DataStax Testing Results and Product Information

Prepared for

Cloud Platform Group, Samsung Electronics

February 2016





Table of Contents

Summary 3

Functionality Tests 3

Case 1 3

Case 2 6

Case 3 11

Case 4 13

Case 5 16

Case 6 18

Case 7.1 20

Case 7.2 21

Case 8 32

Case 9 37

Stability Test 41

24 Hour Stability Test 41

Performance Test 49

Test Environments 49

Results 49

Dynamo DB Comparison Questions 49

Additional Information 61

# Summary

Datastax is pleased to be considered for a true partnership with Samsung Electronics, Mobile Communications Business. We have completed the Functional, Performance, and Stability testing requested. You will also find additional DataStax customer use case reference information directly applicable to the S-Cloud project we discussed, as well as comparative information between DSE and DynamoDB. I believe you will find across our customers we have in depth experience that we want to share with you to help accomplish your project goals. There is a lot of work to be done jointly to deploy the S-Cloud 1.0 platform for Samsung applications and services. We would be honored to work with you on this.

A few words on the importance DataStax puts on a partnership: When DataStax looks to apply engineering and architecture resources in a true partnership it is often defined by the depth, scale and criticality of a project to that customer.  A customer who is willing to undertake an initiative so fundamental to their business and make a technology bet that will define them for the next decade is the type of customer DataStax is looking for to become a true partner. DataStax promises a healthy exchange of honest feedback, ideas and best practices. We build our brand through credibility with our customers and partners. Every quarter we are presented with potential partners, and every quarter the bar gets higher as to who we will enter into a partnership. Samsung Electronics clearly exceeds that bar, and from our CEO Billy on down, we would be honored to live and win with you on this S-Cloud project.

# Functionality Tests

## Case 1

**Base Settings**

DSE base settings were all used for the case tests.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Scenario** | | | | |
| **Case1> CRUD** | * CRUD basic functionality test | | | |
| **#** | **Steps** | **Expected Result** | **Perform Method** | **Actual Result** |
| **1** | Create a 5 node cluster | Database created | CREATE KEYSPACE keyspace1 WITH replication = {'class': 'SimpleStrategy' , 'replication\_factor': 3 }; | Supported  Success  56 ms |
| **2** | [Optional] Create a table | Table created | CREATE TABLE keyspace1.standard1 (key text PRIMARY KEY, C0 text, C1 text, C2 text, C3 text, C4 text, C5 text, C6 text, C7 text, C8 text, C9 text); | Supported  Success  53 ms |
| **3** | Import test data  (total data size : 500GB) | No errors reported | cassandra-stress write n=50000000 -pop dist=UNIFORM\(1..50000000\) -col n=FIXED(10) size=FIXED(1000) -mode native cql3 -schema keyspace="keyspace1" -rate threads=96 -node 172.31.30.83,172.31.30.84,172.31.30.85,172.31.30.87,172.31.30.80 | Supported  Success  67 minutes |
| **4** | Execute Write (put) operation  (value size : 10KB) | No errors reported, Response time | INSERT INTO keyspace1.standard1 (key,"C0","C1","C2","C3","C4","C5","C6","C7","C8","C9") VALUES (0x50313032354f31303639, … ); | Supported  Success  2.5 ms |
| **5** | Execute Read (get) operation | No errors reported, Response time  Expected value read | SELECT \* FROM keyspace1.standard1 WHERE key=0x50313032354f31303639; | Supported  Success  3.8 ms |
| **6** | Execute Update operation | No errors reported, Response time | UPDATE keyspace1.standard1 SET C0 =0x50313032354f32342339 WHERE key=0x50313032354f31303639; | Supported  Success  1.8 ms |
| **7** | Execute Read (get) operation | No errors reported, Response time  Expected value read | SELECT C0 FROM keyspace1.standard1 WHERE key=0x50313032354f31303639; | Supported  Success  3.4 ms |
| **8** | Execute Delete operation | No errors reported, Response time | DELETE FROM keyspace1.standard1 WHERE key=0x50313032354f31303639; | Supported  Success  1.6 ms |
| **9** | Execute Read (get) operation | No errors reported, Response time  Expected value read | SELECT \* FROM keyspace1.standard1 WHERE key=0x50313032354f31303639; | Supported  Success  2.6 ms |
|  | | | | |
| **Detailed Test Result** | | | | |
| * Success/fail * Support/non-support * Operation (put, get, update, delete) response time : ?? sec | | | | |

## Case 2

**Optimization Settings**

Large batch operations across multiple partition keys are highly sub-optimal settings for Cassandra. The reason for this is that Cassandra is optimized to provide low latency individual operations asynchronously. The effect of a large batch across keys is to overload single coordinator node with an overly large unit of work. This results in timeout errors from the batch coordinator node.

The proper Cassandra settings for this test case are to submit individual operations in an asynchronous fashion using prepared statements. This will results in massive throughout and low latency.

The proper use of batch queries in Cassandra is to use a batch when updating a single partition key across multiple tables. The purpose of batches in Cassandra is to provide a single atomic write or read on a single primary key in a single operation. In real work scenarios, this results in batch sizes of two to a a small number of operations, one per query table.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Scenario** | | | | | |
| **Case2> Batch Update/**  **Insert** | * CRUD Batch Functionality Test | | | |
| **#** | **Steps** | **Expected Result** | **Perform Method** | **Actual Result** |
| **1** | Create a 5 node cluster | Database created | CREATE KEYSPACE keyspace1 WITH replication = {'class': 'SimpleStrategy' , 'replication\_factor': 3 }; | Supported  Success  56 ms |
| **2** | [Optional] Create a table | Table created | CREATE TABLE keyspace1.standard1 (key text PRIMARY KEY, C0 text, C1 text, C2 text, C3 text, C4 text, C5 text, C6 text, C7 text, C8 text, C9 text); | Supported  Success  53 ms |
| **3** | Import test data  (total data size : 500GB) | No errors reported | cassandra-stress write n=50000000 -pop dist=UNIFORM\(1..50000000\) -col n=FIXED(10) size=FIXED(1000) -mode native cql3 -schema keyspace="keyspace1" -rate threads=96 -node 172.31.30.83,172.31.30.84,172.31.30.85,172.31.30.87,172.31.30.80 | Supported  Success  67 minutes |
| **4** | Execute Batch Write (put) operation  (value size : 10KB, batch size : 10,000 requests) | No errors reported, Response time | NOTE: this test used the test client in this link: <https://github.com/jshook/testclient> The command is:  time java -Dbatchsize=10 -Dbatchtype=LOGGED -jar testclient.jar --host=54.67.123.84 --activity=WriteTelemetryBatchAsync:10000:100:200 | Supported  Success  5 s |
| **5** | Execute Batch Read (get) operation  (batch size : 10,000 requests) | No errors reported, Response time  Expected value read | Batch reads are technically supported. However due Cassandra’s peer to peer distributed system design, batch operations across multiple partition keys cause suboptimal behavior, resulting in timeouts for this test case. Reads were done asynchronously and values were confirmed | Not Supported in batch.  Values confirmed with async read  5s |
| **6** | Execute Batch Update operation  (batch size : 10,000 requests) | No errors reported, Response time | NOTE: this test used the test client in this link: <https://github.com/jshook/testclient> The command is:  time java -Dbatchsize=10 -Dbatchtype=LOGGED -jar testclient.jar --host=54.67.123.84 --activity=WriteTelemetryBatchAsync:10000:100:200 | Supported  Success  5 s |
| **7** | Execute Batch Read (get) operation  (batch size : 10,000 requests) | No errors reported, Response time  Expected value read | Batch reads are technically supported. However due Cassandra’s peer to peer distributed system design, batch operations across multiple partition keys cause suboptimal behavior, resulting in timeouts for this test case. Reads were done asynchronously and values were confirmed. | Not Supported in batch.  Values confirmed with async read  5s |
| **8** | Execute Batch Delete operation  (batch size : 10,000 requests) | No errors reported, Response time | NOTE: this test used the test client in this link: <https://github.com/jshook/testclient> The command is:  time java -Dbatchsize=10 -Dbatchtype=LOGGED -jar testclient.jar --host=54.67.123.84 --activity=WriteTelemetryBatchAsync:10000:100:200 | Supported  Success  5 s |
| **9** | Execute Batch Read (get) operation  (batch size : 10,000 requests) | No errors reported, Response time  Expected value read | Batch reads are technically supported. However due Cassandra’s peer to peer distributed system design, batch operations across multiple partition keys cause suboptimal behavior, resulting in timeouts for this test case. Reads were done asynchronously and values were confirmed | Not Supported in batch.  Values confirmed with async read  5s |
|  | | | | | |
| **Detailed Test Result** | | | | | |
| * Success/fail * Support/non-support * Operation (put, get, update, delete) response time : ?? sec | | | | | |

## Case 3

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Scenario** | | | | | |
| **Case3> Partial Update** | * Record’s partial attributes update test | | | |
| **#** | **Steps** | **Expected Result** | **Perform Method** | **Actual Result** |
| **1** | Create a 5 node cluster | Database created | CREATE KEYSPACE keyspace1 WITH replication = {'class': 'SimpleStrategy' , 'replication\_factor': 3 }; | Supported  Success  56 ms |
| **2** | [Optional] Create a table | Table created | CREATE TABLE keyspace1.standard1 (key text PRIMARY KEY, C0 text, C1 text, C2 text, C3 text, C4 text, C5 text, C6 text, C7 text, C8 text, C9 text); | Supported  Success  53 ms |
| **3** | Import test data  (total data size : 500GB) | No errors reported | cassandra-stress write n=50000000 -pop dist=UNIFORM\(1..50000000\) -col n=FIXED(10) size=FIXED(1000) -mode native cql3 -schema keyspace="keyspace1" -rate threads=96 -node 172.31.30.83,172.31.30.84,172.31.30.85,172.31.30.87,172.31.30.80 | Supported  Success  67 minutes |
| **4** | Execute Write (put) operation  (value size : 10KB, total attribute count: 8) | No errors reported | INSERT INTO keyspace1.standard1 (key,"C0","C1","C2","C3","C4","C5","C6","C7") VALUES (0x50313032354f31303632,0x…); | Supported  Success  2.1 ms |
| **5** | Execute Read (get) operation | No errors reported  Expected value read | SELECT \* FROM keyspace1.standard1 WHERE key=0x50313032354f31303632; | Supported  Success  15 ms |
| **6** | Execute partial update operation  (update 2 attributes) | No errors reported, Response time | UPDATE keyspace1.standard1 SET C0 = 0x0c6d4ad.., C1 = 0x0c6d4ad.. WHERE key = 0x50313032354f31303632; | Supported  Success  1.8 ms |
| **7** | Execute Read (get) operation | No errors reported  Expected value read | SELECT C0,C1 FROM keyspace1.standard1 WHERE key=0x50313032354f31303632 | Supported  Success  15 ms |
| **8** | Execute partial update operation  (put 2 new attributes) | No errors reported, Response time | UPDATE keyspace1.standard1 SET C8 = 0x0c6d4adc3dff55…, C9 = 0x0c6d4adc3dff55… WHERE key = 0x50313032354f31303632  **;** | Supported  Success  1.6 ms |
| **9** | Execute Read (get) operation | No errors reported  Expected value read | SELECT C8,C9 FROM keyspace1.standard1 WHERE key=0x50313032354f31303632; | Supported  Success  2.4 ms |
| **10** | Execute partial update operation  (delete 2 attributes) | No errors reported, Response time | DELETE C2,C3 FROM keyspace1.standard1 WHERE key=0x50313032354f31303632; | Supported  Success  1.1 ms |
| **11** | Execute Read (get) operation | No errors reported  Expected value read | SELECT C2,C3 FROM keyspace1.standard1 WHERE key=0x50313032354f31303632; | Supported  Success  1.4 ms |
|  | | | | | |
| **Detailed Test Result** | | | | | |
| * Success/fail * Support/non-support * Operation (put, update, delete) response time : ?? sec | | | | | |

## Case 4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Scenario** | | | | |
| **Case4> Conditional Update** | * Checks if a record(whole/part) can be updated/deleted depending on condition(s). | | | |
| **#** | **Steps** | **Expected Result** | **Perform Method** | **Actual Result** |
| **1** | Create a 5 node cluster | Database created | CREATE KEYSPACE keyspace1 WITH replication = {'class': 'SimpleStrategy' , 'replication\_factor': 3 }; | Supported  Success  56 ms |
| **2** | [Optional] Create a table | Table created | CREATE TABLE keyspace1.standard1 (key text PRIMARY KEY, C0 text, C1 text, C2 text, C3 text, C4 text, C5 text, C6 text, C7 text, C8 text, C9 text); | Supported  Success  53 ms |
| **3** | Import test data  (total data size : 500GB) | No errors reported | cassandra-stress write n=50000000 -pop dist=UNIFORM\(1..50000000\) -col n=FIXED(10) size=FIXED(1000) -mode native cql3 -schema keyspace="keyspace1" -rate threads=96 -node 172.31.30.83,172.31.30.84,172.31.30.85,172.31.30.87,172.31.30.80 | Supported  Success  67 minutes |
| **4** | Execute Write (put) operation  (value size : 10KB) | No errors reported | INSERT INTO keyspace1.standard1 (key,"C0","C1","C2","C3","C4","C5","C6","C7") VALUES (0x50313032354f3130363,0x…); | Supported  Success  1.8 ms |
| **5** | Conditional insert  (both true and false) | No errors reported, Response time | INSERT INTO keyspace1.standard1 (key,"C0","C1","C2","C3","C4","C5","C6","C7") VALUES (0x50313032354f31303633,0x…) IF NOT EXISTS; | Supported  Success  18 ms TRUE / 22 ms FALSE |
| **6** | Execute Read (get) operation | No errors reported  Expected value read | SELECT \* FROM keyspace1.standard1 WHERE key=0x50313032354f31303633; | Supported  Success  2.7 ms |
| **7** | Conditional update  (both true and false) | No errors reported, Response time | UPDATE keyspace1.standard1 SET C1=0x0… WHERE key = 0x50313032354f31303633 IF C2=0x0c6d4adc3… ; | Supported  Success  7 ms TRUE / 4 ms FALSE |
| **8** | Execute Read (get) operation | No errors reported  Expected value read | SELECT \* FROM keyspace1.standard1 WHERE key=0x50313032354f31303633; | Supported  Success  2 ms |
| **9** | Conditional delete  (both true and false) | No errors reported, Response time | DELETE C9 FROM keyspace1.standard1 WHERE key=0x50313032354f31303633 IF C2=0x0c6d4adc3d…; | Supported  Success  5 ms TRUE / 3.8 ms FALSE |
| **10** | Execute Read (get) operation | No errors reported  Expected value read | SELECT \* FROM keyspace1.standard1 WHERE key=0x50313032354f31303633; | Supported  Success  1 ms |
| **11** | Conditional update with two conditions combined | No errors reported, Response time | UPDATE keyspace1.standard1 SET C1=0x0… WHERE key = 0x50313032354f31303633 IF C0=0x0c6d4adc3… AND C2 = 0x0c6d4adc3… ; | Supported  Success  6.7 ms TRUE / 3.6 ms FALSE |
| **12** | Execute Read (get) operation | No errors reported  Expected value read | SELECT \* FROM keyspace1.standard1 WHERE key=0x50313032354f31303633; | Supported  Success  1 ms |
|  | | | | |
| **Detailed Test Result** | | | | |
| * Success/fail * Support/non-support * Operation (conditional insert, update, delete) response time : ?? sec | | | | |

## Case 5

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Scenario** | | | | |
| **Case5 > Data Expiration** | * Data expiration functionality test | | | |
| **#** | **Steps** | **Expected Result** | **Perform Method** | **Actual Result** |
| **1** | Create a 5 node cluster | Database created | CREATE KEYSPACE keyspace1 WITH replication = {'class': 'SimpleStrategy' , 'replication\_factor': 3 }; | Supported  Success  56 ms |
| **2** | [Optional] Create a table | Table created | CREATE TABLE keyspace1.standard1 (key text PRIMARY KEY, C0 text, C1 text, C2 text, C3 text, C4 text, C5 text, C6 text, C7 text, C8 text, C9 text); | Supported  Success  53 ms |
| **3** | Import test data  (total data size : 500GB) | No errors reported | cassandra-stress write n=50000000 -pop dist=UNIFORM\(1..50000000\) -col n=FIXED(10) size=FIXED(1000) -mode native cql3 -schema keyspace="keyspace1" -rate threads=96 -node 172.31.30.83,172.31.30.84,172.31.30.85,172.31.30.87,172.31.30.80 | Supported  Success  67 minutes |
| **4** | Execute Write (put) operation  (value size : 10KB, TTL : 3600 sec) | No errors reported | INSERT INTO keyspace1.standard1 (key,"C0","C1","C2","C3","C4","C5","C6","C7","C8","C9") VALUES (0x50313032354f31303636,0x0c…) USING TTL 3600; | Supported  Success  1.6 ms |
| **5** | Execute Read (get) operation | No errors reported  Expected value read | SELECT \* FROM keyspace1.standard1 WHERE key=0x50313032354f31303636; | Supported  Success  3.4 ms |
| **6** | Execute Read (get) operation  (After TTL period : 3600 sec) | No errors reported  Expected value read | SELECT \* FROM keyspace1.standard1 WHERE key=0x50313032354f31303636; | Supported  Success  2.1 ms |
|  | | | | |
| **Detailed Test Result** | | | | |
| * Success/fail * Support/non-support | | | | |

## Case 6

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Scenario** | | | | |
| **Case6> Upsert** | * Check if upsert is supported | | | |
| **#** | **Steps** | **Expected Result** | **Perform Method** | **Actual Result** |
| **1** | Create a 5 node cluster | Database created | CREATE KEYSPACE keyspace1 WITH replication = {'class': 'SimpleStrategy' , 'replication\_factor': 3 }; | Supported  Success  56 ms |
| **2** | [Optional] Create a table | Table created | CREATE TABLE keyspace1.standard1 (key text PRIMARY KEY, C0 text, C1 text, C2 text, C3 text, C4 text, C5 text, C6 text, C7 text, C8 text, C9 text); | Supported  Success  53 ms |
| **3** | Import test data  (total data size : 500GB) | No errors reported | cassandra-stress write n=50000000 -pop dist=UNIFORM\(1..50000000\) -col n=FIXED(10) size=FIXED(1000) -mode native cql3 -schema keyspace="keyspace1" -rate threads=96 -node 172.31.30.83,172.31.30.84,172.31.30.85,172.31.30.87,172.31.30.80 | Supported  Success  67 minutes |
| **4** | Execute Write (put)  (value size : 10KB) | No errors reported | INSERT INTO keyspace1.standard1 (key,"C0","C1","C2","C3","C4","C5","C6","C7","C8","C9") VALUES (0x50313032354f31303640,0x0c6…); | Supported  Success  3.5 ms |
| **5** | Execute Read (get) operation | No errors reported  Expected value read | SELECT \* FROM keyspace1.standard1 WHERE key=0x50313032354f31303640; | Supported  Success  1.5 ms |
| **6** | Upsert a existing record | No errors reported, Response time | INSERT INTO keyspace1.standard1 (key,"C0","C1") VALUES (0x50313032354f31303640,0x0c…); | Supported  Success  1.3 ms |
| **7** | Execute Read (get) operation | No errors reported  Expected value read | SELECT \* FROM keyspace1.standard1 WHERE key=0x50313032354f31303640; | Supported  Success  1.5 ms |
| **8** | Upsert a new record | No errors reported, Response time | INSERT INTO keyspace1.standard1 (key,"C0","C1") VALUES (0x50313032354f31303641,0x0c...); | Supported  Success  1.3 ms |
| **9** | Execute Read (get) operation | No errors reported  Expected value read | SELECT \* FROM keyspace1.standard1 WHERE key=0x50313032354f31303641; | Supported  Success  1.4 ms |
|  | | | | |
| **Detailed Test Result** | | | | |
| * Success/fail * Support/non-support * Operation (Upsert) response time : ?? Sec | | | | |

## Case 7.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Scenario** | | | | |
| **Case7>**  **Query Support** | * Schema example   - UserID(KEY) GameTitle TopSocre LastUpdateTime CompanyName | | | |
| **#** | **Steps** | **Expected Result** | **Perform Method** | **Actual Result** |
| **1** | Create a 5 node cluster | Database created | CREATE KEYSPACE games WITH replication = {'class': 'SimpleStrategy' , 'replication\_factor': 3 }; | Supported  Success  56 ms |
| **2** | [Optional] Create a table | Table created | CREATE TABLE games.scores (  user\_id text,  game\_id text,  high\_score int,  last\_updated timestamp,  company text,  PRIMARY KEY(user\_id,game\_id)  ); | Supported  Success  53 ms |
| **3** | Import test data  (total data size : 500GB) | No errors reported | cassandra-stress write n=50000000 -pop dist=UNIFORM\(1..50000000\) -col n=FIXED(10) size=FIXED(1000) -mode native cql3 -schema keyspace="keyspace1" -rate threads=96 -node 172.31.30.83,172.31.30.84,172.31.30.85,172.31.30.87,172.31.30.80 | Supported  Success  67 minutes |
| **4** | Execute Query | No errors reported, Response time | Select \* from games.scores WHERE user\_id=191 AND game\_id=36; | Supported  Success  5 ms |
|  | | | | |
| **Detailed Test Result** | | | | |
| * Refer to next page. | | | | |

## Case 7.2

**Base Settings**

The given key definition of this test case assumes relational database behavior in the database engine in order to meet the query use cases. Cassandra is not a relational database engine. As such, the given key definition does not allow Cassandra to meet the query logic. See optimization settings below.

**Optimization Settings**

To provide a view of the available query models for solving this use case, the DataStax team chose two solutions to highlight in our response. The first solution is a DSE Search solution that provides very flexible query capabilities with the tradeoff of longer query latencies. The second solution uses CQL queries against denormalized Cassandra tables. This design provide faster query latencies with the tradeoff of less query flexibility. Both solutions are provided below.

|  |  |  |
| --- | --- | --- |
|  | **Query** | |
| 1 | = | 1. SELECT UserID, TopScore FROM Record where “UserID = [id] and GameTitle = [title]” ORDER BY TopScore [ DESC | ASC]  2. SELECT GameTitle, UserID FROM Record WHERE GameTitle = [title]  3. SELECT GameTitle, TopScore FROM Record WHERE GameTile = [title] ORDER BY GameTitle, TopScore [DESC | ASC] |
| 2 | > | 1. SELECT \* from Record WHERE LastUpdateTime > [date\_time] |
| 3 | < | 1. SELECT \* from Record WHERE LastUpdateTime < [date\_time] |
| 4 | >= | 1. SELECT \* from Record WHERE LastUpdateTime >= [date\_time] |
| 5 | <= | 1. SELECT \* from Record WHERE LastUpdateTime <= [date\_time] |
| 6 | between | 1. SELECT \* from Record WHERE LastUpdateTime between [date\_time\_start] and [date\_time\_end] |
| 7 | like | 1. SELECT \* from Record WHERE GameTitle like ‘space%’ |

**Case 7.2 Solution 1: DSE Search**

The first Case 7.2 results show the outcome of a DSE Search solution where the data in Cassandra has been automatically indexed upon CQL insert in a Lucene index distributed across the cluster. The distributed index gains all the benefits of the Cassandra architecture (replication, linear scale out, 100% uptime, etc).

|  |  |  |
| --- | --- | --- |
| TEST 1  1)  2)  3) | SELECT user\_id, high\_score FROM games.scores WHERE solr\_query='{"q":"user\_id:191", "q":"game\_id:36", "sort":"high\_score asc"}';  select game\_id,user\_id from games.scores WHERE solr\_query='game\_id:36';  SELECT game\_id, high\_score FROM games.scores WHERE solr\_query='{"q":"game\_id:36", "sort":"game\_id asc, high\_score desc"}'; | 27ms  10ms  41ms |
| TEST 2 | select \* from games.scores WHERE solr\_query='last\_updated:{2015-12-28T09:58:31Z TO \*}'; | 10ms |
| TEST 3 | select \* from games.scores WHERE solr\_query='last\_updated:{\* TO 2015-12-28T09:58:31Z}'; | 10ms |
| TEST 4: | select \* from games.scores WHERE solr\_query='last\_updated:[2015-12-28T09:58:31Z TO \*]'; | 10ms |
| TEST 5: | select \* from games.scores WHERE solr\_query='last\_updated:[\* TO 2015-12-28T09:58:31Z]'; | 10ms |
| TEST 6: | select \* from games.scores WHERE solr\_query='last\_updated:[2013-12-28T09:58:31Z TO 2014-12-28T09:58:31Z]'; | 15ms |
| TEST 7: | select \* from games.scores WHERE solr\_query='company:"~Blanda"'; | 10ms |

**Case 7.2 Solution 2: Denormalized Cassandra Query Tables**

For optimal query performance for these use cases in Cassandra, we recommend you follow Cassandra data modeling best practices by denormalizing the data into multiple query tables in order to meet all queries. Cassandra is very efficient and low latency at querying slices of time series data sets if modeled and queried correctly. For example, SELECT \* from Record WHERE LastUpdateTime > [date\_time] is a time series query that would get a very low latency result by using a Cassandra table with LastUpdateTime as a clustering key.

|  |  |  |
| --- | --- | --- |
| TEST 1  1)  2)  3) | SELECT user\_id, high\_score FROM games.scores\_by\_usergame\_a WHERE user\_id=? AND game\_id=?;  SELECT game\_id, user\_id FROM games.scores\_by\_game\_d WHERE game\_id=?;  SELECT game\_id, high\_score FROM games.scores\_by\_game\_d WHERE game\_id=?; | 11ms  9ms  4ms |
| TEST 2 | SELECT \* from games.scores\_by\_date\_d WHERE last\_updated>?; | 5ms |
| TEST 3 | SELECT \* from games.scores\_by\_date\_a WHERE last\_updated<?; | 5ms |
| TEST 4: | SELECT \* from games.scores\_by\_date\_d WHERE last\_updated>=?; | 4ms |
| TEST 5: | SELECT \* from games.scores\_by\_date\_a WHERE last\_updated<=?; | 5ms |
| TEST 6: | SELECT \* from games.scores\_by\_date\_d WHERE last\_updated>? AND last\_updated<?; | 7ms |
| TEST 7: | Unsupported – “like” queries only supported with DSE Search | N/A |

Below are the data model and query settings for the denormalized Cassandra query results above.

DATA MODEL

CREATE KEYSPACE games WITH replication = {'class': 'SimpleStrategy', 'replication\_factor': '1'} AND durable\_writes = true;

CREATE TABLE games.scores\_by\_game\_d (

game\_id text,

high\_score int,

company text,

last\_updated timestamp,

user\_id text,

PRIMARY KEY (game\_id, high\_score)

) WITH CLUSTERING ORDER BY (high\_score DESC)

AND bloom\_filter\_fp\_chance = 0.01

AND caching = '{"keys":"ALL", "rows\_per\_partition":"NONE"}'

AND comment = ''

AND compaction = {'class': 'org.apache.cassandra.db.compaction.SizeTieredCompactionStrategy'}

AND compression = {'sstable\_compression': 'org.apache.cassandra.io.compress.LZ4Compressor'}

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 = '99.0PERCENTILE';

CREATE TABLE games.scores\_by\_game\_a (

game\_id text,

high\_score int,

company text,

last\_updated timestamp,

user\_id text,

PRIMARY KEY (game\_id, high\_score)

) WITH CLUSTERING ORDER BY (high\_score ASC)

AND bloom\_filter\_fp\_chance = 0.01

AND caching = '{"keys":"ALL", "rows\_per\_partition":"NONE"}'

AND comment = ''

AND compaction = {'class': 'org.apache.cassandra.db.compaction.SizeTieredCompactionStrategy'}

AND compression = {'sstable\_compression': 'org.apache.cassandra.io.compress.LZ4Compressor'}

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 = '99.0PERCENTILE';

CREATE TABLE games.scores (

user\_id text,

game\_id text,

last\_updated timestamp,

company text,

high\_score int,

PRIMARY KEY (user\_id, game\_id, last\_updated)

) WITH CLUSTERING ORDER BY (game\_id ASC, last\_updated ASC)

AND bloom\_filter\_fp\_chance = 0.01

AND caching = '{"keys":"ALL", "rows\_per\_partition":"NONE"}'

AND comment = ''

AND compaction = {'class': 'org.apache.cassandra.db.compaction.SizeTieredCompactionStrategy'}

AND compression = {'sstable\_compression': 'org.apache.cassandra.io.compress.LZ4Compressor'}

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 = '99.0PERCENTILE';

CREATE TABLE games.scores\_by\_usergame\_d (

game\_id text,

user\_id text,

high\_score int,

company text,

last\_updated timestamp,

PRIMARY KEY ((game\_id, user\_id), high\_score)

) WITH CLUSTERING ORDER BY (high\_score DESC)

AND bloom\_filter\_fp\_chance = 0.01

AND caching = '{"keys":"ALL", "rows\_per\_partition":"NONE"}'

AND comment = ''

AND compaction = {'class': 'org.apache.cassandra.db.compaction.SizeTieredCompactionStrategy'}

AND compression = {'sstable\_compression': 'org.apache.cassandra.io.compress.LZ4Compressor'}

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 = '99.0PERCENTILE';

CREATE TABLE games.scores\_by\_date\_a (

dummykey text,

last\_updated timestamp,

company text,

game\_id text,

high\_score int,

user\_id text,

PRIMARY KEY (dummykey, last\_updated)

) WITH CLUSTERING ORDER BY (last\_updated ASC)

AND bloom\_filter\_fp\_chance = 0.01

AND caching = '{"keys":"ALL", "rows\_per\_partition":"NONE"}'

AND comment = ''

AND compaction = {'class': 'org.apache.cassandra.db.compaction.SizeTieredCompactionStrategy'}

AND compression = {'sstable\_compression': 'org.apache.cassandra.io.compress.LZ4Compressor'}

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 = '99.0PERCENTILE';

CREATE TABLE games.scores\_by\_usergame\_a (

game\_id text,

user\_id text,

high\_score int,

company text,

last\_updated timestamp,

PRIMARY KEY ((game\_id, user\_id), high\_score)

) WITH CLUSTERING ORDER BY (high\_score ASC)

AND bloom\_filter\_fp\_chance = 0.01

AND caching = '{"keys":"ALL", "rows\_per\_partition":"NONE"}'

AND comment = ''

AND compaction = {'class': 'org.apache.cassandra.db.compaction.SizeTieredCompactionStrategy'}

AND compression = {'sstable\_compression': 'org.apache.cassandra.io.compress.LZ4Compressor'}

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 = '99.0PERCENTILE';

CREATE TABLE games.scores\_by\_date\_d (

dummykey text,

last\_updated timestamp,

company text,

game\_id text,

high\_score int,

user\_id text,

PRIMARY KEY (dummykey, last\_updated)

) WITH CLUSTERING ORDER BY (last\_updated DESC)

AND bloom\_filter\_fp\_chance = 0.01

AND caching = '{"keys":"ALL", "rows\_per\_partition":"NONE"}'

AND comment = ''

AND compaction = {'class': 'org.apache.cassandra.db.compaction.SizeTieredCompactionStrategy'}

AND compression = {'sstable\_compression': 'org.apache.cassandra.io.compress.LZ4Compressor'}

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 = '99.0PERCENTILE';

FULL LIST OF QUERIES FOR CASE 7.2 Test 2

1)

a1) SELECT user\_id, high\_score FROM games.scores\_by\_usergame\_a WHERE user\_id=? AND game\_id=?;

11ms

a2) SELECT user\_id, high\_score FROM games.scores\_by\_usergame\_d WHERE user\_id=? AND game\_id=?;

16ms

b) SELECT game\_id, user\_id FROM games.scores\_by\_game\_d WHERE game\_id=?;

9ms

c1) SELECT game\_id, high\_score FROM games.scores\_by\_game\_d WHERE game\_id=?;

4ms

c2) SELECT game\_id, high\_score FROM games.scores\_by\_game\_a WHERE game\_id=?;

3ms

2) SELECT \* from games.scores\_by\_date\_d WHERE last\_updated>?;

5ms

3) SELECT \* from games.scores\_by\_date\_a WHERE last\_updated<?;

5ms

4) SELECT \* from games.scores\_by\_date\_d WHERE last\_updated>=?;

5ms

5) SELECT \* from games.scores\_by\_date\_a WHERE last\_updated<=?;

5ms

6) SELECT \* from games.scores\_by\_date\_d WHERE last\_updated>? AND last\_updated<?;

5ms

7) “like” query only supported through DSE Search.

## Case 8

**Base Settings**

Please see the updated results of Case 8 below.

In addition to Case 8 results below, we are providing you a previously run test case to demonstrate a backup on a live cluster, across EC2 regions, with significant load on the cluster at the time. This is the true value of OpsCenter's backup service, and this past test highlights this fact. What follows are details of this past test. Please let us know if you have any further questions, and we look forward to sharing our results for test case 8:

**Description of the test**

https://ssl.gstatic.com/ui/v1/icons/mail/images/cleardot.gif

As part of our real world simulation, the DataStax Test Engineering team conducted extensive tests of our Backup Services.

Using AWS and m1.xlarge [instances](http://aws.amazon.com/ec2/previous-generation/" \t "_blank), they created a 20 node DSE 4.6 cluster. The nodes were configured into 2 data centers of 10 nodes each.  One of the data centers consisted of pure Cassandra nodes, while the other data center consisted Cassandra and Spark nodes. For this test the cluster was configured to have 3 replicas in the Cassandra data center and 2 in the Analytics/Spark data center.

As part of cluster preparation, they loaded 7.7B entries (5.3 TB of data). In order to test a real world scenario, the following workload was running on the system as the backup part was initiated. 

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Cases | Reads | Creates | Deletes | Updates |
| Data Access Pattern | 20% | 30% | 30% | 20% |

At this time, a cross data center backup to Amazon S3 instance residing in a different region was started in parallel, without any throttling and this operation was run for over 2 days. While the backup & the load test were running in parallel, several OpsCenter metrics were collected and analyzed to determine the overall effect on the cluster and they found out that running backups had a very minimal effect on the performance of the cluster.

Please note that the backup took roughly ~1.5 days to complete and in this scenario the data was backed to a different region and during the time of backups, there was operational workload on the cluster. The test results will significantly vary if the backup region is the same as the region where the DC runs, workload patterns based on your use case and finally the instance type that you have chosen to run your cluster.

For further details, please look @ the blog [http://www.datastax.com/2015/03/datastax-opscenter-5-1-real-world-test-simulation-of-visual-backup-and-restore-services](http://www.datastax.com/2015/03/datastax-opscenter-5-1-real-world-test-simulation-of-visual-backup-and-restore-services" \t "_blank)

As part of backup and restore strategy, the data was restored back to the cluster without any data loss. The restore speed depends on a number of factors including your S3 region. DataStax Opscenter backup has Incremental and PIT backup/restore options that are available to ensure that you can restore your data to the nearest ms based on commit log archiving. Furthermore, you can also perform cloning of data across multiple clusters using OpsCenter backup feature.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Scenario** | | | | | |
| **Case8> Backup/**  **Restore** | * DB Backup and Restore functionality test | | | |
| **#** | **Steps** | **Expected Result** | **Perform Method** | **Actual Result** |
| **1** | Create a 5 node cluster | Database created | CREATE KEYSPACE testks WITH replication = {'class': 'SimpleStrategy' , 'replication\_factor': 3 }; | Supported  Success  56 ms |
| **2** | [Optional] Create a table | Table created | CREATE TABLE testks.testtable\_tenfields (key text PRIMARY KEY, C0 text, C1 text, C2 text, C3 text, C4 text, C5 text, C6 text, C7 text, C8 text, C9 text); | Supported  Success  53 ms |
| **3** | Import test data  (total data size : 500GB) | No errors reported  Total data count/size | java –jar testclient.jar --host=172.31.9.54 --activity=write-tenfields-seq:50000000:100:200 –splitcycles    172.31.9.55 304.22 GB    172.31.9.54 265.23 GB    172.31.9.57 246.3 GB    172.31.9.56 311.96 GB    172.31.9.58 301.55 GB | Supported  Success  67 minutes |
| **4** | Execute backup | No errors reported  File location/size, Execution time | Done two ways:   * nodetool snapshot copied to remote server   OpsCenter backup to S3 | Supported  Success  70 mins snapshot  4 hours OpsCenter |
| **5** | Initialize a new 5 node cluster | No errors reported | Delete cluster info and data, restart as fresh cluster | Success |
| **6** | Check stored data status | No data found | Nodetool status | Success |
| **7** | [Optional] Create a table | Table created | CREATE TABLE testks.testtable\_tenfields (key text PRIMARY KEY, C0 text, C1 text, C2 text, C3 text, C4 text, C5 text, C6 text, C7 text, C8 text, C9 text); | Supported  Success  53 ms |
| **8** | Execute restore with backup data | No errors reported  Total data count/size, Execution time | sstableloader  172.31.9.55 599.49 GB  172.31.9.54 785.59 GB  172.31.9.57 730.35 GB  172.31.9.56 641.42 GB  172.31.9.58 592.3 GB | Supported  Success  4 hours  \*note: the sstableloader, when used for restores, necessarily loads 3x the amount of data on each node, as it streams each backup’s token range to each node, resulting in duplicates. These are compacted out, but nodes are functional as soon as the load is complete, it does not have to wait for data to be compacted down.. Other restore methods do not cause excess data storage, but take longer. This is why you see a larger total data volume shortly after the restore |
|  | | | | | |
| **Detailed Test Result** | | | | | |
| * Backup file size(backup) : [Total: ?? MB] * Backup execution time : ?? sec * Restore execution : ?? sec * Imported data count/size (in Step#3) : [Total: ?? MB, count] [Node 1: ?? MB, count] [Node 2: ?? MB, count] … [Node 5: ?? MB, count] * Restored data count/size (in Step#8) : [Total: ?? MB, count] [Node 1: ?? MB, count] [Node 2: ?? MB, count] … [Node 5: ?? MB, count] | | | | | |

## Case 9

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Scenario** | | | | |
| **Case9 > Dynamic Scale-in/out** | * Scale-in/out functionality test | | | |
| **#** | **Steps** | **Expected Result** | **Perform Method** | **Actual Result** |
| **1** | Create a 5 node cluster | Database created | CREATE KEYSPACE keyspace1 WITH replication = {'class': 'SimpleStrategy' , 'replication\_factor': 3 }; | Supported  Success  56 ms |
| **2** | [Optional] Create a table | Table created | CREATE TABLE keyspace1.standard1 (key text PRIMARY KEY, C0 text, C1 text, C2 text, C3 text, C4 text, C5 text, C6 text, C7 text, C8 text, C9 text); | Supported  Success  53 ms |
| **3** | Import test data  (total data size : 500GB) | No errors reported  Total data count/size | cassandra-stress write n=50000000 -pop dist=UNIFORM\(1..50000000\) -col n=FIXED(10) size=FIXED(1000) -mode native cql3 -schema keyspace="keyspace1" -rate threads=96 -node 172.31.30.83,172.31.30.84,172.31.30.85,172.31.30.87,172.31.30.80 | Supported  Success  67 minutes |
| **4** | Add 2 new nodes  (node#6, node#7) | No errors reported  Execution time (data rebalance time included) | Done | Success  Supported  17 minutes |
| **5** | Check stored data status | No errors reported  Total data count/size | ubuntu@ip-172-31-4-157:~$ nodetool status  Datacenter: Cassandra  =====================  Status=Up/Down  |/ State=Normal/Leaving/Joining/Moving  -- Address Load Tokens Owns Host ID Rack  UN 172.31.9.173 338.86 GB 64 ? da03a286-91fb-46a5-98b8-580554b136ae rack1  UN 172.31.9.172 286.96 GB 64 ? ccc99bb1-6379-4368-87b2-fbc537ed168d rack1  UN 172.31.9.171 310.03 GB 64 ? 694f3278-9667-463b-bf67-bb09802ccd8d rack1  UN 172.31.9.170 308.93 GB 64 ? 56920298-02a1-4f17-be12-d9c192d84f2b rack1  UN 172.31.4.157 226.85 GB 64 ? b960c45b-a5b0-40c5-9e0d-dbefff239a96 rack1  UN 172.31.4.158 271.58 GB 64 ? a800bc88-743d-469c-978b-2fb5c1f9f5d2 rack1  UN 172.31.9.179 304.78 GB 64 ? 8a8ce345-51c1-4ca0-b06e-d7fb4bf0113e rack1 | Success  Supported |
| **6** | Remove 2 nodes except added (node#1, node#2) | No errors reported  Execution time (data rebalance time included) | Done | Success  Supported  19 minutes |
| **7** | Check stored data status | No errors reported  Total data count/size | FINAL DATA SIZE  Datacenter: Cassandra  =====================  Status=Up/Down  |/ State=Normal/Leaving/Joining/Moving  -- Address Load Tokens Owns Host ID Rack  UN 172.31.9.173 313.53 GB 64 ? da03a286-91fb-46a5-98b8-580554b136ae rack1  UN 172.31.9.170 300.84 GB 64 ? 56920298-02a1-4f17-be12-d9c192d84f2b rack1  UN 172.31.4.157 296.45 GB 64 ? b960c45b-a5b0-40c5-9e0d-dbefff239a96 rack1  UN 172.31.4.158 301.35 GB 64 ? a800bc88-743d-469c-978b-2fb5c1f9f5d2 rack1  UN 172.31.9.179 307.33 GB 64 ? 8a8ce345-51c1-4ca0-b06e-d7fb4bf0113e rack1  The results above were after running nodetool cleanup. Data was slightly redistributed to nodes .157 and .158, because they only came in in the expansion step before. So their data set grew when the two other nodes exited the cluster and broadcasted their token ranges. As you can see, the 5 node cluster is always at somewhere around 300GB per node. |  |
|  | | | | |
| **Detailed Test Result** | | | | |
| * Execution time (in Step#4) : ?? sec * Execution time (in Step#6) : ?? sec * Imported data count/size (in Step#3) : [Total: ?? MB, count] [Node 1: ?? MB, count] [Node 2: ?? MB, count] … [Node 5: ?? MB, count] * Rebalanced data count/size (in Step#5) : [Total: ?? MB, count] [Node 1: ?? MB, count] [Node 2: ?? MB, count] … [Node 7: ?? MB, count] * Rebalanced data count/size (in Step#7) : [Total: ?? MB, count] [Node 3: ?? MB, count] [Node 4: ?? MB, count] … [Node 7: ?? MB, count] | | | | |

# Stability Test

## 24 Hour Stability Test

**Test Setup**

The node configuration used for this test was:

i2.2xlarge

8 vCPU

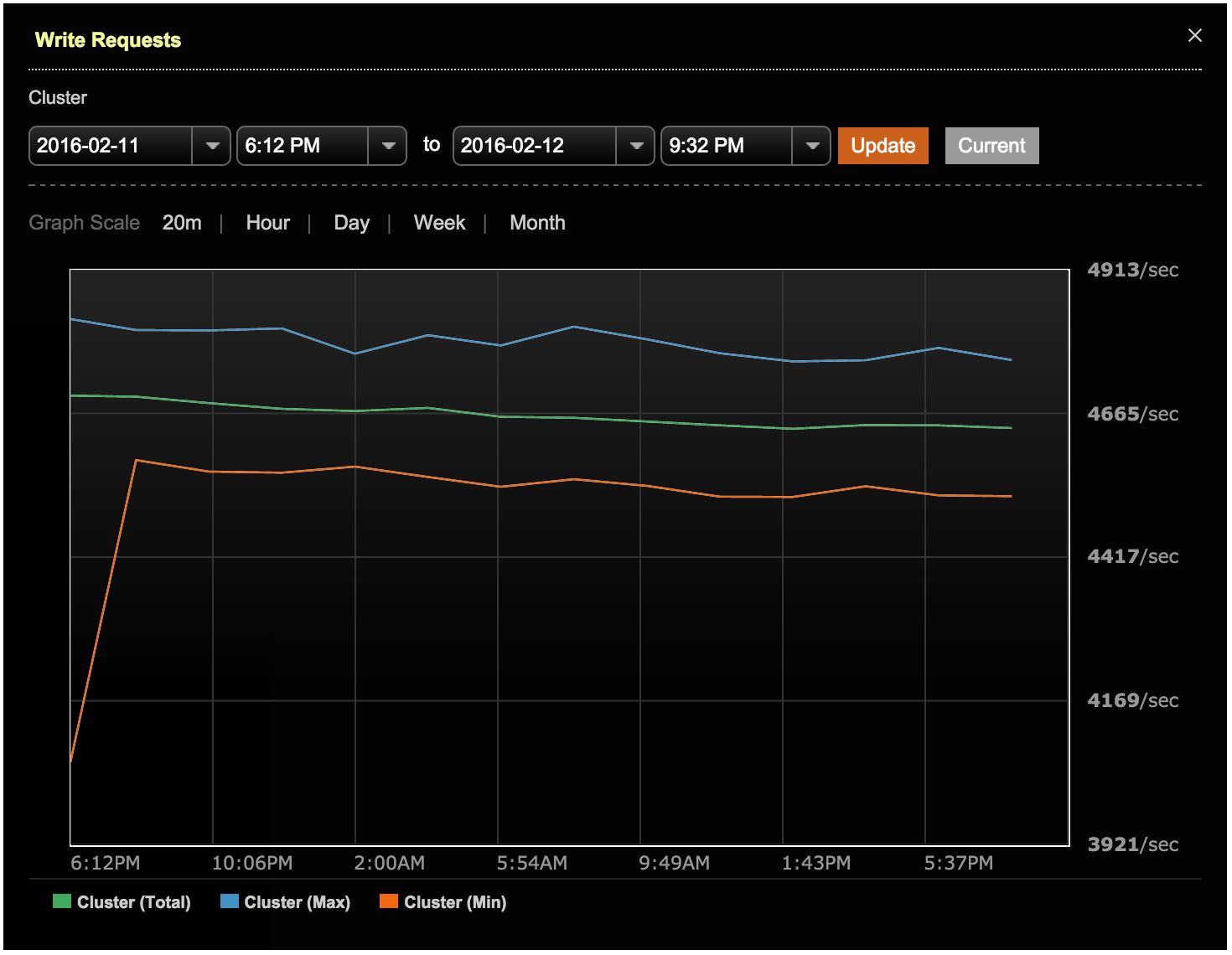
61 GB RAM

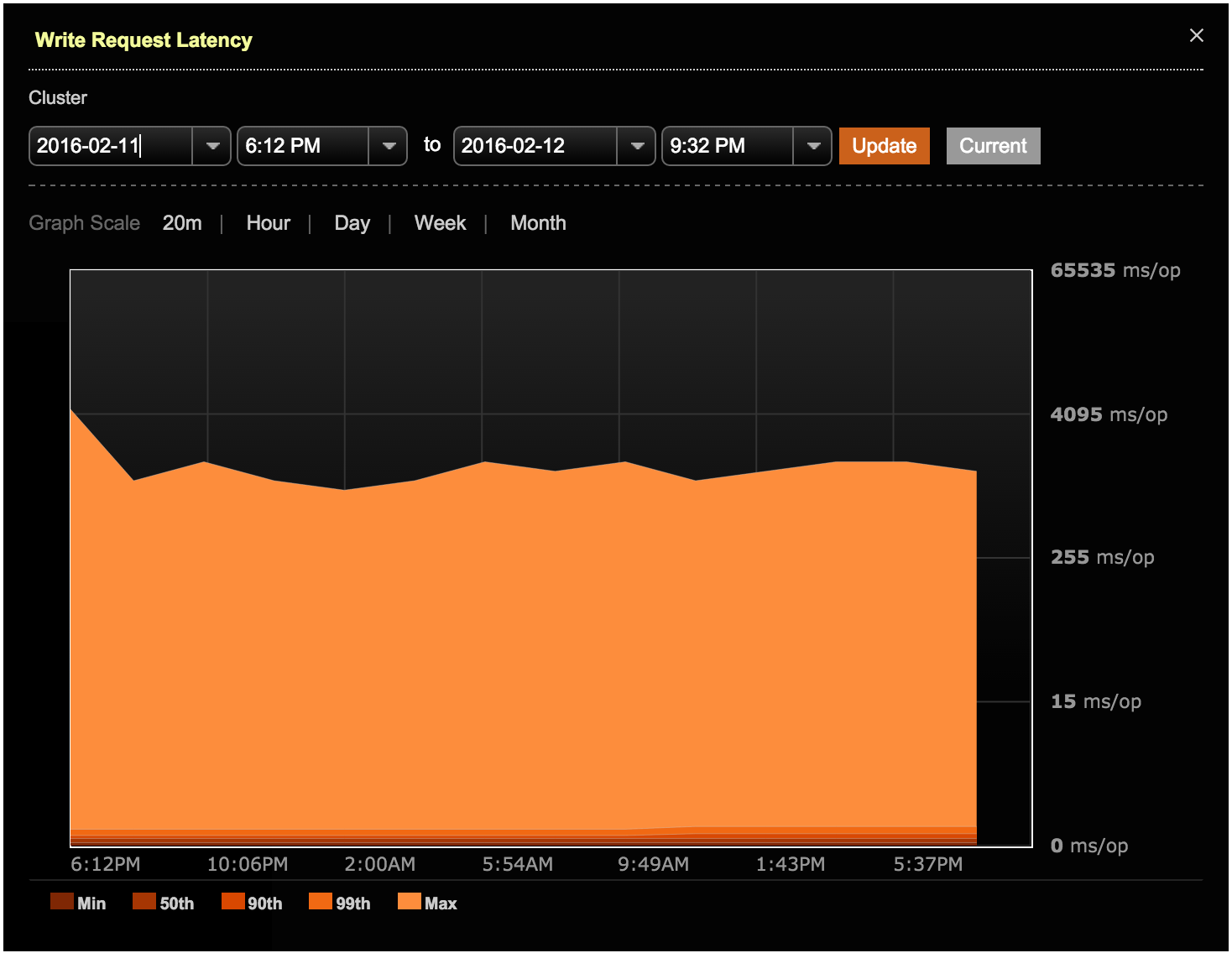
2 x 800 (SSD)

High network performance

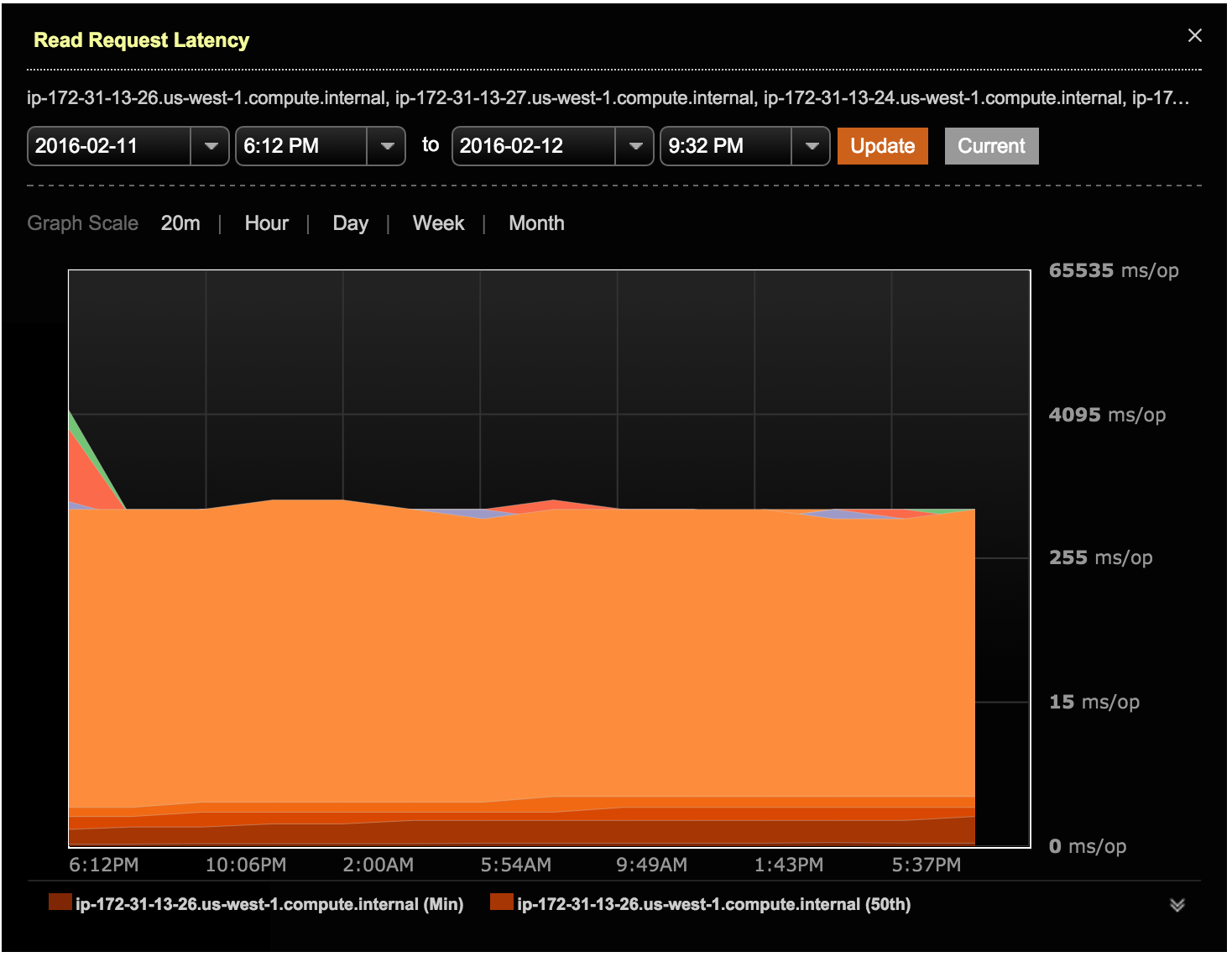
**Results**

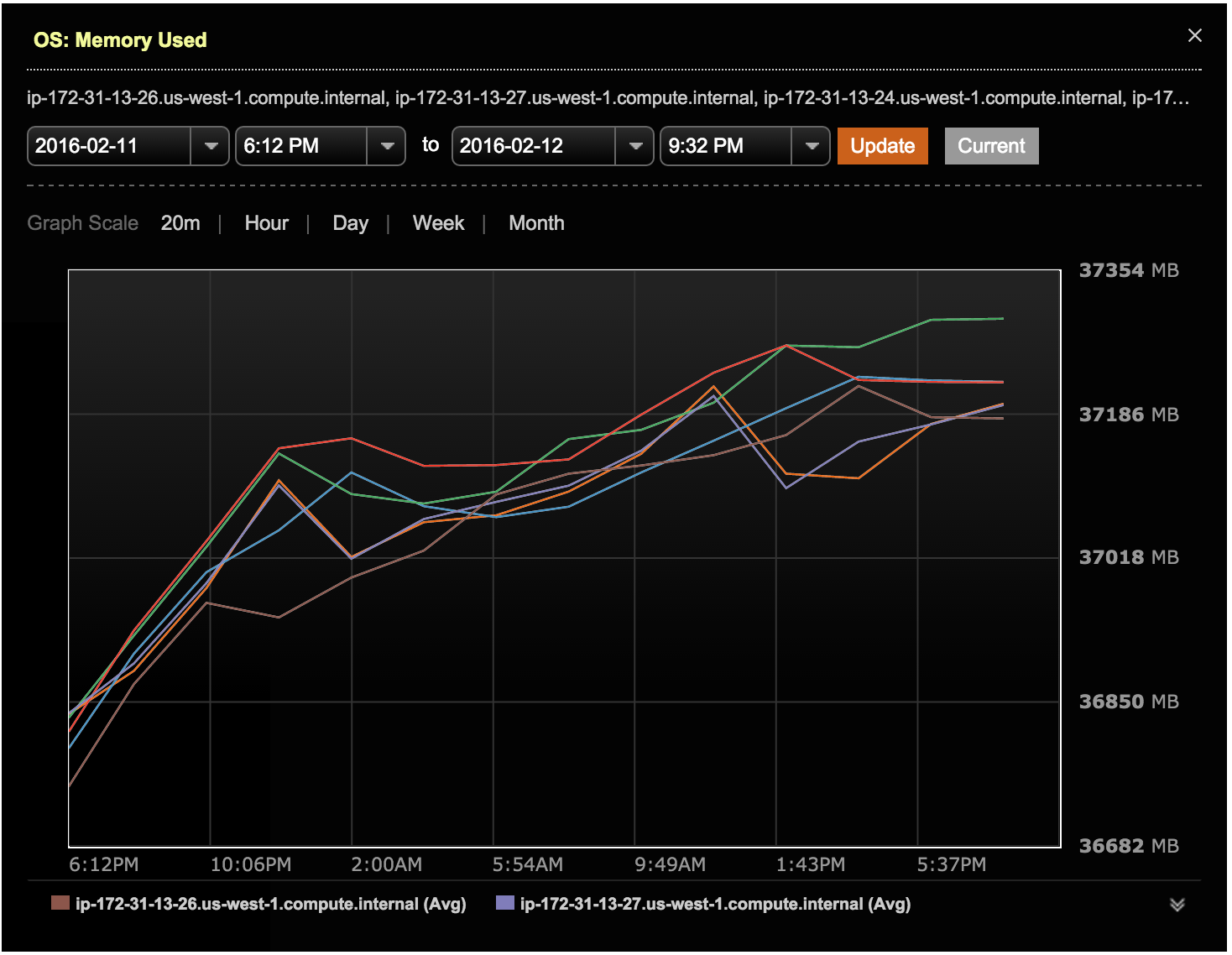
Please find below the test results for the Stability test.

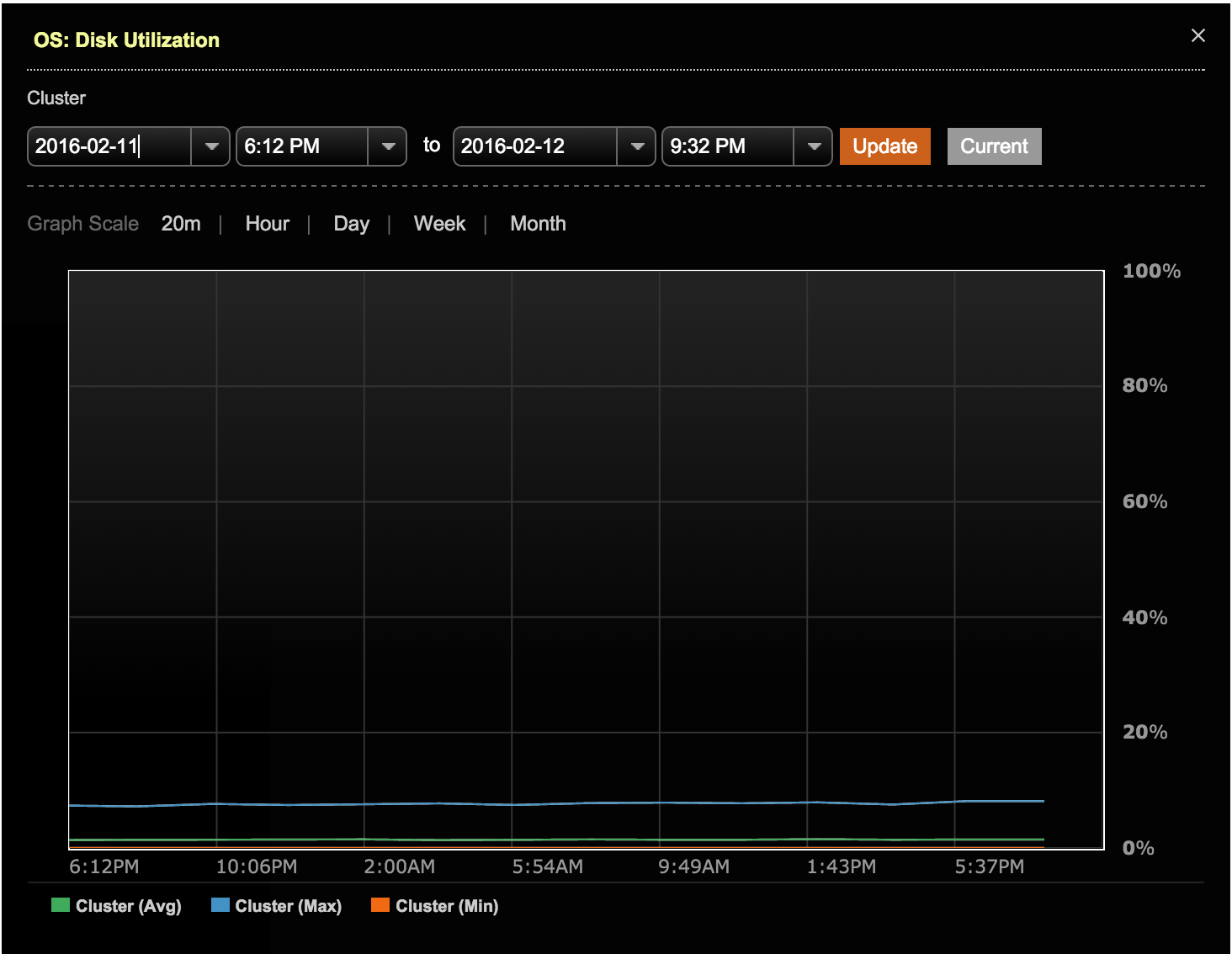














# Performance Test

## Test Environments

All test environments and steps were executed in line with the Performance Test specifications.

## Results

Please see attached the test results for each cluster configuration. There are three zip files, one for each tier of cluster size.

# Dynamo DB Comparison Questions

|  |  |
| --- | --- |
| **1** | **Cost Efficiency (how cost-effective your solution is compared to DynamoDB?)**  DSE Standard is priced on a per node basis.  Pricing is for a multi-year or single year subscription.   Samsung has a price of $US2,700 per node per year. List price is $US6,000 per node per year.   A node is defined as a JVM instance running on a Server or Virtual Server and running up to 12 physical cores.  Taking into consideration all the transaction processing and disk space requirements, the main node sizing factor that will determine the number of nodes is disk space.  Here is what we know with certainty.  For DSE we recommend 3TB of data per node running SSDs.  In order to store 700TB of data, which we calculated from your dataset size of 250B objects at 1k and a replication factor of 3, you will need 230 nodes.  The DSE software licensing costs to do that is $621,000 per year.  With these number of nodes, you will have plenty of transactional processing power.  To calculate DynamoDB costs we went to Amazon’s on-line calculator at: <http://calculator.s3.amazonaws.com/index.html>.  This seems to be a very complicated calculation, however after inputting 700TB of data and accounting for just 5TB of data coming into and going out of AWS daily, we calculated the approximate monthly cost of $446,000 per month or $5.35M per year on AWS with Dynamo DB.  The costs were mainly attributed to the 700TB of data.  These costs don’t include streaming or replication to another Region and any additional costs that we are just not aware.  I would take these numbers with caution as it is not very clear what all goes into this number, including which instances they are using, replication, data transfer to other Regions, etc.  The only conclusion I can make from this is that comparatively the cost of DSE is a fraction of the datacenter costs.  We know that the Samsung Milk team has optimized the disk and AWS instance running DSE.  We recommend looking into that to further reduce costs, making DSE a win with regards to pricing considerations in addition to the other DSE advantages.  Evan Maxey is a good contact.  Dynamo pricing assumptions   * data set size: 1k object metadata payload \* 250B objects = 232.8TB \* RF3 = 698.5TB * throughput: 500M / 24 / 60 / 60 = 5787 TPS / 2 = 2893.5 TPS for each workload of mixed reads and writes * Streams:  For this pricing exercise we did not include pricing.  But this could be a significant cost in the future. * Data Transfer Out/In: We assumed that 50% of data set per day would be active in transferring in or out.  699TB \* 0.5 = 350TB * Cross Region Replication: Dynamo uses Cross Region Replication to replicate data between regions. HOWEVER, it uses a Master/Replica model. The master exists in the local region. Replication to replicas in other regions is one-way *synchronously* using Streams (from local master to its remote replicas). Remote writes to remote replicas do not replicate back to the master. Local masters and local replicas (associated with remote masters) do not talk to each other. *In other words*, for an application to be able to locally read any data in the global data set (active/active), it must either [a] write every transaction directly to every master in every region or [b] write every transaction to the local master and all of the local replicas.  We are assuming the best practice is [b]. <http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Streams.CrossRegionRepl.html> |
| **2** | **High Availability (do you match all the HA features of DynamoDB and provide more?)**  DSE provides more flexibility, and better HA features than DynamoDB. DSE’s combination of best-in-class replication, tunable consistency of each and every operation, ease of cluster operations, and built-in anti-entropy mechanisms such as hinted handoff, read repair, and OpsCenter Repair Service provide superior HA capabilities as compared to Dynamo.  It’s not clear that cross region replication can be accomplished in DynamoDB with the simplicity of a single table definition at the same level DSE provides.   * DSE HA across regions is better. For a given multi-region active/active use case we expect DSE, compared to DynamoDB, to deliver the following benefits.   + Makes active/active data available faster and more reliably across regions   + Simplifies client app design   + Simplifies data modeling   + Simplifies database operations and lowers processing required per operation * Amazon DynamoDB synchronously replicates data across three facilities within an AWS Region only by default. This should not be confused with replication to other AWS Regions.   + DSE uses real time asynchronous replication across the entire cluster (all regions). This occurs at the level of granularity of each transaction. That means data inserted locally into DSE is available in remote regions faster and more reliably than other technologies.   + DynamoDB uses Cross-Region Replication, which is a synchronous, one-directional, batch oriented replication scheme using the AWS Streams feature (increased cost).   + A single DSE table is replicated seamlessly across the entire cluster.     - DSE provides lower latency local data access on the entire global data set. This is because DynamoDB must wait for synchronous replication Streams (batches) to finish before locally inserted data is available remotely. In this way, DSE gets your business data close to the end user compute as fast as possible compared to DynamoDB.     - DSE data modeling and client app design patterns are greatly simplified (see point below). With DSE your client application only needs to submit one write operation into the table in the local data center to insure the data is highly available across the entire cluster.   + DynamoDB uses a Master/Replica physical schema in which a local master copy of a table can only replicate one direction to remote replicas. Remote writes to remote replicas do not replicate back to the local master. Local masters and local replicas (associated with remote masters) do not talk to each other.     - While DataStax are not DynamoDB experts, our understanding is that a DynamoDB client application to be able to locally read any data in the global data set (active/active), it must either [a] write every transaction directly to every master in every region or [b] write every transaction to the local master and all of the local replicas.  We assume the best practice is [b].     - The DynamoDB design:       * Burdens the client app with managing HA active/active data replication across regions       * Makes DynamoDB client app code more complex       * Increases the number of database operations required to perform a single business transaction       * Drives up operational costs and complexity * DSE’s flexible tunable consistency allows you to adjust your queries to provide the right balance of consistency and performance. With Cassandra, you can tune your query operations to perform at optimal throughput and latency while still providing durable writes and good data on reads, avoiding dirty reads. Dynamo seems to offer either strong consistency or low latency as options, each of which drives up cost and cannot be tuned to optimal performance for your use case. * In a single data center, DSE delivers HA similar to Dynamo. Both technologies store multiple local region replicas of the data in case of a disaster scenario.   + Dynamo by default stores 3 copies across the region for HA and durability.   + DSE is data center and rack aware, meaning that you can distribute nodes anywhere you want in the cloud, on premise, or a hybrid of both, in such a way that no two replicas of a data element are in the same physical location. This provides physical distribution of the data to provide HA when an entire data center goes down. * DSE extends HA to more data access models in addition to Cassandra.   + DSE Search provides HA live data indexing of Cassandra data. Search processing is distributed just like Cassandra queries so that each node searches only its own local data set. All of Cassandra’s architecture benefits of linear scale, HA, and low latency extend to DSE’s ability to do a Lucene search on Cassandra cluster data.   + DSE Analytics provides HA Spark, Spark Streaming, and Hadoop on Cassandra data. DSE Analytics includes Spark master auto-election and failover. It does not require Zookeeper (Zookeeper is a single point of failure).   + DSE is expanding its functionality to include Graph data, time series, In-Memory and Hub/Spoke architectures, all of which will be Highly Available. |
| **3** | **Linear Scalability (how linear does your solution scale in/out?)**  DataStax Enterprise scales in and out linearly 1:1 with regards to data set size and/or throughput. For example, if a hypothetical DSE cluster of 5 nodes supports 125k TPS throughput and 15TB of total data set size for a particular use case (data model and access patterns), then 10 nodes will support 250k TPS / 30TB and 20 nodes will support 500k TPS / 60TB. This assumes each node is the same hardware specification.  Since all DSE query model capabilities use DSE as the data storage layer (DSE, Search, Analytics, In-Memory, Graph), all of these capabilities scale linearly as well.  Please see this study conducted by Netflix to prove DSE linear scalability and speed of native DSE replication. Also note that this test was conducted 5 years ago on a significantly slower version of DSE than is available today: [http://techblog.netflix.com/2011/11/benchmarking-DSE-scalability-on.html](http://techblog.netflix.com/2011/11/benchmarking-cassandra-scalability-on.html) |
| **4** | **Support of Data Hot/Cold separation**  DSE accommodates a number of options for hot/cold data separation.   * In use cases where Cassandra is not the most economical way to store cold data, it is common for DSE users to move cold data to HDFS, keep hot data in Cassandra, and use DSE Analytics Spark to read or write data to and from both Cassandra and HDFS, or any other location accessible by Spark. This, of course, is a solution for batch analytics access patterns. * DSE supports a direct two-way integration with Hadoop. This feature, called Bring Your Own Hadoop, allows you to query data from hadoop into DSE along with data from Cassandra. It also allows your Hadoop implementation to see DSE as a data source. * A feature that is currently slated for inclusion into DSE v5.0 is called Tiered Storage. This feature is expected to be implemented as a Cassandra compaction strategy. In this compaction strategy, multiple tiers of storage can be defined as paths and mounted to any location of your choice. As data ages and is compacted into lower levels of SSTables, the compaction strategy will migrate data across your defined tiers. Colder data can be stored in bulk storage while hot data might be stored on SSD. This migration is seamless as the process of compaction requires immutable SSTables to be merged into net new generations of new files and written to storage as background processing in Cassandra. On the write to storage, the compaction strategy will determine the correct data tier. As with any future planned feature, it must be noted that this feature may not be released into production or may be released with a different design than described here. This POC document is no guarantee or warrantee the feature will be available. |
| **5** | **Support of meta data: Count, last-modified-time, size, deleted count, min, max, avg**  **(do you support user defined meta data? Do you have built-in metadata for statistics?)**  DataStax OpsCenter provides extensive analytics, alerts, and metadata support for DSE operations. OpsCenter tracks a variety of data points and histograms for a large number of metrics. All OpsCenter capabilities are accessible from a REST API for integration of DSE metrics into your preferred operational monitoring system if desired.  The metrics available and list of monitoring options are too extensive to list in this document. Please see the metrics documented here: <https://docs.datastax.com/en/opscenter/5.2/opsc/online_help/opscPerformanceMetrics_c.html>  Example basic view of OpsCenter Monitoring  https://lh3.googleusercontent.com/5va9xVFsWmjh01K52N9FOH3bpxf5oiPVvmfE6VxMPIg_9_VAhtG1GT8qNrBYcj91c8u1206talkvvOgQg3X7oi0VIJB7a_bPE_tU5XMqx8p8v8reZtF7QtZsyjjVCFW-WVzJfNa- |
| **6** | **Provide access pattern of each table (access frequency of each table or similar data set)**  DSE Auditing provides a variety of metadata categorized into various buckets that can each be toggled on and off:   * ADMIN - Logs describe schema versions, cluster name, version, ring, and other administration events. * AUTH - Logs login events. * DML - Logs insert, update, delete and other events. * DDL - Logs object and user create, alter, drop, and other events. * DCL - Logs grant, revoke, create user, drop user, and list users events. * QUERY - Logs all queries.   The DSE Performance Service feature tracks and identifies slow running queries, stressed thread pools, and table level latency metrics in order to proactively identify and advise you of situations requiring optimization. See details here: <http://docs.datastax.com/en/opscenter/5.2/opsc/online_help/services/opscPerformanceService.html>  Cassandra Query tracing provides deep tracking of internal Cassandra operations supporting any CQL operation. Cassandra provides a description of each step it takes to satisfy the request, the names of nodes that are affected, the time for each step, and the total time for the request in microseconds. See below a query trace example:  https://lh5.googleusercontent.com/ms60fDxdT2ref-9YP_t01pEHflqFDhpuDlD7TgekxlbSXdl5s-629D7ksSCFX199tdYtYMfgc7qiXpYrZZlTTtnYmY6ysWvtCMMfNGfMgMmt_txCShtyArxfXeRUuE7zDxU2x9Mw |
| **7** | **Support of Complex Query (do you support complex query or plan to support the feature?)**  DSE supports multiple query models.   * Cassandra queries allow selects over a single or a set of partition keys and range queries over one or multiple clustering keys according to the clustering key hierarchy. An example is: [select \* from sentiment3 where ch = 'facebook' and cu = 'red bull' and dt >= 20160102 and dt <= 20160103;] * DSE Search queries embedded in Cassandra CQL3 support any query enabled by the DSE Search Lucene client. See this example [query: SELECT \* FROM amazon.metadata WHERE solr\_query='{"q":"\*:\*", "facet":{"field":"categories"}, "fq":"{!join from=asin to=asin force=true fromIndex=amazon.clicks}area\_code:415"}' limit 5;] A vast majority of Solr queries are possible through the CQL3 interface: <https://cwiki.apache.org/confluence/display/solr/The+Standard+Query+Parser> * DSE Analytics queries enable any query supported by the interface being used. An example query using Spark SQL is [SELECT asin, sum(price) AS max\_price FROM metadata GROUP BY asin ORDER BY max\_price DESC limit 1;]. |
| **8** | **Nested data structure (if support, how a user may use the feature?)**  DSE Cassandra support both UDTs and collections. Collections supported included sets, lists, and maps.   * See UDT details here: <http://docs.datastax.com/en/cql/3.1/cql/cql_reference/cqlRefUDType.html> and an example of usage here: <http://www.datastax.com/dev/blog/cql-in-2-1> * See collections details here: <http://docs.datastax.com/en/cql/3.1/cql/cql_using/use_collections_c.html>  The above blog also details the usage of secondary index on collections. For performance reasons, use secondary indexes in DSE Cassandra with care. |
| **9** | **Support of Data Encryption**  DSE supports Data Encryption for data in motion client to node and node to node via SSL.  Transparent encryption of data at rest in SSTables is provided natively in Cassandra. Encryption is very flexible, defined at the table level at time of table creation by specifying encryption parameters in the CREATE TABLE or ALTER TABLE CQL statement. Each node generates its own unique keys. Options for encryption algorithm are:  cipher\_algorithm secret\_key\_strength  AES/CBC/PKCS5Padding 128, 192, or 256  AES/ECB/PKCS5Padding 128, 192, or 256  DES/CBC/PKCS5Padding 56  DESede/CBC/PKCS5Padding 112 or 168  Blowfish/CBC/PKCS5Padding 32-448  RC2/CBC/PKCS5Padding 40-128  External key management is supported through the KMIP protocol for third party key management systems.  Encryption of the entire node file system for all DSE system components is provided by DataStax partners. More details can be provided by your technical contacts on request. |
| **10** | **History maintenance**  **(do you support logging per table or similar? If support, how a user can retrieve log info?)**  Please see the information above in question 5 regarding Auditing. |

# Additional Information

**Additional Request for Cluster Sizing Recommendations On Given Node Specifications**

Please assume the following input parameters while filling out the table.

\* No Use of AWS EBS

\* Replication factor: 3

\* HA: 99.99999%

\* Avg. Object Size: 10K

\* Total Number Of Stored Objects: 2.6B objects

\* Avg TPS: 200K TPS (read:write = 50:50)

|  |  |  |
| --- | --- | --- |
| AWS Instance Type | Minimum Number of Required Instances | Note |
| c3.8xlarge | 244 | There is a 640GB storage limit per node, and Cassandra needs 50% free for compaction. This means, with replication factor 3 and a total storage footprint of 2.6B objects, we need 78TB of storage space. 78000/320=244 nodes needed. This cluster node count is bound by data set size. |
| i2.4xlarge | 150 | Using an i2.4xlarge, and given our testing, we can assume that with 500GB of data per node, we can maintain 1334 tps per node, giving us a total of 150 nodes needed. This cluster node count is bound by throughput per node. |

[Note] the choice of best practice instance for your solution must be carefully picked so that while minimizing total number of nodes should NOT negatively impact previously guaranteed HA and performance SLA.

**Additional Information**

1. Company information (size, revenue, yearly growth for the past 5 years)

Founded in 2010, Datastax has 450 employees today and 100% year over year revenue growth the past 3 years.  DataStax is a Private Company and does not provide financials.  DataStax has over 500 customers, over 30 of top fortune 100. Our latest financial investment round was $106M bringing our company evaluation to $830M.

2. Additional best success stories that may relevant to Samsung or Samsung Cloud Service

Apple (public info only):

Over 100,000 nodes in production, 10PB+ of data.   From Apple job postings: iCloud, Maps, iTunes, and more looking for Cassandra experience.

Intel:

Deployed across Mobile Security App with 200M users, Cloud Platform, YAP,  MMS and backup/restore use cases.

Nike:

IOT/sensor data, messaging, persistence store, personalization use cases.  Examples are Nike Fuelband, Nike+Platform, Facebook like use cases, global inventory, shopping cart, and more.

Microsoft Office365:

Deployed for authentication, customer facing analytics, security platform, universal store and personalization use cases.

Sony:

Deployed in SNEI for My Library, Live Area, Notifications, Pre-order use cases.  Within Playstation it is deployed for authorization, recommendations, share, and remote downloads.

PayPal:

Transaction payment solutions and BrainTree product line.

Intuit, including Check, Mint, TurboTax, Quickbooks and Payments

Intuit has adopted DSE across the organiztion, including thousands of concurrent users using TurboTax to financial application from Payment transaction and also including Check and Mint.

ING:

Banking transactions, requiring high levels of data consistency.  Video available at [www.datatstax.com/customers](http://www.datatstax.com/customers).

Netflix:

Over 1 Trillion transactions per day.  100% within Amazon datacenters.  Video available at [www.datastax.com/customers](http://www.datastax.com/customers)

UBS Securities:

A global time series use case.  Video available at [www.datastax.com/customers](http://www.datastax.com/customers)

3. Previous benchmarking test results or performance analysis reports (if any)

[http://www.datastax.com/dev/blog/how-not-to-benchmark-DSE](http://www.datastax.com/dev/blog/how-not-to-benchmark-cassandra)

[http://www.datastax.com/dev/blog/how-not-to-benchmark-DSE-a-case-study](http://www.datastax.com/dev/blog/how-not-to-benchmark-cassandra-a-case-study)

4. Comparison study (vs. DynamoDB or any major competitor)

Please find in the DataStax Github s-cloud repository a powerpoint presentation that summarizes many differences between DSE and DynamoDB. Also find attached a Dynamo comparison to DSE. Both are valid as of as of later 2014.

We have also shared with you in the DataStax Github s-cloud repository a YCSB benchmark that was executed by an unbiased third party systems integrator comparing Cassandra to HBase.

5. Evaluation license for your enterprise edition (if you had not provided us yet)

Upon request, or can download from www.datastax.com

6. Main contact and technical contact

Dave Crawford +1 408 859 2789

Avinash Mandava +1 408 691 8402

Rich Reffner - +1 415 713 6673