Memcached, Redis, and Aerospike Key-Value Stores Empirical Comparison

Anthony Anthony
University of Waterloo
200 University Ave W
Waterloo, ON, Canada
+1 (226) 808-9489
a3anthon@uwaterloo.ca

Yaganti Naga Malleswara Rao
University of Waterloo
200 University Ave W
Waterloo, ON, Canada
+1 (226) 505-5900
nmyagant@uwterloo.ca

ABSTRACT
The popularity of NoSQL database and the availability of larger DRAM have generated a great interest in in-memory key-value stores (kv-store) in the recent years. As a consequence, many similar kv-store store projects/products has emerged. Besides the benchmarking results provided by the KV-store developers which are usually tested under their favorable setups and scenario, there are very limited comprehensive resources for users to decide which kv-store to choose given a specific workload/use-case. To provide users with an unbiased and up-to-date insight in selecting in-memory kv-stores, we conduct a study to empirically compare Redis, Memcached and Aerospike on equal ground by trying to keep the default configuration for each database and fair setups (single and cluster mode) and well-rounded workloads (read-heavy, balanced, and write-heavy). We also present our insights not only as performance results and analyses but also as lessons learned and our experience relating to the setup, configuration, and compatibility of some kv-store we have considered.

1. INTRODUCTION
The development of DRAM technology has allowed us to have a memory with a large capacity and relatively cheap price, i.e., The cost for 2x 8GB DIMM DDR3-1600 is 0.0031 USD/Mbyte in 2016 [10]. This advancement of RAM has triggered the NoSQL databases to leverage the fast reads and writes in DRAM to speed up the database query that increases user experience. A study [23] shows that delay in a few hundred milliseconds could lead to potential monetary loss due to bad user experience. Thus, NoSQL key-value stores (A database paradigm used for storing, retrieving and managing Associative arrays/dictionaries) have been increasing in terms of the available choices (e.g. Redis, Memcached, Aerospike, Hazelcast, Voldemort, RiakKV, etc) and widely adopted by wide arrays of software companies. Facebook, Twitter, Zynga, and other companies adopted Memcached [18][19][20]. Github, Weibo, Snapchat, Flicker are among the companies that used Redis [21]. Kayak, Appnexus, Adform chose to use Aerospike [22]. Key-value stores are used for either caching on top of persistent databases or other various other use cases, such as storing web sessions and sharing data among distributed systems.

The key problem is that there is very limited guides or resources that provide the comparison among these key-value stores and most of them are not up to date. Some of the resources are even somewhat published by people affiliated in particular database project. Thus, the results are somewhat biased as the tested DB setup might be set to give more advantage of one of the systems. We will discuss more in Section 8 (Related Work).

In this work, we conduct a thorough experimental evaluation by comparing three major key-value stores nowadays, namely Redis, Memcached, and Aerospike. We first elaborate the databases that we tested in Section 3. Then, the evaluation methodology comprises experimental setups, i.e., single and cluster mode; benchmark setups including the description of YCSB, dataset configurations, types workloads (i.e., read-heavy, balanced, write-heavy), and concurrent access; and evaluation metrics will be discussed in Section 4. The throughput, read latency, write latency, and memory footprint are presented and analyzed in Section 5. Our experiences in choosing the databases, setting them up, and running benchmark are provided in Section 6. Related work and conclusion are presented in Section 7 and 8 respectively.

2. TESTED DATABASES
In this section, we detail the three open-source databases, the reasons for selecting them, and the configuration for each database.

Table 1. Similarities and differences among databases

DB Name	Redis	Memcached	Aerospike
KV-Store Rank	1	2	7
Initial Release	2009	2003	2012
Implementation Language	C	C	C
Number of Supported Languages	33	13	11
Multi-threaded Processing	No	Yes	Yes
Clustering	Yes (2016)	Yes	Yes

Table 1 shows the similarities and differences among the databases. Several essential common characteristics among all the

three databases are that they are (1) able to hold the dataset entirely in memory (RAM); (2) configurable to form a cluster consist of several database instances running in sharding mode; (3) they are on the top 10 (out of 55) list of the most popular key-value stores ranking in December 2016 provided by DB-Engines [1]. Despite the differences in supporting certain features, such as backend storage, supported data types, server-side scripts, replication, consistency, and persistence, the difference in supporting multithreaded query processing between Redis and the other two make it more appealing to compare the databases. In evaluating those databases, we try to disable the configuration related to disk-persistent while keeping the other configurations as default.

2.1 Redis

Redis, started as a one-person project by Salvatore Sanfilippo, is an open source (BSD licensed) key-value store that can be used as a database, cache, and message broker [2]. Compared with Memcached and Aerospike, Redis supports more complex data types including hashes, sets, and sorted sets. When used as a cache, Redis supports six evictions policies. Moreover, it also provides relatively large key and value size up to 512MB which may involve some optimization in their hashing mechanism to maintain good performance. Despite the fact that Redis serialize the data access into a single thread, it is on the first list of the most popular KV-stores [1].

A Redis system consists of Redis server and Redis client. The Redis server can handle multiple client connections concurrently by a wire protocol implemented in the client libraries. To date, there are up to 33 programming languages support Redis client libraries. Redis has supported clustering since version 3.0. released Jan. 2016. In the sharding cluster mode, the client library is responsible for the distributed hashing over the servers. In terms of concurrency control, Redis operations are atomic and no synchronization method needed as a consequence of single thread event loop.

In configuring Redis in our experiment, both in single and cluster mode, we turn off the snapshotting option (save <seconds> <changes>) and we allow connections from other hosts to connect by setting protected-mode off. The Redis version used in the experiment is stable version 3.2.5.

2.2 Memcached

Memcached is another open source (BSD licensed) in-memory key-value store for relatively small data and claimed to be high-performance, distributed caching systems [3]. Being different to Redis, Memcached processed queries / data access in a multi-threaded manner. In more detail, there is a single thread accepting the connections and creates worker threads that run it own event loop and handle its own clients.

The design principle of Memcached differ from that of Redis, it adheres to the concept of keeping the data types and commands simple. Thus, other data types need to be pre-processed or serialized to string or binary prior to storing. There are less complex commands compared with those of Redis; the reason is that all commands in Memcached are implemented to be fast and lock-friendly to give a near-predictable query speed [4]. Some other drawbacks with Memcached are the max size for the value is 1MB, the max key size is 250 bytes, and it only supports LRU eviction policy when used as a cache layer. Similar with Redis, the client side does the distributed hashing so it knows which server to access for an item. In terms of concurrency control, Memcached guarantees operations are internally atomic.

In our experiment, Memcached version 1.4.13 is used and use the default configuration since it does not have support for persistent and our experiment's setup, will be further described in Section 5, will not exhaust the available memory. In this default configuration, Memcached will create 4 threads to handle clients' requests.

2.3 Aerospike

Aerospike is a distributed flash-optimized in-memory key-value store. It was first known as Citrusleaf 2.0 in August 2012 before the company rebranded [5]. The Aerospike company releases the database in two editions i.e., enterprise edition and community edition (AGPL licensed) that differ in several ways, such as advanced monitoring console, cross datacenter replication, and Fast Restart, Rapid Rebalancing, security and IPV6 supports.

Srinivasan and Bulkowski (2011) states that Aerospike architecture comprises three layers, i.e., Client Layer (library) that tracks node and knows where the data resides in the cluster; Data Distribution Layer that manages cluster communications and handles failover, replication, synchronization, rebalancing, and data migration; Data Storage Layer that stores data in memory and flash memory for fast retrieval. Although Data Storage Layer is optimized for flash memory (SSDs), it can also be configured to store data in memory (RAM). Moreover, the record value size supported is up to 1 MB. Moreover, Aerospike claims a strict guarantee about the read/write atomicity (no stale read).

In setting up Aerospike for our experiment, we keep the default configuration which has the *service-threads*, *transaction-queues*, and *transaction-threads-per-queue* value set to 4 and use multicast mode in heartbeat configuration. We only modify the replication factor to 1 so that it does not replicate the data.

3 EVALUATION METHODOLOGY

3.1 Experimental Setup

In this section, we explain the hardware and software details used in performing our benchmarking.

3.1.1 Single Node

We use Ubuntu 12.04 (Linux 3.2.0-23-generic) machine with a total memory of 16GB and 12 CPU cores powered by AMD Opteron Processor. We choose this configuration because a machine equipped with 16 GB is able to host a database to run a decent workload. This machine is equipped with Broadcom Corporation NetXtreme II BCM5716 Gigabit Ethernet. All the three databases evaluated in this work are installed on this machine in single node configuration. Lastly, there is no other noticeable programs sharing the machine's resources while we do our benchmarking.

3.1.2 Cluster mode

In cluster mode, three machines with the configurations identical to the one mentioned above were set up to work form a cluster. Then, the three tested databases are installed on each machine with the databases' configurations explained Section 3. The machines are connected with each other by network links capable of transferring up to 1Gbits/sec. All the machines are provided and by the University of Waterloo and located at the same data center. Lastly, there is no other noticeable program sharing the machine's resources while we were performing the benchmarking.

3.2 Benchmark Setup

In this subsection, the detailed description of the benchmarking tool, various workloads, and concurrent access scenarios will be explained.

3.2.1 Yahoo! Cloud Serving Benchmark (YCSB)

"The goal of the YCSB project is to develop a framework and common set of workloads for evaluating the performance of different "key-value" and "cloud" serving stores [7]." With this goal of YCSB, it is suitable for us to leverage YCSB in benchmarking the three key-value stores. YCSB implements the client side for a number of databases including Redis, Memcached, and Aerospike. The YCSB program reads a set of predefined modifiable workload files, generates the dataset accordingly, loads the dataset to the corresponding database, runs the operations specified in the workload file, and finally collects the performance for the load and run phase. Figure 1 depicts the architecture of YCSB [9]. It is mainly implemented in Java as many of the DB have Java API. YCSB allows programmers to extend the project to implement a new database interface or modify the current interface if a DB update changes the API.

In our experiment, YCSB version 0.11.0 is used. The YCSB's default DB interfaces are used for all the three databases in both single and cluster modes, except for Redis cluster. The provided Redis interface in YCSB 0.11.0 has not yet supported to communicate with Redis cluster (v3.0 or higher). Hence, the interface is updated to use Jedis (Redis client library for Java) version 2.8.0 instead of version 2.0.0. Minor modification is also performed in accordance with Jedis 2.8.0 API to support clustering. The modification is made public and available in the Github [8].

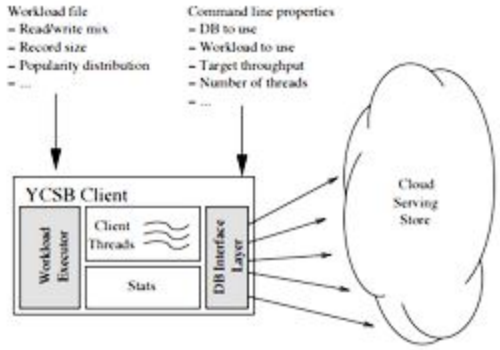

Figure 1. YCSB Client Architecture

In running the YCSB benchmark, there are two phases: the load phase and run phase. In the load phase, YCSB will generate data and send it to the corresponding DB to populate the dataset in the DB. In the run phase, YCSB will perform the operations according to the workload file. More detail about the workload file used during both phases will be discussed in the next subsection.

3.2.2 Datasets

In order to provide a fair and useful insight for the benchmarking, choosing the right combinations of workload is very critical. The combinations might consist of the number of records to be loaded during YCSB load phase, the number of operation to be performed during YCSB run phase, read-write proportion, the size of each record, as well as the distribution of the requests.

In YCSB's configuration, the number of records and operations in the workload file are kept at a constant value of 1,000,000 in evaluating the performance of the three databases across three distinct workloads. The value in each record is further divided into fields. The field count is set to 10 and the field length is set to 800 bytes. As a result, each record has a value size of 8KB and the total dataset is approximately 8GB. The YCSB is also tuned to follow the Uniform distribution in accessing the records during the run phase. In this case, each record has the same probability to be accessed. The three different workloads used to evaluate the databases are explained in the following subsections.

3.2.2.1 Read-heavy Workload

In this workload, the read proportion is set to 0.9 while the write proportion is set to 0.1. Since the operations count is 1,000,000, there are 900,000 read operations and 100,000 write operations performed during the run phase. This workload will enable us to identify which databases are robust with the read-heavy use case.

3.2.2.2 Balanced Workload

To simulate the use case of having the same number of reads and writes, we set the read proportion to 0.5 and the write proportion to 0.5 of the total operations. Hence, the read and write operations are equally 500,000.

3.2.2.3 Write-heavy Workload

The last workload tries to stress the databases with write mostly operation so that we will know which database is more resistant to a high ratio of writes. Thus, the read proportion is 0.05 and the write proportion is 0.95 of the total operations. Hence the read operations are 50,000 and write operations are 950,000.

3.2.3 Concurrent Access

The efficiency of a database in handling concurrency is a key factor in choosing a particular database. Nowadays, since more and more improvements to web-servers and application servers increase the load on the database, benchmarking the database against a various number of concurrent clients is important to give practical results.

We start with 1 client to understand the performance metrics in base case for all databases and continue from 4 to 32 with an increment of 4 at a time. Hence, the number of clients are configured to 1, 4, 8, 12, 16, 20, 24, 28, and 32. We used the YCSB's inbuilt mechanism to measure the performance under multiple concurrent clients by specifying the parameter *threadcount* to tune the number of concurrent threads used to test the database.

3.3 Evaluation Metrics

The comparison among the three databases are based on the following metrics: throughput, read latency, write latency and memory footprint with the variation in workloads as well as the number of concurrent clients. The throughput and latencies are collected in the client side while memory footprint is collected on the database server machine(s).

4 EXPERIMENT RESULTS

4.1 Single Node

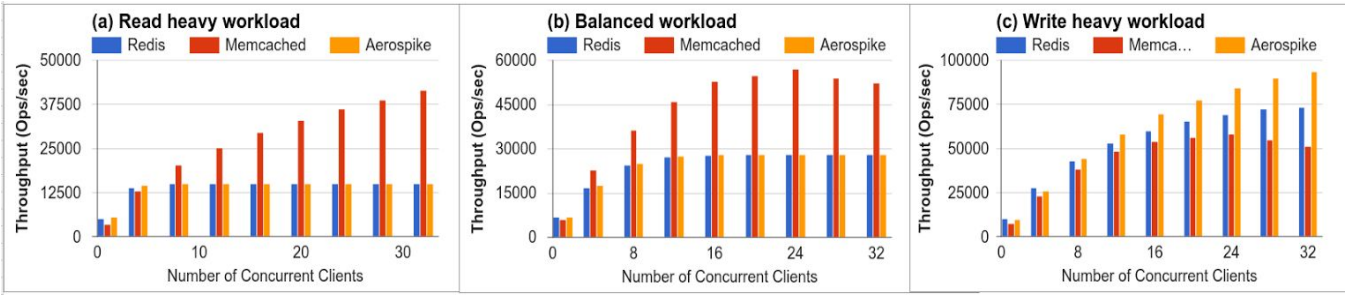

Figure 2 Throughput Analysis in single node mode

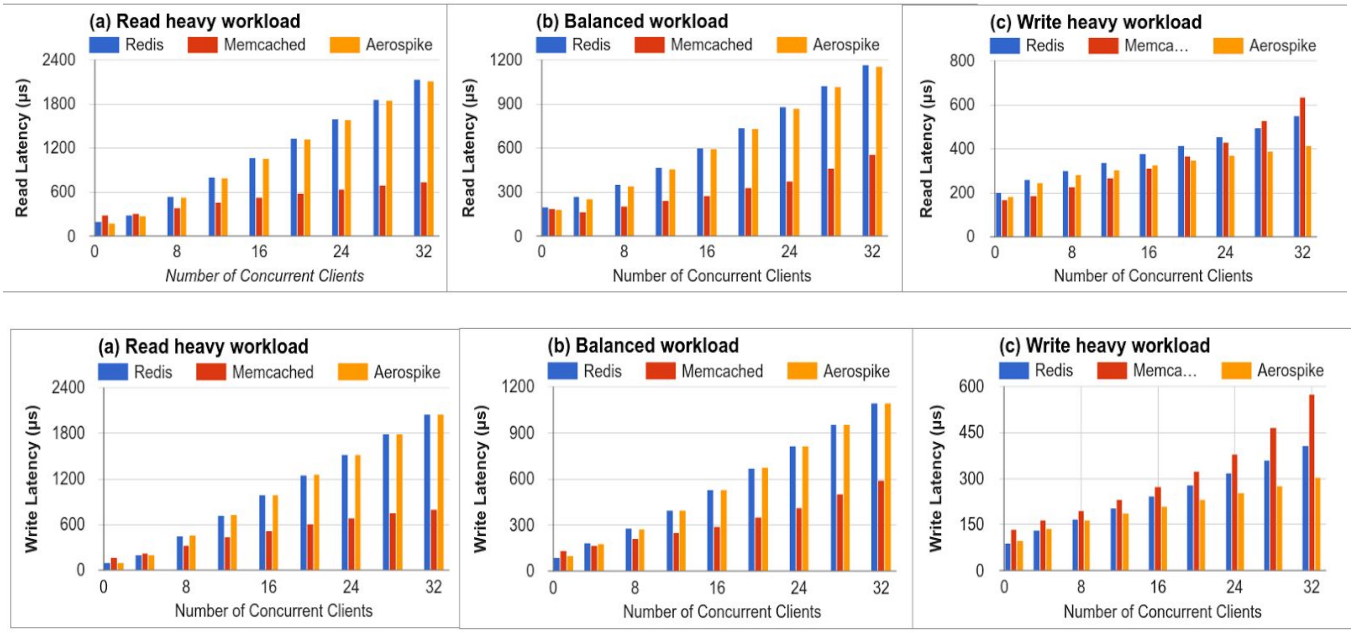

Figure 3 Write Latency Analysis in single node mode

As many production systems start from a single node setup to reduce the upfront investment and complexity in configuring a cluster, it is essential that database have a good performance before it scales to multiple nodes. We use the same evaluation metrics as mentioned in Section 4.3 and run the types of workloads detailed in Section 6

4.1.1 Throughput Analysis

The Figure 2 shows the results explained in this section. In the read-heavy and balanced workload scenarios, Memcached is obviously the leading in terms of throughput with Aerospike and Redis follow behind it. The throughputs of Aerospike and Redis are almost the same in these two workload scenarios as shown in Figure 2(a) and 2(b).

Interestingly in write-heavy workload scenario, the throughput of Memcached is 1.1 - 1.6x less than the average of Redis and Aerospike's throughputs. Aerospike does a better job (up to 19k ops/sec more) than Redis. We attribute this to data storage mechanism that varies among systems. In memcached, the concept of reusing slab is introduced. Redis does not provide detailed explanations on managing the hash-table and storage mechanism on their documentation. Aerospike, on the other hand, has a storage layer designed for optimization when flash memory used as backend storage. Aerospike models the data with the concept of namespaces, sets, and records as delineated in [6].

In the 1 client setting (single node), Redis and Aerospike outperform Memcached across the three workloads. The highest throughput is in the case of the write-heavy workload with Aerospike database with 93,334 operations-per-second. It can be observed that Memcached attains its peak performance at 24 clients for both write heavy and balanced workloads and gradually decreases as more concurrent clients are added.

4.1.2 Latency Analysis

Figure 3 show that Memcached maintains a 0.6 - 2.9x lower read latency compared with the other two databases for read-heavy and balanced workloads. On the other hand, Redis has a slightly higher latency compared with Aerospike. Although Aerospike server is configured to run 4 threads, it is interesting that the performance is not similar to that of Memcached. We attribute this to the Client

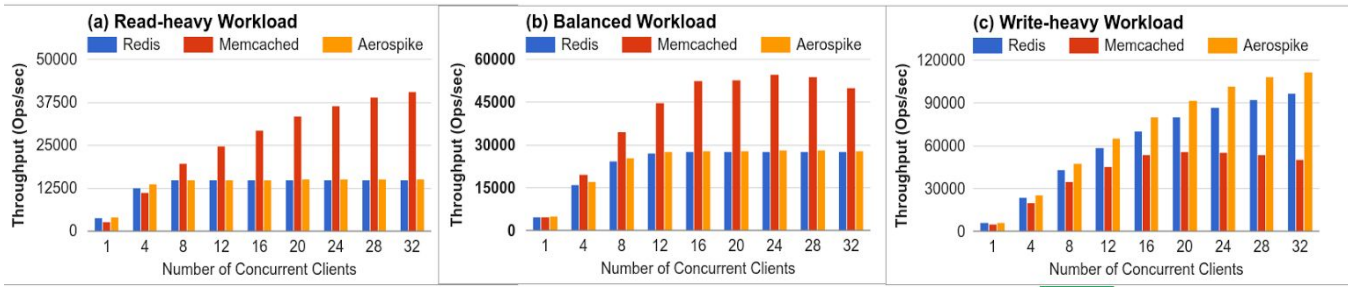

Figure 4: Throughput Analysis in Cluster mode

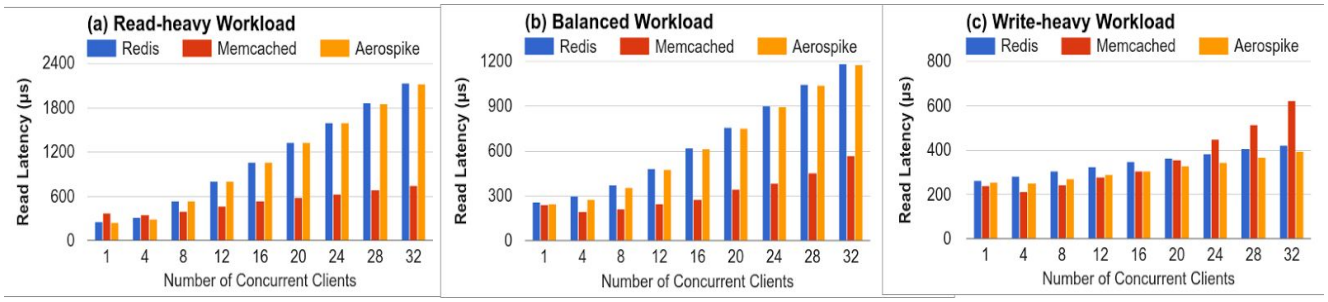

Figure 5 : Read Latency Analysis in Cluster mode

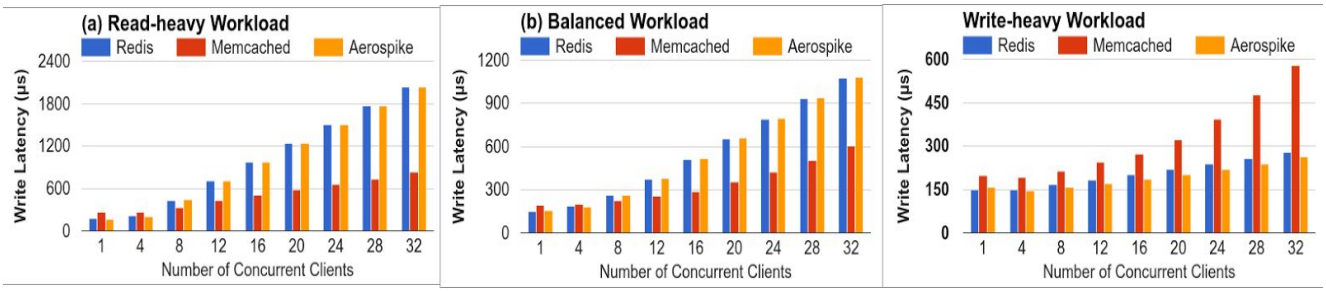

Figure 6 : Write Latency Analysis in Cluster mode

Layer that performs node-check and other more complicated features that Aerospike adds to detect faulty nodes.

The lowest read latency for redis is 195.4 µs that can be observed in 1 client setting in read-only workload. For memcached, the lowest is 170 µs found in 1 client in write-heavy workload. For Aerospike, the lowest read latency is 180 µs also seen in 1 client setting in read-heavy workload. These results suggest that there is significant overhead for the database server to handle and process queries coming from more than 1 client. Lastly, as expected theoretically, Redis's single-threaded design leads to lower performance compared to Memcached that implement multi-threaded event-loops.

Compared with the read latency, the trend in write latency is identical for read-heavy and balanced workloads with Memcached as the lowest latency system after more than 4 clients configuration. In our opinion, it should be attributed to the multi-threaded architecture of the database.

4.2 Cluster Mode

The memory capacity of a single machine could easily run out when it comes to storing the whole dataset into memory. Thus, one of the important reasons for clustering is the ability to partition the dataset into several independent machines without giving up performance or even with speed-up of concurrent processing across machines.

4.2.1 Throughput Analysis

Figure 4 shows the throughput of the three databases under three different workloads as mentioned in Section 3.2.2. Similar to the result in single node scenario, Memcached's throughput is the most noticeable in both read-heavy workload, reaching slightly more than 2.5x that of Redis and Aerospike. We attribute this to the design principle of memcached to only support simple fast commands and does not support complex data types and commands. The fact that each database handles concurrency control uniquely may also lead to different throughputs.

Table 2 : Average Memory Consumption of Each Machine in Cluster mode

	Redis (MB)		Memcached (MB)		Aerospike (MB)	
Workload	Before Run	After Run	Before Run	After Run	Before Run	After Run
Read Heavy	3,356	3,361	2,832	3,082	2,988	2,992
Balanced	3,355	3,360	2,830	3,134	2,990	2,992
Write Heavy	3,356	3,361	2,831	3,135	2,989	2,988

In write-heavy workload, It is also worthwhile to note that while Memcached maintains consistent throughput, interestingly Redis and Aerospike outdo Memcached by 1.2 - 2.2x. Similar to the single node case, we attribute this to different the underlying storage management system.

In cluster setting, Redis achieves its stable throughput around 14.9k ops/sec in read-heavy workload with 16-32 concurrent clients; around 27.7k ops/sec in balanced workload with 16-32 concurrent clients; and about 96,962 ops/sec in write-heavy workload with 32 concurrent clients connected. Memcached gains its best throughput 40,6k ops/sec in read-heavy workload with 32 clients; 54,6k ops/sec in balanced-workload with 24 clients and starts dropping when the number of clients is increased; 56k ops/sec in write-heavy with 20 clients.

4.2.2 Latency Analysis

As shown in Figures 4 and 5, the relationship between throughput and latency of the system is roughly inversely proportional. Hence, similar reasoning can be applied to explain the figures. For instance, Memcached which yields the highest throughput and the lowest latency in read-heavy and balanced workload.

In Figure 4 and 5, case (a) and (b), memcached manages to have significant lower latency in both read and write latencies relatively to the other two systems. The case (c) in Figure 6 shows that write latency of Memcached stands out among the three systems. However, compared with that in the single node, the write latency of Memcached is only slightly lower. Whereas Redis's write latency decreases by up to 130 µs and Aerospike's write latency decreases by up to 40.5 µs.

The above observation shows that memcached is having significantly lower latency in read-heavy and balanced workload and having higher latency in write-heavy latency suggests us to attribute this to the fact that Memcached is designed for caching purpose only. It does not have features related to keeping data persistence or having a secondary persistent storage. Whereas in Aerospike and Redis, they need to take care of replication and consistency issues in the code. This implies that supporting other features might degrade the performance.

4.3 Memory Analysis in Cluster Mode

Understanding the memory used to store the dataset as well as to run the DB server and other bookkeeping works (indexing, hash table, etc) is important before adopting an in-memory database.
The free command is used to measure the amount of memory consumed in our test machines in three states: before YCSB loads the dataset, after the loading and after running the benchmark. We have provided the results in Table 2. The results provided are an average of memory required across the tree machines. It can be seen that Memcached consumes less memory compared to Redis and Aerospike. On the average of each of the three machines in the cluster, Redis consumes about 3,356 MB after loading the database and about 3,651 MB after serving the database. It is the highest among all the databases compared. The smallest memory footprint achieved by Memcached that consumes about 3,000 MB on average at each machine to serve the database

5 EXPERIENCES

5.1 Redis

Redis has a huge user base and a very good documentation policy. It made our job very easy. The Server installation and configuration is pretty straight forward. The default configuration is almost the best configuration for most of the use cases. We were able to find the architecture documents and possible use cases without much difficulties. The use cases are also provided with sample codes on the project's website. It also has a wide range of client libraries for various languages. YCSB uses Jedis, a Java library of Redis.

Finally, Redis has excellent usability because of the presence of strong monitoring capabilities and inbuilt commands which helped create an ecosystem of tools around it.

5.2 Memcached

Installing and running Memcached is very simple. Although we could also build from the provided source code, we chose to install via Advanced Package Tool (apt) repositories. Compared with the other two DBs, the configuration file in Memcached is very short (less than 50 lines). It also has the stats command which returns the statistic regarding memory and storage details.

In terms of the technical documentation, Memcached has a wiki page that encompasses all the resources including the system overview; details of the protocol; configuration for client, server, cluster; and other related documents which are very helpful.

In terms of interoperability, we did not encounter any issue between the client library implemented in YCSB and the latest Memcached server both in single node and cluster setups.

5.3 Aerospike

At the beginning, we thought that Aerospike open source version would not be easy to configure and be limited in the features. However, it turned out that it has a very organized instruction on how to install and what features are different between the enterprise and open source version.

It is properly documented in terms of the details of implementation and architecture. There is also one research paper published in VLDB 2011 by Aerospike's developers in the early stage of the development of the DB [6].

Aerospike also has a very good monitoring console/tool, such as *asadm*, *asinfo*, etc that provide us a good status overview of the running Aerospike server (e.g. number of nodes connected in a cluster, memory usage for each node, the IP address of each node, number of keys , and replication status). Whereas in Redis or memcached, the information related to monitoring or system status are dumped into log files.

5.4 Other Databases Considered

NoSQL databases are the trending database type now. Many new NoSQL databases are developed in the recent years and many of them have grown to rival the well established SQL based databases.

Riak is considered for benchmarking against Memcached and Redis but the performance of Riak is very low in single and cluster mode and hence it is dropped from this benchmarking project. Riak consumes a lot of memory compared to Redis to store the same amount of data and probably slowing down the entire system.

MongoDB is another NoSQL system which has garnered a lot of respect from developers. It is recommended to setup MongoDB with one replica per machine and requires config servers setup along with the sharded database server. Hence the idea to compare MongoDB with Redis and Memcached is dropped to be fair in benchmarking.

Cassandra is categorized under Wide-column store family of NoSQL databases. Although there are certain hacks use this database as a key-value store it is not an ideal choice for a key-value store. Moreover, there is no first class in-memory backed storage engine support making the database slow compared to Redis and Memcached.

6 RELATED WORK

Similar work aiming to examine a number of SQL and NoSQL data store has been done by Rick Cattell in 2011 and published in ACM SIGMOD Record [11]. This paper focuses on examining the databases based on their data model, consistency mechanism, durability guarantee and other dimensions.

Other has tried to instrument throughput of VoldemortDB, Redis, and other DBs in the context of application performance monitoring (APM) for big data. In such context, there are scanning operations involved [12].

A number of benchmarking results against Redis and Memcached have been done previously by different people. Sys/admin [13] released an article in 2010 discussing Redis and Memcached performance that stress the system internals by varying key and value size. Salvatore Sanfilippo (the founder or Redis) released comparison results in 2010 [14] [16] to counter the results released by sys/admin. In his results, he considers multiple clients instead of one single client and also comparing Memcached running with 2 threads on two cores with two Redis servers running on two cores. Dormando (2010) also released the comparison of the two DBs with different client configuration [15]. The results are depending on the configuration and how clients generate the requests.

Another related work that has been done is examining Memcached under multi-threaded access scenarios [17].

7 CONCLUSION

Choosing the proper in-memory key-value stores is becoming more important as they are very helpful in reducing the latency of requests and can be used to store the whole database. To this end, we presented a thorough empirical comparison of three in-memory key-value stores: Redis, Memcached, and Aerospike benchmarked using three different kinds of workloads (read-heavy, balanced, and write-heavy) with a variable number of concurrent clients initiated by YCSB. The benchmark is done in single node and cluster settings.

We observed that Memcached is the best system in the tested key-value stores when it is used as caching layer. In other words, the practical use-cases of in-memory key-value stores are mostly read-heavy and balanced. Thus, based on our experimental results, Memcached yields the best performance in those cases. The performance of Redis and Aerospike is very close to each other, but the memory footprint of Redis is higher. Thus, we nominate Aerospike to be the second best system.

However, if the use-case is mostly single client and requires complex value types as well as longer value and key size, Redis is a better option compared to Memcached and Aerospike.

8 ACKNOWLEDGEMENTS

We would like to thank Professor Khuzaima Daudjee for his guidance and helpful comments and University of Waterloo for providing the infrastructure to benchmark the databases.

9 REFERENCES

[1] DB-Engines. DB-Engines Ranking. Retrieved December 8, 2016 from http://db-engines.com/en/ranking/key-value store

[2] Redis. Introduction to Redis. Retrieved December 8, 2016 from https://redis.io/topics/introduction

[3] Dormando. memcached - a distributed memory object caching system. Retrieved December 8, 2016 from http://www.memcached.org/

[4] What is it Made Up Of?: https://github.com/memcached/memcached/wiki/Overview#what-is-it-made-up-of. Accessed: 2016-12-08.

[5] Aerospike Frequently Asked Questions (FAQ): http://www.aerospike.com/docs/guide/FAQ.html. Accessed: 2016-12-08.

[6] Bulkowski, B., & Srinivasan, V. 2011. Citrusleaf: A Real-Time NoSQL DB which Preserves ACID. PVLDB, 4, 1340-1350. DOI=http://www.vldb.org/pvldb/vol4/p1340-srinivasan.pdf

[7] Yahoo! Cloud Serving Benchmark (YCSB): https://github.com/brianfrankcooper/YCSB/wiki. Accessed: 2016-12-08.

[8] Ycsb-kvstore-comparison: 2016. https://github.com/anthonyaje/ycsb-kvstore-comparison. Accessed: 2016-12-08.

[9] Cooper, B.F. et al. 2010. Benchmarking cloud serving systems with YCSB. Proceedings of the 1st ACM symposium on Cloud computing - SoCC '10. (2010).

[10] Memory Prices (1957-2016): http://www.jcmit.com/memoryprice.htm. Accessed: 2016-12-08.

[11] Cattell, R. 2011. Scalable SQL and NoSQL data stores. ACM SIGMOD Record. 39, 4 (Jun. 2011), 12.

[12] Rabl, T. et al. 2012. Solving big data challenges for enterprise application performance management.

Proceedings of the VLDB Endowment. 5, 12 (Jan. 2012), 1724–1735.

[13] Redis vs Memcached: 2010. https://systoilet.wordpress.com/2010/08/09/redis-vs-memcached. Accessed: 2016-12-08.

[14] On Redis, Memcached, Speed, Benchmarks and The Toilet: 2010 http://oldblog.antirez.com/post/redis-memcached-benchmark.html. Accessed: 2016-12-08.

[15] Redis VS Memcached (slightly better bench): 2010. http://dormando.livejournal.com/525147.html. Accessed: 2016-12-08.

[16] An update on the Memcached/Redis benchmark: 2010. http://oldblog.antirez.com/post/update-on-memcached-redis-benchmark.html. Accessed: 2016-12-08.

[17] Stjepanovic, D. et al. 2015. Performance measurements of some aspects of multi-threaded access to key-value stores. 2015 23rd Telecommunications Forum Telfor (TELFOR). (2015).

[18] twitter/twemcache: 2015. https://github.com/twitter/twemcache. Accessed: 2016-12-08.

[19] How Zynga Survived FarmVille: 2010. https://gigaom.com/2010/06/08/how-zynga-survived-farmville/. Accessed: 2016-12-08.

[20] Who's using Redis?: https://redis.io/topics/whos-using-redis. Accessed: 2016-12-08

[21] NoSQL Examples | NoSQL Use Cases | Aerospike: http://www.aerospike.com/customers/. Accessed: 2016-12-08

[22] Aerospike, Memcached and Redis comparision: http://db-engines.com/en/system/Aerospike;Memcached;Redis. Accessed: 2016-12-08

[23] Marissa Mayer at Web 2.0: 1970 http://glinden.blogspot.ca/2006/11/marissa-mayer-at-web-20.html Accessed: 2016-12-08