Performance Evaluation of MySQL, Cassandra and HBase for Heavy Write Operation

Vishal Dilipbhai Jogi
Intelligent Services Division
Samsung R&D Institute India
Bangalore, India
vishal.jogi@samsung.com

Ashay Sinha
Department of Computer Science and Engineering
Indian School of Mines
Dhanbad, India
ashaysinha@gmail.com

***Abstract*— NoSQL databases have gained prominence in recent years because of their ability to scale easily and provide support for large amount of structured as well as unstructured data. User applications that need to perform heavy write operations have started making a paradigm shift to NoSQL databases with the overall intention to enhance performance. Eventually, various types of NoSQL databases have come up that have several features that suit different user specific needs. In this paper, the authors evaluate the performance of MySQL, Cassandra and HBase for heavy write operations by a web based REST application.**

Keywords— NoSQL; Distributed Systems; Cassandra; HBase; MySQL

I. INTRODUCTION

Until recently, the relational databases were, undoubtedly, the most popular kind of databases used to manage data in a practical sense. One of the many remarkable features of relational databases is their ability to perform transactional updates and handle the underlying consistency issues considerably well. The advent of big data and cloud computing has stressed upon the need for databases that can handle and process big data effectively. Relational databases, however, lack several functionalities that are keys to managing the big data. Relational databases do not scale up easily, and moreover, cannot handle unstructured data and data in unexpected formats effectively [1].

With the current emphasis on big data, NoSQL databases have surged in popularity. NoSQL databases surpass the shortages of the relational databases by providing a mechanism for storing and retrieving structured as well as unstructured data [2][3]. NoSQL databases, today, have become the preferred choice for big data processing that can be attributed to their high data access speeds, support for flexible schemas and distributed databases and their ability to scale up easily.

In this paper, we analyze different types of NoSQL databases and compare the throughput, in terms of Transactions per Second (TPS), for heavy write operations for certain specific NoSQL databases. The paper organization is as follows: Section 2 gives a brief overview of NoSQL databases; Section 3 enlists certain specific features of two NoSQL databases: Cassandra and HBase; Section 4 describes the experimental framework; Section 5 analyzes the performance of MySQL, Cassandra and HBase for heavy write operations; Section 6 concludes the paper.

II. NOSQL AND CAP THEOREM

The term NoSQL stands for 'Not only SQL'. NoSQL is not a replacement but rather an alternative for the traditional SQL. In many polyglot persistence applications, NoSQL databases are used in conjunction with relational databases. There is no sound and complete definition for NoSQL, but most NoSQL databases are believed to have certain common characteristics. NoSQL databases do not use the relational model, are schema-less and run well on clusters [4]. Most of these NoSQL databases are open source.

In 2006, *Google Inc.* published a paper on Big Table [4] which was followed by *Amazon.com*'s paper on the Dynamo [5] in 2007. Big Table brought in the notion of data being stored in column families rather than in traditional tabular formats. Column families were defined to be a collection of interrelated data that were often accessed together. Dynamo was conceptualized as a highly available key-value store that sacrificed database consistency in certain failure scenarios. This behavior is common to most NoSQL databases where the database gives up on one amongst Consistency (C), Availability (A) and Partition tolerance (P). This is known as CAP theorem. According to the CAP Theorem, a typical NoSQL database in a networked-shared environment can have only two out of three properties - consistency, availability and partition tolerance. However, consistency and availability can be optimized by explicitly handling partitions and consequently, some trade-off amongst all three properties can be achieved [7].

The NoSQL databases can, in general, be divided into four main categories, namely, Key-Value Databases, Column-Family Databases, Document-Oriented Databases and Graph Databases.

Key-Value Stores maintain a hash-table that is indexed by a key which in turn points to the specific set of values that the database wants to store. Except for the key, the database usually stores the information as a blob of data. Consequently, data can only be queried by the key. It is not of the concern of the database to know what is getting stored. It is the responsibility of the application program to extract data from the database in a way that would make the data comprehensible. Key-Value Databases are used in

978-1-4799-8579-1/16/$31.00 ©2016 IEEE

applications that need to maintain user specific session information, profile, preferences, etc. Key-Value Databases cannot be used in situations where the application needs to query by data.

The second category of the NoSQL databases, the Column Family Database, stores data in column families. The data is stored in the column family as rows that have many columns associated with a row key [4]. Although at a logical level the database may appear as storing data in tables as is the case with the relational databases but at the physical level the data is stored on a per-column family basis. Column family is a collection of columns and columns can be added to the column family at any point without pre-announcing them. Column Family Databases are specifically useful in applications that require heavy write operations like log aggregation. One should however, avoid using Column Family Databases for applications that are in early phases of development or have changing query patterns.

Document-Oriented Databases can be seen as a sub-category of Key-Value Store that store documents in the "value" part of the Key-Value Database. The key difference between these two kinds of databases lies in the fact that while in case of Key-Value Stores, the data is inherently opaque to the database, the Document-Oriented Database uses the internal structure of the database to extract metadata that the databases engine uses for further evaluation and analysis. Document-Oriented Databases have rich query language and are really useful in applications that need to migrate from a relational to NoSQL database. The Document-Oriented Databases cannot be used in cases where the application needs to perform cross-document atomic operations or execute queries against varying aggregate structures.

Graph Databases are used to represent entities and relationship between entities where entities are represented as nodes in the database and the relationships between the entities are represented as directed edges. Graph databases are gaining importance as they are now being deployed in organizations for managing data within applications like social networking [8]. Table I presents different type of NoSQL database with a few examples.

TABLE I. EXAMPLES OF DIFFERENT KINDS OF NOSQL DATABASES

Type of Database	Example
Key-Value Store	Azure Table Storage, Amazon DynamoDB, Redis, Riak, Berkely DB, etc.
Column-Family Database	HBase, Cassandra, Hypertable, Amazon SimpleDB, etc.
Document-Oriented Database	MongoDB, RavenDB, CouchDB, OrientDB, etc.
Graph Databases	Neo4J, Infinite Graph, etc.

III. CASSANDRA AND HBASE

In this section, we present certain characteristics of two NoSQL databases, Cassandra and HBase.

A. Cassandra

Cassandra is an open-source distributed database management system that can be described as Key-Value as well as a Column-Family database because it is eventually consistent like the Dynamo [6] and it stores data in column-families just like the Big Table [5]. Cassandra is designed to maintain huge amount of structured data and is available under the Apache license. Other properties of Cassandra include its ability to scale elastically as well as linearly. The performance of Cassandra increases as we increase the number of nodes in the cluster. Like the relational databases, Cassandra supports ACID (Atomicity, Consistency, Isolation, Durability) properties and has blazingly fast write speeds. Cassandra supports easy data distribution by replicating data across several datacenters. Cassandra is written in Java.

B. HBase

HBase is a Column-Family NoSQL database built up in Java on the lines similar to that of Google's Big Table [5]. It is built on top of the Hadoop Distributed File System (HDFS). Consequently, it provides Big Table like features for Hadoop and provides quick random access to huge amount of structured as well as unstructured data stored in the HDFS. While Hadoop can perform only batch processing, HBase has an added functionality of performing random searches by creating indexes on the columns of the column families. HBase, like Cassandra, is linearly scalable and is built up on the master-slave concept [9]. If multiple HMaster servers are used, then upon failure of the HMaster that balances the load amongst the region servers, another HMaster automatically assumes this responsibility. In such a way, HBase provides an automatic failure support. However, a single point of failure occurs in cases where only one HMaster server is used. HBase is used in scenarios where the application deals with random read/write operations to big data.

IV. EXPERIMENTAL SETUP

In our application, we compare the throughput in terms of Transactions per Second (TPS) for heavy write operations in a relational database MySQL and two NoSQL databases, namely, Cassandra and HBase.

The application is a web based REST (Representational State Transfer) application written in Java that takes in data from a webpage in the JSON format and puts it into the database via HTTP POST request. The application is hosted on the Tomcat server (version 7). The data in the JSON text is mapped to the corresponding POJO via *@JsonAutoDetect* and *@JsonProperty* annotations in Java. The data stored in the POJO is further inserted into the database.

The detailed database (database / keyspace / column family) organization of MySQL, Cassandra and HBase is presented below.

978-1-4799-8579-1/16/$31.00 ©2016 IEEE

A. MySQL Database Organization

The MySQL database maintains two tables, *Table_1* and *Table_2* in the database *mysql_db* that is hosted on the MySQL Community Server (version 5.6). *Table_1* has 14 columns and *Table_2* has 5 columns. The entries in these columns are of the type integer, variable character, Boolean and date. Some of the column fields are encrypted by the Secure Hash Algorithm (SHA and SHA-1) which is done by the application program itself. Fig. 1 shows the pictorial representation of the MySQL database used by the web application.

B. Cassandra Keyspace Organization

A keyspace in Cassandra is analogous to a database in MySQL except for one difference that it also takes into consideration how data is replicated across nodes in the cluster. We define a keyspace *cassandra_keyspace*, with Keyspace Replication Strategy set to simple and Keyspace Replication Factor set to 1. The keyspace maintains two tables (can also be referred to as column families) *Table_1* and *Table_2* just like *mysql_db*. These tables have 14 and 5 fields respectively. The web application is integrated with Apache Cassandra 2.1.8. The keyspace organization for Cassandra is shown in Fig. 2.

C. HBase Column Family Organization

Unlike Cassandra and MySQL, HBase doesn't have a concept of a keyspace or a database. It rather stores data in column families and adds a column family to the tables as and when required. For our application we create two column families, namely, *columnFamily_1* and *columnFamily_2*. *columnFamily_1* in HBase would store the exact data that is stored in *Table_1* in Cassandra and MySQL. Likewise, *columnFamily_2* would store the exact data that is stored in *Table_2*. We do not initialize the columns within the column families as they can be propagated without being pre-announced. Subsequently, we add the column families in a table and call it *HBase_Table*.

The web-application is integrated with HBase (version 0.98.8-hadoop2) and Hadoop (version 2.6.0) both of which run in pseudo-distributed mode. The HBase table organization is shown in Fig. 3.

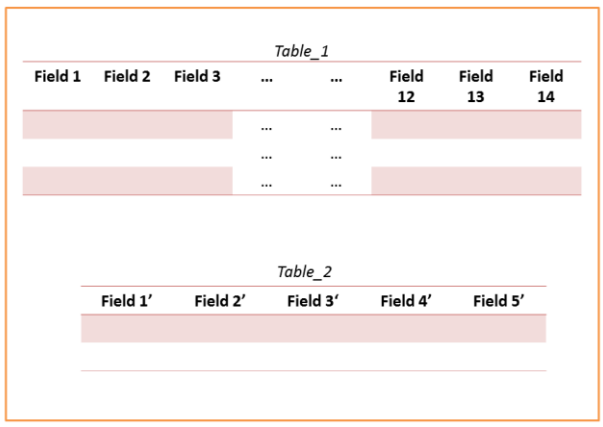

Fig. 1. Organization of the MySQL database *mysql_db*

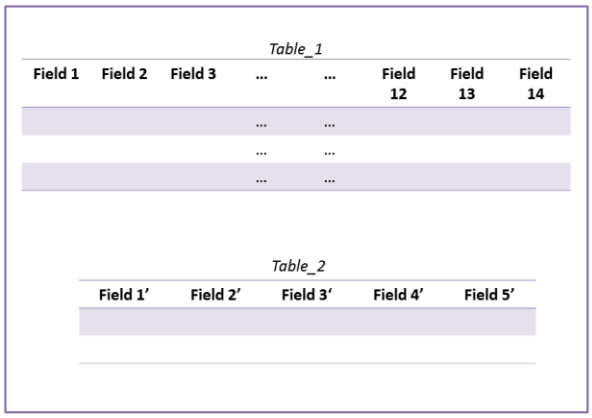

Fig. 2. Organization of the Cassandra keyspace *cassandra_keyspace*

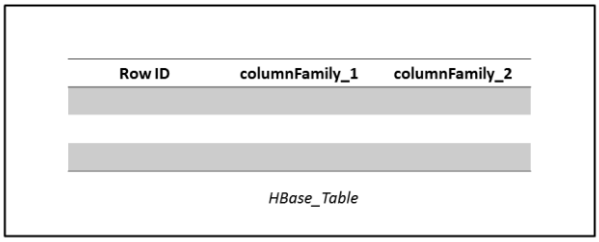

Fig. 3. *HBase_Table* Organization

V. Performance Evaluation for Heavy Write Operation

The web application was, one at a time, connected to the MySQL, Cassandra and HBase databases and the load testing was done using nGrinder (nGrinder is a web application that is used as a load testing tool. It calculates the load in terms of Transactions per Second (TPS)). nGrinder formulated the JSON text in such a manner that in every transaction, a new row was inserted in both tables in MySQL and Cassandra and in both column families in case of HBase. Thus, in each transaction we perform 2 write operations. The TPS graph as obtained by nGrinder after running the load test for 10 minutes for each database is depicted in Fig. 4, Fig. 5 and Fig. 6. It is worth mentioning that nearly same results were obtained after running the load test on each database for 30 minutes.

As is described in the below given graphs (every point on the graph represents the total number of transactions at any given time), TPS for heavy write operations for MySQL, Cassandra and HBase were obtained to be 71.9, 745.8 and 150.4 respectively. The peak TPS was found to be 114.5, 898 and 324.5 respectively for MySQL, Cassandra and HBase. Various experimental parameters and results are summarized in Table 2.

Number of processes, threads and agents as mentioned in Table II are nGrinder parameters and are kept same for all three databases to provide a uniform testing environment for precise results.

978-1-4799-8579-1/16/$31.00 ©2016 IEEE

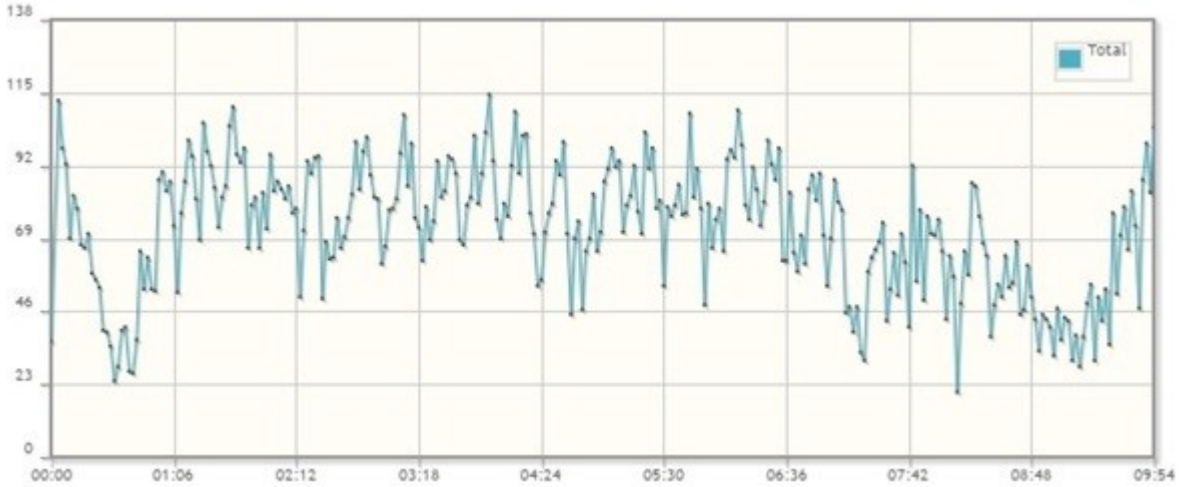

Fig. 4. nGrinder generated TPS graph for heavy write operations in *mysql_db* (X-axis: Time in Seconds, Y-axis: No of Transactions)

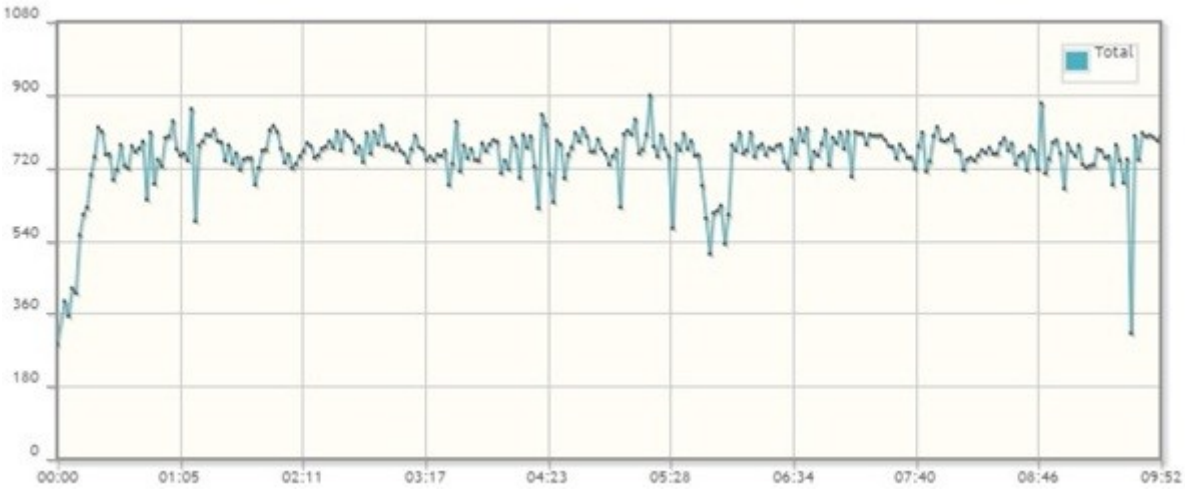

Fig. 5. nGrinder generated TPS graph for heavy write operations in *cassandra_keyspace* (X-axis: Time in Seconds, Y-axis: No of Transactions)

Fig. 6. nGrinder generated TPS graph for heavy write operations in *HBase_Table* (X-axis: Time in Seconds, Y-axis: No of Transactions)

978-1-4799-8579-1/16/$31.00 ©2016 IEEE

TABLE II. SUMMARY OF HEAVY WRITE OPERATION ON MYSQL, CASSANDRA AND HBASE

Experimental Parameters	MySQL	Cassandra	HBase
No. of Processes	3	3	3
No. of Threads	33	33	33
No. of Agents	1	1	1
TPS	71.9	745.8	150.4
Peak TPS	114.5	898	324.5
Mean Test Time (ms)	1155.93	131.66	615.25
Executed Tests	43674	442934	92475
Errors (no. of transactions)	0	0	384
Error Percentage (%)	0	0	0.41524

VI. CONCLUSION

Cassandra scaled up the most amongst the 3 databases and provided blazing fast write speeds. Fast writes into the database is a unique feature of Cassandra. HBase, on the other hand, provided write speeds that were nearly twice as fast as the traditional relation database, MySQL. The fact that Cassandra incorporates within itself, simultaneously, the properties of Amazon's Dynamo and Google's Big Table makes it really suitable for applications that perform heavy write operations and need to have low write latency.

ACKNOWLEDGEMENT

This research work was carried out by Ashay Sinha under the guidance of Vishal Dilipbhai Jogi at Samsung R&D Institute India, Bangalore (SRI-B). The authors would like to express their gratitude towards SRI-B for providing all necessary support for carrying out the research work.

REFERENCES

[1] April Reeve, Big Data and NoSQL: The Problem with Relational Databases, infocus.emc.com, September 2012.

[2] Yishan Li and Sathiamoorthy Manoharan, A performance comparison of SQL and NoSQL databases, *in proc. of* Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM), IEEE, 2013, pp 15-19.

[3] Qiaoying Lin, Libo Li, Zijing Li and Gansen Zhao, Schema Conversion Model of SQL Database to NoSQL, *in proc. of* Ninth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), IEEE, 2014, pp 355-362.

[4] Pramod Sadalage, NoSQL Databases: An Overview, thoughtworks.com, October 2014.

[5] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach Mike Burrows, Tushar Chandra, Andrew Fikes and Robert E. Gruber, Bigtable: A Distributed Storage System for Structured Data, *in proc. of* 7th Usenix Symposium on Operating System Design and Implementation, 2006, pp 15-28

[6] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall and Werner Vogels, Dynamo: Amazon's Highly Available Key-value Store, *in proc. of* 21st ACM SIGOPS symposium on Operating systems principles, ACM, 2007, pp. 205-220.

[7] Eric Brewer, CAP Twelve Years Later: How the "Rules" Have Changed, Computer Volume 4, Issue 2, IEEE, January 2012, pp. 23-29.

[8] A. Castelltort and A. Laurent, Representing history in graph-oriented NoSQL databases: A versioning system, *in proc. of* Eighth International Conference on Digital Information Management (ICDIM), IEEE, 2013, pp. 228-234

[9] Vijayalakshmi Bhupathirajul and Ravi Prasad Ravud, The dawn of Big Data – Hbase, *in proc. of* Conference on IT in Business, Industry and Government (CSIBIG), IEEE, 2014, pp. 1-4