GridDB Achieves 5 Million Writes Per Second & 60 Million Reads Per Second with only 20 Nodes on Google Cloud

March 20, 2020
Revision 1.00

Table of Contents

Introduction

GridDB is a super fast database for IoT and Big Data applications developed by Toshiba Digital Solutions Corporation. It has a unique key-container data model that is ideal for storing sensor data, a “memory first, storage second” architecture that provides incredible performance and easily scales out to up to 1,000 nodes.

Several reports have been published over the last few years of databases including Cassandra[1][2], Aerospike[3], and Couchbase[4] that were able to sustain one million writes-per-second running on public cloud services. We (Fixstars) and the GridDB team took that as a challenge and decided to see how much further we could scale GridDB and drive down the performance-per-dollar.

While these tests showcase GridDB’s excellent performance, they also showcase best practices when designing a GridDB IoT, IIoT, and other applications that require high velocity write performance.

Cloud Configuration

Google Cloud Platform was used to run the benchmark. We used n1-standard-8 instance types with a 375GB SSD each and with n1-standard-8 instances eight vCPUs and 30GB of memory published per instance pricing was 0.042/hour. A 1:1 ratio of server and client

[1] https://medium.com/netflix-techblog/revisiting-1-million-writes-per-second-c191a84864cc

[2] https://cloudplatform.googleblog.com/2014/03/cassandra-hits-one-million-writes-per-second-on-google-compute-engine.html

[3] https://cloudplatform.googleblog.com/2014/12/aerospike-hits-one-million-writes-Per-Second-with-just-50-Nodes-on-Google-Compute-Engine.html

[4] https://blog.couchbase.com/couchbase-server-hits-1m-writes-with-3b-items-with-50-nodes-on-google-cloud/

instances were used: for example, for 3 servers, 3 clients were used and for 5 servers, 5 clients were used and so on.

From prior experience, we knew that GridDB workloads like we had planned would be I/O bound and that GCP's SSD offered the best performance value ratio. The fast storage combined with GCP's Jupiter[5] networking fabric that offers faster 16 gigabit network egress means that GCP is an ideal platform for GridDB allowing us to achieve near linear scalability from 3 - 20 server instances.

Client Software

A new benchmark client was written in Java with maximum performance in mind. The client would read or write to one container per thread per process eliminating any lock contention.

Like the other databases who have published "1 Million Writes" successes, each record was 200 bytes and 100M records were inserted and read for each trial.

GridDB Configuration

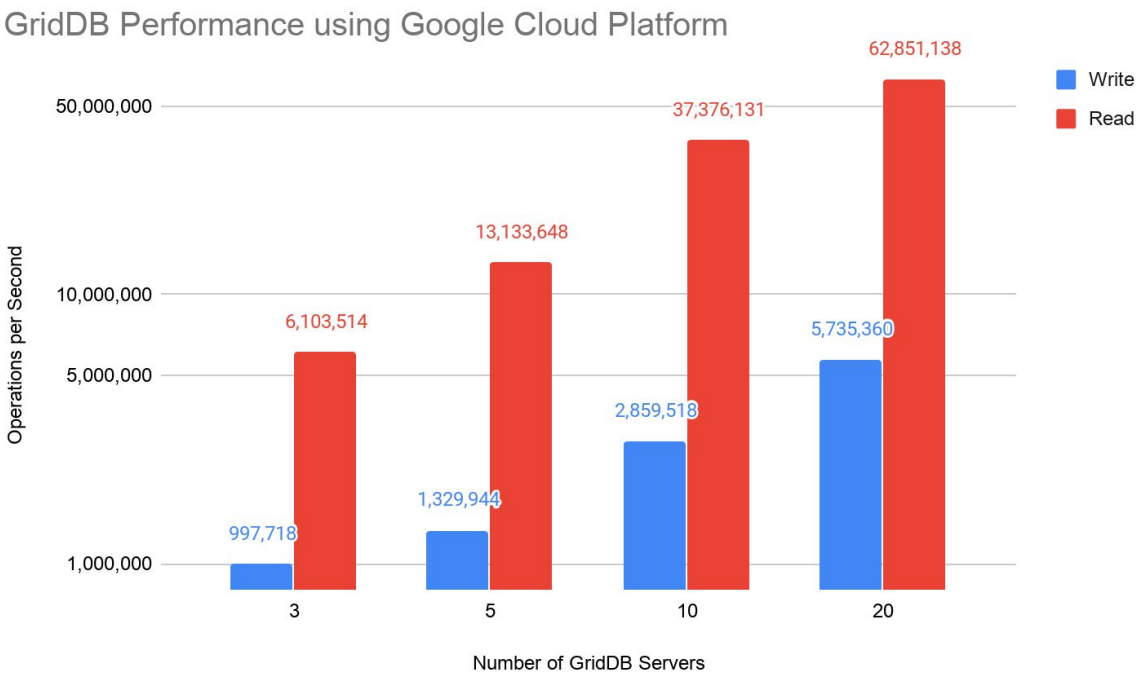
GridDB's configuration was changed from the default parameters with the following settings:

- storeMemoryLimit: 20480MB
- checkpointMemoryLimit: 4096MB
    - Use as much memory for GridDB as possible leaving just enough memory for operating system and other required applications.
- concurrency: 8
    - One worker thread per CPU core.
- storeCompressionMode: COMPRESSION
    - Reduce disk I/O.
- replicationNum: 3
    - Each piece of data is stored on three instances.

Results

With just 3 nodes, GridDB was close to completing 1 million writes per second with 897,718 operations per second. Performance was able to scale linearly as servers were added, with 20 servers being able to achieve our goal of 5 million writes-per-second.

[5] https://cloud.google.com/blog/products/networking/google-cloud-networking-in-depth-how-andromeda-2-2-enables-high-throughput-vms



Three GridDB servers are able to perform just over 6 million reads per second while 20 servers performed over 60 million reads per second.

Operations Per Second

Each GridDB server was able to perform at least 250,000 writes per second and read at least 2 million records per second. This consistent performance makes GridDB ideal for applications where data is ingested continuously at high volumes such as Industry 4.0, financial markets, and both Consumer and Industrial IoT.

Number of Servers	Write Ops/Sec	Read Ops/Sec
3	897,718	6,103,514
5	1,329,944	13,133,648
10	2,859,518	37,376,131
20	5,735,360	62,851,138

Latency

Like YCSB, our benchmark tool doesn't directly measure latency instead it calculates median latency per host using the formula, $latency = 1 / operations\ per\ second$ which is the average time each operation takes. Low latency is imperative for real time analytics,

monitoring, and visualization applications where having the most recent data available ensures the optimal business decisions are made.

		Min Latency	Max Latency	Average Latency
3 Servers	Write	2.55 us	4.05 us	3.34 us
	Read	0.45 us	0.51 us	0.49 us
5 Servers	Write	1.64 us	5.73 us	3.76 us
	Read	0.29 us	0.41 us	0.38 us
10 Servers	Write	0.35 us	4.08 us	1.92 us
	Read	0.03 us	0.29 us	0.15 us
20 Servers	Write	2.33 us	4.85 us	3.49 us
	Read	0.29 us	0.38 us	0.32 us

Cost Metrics

A single server used for these tests costs $0.42 per hour. A small budget of only $1,500 per month is required to achieve 1M writes per second and scale to $6,000 per month for 5M writes per second meaning better cost efficiency with larger workloads. Absolute costs per 100M reads or writes provide longevity as costs per user or device do not escalate as your business grows.

It must be reiterated that 1M writes per second costs only $2.10 per hour while 5M writes per second costs only $8.40 per hour. Reads are even less expensive, with 5M reads per second costing $1.26 per hour and over 50M reads per second costing $8.40 per hour.

Number of Servers	$/hr	$/100M Writes	$/100M Reads
3	$1.26	$0.000390	$0.000057
5	$2.10	$0.000263	$0.000027
10	$4.20	$0.000122	$0.000009
20	$8.40	$0.000061	$0.000006

Conclusion

As expected, GridDB was easily able to achieve 1 million writes per second with four servers and then scaled linearly to 5 million writes per second with 20 servers. Read performance was also very good with over 6 million reads per second with 3 servers and over 60 million reads per second with 20 servers.

These results show the cost effectiveness of GridDB, requiring significantly less computational and storage resources than other databases.

The source code for the utility used to generate the workload in the above results is available http://www.griddb.net/.