ARL-CR-0809 ● Nov 2016

US Army Research Laboratory

Accumulo/Hadoop, MongoDB, and Elasticsearch Performance for Semi-Structured Intrusion Detection (IDS) Data

prepared by Ralph P Ritchey
ICF, Inc.
7125 Thomas Edison Drive, Suite 100
Columbia, MD 21046

under contract W911QX-14-F-0020

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

ARL-CR-0809 ● Nov 2016

US Army Research Laboratory

Accumulo/Hadoop, MongoDB, and Elasticsearch Performance for Semi-Structured Intrusion Detection (IDS) Data

prepared by Ralph P Ritchey
ICF, Inc.
7125 Thomas Edison Drive, Suite 100
Columbia, MD 21046

under contract W911QX-14-F-0020

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED (From - To)
November 2016	Contractor Report	01/2016–08/2016

4. TITLE AND SUBTITLE
Accumulo/Hadoop, MongoDB, and Elasticsearch Performance for Semi-Structured Intrusion Detection (IDS) Data

5a. CONTRACT NUMBER
W911QX-14-F-0020

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Ralph P Ritchey

5d. PROJECT NUMBER
ARL APP

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
ICF, Inc.
7125 Thomas Edison Drive, Suite 100
Columbia, MD 21046

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
US Army Research Laboratory
ATTN: RDRL-CIN-D
2800 Powder Mill Road
Adelphi, MD 20783-1138

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
ARL-CR-0809

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

13. SUPPLEMENTARY NOTES

14. ABSTRACT
NoSQL data stores are highly recognized for their ability to easily scale and store vast amounts of information. When considering converting to a NoSQL data store, a fact-based analysis should be applied to address the issues inherent in such an architectural-based, critical, core component change. As such, we evaluate Hadoop, MongoDB, and Elasticsearch as a replacement for data stored in a custom intrusion detection system infrastructure. In this type of environment, the number of records is voluminous, the records contain semi-structured data of varying data types, and both across-the-board analytics and surgical queries must be supported.

15. SUBJECT TERMS
NoSQL performance, intrusion detection system, data storage, Hadoop, MongoDB, Elasticsearch

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Ralph P Ritchey
Unclassified	Unclassified	Unclassified	UU	44	19b. TELEPHONE NUMBER (Include area code) 301-394-0780

Contents

List of Figures

INTENTIONALLY LEFT BLANK.

1. Introduction

NoSQL data stores, such as Accumulo/Hadoop,[1,2] MongoDB,[3] and Elasticsearch,[4] are highly recognized for their ability to easily scale and store vast amounts of information in ways that a traditional relational database management system (RDBMS)[5] cannot. When considering a migration from an RDBMS to NoSQL, a thoughtful, fact-based analysis should be used, just as when any other critical, core architectural component is changed. In this report, we evaluate Hadoop, MongoDB, and Elasticsearch as a replacement for RDBMS data storehouse in our custom intrusion detection system (IDS). In this particular use case, the number of records is voluminous and the records contain semi-structured data[6] of varying data types. The workload is varied, and both across-the-board analytics and surgical queries must be supported.

During research and development, our in-house architected, built, and deployed IDS used Oracle Database[7] as the RDBMS component. Through the use of various features such as table partitioning and bitmap indexes, Oracle Database scaled and performed well. However, licensing and support costs have continued to increase over the years, driving a desire to find a less-costly solution. After examining available options, the open-source database PostgreSQL[8] was chosen as a potential replacement. PostgreSQL offered several architectural similarities to Oracle Database that, prior to actual testing, we believed might allow it to scale to handle the projected data and workloads. A research project was performed and a report[9] was written that, although not quite a perfect apples-to-apples comparison, provided us with sufficient confidence to move forward with our migration plan.

Several years have passed since the successful migration to PostgreSQL; data volumes have continued to increase and it is again time to review options to replace PostgreSQL with a component that will allow easier and more cost-effective scaling. In addition, new techniques for detecting malicious and potentially malicious attacks in our data are being developed that will require more efficient access to our raw data and better access to computational resources. We believe that the features and capabilities provided by a NoSQL solution will better meet our needs, but this requires due diligence and evaluation to select the best option for our current as well as future use case.

2. Background

Hadoop and MongoDB, classified as key-value and document-oriented NoSQL solutions, respectively, are both recognized as the most popular within their respective classifications.[10] Elasticsearch is similar to MongoDB and is classified

as a document-oriented NoSQL solution. Although all offer excellent scalability and performance, the correct one that will maximize performance over the long term for our specific environment and use case when handling huge amounts of data needs to be carefully considered. Key factors that should be included in the decision-making process include the following:

- Are the data structured, semi-structured, or unstructured?
- Once inserted, what are the primary operations (transactions) that will be performed—analytical iterations across all or large parts of the data, queries to pull a few specific records at a time, or a combination of both?

In our environment, the largest set of data is semi-structured. Although much of the information we collect is highly structured, the largest portion of any given record is an unstructured, binary-type blob. With regard to the daily operational use of the collected data, the current majority of operations occurring after the data are inserted are primarily targeted queries. However, in the future, it is expected that new tools will be developed that will perform analytics across vast numbers of records, making its use more of a hybrid analytic-transactional system.

3. Yahoo! Cloud Serving Benchmark (YCSB)

3.1 Data Loading and Performance Testing Framework

When originally setting out to perform the research and experiments to determine the best NoSQL solution for our environment, the expectation was that we would need to build our own data loading and performance testing framework. While performing literature research to review work performed by others in this area, we discovered a paper[11] that referred to a data loading and performance testing framework, Yahoo! Cloud Serving Benchmark (YCSB).[12] This framework is freely available and, if proven sufficient for our testing, it would greatly reduce our startup work on this research.

At first, YCSB was thought to come very close to matching our needs, but it lacked one important item—the ability to create semi-structured data shaping during initial loading that matched closely enough to our expected data layout to provide us with results, thus allowing us to confidently make a decision. While digging deeper into YCSB's documentation and code, a `fieldlengthhistogram` configuration parameter was discovered. This parameter, although apparently not as popular to use (based on a lack of references in publications), allows a user to specify a specially crafted histogram file as input to shape the data generated for each individual record used to populate the data store being tested.

The first set of tests performed to determine the suitability of `fieldlengthhistogram` looked very promising. However, we quickly discovered that in adjusting the workload and histogram input file parameters, the data shaping we required was an edge case not directly supported by the provided implementation. Analysis of the algorithm supplied in `HistogramGenerator.java` confirmed that our edge case could not be effectively tuned, and a replacement `HistogramGenerator.java` file that worked specifically for our needed data shaping was easily developed and tested.

Note: To use YCSB, several additional packages were needed that are not natively included with version 6.8 of Red Hat Enterprise Linux (RHEL). Individuals trying to recreate the experiments or set up their environment for running their own experiments will need to install Apache Maven[13] (a tool used for making the software build process easier) and Python[14] (a popular scripting language), version 2.7.x or greater.

3.2 Sharded Table Support

In a real-world deployment, it is highly likely that large table(s) storing data will be sharded (partitioned) to facilitate faster querying. Although YCSB will dynamically create the table, it will not generate and execute the necessary commands to shard a table if the NoSQL data store being tested does not automatically perform sharding. When we searched the web, we found others with an interest in performance testing against a sharded table, but there were no sample configurations or methodologies on ways to actually execute it.

Through testing MongoDB, however, we discovered a methodology that would allow us to shard the table YCSB would use, even though YCSB would not directly shard the table itself. These procedures are as follows:

1. Properly configure YCSB and launch it to start the initial data load.
2. Stop YCSB after a few records have been inserted.
3. Delete the data from the table YCSB created.
4. Shard the table within the MongoDB environment.
5. Use YCSB to perform the initial data load.

These steps leverage YCSB for the initial creation of the table it will load and test against. We discovered that YCSB will not drop the table if it already exists. Moreover, we simply needed to remove any loaded data, shard the table, and restart the initial data loading. Due to the inability to control which fields are used for what

purposes during performance testing, sharding was performed on the "`_id`" field in MongoDB. Although not necessarily an ideal match to real-world use, it did allow us to evenly distribute the initial data load across all shards.

For Accumulo/Hadoop and Elasticsearch, the technique (described previously) to create a properly sharded table was not needed. In both cases, the table was automatically sharded due to the way these NoSQL servers natively operate; namely, once their software is configured for sharding, any newly created tables are automatically sharded across the available data nodes.

3.3 Workload Configuration

When using YCSB, a configuration file is used that contains values to define the *workload* with which to test. By defining the parameters of a workload, we model the usage pattern of the performance tests to match what the real-world usage would be. The definition of workload used for performance testing was as follows:

- Initially populate the NoSQL data store with 100,000,000 records before performance testing. Although this amount is not close to the number of records expected in our environment, given the available hardware for each data node and the number of available nodes, it would provide a large enough record set to see how various operations would perform and prove sufficiently large enough to show any performance differences between the NoSQL solutions being tested.
- Perform 8,000,000 transactions. This number was chosen because it would mimic the constant, high-volume usage expected in our type of environment and the usage pattern over an extended period of time. In addition, by executing a significant number of transactions, any caching of data performed by the hardware, operating system, and data store should be thoroughly used and regularly refreshed. The configuration of transactions for the workload during testing consisted of the following:
    - 10% reads
    - 20% updates
    - 50% inserts
    - 10% read-modify-writes
    - 10% scans (maximum number of records to access set to 1,000)
- A single table, modeled after a real-world design, consisting of the following:

- 32 fields of different capacity:
  - 30 of small capacity (random size between 1 and 10 bytes)
  - 1 of medium capacity (random size between 1 and 512 bytes)
  - 1 of large capacity (random size between 1 and 1,024 × 5 bytes)

It is interesting to note that YCSB does not support record deletion as part of the workload modeling. One of the 4 operations used in typical persistent storage operations (typically referred to as "CRUD"[15]) is the ability to delete data. Depending on usage, deletion of data typically falls into one of 2 general categories: deletion of individual records one at a time or bulk deletion of large quantities of records. Although it is not surprising that bulk deletion, typically done on an entire shard (partition) of data, is not supported because YCSB does not directly support sharded table creation, it is a bit surprising that individual record deletion support is missing.

Deleting records can have varying degrees of performance impact depending on how a data store handles the physical deletion process. Some data stores will not reuse freed space within a file on disk, requiring a maintenance action to be performed that compacts the file. This compaction may or may not be performed on a live system, which impacts availability, especially in 24/7 operational environments. Other data stores use custom structures within the data files storing the records on disk, allowing them to mark specific parts of a file as available for reuse. Depending on how the data store writes new records into these newly available slots, it may cause individual records to be written in a fragmented manner or leave small, unused areas within the file. Unusable space within the file will cause the data files to artificially inflate, possibly forcing multiple physical read operations for fragmented records or deeper seeks into a file than would be normally necessary if the disk space was freed by compaction.

YCSB generates statistics for both of the workload commands: load and run. Although loading data are highly important to our environment, we are more interested in overall performance, because the NoSQL solution operates when it is fully loaded with records. The statistics YCSB generates during initial data load were collected but not included in this report. The statistics generated during each run for each NoSQL solution are included in the Appendixes.

4. Hardware Configuration

For testing all NoSQL configurations, the following hardware was used with RHEL, version 6.8, and all recent RHEL patches were applied:

- 1 Dell PowerEdge R710 server
  - One 2.26-GHz Xeon 4 Core central processing unit (CPU)
  - Two 250-GB, 7,200-RPM SATA drives
  - Four 1-TB, 7,200-RPM SATA drives
  - Eight 4-GB memory sticks (32 GB total)
  - RHEL, version 6.8
  - Use:
    - YCSB
    - MongoDB, Accumulo/Hadoop, and Elasticsearch "master" server
- 4 Dell PowerEdge R420 servers
  - Two 2.2-GHz Xeon E5-2430 6 Core CPU
  - Four 2-TB, 7,200-RPM SATA Drives
    - PERC H710 mini configured for hardware-based redundant array of independent disks (RAID)-0 across all drives
  - Eight 16-GB memory sticks (128 GB total)
  - RHEL, version 6.8
  - Use:
    - MongoDB, Accumulo/Hadoop, and Elasticsearch sharded data nodes

5. MongoDB

5.1 Configuration

MongoDB, version 3.2.7, the most recent version available at the time, was obtained from MongoDB's download website[16] and used for performance testing. The Dell R710 was used as the master node, running both the Config Server and Query Router portions of MongoDB, as well as YCSB for both initial data loading and workload performance testing. In an ideal situation, YCSB would have been installed and run from another server so as not to impact the performance. Due to the limited availability of hardware for testing and the light load MongoDB's

Config Server and Query Router would place on the Dell R710, it was deemed acceptable to collapse the preferred setup onto this single server (Fig. 1).

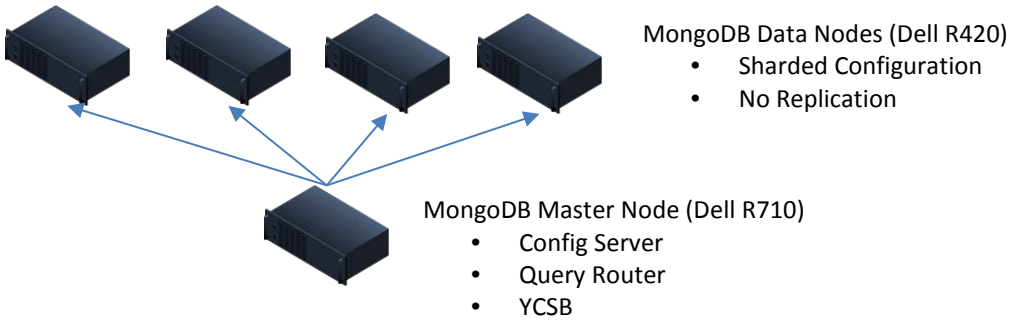

Fig. 1 MongoDB configuration

The remaining servers, all Dell R420s, were reserved as data nodes for storing the synthesized records generated by YCSB. These servers were configured for a sharded setup, similar to what would be expected in our production environment. Replication, which would provide data redundancy (similar to a RAID), was not enabled due to the limited availability of hardware.

5.2 Performance Experiment and Results

Four performance runs using YCSB were executed against fresh, initial data loads. The statistics generated by YCSB from each performance run were collected and then analyzed. MongoDB's overall throughput (Fig. 2) was consistent between performance runs, peaking at 152 transactions per second and bottoming out at 142 transactions per second, a range of approximately 9%. There is no known cause for the overall throughput performance variance; however, if future study is performed as outlined in Section 10, the use of additional performance tracking tools may reveal the cause. Similar to the other NoSQL solutions tested, MongoDB released all disk space used when the table being used for testing was dropped between performance runs.

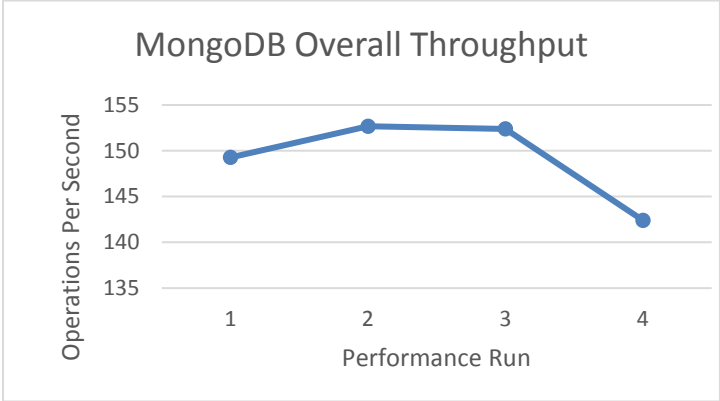

Fig. 2 MongoDB throughput

When comparing average latencies between various transaction types within MongoDB, the results shown in the graph (Fig. 3) are not unexpected. All of the transactions related to small, discrete-type transactions are very tightly grouped at the bottom of the graph due to low, average latencies (lower latency is better). MongoDB is designed more for a transactional-type environment rather than environments solely performing analytics, which require retrieving and processing huge numbers of records. In a transactional-type environment, such as backing a highly responsive website, small data requests must be served instantaneously. Based on the results from performance testing with YCSB, MongoDB's strongest performance appears to be in smaller, discrete transactions rather than running analytics across large data sets.

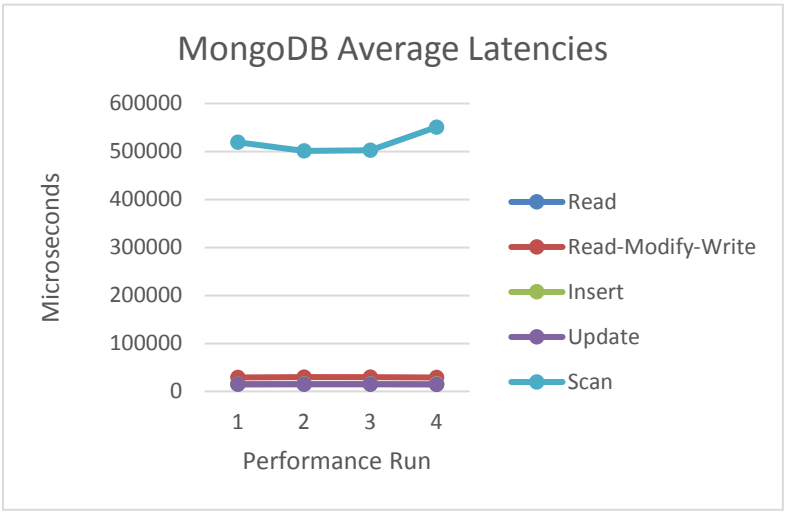

Fig. 3 Comparison of average latencies within MongoDB

6. Accumulo/Hadoop

6.1 Configuration

Hadoop, version 2.7.1;[17] Zookeeper (a required dependency for Accumulo), version 3.4.6;[18] and Accumulo, version 1.7.0,[19] were downloaded from their respective project sites and used for performance testing. The Dell R710 was used as the master node for all 3 services, as well as YCSB, for both initial data loading and workload performance testing. In an ideal situation, YCSB would have been installed and run from another server so as not to impact the performance. Due to the limited availability of hardware for testing, it was deemed acceptable to collapse the preferred setup onto this single server (Fig. 4).

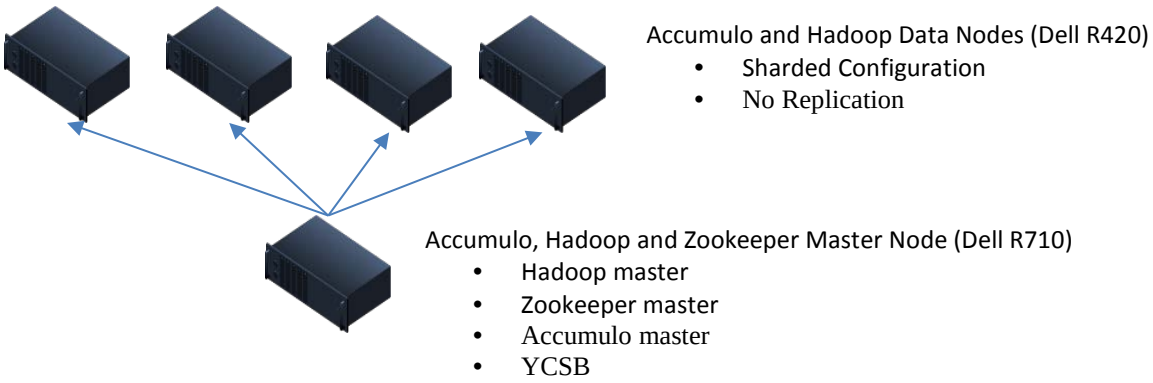

Fig. 4 Accumulo/Hadoop configuration

The remaining servers, all Dell R420s, were reserved as data nodes for storing the synthesized records generated by YCSB. These servers were configured for a sharded setup, similar to what would be expected in our production environment. Replication, which would provide data redundancy (similar to RAID), was not enabled due to the limited availability of hardware.

6.2 Performance Experiment and Results

Four performance runs were initiated, and the statistics generated by YCSB were collected for analysis after each run. Similar to the other NoSQL solutions tested, the initial run was performed on a pristine installation, and subsequent runs were performed after the table created by YCSB for testing was dropped (Fig. 5).

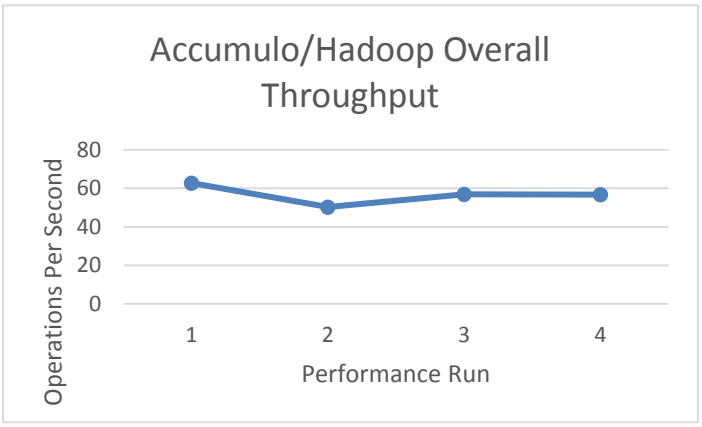

Fig. 5 Accumulo/Hadoop throughput

Hadoop's performance was consistent throughout all 4 performance tests. The variance between the highest throughput (62) and lowest throughput (50) was 19%. The consistently lower throughput after the initial performance indicates that Hadoop will perform slightly better on fresh, previously unused instances than it will when the environment is reused, even though tables and data were removed between performance tests.

Similar to MongoDB, Hadoop's latencies are very, very close (Fig. 6) for all transaction types except scans, which had the highest latencies. The latencies for non-scan transactions are so close that the latency lines for several transaction types are masked in the graph by others. Scan latencies are much higher, which is not unexpected due to the larger number of records being processed during that type of transaction.

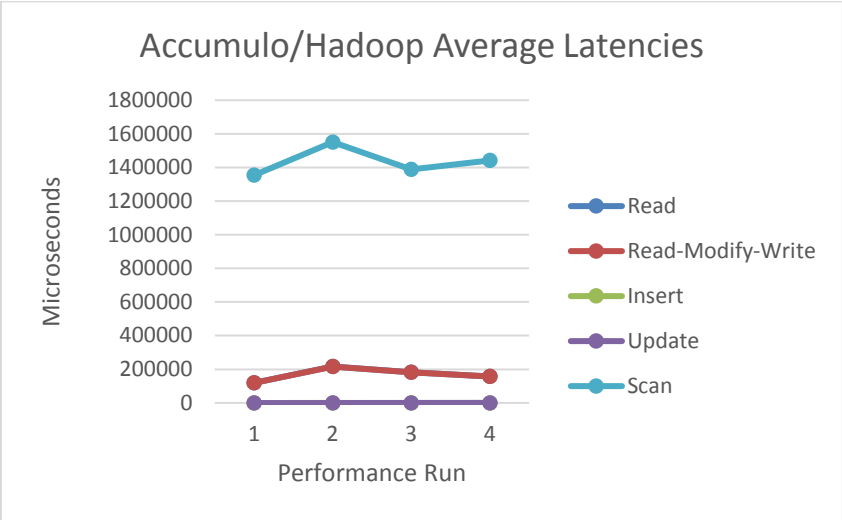

Fig. 6 Comparison of average latencies within Accumulo/Hadoop

7. Elasticsearch

7.1 Configuration

Elasticsearch, version 2.2.0, the most recent version available at the time, was obtained from Elasticsearch's download website[20] and used for performance testing. The Dell R710 was used as the master node, acting as the master Elasticsearch server, as well as YCSB, for both initial data loading and workload performance testing. In an ideal situation, YCSB would have been installed and run from another server. Due to the limited availability of hardware for testing, and the light load the Elasticsearch server process would place on the Dell R710, it was deemed acceptable to collapse the preferred setup onto this single server (Fig. 7).

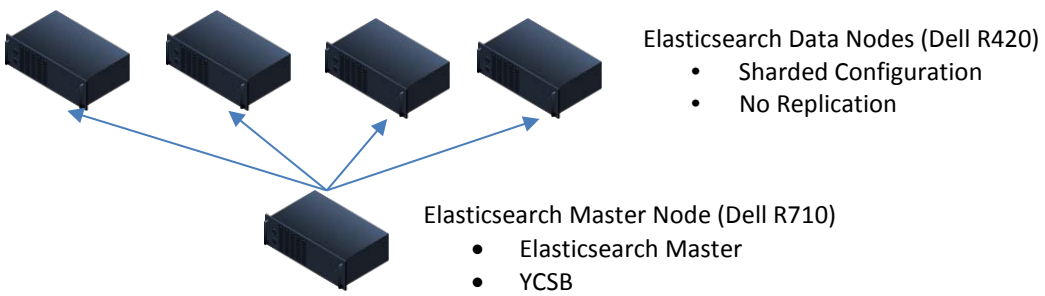

Fig. 7 Elasticsearch configuration

The remaining servers, all Dell R420s, were reserved as data nodes for storing the synthesized records generated by YCSB. These servers were configured for a sharded setup, similar to what would be expected in our production environment. Replication, which would provide data redundancy (similar to RAID), was not enabled due to the limited availability of hardware.

7.2 Performance Experiment and Results

Four performance runs were initiated, and the statistics generated by YCSB were collected for analysis after each run (Fig. 8). Similar to the other NoSQL solutions tested, the initial run was performed on a pristine installation, and subsequent runs were performed after the table created by YCSB for testing was dropped.

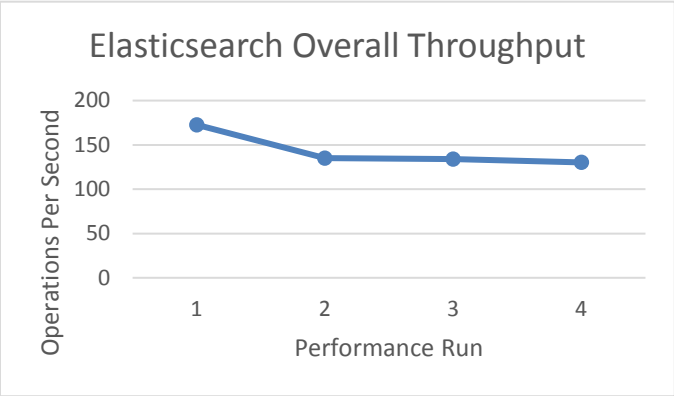

Fig. 8 Elasticsearch throughput

Although Elasticsearch frees disk space when a table is dropped, an approximate 23% performance decrease between the first and subsequent performance runs was observed on overall throughput. Based upon the individual statistics for each transaction type (available in the Appendixes), this performance decrease was not solely attributable to one specific transaction type that experienced a performance degradation on performance runs after the first execution.

Figure 9 shows the increase in average latencies across all transaction types for performance. The largest contributors to the general drop in throughput performance are read-write-modify and scan transactions, whereas the remaining types of transactions show more modest latency increases. As expected, after pulling larger volumes of records, scan-type transactions showed the largest average latency.

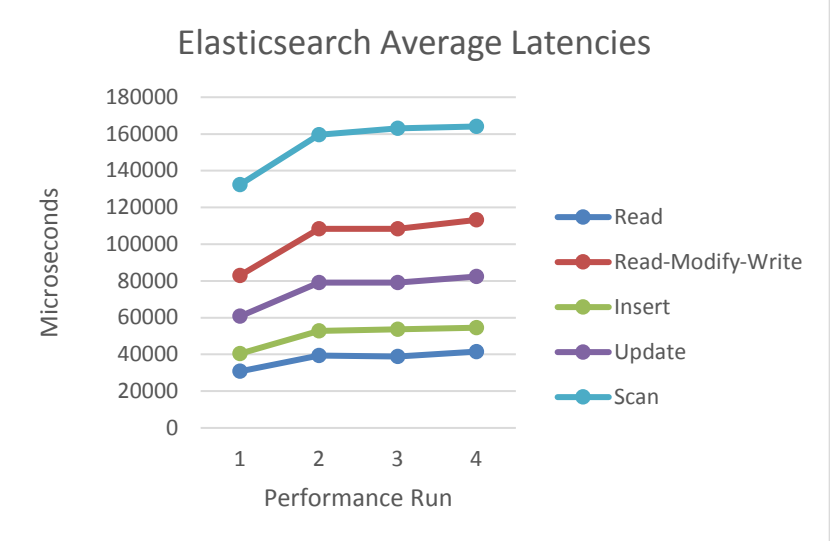

Fig. 9 Comparison of average latencies within Elasticsearch

8. Comparison of the Tested NoSQL Solutions

As shown in Fig. 10, overall, the Elasticsearch and MongoDB performance was very similar to the simulated performance testing workload. However, Accumulo/Hadoop showed a significant increase in the amount of time executing the same workload compared with MongoDB and Elasticsearch. This performance difference is most likely due to YCSB's workload simulator not leveraging Hadoop's mapreduce capability for table scans, potentially causing a large increase in latency for that specific transaction type that may have been significantly reduced otherwise.

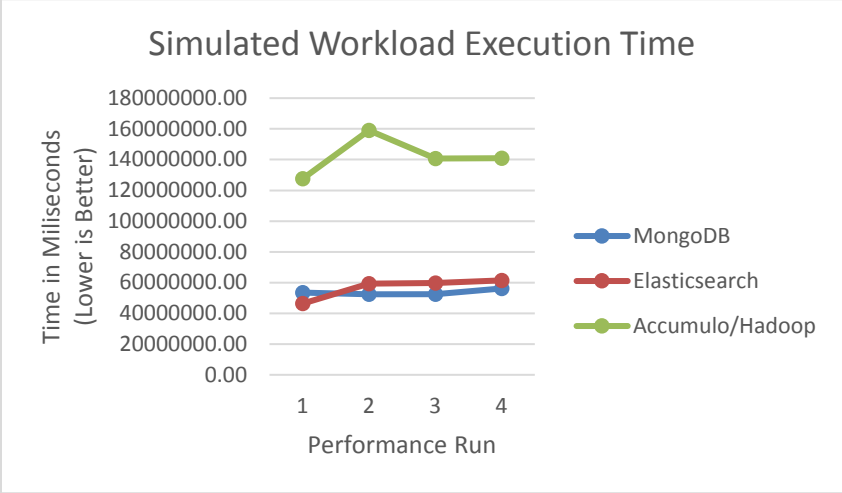

Fig. 10 Simulated execution time comparison

With regard to average throughput, both Elasticsearch and MongoDB showed more than twice the throughput as that of Accumulo/Hadoop. Again, Hadoop's performance may have been hampered by the lack of leveraging mapreduce, which may significantly improve scan transactions (Fig. 11).

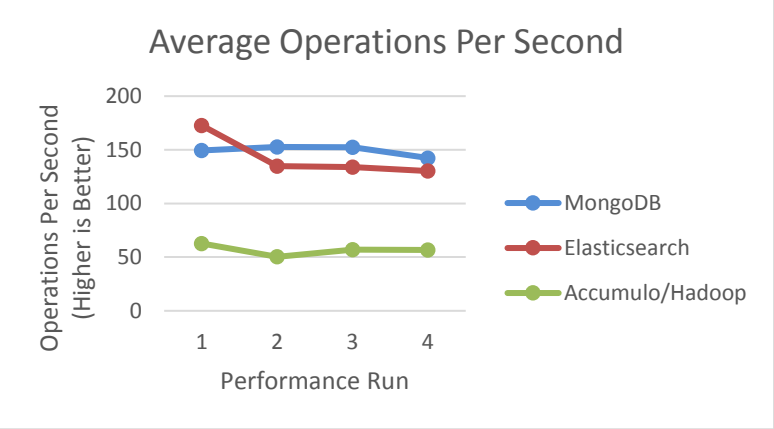

Fig. 11 Performance comparison of Elasticsearch, MongoDB, and Accumulo/Hadoop

9. Conclusions

Based on the statistics generated for the time taken to complete the simulated workloads and the average operations per second, both MongoDB and Elasticsearch appear to be generally well suited for our specific use case. Both NoSQL solutions completed the workloads in approximately the same amount of time and with approximately the same throughput. For our particular use case, Accumulo/Hadoop did not perform as well and most likely would not be a suitable solution due to the mixed use (backing web applications requiring fast response times for discrete data requests and analytics).

Particularly interesting are the average latency statistics for each of the individual transaction types. For insert and update transactions, Elasticsearch had the highest latency by a wide margin. Although Elasticsearch's performance was average for reads and read-modify-write transactions, its performance was notably better than the other NoSQL solutions in scans. Scan performance was so significantly faster for Elasticsearch that it made up for lesser performance in other areas to allow it to match MongoDB's performance overall.

10. Future Work

Although every effort was made to create the best possible experimental design and execution, a few improvements that are not currently possible could be undertaken in the future. Although YCSB provided a solid framework for initial loading, testing generalized workloads and gathering statistics, a few improvements would have made each step align more closely with our real-world data sets and uses. For example, providing an existing table loaded with real data to generate the load model would have resulted in an initial data load that mimics the real world, including the actual data types, instead of using random text-based data across the board. Coupled with real-world data, real-world queries could be gathered from the existing production environment, analyzed, and then used for generating the types of queries and data processing that currently exist. Both of these would improve the accuracy of the test results to get closer to what real-world performance may actually be when the system is deployed. These alterations to the schema would also facilitate the testing and impact of various indexing techniques, allowing better performance comparisons in better optimized configurations.

Duplicating the workload performance testing with replication enabled would be highly informative. In addition to determining the performance impact enabling replication has within a specific NoSQL server, performance comparisons between the different NoSQL server solutions would be useful. Due to the techniques used

to implement replication, there is a likelihood that the performance comparisons seen in this report will be different when replication is enabled. Future experimentation in which additional hardware is available could also be performed with replication enabled and disabled to determine the performance impact on the NoSQL servers as they scale out.

An additional improvement would be to monitor and collect individual node performance characteristics (CPU, memory, disk input/output use) using a tool such as Ganglia.[21] Once any performance impact introduced by running a tool such as Ganglia is removed, the collected results would provide insight into how hard the tested NoSQL solutions are using available resources, providing an additional point for comparison rather than solely relying on raw transaction throughput statistics.

During performance testing, all executions were performed with 10 concurrent YCSB threads running on the master node. Additional performance experimentation could be done to determine the optimum number of threads before performance of the NoSQL data store began to degrade. In real-world use, the number of connections to the data store can vary wildly as workloads shift throughout normal daily operations. Knowing the maximal number of connections before performance degrades against a known number of data nodes would be very useful to systems administrators for capacity planning.

11. References

1. Apache Accumulo. Forest Hill (MD): The Apache Software Foundation; 2016 [accessed 2016 Aug 2]. https://accumulo.apache.org/.

2. Wecome to Apache(tm) Hadoop(r)! Forest Hill (MD): The Apache Software Foundation; 2014 [accessed 2015 Oct 28]. https://hadoop.apache.org.

3. MongoDB for giant ideas. New York (NY): MongoDB, Inc.; 2016 [accessed 2015 Oct 28]. https://www.mongodb.org.

4. Elasticsearch webpage. Mountain View (CA): Elasticsearch BV; 2016 [accesed 2016 May 23]. https://www.elastic.co/.

5. Relational database management system. San Franscico (CA): Wikipedia Foundation, Inc.; 2015 [accessed 2015 Oct 28]. https://en.wikipedia.org/wiki/Relational_database_management_system.

6. Semi-structured data. San Franscico (CA): Wikipedia Foundation, Inc.; 2015 [accessed 2015 Oct 28]. https://en.wikipedia.org/wiki/Semi-structured_data.

7. Oracle, Database 12c. Redwood Shores (CA): Oracle; 2015 [accessed 2015 Oct 28]. https://www.oracle.com/database/index.html.

8. PostgreSQL: The world's most advanced open source database. PostgreSQL Global Development Group; 2015 [accessed 2015 Oct 28]. http://www.postgresql.org/.

9. Parker T, Ritchey P. High capacity single table performance design using partitioning in oracle or PostgreSQL. Adelphi (MD): Army Research Laboratory (US); 2012. Report No.: ARL-CR-0689.

10. Woodie A. Forrester Ranks the NoSQL Database Vendors. San Diego (CA): Datanami; 2014 Oct 3 [accessed 2015 Oct 28]. http://www.datanami.com/2014/10/03/forrester-ranks-nosql-database-vendors /.

11. Cooper BF, Silberstein A, Tam E, Ramakrishnan R, Sears R. Benchmarking cloud serving systems with YCSB. Proceedings of the 1st ACM Symposium on Cloud Computing; 2010 Jun 10–11. Indianapolis, IN.

12. Cooper BF. brianfrankcooper/YCSB. GitHub; 2016 [accessed 2015 Oct 28]. https://github.com/brianfrankcooper/YCSB.

13. Welcome to Apache Maven. Forest Hill (MD): The Apache Software Foundation; 2016 [accessed 2015 Nov 3]. https://maven.apache.org/.

14. Welcome to Python.org. Python Software Foundation; 2016 [accessed 2015 Nov 3]. https://www.python.org/.

15. Create, read, update and delete, San Franscico (CA): Wikimedia Foundation, Inc.; 12 January 2016 [accessed 2015 Jan 13]. https://en.wikipedia.org/wiki/Create,_read,_update_and_delete.

16. Downloads. New York (NY): MongoDB, Inc.; 2016 [accessed 2015 Nov 2]. https://www.mongodb.org/downloads#production.

17. Apache Hadoop Releases. Forest Hill (MD): The Apache Software Foundation; 2014 [accessed 2016 Jun 21]. http://hadoop.apache.org/releases.html.

18. Apache Zookeeper – Releases. Forest Hill (MD): The Apache Software Foundation; 2016 [accessed 2016 Jun 21]. http://zookeeper.apache.org/releases.html.

19. Downloads. Forest Hill (MD): The Apache Software Foundation; 2016 [accessed 2016 Jun 21]. http://zookeeper.apache.org/releases.html.

20. The Elastic Stack download. Mountain View (CA): Elasticsearch BV; 2016 [accessed 2016 May 23]. https://www.elastic.co/downloads.

21. Ganglia Monitoring System. Berkeley (CA): Ganglia Project; n.d. [accessed 2015 Dec 8]. http://www.ganglia.info.

INTENTIONALLY LEFT BLANK.

Appendix A. Overall Average Latencies

The following charts (Fig. A-1 through A-5) compare the average latencies for each transaction type between the tested NoSQL solutions.

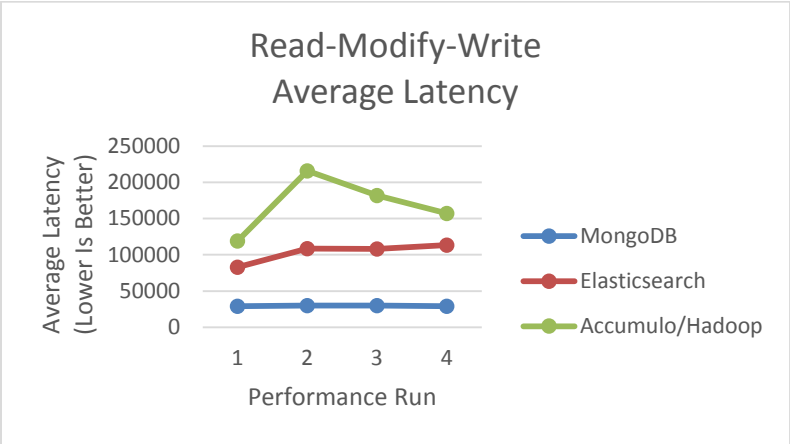

Fig. A-1 Read-modify write average latency

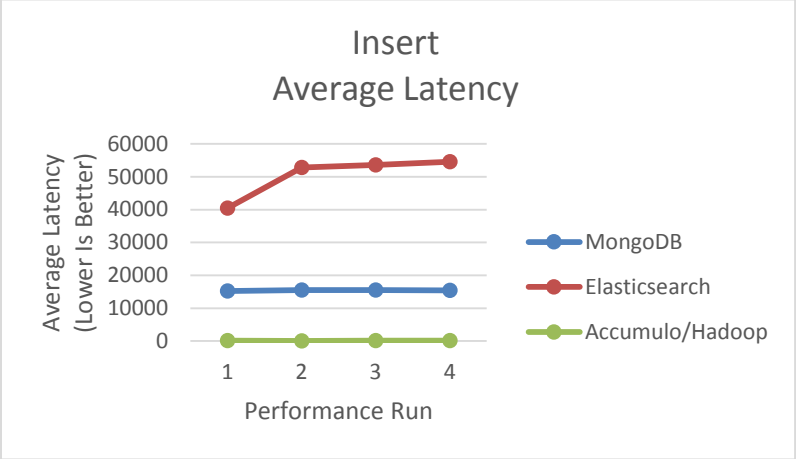

Fig. A-2 Insert average latency

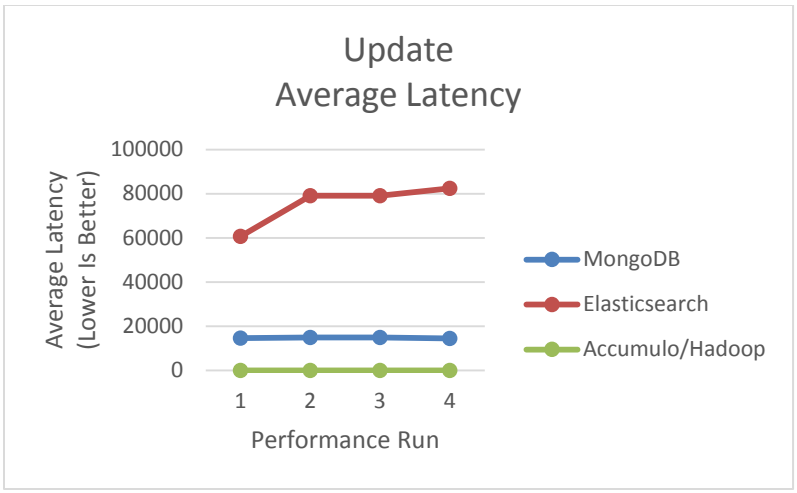

Fig. A-3 Update average latency

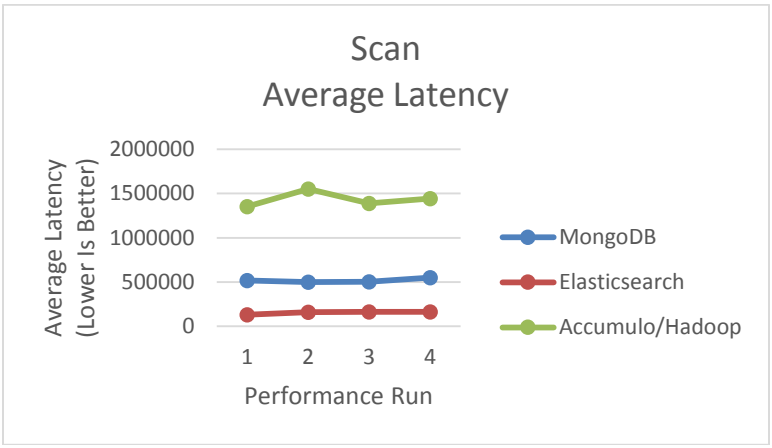

Fig. A-4 Scan average latency

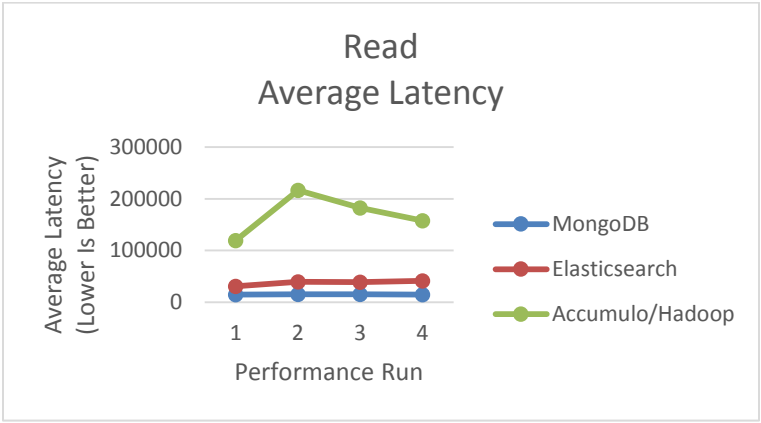

Fig. A-5 Read average latency

INTENTIONALLY LEFT BLANK.

Appendix B. MongoDB Raw Performance Numbers

Statistic	Perf Run 1	Perf Run 2	Perf Run 3	Perf Run 4
OVERALL[a]				
Runtime[b]	53589116	52397223	52498683	56177868
Throughput (ops/sec)	149.2840449	152.6798472	152.3847751	142.4048346
READ				
Operations	1600593	1599737	1601905	1601143
Avg Lat[c]	14655.49946	14989.27076	15009.2778	14729.48794
Lat Var[c]	296200041.82	257119308.07	226897250.54	661175799.30
Min Lat[c]	510	546	504	507
Max Lat[c]	3052680	2282599	1563543	15346122
95th Per Lat[b]	32000	32000	32000	32000
99th Per Lat[b]	39000	39000	39000	39000
Return	1600593	1599737	1601905	1601143
READ-MODIFY-WRITE				
Operations	799740	800089	801564	799791
Avg Lat[c]	29129.69781	29822.38637	29736.23325	28990.01811
Lat Var[c]	2057319877.21	1088404545.36	913569712.27	1032167145.38
Min Lat[c]	946	1051	973	892
Max Lat[c]	18943978	5037281	4641542	4234861
95th Per Lat[b]	63000	63000	63000	63000
99th Per Lat[b]	76000	76000	76000	76000
CLEANUP				
Operations	10	10	10	10
Avg Lat[c]	639.7	853	925.9	713.4
Lat Var[c]	3654856.81	6505905	7669401.49	4549125.24
Min Lat[c]	1	1	1	1
Max Lat[c]	6375	8505	9234	7112
95th Per Lat[b]	6000	8000	9000	7000
99th Per Lat[b]	6000	8000	9000	7000
INSERT				
Operations	4000009	3999669	4000482	3997531
Avg Lat[c]	15259.00436	15532.29465	15502.30262	15376.04012
Lat Var[c]	1329815744.56	750759403.10	817869519.89	711128877.11
Min Lat[c]	484	499	506	490
Max Lat[c]	19261144	5155493	11238529	7951853
95th Per Lat[b]	32000	32000	32000	32000
99th Per Lat[b]	39000	39000	39000	40000
Return	4000009	3999669	4000482	3997531
UPDATE				

Operations	2399305	2400604	2399164	2400452
Avg Lat[c]	14691.12889	15038.06724	14953.64515	14537.20209
Lat Var[c]	1146098647.51	671476718.65	595633461.28	551782098.48
Min Lat[c]	398	369	417	373
Max Lat[c]	19241419	5021646	10620600	7151262
95th Per Lat[b]	31000	32000	32000	32000
99th Per Lat[b]	39000	39000	39000	39000
Return	2399305	2400604	2399164	2400452
SCAN				
Operations	799833	800079	800013	800665
Avg Lat[c]	519284.4313	501173.3909	502323.5175	550773.376
Lat Var[c]	284121264615.24	245142958209.89	257999567148.60	504905485224.05
Min Lat[c]	1374	1362	1339	1340
Max Lat[c]	9827641	9489587	13317476	40689574
95th Per Lat[b]	N/A	N/A	N/A	N/A
99th Per Lat[b]	N/A	N/A	N/A	N/A
Return	799833	800079	800013	800665

Abbreviations: Avg, average; Lat, latency; Max, maximum; MDB, MongoDB; Min, minimum; N/A, not available; ops/sec, operations per second; Per, percentile; Perf, performance; Var, Var.

[a]Bold italics represent category.

[b]Milliseconds.

[c]Microseconds.

INTENTIONALLY LEFT BLANK.

Appendix C. Elasticsearch Raw Performance Numbers

Statistic	Perf Run 1	Perf Run 2	Perf Run 3	Perf Run 4
OVERALL[a]				
Runtime[b]	46341304.00	59290521.00	59756166.00	61435485.00
Throughput (ops/sec)	172.6321728	134.9288194	133.8773977	130.2179026
READ				
Operations	1600253	1600573	1600177	1600322
Avg Lat[c]	30797.95472	39466.02557	38883.62301	41537.75591
Lat Var[c]	9788983140.10	13135376466.84	13055686446.52	13917133929.15
Min Lat[c]	400	354	345	359
Max Lat[c]	7244190	7655769	2024994	2644727
95th Per Lat[b]	169000	318000	312000	345000
99th Per Lat[b]	497000	545000	549000	563000
Return	1600253	1600573	1600177	1600322
READ-MODIFY-WRITE				
Operations	800821	800231	799266	800079
Avg Lat[c]	83013.28272	108396.7725	108325.9005	113227.5195
Lat Var[c]	34191436625.70	46473986149.00	46867125986.38	49152908071.01
Min Lat[c]	2479	2485	2579	2498
Max Lat[c]	7380199	7832373	5023198	4992357
95th Per Lat[b]	508000	612000	618000	641000
99th Per Lat[b]	873000	943000	946000	958000
CLEANUP				
Operations	10	10	10	10
Avg Lat[c]	15647.4	40373.2	15661.6	55854.3
Lat Var[c]	122506987.64	1599941306.96	110821661.84	874308637.41
Min Lat[c]	8127	11639	7668	28509
Max Lat[c]	40946	122493	36765	117662
95th Per Lat[b]	40000	122000	36000	117000
99th Per Lat[b]	40000	122000	36000	117000
INSERT				
Operations	3999358	3997985	4001346	4001160
Avg Lat[c]	40430.26769	52837.69079	53632.07123	54538.87128
Lat Var[c]	14931073111.66	20271030187.45	20741229164.17	21088883919.97
Min Lat[c]	1420	1365	1471	1434
Max Lat[c]	7182325	7499094	4073622	5116669
95th Per Lat[b]	312000	440000	444000	448000
99th Per Lat[b]	635000	708000	716000	721000
Return	3999358	3997985	4001346	4001160
UPDATE				

Operations	2401341	2400584	2398337	2398893
Avg Lat[c]	60741.95254	79113.22524	79031.09753	82390.33232
Lat Var[c]	23828836120.88	32487312983.86	32719869613.18	34215699719.42
Min Lat[c]	1622	1677	1677	1657
Max Lat[c]	7250953	7816412	4914606	4981801
95th Per Lat[b]	459000	495000	498000	506000
99th Per Lat[b]	776000	851000	856000	868000
Return	2401341	2400584	2398337	2398893
SCAN				
Operations	799869	801089	799406	799704
Avg Lat[c]	132453.6022	159626.9182	162975.7695	164073.6762
Lat Var[c]	40294829656.79	46396021364.21	47094888256.02	47787227419.25
Min Lat[c]	2026	2168	2048	2200
Max Lat[c]	8320897	7601090	2706485	5098816
95th Per Lat[b]	551000	597000	606000	611000
99th Per Lat[b]	815000	852000	857000	861000
Return	799869	801089	799406	799704

Abbreviations: Avg, average; Lat, latency; Max, maximum; Min, minimum; ops/sec, operations per second; Per, percentile; Perf, performance; Var, Var.

[a]Bold italics represent category.

[b]Milliseconds.

[c]Microseconds.

INTENTIONALLY LEFT BLANK.

Appendix D. Accumulo/Hadoop Raw Performance Numbers

Statistic	Perf Run 1	Perf Run 2	Perf Run 3	Perf Run 4
OVERALL[a]				
Runtime[b]	127543725.00	159051237.00	140703245.00	140869092.00
Throughput (ops/sec)	62.72358754	50.29825703	56.85725301	56.79031423
READ				
Operations	1600457	1601363	1603507	1599383
Ave Lat[c]	119197.3758	216709.9007	182378.9185	157614.01
Lat Var[c]	179313305208.40	399400795495.80	297794074059.09	254209719476.46
Min Lat[c]	1449	1858	1769	1911
Max Lat[c]	6953521	7760428	6854180	7081101
95th Per Lat[b]	125000	253000	212000	183000
99th Per Lat[b]				
Return	1600457	1601363	1603507	1599383
READ-MODIFY-WRITE				
Operations	800519	801133	802723	800027
Ave Lat[c]	119114.2019	215888.3647	182022.1489	157289.8678
Lat Var[c]	178702351818.39	396686487083.27	296608679889.01	253192022096.29
Min Lat[c]	1716	1867	2288	2076
Max Lat[c]	6953568	7760450	6854216	7081136
95th Per Lat[b]	125000	252000	212000	183000
99th Per Lat[b]				
CLEANUP				
Operations	10	10	10	10
Ave Lat[c]	9950.9	10421.7	9023.7	11724
Lat Var[c]	19366610.89	16400429.41	8666881.61	200110545.40
Min Lat[c]	4999	4074	5012	3956
Max Lat[c]	18963	16231	14550	53849
95th Per Lat[b]	18000	16000	14000	53000
99th Per Lat[b]	18000	16000	14000	53000
INSERT				
Operations	4001123	3999008	3996568	3998774
Ave Lat[c]	78.62225755	76.62617754	92.43922435	102.4986828
Lat Var[c]	24422792.92	25885203.45	65993332.57	60300590.97
Min Lat[c]	10	10	10	11
Max Lat[c]	3709516	5109179	5966668	5939528
95th Per Lat[b]	0	0	0	0
99th Per Lat[b]	0	0	0	0
Return	4001123	3999008	3996568	3998774
UPDATE				

Operations	2399758	2400425	2401898	2401903
Ave Lat[c]	16.12536639	20.9917223	22.51156169	15.24438705
Lat Var[c]	13141745.30	47910420.31	47088780.57	12140842.87
Min Lat[c]	0	0	0	0
Max Lat[c]	3296180	6351485	6149367	3223248
95th Per Lat[b]	0	0	0	0
99th Per Lat[b]	0	0	0	0
Return	2399758	2400425	2401898	2401903
SCAN				
Operations	799181	800337	800750	799967
Ave Lat[c]	1354098.45	1550875.565	1389025.324	1442443.907
Lat Var[c]	2928237791696.45	3211134761763.23	2842257166000.84	3084908137706.06
Min Lat[c]	14505	15932	17606	19061
Max Lat[c]	9063582	14027499	9224391	8223699
95th Per Lat[b]	N/A	N/A	N/A	N/A
99th Per Lat[b]	N/A	N/A	N/A	N/A
Return	799181	800337	800750	799967

Abbreviations: Avg, average; Lat, latency; Max, maximumMin, minimum; N/A, not available; ops/sec, operations per second; Per, percentile; Perf, performance; Var, Var.

[a]Bold italics represent category.

[b]Milliseconds.

[c]Microseconds.

List of Symbols, Abbreviations, and Acronyms

CPU	central processing unit
IDS	intrusion detection system
RAID	redundant array of independent disks
RDBMS	relational database management system
RHEL	Red Hat Enterprise Linux
YCSB	Yahoo! Cloud Serving Benchmark

1 DEFENSE TECHNICAL
(PDF) INFORMATION CTR
DTIC OCA

2 DIRECTOR
(PDF) US ARMY RESEARCH LAB
RDRL CIO L
IMAL HRA MAIL & RECORDS MGMT

1 GOVT PRINTG OFC
(PDF) A MALHOTRA

1 DIR USARL
(PDF) RDRL CIN D
R RITCHEY

INTENTIONALLY LEFT BLANK.