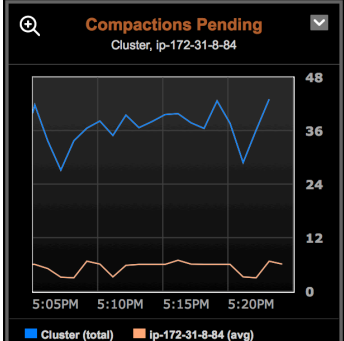
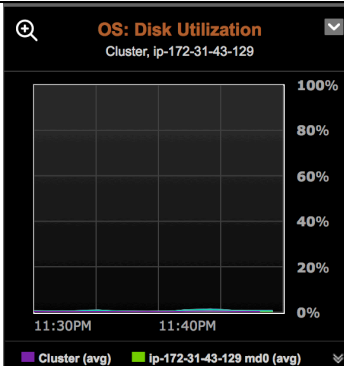
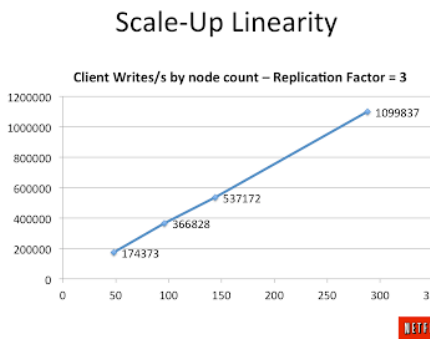
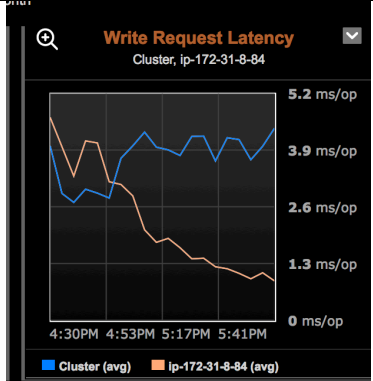


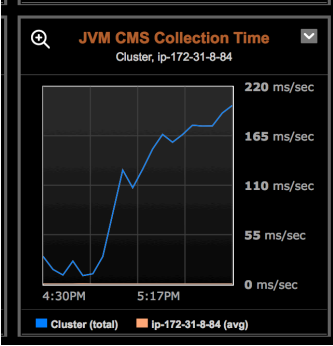
SCORECARD ITEMS

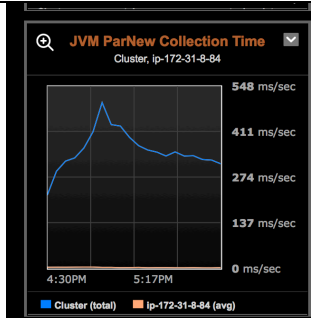
Graded items for Cassandra Data Model

Each item will be “Pass/Fail”

NUMBER	RESULT	MEASUREMENT	PASSING VALUE	SIGNIFICANCE	SCREENSHOT																								
1	FAIL	Pending compactions with steady “read only” load	< 15 at all times	High number of pending compactions has a negative impact on read performance because of I/O contention between SSTables and compacting them																									
2	PASS	Disk utilization with steady “writes only” load	< 40% most of the time. Never higher than 70%	We use writes only because reads tend to get drawn from cache																									
3	PASS	Scalability Steady state work load run against 3, 6 and 9 nodes	Throughput increases, but latency does not.	Although it is widely acknowledged that Cassandra scales in a linear fashion, this test provides concrete proof that the current project follows this pattern	<div>Scale-Up Linearity</div> <div>Client Writes/s by node count – Replication Factor = 3</div> 																								
4	PASS	TPSTATS Steady state load	No blocked processes or dropped messages	Dropped or blocked processes indicate a bottleneck in hardware or configuration tuning	<table><tr><td>Pool Name blocked</td><td>Active</td><td>Pending</td><td>Complete</td></tr><tr><td>ReadStage</td><td>0</td><td>0</td><td>8790069</td></tr><tr><td>RequestResponseStage</td><td></td><td>0</td><td>011077428</td></tr><tr><td>MutationStage</td><td>0</td><td>0</td><td>120064920</td></tr><tr><td>ReadRepairStage</td><td>0</td><td>0</td><td>547374</td></tr><tr><td>ReplicateOnWriteStage</td><td></td><td>0</td><td>0</td></tr></table>	Pool Name blocked	Active	Pending	Complete	ReadStage	0	0	8790069	RequestResponseStage		0	011077428	MutationStage	0	0	120064920	ReadRepairStage	0	0	547374	ReplicateOnWriteStage		0	0
Pool Name blocked	Active	Pending	Complete																										
ReadStage	0	0	8790069																										
RequestResponseStage		0	011077428																										
MutationStage	0	0	120064920																										
ReadRepairStage	0	0	547374																										
ReplicateOnWriteStage		0	0																										

					GossipStage0018308956 CacheCleanupExecutor0014 MigrationStage001230 MemoryMeter0019459 FlushWriter00458110 ValidationExecutor00206469 InternalResponseStage00206647 AntiEntropyStage001045519 MemtablePostFlusher00333805 MiscStage002064710 PendingRangeCalculator00157 commitlog_archiver000 CompactionExecutor33879111 AntiEntropySessions0010631 HintedHandoff00650 Message typeDropped RANGE_SLICE0 READ_REPAIR0 PAGED_RANGE0 BINARY0 READ0 MUTATION0 _TRACE0 REQUEST_RESPONSE0 COUNTER_MUTATION0
5	PASS	Write latency with steady state load	Flat or downward trend	Increasing latency indicates Commit log or Memtable could be overrun with write requests	

6		<p>CMS collections</p> <p>(Concurrent Mark Sweep)</p> <p>Steady state (any work load)</p>	No OldGen activity lasting > 5 min	OldGen garbage collection is expensive. If heap is not released consistently, excessive CMS activity can cause OutOfMemoryExceptions in Cassandra	
7		<p>Run Queries in cqlsh (shell) with TRACE enabled</p> <p>Can be run on any node from the cluster</p>	Query trace shows no more than 2 partitions used to get results for Read requests.	Using TRACE mode while executing a CQL query can expose inefficient query patterns and/or table designs	
8	FAIL	<p>Row sizes in cfhistogram</p> <p>Also dual hump latency is indicative of a problem</p>	Graduated column sizes (no one partition/row greatly outsizes the others)	A single partition that is much larger than any others is a sign of problems with client application and/or data model	<pre>[kratos@ip-172-31-28-94 ~]\$ nodetool -h 172.31.28.5 assessment_keyspace assessments_by_date assessment_keyspace/assessments_by_date histogram Partition Size (bytes) 3311 bytes: 9198 3973 bytes: 0 4768 bytes: 0 5722 bytes: 2794 6866 bytes: 0 8239 bytes: 0 9887 bytes: 1557 11864 bytes: 0 14237 bytes: 0 17084 bytes: 795 20501 bytes: 420 24601 bytes: 274 29521 bytes: 309 35425 bytes: 425 42510 bytes: 508 51012 bytes: 1124 61214 bytes: 967 73457 bytes: 678 88148 bytes: 188 105778 bytes: 24 126934 bytes: 0 152321 bytes: 0 182785 bytes: 0 219342 bytes: 0 263210 bytes: 0 315852 bytes: 0 379022 bytes: 0 454826 bytes: 0 545791 bytes: 0 654949 bytes: 0 785939 bytes: 0 943127 bytes: 0 1131752 bytes: 88 1358102 bytes: 334 1629722 bytes: 5 1955666 bytes: 0 2346799 bytes: 0 2816159 bytes: 0 3379391 bytes: 0 4055269 bytes: 0 4866323 bytes: 0 5839588 bytes: 1</pre>

					7007506 bytes: 49
9		Cassandra Anti-Patterns – Static analysis of source code, table schemas and keyspace definition	Review code to verify best usage of the Java driver, CQL3 and Cassandra features	AntiPatterns fall into different categories: <ul style="list-style-type: none"> - Deployment - Data model - Access Patterns 	Examples of Anti-patterns: (there are many more) <ul style="list-style-type: none"> - client side joins (using application code to join tables on a column) - nonUse of PreparedStatements - nonUse of async execution for multiple queries - multi-partition queries - misuse of batches (logged, unlogged) Batches should NOT be used to Insert large number of rows - Distributed mutable state - Excessive usage of Deletes / Use of Cassandra as a queue - Secondary Indexes on tables (should be used with caution) - Inappropriate read or write retry policy
10		Parnew Garbage collection	Should not last more than 1000 ms per 1 sec interval	Excessive ParNew collection indicates overload of the write paths and can lead to unresponsiveness	 <p>The graph displays 'JVM ParNew Collection Time' for a cluster. The y-axis represents time in milliseconds per second (ms/sec), with a scale from 0 to 548. The x-axis shows time from 4:30 PM to 5:17 PM. A blue line represents the 'Cluster (total)' and an orange line represents 'ip-172-31-8-84 (avg)'. Both lines show a significant peak around 4:45 PM, reaching approximately 548 ms/sec.</p>

In addition to the score, report will include bullet points to address the following questions:

Does our data model scale?

Usually “scale” means we can increase number of nodes without seeing an increase in latency. Do we want this kind of definition for a data model, or do we want to target something else?

Do we meet the defined sla? → Note: we don’t have SLAs defined for the database layer. Need to find something. Perhaps we can start with expected internal latency for C* deployed on SSDs and use our SLAs from the consuming application to derive something.

What is our breaking point?

What are we bound on (i/o, cpu, network, compaction, etc)

Cards that are created to account for any discoveries

Report will have an attachment containing details and artifacts from the test session, to make the test be more repeatable.

Sample data>

- 1) Table DDL obtain from OpsCenter or by ssh'ing to a node and using DESCRIBE KEYSPACE from the cqlsh prompt
- 2) CQL queries. Access patterns are part of the data model
- 2) Number of nodes
- 3) Deployed Machine types
- 4) Keyspace definition
- 5) Console printouts from the test tool
- 6) JVM arguments used for each test run
- 7) printout from command line utilities