Design and Analysis of High Performance Crypt-NoSQL

Ming-Hung Shih and J. Morris Chang

***Abstract*—NoSQL databases have become popular with enterprises due to their scalable and flexible storage management of big data. Nevertheless, their popularity also brings up security concerns. Most NoSQL databases lacked secure data encryption, relying on developers to implement cryptographic methods at application level or middleware layer as a wrapper around the database. While this approach protects the integrity of data, it increases the difficulty of executing queries. We were motivated to design a system that not only provides NoSQL databases with the necessary data security, but also supports the execution of query over encrypted data. Furthermore, how to exploit the distributed fashion of NoSQL databases to deliver high performance and scalability with massive client accesses is another important challenge. In this research, we introduce Crypt-NoSQL, the first prototype to support execution of query over encrypted data on NoSQL databases with high performance. Three different models of Crypt-NoSQL were proposed and performance was evaluated with Yahoo! Cloud Service Benchmark (YCSB) considering an enormous number of clients. Our experimental results show that Crypt-NoSQL can process queries over encrypted data with high performance and scalability. A guidance of establishing service level agreement (SLA) for Crypt-NoSQL as a cloud service is also proposed.**

***Index Terms*—Big Data, NoSQL, Cassandra, Security, Performance, Cloud Service.**

I. Introduction

As the amount of data increases dramatically, new technology to efficiently store big data is in urgent need. Traditional databases like Oracle and MySQL are becoming inadequate to fulfill the need of big data analytics and storage because of their relational and well-structured models. For example, big data analytics could involve data collected from many different resources such as social media or log files on machines. These data are not well-structured and moreover, the structure may change from time to time [1]. Relational database management systems (RDBMs) have limited supports of unstructured data or schema changes. As a result, NoSQL database systems have been developed to resolve the limitations of RDBMs over the past few years. While NoSQL databases have become popular among enterprises, their poor security measures keep most database developers at bay. In this paper, we introduce the security concerns about data encryption on NoSQL databases, and we propose a solution to not only enforce the data security, but leverage the high performance and scalability of NoSQL.

NoSQL, known as Not-Only SQL, stands for non-relational database management systems. Comparing to RDBMs, NoSQL databases provide more support to semi-structured or unstructured data, including the capability of changing database schema if needed. Furthermore, because of the distributed and clustered structure, NoSQL databases can deliver high performance, high data availability, and high scalability to their clients. These are all very important advantages over traditional databases for big data analytics.

Different types of NoSQL databases have been developed based on different demands. Generally speaking, NoSQL databases on the current market can be categorized into the following four types [2]: (1) Key-value: Riak, Redis, Windows Azure Storage. (2) Column-family: HBase, Cassandra, Google BigTable. (3) Document-oriented: MongoDB, CouchDB. (4) Graph: Neo4J. Each type of those NoSQL databases aims at providing a solution to different challenges in big data analytics. Based on a popularity survey from DB-Engine ranking [3] by March 2016, Redis, Cassandra, and MongoDB are the top-3 NoSQL databases that most developers would choose.

Despite the tradeoffs from CAP theorem [4], security is a more important issue people concerned about when migrating from relational databases to NoSQL databases. As addressed by the Cloud Security Alliance (CSA) in 2013 [5], most NoSQL data stores suffer from weak authentication and no protection of data integrity. Data encryption is the most common method to protect data integrity, and it is usually implemented at application level or middleware layer by database users. As a result, NoSQL databases cannot directly perform any requested queries over encrypted data in storage. A typical way to retrieve encrypted records is to give every record an index in plaintext so that NoSQL databases would be able to retrieve requested records based on given indices. Nevertheless, it limits the execution of most queries supported by the original NoSQL databases. In order to overcome this difficulty, we were motivated to study the possibility of directly executing queries over encrypted data.

CryptDB [6] is a well-known system developed by Popa *et al.* at MIT that utilized a proxy server to translate and rephrase the clients query before it goes into database, making this query executable over encrypted contents. They apply several encryption schemes that are called SQL-aware to data, which means the execution of some specific queries over the encrypted contents will become possible by using their encryption schemes. More details of those encryption schemes will be discussed in Section III.

When CryptDB was developed, they only considered relational databases. However, NoSQL databases have been getting more and more attentions from researchers in the last few years. A study on how to design and implement a system

978-1-5090-5569-2/17/$31.00

for NoSQL databases that provides the same functionality as CryptDB, which is the ability to query over encrypted data, is proposed. In addition, system performance is also one of the concerns in the era of big data research since a database may be accessed by a huge amount of clients at the same time. Therefore, testing of many clients' requests with performance measurements is also considered in this study.

Our research aims to accomplish (1) provide the full functionality of query over encrypted data on NoSQL databases, and (2) analyze the performance of our system models, with a design of cryptographic system specifically for NoSQL databases, Crypt-NoSQL, that supports query over encrypted contents with high performance. In this research, we implemented the ideas of Crypt-NoSQL system on a Cassandra database, along with a testing environment to evaluate the system performance by using Yahoo Cloud Service Benchmark (YCSB) [7]. We also considered Crypt-NoSQL as a potential cloud service, and provided a guidance to establish the service level agreement (SLA) based on experimental results.

This paper is organized as follows. In Section II, we enumerate previous research related to the security and performance of NoSQL databases. In Section III, we introduce the cryptographic part of the proposed Crypt-NoSQL system. In Section IV, we design different system models for the proposed system and evaluate their performance. We represent the experimental results in Section V. Section VI will summarize our work along with future works.

II. Related Work

Database security has been a very important issue for decades. Previous work in this area mainly focused on relational databases. However, as more and more data being collected everyday, relational databases become adequate as a data storage. NoSQL databases were developed to conquer certain challenges in big data, but they also brought more challenges to database security. Cloud Security Alliance (CSA), a non-profit organization that studies on cloud computing and its security issues, highlighted top-ten issues in their research report in 2013 [5]. These ten issues included the study of securing non-relational data stores. Based on their research, security was not a major concern when most NoSQL databases were designed. Weak or even zero security protection was provided by these data stores. Okman *et al.* [8] also pointed out several security issues of NoSQL databases in their research. They mainly focused on two most popular NoSQL databases, MongoDB and Cassandra, and analyze their security problems for the perspectives of data-at-rest protection, user authentication, user authorization, auditing, etc. They concluded that these two NoSQL databases lack of data encryption support and their authentication schemes are weak.

Besides security issues, many researchers are also interested in the overall performance of different NoSQL data stores. Cooper *et al.* [7] from Yahoo! have developed a standardized benchmark scheme, YCSB, in order to evaluate performance of each NoSQL database. Their benchmark generates a workload with operations (read, update, insert, and delete) and perform them on a specified NoSQL database to test its performance. The performance evaluation includes overall throughput, read latency, and write latency. Klein *et al.* [9] studied on the performance of Riak, Cassandra, and MongoDB respectively, with their custom electronic health records (EHRs) by using YCSB. Their study shows that Cassandra database offers the highest throughput yet also the highest latency among the three databases. Waage and Wiese [10] used YCSB as a benchmark to test the performance overhead of data encryption on Cassandra and HBase. Their result shows that enabling encryption will slow down about 10% - 40% of system throughput. This could be considered as a trade-off between system performance and data security.

III. Query Over Encrypted Data

As described in Section I, the first objective of Crypt-NoSQL is to provide the functionality of query over encrypted data on NoSQL databases. In this research, Cassandra is selected as the targeted NoSQL database. Since CryptDB was designed for SQL databases, if we want to adapt CryptDB into Cassandra, we must redesign some major components so the system could fully support Cassandra query language (CQL) over encrypted data. In this section, we review the major concepts of cryptography in CryptDB, and introduce how to design the functionality of the proposed system.

A. Cryptography in CryptDB

CryptDB applies several SQL-aware encryption schemes to data so that the execution of some specific queries over these encrypted data would be possible. There are several types of SQL-aware encryption introduced by CryptDB: Random (RND), Deterministic (DET), Order-Preserving Encryption (OPE), Homomorphic (HOM), Join (JOIN/OPE-JOIN), and Word Search (SEARCH).

RND stands for an encryption scheme that the same plaintext will result in random ciphertexts every time. This is the strongest protection to the data and we could achieve this objective by applying Advanced Encryption Standard (AES) with a random initialization vector (IV). The random ciphertext here is considered computationally secure and it also prevents the data from plaintext attacks. However, data encrypted with RND will not allow any query to be performed. DET is a scheme that the same plaintext will produce the same ciphertext every time when the encryption is performed, which means each encrypted value is corresponding to a specific data value. This could be achieved by using AES with zero or a fixed IV, which can be considered the same as AES electronic codebook mode (ECB). DET is less secure than RND since it provides the information of plaintext-ciphertext pair, yet it also grants us the ability to perform queries with equality check. OPE is another scheme that reveals the order relations between data, therefore its less secure than DET yet it provides the functionality of queries that require the order of data. HOM scheme supports the queries that contain calculation of data records. JOIN and SEARCH schemes support the join operations and like operations in SQL.

TABLE I
COMPARISON BETWEEN SQL AND CQL.

SQL Operations	Corresponding CQL
JOIN	Not supported.
GROUP BY	WITH CLUSTERING ORDER BY
SELECT * FROM *table* WHERE *id*='1';	Same, when *id* is a partition key.
SELECT * FROM *table* WHERE *field*='f1';	Not recommended. Secondary index is required.
INSERT INTO *table*(*id*, *field*) VALUES ('1','f1');	Same, except the usage of "IF EXISTS" is suggested.
UPDATE *table* SET *field* ='f1' WHERE *id*='1';	Same, when the partition key was given.
DELETE FROM *table* WHERE *id*='1';	Same, when the partition key was given.

TABLE II
SUPPORTED OPERATIONS IN OUR SYSTEM.

READ	SELECT * FROM *table* WHERE *id*='1';
UPDATE	UPDATE *table* SET *field*='f1' WHERE *id*='1';
INSERT	INSERT INTO *table*(*id*, *field*) VALUES ('1','f1');
DELETE	DELETE FROM *table* WHERE *id*='1';

TABLE III
SUPPORTED OPERATIONS AFTER ENCRYPTION.

READ	SELECT * FROM *table* WHERE *id*=DET('1');
UPDATE	UPDATE *table* SET *field*=RND('f1') WHERE *id*=DET('1');
INSERT	INSERT INTO *table* (*id*, *field*) VALUES (DET('1') , RND('f1'));
DELETE	DELETE FROM *table* WHERE *id*=DET('1');

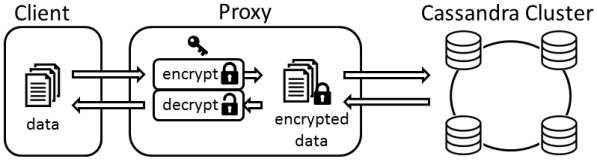


Fig. 1. System structure of our Crypt-NoSQL.

B. Cryptography in Crypt-NoSQL

We chose Cassandra as the NoSQL database in our Crypt-NoSQL. Therefore, we have to study on the difference between CQL and the traditional SQL before we design this system. We conclude the comparison as shown in Table I. CQL and SQL have similar operations except JOIN operations are not supported in CQL at all. SELECT operations in CQL are also slightly different from SQL: in SQL, searching values in any column is always supported. However, Cassandra does not recommend the search of column values because this requires the usage of secondary index and it reduces Cassandra's overall performance under certain cases [11]. Cassandra relies on the partition key to indicate the locations of each data record, since each data record may have multiple replicas stored in different Cassandra nodes depending on the setting of replication factor. Therefore, it's important that each query in Cassandra must specify the partition key.

Based on our study, most queries supported in CQL only require RND, DET, and OPE encryption schemes from CryptDB. In this research, we implemented RND and DET in our system so that it can support related queries that involve equality checks. We leave OPE-related queries as a potential future work in our continuous research. With the implementation of RND and DET, our system will allow four major operations to be executed over encrypted contents: READ, UPDATE, INSERT, and DELETE. An example query format for each operation is shown in Table II, assuming we have created a table *table*(*id*, *field*) in Cassandra database, where *id* is the partition key to this record, and *field* stores the data value of this record.

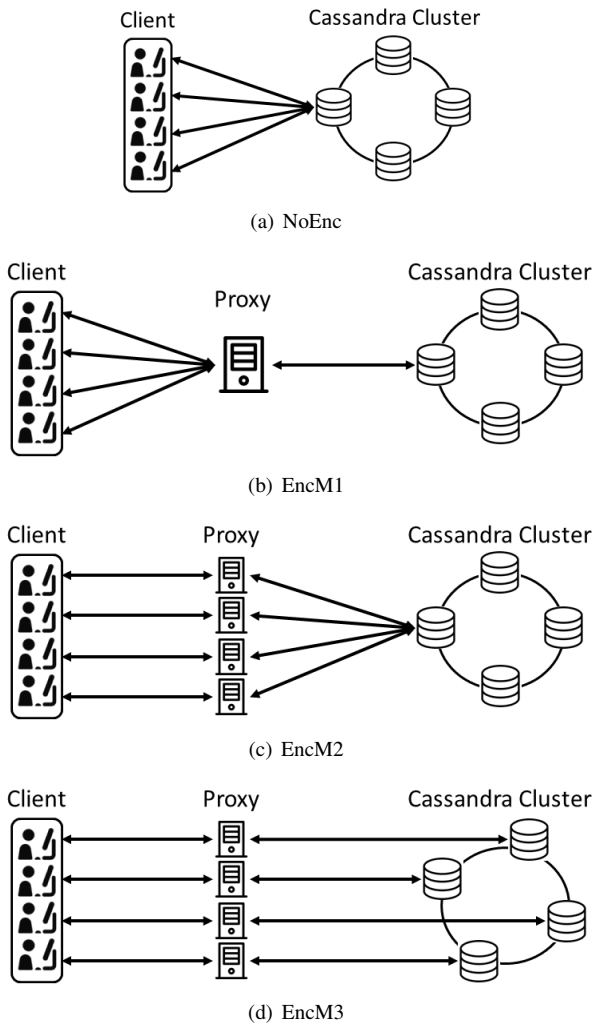
When writing a record into Cassandra (UPDATE or INSERT), we encrypt the *id* with DET and the value of *field* with RND. DET encryption allows *id* to be searched by future queries, and since searching the value of *field* is not recommended in Cassandra, we apply RND to the data values for the best security. When reading a record from Cassandra, a READ operation will first encrypt the *id* given by user's query, and use it to search the corresponding records in the database. After the encrypted record has been retrieved, an RND decryption to the data values must be applied. Each supported operation after applying RND and DET schemes is shown in Table III. These encryption schemes could be applied to data with a proxy-based structure similar to CryptDB. Therefore, we propose the structure of our Crypt-NoSQL system in Figure 1. In Section IV, we introduce different approaches of our Crypt-NoSQL and evaluate the system performance.

IV. PROPOSED SYSTEM STRUCTURE

The second objective is to analyze the performance of our proposed system. Unlike SQL databases that usually consist of only one centralized server machine, NoSQL databases can have a cluster that contains multiple machines. We leverage this property and proposed three different models (EncM1, EncM2, and EncM3) considering the distributed fashion of Cassandra database as shown in Figure 2. We evaluate the system performance of each model with a series of experiments in Section V.

A. NoEnc

NoEnc stands for "no encryption", as we want to see how much the performance overhead will be added when those query encryption schemes are applied. NoEnc involves one client machine that connects to one of the nodes in a Cassandra database cluster, where this node is called a "coordinator" in Cassandra as it coordinates client's requests. The client machine is installed with YCSB to generate multiple client threads to simulate the situation when many clients are accessing the service at the same time. Figure 2(a) illustrates the structure of NoEnc.



(a) NoEnc

(b) EncM1

(c) EncM2

(d) EncM3

Fig. 2. System structure of NoEnc, EncM1, EncM2, and EncM3.

B. EncM1

EncM1 is the first model of our proposed Crypt-NoSQL with one single proxy machine that handles the cryptography. As shown in Figure 2(b), an extra proxy machine was established between the client and Cassandra database comparing to NoEnc. This proxy machine will encrypt the data sent from client, and decrypt the data received from database, as mentioned earlier in Figure 1. In this research, we intended to design this proxy machine to be a dedicated server on the cloud that always listens to any incoming client request, unlike CryptDB which is more similar to an add-on application on the client's side. Extra performance cost due to connection between client and proxy machine is expected, but we think a dedicated proxy machine would be more realistic considering today's cloud computing structure.

C. EncM2

With increasing number of client threads, one single proxy machine may become a bottleneck due to network congestion. We design our second model, EncM2, that provides more proxy machines to clients as shown in Figure 2(c). Each proxy machine works in parallel with other proxy machines. It would be interesting to see if EncM2 could improve the system performance when more client threads are involved.

D. EncM3

Some NoSQL databases such as MongoDB were designed with a master-slave structure that every client must connect to a single master node first, and then this master node will distribute works to its slaves (similar to the structure of EncM2). However, Cassandra is well-known for its "master-master" property that each node in a Cassandra cluster are equal. In other words, every Cassandra node can be a coordinator to accept client's requests. This grants us the ability to design an extra model, EncM3, that connects multiple proxy machines to multiple Cassandra nodes in parallel as shown in Figure 2(d). We would like to see if this would result in better system performance than EncM2.

V. Experimental Results

Performance of each Crypt-NoSQL model under different numbers of client threads is evaluated in this section. The testing environment consists of three physically separated computers that belong to the same local network. Each machine plays a role of client, proxy, and database, respectively. We installed VMware Workstation 12 on each computer so that multiple virtual machines (VMs) could be created on each computer for simulation. For the client computer, it contains only one VM called ClientVM. For the proxy computer, it could have multiple ProxyVMs created for each experiment in this study. For the Cassandra database computer, we created four CassandraVMs as nodes in a Cassranda cluster, which means the size of this cluster is four. Each ClientVM and ProxyVM was assigned with a 2-core CPU and 2GB memory, and for each CassandraVM we assigned 1-core CPU and 4GB memory following the hardware requirements of Cassandra database [12]. For the software, each VM was installed with Ubuntu 14.04 LTS 64-bit as the operating system, along with Java SE Development Kit 8 (JDK 8) and Python 2.7.10 for system development purposes. We include more details of each VM as following.

ClientVM: installed with the latest YCSB 0.7.0 from [7] that supports Cassandra 2.x versions. It will generate multiple client threads with numbers of different requests based on workloads to evaluate the system performance. A YCSB workload specifies the portion of read and write operations that will be generated. In this research, we will use Workload A (50% reads 50% writes) and Workload B (95% reads and 5% writes).

ProxyVM: installed with our Java implementation of cryptography schemes introduced in Section III. AES-128 cipher block chaining mode (CBC) is chosen as the encryption method in both RND and DET. We also implemented the socket connections between ClientVM and ProxyVM so that proxy machines can be treated as a dedicated server which listens to the client requests all the time, as we mentioned in

Section IV. In addition, our ProxyVM will create a thread for each connected client thread coming from ClientVM so that all client threads can be processed in parallel.

CassandraVM: installed with Apache Cassandra version 2.1 to serve as a database server. In this study, four CassandraVMs were created to form a cluster with replication factor equals to four, and in total of 40000 data records generated by YCSB were pre-loaded into this database. We also set the consistency level at ClientVM to be always equal to one. With settings of replication factor and consistency level being fixed, we aim at controlling the performance variance caused by database settings and focusing on the changes caused by different structure of proxy machines.

Three types of performance measurements provided by YCSB are evaluated: throughput, read latency, and write latency. Throughput stands for the average number of operations being accomplished by the database per second, and it's defined as the total number of operations divided by the total elapsed time. Latency is the average response time of each operation, which starts from the moment when clients sends out the request and ends at the moment when client receives response from database. The latency could be calculated by YCSB for either read or write operations. A database system with high performance is to deliver higher throughput and lower latency.

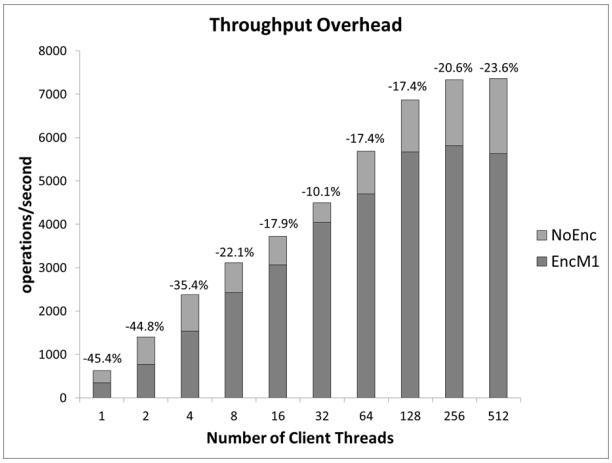
A. Proxy Overhead

In this experiment, we would like to evaluate the performance overhead caused by having a proxy machine. We observe the overhead by comparing the performance of NoEnc and EncM1. Each model was simulated with YCSB workload A with the number of client threads equals to 1, 2, 4, 8, ..., 256, and 512. Having 512 client threads simulates the situation when there are 512 clients accessing the system at the same time. Each configuration was performed ten times and the average value was calculated.

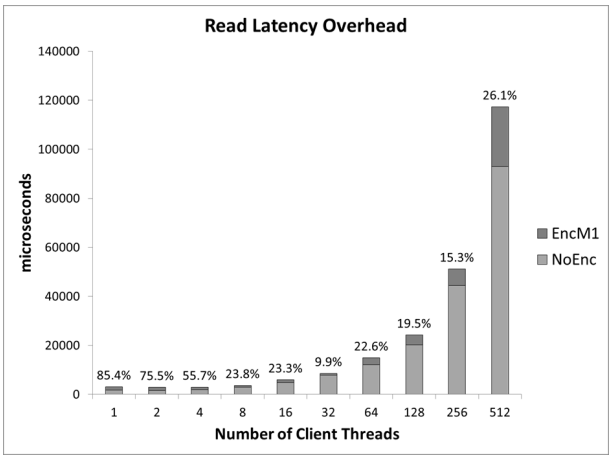
Figure 3 shows our experimental results regarding the overhead of throughput, read latency, and write latency under different numbers of client threads. When there's only one client, the proxy machine decreases system throughput by 45.4%, along with an increasing of read and write latency by 85.4% and 80.5% respectively. As the number of clients increases, the percentage of throughput overhead gradually decreases in Figure 3(a) because the percentage of each latency overhead also decreases in both Figure 3(b) and 3(c). Since the proxy machine can process all clients in parallel, the proxy becomes less dominant to the latency overhead when there are multiple clients accessing at the same time. However, more clients will also result in the congestion on the proxy machine, as we can see that after the number of clients reaches 256, the latency overhead becomes more dominant to the performance, resulting in an increasing percentage of throughput overhead.

B. Performance of Proxy Models

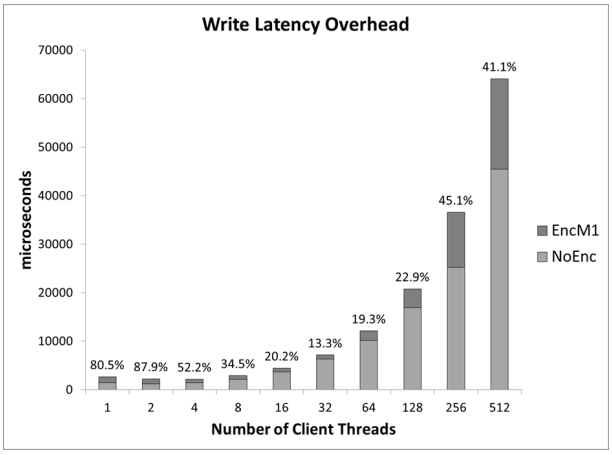
As mentioned in Section IV, we are interested in the performance of each model. In this experiment, we evaluated



(a) Throughput



(b) Read latency



(c) Write latency

Fig. 3. Performance overhead of throughput and latency when using proxy.

the throughput, read latency, and write latency, for each model respectively. Similar to previous experiment, we test each model with YCSB workload A and we increased the number of threads from 1 to 512. Again, the performance of each model was an average value of ten samples. The experimental results are shown in Figure 4.

Figure 4(a) shows the overall throughput of different models. Without a doubt, NoEnc yields the highest throughput of all time since it does not apply cryptography. EncM1 has similar throughput with EncM2 and EncM3 until the number of client threads exceeds 64, where EncM2 and EncM3 are having better throughput than EncM1 afterwards. This means that by having more proxy machines processing client threads in parallel, the system throughput could be greatly increased when the number of client threads is high. By comparing EncM2 and EncM3 together, we notice that they almost have identical curves regarding the overall throughput. The reason is because each node in Cassandra has the ability to "forward" client's requests to other nodes when it is congested. Therefore, no matter if you connect to one node (EncM2) or all nodes at the same time (EncM3), Cassandra will distribute the jobs to every node with a balance as a result of its master-master structure. One more thing worth noting is that the curves of throughput in EncM1, EncM2, EncM3, and even NoEnc, become flat after 256 client threads. This indicates a critical point that the system cannot produce a higher throughput and it sometimes even goes lower than before, possibly due to thread congestion and system limitations.

The average latency of read and write operations are shown in Figure 4(b) and 4(c). As expected, NoEnc has the lowest latency among all models. One interesting fact we observed is that having more proxy machines working in parallel (EncM2 and EncM3) does not reduce the read latency, but it does improve the write latency a lot as the curves are almost identical to NoEnc. We believe the reason is because read operations are required to send the data from database back to client, using more bandwidth than write operations between client and proxy since write operations only need to return a feedback for a successful transmission. Therefore, the read latency does not improve a lot even if more proxy machines are used.

As a conclusion of this experiment, NoEnc has yielded the highest throughput and the lowest latency among all models, but it does not provide cryptography. EncM1 has the lowest throughput and the highest latency since it only has one proxy machine in its structure. EncM2 and EncM3 are almost identically providing the same performance, which is better than EncM1 because multiple proxy machines are used. However, when client threads are less, EncM1 may be a better choice over EncM2 and EncM3 since the costs on those extra proxy machines may be saved.

C. Different Workloads

Previous experiments were simulated based on YCSB workload A, which is the combination of 50% read operations and 50% write operations. In this experiment, we would like to test workload B, another workload provided by YCSB that contains 95% read operations and only 5% write operations, to see if there's any effect on performance. Note that read operations have higher latency than write operations in Cassandra, as already being discovered by our previous experiments and other researches [9][10]. Since workload B contains more read operations, when the total number of operations is fixed, we expect to see a performance drop on overall throughput comparing to workload A. Figure 5 shows our experimental results on the performance of different proxy models using workload B.

As expected, the overall throughputs of all models with workload B are lower than the ones with workload A because read operations in Cassandra result in higher latency than write operations, and now we are performing more read operations within a fixed number of total operations. Note that the relationships between each model almost remained the same as in workload A though. One more thing worth mentioning is when in the experiment of workload A, the critical point of performance drop was at the case when client threads are larger than 256. However, in this experiment of workload B, we clearly see that the throughput dropped after the client threads exceeded 64. This indicates that when most clients are performing read operations at the same time, the system may not be able to sufficiently handle more than 64 clients. The results of latency are shown in Figure 5(b) and 5(c). As we may expect, the read latency of all models are higher than workload A under the same number of client threads, while the write latency roughly remain the same as workload A.

This experiment shows us that different workloads will result in different performances since read operations are slower in Cassandra. Choosing a suitable workload that best describes the percentage of read and write operations in practical is relying on the developers' decisions when simulating the system performance. However, as we can see from both Figure 4 and 5, the relationships between different proxy models remain the same: NoEnc is always having the highest throughput and the lowest latency, but it has no cryptography protecting the data. EncM1 is always having the lowest throughput and highest latency, but it only requires one proxy machine. EncM2 and EncM3 provide a solution between NoEnc and EncM1 with the extra costs of having more proxy machines working in parallel.

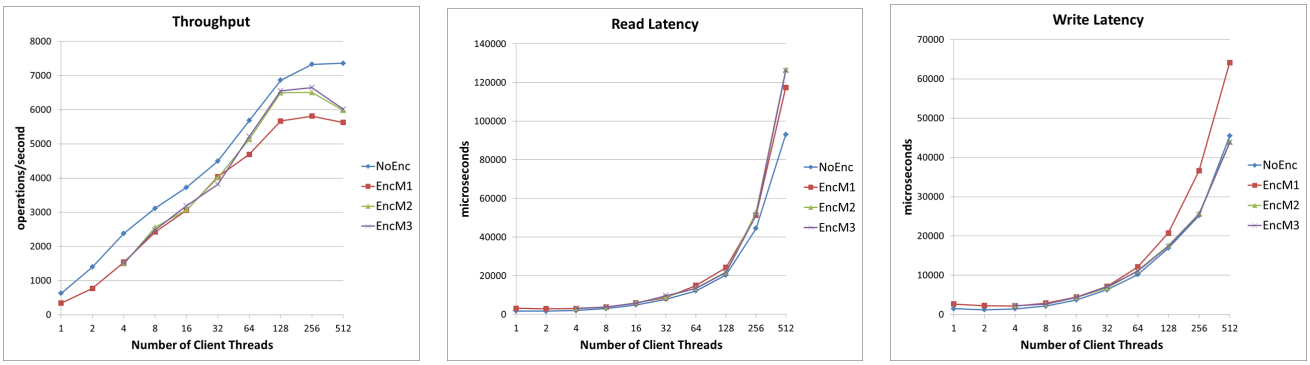
D. Crypt-NoSQL as a Cloud Service with SLA

With the number of clients increases, the performance of Crypt-NoSQL could be maintained by adding more proxy machines, along with extra operating costs. If considering Crypt-NoSQL as a cloud service, a well-defined SLA on the system performance must be established between service provider and service user to guarantee the quality of service. How to utilize simulation of system performance as a guidance to establish a SLA for Crypt-NoSQL should be further discussed. First, we define the Crypt-NoSQL SLA as following.

Crypt-NoSQL SLA: Suppose the service user has a number of clients $n \leq N$ accessing the service. With the operation of proxy machines $p \in P$, the service provider can guarantee

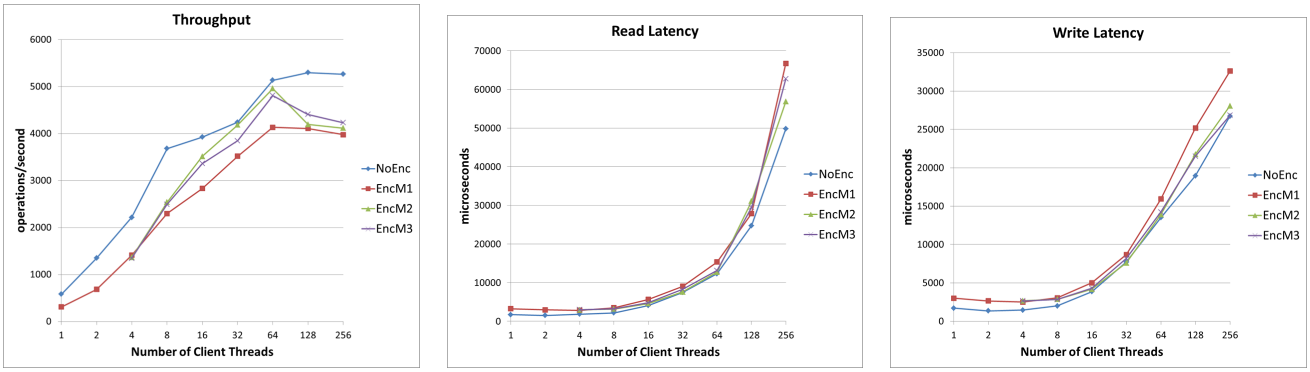
1) average throughput up to T (ops/sec)
2) average read latency lower than l_r (μs)
3) average write latency lower than l_w (μs)

, where N denotes maximum number of clients and P is a set of proxy machines operated by service provider.



(a) Throughput (b) Read Latency (c) Write Latency

Fig. 4. Performance evaluation of different models with workload A.



(a) Throughput (b) Read Latency (c) Write Latency

Fig. 5. Performance evaluation of different models with workload B.

Given any numbers of n and p, the guaranteed values of T, l_r, and l_w should be estimated properly. Performance evaluation of Crypt-NoSQL could help the service provider generate statistical models for throughput and latency with different numbers of client threads and proxy machines. We illustrate this idea a little further with a simple example.

Suppose the service provider runs simulation with n client threads and p proxy machines, where $n \leq 128$ and $p \in \{1,2,4\}$. Figure 6 represents the simulation results of throughput, read latency, and write latency. The next step is to model each performance metric with respect to n and p. We accomplish this step by introducing the data fitting tools in Matlab to fit all the data points into polynomial equations. With the help of `fit` function and the fitType set to `'poly23'`, each performance metric could be formulated into the following format of equation:

$$\begin{aligned} f(p,n) = & c_{00} + c_{10}p + c_{01}n + c_{20}p^2 + c_{11}pn \\ & + c_{02}n^2 + c_{21}p^2n + c_{12}pn^2 + c_{03}n^3 \end{aligned} \quad (1)$$

, where c_{ij} are coefficients derived from performance curves. Therefore, the throughput function $T(p,n)$, read latency function $l_r(p,n)$, and write latency function $l_w(p,n)$ could be derived respectively. Figure 7 displays a graphical view of throughput function $T(p,n)$, where the data points in blue color were fitted by a surface model. While we chose a polynomial model for this example, other statistical models such as exponential or linear model could also be considered.

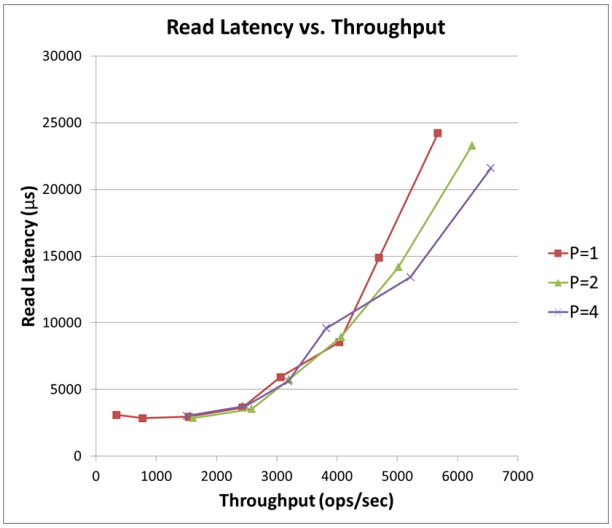
Based on the derived functions, Table IV could be generated with different numbers of n and p. For $n = 128$ clients, the service provider can offer three levels of services ($p = 1$, $p = 2$, or $p = 4$) with different service fees. If service user chooses $P = 2$ service, service provider guarantees the throughput will be up to 5673.0 (ops/sec) and the latency will be no higher than 24210.6 (μs) for read operations and 20803.9 (μs) for write operations. Obviously, more details are still required to establish a well-defined SLA, but we hope this example could initiate the discussion of SLA when considering Crypt-NoSQL as a cloud service.

TABLE IV
CRYPT-NOSQL SLA FOR $n = 128$.

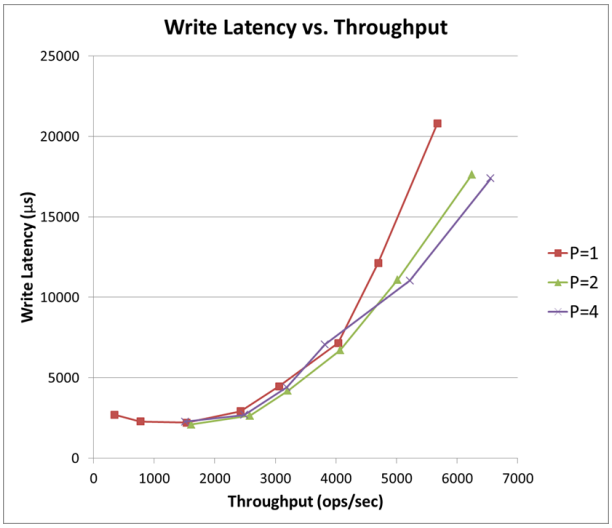
	Throughput (ops/sec)	Read/Write Latency (μs)
p=1	5673.0	24210.6/20803.9
p=2	6243.1	23284.6/17616.7
p=4	6553.3	21612.6/17377.6

VI. CONCLUSION

Data encryption on most NoSQL databases is usually enforced by user's own implementation in applications or



(a) Read operations



(b) Write operations

Fig. 6. Latency vs. Throughput with number of clients less than 128.

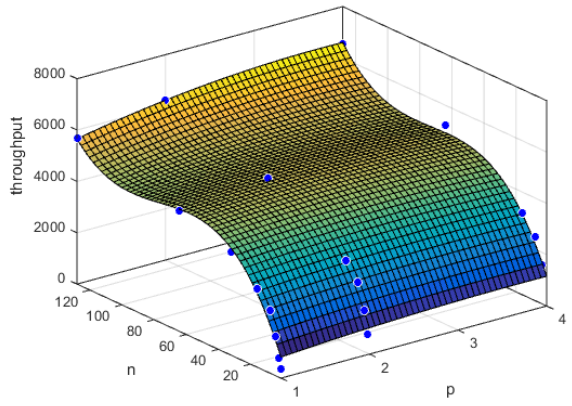


Fig. 7. The equation of throughput $T(p, n)$.

middlewares. This protects data integrity but also increases the difficulty of executing queries over data. We were motivated to design a system that supports query over encrypted data on NoSQL databases. To the best of our knowledge, none of previous research has designed such a system for NoSQL databases. Furthermore, performance of the proposed system with multiple clients accessing is also an important concern in the field of big data research.

In this study, we introduced the design and analysis of the proposed Crypt-NoSQL system that provides the functionality of query over encrypted data with high performance. Three different structures of Crypt-NoSQL were proposed and the performance of each structure was evaluated with YCSB. Our experimental results showed that with more clients accessing the system at the same time, Crypt-NoSQL was able to deliver a high performance by adding more proxy machines working in parallel. A guidance to establish SLA for Crypt-NoSQL as a cloud service was also discussed.

REFERENCES

[1] C. Mohan, "History repeats itself: sensible and nonsensql aspects of the nosql hoopla," in *Proceedings of the 16th International Conference on Extending Database Technology*, pp. 11–16, ACM, 2013.

[2] B. G. Tudorica and C. Bucur, "A comparison between several nosql databases with comments and notes," in *Roedunet International Conference (RoEduNet), 2011 10th*, pp. 1–5, IEEE, 2011.

[3] DB-Engines, "DB-Engines Ranking." http://db-engines.com/en/ranking, 2016. [Online].

[4] E. Brewer, "Cap twelve years later: How the" rules" have changed," *Computer*, vol. 45, no. 2, pp. 23–29, 2012.

[5] Cloud Security Alliance, "Expanded Top Ten Big Data Security and Privacy Challenges." https://downloads.cloudsecurityalliance.org/initiatives/bdwg/Expanded_Top_Ten_Big_Data_Security_and_Privacy_Challenges.pdf, 2013. [Online].

[6] R. A. Popa, C. Redfield, N. Zeldovich, and H. Balakrishnan, "Cryptdb: protecting confidentiality with encrypted query processing," in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pp. 85–100, ACM, 2011.

[7] 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*, pp. 143–154, ACM, 2010.

[8] L. Okman, N. Gal-Oz, Y. Gonen, E. Gudes, and J. Abramov, "Security issues in nosql databases," in *Trust, Security and Privacy in Computing and Communications (TrustCom), 2011 IEEE 10th International Conference on*, pp. 541–547, IEEE, 2011.

[9] J. Klein, I. Gorton, N. Ernst, P. Donohoe, K. Pham, and C. Matser, "Performance evaluation of nosql databases: A case study," in *Proceedings of the 1st Workshop on Performance Analysis of Big Data Systems*, pp. 5–10, ACM, 2015.

[10] T. Waage and L. Wiese, "Benchmarking encrypted data storage in hbase and cassandra with ycsb," in *Foundations and Practice of Security*, pp. 311–325, Springer, 2014.

[11] Datastax, "CQL for Cassandra 2.0 and 2.1." https://docs.datastax.com/en/cql/3.1/cql/ddl/ddl_when_use_index_c.html, 2016. [Online].

[12] Cassandra Wiki, "Cassandra Hardware Requirements." https://wiki.apache.org/cassandra/CassandraHardware, 2016. [Online].