Performance Benchmarking of Key-Value Store NoSQL Databases

Omoruyi Osemwegie, Kennedy Okokpujie, Nsikan Nkordeh, Charles Ndujiuba, Samuel John, Uzairue Stanley
Department of Electrical and Information Engineering, Covenant University, Nigeria

Article Info

Article history:

Received Sep 6, 2017
Revised May 3, 2018
Accepted Jun 11, 2018

Keyword:

NoSQL
Redis
SSDB database
Yahoo Cloud Serving Benchmark (YCSB)

ABSTRACT

Increasing requirements for scalability and elasticity of data storage for web applications has made Not Structured Query Language NoSQL databases more invaluable to web developers. One of such NoSQL Database solutions is Redis. A budding alternative to Redis database is the SSDB database, which is also a key-value store but is disk-based. The aim of this research work is to benchmark both databases (Redis and SSDB) using the Yahoo Cloud Serving Benchmark (YCSB). YCSB is a platform that has been used to compare and benchmark similar NoSQL database systems. Both databases were given variable workloads to identify the throughput of all given operations. The results obtained shows that SSDB gives a better throughput for majority of operations to Redis's performance.

Corresponding Author:

Kennedy Okokpujie,
Department of Electrical and Information Engineering,
Covenant University,
KM 10 Idiroko Road, Ota, Ogun State, Nigeria.
Email: okokpujiekennedy@covenantuniverrsity.edu.ng

1. INTRODUCTION

There is an increasing proliferation of Not Structured Query Language (NoSQL) databases. Amongst their key advantages is the promise of faster and efficient performance than the legacy Relational Database Management Systems (RDBMS) [1]. NoSQL databases are also tailor fitted to the fast growing world of cloud computing, allowing for massive scaling "on demand" (elasticity) and simplified application development [2]. However, there are words of caution to the bandwagon of adoption in big data and web application development noting that not all NoSQL databases are created alike where performance is concerned [3]. [4] asserts that since NoSQL solutions are not mature and are progressing at different speeds, database administrators have to choose carefully between NoSQL and relational databases according to their specific needs in terms of consistency, performances, security, scalability, costs and other non-functional criteria.

With the substantial number of open-source and readily available NoSQL systems, web applications developer also experience the headache of selecting amongst such NoSQL alternatives. This then suggests a Benchmarking among peers with scenarios produced in web application activity used to determine the best fit for various scenarios. Benchmarking in this respect refers to a performance evaluation of NoSQL solutions proposed or in use. The demand therefore is that sample interactions mimicking similar behaviour or actions as case may be in such web applications be used in a probabilistic or deterministic fashion to benchmark the performance of selected NoSQL database. Only in such ways can its selection be deemed reasonably suitable to be faster and more suited to a particular set of user interaction than a peer.

NoSQL databases can be classed into four categories namely: Key-value stores, Document stores, Wide column stores, Graph Databases [4], [5]. A key-value store can be viewed as a collection of registers, each identified by a key [6]. A common use case for these systems is as a layer in the data-retrieval hierarchy: a cache for expensive-to-obtain values, indexed by unique keys [7]. [5] Assert that key-value stores are generally good solutions if these have a simple application with only one kind of object, and only need to look up objects based on one attribute. Examples include Redis, Dynamo, Memcached, Voldermort etc. Document stores also known as document-oriented databases store document-oriented data in the form of Binary Javascript Object Notation (BSON) [8] or Javascript Object Notation (JSON). These systems are appealing to Web 2.0 programmers since these are generally supported by JSON as their data model [9]. Unlike the key-value stores, these systems generally support secondary indexes and multiple types of documents (objects) per database and nested documents or lists [10]. These are not required to adhere to a standard schema, the flexibility of JSON allows the user to work with data without having to define a schema upfront [8], [9]. Examples of such database is MongoDB, CouchDB etc.

Wide column stores also referred by some as extensible record stores seem to have been motivated by Google’s success with BigTable [10]. A column-store stores each attribute in a database table separately, such that successive values of that attribute are stored consecutively [11]. It can be argued that the equivalent of relational databases for Big Data, retaining the notion of tables, rows and columns [5]. Wide Column databases are based on a hybrid approach that relies on relational databases declarative characteristics and various key-value stores schema [4]. Examples of these includes HBase, Cassandra, and Accumulo etc.
Graph Databases are suitable to store not only information about objects but also all relationships that exist among them [4]. In this regard, Graph databases employs Graph theory concepts like nodes and edges. Nodes are entities in the data domain representing a tuple or row in a database, or an XML element and edges are the relationship between two entities like a foreign key/primary key relationship [5], [12]. Examples include Neo4J, and OrientDB. However, the focus of this research work is on key value data stores.

1.1. In-Memory and On Disk Key-Value Data stores

Data stores can be classed as either In-Memory or On Disk data stores. When in-memory data stores run, data is entirely loaded into memory, so all its operations are run from memory [13]. Typically, such systems may require that data be stored periodically and asynchronously on disk but all working data is retrieved from memory. [10] Shows that in-memory data can be copied to disk for backup or system shutdown. The key advantages of in-memory data stores are low latency and improved throughput. In-memory key-value storage also requires low overhead network communication between clients and servers [13], this contributes significantly to its high throughput. Redis and Memcached are typical examples of in-Memory key-value Data stores [7], [14].

Another Category of key-value data stores are Disk-based key-value data stores. Typically, disk-based data stores can be Distributed storage systems [15], or Single node platforms, storages that hold data in hard disk with frequently accessed data in memory, or data stores that can store data in RAM, but it also permits plugging in a storage engine [10]. The advantage of on disk capabilities includes reducing cost per byte of storage and increasing storage capacity. Indeed, on disk data stores can serve as alternative to in memory types when exhaustion of memory space is anticipated or expected [15]. Also, these are referred to as persistent key-value stores which include data stores such as BerkeleyDB, Voldemort and Riak [16].

1.2. Redis

Redis is an open source (BSD licensed), in-memory data structure store, used as database, cache and message broker [14]. The data model is key-value, although many different kinds of data types are supported: Strings, Lists, Sets, Sorted Sets, Hashes, HyperLogLogs, and Bitmaps [17]. Redis has built-in replication, it can be replicated using the master-slave model and a master can have multiple slaves [14], [3]. Redis provides access to mutable data structures via a set of commands, which are sent using a server-client model with TCP sockets and a simple protocol [17]. Redis can be used as a Least recently used (LRU) cache, using an approximate LRU algorithm to evict old data as a new one is added [14]. It also offers scripting capability using Lua, a powerful, light-weight scripting language written in C and embedded in Redis [18]. Redis supports automatic failover if a master is not working as expected using a feature called Redis Sentinel, this starts a failover process where a slave is promoted to master, additional slaves are also reconfigured to use the new master [14]. Sharding is executed via Redis cluster a platform where data is automatically sharded across multiple Redis nodes [14].

1.3. SSDB

SSDB is a fast NoSQL database for storing big list of billions of elements; it supports data structures including Key-Value pair, List, Map or Hash and Sorted Set [19]. SSDB is written in C/C++ with Google LevelDB as its storage engine [20]. Conceived by its author as an alternative to Redis [19, 20], it supports Redis network protocol and open-sourced Redis clients [21]. SSDB other features include Replication

(Master-Slave and Master-Master configuration), Time to Live key expiration (can serve as a persistent cache service) [21] and easy to use client APIs for Development and Deployment.

2. RESEARCH METHOD

2.1. Benchmarking Tool and System Specification

Benchmarking of NoSQL Databases with any standard benchmark tools is done using two approaches namely: trace and vector-based load (database operations) generation [22], [23]. Trace based benchmark systems use actual application workload generated from specific applications overtime. Whilst Vector based benchmark systems creates application behaviours using vectors and applying the vectors using known statistical distribution models, mimicking actual application request and response in hardware or virtual platform. Benchmark tools can be classified as either inbuilt or custom. An example of the former is redis-benchmark. Custom benchmark tools include Yahoo! Cloud System Benchmark (YCSB) [24], BigBench [25] and GraySort [26]. The objective of this benchmarking process is to compare the performance of Redis and SSDB NoSQL databases using single node instances. The results would validate the claims of SSDB's suitability as an alternative to Redis as the authors have suggested [19]. The tool of choice is YCSB, YCSB's plugin-based architecture and ease of extensibility using scripts [27] makes it a splendid choice. [4] details YCSB's use in measuring the performance of four NoSQL systems including Redis. [2] Describes two YCSB benchmark tiers namely: Performance and Scaling. The focus for this study is to Benchmark SSDB's performance in comparison to Redis for a range of Workloads. These workloads imitate a variety of web application request behaviours like heavy read and write scenarios. The workloads considered includes:

Workload A (Heavy Updating)
In this workload 50% of the operations are reads and 50% are writes .
Workload B (Heavy Read)
In this workload 95% of the operations are reads and the rest 5% are writes.
Workload C (Only Read)
Workload with 100% read operations.
Workload D (Read Latest)
Workload with 95% read Operations and 5% insert operations. Workload inserts new records and the most recently inserted records are the most popular.
Workload E
Workload with 95% Short ranges Scan Operations and 5% insert operations. Workload queries short ranges of records, instead of individual records.
Workload F

Workload where the client will read a record, modify it, and write back the changes. Table 1 shows the details system configuration and specification for the benchmarking process.manuscript.

Table 1: Detailing System configuration and Specification

Processor	Intel Pentium CPU B960
Clock Speed	2.20GHz
Number of Cores	2
Number of Threads	2
Host Instruction Set	64 bit
Host Operating System	Windows
Host Memory	4096MB
Virtual Operating System	CentOS 6.3
Virtual Memory	756MB
Virtual Instruction Set	32 bit
Kernel	Linux 2.6.32-279.el6.i686
Java	Java SE Runtime 1.8.0_45
YCSB Version	0.1.4
Redis Version	3.2.0
SSDB Version	1.9.3
Virtual Machine	Oracle VirtualBox

3. RESULTS AND ANALYSIS

3.1. Workload A (Heavy Updating)

The results of the Heavy Update operations are shown in Figures 1, Figure 2 and Figure 3. In this result, SSDB outperforms Redis as the number of threads increases and its throughput diverges significantly with an increase in the number of threads. This suggests that an increase in software threads will increase the performance of SSDB as shown in the Figures 1-3

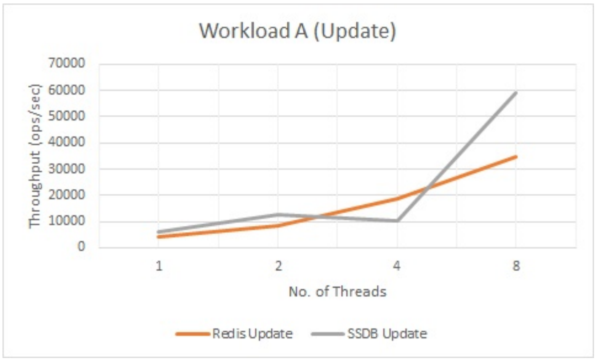

Figure 1. Comparison of Redis and SSDB with 1000 Records

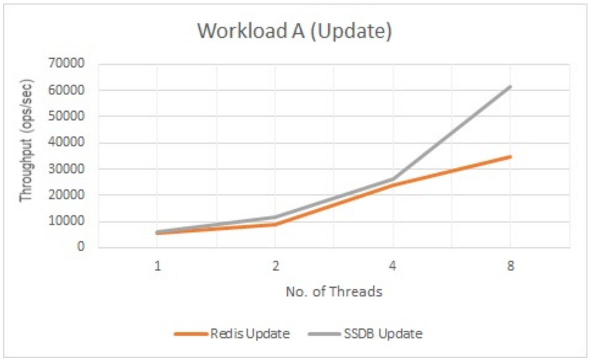

Figure 2. Comparison of Redis and SSDB with 5000 Records

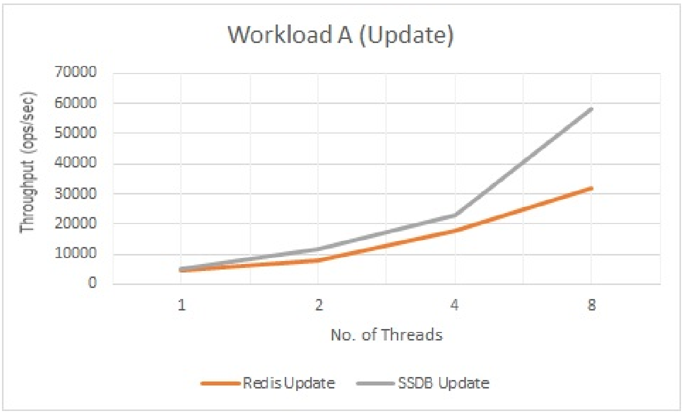

Figure 3. Comparison of Redis and SSDB with 10000 Records and 1000 Operations

3.2. Workload B (Heavy Read)

The result of the Heavy Read workload as shown in Figure 4, Figure 5 and Figure 6 has similarity with the heavy update operations. SSDB outperforms Redis for Heavy Read operations as shown in Figure 1, Figure 2 and Figure 3. However this advantage of performance seems to be lost as the amount of records approaches 10,000. Redis clearly seems to recover any lost grounds at this level.

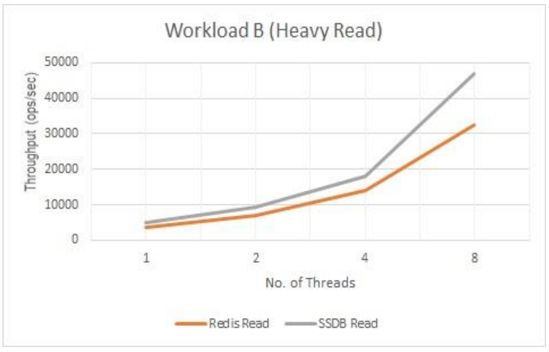

Figure 4. Comparison of Redis and SSDB with 1000 Records and 1000 Operations

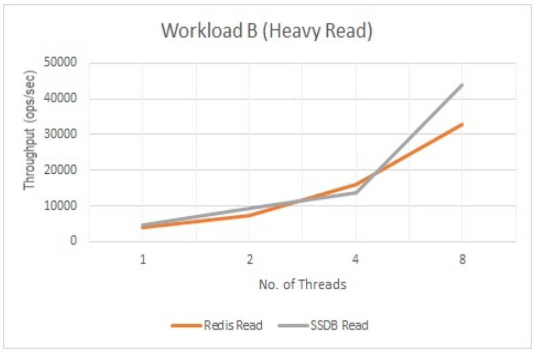

Figure 5. Comparison of Redis and SSDB with 5000 Records and 1000 Operations

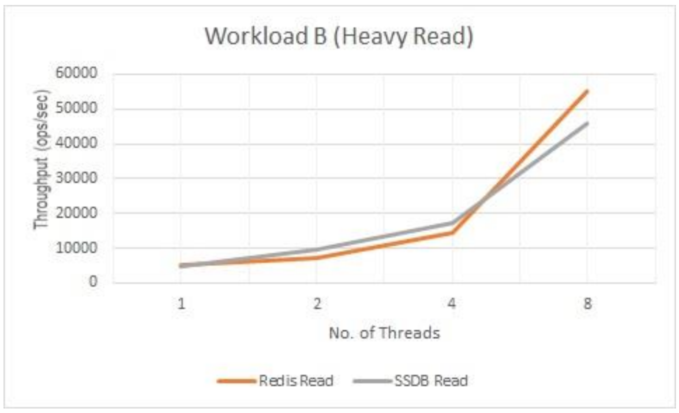

Figure 6. Comparison of Redis and USSDB with 10000 Records and 1000 Operations

3.3. Workload C (Only Read)

The result of the throughput for SSDB as shown in Figures 7, Figure 8 and Figure 9 diverges significantly against that of Redis as the number of threads deployed increases.

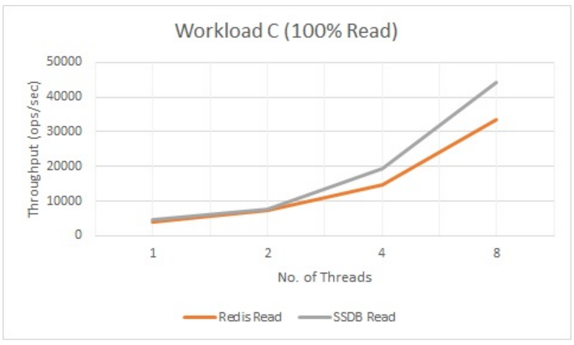

Figure 7. Comparison of Redis and SSDB with 1000 Records and 1000 Operations

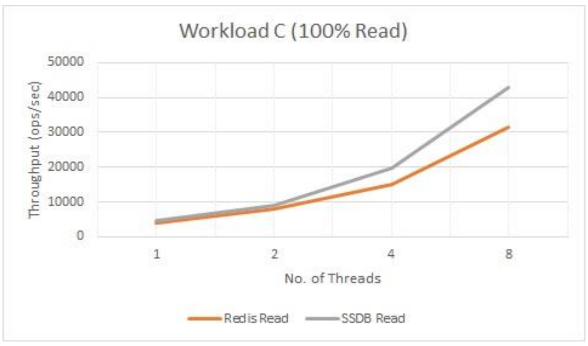

Figure 8. comparison of Redis and SSDB with 5000 Records and 1000 Operations

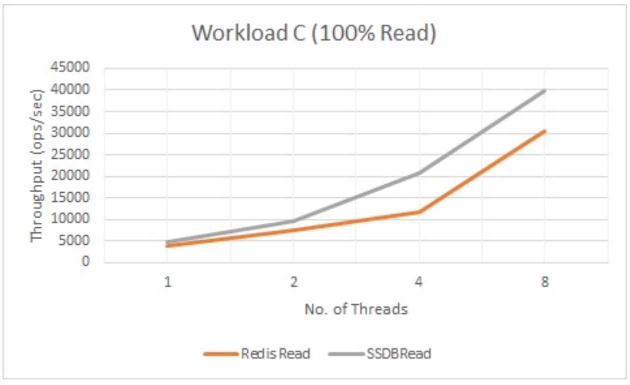

Figure 9. Comparison of Redis and SSDB with 10000 Records and 1000 Operations

3.4. Workload D (Read Latest)

The Figure 10, Figure 11 and Figure 12 shows, how the result of SSDB throughput surpasses that of Redis. This also shows a significant boost of the throughput as the number of threads increases to 8.

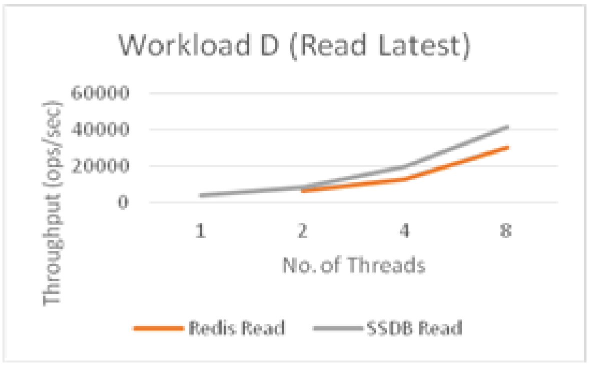

Figure 10. Comparison of Redis and SSDB with 1000 Records and 1000 Operations

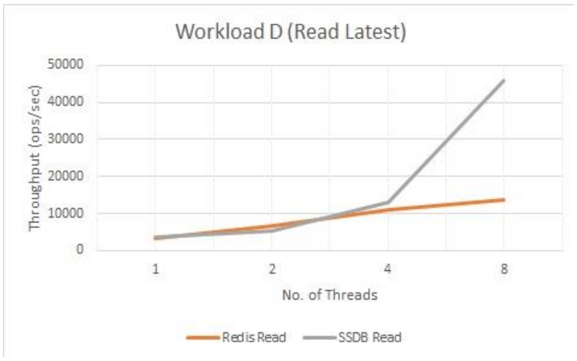

Figure 11. Comparison of Redis and SSDB with 5000 Records and 1000 Operations

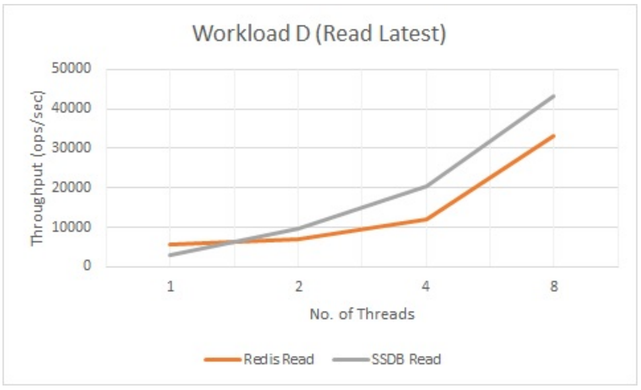

Figure 12. Comparison of Redis and USSDB with 10000 Records and 1000 Operations

3.5. Workload E

The Figure 13, Figure 14 and Figure 15 shows, how the Redis clearly outperforms SSDB in terms of throughput. This indicates that the SSDB is unsuitable for SCAN operations.

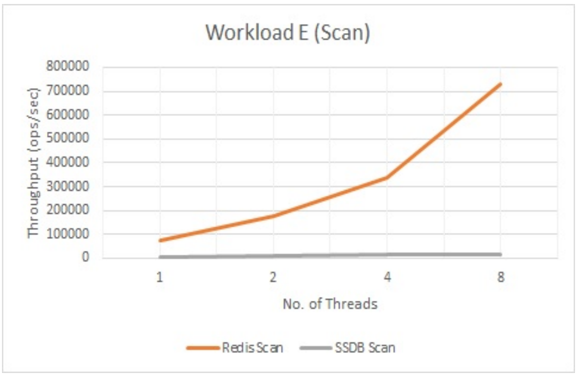

Figure 13. Comparison of Redis and SSDB with 1000 Records and 1000 Operations

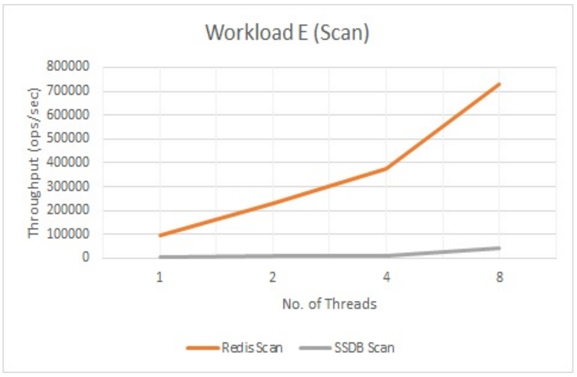

Figure 14. Comparison of Redis and SSDB with 5000 Records and 1000 Operations

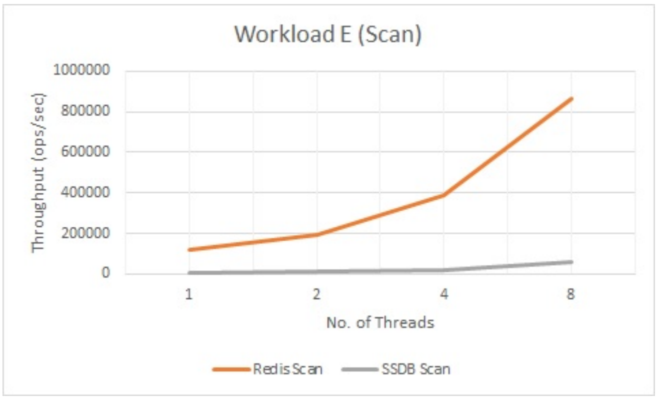

Figure 15. Comparison of Redis and SSDB with 10000 Records and 1000 Operations

3.6. Workload F

In the Read-Modify-Write Workload as shown in Figure 16, Figure 17 and Figure 18, there are shades of similarity to Workload D (Read Latest) see Figure 10 to Figure 12. SSDB paces Redis for outputs from 1 2, and 4 threads respectively. However there is a divergence when thread length increases to 8.

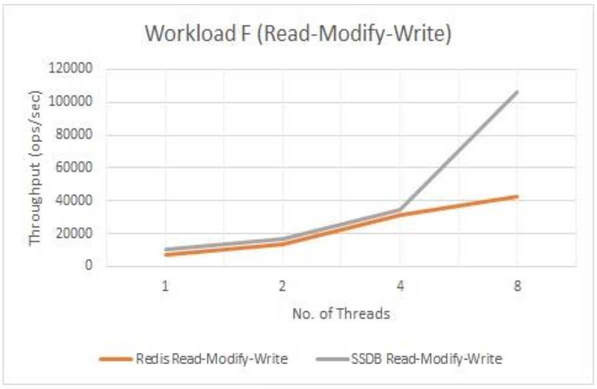

Figure 16. Comparison of Redis and SSDB with 1000 Records and 1000 Operations

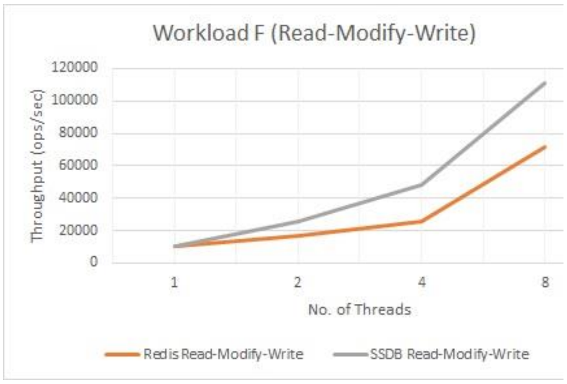

Figure 17. Comparison of Redis and SSDB with 5000 Records and 1000 Operations

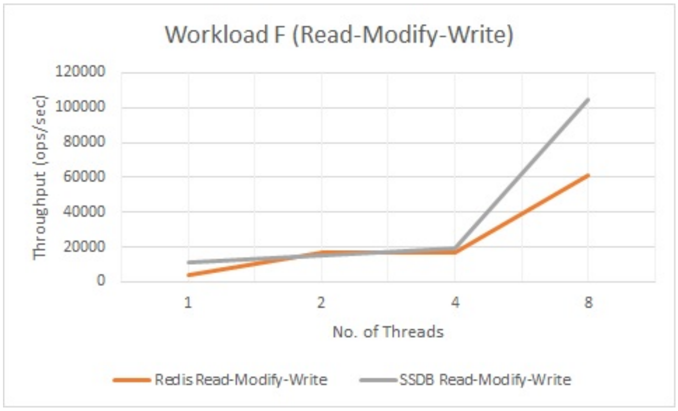

Figure 18. Comparison of Redis and SSDB with 10000 Records and 1000 Operations

For the series of experiments conducted, only single node instances of both databases were used. The test carried out involved all six (6) generic workloads of YCSB (Workload A to Workload F). SSDB performs better than Redis's throughput in heavy read operations by a significant margin. However, the margin indicates a significant reduction with read-modify and write operations. This shows that developers can consider adopting SSDB for elasticity in application scenarios where updates, heavy read & read-modify and write operations are undertaken. While, SSDB's claim to be a suitable alternative to Redis has been shown to be valid for Five out of six workloads, it is not suited for short or small (<10,000) range scan queries.

4. CONCLUSION

In conclusion, the phenomenoms that has led to the increased visibility of NoSQL is the rising need for unstructured data and documents in Mobile computing and Social network websites [28]. Whilst cheaper and faster memory units are not ruled out, the growing trend of Big Data and the Internet of Things is decentralized database systems that improve fault tolerance in database systems. Performance concerns are not only influenced by location alone but by data security issues also. Security concerns arise because of the nature and characteristics of Big Data (the huge volume, velocity, variety and veracity of data) [29]. One of such concerns is how to query encrypted database systems without degrading performance of applications [30, 31]. Although this study has not addressed such security issues it is certain that this will feature in lots of studies going forward.

ACKNOWLEDGEMENTS
We acknowledge the support of Covenant University in conducting this research and the cost of publication.

REFERENCES

[1] C.U. Kumarasinghe, K.L.D.U. Liyanage, W.A.T. Madushanka and R.A.C.L. Mendis. (2015, September). Performance Comparison of NoSQL Databases in Pseudo Distributed Mode: Cassandra, MongoDB & Redis [Online]. Available: https://www.researchgate.net/profile/Tiroshan_Madushanka/publication/281629653_Performance_Comparison_of_NoSQL_Databases_in_Pseudo_Distributed_Mode_Cassandra_MongoDB_Redis/links/55f113ba08aedecb68ffd294.pdf, Accessed: July. 1, 2016

[2] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with ycsb". In *Proceedings of the 1st ACM symposium on Cloud computing* (New York, NY, USA, 2010), SoCC '10, ACM, pp.143-154.

[3] Y. Abubakar, T. S. Adeyi, and I. G. Auta, "Performance Evaluation of NoSQL Systems using YCSB in a Resource Austere Environment," *International Journal of Applied Information Systems*, vol. 7, pp. 23-27, Sep. 2014.

[4] [A. Oussous, F.Z. Benjelloun, A.A. Lahcen, S. Belfkih, "Comparison and Classification of NoSQL Databases for Big Data," in *Proc. of the 2015 International Conference on Big Data, Cloud and Applications*, *BDCA* 2015, 25-26 May 2015, Tetuan, Morocco [Online]. Available: ResearchGate , https://www.researchgate.net. [Accessed: 24 June 2016].

[5] H. Khazaei, M. Fokaefs, S. Zareian, N. Beigi-Mohammadi, B. Ramprasad, M.Shtern, P. Gaikwad, and M. Litoiu. "How do I choose the right NoSQL solution?A comprehensive theoretical and experimental survey." In: Submitted to *Journal of Big Data and Information Analytics(BDIA)*(2015).[Online].Available: https://www.researchgate.net/profile/Hamzeh_Khazaei/publication/282679529_How_Do_I_Choose_The_Right_Nosql_Solution_A_Comprehensive_Theoretical_And_Experimental_Survey/links/5618781808ae044edbad2437.pdf. Accessed:Jun. 29, 2016.

[6] E. Anderson, X. Li, M. A. Shah, J. Tucek, and J. J. Wylie, "What consistency does your key-value store actually provide?" *HotDep*, vol. 10, pp. 1–16, 2010.

[7] B. Atikoglu, Y. Xu, E. Frachtenberg, S. Jiang, and M. Paleczny. Workload analysis of a large-scale key-value store. *In Proceedings of the SIGMETRICS'12*, June 2012.

[8] K. Ma, A. Abraham, "Toward lightweight transparent data middleware in support of document stores" in WICT 2013: *Proceedings of the third World Congress on Information and Communication Technologies* (2013)

[9] C. Chasseur, Y. Li, and J. M. Patel. Enabling JSON document stores in relational systems. In *Proceedings of the 16th International Workshop on the Web and Databases (WebDB)*, pages 1-6, 2013.

[10] R. Cattell, "Scalable sql and nosql data stores," *ACM SIGMOD Record*, vol. 39, no. 4, pp. 12–27, 2010.

[11] D.J. Abadi. Column stores for wide and sparse data. In *CIDR*, Asilomar, CA, USA, 2007.

[12] V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, R. Desai, and H. Karambelkar, "Bidirectional expansion for keyword search on graph databases," in *Proc. of VLDB Conference*, 2005, pp. 505-516.

[13] J. Han, E. Haihong, G. Le, and J. Du, "Survey on NoSQL database," In *Pervasive Computing and Applications (ICPCA), 2011 6th International Conference on*, Oct. 2011, pp. 363-366.

[14] [Online].Available: http://redis.io Accessed: July. 1, 2016.

[15] Online].Available:http://www.ideawu.com/blog/post/category/ssdb, Accessed: July. 1, 2016

[16] Katarina Grolinger, Wilson A. Higashino, Abhinav Tiwari and Miriam A. M. Capretz, “Data management in cloud environments: NoSQL and NewSQL data stores”, Journal of Cloud Computing: Advances, Systems and Applications 2013, 2:22; http://www.journalofcloudcomputing.com/content/2/1/22
[17] [Online].Available:https://github.com/antirez/redis Accessed: July. 1, 2016.
[18] R. Ierusalimschy, L. H. de Figueiredo, W. Celes. Lua 5.1 Reference Manual. Lua.org, 2006.
[19] [Online].Available: https://ssdb.io Accessed: July. 2, 2016.
[20] [Online].Available:https://github.com/ideawu/ssdb Accessed: July. 2, 2016.
[21] [Online].Available:http://www.ideawu.com/blog/post/category/ssdb Accessed: July. 2, 2016.
[22] R. Sears, C. Van Ingen, and J. Gray, “To BLOB or not to BLOB: Large object storage in a database or a filesystem?,” arXiv Prepr. cs/0701168, pp. 1–11, 2007.
[23] M. Seltzer, D. Krinsky, K. Smith, and X. Zhang, “The case for application-specific benchmarking,” in *Hot Topics in Operating Systems, 1999. Proceedings of the Seventh Workshop on*, 1999, pp. 102–107.
[24] B. Cooper, YCSB: Yahoo! Cloud Serving Benchmark. 2017.
[25] T. Rabl et al., “BigBench Specification V0. 1,” in *Specifying Big Data Benchmarks*, Springer, 2014, pp. 164–201.
[26] “Sort Benchmark Home Page.” [Online]. Available: http://sortbenchmark.org./. [Accessed: 01-Jan-2018].
[27] H. Khazaei et al., “How do I choose the right nosql solution? a comprehensive theoretical and experimental survey.”
[28] S.-H. Jung, J.-C. Kim, and C.-B. Sim, “Prediction Data Processing Scheme using an Artificial Neural Network and Data Clustering for Big Data,” *International Journal of Electrical and Computer Engineering (IJECE)*., vol. 6, no. 1, p. 330, 2016.
[29] S. A. Thanekar, K. Subrahmanyam, and A. B. Bagwan, “Big data and MapReduce challenges, opportunities and trends,” *International Journal of Electrical and Computer Engineering (IJECE).*, vol. 6, no. 6, pp. 2911–2919, 2016.
[30] Y. D. Jang and J. H. Kim, “A comparison of the query execution algorithms in secure database system,” *International Journal of Electrical and Computer Engineering (IJECE),* vol. 6, no. 1, pp. 337–343, 2016.
[31] Chinonso, Okereke, Osemwegie Omoruyi, Kennedy Okokpujie, and Samuel John. "Development of an Encrypting System for an Image Viewer based on Hill Cipher Algorithm", *Covenant Journal Of Engineering Technology* 1, no. 2 (2017).