FUNGIBLE STORAGE CLUSTER DELIVERS BETTER THAN DAS PERFORMANCE FOR MYSQL

Traditionally, there is a tradeoff between the performance of Direct Attached Storage (DAS) and the agility of network pooled storage for MySQL. That is no longer the case with Fungible Storage Cluster (FSC). You can get the full benefits of pooled storage without suffering the performance bottlenecks with the Fungible Storage Cluster (FSC) serving as the hyperdisaggregated shared storage platform for MySQL. It gives you the efficiency and performance.

MYSQL DATABASE OVERVIEW

MySQL is the world's most popular open source RDBMS database for cost-effectively delivering reliable, high-performance and scalable e-commerce, online transaction processing (OLTP) and embedded database applications. It is a fully integrated transaction-safe, ACID compliant database with full commit, rollback, crash recovery and row level locking capabilities. MySQL delivers the ease of use, scalability, and performance, as well as a full suite of database drivers and visual tools to help developers and DBAs build and manage their MySQL applications. Many of the world's most trafficked websites like Amazon, Netflix, Uber and eBay rely on MySQL for their business-critical applications. MySQL Database includes significant performance and high availability improvements enabling the next generation of web, embedded and Cloud applications.

BENCHMARK TOPOLOGY

To demonstrate MySQL database performance with the Fungible Storage Cluster (FSC), we set up a single MySQL instance attached to a Fungible Storage Cluster via one 100GbE Mellanox CX5 network card running NVMe over TCP (NVMe/TCP). There is a separate server acting as a client running the Yahoo! Cloud Serving Benchmark (YCSB) tool. For comparison with a locally attached NVMe/TCP SSD, the same MySQL server is used to eliminate any hardware discrepancies.

To have a more meaningful MySQL deployment, we added more MySQL instances to the same FSC to make sure there is no performance degradation. Scaling the test environment to 6 MySQL servers, we did not see any performance drop on the same FSC.

Figure 1 below shows the test topology for the two performance tests.

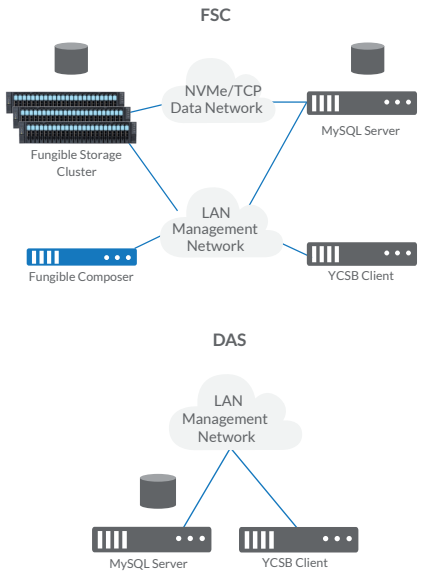

Figure 1: YCSB Benchmark Topology

BENCHMARK RESOURCES

The tables below list all the hardware and software used for the benchmark.

MySQL Server Components	Quantity / Description
Server Type	1 x Supermicro
Memory	256GB
CPU	2x20 Intel(R) Xeon(R) Gold 6138 CPU @ 2.00GHz
Network Card	1 x Mellanox ConnectX-5 – 100GbE 1 x 10GbE
Direct Attached Storage	1 x SATA SSD as Boot Disk 1 x NVMe/TCP SSD (MySQL DB)
Disaggregated Storage	1 x FSC NVMe/TCP volume (MySQL DB)

Table 1: MySQL Server Component

YCSB Client Components	Quantity / Description
Supermicro	1 x Supermicro
Memory	256GB
CPU	2x20 Intel(R) Xeon(R) Gold 6138 CPU @ 2.00GHz
Network Card	1 x 10GbE
Direct Attach Storage	1 x SATA SSD as Boot Disk

Table 2: YCSB Client Component

Software	Description
Operating System	CentOS 8.2
Linux Kernel	4.18.0-193.6.3.el8_2.x86_64
MySQL Server	Version 8.0.17
YCSB	Version 0.17.0

Table 3: Software Resources

Network Component	Purpose / Model
Network Switches	NVMe/TCP / Juniper

Table 4: Network Component

FSC Component	Description
Fungible FSC	2 x FS1600 nodes
FS1600 Network Ports	6 x 100GbE
Fungible Composer	Control Plane

Table 5: Fungible FSC Component

BENCHMARK METHODOLOGY

The FSC allows 2 types of durable volumes, Erasure Coded (EC) and Replicated volume. In this test, we used RF=2 volume and attached it to the MySQL server over NVMe/TCP using a 100GbE Mellanox CX5 network card. We formatted the volume with XFS filesystem and mounted on /var/lib/mysql directory. We created a database named "dbtest" and a table named "usertable". This table was created without compression enabled.

Innodb storage engine was used for this testing and the table below shows the parameters that were set in the /etc/my.cnf configuration file. Please note that we specifically set the innodb_buffer_pool to 16GB because we wanted to force frequent disk I/O since we are testing storage capability. For more information on innodb_buffer_pool_size, you can refer to https://dev.mysql.com/doc/refman/8.0/en/memory-use.html.

```
[mysqld]
default-time-zone='-7:00'
datadir=/var/lib/mysql
socket=/var/lib/mysql/mysql.sock
pid-file=/var/run/mysqld/mysqld.pid
log-error=/var/log/mysql/mysqld.log
skip-log-bin
character_set_server = utf8
collation_server = utf8_general_ci
innodb_log_files_in_group = 4
innodb_log_file_size = 1G
innodb_log_buffer_size = 1049000
innodb_buffer_pool_size = 16G
innodb_buffer_pool_instances = 1
innodb_io_capacity = 1000
innodb_io_capacity_max = 2500
innodb_read_io_threads = 64
innodb_write_io_threads = 64
innodb_flush_method = O_DIRECT_NO_FSYNC
innodb_flush_sync = OFF
max_prepared_stmt_count = 65536
innodb_numa_interleave = 1
max_connections = 1000
sql_mode = NO_ENGINE_SUBSTITUTION
```

Table 6: /etc/my.cnf file

Here is the create table command that was used for the FSC test.

```
CREATE TABLE `usertable` (
 `ycsb_key` varchar(255) NOT NULL,
 `field0` text,
 `field1` text,
 `field2` text,
 `field3` text,
 `field4` text,
 `field5` text,
 `field6` text,
 `field7` text,
 `field8` text,
 `field9` text,
 PRIMARY KEY (`ycsb_key`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
```

Table 7: FSC Table Creation Command

We then loaded the table with 32 million rows with 4KB record size. After the data load completed, we cleared the filesystem cache to avoid any caching and started the YCSB workloadA (50/50) test. The purpose of the test was to run with a targeted transaction per second (TPS) and look at the read/write average latency and tail (99th percentile) latency.

For the direct attached storage (DAS) test, we also formatted the local NVMe/TCP SSD with XFS and mounted on /var/lib/mysql directory. The settings in /etc/my.cnf configuration file remain the same except we created the "usertable" with "zlib" compression. The reason is we want to see the performance difference when using compression on the database server vs. letting the storage do the compression; hence reducing the CPU cycles on the database server so it can spend more time processing SQL statements. The table below shows the command that was used to create a compressed table for DAS testing.

```
CREATE TABLE `usertable` (
 `ycsb_key` varchar(255) NOT NULL,
 `field0` text,
 `field1` text,
 `field2` text,
 `field3` text,
 `field4` text,
 `field5` text,
 `field6` text,
 `field7` text,
 `field8` text,
 `field9` text,
 PRIMARY KEY (`ycsb_key`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COMPRESSION='zlib'
```

Table 8: DAS Table Creation Command

BENCHMARK RESULTS

The graphs below show the tail latency results of MySQL workload A (50/50) using YCSB on the Fungible FSC and direct attached storage (DAS).

With the target of 60k, 80k and 100k transactions per second, the FSC is better than DAS in tail latency for both reads and updates. It is important to note that tail latency is a critical metric for large scale deployed applications because it is the small percentage of response times from a system that takes the longest in comparison to the bulk of its response times.

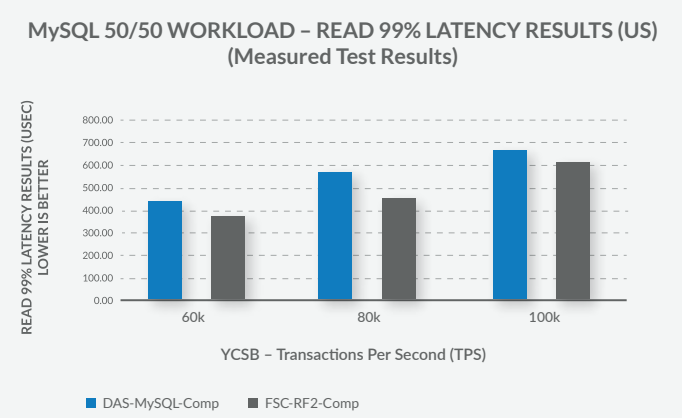

Figure 2: Workload a Read Tail (99%) Latency

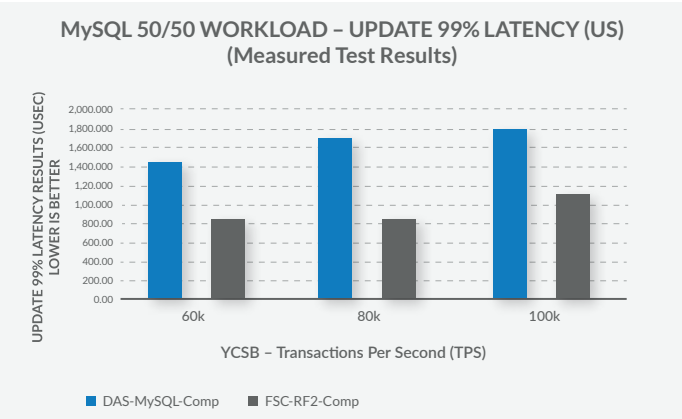

Figure 3: Workload a Update Tail (99%) Latency

KEY TAKEAWAYS

The Fungible FSC provides a highly performant and simple to use NVMe/TCP storage solution for enterprises that may have many RDBMS or NoSQL databases. This solution would replace locally attached storage with network pooled storage to provide centralized storage management, higher utilization, independent scaling of storage and compute, and free up CPU cycles on the server for other workloads. All with better than DAS performance!

It is also a perfect storage solution for a private cloud. Imagine that you want to set up many MySQL VMs for your end users and still want to leverage NVMe/TCP block devices using VirtIO so the VMs live migration can be achieved quicker compared to shared-nothing storage. Below is a sample configuration that a single Fungible FSC can provide many MySQL hosts or KVM hosts that can be used to provide MySQL VMs.

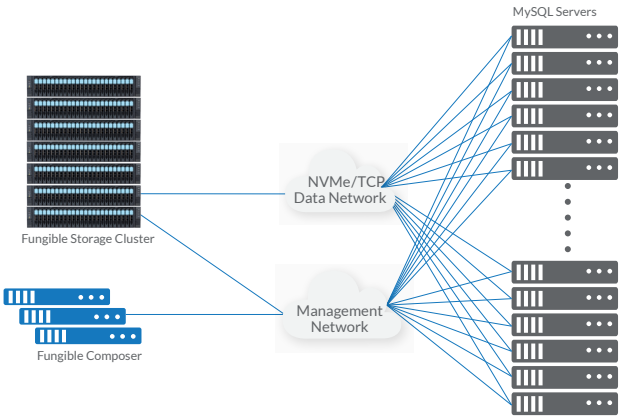

Figure 4: Sample Configuration with Multiple Servers

CONCLUSION

This whitepaper showed that running MySQL database on the Fungible Storage Cluster would give you better tail latency than direct attached storage (DAS). Those results are achieved due to multiple reasons:

- The Fungible Data Processing Unit which was designed to handle data centric computations 10x more efficiently than CPUs.
- The Fungible Storage Cluster software optimizes the data placement on the SSDs
- CPU based software compression has been replaced by DPU based hardware compression.

Though not shown in this whitepaper, it is also the case that the FSC hardware compression is superior to the table compression done by MySQL and will lead to more space savings. In addition, all SSDs will have wear and tear as they are used, especially in a database environment where updates are frequent, running as DAS will increase SSD wear and tear compared to pooled SSDs placed on the FSC.

Customers no longer have to compromise when deploying disaggregated storage for MySQL as with Fungible Storage Cluster they get all the benefits of storage hyperdisaggregation combined with higher performance and lower latency!

ABOUT FUNGIBLE

Silicon Valley-based Fungible is reimagining the performance, economics, reliability, security and agiliy of today's data centers.

CONTACT US

sales@fungible.com

FUNGIBLE, INC.

3201 Scott Blvd., Santa Clara, CA 95054, USA
669-292-5522

www.fungible.com