ScyllaDB and Samsung NVMe SSDs Accelerate NoSQL Database Performance

February, 2017

Arash Rezaei
Zvika Guz
Vijay Balakrishnan

Please send comments or corrections to {a.rezaei, zvika.guz, vijay.bala}@samsung.com

Table of Contents

1 Executive Summary

We have characterized the performance of ScyllaDB [1] – an open-source, high-performance NoSQL data store. ScyllaDB provides a drop-in replacement for Cassandra, which is a well-known NoSQL data store implemented in Java. However, ScyllaDB has been shown to deliver better performance due to a finely-crafted C++ implementation that manages low-level resources, including memory, thread queues and the network stack.

Here we evaluate the performance of ScyllaDB and Cassandra when serving requests from memory as well as from a first-in-class Samsung NVMe SSD. To this end, we characterize system performance, as well as overall throughput and latency, using different workloads in the Yahoo! Cloud Serving Benchmark (YCSB). This document reports on our findings, and makes the following observations:

- Using the standard YCSB workload, a three-node ScyllaDB cluster with a 2TB dataset can sustain up to 936 KOPS (Kilo Operations Per Second). With a target of sub 0.5 millisecond average latency, it delivers up to 350 KOPS.
- A ScyllaDB cluster, serving 40 percent of the requests from Samsung NVMe SSDs (rather than memory), outperforms an in-memory Cassandra cluster (serving 100 percent of the requests from memory) by 2.28-3x.
- ScyllaDB performance degrades gracefully as more requests are served from the SSD rather than DRAM. Specifically, the performance gap when 40 percent of the requests are served from the SSD is 1.85x to 3.2x (depending on the workload).

2 Introduction

NoSQL databases provide an alternative mechanism for data storage and retrieval compared to tabular relations in relational databases. In general, NoSQL databases sacrifice strong consistency for the sake of scalability, availability and performance, thus providing a better fit to the requirements of modern Web 2.0 applications. Applications from this domain do not require *strict consistency,* and as a result, *eventual consistency* is good enough for their operation.

There are multiple variations of NoSQL databases. A taxonomy based on this data model is presented in Table 1. The classification is based on Yen [2] where data models are column, document, key-value, graph and multi-model. In this document, we focus on Cassandra and ScyllaDB under the column-based data model.

Table 1: Taxonomy and examples of NoSQL databases

Data Model	Examples
Column	Accumulo, Cassandra, Druid, HBase, Vertica, ScyllaDB
Document	CouchDB, Clusterpoint, Couchbase, DocumentDB, IBM Domino, MongoDB
Key-Value	Aerospike, Couchbase, Dynamo, MemcacheDB, Redis, Riak, Berkeley DB
Graph	AllegroGraph, ArangoDB, InfiniteGraph, OrientDB, Virtuoso, Stardog
Multi-model	Alchemy Database, ArangoDB, CortexDB, Couchbase

Cassandra is a widely-adopted NoSQL data store used in many production settings [3]. Several key features have contributed to its popularity:

- Decentralized implementation: Every node in a cluster has the same functionality and role. Since Cassandra does not have a master-slave organization, each cluster has no single point of failure.
- Customizable data replication and data consistency: Cassandra stores multiple data copies. The replication factor and the consistency level can both be adjusted according to specific needs.
- Linear Scalability. Cassandra's throughput (theoretically) scales linearly as more nodes are added to the cluster [4].

Despite Cassandra's high-level architectural merits, several implementation decisions limit its performance and make it relatively hard to configure, as well as to maintain stable performance. ScyllaDB is a Cassandra-compatible NoSQL data store that has been optimized for modern hardware in order to deliver higher performance. While supporting the same API as Cassandra, enabling a drop-in replacement, ScyllaDB takes a radically different design approach. Some of the key features are:

- Direct access to resources: ScyllaDB is implemented in C++ to avoid having an external management software layer (JVM) and its subsequent problems. Conversely, Cassandra is implemented in Java and thus runs inside a JVM instance. Such a design can suffer from sudden latency hiccups, expensive locking and low throughput due to low processor utilization.
- Shared-nothing model: ScyllaDB is built on top of the Seastar framework [5] for extreme performance levels on multi-cores. Since locking and coordination among cores slow down modern servers, Seastar utilizes a lock-less shared-nothing model to completely avoid locks for cross-CPU communications.
- High-performance network I/O: ScyllaDB efficiently utilizes a DPDK driver framework [6], such that multiple TCP/IP instances can run on each core. By running the network stack in user space, DPDK eliminates the need for network system calls, costly context switches and data copying. As the default setting, ScyllaDB uses the Linux TCP/IP stack, which is the setting used in our experiments.

We evaluate the performance of ScyllaDB and Cassandra using Samsung NVMe SSDs. The ScyllaDB results are an indicator to the performance that can be obtained from Samsung NVMe SSDs with a high-performance NoSQL system that makes every effort to improve throughput, latency and scalability. Our Cassandra experiments demonstrate that commodity NoSQL data stores have lower performance even when the whole database fits in DRAM. Consequently, this study shows that NoSQL servers with large amounts of DRAM for holding the entire working set do not necessarily provide better performance, and therein, the efficiency of the software layer becomes a much more important factor. As a result, NVMe SSDs combined with a high-performance software implementation can be substituted for the DRAM.

3 Methodology

We used the YCSB benchmark [7] for performance evaluation, as it was developed to facilitate comparison among cloud OLTP applications. Table 2 presents the workloads in the YCSB package. We utilized workloads A, B, C and D as they best represent real-world applications. The benchmark is executed by the YCSB *client* which is a Java program that runs in two phases: a load phase and a transaction phase.

Table 2: Workloads in YCSB package [7]

Workload	Operations	Application Examples
A – Update Heavy	Read: 50%, Update: 50%	Session store recording recent actions in a user session
B – Ready Heavy	Read: 95%, Update: 5%	Photo tagging: can add a tag in an update, but most operations are to read tags
C – Read Only	Read: 100%	User profile cache, where profiles are constructed elsewhere (e.g., Hadoop)
D – Read Latest	Read: 95%, Insert: 5%	User status updates
E – Short Ranges	Scan: 95%, Insert: 5%	Threaded conversations, where each scan is for the posts in a given thread

We use three servers for the ScyllaDB cluster, and nine machines to run the YCSB clients. Table 3 summarizes our cluster setting. Each server is equipped with four NVMe SSDs having a XFS filesystem organized into a level 0 software RAID. The database is populated with 2TB data, replicated across three servers (i.e., triple replication), with compression disabled.

Table 4 details our Cassandra setting. We made an explicit effort to set up a tuned Cassandra configuration, and the outlined settings provided the best performance. We used Cassandra 3.9 and Java 1.8 with a G1 garbage collector.

Table 3: Description of the hardware/software stack

Cluster Size	3 nodes
Server	Dell PowerEdge R730xd (dual socket)
Processors	2x Xeon E5-2690 v3 @2.6Ghz Per-processor: 12 cores, 24 threads Overall: 24 cores, 48 threads
Memory	256GB ECC DDR4
Network	10 Gigabit Ethernet
NVMe Storage	Samsung PM1725 NVMe [8] Capacity: 3.2 TB Sequential Read: 3100 MB/s Sequential Write: 2000 MB/s Random Read: 750 KIOPS Random Write: 120 KIOPS
	Ubuntu 16.04.1
Linux kernel version	4.4.0-45-generic
ScyllaDB version	1.4.1-20161107.5ef3211

Table 4: Setting for Cassandra servers

Cassandra Version	3.9
Garbage Collector	G1, with the following settings: -XX:+UseG1GC -XX:G1RSetUpdatingPauseTimePercent=5 -XX:MaxGCPauseMillis=500 -XX:InitiatingHeapOccupancyPercent=70 -XX:ParallelGCThreads=48
cassandra.yaml	row_cache_size_in_mb: 10240 buffer_pool_use_heap_if_exhausted: true disk_optimization_strategy: ssd concurrent_compactors: 48

YCSB's usertable	cqlsh:ycsb> describe usertable;
	CREATE TABLE ycsb.usertable (
	   y_id text PRIMARY KEY,
	   field0 text,
	  ...
	   field9 text
	) WITH bloom_filter_fp_chance = 0.01
	   AND caching = {'keys': 'ALL', 'rows_per_partition': 'NONE'}
	   AND comment = ''
	   AND compaction = {'class':
	'org.apache.cassandra.db.compaction.SizeTieredCompaction
	Strategy', 'max_threshold': '32', 'min_threshold': '4'}
	   AND compression = {'enabled': 'false'}
	   AND crc_check_chance = 1.0
	   AND dclocal_read_repair_chance = 0.1
	   AND default_time_to_live = 0
	   AND gc_grace_seconds = 864000
	   AND max_index_interval = 2048
	   AND memtable_flush_period_in_ms = 0
	   AND min_index_interval = 128
	   AND read_repair_chance = 0.0
	   AND speculative_retry = '99PERCENTILE';

4 ScyllaDB Performance

This section covers our ScyllaDB characterization results. First, in Section 4.1 we used two-hour runs to characterize the system in steady state. Then, in Section 4.2 we studied ScyllaDB sensitivity to data location, or more specifically, how performance changes when more (or less) data is served from the SSDs rather than DRAM.

4.1 Steady-State System Characterization

In this section, we focus on performance during steady state (two-hour run), measuring the server, disk and network throughput for different workloads. We used 20 YCSB client processes on every client node with 10 threads per process for a total of $9 \times 20 \times 10 = 1800$ client threads. Table 5 shows the average throughput for ScyllaDB. ScyllaDB achieves 549, 715, 833 and 657 KOPS for workloads A, B, C and D, respectively.

Table 5: Throughput of ScyllaDB with different workloads of YCSB

Workload	Throughput (KOPS)
A	549
B	715
C	833
D	657

Figure 1 depicts the disk and network throughput for various workloads. Each row in Figure 1 corresponds to one workload. Plots on the left show the disk throughput and plots on the right show the network throughput. Each disk's read throughput was high at the start of the run, and then dropped as time progressed. This behavior is due to DRAM caching: after warm-up, most of the working-set fits in the memory, so disk access during the steady state is relatively rare. The maximum read throughputs for the different workloads were 1.8GB/sec, 2.6GB/sec, 2.5GB/sec and 2.1GB/sec per disk (for workloads A, B, C and D, respectively). Since Workload A executes many updates (50 percent of its transactions), the higher write throughput for disk in that case was expected.

The right column in Figure 1 depicts the network throughput, which correlates well with the workload throughput. Workload A has more writes than workload B and its received throughput is 1.4 Gb/sec compared to a throughput of 1 Gb/sec for workload B. On the opposite side, workload A has less reads than workload B and the transferred throughput (2.1Gb/sec) is lower than workload B (3.1Gb/sec). Workload B (95 percent read, 5 percent update) and C (100 percent) have similar network throughput, but the served throughput of B is slightly higher and the transferred throughput of C is also slightly higher. Workload D shows more variations in its network throughput plot.

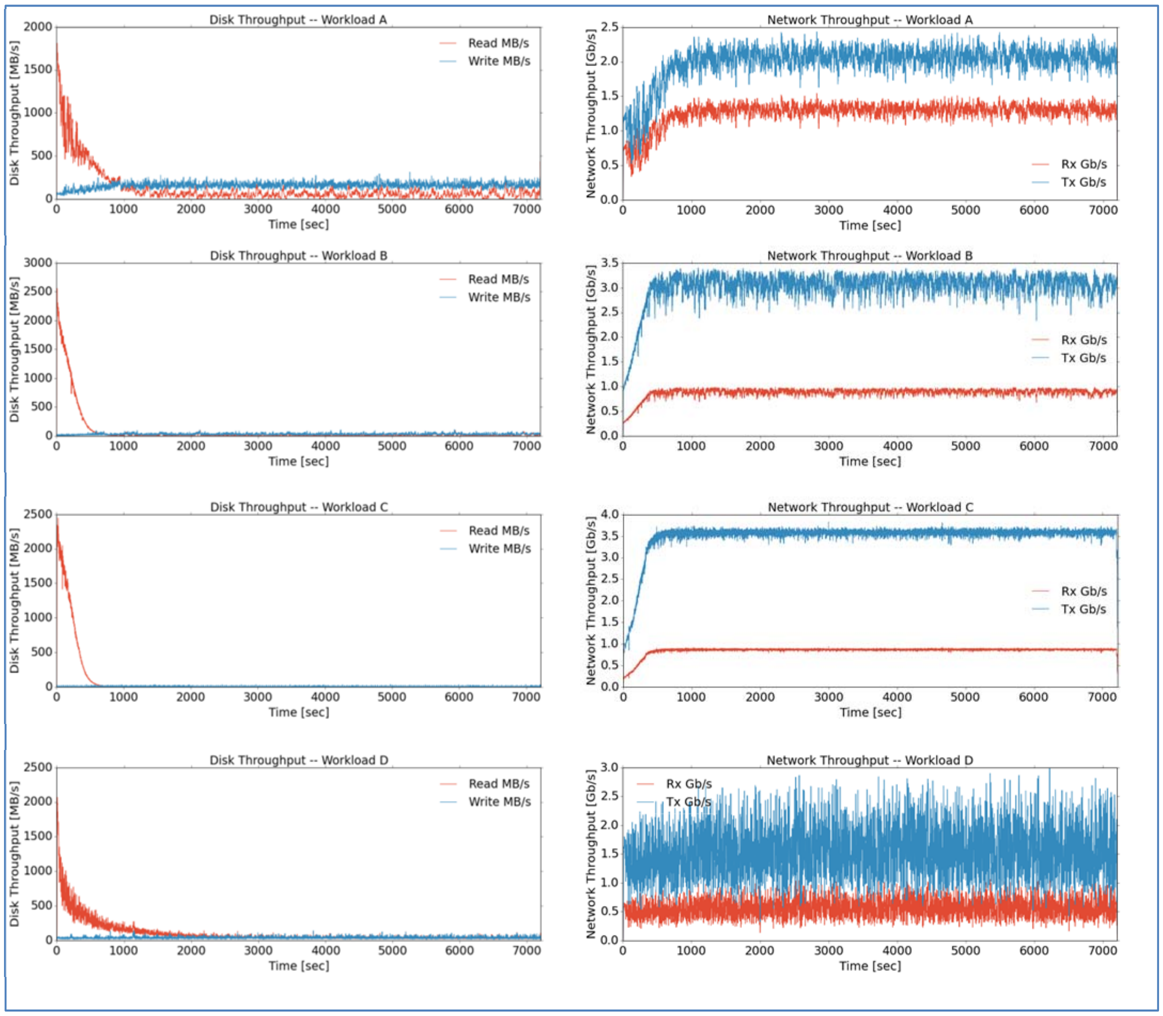

Figure 1: Average throughput of YCSB benchmark during the steady state experiment (2-hour run)

Figure 2 depicts CPU utilization for the same experiments. CPU is mostly saturated, consuming more than 92 percent in all workloads. Workload D has variations in CPU utilization.

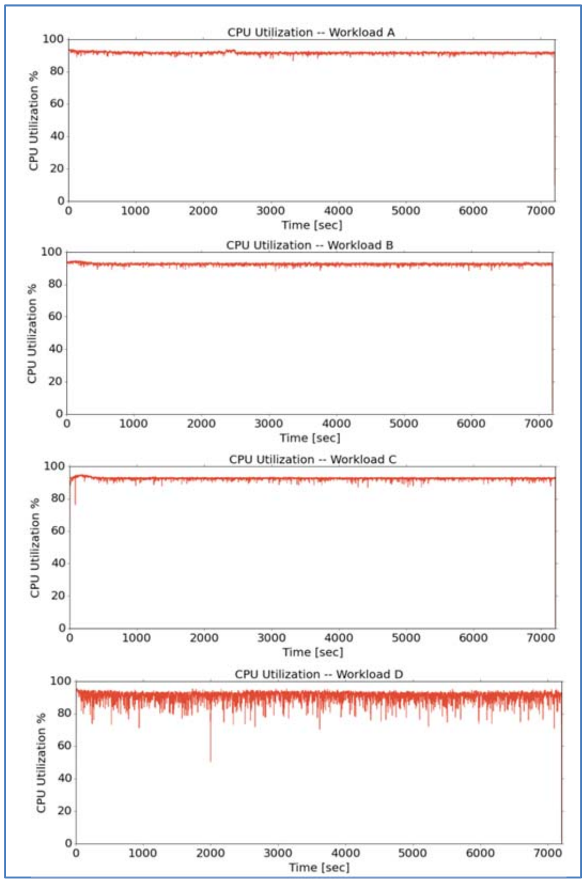

Figure 2: CPU utilization of YCSB benchmark during the steady state experiment (2-hour run)

4.2 Hit Rate Sweep

In this set of experiments, we compared performance sensitivity to data location, or more specifically, how performance changes as more (or less) data is served from SSDs rather than DRAM. To this end, we have created scenarios where a portion of the data resides in DRAM and the rest on the SSD: we repeated the same experiment multiple times, logging the hit rate metric in every experiment (number of requests served from DRAM divided by total number of requests). As we repeated the experiments, the hit rates kept increasing due to data caching. We repeated each experiment nine times and have plotted the results for workloads A, B, C and D.

4.2.1 Throughput

Figure 3 depicts YCSB overall throughput as a function of the memory hit rate. As can be seen, ScyllaDB performance for Workload A in the first run is 290 KOPS. In that run, 40 percent of the requests were served from an SSD (i.e., 0.6 hit rate). In subsequent runs, the hit rate and throughput increased, and finally in the ninth run, the hit rate was 0.988 with a throughput of 539 KOPS. Note that the performance gap between the last and first run is only 1.85x, despite the fact that the last 99 percent of the requests were served from DRAM, while in the first run only 60 percent of the requests were served from DRAM.

Workload B resulted in a 0.65 hit rate with 289 KOPS in the first run, and a 0.9999 hit rate with 725 KOPS in the last run. In this case, the gap was 2.5x. The wider gap for workload B can be explained by the type of operations in each workload. Workload A was 50 percent read and 50 percent update, while workload B was 95 percent read and five percent update. In workload B, there were more reads and consequently more reading from the SSDs, while workload B had more updates that were cached in DRAM and flushed later.

ScyllaDB served workload C with 292 KOPS in the first run (hit rate 0.66) and 936 KOPS in the last run (hit rate 0.99999), which results in a gap of size 3.2x. Results for workload D indicated 291 KOPS in the first run (hit rate 0.81) and 745 KOPS in the last run (hit rate 0.994) with a 2.5x gap. Workload D was similar to workload B in combined operation (here 5 percent *insert* instead of *update*) and both show a 2.5x gap.

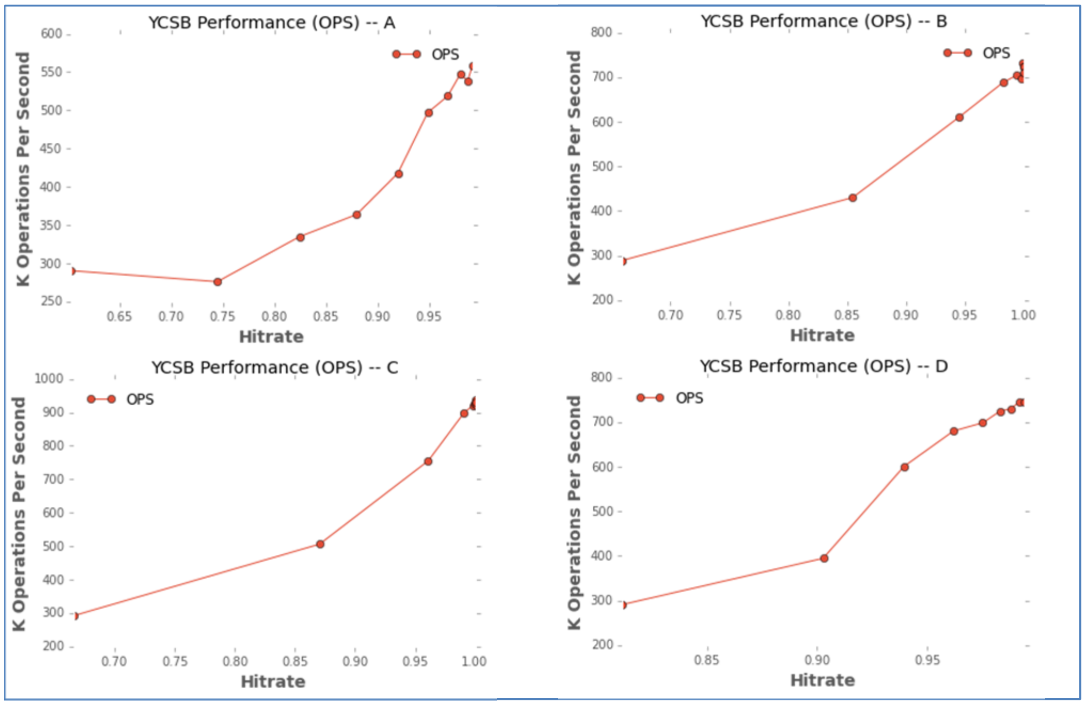

Figure 3: Average throughput of YCSB benchmark with regards to the hit-rate in different runs

Table 6 summarizes the performance gap observed in these experiments. It shows that in workload A even reading 40 percent of data from the SSD leads to only 1.85x lower performance compared to perfectly fitting the working set into the DRAM. From a cost perspective, 40 percent of the DRAM can be replaced by NVMe SSDs and still attain competitive throughput. A similar conclusion can be made about the other workloads.

Table 6: Summary of performance gap (throughput) between first and last runs

Workload	Portion of reads served from SSD in 1st run	Performance gap relative to all in-memory
A	40%	1.85x
B	34%	2.5x
C	33%	3.2x
D	19%	2.5x

4.2.2 Latency

In this section, we examined ScyllaDB latency under medium load, varying the fraction of requests served from storage. We reduced the number of YCSB clients compared to the maximum throughput experiments (see Section 4.1) in order to scale down the load to 50 percent of the server's maximum throughput. We ran five YCSB processes (each having 10 threads) on each node for a total of $9 \times 5 \times 10 = 450$ client threads. For every workload, we restarted the servers to begin with an empty DRAM cache, and then repeated the same experiment nine times (similar to Section 4.1) and reported the hit rate and latency for each.

Figure 4 depicts the latency results for workloads A, B, C and D. The right Y axis shows the throughput as reference points, but the main goal of these experiments was to measure latency. Workload C was served with an average latency of 2.64 milliseconds in the first run (hit rate of 0.58), while in the last run latency was 1.56 milliseconds (hit rate of 0.97). The performance gap between the two was 1.67x. ScyllaDB completed workload B with an average latency of 1.72 milliseconds in the first run (hit rate of 0.64) and an average latency of 0.69 milliseconds in the last run (hit rate of 0.97). This resulted in a performance gap of size 2.49x. Workload C showed latency equal to 1.71 milliseconds in the first run and 0.46 milliseconds in the last run (hit rates of 0.64 and 0.99). The gap in this case was 3.7x. Workload D was completed with an average latency of 1.25 milliseconds in the first run (hit rate of 0.82) and 0.61 milliseconds in the last run (hit rate of 0.98). Thus, the performance gap between the two was 2.04x.

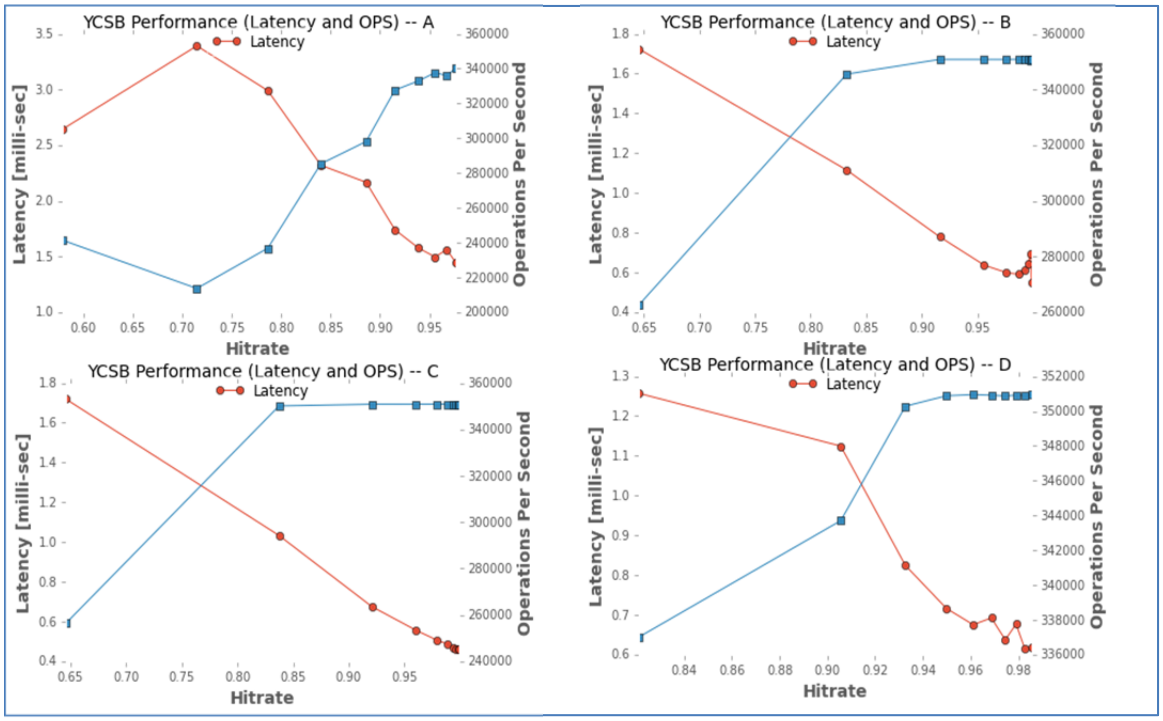

Figure 4: Average latency of YCSB benchmark with regards to hit-rate in different runs (for reference, 2nd axis shows the throughput)

Table 7 summarizes the performance gap (latency) between the first run and last run and also shows the portion of reads that were served from SSDs during the first run. The performance gap varied from 1.67x to 3.7x depending on the benchmark class (type of operations). In general, workloads with less reads showed a narrower gap, while workload C with 100 percent read showed the widest gap.

Table 7: Summary of performance gap (latency) between first and last runs

Workload	Portion of reads served from SSD in 1st run	Performance gap relative to all in-memory
A	42%	1.67x
B	35%	2.49x
C	35%	3.7x
D	18%	2.04x

5 Cassandra Performance Comparison

In this section, we compared the performance of ScyllaDB and Cassandra. Figure 5 shows the average throughput of ScyllaDB and Cassandra, when both systems are set with the same 2TB database and run for two hours. While ScyllaDB achieved 549, 715, 833 and 657 KOPS for workloads A, B, C and D, Cassandra performance varied from 19.3 KOPS for workload B to 63.8 KOPS for workload D –

significantly lower than ScyllaDB. ScyllaDB performance levels were thus 11x, 37x, 20x and 10x compared to Cassandra for workloads A, B, C and D, respectively.

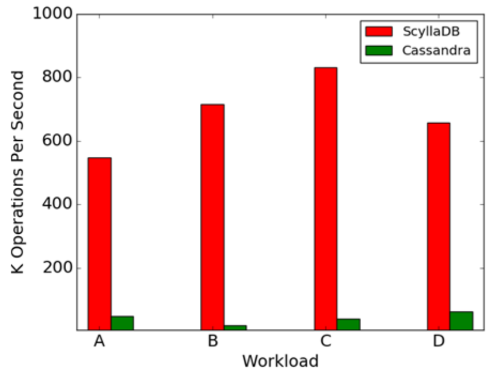

Figure 5: Throughput of ScyllaDB and Cassandra with 2TB data for different YCSB workloads

Next, we reduced Cassandra's database size and measured its performance when the entire database fits in DRAM. Figure 6 depicts Cassandra throughput for database sizes of 50GB, 100GB, 150GB and 200GB (DRAM size is 256GB). Cassandra served workload A with 107-127 KOPS and workload B with 74-95 KOPS. It served workload C, which is 100 percent read, with 69-108 KOPS.

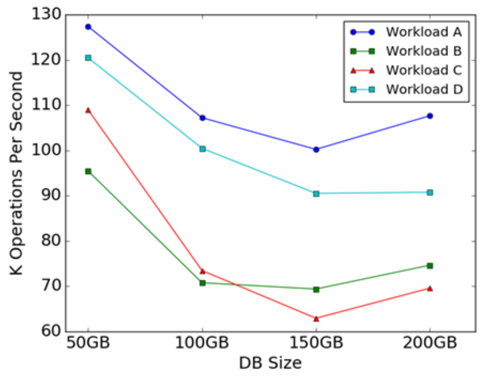

Figure 6: Throughput of Cassandra for YCSB benchmark with different database sizes (All fit in DRAM)

Lastly, Figure 7 depicts the performance of ScyllaDB and Cassandra for three cases: (1) ScyllaDB with 100 percent hit rate, (2) ScyllaDB with 60 percent hit rate, and (3) in-memory Cassandra. For Cassandra, we used the 50GB dataset that provided the best performance (See Figure 6). For ScyllaDB we used the same 2TB dataset as in Section 4.2.

ScyllaDB with a 100% hit rate delivered 558, 709, 930, 745 KOPS, while the in-memory Cassandra system performed at 127, 95, 108, 120 KOPS, for workloads A, B, C and D respectively. The resulting speed-up was 4.39x, 7.46x, 8.61x and 6.2x for these workloads. Moreover, ScyllaDB with a 60 percent hit rate outperformed Cassandra by 2.28x for workload A (290 KOPS compared to 127 KOPS). The result for workload B showed an even wider performance gap between Cassandra and ScyllaDB at 3x (289 KOPS compared to 95 KOPS) while ScyllaDB had only a 64 percent hit ratio. ScyllaDB performance was higher at 2.70x in workload C (292 KOPS compared to 108 KOPS), with a hit rate of 60 percent.

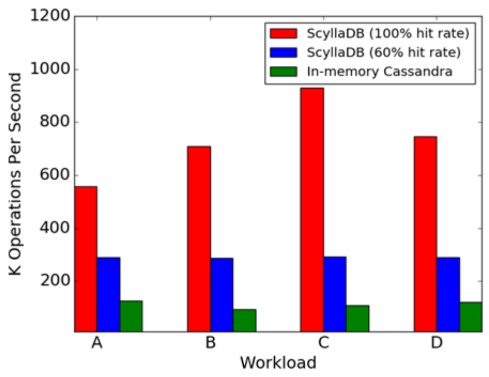

Figure 7: ScyllaDB with 2TB data (2 cases: 100% in DRAM and 60% in DRAM) compared to Cassandra with 50GB data (100% in DRAM)

6 Conclusions

For this document, we evaluated the performance of ScyllaDB, a high-performance NoSQL data store, on Samsung NVMe SSDs, using the YCSB benchmark. Our experiments showed that for a database of 2TB, ScyllaDB provided up to 936 KOPS (workload C) when the entire working set fits into DRAM. We also showed that a ScyllaDB cluster, serving 40 percent of the requests from the NVMe SSDs, outperformed an in-memory Cassandra cluster by 2.28-3x depending on the workload type. Table 8 summarizes the comparative results.

Table 8 : Throughput of Cassandra and ScyllaDB using YCSB benchmark (in 3 scenarios)

Workload	Cassandra 100% in DRAM	ScyllaDB with ~40% in SSD and ~60% in DRAM	ScyllaDB 100% in DRAM
A	127 KOPS	290 KOPS	539 KOPS
B	95 KOPS	289 KOPS	725 KOPS
C	108 KOPS	292 KOPS	936 KOPS
D	120 KOPS	291 KOPS	745 KOPS

7 References

[1] ScyllaDB, "Scylla," ScyllaDB, [Online]. Available: http://www.scylladb.com/.

[2] S. Yen, "NoSQL is a Horseless Carriage," NorthScale, 2014-06-26.

[3] "Cassandra at Twitter Today," Twitter, 2010. [Online]. Available: https://blog.twitter.com/2010/cassandra-twitter-today.

[4] T. Rabl, S. Gomez-Villamor, M. Sadoghi, V. Muntes-Mulero, H.-A. Jacobsen and S. Mankovskii, "Solving big data challenges for enterprise application performance management," in *VLDB*, 2012.

[5] "Seastar," [Online]. Available: http://www.seastar-project.org/.

[6] "Data Plane Development Kit (DPDK)," Intel, [Online]. Available: http://www.intel.com/content/www/us/en/communications/data-plane-development-kit.html.

[7] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan and R. Sears, "Benchmarking Cloud Serving Systems with YCSB," in *1st ACM Symposium on Cloud Computing*, 2010.

[8] "PM1725 NVMe PCIe SSD," Samsung, [Online]. Available: http://www.samsung.com/semiconductor/global/file/insight/2015/11/pm1725-ProdOverview-2015-0.pdf.