Performance Evaluation of NoSQL Databases: A Case Study

John Klein, Ian Gorton, Neil Ernst, Patrick Donohoe
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA, USA
{jklein, igorton, nernst, pd}@sei.cmu.edu

Kim Pham, Chrisjan Matser
Telemedicine and Advanced Technology Research Center
US Army Medical Research and Material Command
Frederick, MD, USA
kim.solutionsit@gmail.com, cmatser@codespinnerinc.com

ABSTRACT
The choice of a particular NoSQL database imposes a specific distributed software architecture and data model, and is a major determinant of the overall system throughput. NoSQL database performance is in turn strongly influenced by how well the data model and query capabilities fit the application use cases, and so system-specific testing and characterization is required. This paper presents a method and the results of a study that selected among three NoSQL databases for a large, distributed healthcare organization. While the method and study considered consistency, availability, and partition tolerance (CAP) tradeoffs, and other quality attributes that influence the selection decision, this paper reports on the performance evaluation method and results. In our testing, a typical workload and configuration produced throughput that varied from 225 to 3200 operations per second between database products, while read operation latency varied by a factor of 5 and write latency by a factor of 4 (with the highest throughput product delivering the highest latency). We also found that achieving strong consistency reduced throughput by 10-25% compared to eventual consistency.

Categories and Subject Descriptors
C.4 [Performance of Systems]: Measurement techniques, Performance attributes,

General Terms: Performance, Measurement.

Keywords: Performance, NoSQL, Big Data

1. INTRODUCTION
COTS product selection has been extensively studied in software engineering [1][2][3]. In complex technology landscapes with multiple competing products, organizations must balance the cost and speed of the technology selection process against the fidelity of the decision [4]. While there is rarely a single 'right' answer in selecting a complex component for an application, selection of inappropriate components can be costly, reduce downstream productivity due to rework, and even lead to project cancelation. This is especially true for large scale, big data systems due to their complexity and the magnitude of the investment.

In this context, COTS selection of NoSQL databases for big data applications presents several unique challenges:

- This is an early architecture decision that must inevitably be made with incomplete knowledge about requirements;
- The capabilities and features of NoSQL products vary widely, making generalized comparisons difficult;
- Prototyping at production scale for performance analysis is usually impractical, as this would require hundreds of servers, multi-terabyte data sets, and thousands or millions of clients;
- The solution space is changing rapidly, with new products constantly emerging, and existing products releasing several versions per year with ever-evolving feature sets.

We faced these challenges during a recent project for a healthcare provider seeking to adopt NoSQL technology for an Electronic Health Record (EHR) system. The system supports healthcare delivery for over nine million patients in more than 100 facilities across the globe. Data currently grows at over one terabyte per month, and all data must be retained for 99 years.

In this paper, we outline a technology evaluation and selection method we have devised for big data systems. We then describe a study we performed for the healthcare provider described above. We introduce the study context, our evaluation approach, and the results of both extensive performance and scalability testing and a detailed feature comparison. We conclude by, describing some of the challenges for software architecture and design that NoSQL approaches bring. The specific contributions of the paper are as follows:

- A rigorous method that organizations can follow to evaluate the performance and scalability of NoSQL databases.
- Performance and scalability results that empirically demonstrate variability of up to 14x in throughput and 5x in latency in the capabilities of the databases we tested to support the requirements of our healthcare customer.

2. EHR CASE STUDY
Our method is inspired by earlier work on middleware evaluation [5][6] and is customized to address the characteristics of big data systems. The basic main steps are depicted in Figure 1 and outlined below:

2.1 Project Context
Our customer was a large healthcare provider developing a new electronic healthcare record (EHR) system to replace an existing system that utilizes thick client applications at sites around the world that access a centralized relational database. The customer decided to consider NoSQL technologies for two specific uses, namely:

- the primary data store for the EHR system
- a local cache at each site to improve request latency and availability

PABS'15, February 01m 2015, Austin, TX, USA

http://dx.doi.org/10.1145/2694730.2694731

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
16 JAN 2015	**N/A**	

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Performance Evaluation of NoSQL Databases: A Case Study	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
Matser /John Klein Ian Gorton Neil Ernst Patrick Donohoe Kim Pham Chrisjan	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release, distribution unlimited.

13. SUPPLEMENTARY NOTES
The original document contains color images.

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**SAR**	**6**	

As the customer was familiar with RDMS technology for these use cases, but had no experience using NoSQL, they directed us to focus the technology evaluation only on NoSQL technology.

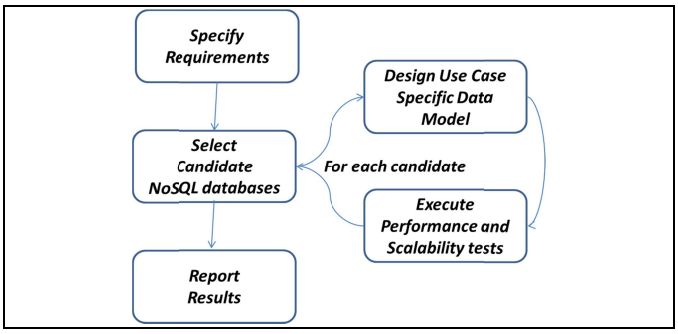

Figure 1 – Lightweight Evaluation and Prototyping for Big Data (LEAP4BD)

2.2 Specifying Requirements

We conducted a stakeholder workshop to elicit requirements and drivers. A key need that emerged was to understand the inherent performance and scalability that is achievable with each candidate NoSQL database.

We worked with the customer to define two driving use cases for the EHR system, which formed the basis for our performance and scalability assessment. The first use case was retrieving recent medical test results for a particular patient. The second use case was achieving strong consistency for all readers when a new medical test result is written for a patient, because all clinicians using the EHR to make patient care decisions need to see the same information about that patient, regardless of location. .

2.3 Select Candidate NoSQL Databases

Our customer was specifically interested in understanding how different NoSQL data models (key-value, column, document, graph) would support their application, so we selected one NoSQL database from each category to investigate in detail. We later ruled out graph databases, as they did not support the horizontal partitioning required for the customer's requirements. Based on a feature assessment of various products, we settled on Riak, Cassandra and MongoDB as our three candidate technologies, as these are market leaders in each NoSQL category.

2.4 Design and Execute Performance Tests

A thorough evaluation of complex database platforms requires prototyping with each to reveal the performance and scalability capabilities and allow comparisons [4]. To this end, we developed and performed a systematic procedure for an "apples to apples" comparison of the three databases we evaluated. Based on the use cases defined during the requirements step, we:

- Defined a consistent test environment for evaluating each database, which included server platform, test client platform, and network topology.
- Mapped the logical model for patient records to each database's data model and loaded the database with a large synthetic test data set.
- Created a load test client that performs the database read and write operations defined for each use case. This client can issue many concurrent requests, to characterize how each product responds as the request load increases.
- Defined and executed test scripts that exerted a specified load on the database using the test client.

Test cases were executed on several distributed configurations to measure performance and scalability, ranging from baseline testing on a single server to nine server instances that sharded and replicated data.

Based on this approach, we were able to produce test results that allow comparison of performance and scalability of each database for this customer's EHR system. Our test results were not intended to be general-purpose benchmarks, so tests using other types of workloads, server configurations, and data set sizes were not performed.

3. EVALUATION SETUP

3.1 Test Environment

The three databases we tested were:

- MongoDB version 2.2, a document store (http://docs.mongodb.org/v2.2/);
- Cassandra version 2.0, a column store (http://www.datastax.com/documentation/cassandra/2.0);
- Riak version 1.4, a key-value store (http://docs.basho.com/riak/1.4.10/).

Tests were performed on two database server configurations: A single node server, and a nine-node configuration that was representative of a production deployment. A single node test validated our base test environment for each database. The nine-node configuration used a topology that represented a geographically distributed deployment across three data centers. The data set was partitioned (i.e. "sharded") across three nodes, and replicated to two additional groups of three nodes each. We used MongoDB's primary/secondary feature, and Cassandra's data center aware distribution feature. Riak did not support this "3x3" data distribution, so we used a flattened configuration where data was sharded across all nine nodes, with three replicas of each shard stored across the nine nodes.

All testing was performed using the Amazon EC2 cloud (http://aws.amazon.com/ec2/). Database servers executed on "m1.large" instances, with the database data and log files stored on separate EBS volumes attached to each server instance. The EBS volumes were not provisioned with the EC2 IOPS feature, to minimize the tuning parameters used in each test configuration. Server instances ran the CentOS operating system (http://www.centos.org). The test client also executed on an "m1.large" instance, and also used the CentOS operating system. All instances were in the same EC2 availability zone (i.e. the same cloud data center).

3.2 Mapping the data model

We used the HL7 Fast Healthcare Interoperability Resources (FHIR) (http://www.hl7.org/implement/standards/fhir/) for our prototyping. The logical data model consisted of FHIR Patient Resources (e.g., demographic information such as names, addresses, and telephone numbers), and laboratory test results represented as FHIR Observation Resources (e.g., test type, result quantity, and result units). There was a one-to-many relation from each patient record to the associated test result records.

A synthetic data set was used for testing. This data set contained one million patient records, and 10 million lab result records. Each patient had between 0 and 20 test result records, with an average of seven. These Patient and Observation Resources were mapped into the data model for each of the databases we tested. This data set size was a tradeoff between fidelity (the actual system could be larger, depending on data aging and archiving strategy) versus test

execution time and the substantial costs of constructing and loading the data set for different data models and test configurations.

3.3 Create load test client

The test client was based on the YCSB framework [7], which provides capabilities to manage test execution and perform test measurement. For test execution, YCSB has default data models, data sets, and workloads, which we modified and replaced with implementations specific to our use case data and requests.

We were able to leverage YCSB's test execution management capabilities to specify the total number of operations to be performed, and the mix of read and write operations in the workload. The test execution capabilities also allow creation of concurrent client sessions using multiple execution threads.

The YCSB built-in measurement framework measures the latency for each operation performed, as the time from when the request is sent to the database until the response is received back from the database. The YCSB reporting framework records latency measurements separately for read and write operations. Latency distribution is a key scalability measure for big data systems [8] [9], so we collected both average and 95th percentile values.

We extended the YCSB reporting framework to report overall throughput, in operations per second. This measurement was calculated by dividing the total number of operations performed (read plus write) by the workload execution time, measured from the start of the first operation to the completion of the last operation in the workload execution, not including pre-test setup and post-test cleanup times.

3.4 Define and execute test scripts

The stakeholder workshop identified a typical workload for the EHR system of 80% read and 20% write operations. For this operation mix, we defined a read operation to retrieve the five most recent observations for a single patient, and a write operation to insert a single new observation record for a single existing patient.

Our customer was also interested in using NoSQL technology for a local cache, so we defined a write-only workload to represent the daily download from a centralized primary data store of records for patients with scheduled appointments for that day. We also defined a read-only workload to represent flushing the cache back to the centralized primary data store.

Each test ran the selected workload three times, in order to minimize the impact of any transient events in the cloud infrastructure. For each of these three runs, the workload execution was repeated using a different number of client threads (1, 2, 5, 10, 25, 50, 100, 200, 500, and 1000). Results were post-processed by averaging measurements across the three runs for each thread count.

4. PERFORMANCE AND SCALABILITY RESULTS

We report here on our results for a nine-node configuration that reflected a typical production deployment. As noted above, we also tested other configurations, ranging from a single server up to a nine-node cluster. The single-node configuration's availability and scalability limitations make it impractical for production use, however, in the discussion that follows we compare the single node configuration to distributed configurations, to provide insights into the efficiency of a database's distributed coordination mechanisms and resource usage, and guides tradeoffs between scaling by adding more nodes versus using faster nodes with more storage.

Defining a configuration required several design decisions. The first decision was how to distribute client connections across the server nodes. MongoDB uses a centralized router node with all clients connected to that single node. Cassandra's *data center aware distribution* feature was used to create three sub-clusters of three nodes each, and client connections were spread uniformly across the three nodes in one of the sub-clusters. In the case of Riak, the product architecture only allowed client connections to be spread uniformly across the full set of nine nodes. An alternative might have been to test Riak on three nodes with no replication, however other constraints in the Riak architecture resulted in extremely poor performance in this configuration, and so the nine-node configuration was used.

A second design decision was how to achieve the desired level of consistency, which requires coordinating write operation settings and read operation settings [10]. Each of the three databases offered slightly different options, and we explored two approaches, discussed in the next two sections. The first reports results using strong consistency, and the second reports results using eventual consistency.

4.1 Evaluation Using Strong Consistency

The selected options are summarized in Table 1. For MongoDB, the effect is that all writes were committed on the primary server, and all reads were from the primary server. For Cassandra, the effect is that all writes were committed on a majority quorum at each of the three sub-clusters, while a read required a majority quorum only on the local sub-cluster. For Riak, the effect was to require a majority quorum on the entire nine-node cluster for both write operations and read operations.

Table 1 - Settings for representative production configuration

Database	Write Options	Read Options
MongoDB	Primary Ack'd	Primary Preferred
Cassandra	EACH_QUORUM	LOCAL_QUORUM
Riak	quorum	quorum

The throughput performance for the representative production configuration for each of the workloads is shown in Figures 2, 3, and 4. Cassandra performance peaked at approximately 3200 operations per second, with Riak peaking at approximately 480 operations per second, and MongoDB peaking at approximately 225 operations per second.

In all cases, Cassandra provided the best overall performance, with read-only workload performance roughly comparable to the single node configuration, and write-only and read/write workload performance slightly better than the single node configuration. This implies that, for Cassandra, the performance gains that accrue from decreased contention for disk I/O and other per node resources (compared to the single node configuration) are greater than the additional work of coordinating write and read quorums across replicas and data centers. Furthermore, Cassandra's "data center aware" features provide some separation of replication configuration from sharding configuration. In this test configuration, this allowed a larger portion of the read operations to be completed without requiring request coordination (i.e. peer-to-peer proxying of the client request), compared to Riak.

Riak performance in this representative production configuration was nearly 4x better than the single node configuration. In test runs using the write-only workload and the read/write workload, our Riak client had insufficient socket resources to execute the workload for 500 and 1000 concurrent sessions. These data points are hence reported as zero values in Figures 3 and 4.

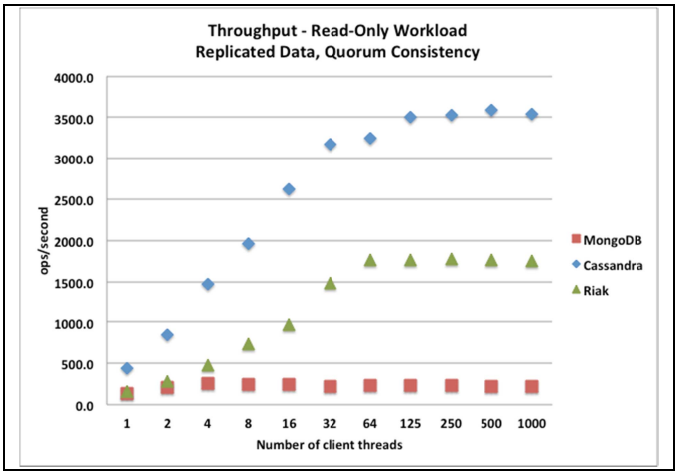

Figure 2 - Throughput, Representative Production Configuration, Read-Only Workload (higher is better)

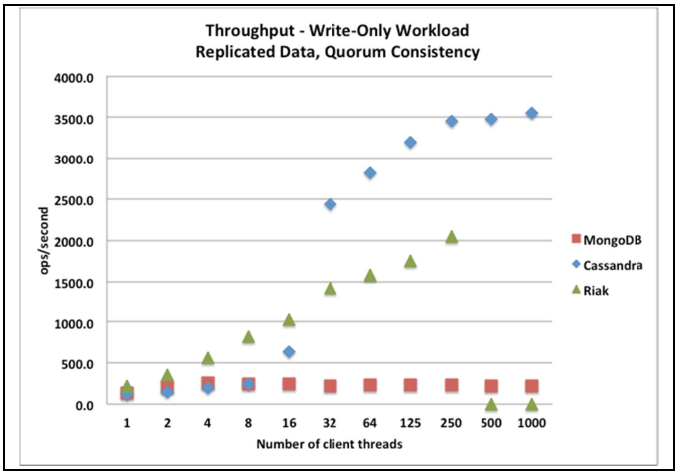

Figure 3 - Throughput, Representative Production Configuration, Write-Only Workload

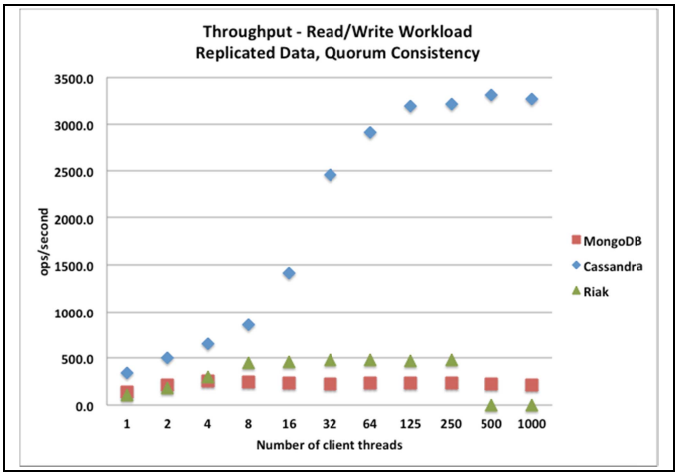

Figure 4 - Throughput, Representative Production Configuration, Read/Write Workload

We later determined that this resource exhaustion was due to ambiguous documentation of Riak's internal thread pool configuration parameter, which creates a pool for *each* client session and not a pool shared by *all* client sessions. After determining that this did not impact the results for one through 250 concurrent sessions, and given that Riak had qualitative capability gaps with respect to our strong consistency requirements (as discussed below), we decided not to re-execute the tests for those data points.

MongoDB performance is significantly lower here than the single node configuration, achieving less than 10% of the single node throughput. Two factors influenced the MongoDB results. First, the sharded configuration introduces the MongoDB router and configuration nodes into the deployment. The router node request proxying became a performance bottleneck. The read and write operation latencies shown in Figures 5 and 6 have nearly constant average latency for MongoDB as the number of concurrent sessions is increased, which we attribute the rapid saturation of the router node.

The second factor affecting MongoDB performance is the interaction between the sharding scheme used by MongoDB and our workloads. MongoDB used a range-based sharding scheme with rebalancing (http://docs.mongodb.org/v2.2/core/sharded-clusters/). Our workloads generated a monotonically increasing key for new records to be written, which caused all write operations to be directed to the same shard. While this key generation approach is typical (in fact, many SQL databases provide "autoincrement" key types that do this automatically), in this case it concentrates the write load for all new records on a single node and thus negatively impacts performance. A different indexing scheme was not available to us, as it would impact other systems that our customer operates. (We note that MongoDB introduced hash-based sharding in v2.4, after our testing had concluded.)

Our tests also measured latency of read and write operations. While Cassandra achieved the highest overall throughput, it also delivered the highest average latencies. For example, at 32 client connections, Riak's read operation latency was 20% of Cassandra (5x faster), and MongoDB's write operation latency was 25% of Cassandra's (4x faster). Figures 5 and 6 show average and 95th percentile latencies for each test.

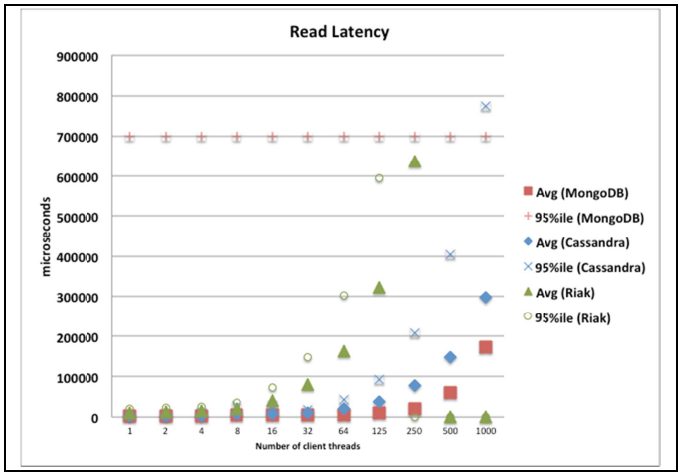

Figure 5 - Read Latency, Representative Production Configuration, Read/Write Workload

4.2 Evaluation Using Eventual Consistency

We also report performance results for the performance "cost" of strong replica consistency. These results do not include MongoDB data – the performance of MongoDB did not warrant additional characterization of that database for our application. The tests used a combination of write and read operation settings that resulted in

eventual consistency, rather than the strong consistency settings used in the tests described above. Again, each of the databases offered slightly different options. The selected options are summarized in Table 2. The effect of these settings was that writes were committed on one node (with replication occurring after the request acknowledgement was sent to the client), and read operations were executed on one replica.

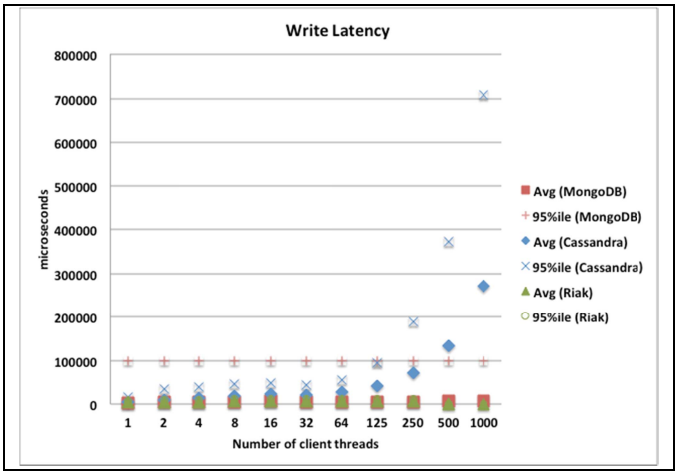

Figure 6 - Write Latency, Representative Production Configuration, Read/Write Workload

Table 2 - Settings for eventual consistency configuration

Database	Write Options	Read Options
Cassandra	ONE	ONE
Riak	Noquorum	noquorum

For Cassandra, at 32 client sessions, there is a 25% reduction in throughput moving from eventual to strong consistency. Figure 7 shows throughput performance for the read/write workload on the Cassandra database, comparing the representative production configuration with the eventual consistency configuration...

The same comparison is shown for Riak in Figure 8. Here, at 32 client sessions, there is only a 10% reduction in throughput moving from eventual to strong consistency (As discussed above, test client configuration issues resulted in no data recorded for 500 and 1000 concurrent sessions.)

In summary, the Cassandra database provided the best throughput performance, but with the highest latency, for the specific workloads and configurations tested here. We attribute this to several factors. First, hash-based sharding spread the request and storage load better than MongoDB. Second, Cassandra's indexing features allowed efficient retrieval of the most recently written records, particularly compared to Riak. Finally, Cassandra's peer-to-peer architecture and data center aware features provide efficient coordination of both read and write operations across replicas and data centers.

5. RELATED WORK

Systematic evaluation methods allow data-driven analysis and insightful comparisons of the capabilities of candidate components for an application. Prototyping supports component evaluation by providing both quantitative characterization of performance and qualitative understanding of other factors related to adoption. Gorton describes a rigorous evaluation method for middleware platforms, which can be viewed as a precursor for our work [4].

Benchmarking of database products is generally performed by executing a specific workload against a specific data set, such as the Wisconsin benchmark for general SQL processing [11] or the TPC-B benchmark for transaction processing [12]. These publically available workload definitions have long enabled vendors and others to publish measurements, which consumers can attempt to map to their target workload, configuration, and infrastructure to compare and select products. These benchmarks were developed for relational data models, and are not relevant for NoSQL systems.

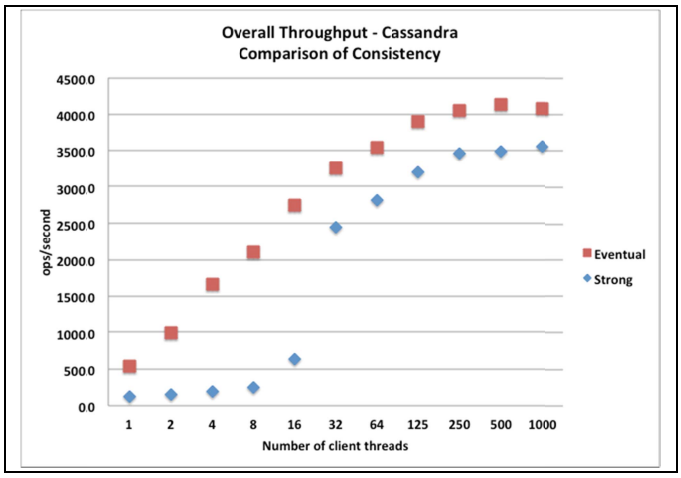

Figure 7 - Cassandra – Comparison of strong and eventual consistency

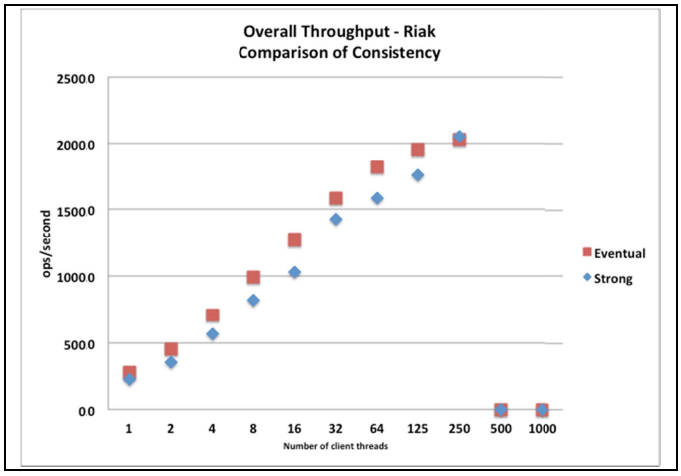

Figure 8 - Riak – Comparison of strong and eventual consistency

YCSB [7] has emerged as the *de facto* standard for executing simple benchmarks for NoSQL systems. YCSB++ [13] extends YCSB with multi-phase workload definitions and support for coordination of multiple clients to increase the load on the database server. There is an growing collection of published measurements using YCSB and YCSB++, from product vendors [14][15] and from researchers [16][17]. In this project, we built on the YCSB framework, customizing it with a more complex data set and application-specific workload definition.

6. FURTHER WORK AND CONCLUSIONS

NoSQL database technology offers benefits of scalability and availability through horizontal scaling, replication, and simplified data models, but the specific implementation must be chosen early

in the architecture design process. We have described a systematic method to perform this technology selection in a context where the solution space is broad and changing fast, and the system requirements may not be fully defined. Our method evaluates the products in the specific context of use, starting with elicitation of quality attribute scenarios to capture key architecture drivers and selection criteria. Next, product documentation is surveyed to identify viable candidate technologies, and finally, rigorous prototyping and measurement is performed on a small number of candidates to collect data to make the final selection.

We described the execution of this method to evaluate NoSQL technologies for an electronic healthcare system, and present the results of our measurements of performance, along with a qualitative assessment of alignment of the NoSQL data model with system-specific requirements.

There were a number of challenges in carrying out such an performance analysis on big data systems. These included:

- Creating the test environment – performance analysis at this scale requires very large data sets that mirror real application data. This raw data must then be loaded into the different data models that we defined for each different NoSQL database. A minor change to the data model in order to explore performance implications required a full data set reload, which is time-consuming.
- Validating quantitative criteria - Quantitative criteria, with hard "go/no-go" thresholds, were problematic to validate through prototyping, due to the large number of tunable parameters in each database, operating system, and cloud infrastructure. Minor configuration parameter changes can cause unexpected performance effects, often due to non-obvious interactions between the different parameters. In order to avoid entering an endless test and analyze cycle, we framed the performance criteria in terms of the shape of the performance curve, and focused more on sensitivities and inflection points.

ACKNOWLEDGMENT

Copyright 2014 ACM. This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. References herein to any specific commercial product, process, or service by trade name, trade mark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by Carnegie Mellon University or its Software Engineering Institute. This material has been approved for public release and unlimited distribution. T-Check℠. DM-0001936

7. REFERENCES

[1] S. Comella-Dorda, J. Dean, G. Lewis, et al., "A Process for COTS Software Product Evaluation." Software Engineering Institute, Technical Report, CMU/SEI-2003-TR-017, 2004.

[2] J. Zahid, A. Sattar, and M. Faridi. "Unsolved Tricky Issues on COTS Selection and Evaluation." *Global Journal of Computer Science and Technology* 12.10-D (2012).

[3] Becker, C., Kraxner, M., Plangg, M., & Rauber, A. Improving decision support for software component selection through systematic cross-referencing and analysis of multiple decision criteria. In *Proc. 46th Hawaii Intl. Conf. on System Sciences (HICSS)*, 2013, pp. 1193-1202.

[4] I. Gorton, A. Liu, and P. Brebner, "Rigorous evaluation of COTS middleware technology," *Computer,* vol. 36, no. 3, pp. 50-55, 2003.

[5] Y. Liu, I. Gorton, L. Bass, C. Hoang, & S. Abanmi. MEMS: a method for evaluating middleware architectures. In Proceedings of the 2nd Intl. Conf. on Quality of Software Architectures (QoSA'06), 2006, pp. 9-26.

[6] A. Liu and I. Gorton. 2003. Accelerating COTS Middleware Acquisition: The i-Mate Process. IEEE Software. 20, 2 (March 2003), 72-79.

[7] B. F. Cooper, A. Silberstein, E. Tam, et al., "Benchmarking Cloud Serving Systems with YCSB," in *Proc. 1st ACM Symp. on Cloud Computing (SoCC '10)*, 2010, pp. 143-154. doi: 10.1145/1807128.1807152

[8] G. DeCandia, D. Hastorun, M. Jampani, et al., "Dynamo: Amazon's Highly Available Key-value Store," in *Proc. 21st ACM SIGOPS Symp. on Operating Systems Principles (SOSP '07)*, Stevenson, Washington, USA, 2007, pp. 205-220. doi: 10.1145/1294261.1294281

[9] J. Dean and L. A. Barroso, "The Tail at Scale," *Communications of the ACM,* vol. 56, no. 2, pp. 74-80, February 2013. doi: 10.1145/2408776.2408794

[10] I. Gorton and J. Klein, "Distribution, Data, Deployment: Software Architecture Convergence in Big Data Systems," *IEEE Software,* vol. PP, no. 99, 18 March 2014. doi: 10.1109/MS.2014.51

[11] D. Bitton, D. J. DeWitt, and C. Turbyfill, "Benchmarking Database Systems: A Systematic Approach," in *Proc. 9th Int'l Conf. on Very Large Data Bases (VLDB '83)*, 1983, pp. 8-19.

[12] Anon, D. Bitton, M. Brown, et al., "A Measure of Transaction Processing Power," *Datamation,* vol. 31, no. 7, pp. 112-118, April 1985.

[13] S. Patil, M. Polte, K. Ren, et al., "YCSB++: Benchmarking and Performance Debugging Advanced Features in Scalable Table Stores," in *Proc. 2nd ACM Symp. on Cloud Computing (SOCC '11)*, 2011, pp. 9:1--9:14. doi: 10.1145/2038916.2038925

[14] D. Nelubin and B. Engber, "Ultra-High Performance NoSQL Benchmarking: Analyzing Durability and Performance Tradeoffs." Thumbtack Technology, Inc., White Paper, 2013.

[15] Datastax, "Benchmarking Top NoSQL Databases." Datastax Corporation, White Paper, 2013.

[16] V. Abramova and J. Bernardino, "NoSQL Databases: MongoDB vs Cassandra," in *Proc. Int'l C* Conf. on Computer Science and Software Engineering (C3S2E '13)*, 2013, pp. 14-22. doi: 10.1145/2494444.2494447

[17] A. Floratou, N. Teletia, D. J. DeWitt, et al., "Can the Elephants Handle the NoSQL Onslaught?," *Proc. VLDB Endowment,* vol. 5, no. 12, pp. 1712-1723, 2012.