* [23] is an enhancement of Memcached. Through a set of engineering and algorithmic improvements, it outperformed the original implementation by 3x in terms of throughput. *Pilaf* [22] is an in-memory key-value store that takes advantage of RDMAs (Remote Direct Memory Access). Similarly, *FaRM* [5] is a distributed in-memory storage system based on RDMAs with transactional support.

Evaluating storage systems performance Many studies have been conducted to characterize the performance of storage systems and their workloads. In [3] a deep analysis is made on traces from Facebook's Memcached deployment and gives insight about the characteristics of a large scale caching workload. In another dimension, the work in [27] proposes to study the throughput of six storage systems. However, our goal is different since we study features implemented in RAMCloud and relate them to performance and energy-efficiency. It is noteworthy that performance evaluations of other in-memory storage systems corroborate our results [7]. More specifically, for the throughput of read-only and read-write workloads we see the same trends. These studies however do not discuss energy aspects.

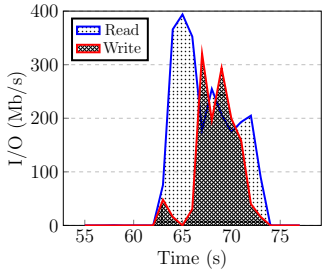


Fig. 12: Total aggregated disk activity (read and write) of 9 nodes during crash recovery. The size of data to recover corresponds to 1.085GB. The dark green part is the overlap between reads and writes.

Energy efficiency Many systems have been proposed to tackle the energy efficiency issue in storage systems. For instance *Rabbit* [16] is a distributed file-system built on top of *HDFS* (Hadoop File System) that tries to turn off the largest possible subset of servers in a cluster to save energy with the least performance decrease. It tries to minimize the number of nodes to be turned-on in case of primary replica failure by putting data in a minimal subset of nodes. In a similar way *Sierra* [15] is a distributed object store that aims at power-proportionality. Through a three-way replication, it allows servers to be powered-down or put in stand-by in times of low I/O activity. This is made possible through a predictive model that observes daily I/O activity and schedules power-up or power-down operations. In [29], authors propose a placement strategy for hot and cold objects in DRAM and NVRAM respectively. On the hardware level, *MemScale* [30] proposes to scale down DRAM frequency to save energy. In [31] authors explore the energy-consistency trade-off and reveal that energy can vary considerably according to the level of consistency. In contrast with these works, we aim at characterizing the energy footprint of a set of features implemented in the RAMCloud storage system.

IX. Discussion

From our study, a set of questions arises on how to mitigate the bottlenecks we identified, more precisely:

A. RAMCloud-centric optimizations

How to choose the right cluster size? By adapting the number of servers we can make substantial energy savings. However, this is not trivial since it can depend on the configuration and the workload as we have shown in our study, e.g., in terms of energy-efficiency it is better to have less servers for read-only workloads, while it is better to have more servers with heavy-updates with replication.

Therefore, a smart approach can be considered at the coordinator level for example, which can decide whether to add or remove nodes depending on the workload. These types of approaches have shown their effectiveness in Cloud environments [15], [16].

Adapting the degree of concurrency? The concurrency level, i.e., number of servicing threads can play a role in performance. Sometimes having more threads than needed can lead to useless context switching, therefore, degrading performance. A solution can be to empirically define the number of threads at the server level. The challenge however is that it depends on the hardware configuration and on the workloads.

Faster data reconstruction? In our deployments, we find it is better to have a lower replication factor for availability, i.e., this is a trade-off between availability and durability. While increasing the replication factor decreases the probability of data loss, it will increase the time during which the system is unavailable during crash recovery.

Another way of getting more backups involved in recovery is to tune the segment size (see II-B). For instance, 1GB of data is equivalent to 128 and 1024 segments of 8MB and

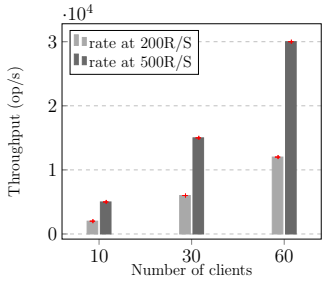


Fig. 13: Aggregated throughput while varying the number of clients (running update-heavy workload) for a cluster of 10 servers with a replication factor of 2. The rate at which a client requests data is limited.

1MB, respectively. However, while tuning the segment size from 1MB to 32MB we find that 8MB, as hard-coded in RAMCloud, gives the best recovery times with our machines. With machines equipped with SSDs smaller values can be chosen to have more parallelism during crash recovery [8].

Request throttling? In a resource-limited environment, it can be better to reduce the request rate rather than saturating the system. Throttling requests at the client level turns out to be a solution to avoid overall system performance degradation, e.g., at Facebook, Memcached clients have back-off mechanisms [1]. For instance, in figure 13, while limiting the throughput at client level we could run the scenario with 10 servers presented in Section VI while avoiding crashes and having linear throughput increase.

However, the challenge would be to identify the threshold at which clients can send without degrading overall performances, which needs to be identified empirically, or using back-off algorithms.

B. General guidelines

Better communication for replication? Replication is used basically to provide availability and durability, however, we have seen that it can have huge overhead due to the trade-off with consistency.
In-memory storage can greatly benefit from network primitives such as RDMAs. It can dramatically reduce the overhead on servers by completely removing the CPU overhead of replication requests [22]. While there are implementations of such a scheme in [5], [32], they rely on a polling mechanism that makes extensive usage of the CPU. In order to achieve a better energy-efficiency it is essential to explore other paths to completely alleviate the load on the CPU. e.g., one-sided RDMA writes.

Tuning the consistency-level? With huge performance and energy overheads that results from guaranteeing strong consistency, it is natural to wonder whether it is more interesting to relax the consistency level. For PBR-based systems, which are prominent in in-memory storage [7], [8], [22], [23], we can think of simply sending the response to the client after an update request, without waiting for the acknowledgement from the backups, if the application tolerates inconsistencies.

X. Conclusion and Future Work

In this paper we propose to characterize the performance and energy consumption of a representative in-memory storage system, namely RAMCloud, to reveal the main factors contributing to performance degradation and energy-inefficiency.

Firstly, we reveal that although RAMCloud scales linearly in throughput for read-only applications, it has a non-proportional power consumption. Mainly because it exhibits the same CPU usage under different levels of access.

Secondly, we show that prevalent Web workloads [3], i.e., read-heavy and update-heavy workloads, can impact significantly the performance and the energy consumption. We relate it to the impact of concurrency, i.e., RAMCloud poorly handles its threads under highly-concurrent accesses.

Thirdly, we show that replication can be a major bottleneck for performance and energy. With update-heavy workloads, it can lead to 68% throughput degradation and 3.5x more energy consumption when increasing the replication factor from 1 to 4. The root cause of this issue is the contention at the server level between clients' requests and replication requests. Moreover, the replication scheme, which implies to wait for acknowledgements from all backups exacerbates this problem.

Finally, we quantify the overhead of a crash-recovery, which can end up in 90% CPU usage and 8% more power consumption on its own. We study the impact of replication on this mechanism and find that, surprisingly, it has a negative effect on it, i.e., increasing the replication factor increases recovery time and the energy consumption.

A natural extension for this work is to consider more workloads in order to cover more aspects of the system. E.g., one could think of scans to assess the indexing mechanism of the system. We consider as well evaluating the system with different request distributions.

We are also considering to evaluate more systems, e.g., Memcached, Redis, etc. It can help in building a broader vision of the energy efficiency in in-memory storage systems.

Finally, we will explore the possibility to mitigate the replication overhead (in terms of performance and energy) as suggested by leveraging RDMAs. Naturally this implied tackling the availability vs durability trade-off under crashes. An interesting aspect to consider then would be correlated failures [33].

Acknowledgment

This work has been supported by the BigStorage project, funded by the European Union under the Marie Sklodowska-Curie Actions (H2020-MSCA-ITN-2014-642963), by the Spanish Government (grant SEV2015-1305 0493 of the Severo Ochoa Program), by the Spanish Ministry of Science and Innovation (contract TIN2015-65316), by Generalitat de Catalunya (contract 2014-SGR-1051), and by the ANR KerStream project (ANR-16- CE25-0014-01). The experiments presented in this paper were carried out using the Grid5000/ALADDIN-G5K experimental testbed, an initiative from the French Ministry of Research through the ACI GRID incentive action, INRIA, CNRS and RENATER and other contributing partners (see http://www.grid5000.fr/ for details).

REFERENCES

[1] R. Nishtala, H. Fugal, S. Grimm, M. Kwiatkowski, H. Lee, H. C. Li, R. McElroy, M. Paleczny, D. Peek, P. Saab, D. Stafford, T. Tung, and V. Venkataramani, "Scaling memcache at facebook," in *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, Lombard, IL, 2013, pp. 385–398.

[2] M. Iravani, "How twitter uses redis to scale - 105tb ram, 39mm qps, 10,000+ instances," 2015. [Online]. Available: https://www.linkedin.com/pulse/how-twitter-uses-redis-scale-105tb-ram-39mm-qps-10000-iravani

[3] B. Atikoglu, Y. Xu, E. Frachtenberg, S. Jiang, and M. Paleczny, "Workload analysis of a large-scale key-value store," in *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems*, ser. SIGMETRICS '12, 2012, pp. 53–64.

[4] J. Ousterhout, A. Gopalan, A. Gupta, A. Kejriwal, C. Lee, B. Montazeri, D. Ongaro, S. J. Park, H. Qin, M. Rosenblum, S. Rumble, R. Stutsman, and S. Yang, "The ramcloud storage system," *ACM Trans. Comput. Syst.*, vol. 33, no. 3, pp. 7:1–7:55, Aug. 2015.

[5] A. Dragojević, D. Narayanan, M. Castro, and O. Hodson, "Farm: Fast remote memory," in *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, Seattle, WA, Apr. 2014, pp. 401–414.

[6] [Online]. Available: https://www.voltdb.com/

[7] H. Lim, D. Han, D. G. Andersen, and M. Kaminsky, "Mica: A holistic approach to fast in-memory key-value storage," in *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, Seattle, WA, Apr. 2014, pp. 429–444.

[8] D. Ongaro, S. M. Rumble, R. Stutsman, J. Ousterhout, and M. Rosenblum, "Fast crash recovery in ramcloud," in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, ser. SOSP '11, 2011, pp. 29–41.

[9] H. Zhang, M. Dong, and H. Chen, "Efficient and available in-memory kv-store with hybrid erasure coding and replication," in *14th USENIX Conference on File and Storage Technologies (FAST 16)*, Santa Clara, CA, Feb. 2016.

[10] C. Lee, S. J. Park, A. Kejriwal, S. Matsushita, and J. Ousterhout, "Implementing linearizability at large scale and low latency," in *Proceedings of the 25th Symposium on Operating Systems Principles*, ser. SOSP '15, 2015, pp. 71–86.

[11] C. Tinnefeld, D. Kossmann, M. Grund, J.-H. Boese, F. Renkes, V. Sikka, and H. Plattner, "Elastic online analytical processing on ramcloud," in *Proceedings of the 16th International Conference on Extending Database Technology*, ser. EDBT '13, 2013, pp. 454–464.

[12] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, and G. Parulkar, "Onos: Towards an open, distributed sdn os," in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '14, 2014, pp. 1–6.

[13] M. Kablan, A. Alsudais, E. Keller, and F. Le, "Stateless network functions: Breaking the tight coupling of state and processing," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, Boston, MA, 2017, pp. 97–112.

[14] J. Blomer and G. Ganis, "Large-scale merging of histograms using distributed in-memory computing," *Journal of Physics: Conference Series*, vol. 664, no. 9, p. 092003, 2015.

[15] E. Thereska, A. Donnelly, and D. Narayanan, "Sierra: Practical power-proportionality for data center storage," in *Proceedings of the Sixth Conference on Computer Systems*, ser. EuroSys '11, 2011, pp. 169–182.

[16] H. Amur, J. Cipar, V. Gupta, G. R. Ganger, M. A. Kozuch, and K. Schwan, "Robust and flexible power-proportional storage," in *Proceedings of the 1st ACM Symposium on Cloud Computing*, ser. SoCC '10, 2010, pp. 217–228.

[17] D. Wong and M. Annavaram, "Knightshift: Scaling the energy proportionality wall through server-level heterogeneity," in *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-45, 2012, pp. 119–130.

[18] D. G. Andersen, J. Franklin, M. Kaminsky, A. Phanishayee, L. Tan, and V. Vasudevan, "Fawn: A fast array of wimpy nodes," in *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles*, ser. SOSP '09, 2009, pp. 1–14.

[19] A. N. Udipi, N. Muralimanohar, N. Chatterjee, R. Balasubramonian, A. Davis, and N. P. Jouppi, "Rethinking dram design and organization for energy-constrained multi-cores," in *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ser. ISCA '10, 2010, pp. 175–186.

[20] [Online]. Available: http://www.nrdc.org/energy/data-center-efficiency-assessment.asp

[21] [Online]. Available: www.grid5000.fr/

[22] C. Mitchell, Y. Geng, and J. Li, "Using one-sided rdma reads to build a fast, cpu-efficient key-value store," in *Presented as part of the 2013 USENIX Annual Technical Conference (USENIX ATC 13)*, San Jose, CA, 2013, pp. 103–114.

[23] B. Fan, D. G. Andersen, and M. Kaminsky, "Memc3: Compact and concurrent memcache with dumber caching and smarter hashing," in *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, Lombard, IL, 2013, pp. 371–384.

[24] Y. Taleb, S. Ibrahim, G. Antoniu, and T. Cortes, "Understanding how the network impacts performance and energy-efficiency in the RAMCloud storage system," Oct. 2016, working paper or preprint. [Online]. Available: https://hal.inria.fr/hal-01376923

[25] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with ycsb," in *Proceedings of the 1st ACM Symposium on Cloud Computing*, ser. SoCC '10, 2010, pp. 143–154.

[26] S. M. Rumble, A. Kejriwal, and J. Ousterhout, "Log-structured memory for dram-based storage," in *Proceedings of the 12th USENIX Conference on File and Storage Technologies (FAST 14)*, Santa Clara, CA, 2014, pp. 1–16.

[27] T. Rabl, S. Gómez-Villamor, M. Sadoghi, V. Muntés-Mulero, H.-A. Jacobsen, and S. Mankovskii, "Solving big data challenges for enterprise application performance management," *Proc. VLDB Endow.*, vol. 5, no. 12, pp. 1724–1735, Aug. 2012.

[28] A. Cidon, S. Rumble, R. Stutsman, S. Katti, J. Ousterhout, and M. Rosenblum, "Copysets: Reducing the frequency of data loss in cloud storage," in *Presented as part of the 2013 USENIX Annual Technical Conference (USENIX ATC 13)*, San Jose, CA, 2013, pp. 37–48.

[29] A. Hassan, H. Vandierendonck, and D. S. Nikolopoulos, "Energy-efficient hybrid dram/nvm main memory," in *2015 International Conference on Parallel Architecture and Compilation (PACT)*, 2015, pp. 492–493.

[30] Q. Deng, D. Meisner, L. Ramos, T. F. Wenisch, and R. Bianchini, "Memscale: Active low-power modes for main memory," *SIGPLAN Not.*, vol. 47, no. 4, pp. 225–238, Mar. 2011.

[31] H. E. Chihoub, S. Ibrahim, Y. Li, G. Antoniu, M. S. Prez, and L. Boug, "Exploring energy-consistency trade-offs in cassandra cloud storage system," in *2015 27th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, 2015, pp. 146–153.

[32] Y. Wang, L. Zhang, J. Tan, M. Li, Y. Gao, X. Guerin, X. Meng, and S. Meng, "Hydradb: A resilient rdma-driven key-value middleware for in-memory cluster computing," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '15, 2015, pp. 22:1–22:11.

[33] J. Li and B. Li, "Beehive: Erasure codes for fixing multiple failures in distributed storage systems," in *7th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 15)*, Santa Clara, CA, 2015.