GridDB and Cassandra Performance and Scalability.

A YCSB Performance Comparison on Microsoft Azure.

October 31, 2016

Revision 1.2.0

Table of Contents

Executive Summary

With the introduction of Toshiba's GridDB NoSQL database, Fixstars performed benchmarks using YCSB on Microsoft Azure to compare GridDB with one of the leading NoSQL databases: Apache Cassandra. These benchmarks were performed on 1 through 32 node clusters with different total database sizes. These varied conditions hoped to show how the different databases compared across different workload parameters.

The overall conclusions of the performance benchmarks are that GridDB outperformed Cassandra in both throughput and latency, and that GridDB is truly scalable and capable of consistent performance in long-run operations.

Introduction

NoSQL databases were designed to overcome some of the limitations of relational databases and to offer greater scalability, reliability, and flexibility over their predecessors. With emerging technologies such as cloud and mobile computing, the Internet of Things, and ever increasing amounts of data being collected and processed, NoSQL databases are the first choice for today's and tomorrow's applications.

GridDB is a distributed NoSQL database developed by Toshiba that can be operated as an in-memory database or with a hybrid composition. It is fully ACID-compliant (Atomicity, Consistency, Isolation, Durability) at the container level and has a rich set of features. It can also be used as either a Key-Container or TimeSeries database.

Cassandra is a free and open source distributed NoSQL database. Cassandra is considered to have some of the best performance of the major NoSQL databases while maintaining high availability and a decentralized design.

The Yahoo! Cloud Serving Benchmark, or YCSB, is a modular benchmark for NoSQL or Key-store databases that is written in Java.

Environment

Azure Configuration

Three resource groups were created, each with 65 Standard D2 instances in the West-US region. The Standard D2 instances feature two Intel Xeon CPU E5-2673 cores running at 2.40GHz, 7GB of memory, 1Gpbs networking, and a 100GB local SSD. The local SSD was used to store the GridDB and Cassandra's data files while a persistent page blob stored the OS disk.

Each instance was based on the OpenLogic Centos 6.5 Linux image. The first instance contained a public IP address and acted as a headnode while up to 32 instances would be used as NoSQL database servers and an equal number of instances ran YCSB clients.

The headnode was responsible for starting the servers, executing the YCSB clients, collecting resource utilization statistics, and aggregating the results.

Software Versions

GridDB version 3.0 CE was installed into the Azure nodes using RPM packages provided by Toshiba Corporation.

For Cassandra, version 3.4 was installed from Datastax's community YUM repository.

YCSB was cloned from its github repository on July 5, 2016. The Cassandra2 Database driver remained unmodified. Toshiba provided their YCSB GridDB driver in August 2016 and it was modified to use the Fixed List connection method instead of Multicast.

Software Configuration

GridDB

For the most part, GridDB used the default or recommended configuration. Experimentation confirmed that these were the ideal values for the setup. Concurrency was set to 2 to match the number of cores, while checkMemoryLimit was set to 512MB and storeMemoryLimit was set to 6144MB. This configuration allowed plenty of space to keep 4GB of records in memory and allowed approximately 512MB for other system actions.

GridDB used the fixed list method of communicating with other GridDB servers instead of the more typical multicast method because Azure and most other Cloud providers do not support multicast between instances.

The only other change from the default values was setting storeBlockSize to 32KB from 64KB.

Cassandra

As many other benchmarks with Cassandra reported, write timeouts were a problem with Cassandra. To address this, core_workload_insertion_retry_limit was increased to 10 from 0 in the YCSB workload file and read_request_timout_in_ms was increased to 5 seconds while write_request_timeout_in_ms, counter_write_request_timeout_in_ms, and range_request_timeout_in_ms were all increased to 10 seconds.

To reduce cluster start up times, n-1 seeds were used where n is the number of nodes. 1 and 4 seeds were also experimented with but were deemed to have no impact on performance in a single rack/datacenter environment.

Concurrent readers, writers, and were all set to 32 as confirmed by experimentation.

General

For all systems, the maximum number of open files was increased to 64000 via limits.conf.

Test Methodology

Test Design

The goal of the testing was to see how each database performed under a variety of conditions while maintaining similar parameters as other high profile NoSQL benchmarks.

In earlier, smaller scale testing, it was discovered that thread counts between 32 and 192 all produced similar results, but 128 threads was the most consistent. A thread count of 128 was therefore used for all subsequent testing except for Cassandra loads which use 32 threads to prevent Timeout exceptions. Further research has shown that this is fairly common behavior with Cassandra and while the configuration can be modified, best performance is achieved with fewer threads.

It was determined that two data sets would be used, a small data set of 4M records per node, and a larger dataset that would store 12M records per node. Each record would consist of ten 100 byte strings (1Kbyte per record), therefore the per node database size would be 4GB or 12GB respectively. The small data set could fit entirely within memory, while 50% of the large data set would need to be flushed from local memory stores or caches. The transactional workloads would each perform 10M operations per client and would have access to the entire data set. This configuration would give Cassandra sufficient time for JVM warm up and ensure that rows would be both in and out of cache.

Methodology

Due to the inherent inconsistency of running a workload on shared cloud services, each series of tests were run three times in a different resource group and the results shown here are from the individual best throughput for each workload. This decision was made to minimize performance fluctuations caused by Azure as the benchmark's goal was to evaluate GridDB and Cassandra, not Azure.

Starting from a state where all instances are "deallocated", the headnode would first start the required number of instances. Once they are running, deploy configuration files, mount the local SSD, remove any existing database data files, and finally start GridDB or Cassandra via their initscript.

Once the servers have finished starting, server statistics via gs_stat for GridDB or nodetool for Cassandra would be captured and stored for further analysis.

YCSB load would be executed concurrently on all the client nodes with the appropriate insertstart, insertcount, recordcount parameters. After the load completes, workloads are run in the following order according to the YCSB recommendation (see https://github.com/brianfrankcooper/YCSB/wiki/Core-Workloads):

- Workload A -- Update heavy
- Workload B -- Mostly reads
- Workload C -- Read only
- Workload F -- Read, modify, write
- Workload D -- Read latest

Server statistics are once again captured after each workload finishes.

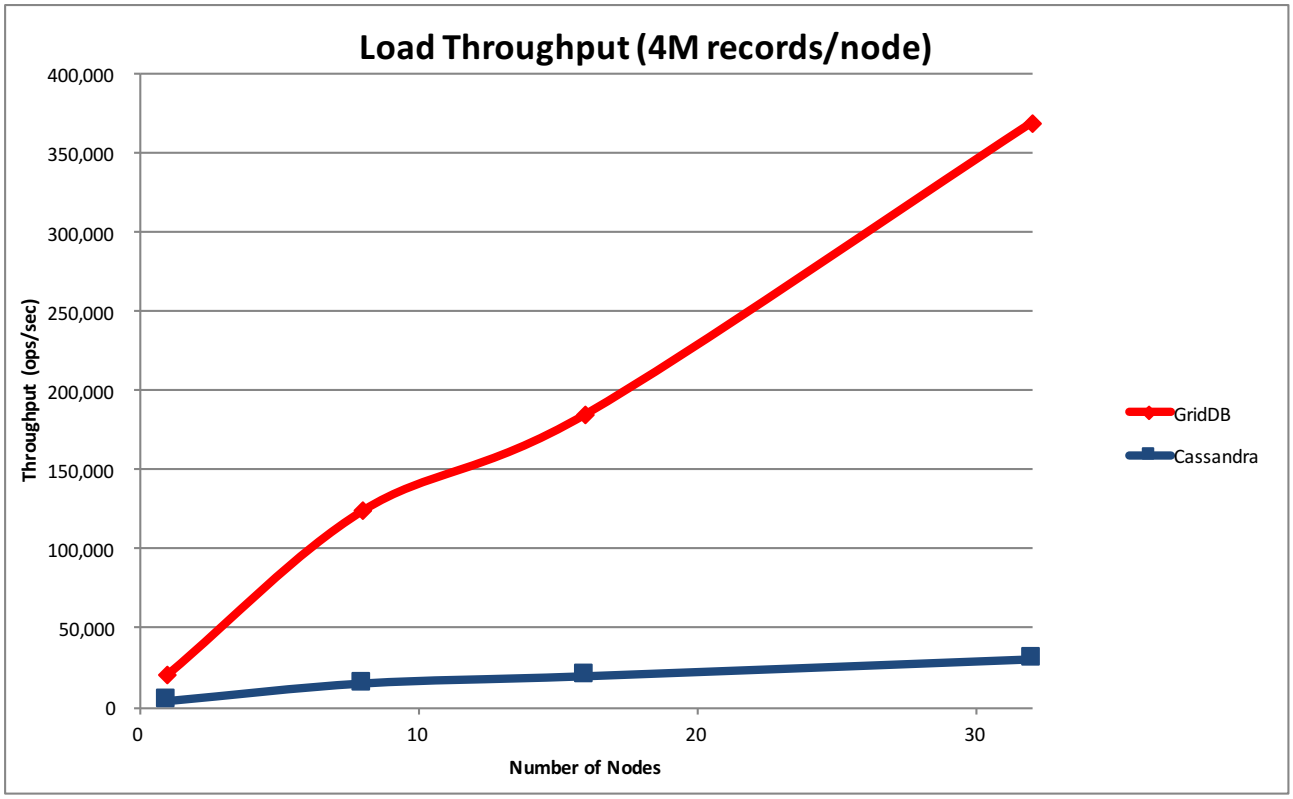
Collection and Aggregation

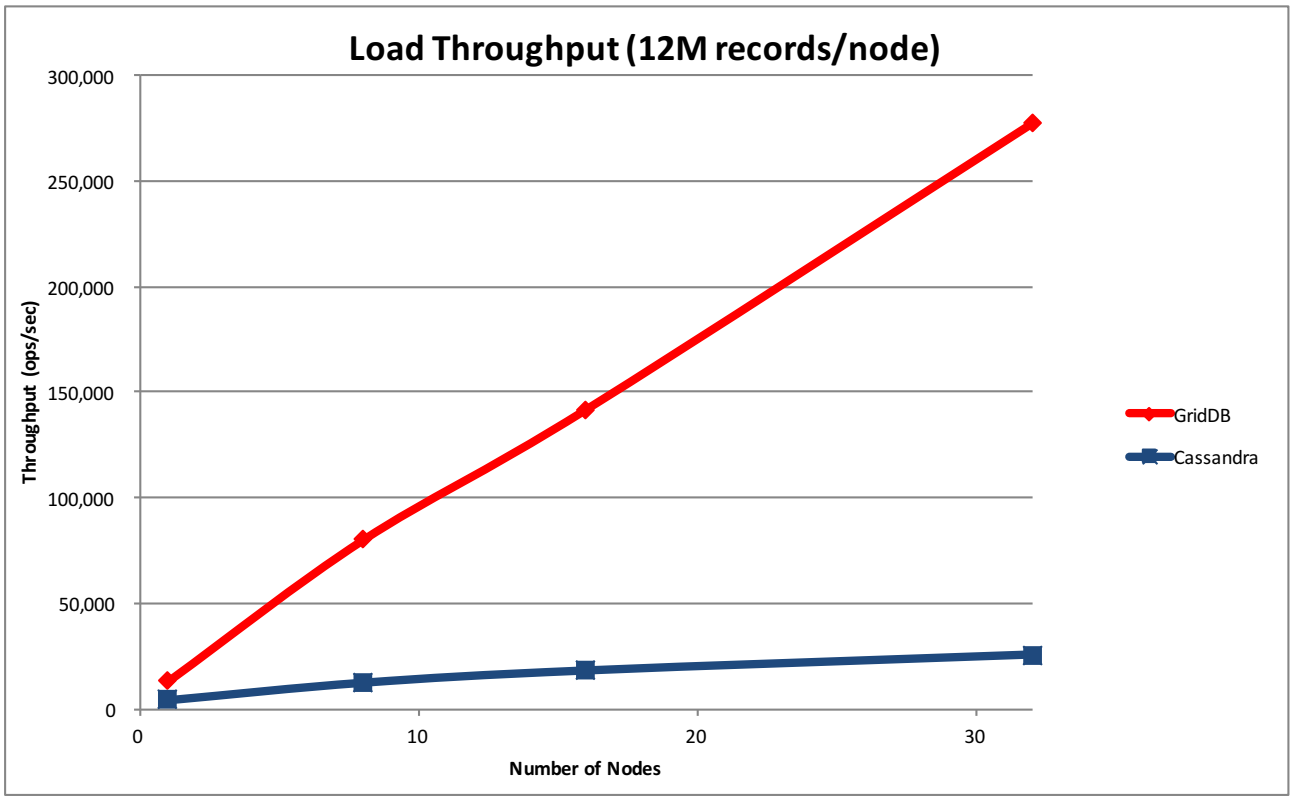
All YCSB output is captured for post processing and result compilation. Simple bash scripts using awk and grep were used to output a single test run per line in CSV format and were then further processed in a spreadsheet.

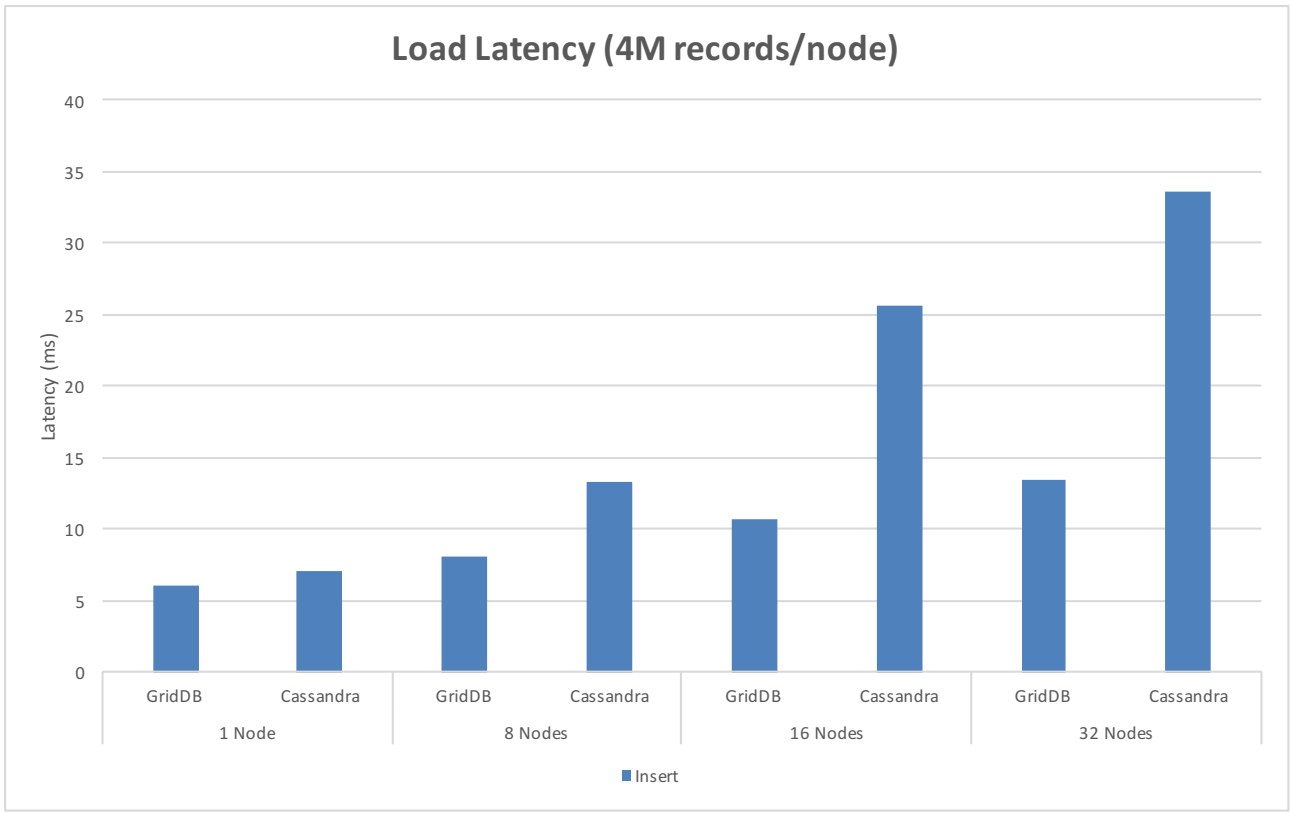
Benchmark Results

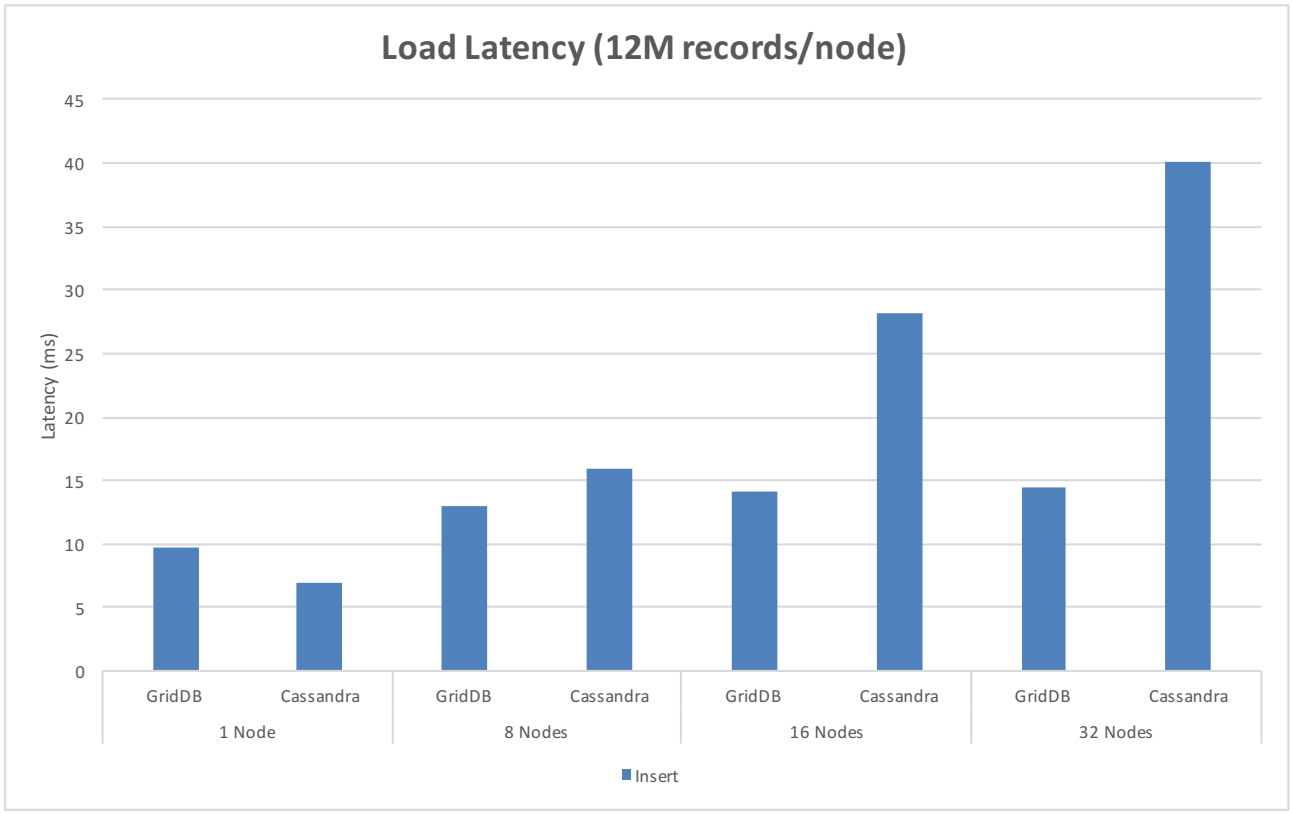
Load

Other Cassandra benchmark reports have reported difficulties in loading data into Cassandra — Fixstars encountered similar issues. Increasing concurrent writers from the recommended settings, extending timeouts by a factor of 10, and reducing the number of threads to 32 corrected all TimeoutExceptions. For throughput, higher is better and for latency, lower is better.



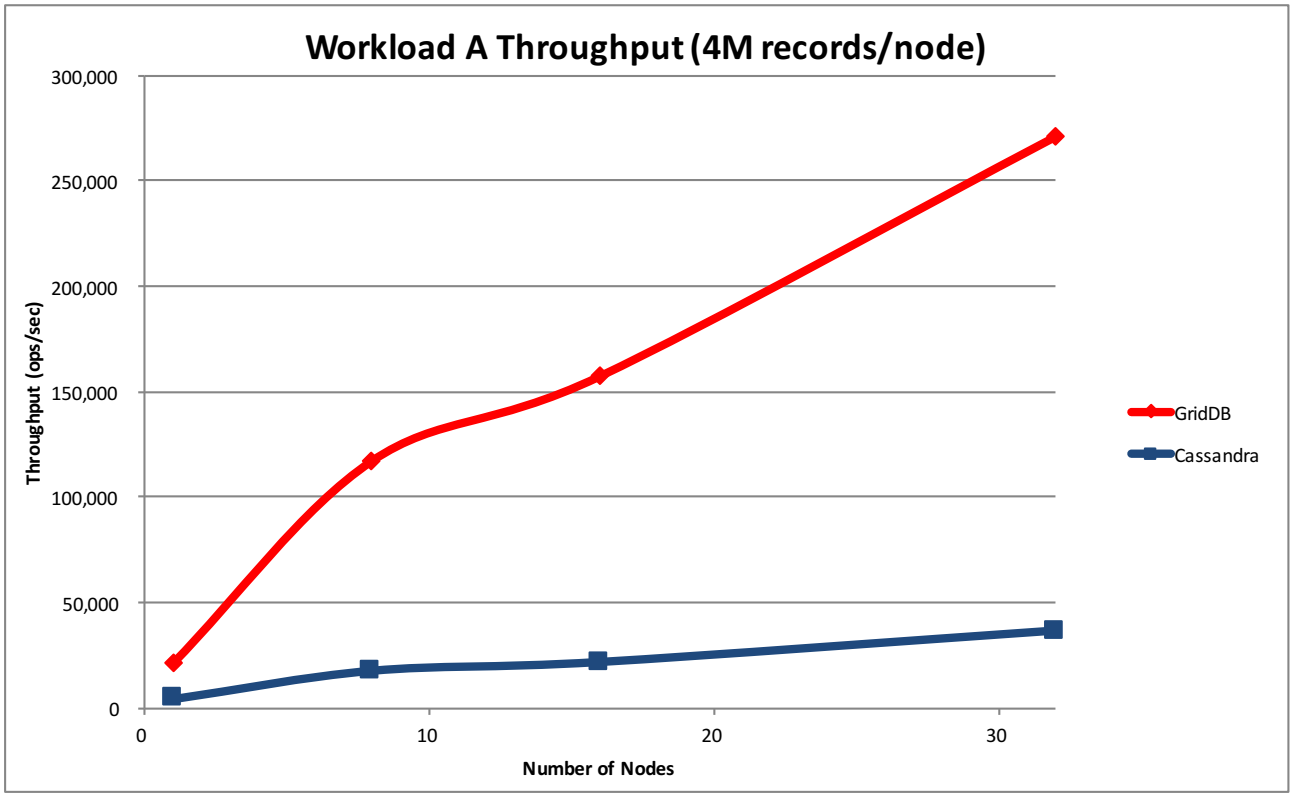


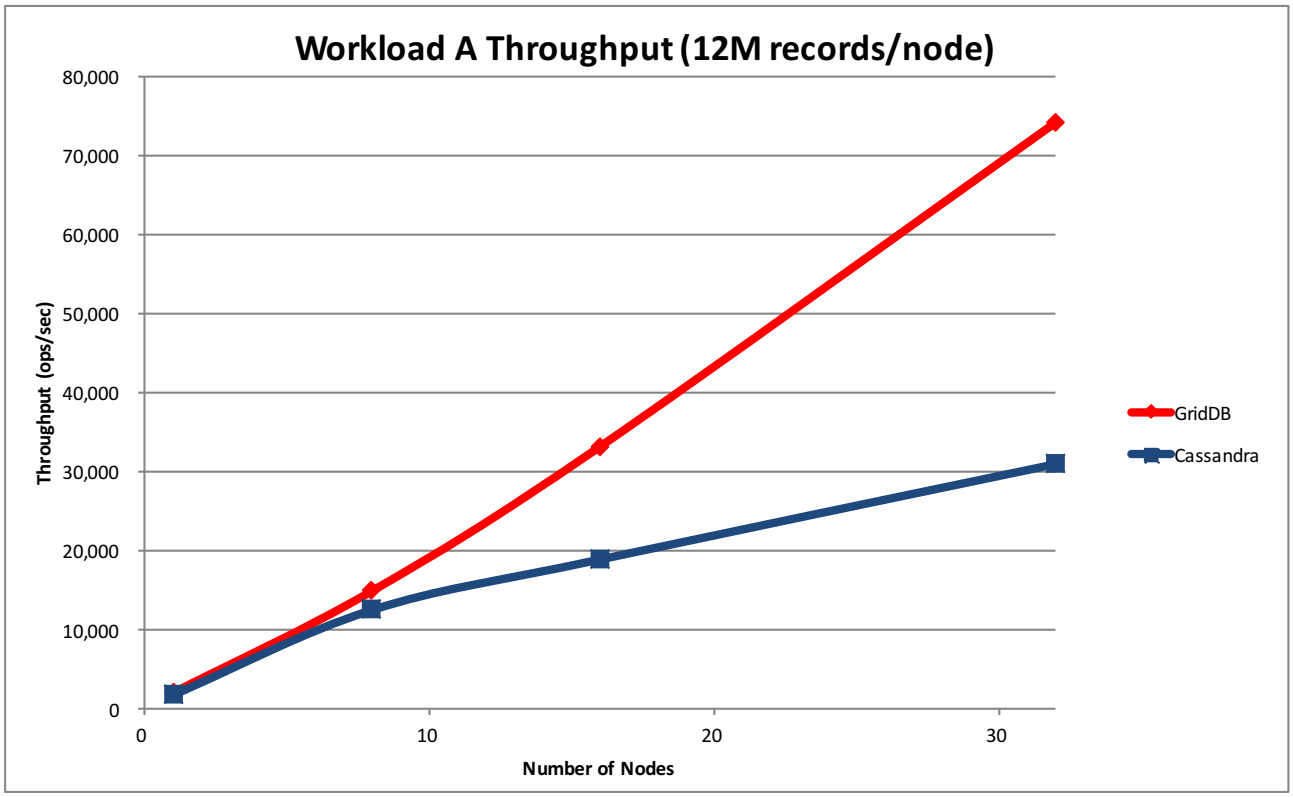

Load Latency (4M records/node)
Latency (ms)
40
35
30
25
20
15
10
5
0
GridDB
Cassandra
1 Node
8 Nodes
16 Nodes
32 Nodes
Insert

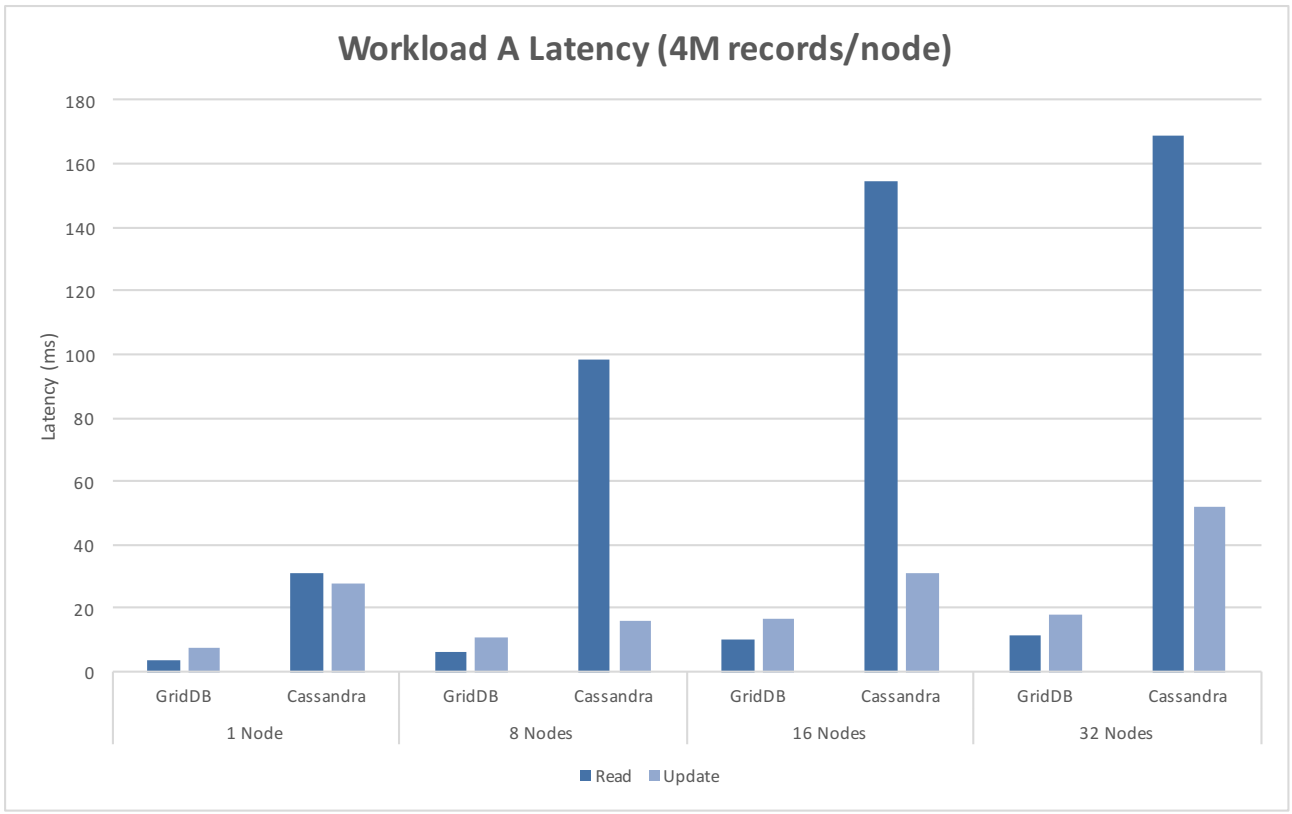

Load Latency (12M records/node)
Latency (ms)
45
40
35
30
25
20
15
10
5
0
GridDB
Cassandra
1 Node
8 Nodes
16 Nodes
32 Nodes
Insert

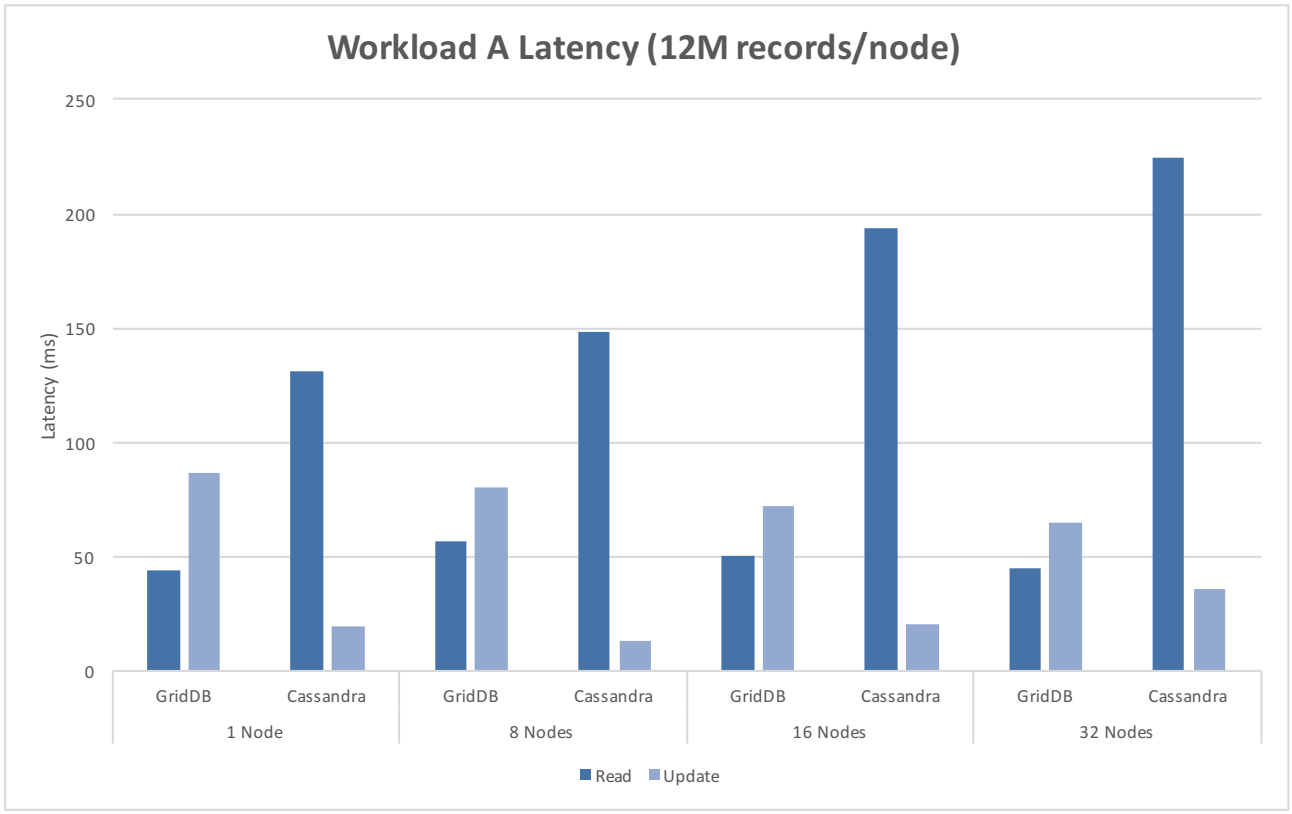
Workload A

Workload A is an update intensive workload. For throughput, higher is better and for latency, lower is better.



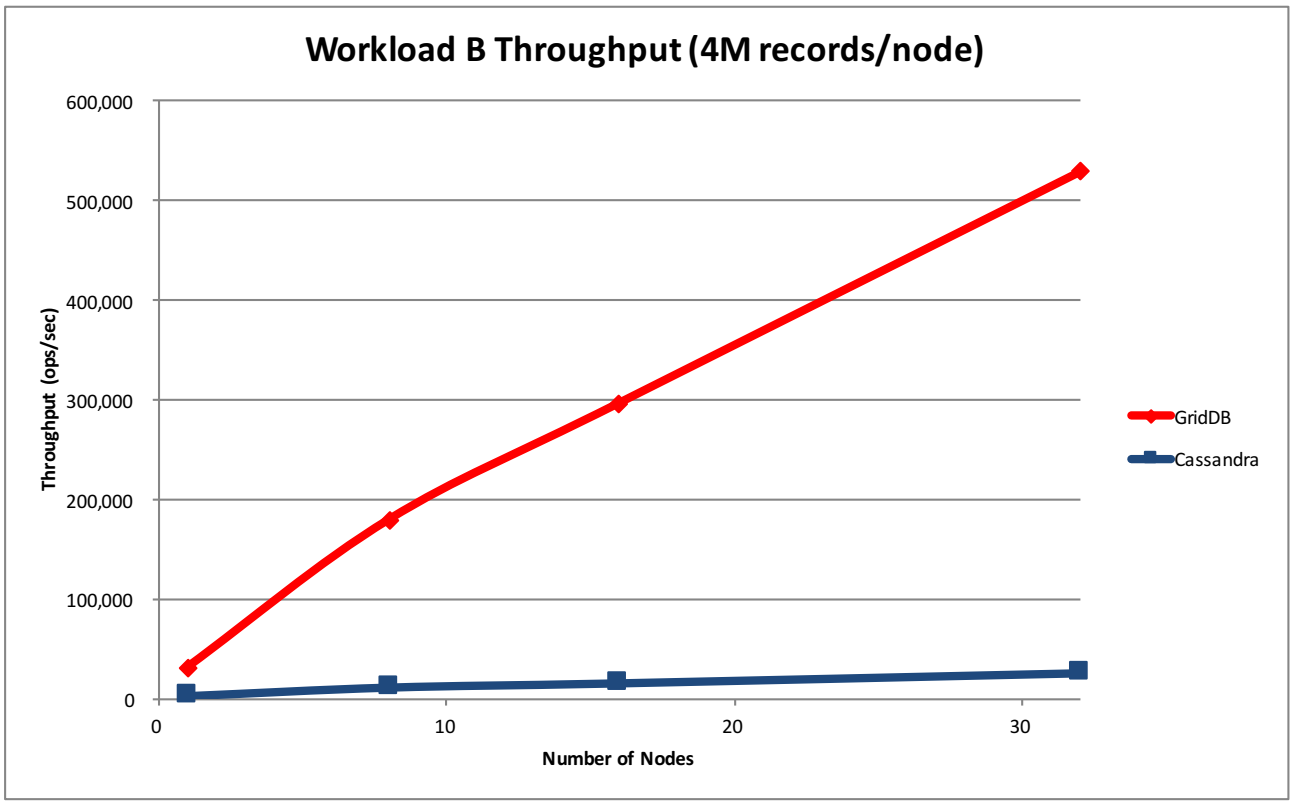


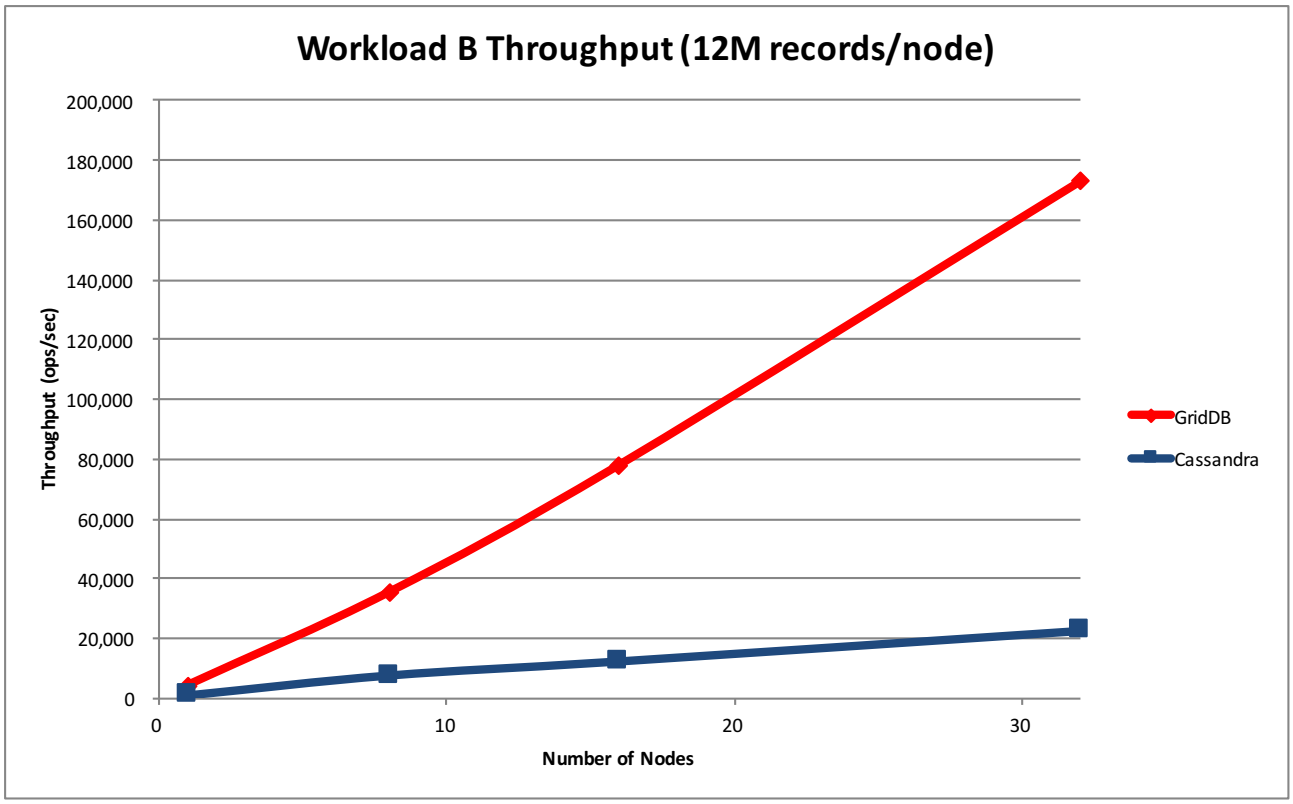

Workload A Latency (4M records/node)
Latency (ms)
180
160
140
120
100
80
60
40
20
0
GridDB
Cassandra
1 Node
8 Nodes
16 Nodes
32 Nodes
Read
Update

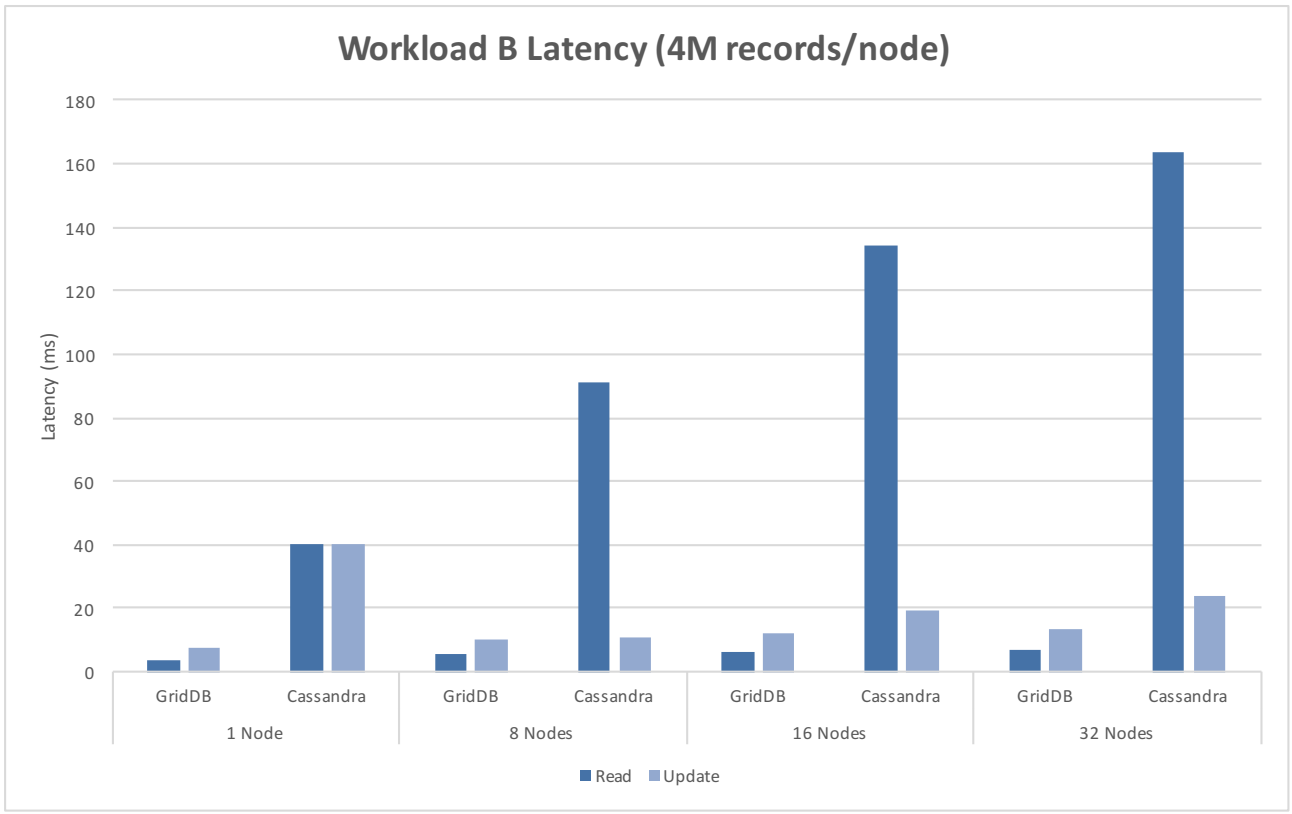

Workload A Latency (12M records/node)
Latency (ms)
250
200
150
100
50
0
GridDB
Cassandra
1 Node
8 Nodes
16 Nodes
32 Nodes
Read
Update

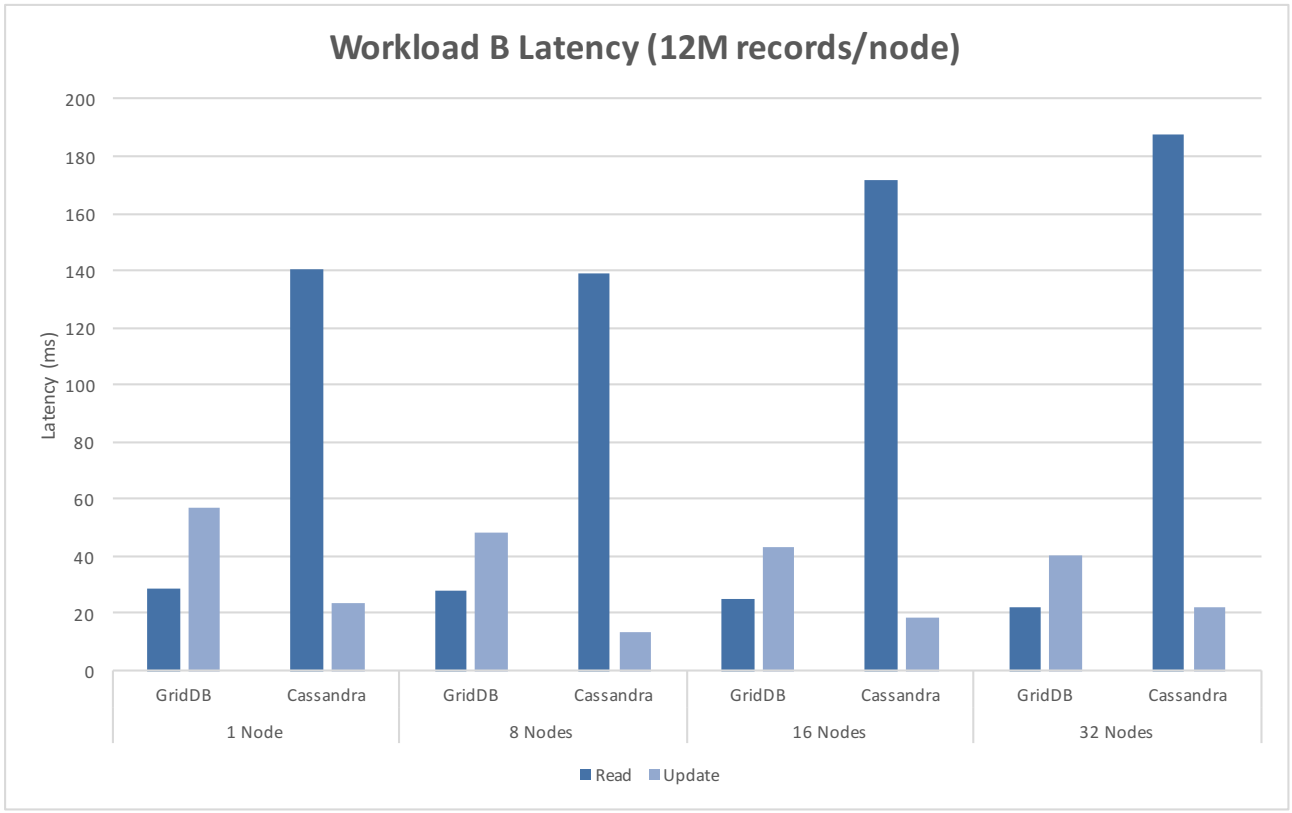
Workload B

Workload B contains 95% read operations and 5% write operations. For throughput, higher is better and for latency, lower is better.



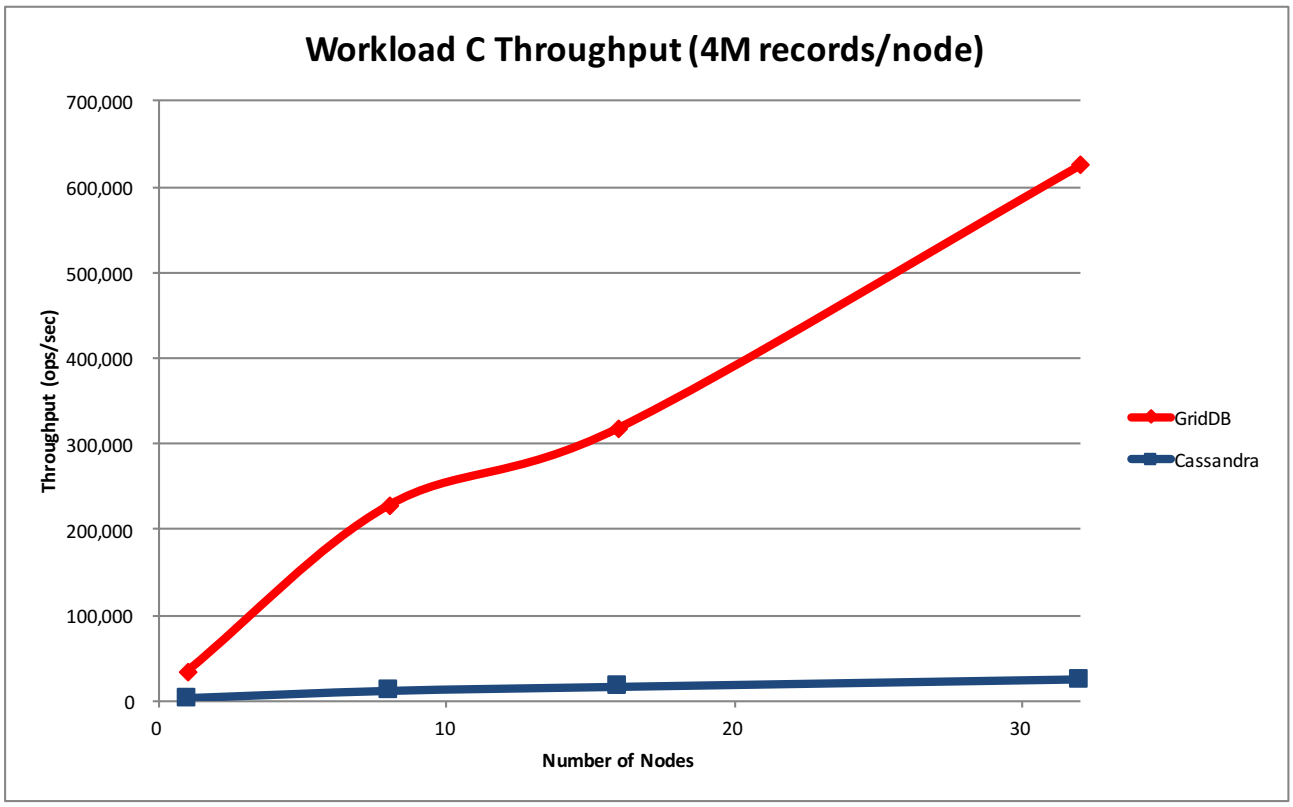


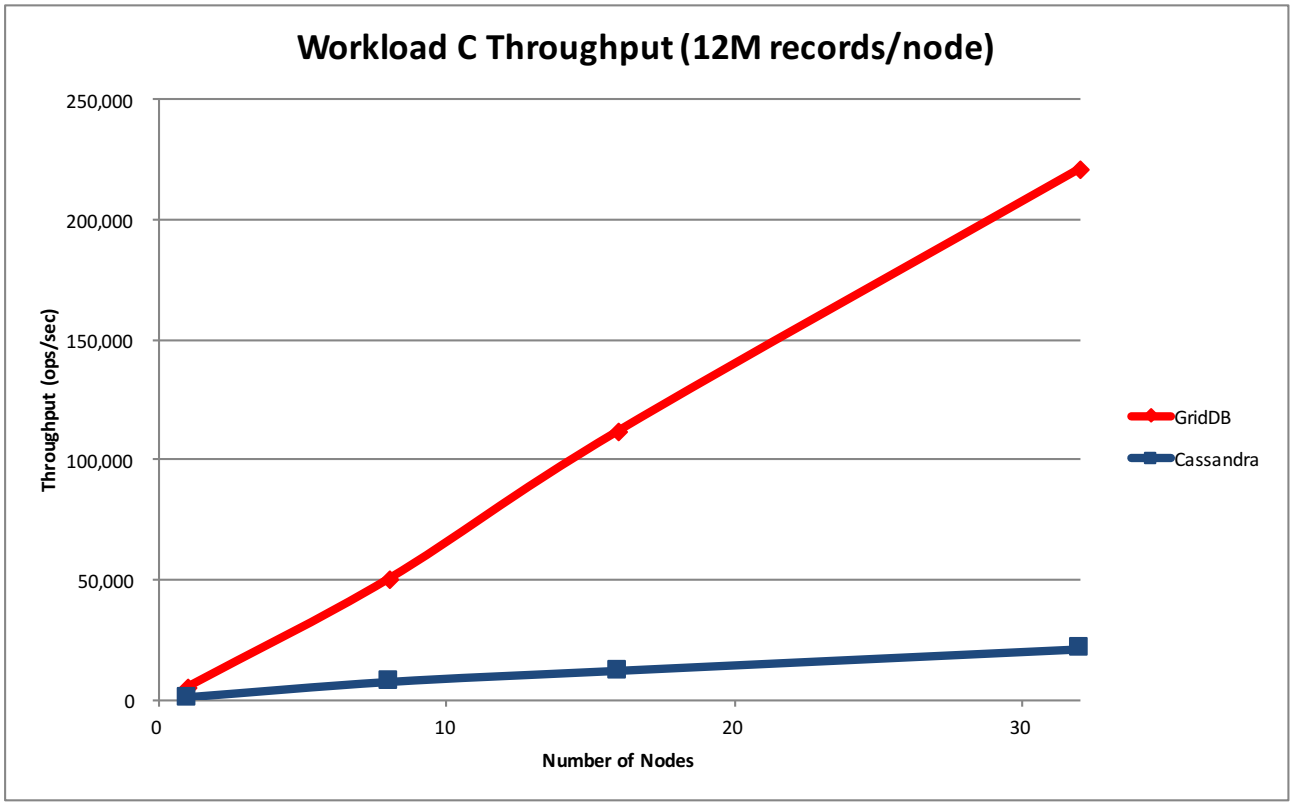

Workload B Latency (4M records/node)
Latency (ms)
180
160
140
120
100
80
60
40
20
0
GridDB
Cassandra
1 Node
8 Nodes
16 Nodes
32 Nodes
Read
Update

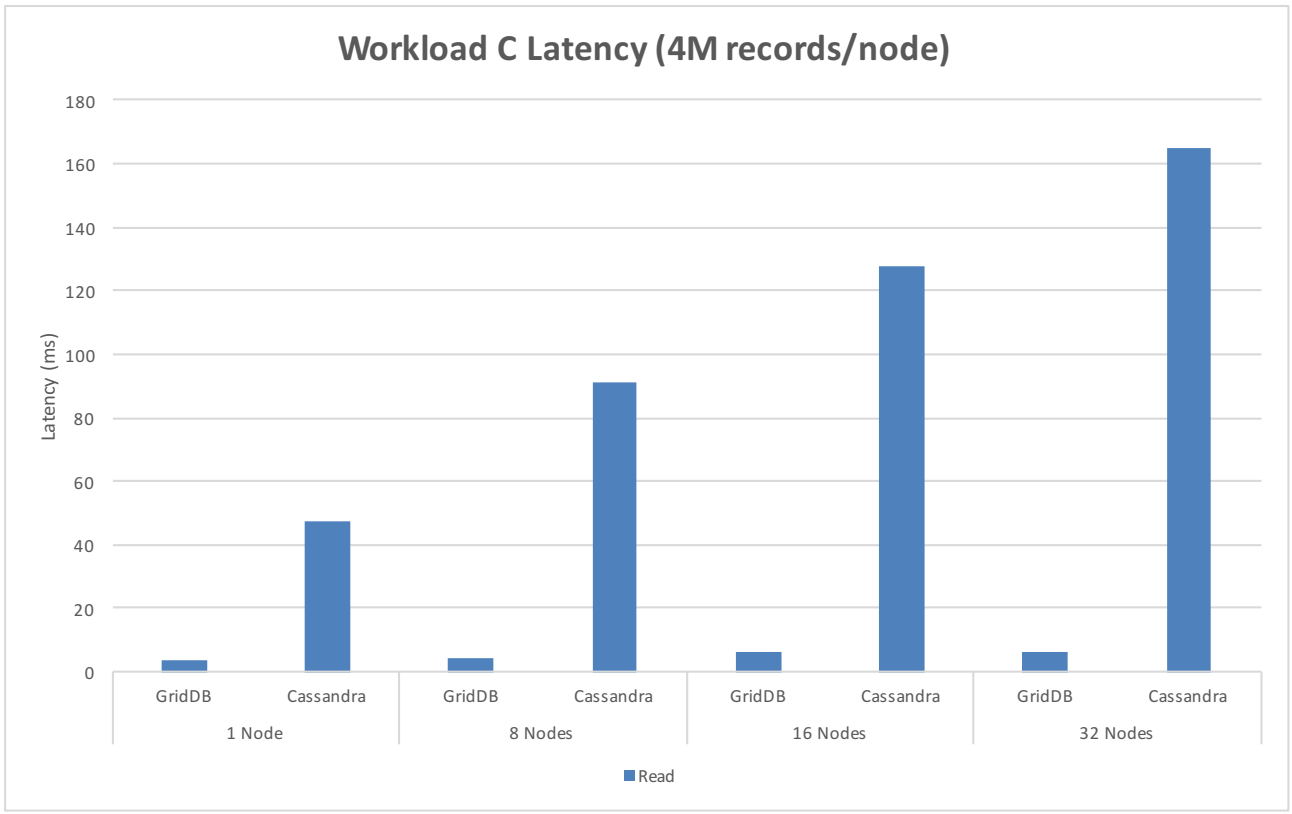

Workload B Latency (12M records/node)
Latency (ms)
200
180
160
140
120
100
80
60
40
20
0
GridDB
Cassandra
1 Node
8 Nodes
16 Nodes
32 Nodes
Read
Update

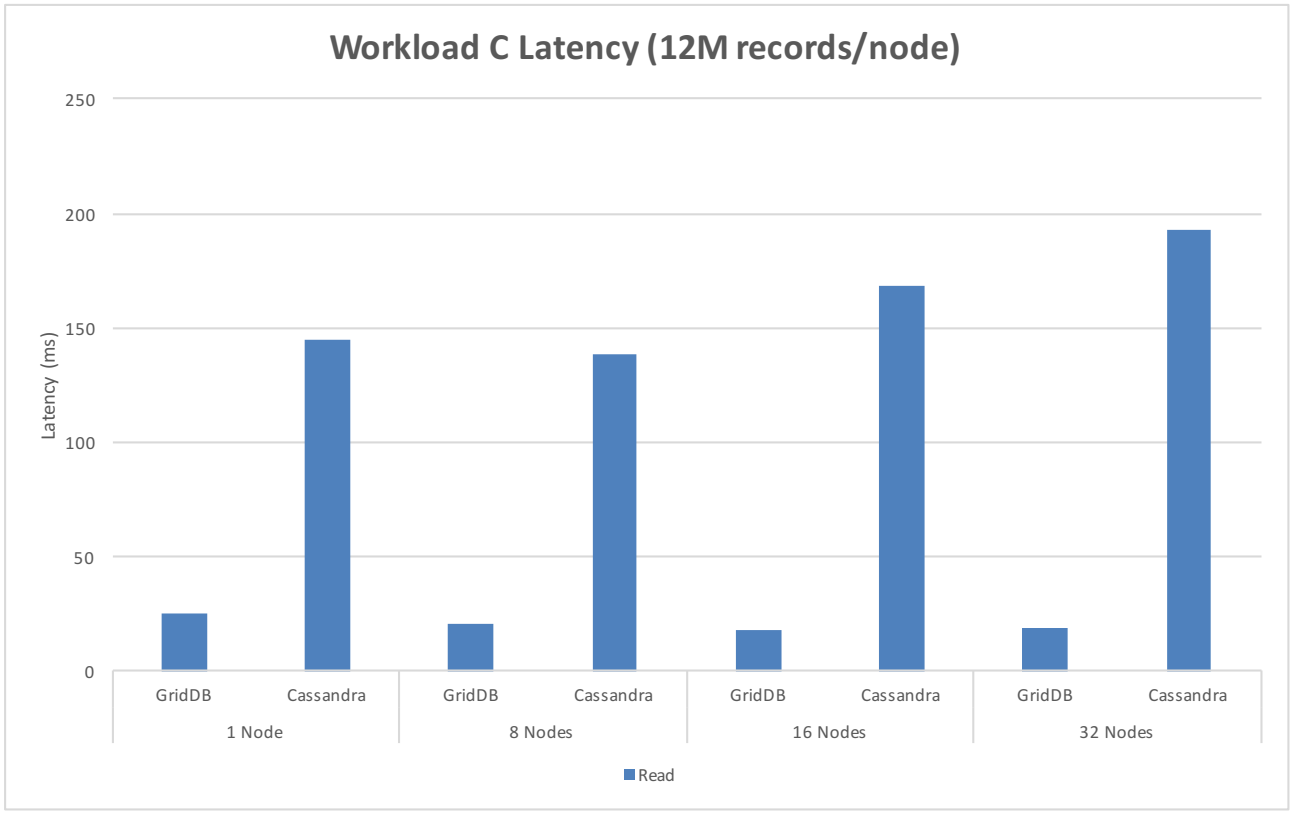
Workload C

Workload C is only read operations. For throughput, higher is better and for latency, lower is better.



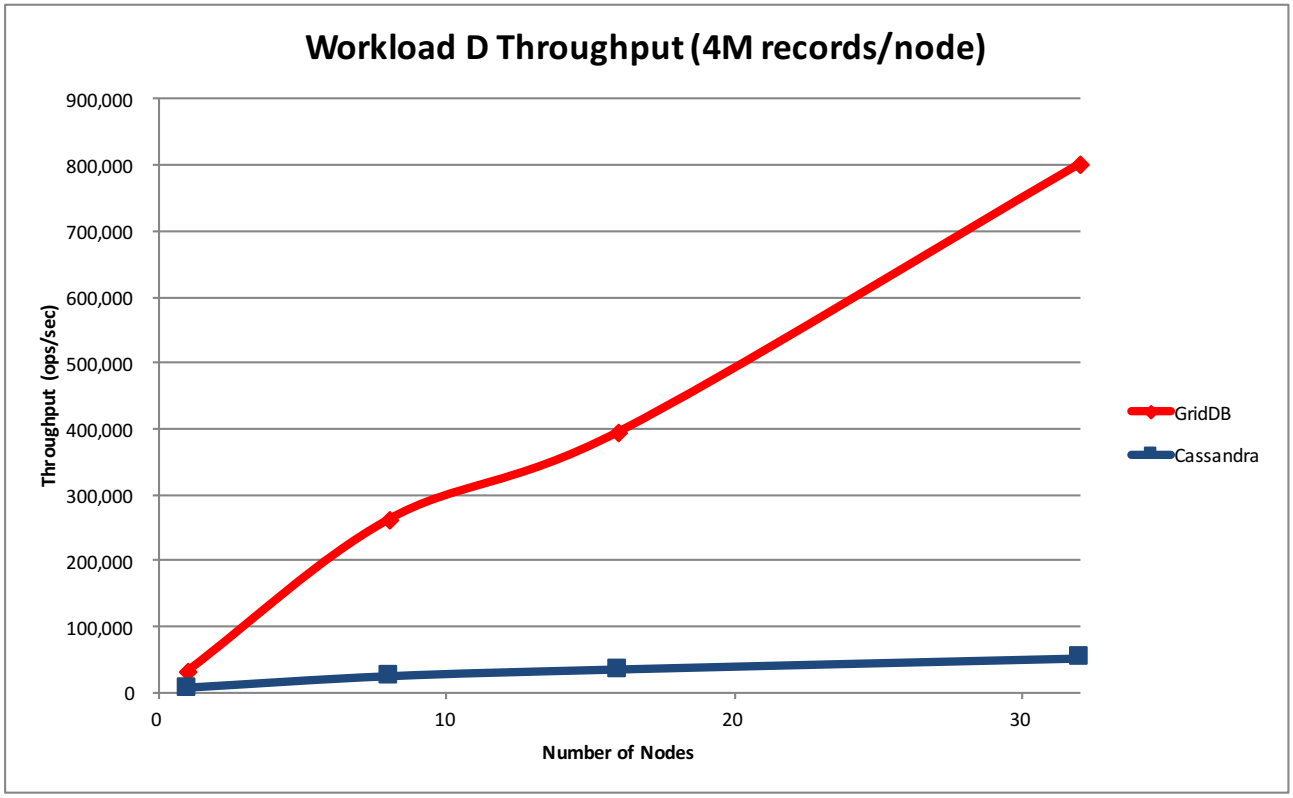


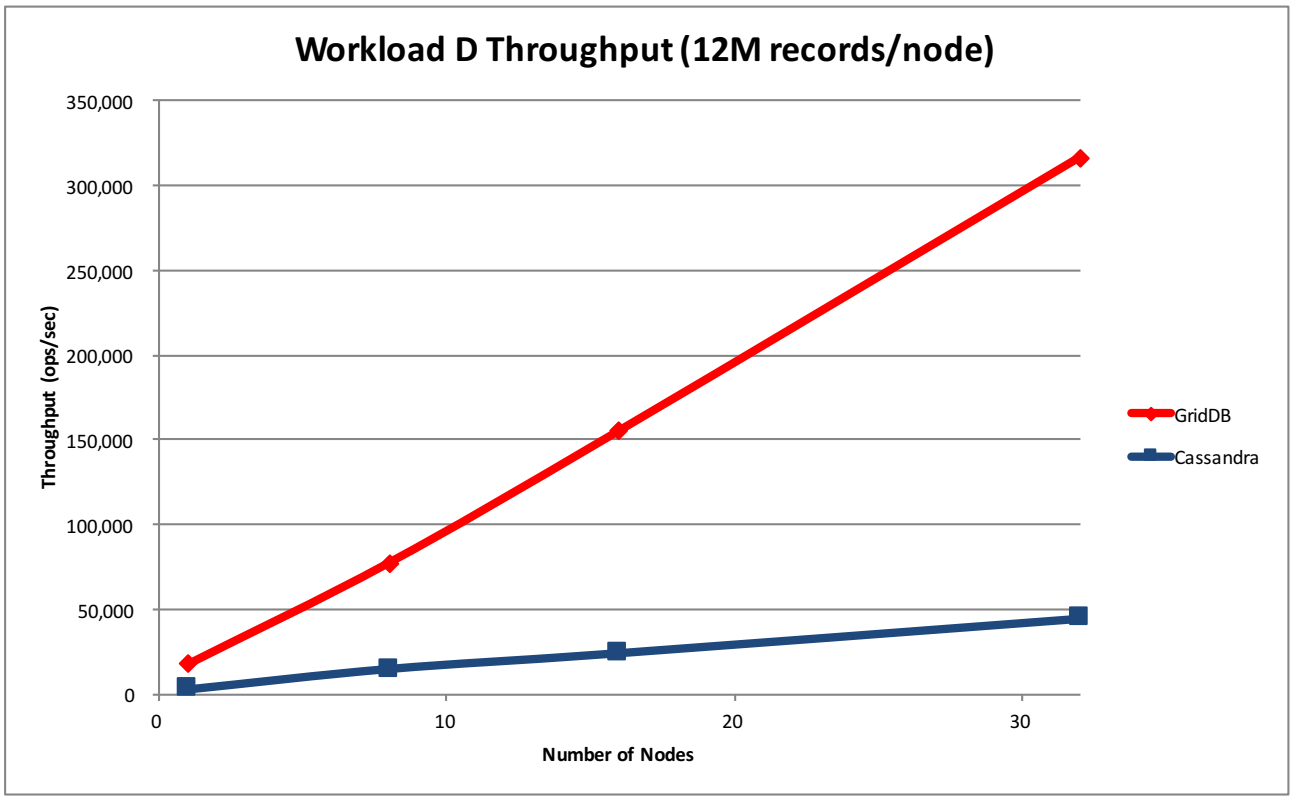

Workload C Latency (4M records/node)
Latency (ms)
180
160
140
120
100
80
60
40
20
0
GridDB
Cassandra
1 Node
8 Nodes
16 Nodes
32 Nodes
Read

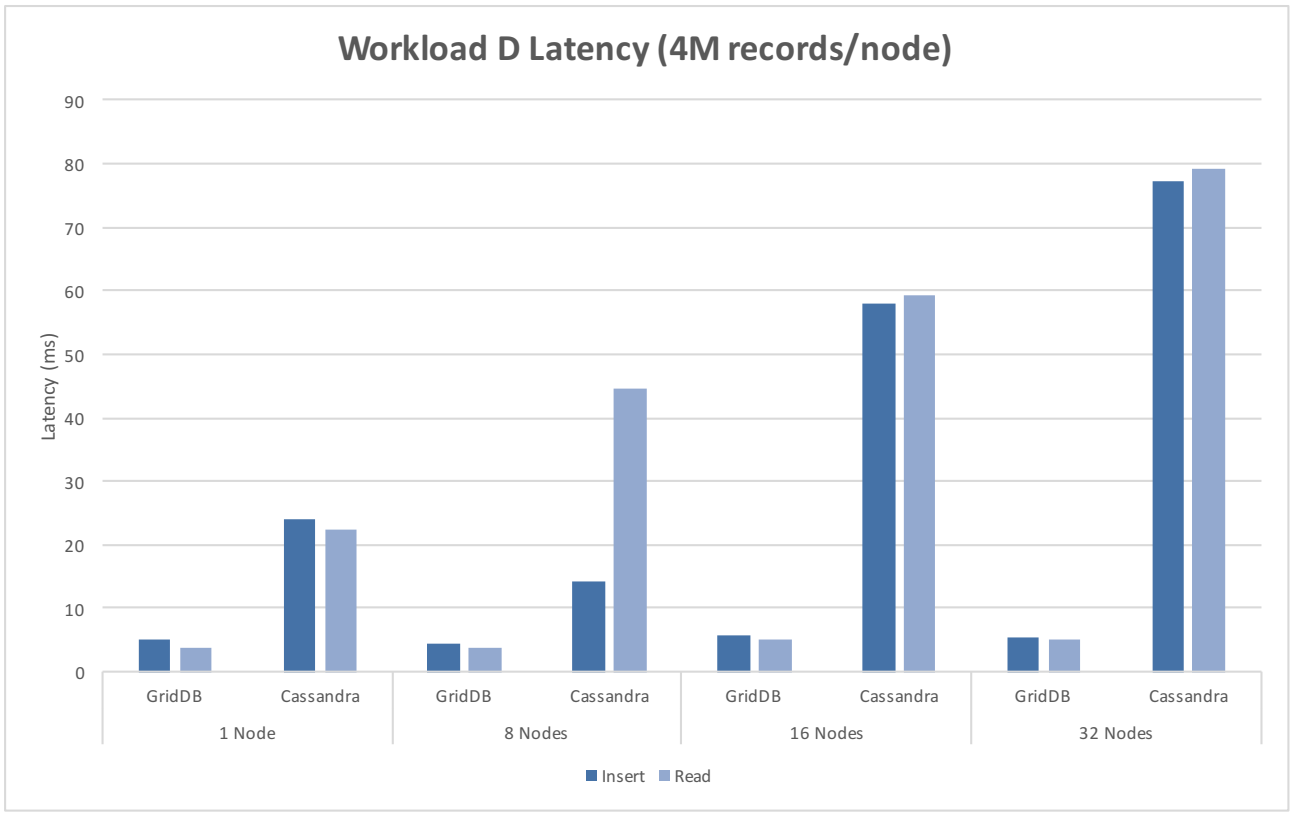

Workload C Latency (12M records/node)
Latency (ms)
250
200
150
100
50
0
GridDB
Cassandra
1 Node
8 Nodes
16 Nodes
32 Nodes
Read

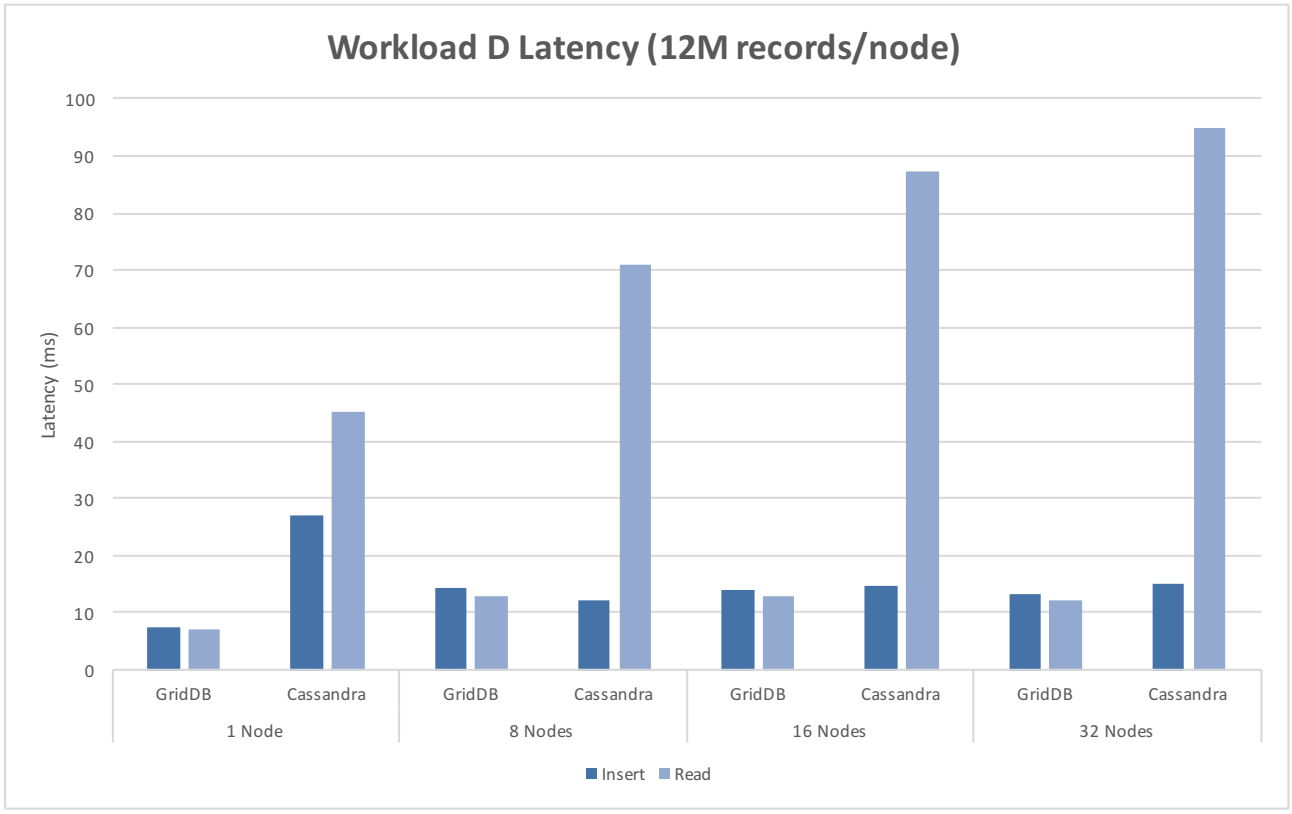
Workload D

Workload D inserts new records and then reads those new records. For throughput, higher is better and for latency, lower is better.



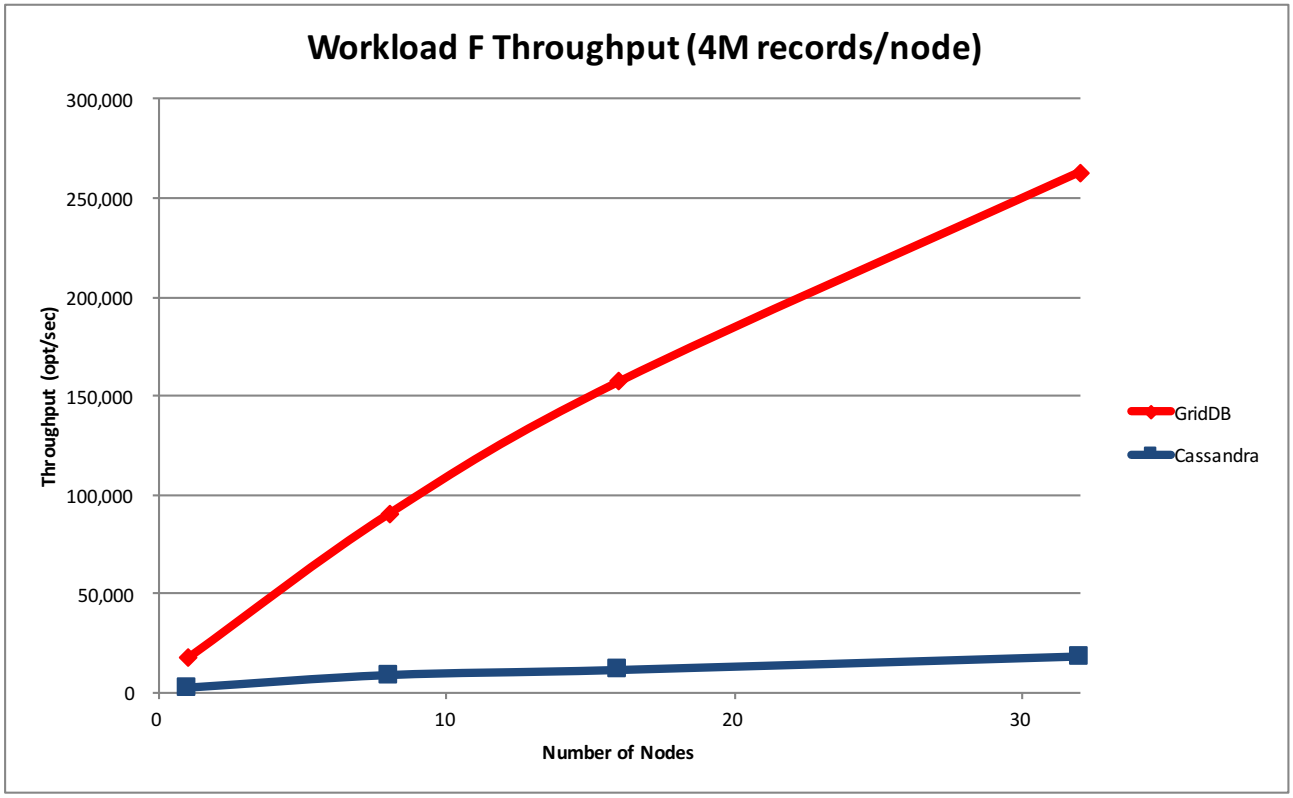


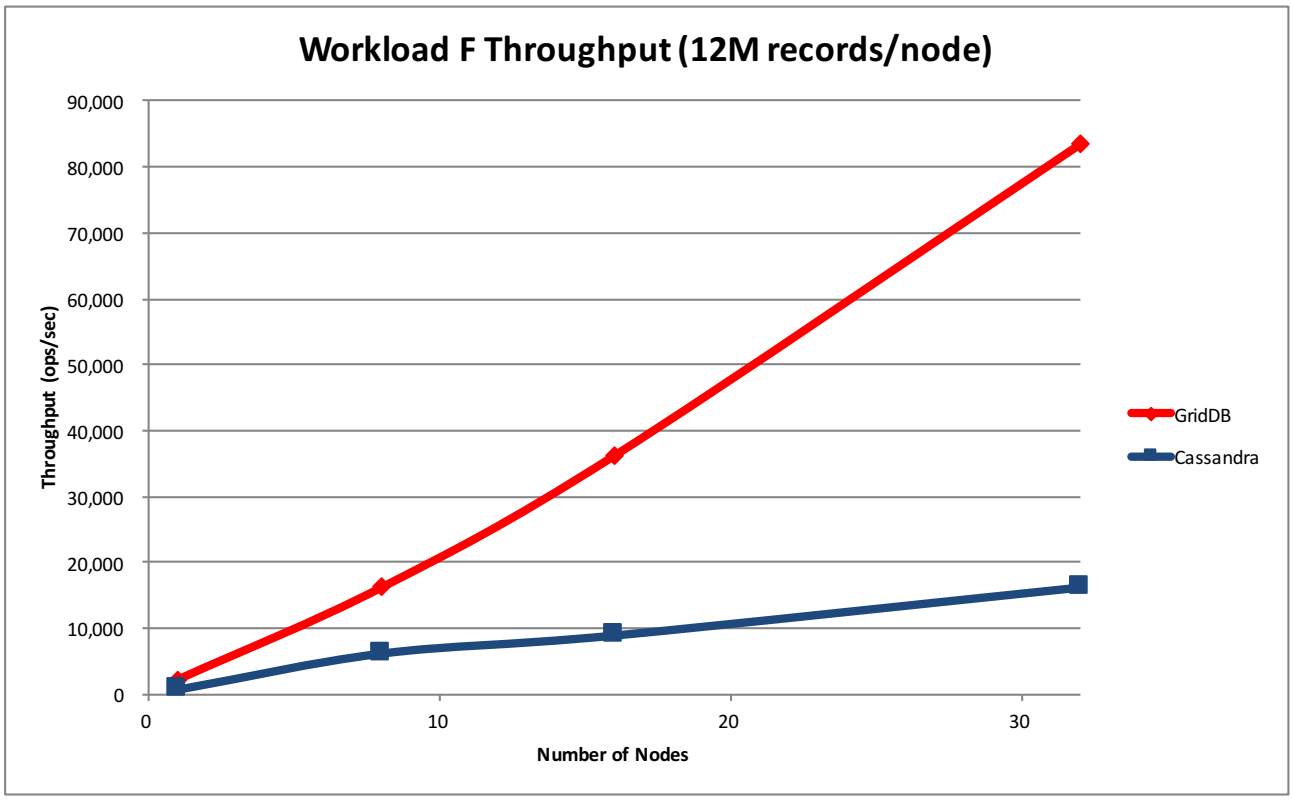

Workload D Latency (4M records/node)
Latency (ms)
90
80
70
60
50
40
30
20
10
0
GridDB
Cassandra
1 Node
8 Nodes
16 Nodes
32 Nodes
Insert
Read

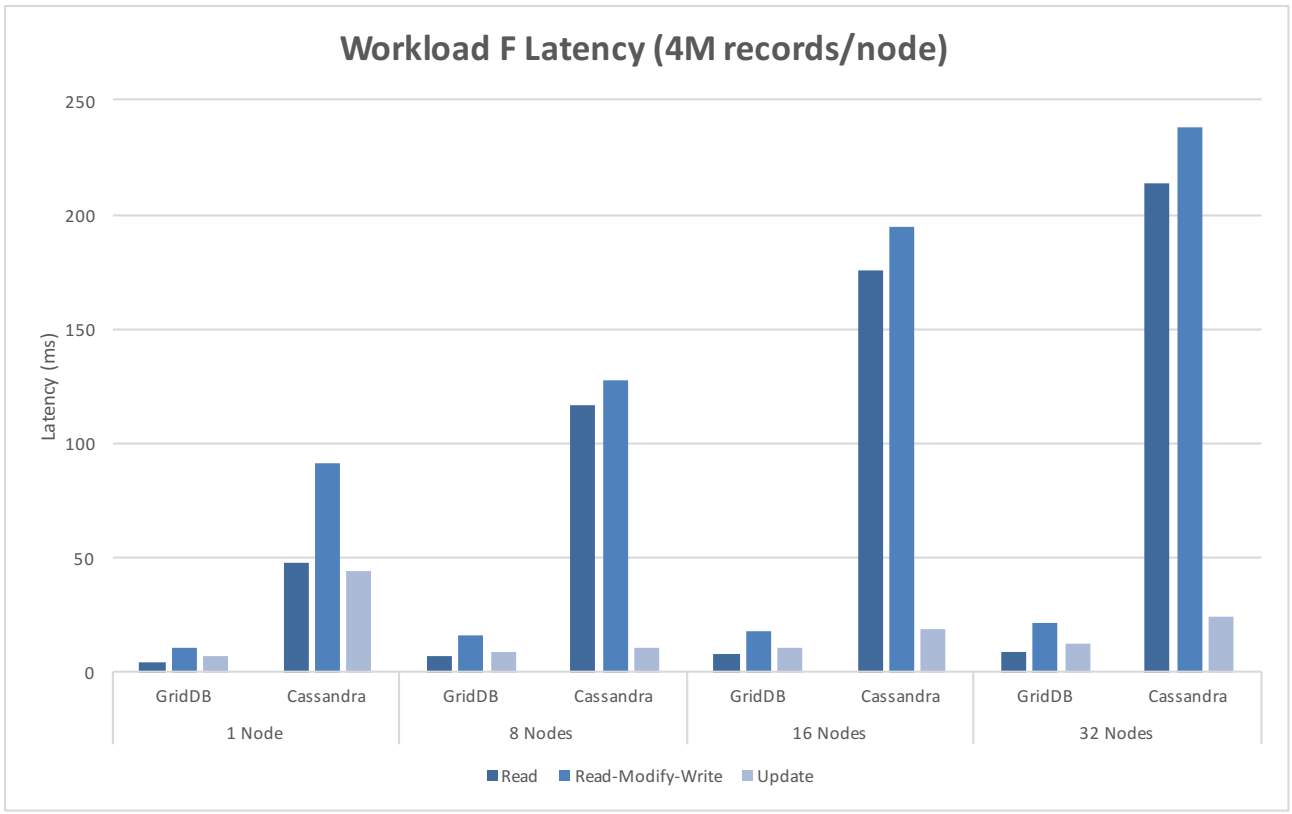

Workload D Latency (12M records/node)
Latency (ms)
100
90
80
70
60
50
40
30
20
10
0
GridDB
Cassandra
1 Node
8 Nodes
16 Nodes
32 Nodes
Insert
Read

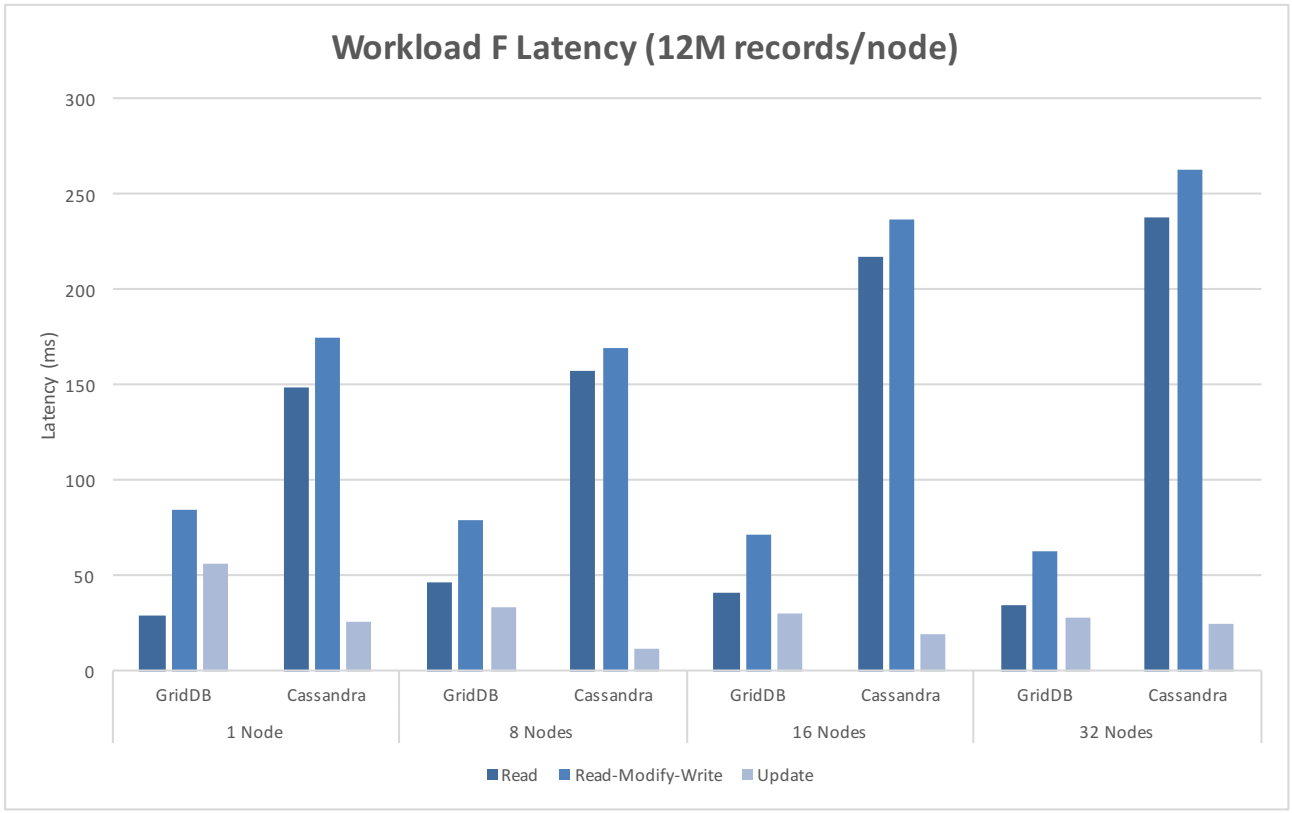
Workload F

Workload F reads a record, modifies it, and then writes it back. For throughput, higher is better and for latency, lower is better.





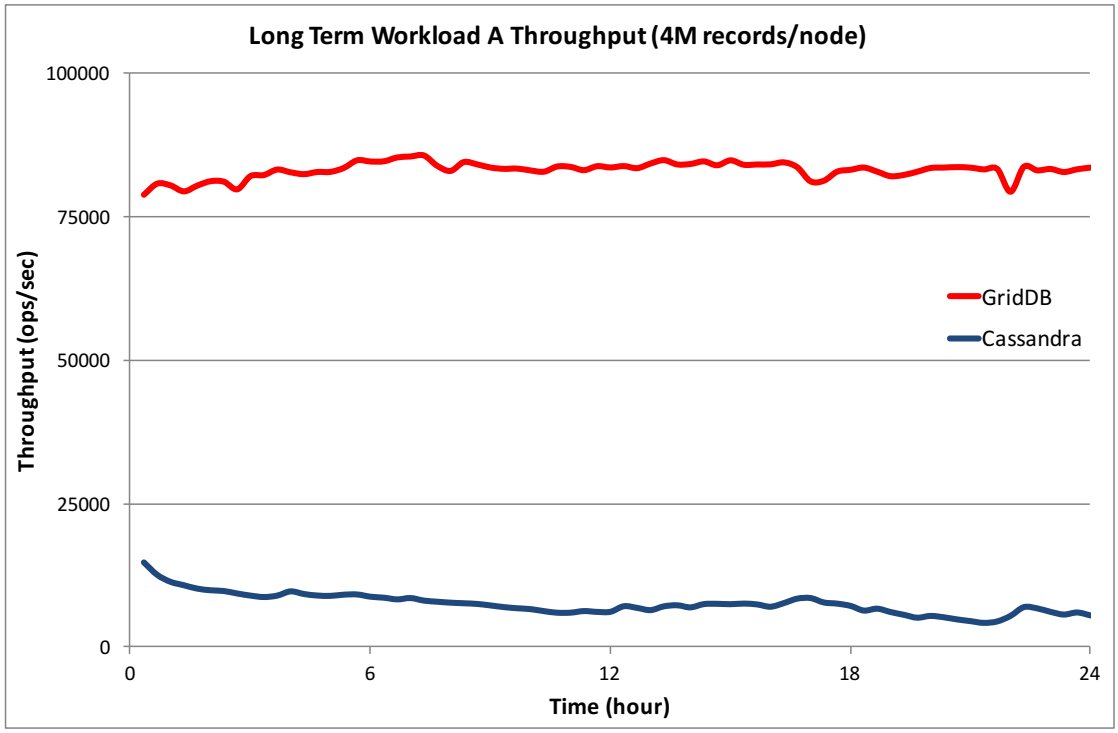

Workload F Latency (4M records/node)
Latency (ms)
250
200
150
100
50
0
GridDB
Cassandra
1 Node
8 Nodes
16 Nodes
32 Nodes
Read
Read-Modify-Write
Update

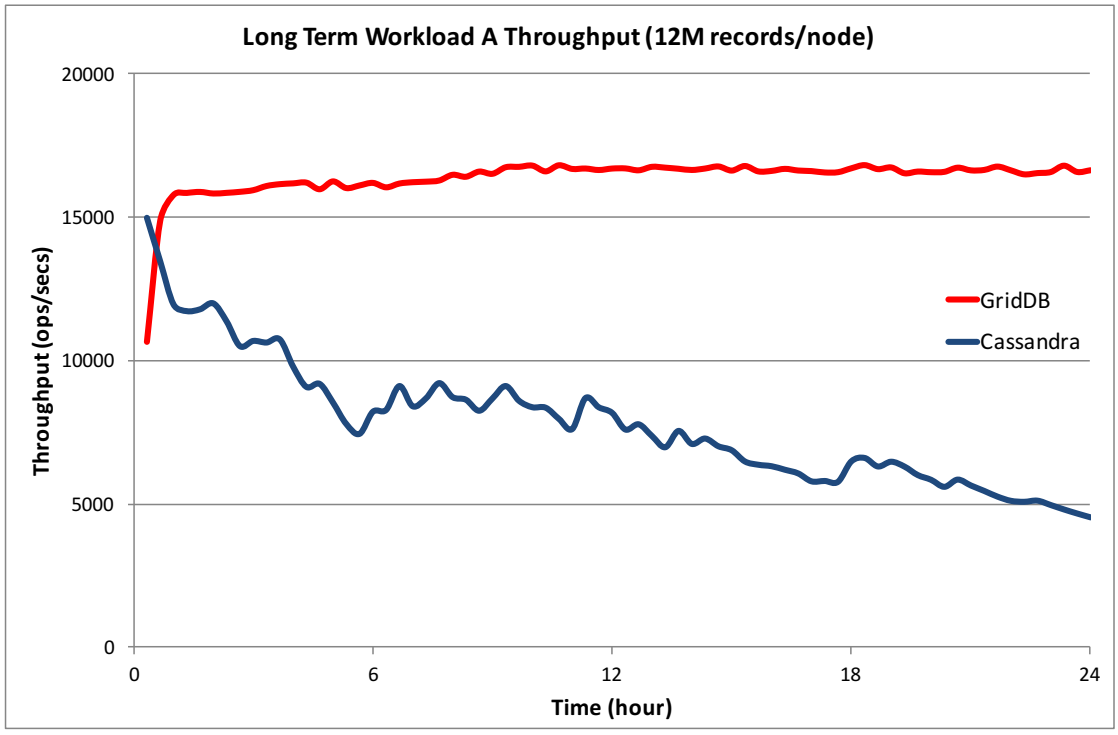

Workload F Latency (12M records/node)
Latency (ms)
300
250
200
150
100
50
0
GridDB
Cassandra
1 Node
8 Nodes
16 Nodes
32 Nodes
Read
Read-Modify-Write
Update

Long Term Workload A

In update-intensive workloads such as Workload A, Cassandra's initial results are quite favorable as its log based architecture allows it to quickly mark a row as deleted and then append the new value to the end of the log. Fixstars noticed that over time Cassandra began to slow down. Fixstars configured an 8-node cluster and loaded 4M and 12M records per node and set operationcount to 2^32-1 and let the test run for twenty-four hours.

Although it is easier to see the trend with the larger data set, with both tests, Cassandra's throughput is less than 50% of what it was in the twenty forth hour versus the first. Meanwhile GridDB's performance was stable when doing both in and out of memory operations.





Tabular Results

All throughput values are “operations per second” and all latency values are “microseconds”.

Throughput with Small Data Set (4M Records Per Node)

		1 Node	8 Nodes	16 Nodes	32 Nodes
Load	GridDB	20,425	123,859	184,836	369,046
	Cassandra	4,246	15,223	19,753	30,304
Workload A	GridDB	21,286	117,284	157,347	270,690
	Cassandra	4,330	17,656	21,781	36,496
Workload B	GridDB	31,449	179,842	296,967	529,091
	Cassandra	3,171	11,657	15,865	25,832
Workload C	GridDB	33,796	227,802	318,485	624,954
	Cassandra	2,707	11,174	15,886	24,623
Workload D	GridDB	31,010	261,624	395,112	801,982
	Cassandra	5,672	23,654	34,246	51,389
Workload F	GridDB	17,300	90,310	157,144	262,940
	Cassandra	1,837	8,351	10,971	17,942

Latency with Small Data Set (4M Records Per node) -- 1 Node

		Insert	Read	Read-Mod-Write	Update
Load	GridDB	6.0			
	Cassandra	7.0			
Workload A	GridDB		3.9		7.7
	Cassandra		30.9		28.1
Workload B	GridDB		3.8		7.6
	Cassandra		40.2		40.0
Workload C	GridDB		3.6		
	Cassandra		47.2		
Workload D	GridDB	5.1	3.9		
	Cassandra	24.1	22.4		
Workload F	GridDB		3.7	10.7	6.9
	Cassandra		47.5	91.6	44.1

Latency with Small Data Set (4M Records Per node) -- 8 Nodes

		Insert	Read	Read-Mod-Write	Update
Load	GridDB	8.0			
	Cassandra	13.3			
Workload A	GridDB		6.2		11.0
	Cassandra		98.6		16.3
Workload B	GridDB		5.3		10.3
	Cassandra		91.3		10.6
Workload C	GridDB		4.4		
	Cassandra		91.0		
Workload D	GridDB	4.5	3.8		
	Cassandra	14.2	44.5		
Workload F	GridDB		6.7	15.7	9.0
	Cassandra		47.5	91.6	44.1

Latency with Small Data Set (4M Records Per node) -- 16 Nodes

		Insert	Read	Read-Mod-Write	Update
Load	GridDB	10.7			
	Cassandra	25.7			
Workload A	GridDB		10.2		16.5
	Cassandra		154.7		30.9
Workload B	GridDB		6.4		12.3
	Cassandra		134.0		19.3
Workload C	GridDB		6.3		
	Cassandra		128.0		
Workload D	GridDB	5.7	5.0		
	Cassandra	57.9	59.3		
Workload F	GridDB		7.5	18.2	10.6
	Cassandra		176.0	194.5	18.6

Latency with Small Data Set (4M Records Per node) -- 32 Nodes

		Insert	Read	Read-Mod-Write	Update
Load	GridDB	13.4			
	Cassandra	33.6			
Workload A	GridDB		11.3		18.0
	Cassandra		168.7		52.2
Workload B	GridDB		6.8		13.2
	Cassandra		163.8		23.7
Workload C	GridDB		6.2		
	Cassandra		164.7		
Workload D	GridDB	5.5	4.9		
	Cassandra	77.2	79.1		
Workload F	GridDB		8.7	21.3	12.6
	Cassandra		213.8	237.8	24.0

Throughput with Large Data Set (12M Records Per Node)

		1 Node	8 Nodes	16 Nodes	32 Nodes
Load	GridDB	13,082	80,074	141,847	277,243
	Cassandra	4,325	12,405	18,063	25,412
Workload A	GridDB	1,945	14,847	33,078	74,053
	Cassandra	1,699	12,485	18,892	30,973
Workload B	GridDB	4,233	35,419	78,117	173,166
	Cassandra	951	7,674	12,431	22,684
Workload C	GridDB	5,149	50,211	111,996	220,950
	Cassandra	884	7,353	12,082	21,129
Workload D	GridDB	17,575	77,486	155,445	316,608
	Cassandra	2,881	15,003	24,349	44,677
Workload F	GridDB	2,242	16,209	36,188	83,399
	Cassandra	788	6,236	8,960	16,212

Latency with Large Data Set (12M Records Per Node) -- 1 Node

		Insert	Read	Read-Mod-Write	Update
Load	GridDB	9.7			
	Cassandra	7.0			
Workload A	GridDB		44.2		87.0
	Cassandra		130.8		19.5
Workload B	GridDB		28.7		56.9
	Cassandra		140.2		23.8
Workload C	GridDB		24.8		
	Cassandra		144.6		
Workload D	GridDB	7.3	7.2		
	Cassandra	27.0	45.3		
Workload F	GridDB		29.1	84.7	55.6
	Cassandra		149.1	175.2	26.1

Latency with Large Data Set (12M Records Per Node) -- 8 Nodes

		Insert	Read	Read-Mod-Write	Update
Load	GridDB	12.9			
	Cassandra	15.9			
Workload A	GridDB		56.4		80.6
	Cassandra		148.4		13.1
Workload B	GridDB		27.7		48.4
	Cassandra		139.0		13.2
Workload C	GridDB		20.2		
	Cassandra		138.6		
Workload D	GridDB	14.2	13.0		
	Cassandra	12.1	70.8		
Workload F	GridDB		46.1	79.3	33.2
	Cassandra		47.5	91.6	44.1

Latency with Large Data Set (12M Records Per Node) -- 16 Nodes

		Insert	Read	Read-Mod-Write	Update
Load	GridDB	14.2			
	Cassandra	28.2			
Workload A	GridDB		50.4		72.3
	Cassandra		194.0		20.2
Workload B	GridDB		25.1		43.4
	Cassandra		171.5		18.4
Workload C	GridDB		18.1		
	Cassandra		168.6		
Workload D	GridDB	13.9	13.0		
	Cassandra	14.6	87.1		
Workload F	GridDB		41.0	71.1	30.1
	Cassandra		217.7	236.6	18.9

Latency with Large Data Set (12M Records Per Node) -- 32 Nodes

		Insert	Read	Read-Mod-Write	Update
Load	GridDB	14.5			
	Cassandra	40.1			
Workload A	GridDB		44.6		65.1
	Cassandra		224.8		35.7
Workload B	GridDB		22.4		40.6
	Cassandra		187.4		22.3
Workload C	GridDB		18.2		
	Cassandra		192.4		
Workload D	GridDB	13.4	12.3		
	Cassandra	14.9	94.8		
Workload F	GridDB		34.7	62.3	27.6
	Cassandra		238.3	262.6	24.3

Conclusion

GridDB's hybrid storage architecture, in-memory-oriented architecture, outperforms Cassandra both in-memory and in operations required using out-of-memory storage. GridDB accomplishes this while maintaining the same reliability and consistency through twenty-four hours of operation.

The internode communication of GridDB scales significantly better than Cassandra's decentralized peer-to-peer system, at least up through 32 nodes. GridDB's performance increases by nearly the same factor as the number of nodes added; Cassandra is only able to scale at 50% of that same factor with this particular Azure instance type.

Appendices

Configuration Files

gs_node.json

```
{
    "dataStore":{
        "dbPath":"data",
        "storeMemoryLimit":"6144MB",
        "storeWarmStart":true,
        "concurrency":2,
        "logWriteMode":1,
        "persistencyMode":"NORMAL",
        "affinityGroupSize":4
    },
    "checkpoint":{
        "checkpointInterval":"1200s",
        "checkpointMemoryLimit":"512MB",
        "useParallelMode":false
    },
    "cluster":{
        "servicePort":10010
    },
    "sync":{
        "servicePort":10020
    },
    "system":{
        "servicePort":10040,
        "eventLogPath":"log"
    },
    "transaction":{
        "servicePort":10001,
        "connectionLimit":10000
    },
    "trace":{
        "default":"LEVEL_ERROR",
        "dataStore":"LEVEL_ERROR",
        "collection":"LEVEL_ERROR",
        "timeSeries":"LEVEL_ERROR",
        "chunkManager":"LEVEL_ERROR",
        "objectManager":"LEVEL_ERROR",
        "checkpointFile":"LEVEL_ERROR",
        "checkpointService":"LEVEL_INFO",
        "logManager":"LEVEL_WARNING",
        "clusterService":"LEVEL_ERROR",
        "syncService":"LEVEL_ERROR",
        "systemService":"LEVEL_INFO",
        "transactionManager":"LEVEL_ERROR",
        "transactionService":"LEVEL_ERROR",
        "transactionTimeout":"LEVEL_WARNING",
        "triggerService":"LEVEL_ERROR",
        "sessionTimeout":"LEVEL_WARNING",
        "replicationTimeout":"LEVEL_WARNING",
        "recoveryManager":"LEVEL_INFO",
        "eventEngine":"LEVEL_WARNING",
        "clusterOperation":"LEVEL_INFO",
        "ioMonitor":"LEVEL_WARNING"
```

```
    }
}
```

gs_cluster.json

```
{
    "dataStore":{
        "partitionNum":128,
            "storeBlockSize":"32KB"
    },
        "cluster":{
            "clusterName":"defaultCluster",
            "replicationNum":1,
            "heartbeatInterval":"5s",
            "loadbalanceCheckInterval":"180s",
            "notificationMember": [
            {

                "cluster": {"address":"10.0.0.13", "port":10010},
                "sync": {"address":"10.0.0.13", "port":10020},
                "system": {"address":"10.0.0.13", "port":10040},
                "transaction": {"address":"10.0.0.13", "port":10001},
            }
            ]
        },
        "sync":{
            "timeoutInterval":"30s"
        }
}
```

cassandra.yaml

```
cluster_name: 'Test Cluster'
num_tokens: 256
hinted_handoff_enabled: true
hinted_handoff_throttle_in_kb: 1024
max_hints_delivery_threads: 2
hints_directory: /var/lib/cassandra/hints
hints_flush_period_in_ms: 10000
max_hints_file_size_in_mb: 128
batchlog_replay_throttle_in_kb: 1024
authenticator: AllowAllAuthenticator
authorizer: AllowAllAuthorizer
role_manager: CassandraRoleManager
roles_validity_in_ms: 2000
permissions_validity_in_ms: 2000
credentials_validity_in_ms: 2000
partitioner: org.apache.cassandra.dht.Murmur3Partitioner
data_file_directories:
    - /var/lib/cassandra/data
commitlog_directory: /var/lib/cassandra/commitlog
disk_failure_policy: stop
commit_failure_policy: stop
key_cache_size_in_mb:
key_cache_save_period: 14400
row_cache_size_in_mb: 0
row_cache_save_period: 0
counter_cache_size_in_mb:
counter_cache_save_period: 7200
saved_caches_directory: /var/lib/cassandra/saved_caches
commitlog_sync: periodic
commitlog_sync_period_in_ms: 10000
```

```
commitlog_segment_size_in_mb: 32
seed_provider:
    - class_name: org.apache.cassandra.locator.SimpleSeedProvider
      parameters:
          - seeds:  ${SEEDS}
concurrent_reads: 32
concurrent_writes: 32
concurrent_counter_writes: 32
concurrent_materialized_view_writes: 32
memtable_allocation_type: heap_buffers
index_summary_capacity_in_mb:
index_summary_resize_interval_in_minutes: 60
trickle_fsync: false
trickle_fsync_interval_in_kb: 10240
storage_port: 7000
ssl_storage_port: 7001
start_native_transport: true
native_transport_port: 9042
start_rpc: false
rpc_port: 9160
rpc_keepalive: true
rpc_server_type: sync
thrift_framed_transport_size_in_mb: 15
incremental_backups: false
snapshot_before_compaction: false
auto_snapshot: true
tombstone_warn_threshold: 1000
tombstone_failure_threshold: 100000
column_index_size_in_kb: 64
batch_size_warn_threshold_in_kb: 5
batch_size_fail_threshold_in_kb: 50
compaction_throughput_mb_per_sec: 16
compaction_large_partition_warning_threshold_mb: 100
sstable_preemptive_open_interval_in_mb: 50
read_request_timeout_in_ms: 50000
range_request_timeout_in_ms: 100000
write_request_timeout_in_ms: 100000
counter_write_request_timeout_in_ms: 100000
cas_contention_timeout_in_ms: 10000
truncate_request_timeout_in_ms: 600000
request_timeout_in_ms: 900000
cross_node_timeout: false
endpoint_snitch: SimpleSnitch
dynamic_snitch_update_interval_in_ms: 100
dynamic_snitch_reset_interval_in_ms: 600000
dynamic_snitch_badness_threshold: 0.1
request_scheduler: org.apache.cassandra.scheduler.NoScheduler
server_encryption_options:
    internode_encryption: none
    keystore: conf/.keystore
    keystore_password: cassandra
    truststore: conf/.truststore
    truststore_password: cassandra
client_encryption_options:
    enabled: false
    optional: false
    keystore: conf/.keystore
    keystore_password: cassandra
```

```
internode_compression: all
inter_dc_tcp_nodelay: false
tracetype_query_ttl: 86400
tracetype_repair_ttl: 604800
gc_warn_threshold_in_ms: 1000
enable_user_defined_functions: false
enable_scripted_user_defined_functions: false
windows_timer_interval: 1
transparent_data_encryption_options:
    enabled: false
    chunk_length_kb: 64
    cipher: AES/CBC/PKCS5Padding
    key_alias: testing:1
    key_provider:
      - class_name: org.apache.cassandra.security.JKSKeyProvider
        parameters:
          - keystore: conf/.keystore
            keystore_password: cassandra
            store_type: JCEKS
            key_password: cassandra
```

Cassandra Schema

```
create keyspace ycsb  WITH REPLICATION = {'class' : 'SimpleStrategy',
'replication_factor': 1 };"

create table ycsb.usertable ( y_id varchar primary key, field0 varchar,
field1 varchar, field2 varchar, field3 varchar, field4 varchar, field5
varchar, field6 varchar, field7 varchar, field8 varchar, field9 varchar,
field10 varchar );"
```