BENCHMARKING CASSANDRA

Megha Shah[1], Poonam Pany[2], Priyanka Makhija[3]
[1,2,3]Dept. of Computer Engineering, Sinhgad Academy of Engineering, University of Pune ,India

Abstract— **With the increasing need for storage of unstructured data, the need of NoSql databases have increased. The most widely used NoSql database is the column based Cassandra. While there has been growth in the usage of Cassandra, evaluating its performance becomes important and crucial to applications using Cassandra on a large scale for storage. Further, they are being applied to a diverse range of applications that differ considerably from traditional serving workloads. The number of emerging cloud serving systems and the wide range of proposed applications, coupled with a lack of performance comparisons, makes it difficult to understand the tradeoffs between systems and the workloads for which they are suited. We aim to benchmark Cassandra , with the goal of facilitating performance comparisons between versions of Cassandra while using YCSB to generate different workloads. We define a core set of benchmarks and report results for Cassandra evaluating it against various performance parameters.**

Keywords—**NoSQL, Cassandra, YCSB(yahoo cloud service benchmark), workloads**

I. INTRODUCTION

Cassandra is a massively scalable open source NoSQL database. Cassandra is perfect for managing large amounts of structured, semi-structured, and unstructured data across multiple data centers and the cloud. Cassandra delivers linear scalability and performance across many commodity servers with no single point of failure, and provides a powerful dynamic data model designed for maximum flexibility and fast response times.. When comparing Cassandra to a relational database, the column family is similar to a table in that it is a container for columns and rows. However, a column family requires a major shift in thinking for those coming from the relational world.

In a relational database, you define tables, which have defined columns. The table defines the column names and their data types, and the client application then supplies rows conforming to that schema: each row contains the same fixed set of columns. In Cassandra, you define column families. Column families can (and should) define metadata about the columns, but the actual columns that make up a row are determined by the client application. Each row can have a different set of columns. There are two types of column families:

- Static column family (Typical Cassandra column family design) refer fig 1 and
- Dynamic column family (Use with a custom data type) refer fig 2

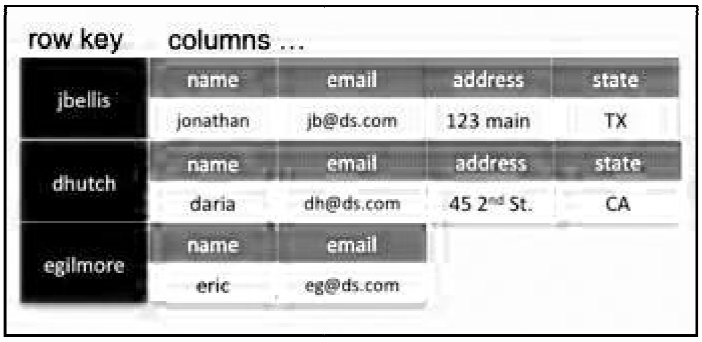

Figure 1 Static Column Family

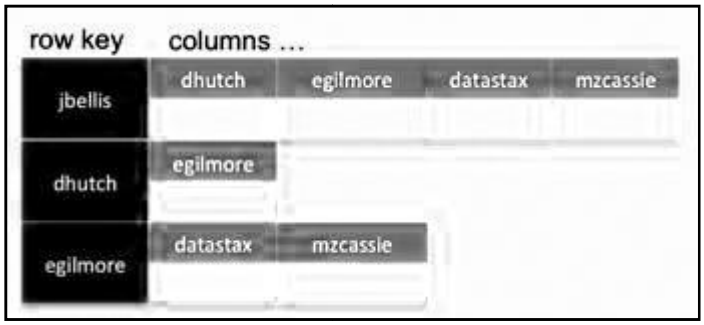

Figure 2 Dynamic Column Family

Column families consist of these kinds of columns:

- Standard: Has one primary key.
- Composite: Has more than one primary key, recommended for managing wide rows.
- Expiring: Gets deleted during compaction.
- Counter: Counts occurrences of an event.
- Super: Used to manage wide rows, inferior to using composite columns.

Although column families are very flexible, in practice a column family is not entirely schema-less. However, the data

models can be documented and compared qualitatively. Evaluating the performance of the system is the harder problem. Cassandra has made the decision to optimize for writes by using on-disk structures that can be maintained using sequential I/O . Furthermore, decisions about data partitioning and placement, replication, transactional consistency, and so on all have an impact on performance. Understanding the performance implications of these decisions for a given type of application is challenging. Developers of various systems report performance numbers for the typical workloads for their system, which may not match the workload of a target application. Exact comparison is hard, given numbers based on different workloads. Thus, developers often have to download and manually evaluate performance. This process is time-consuming and expensive. Our goal is to create a standard benchmark and evaluate Cassandra under different scenarios by creating various stress tests as workloads. These are standard workloads that cover interesting parts of the performance space (read-heavy workloads, write-heavy workloads, scan workloads, etc.). The workload generator makes it easy to define new workload types, and it is also straightforward to adapt the client to benchmark new data serving systems. In this paper, we describe Cassandra benchmark, and aim to report performance results. Although our focus in this paper is on performance and elasticity, the framework is intended to serve as a tool for evaluating other aspects such as availability and replication. [3]

II NOSQL

Interactive applications have changed dramatically over the last 15 years, and so have the data management needs of those applications. Today, three interrelated megatrends – Big Data, Big Users, and Cloud Computing – are driving the adoption of NoSQL technology. And NoSQL is increasingly considered a viable alternative to relational databases, especially as more organizations recognize that operating at scale is better achieved on clusters of standard, commodity servers, and a schema-less data model is often better for the variety and type of data captured and processed today.

A NoSQL non-relational database provides a mechanism for storage and retrieval of data that uses looser consistency models than traditional relational databases. This approach is a schema-free data model which includes simplicity of design, horizontal scaling and finer control over availability. NoSQL databases are often highly optimized key–value stores intended for simple retrieval and appending operations, with the goal being significant performance benefits in terms of latency and throughput. NoSQL is less structured than RDBMS and does not guarantee ACID.

NoSQL Databases follow the CAP theorem.

- Consistency: All database clients will read the same value for the same query, even given concurrent updates.
- Availability: All database clients will always be able to read and write data.
- Partition Tolerance: The database can be split into multiple machines; it can continue functioning in the face of network segmentation breaks.

III. CASSANDRA ARCHITECTURE

Cassandra architecture helps to understand some of its strengths and weaknesses from a distributed systems point of view. The Cassandra architecture consists of several Cassandra nodes together forming a Cassandra cluster. Figure shows the architecture of a Cassandra cluster. The first observation is that Cassandra is a distributed system. Cassandra consists of multiple nodes, and it distributes the data across those nodes (or shards them, in the database terminology). [1] Cassandra uses consistent hashing to assign data items to nodes. In simple terms, Cassandra uses a hash algorithm to calculate the hash for keys of each data item stored in Cassandra (for example, column name, row ID). The hash range or all possible hash values (also known as keyspace) is divided among the nodes in the Cassandra cluster. Then Cassandra assigns each data item to the node, and that node is responsible for storing and managing the data item. [4]. Each Cassandra server [node] is assigned a unique Token that determines what keys it is the first replica for. If you sort all nodes' Tokens, the Range of keys each is responsible for is (PreviousToken, MyToken], that is, from the previous token (exclusive) to the node's token (inclusive). The machine with the lowest Token gets both all keys less than that token, and all keys greater than the largest Token; this is called a "wrapping Range."

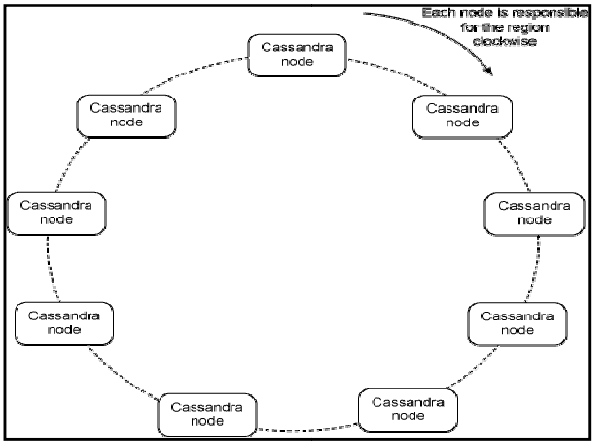

Figure 3 Cassandra Architecture

IV. BENCHMARKING TIERS

Tier 1—Performance

The Performance tier of the benchmark focuses on the latency of requests when the database is under load. Latency is very important in serving systems, since there is usually an impatient human waiting for a web page to load. However, there is an inherent tradeoff between latency and Throughput. As load increases, the latency of individual requests increases as well since there is more contention for disk, CPU, network, and so on. Typically application designers must decide on an acceptable latency, and provision enough servers to achieve the desired throughput while preserving acceptable latency. A system with better performance will achieve the desired latency and throughput with fewer servers. The Performance tier of the benchmark aims to characterize this tradeoff for Cassandra by measuring latency as we increase throughput, until the point at which the database system is saturated and throughput stops increasing. To conduct this benchmark tier, we need a workload generator which serves two purposes: first, to define the dataset and load it into the database; and second, to execute operations against the dataset while measuring performance. A set of parameter files defines the nature of the dataset and the operations (transactions) performed against the data.[14]

Tier 2—Scaling

A key aspect of cloud systems is their ability to scale elastically, so that they can handle more load as applications add features and grow in popularity. The Scaling tier of the database examines the impact on performance as more machines are added to the system. There are two metrics to measure in this tier:

Scale up—how does Cassandra perform as the number of machines increases? In this case, we load a given number of servers with data and run the workload. Then, we delete the data, add more servers, load a larger amount of data on the larger cluster, and run the workload again. If the database system has good scaleup properties, the performance (e.g., latency) should remain constant, as the number of servers, amount of data, and offered throughput scale proportionally.[15]

Elastic speedup—How does the database perform as the number of machines increases while the system is running? In this case, we load a given number of servers with data and run the workload. As the workload is running, we add one or more servers, and observe the impact on performance. A system that offers good elasticity should show a performance improvement when the new servers are added, with a short or non-existent period of disruption while the system is reconfiguring itself to use the new server.

V. BENCHMARK WORKLOADS

We hope to use a core set of workloads to evaluate different aspects of a system's performance. We can use a package which is a collection of related workloads. Each workload represents a particular mix of read/write operations, data sizes, request distributions, and so on, and can be used to evaluate systems at one particular point in the performance space.[18] A package, which includes multiple workloads, examines a broader slice of the performance space. Our goal was to examine a wide range of workload characteristics, in order to understand in which portions of the space of workloads systems performed well or poorly. For example, some systems may be highly optimized for reads but not for writes, or for inserts but not updates, or for scans but not for point lookups. The workloads in the core package can be chosen to explore these tradeoffs directly. The workloads in the core package are a variation of the same basic application type. In this application, there is a table of records, each with F fields. Each record is identified by a primary key, which is a string like "user234123". Each field is named field0, field1 and so on. The values of each field are a random string of ASCII characters of length L. For example, in the results reported in this paper, we construct 1,000 byte records by using F = 10 fields, each of

L = 100 bytes. Each operation against the data store is randomly chosen to be one of:

- Insert: Insert a new record.
- Update: Update a record by replacing the value of one field.
- Read: Read a record, either one randomly chosen field or all fields.
- Scan: Scan records in order, starting at a randomly chosen record key.

The number of records to scan is randomly chosen. For scan specifically, the distribution of scan lengths is chosen as part of the workload. Thus, the scan() method takes an initial key and the number of records to scan.

Workload	Operations	Record selection	Application example
A—Update heavy	Read: 50% Update: 50%	Zipfian	Session store recording recent actions in a user session
B—Read heavy	Read: 95% Update: 5%	Zipfian	Photo tagging; add a tag is an update, but most operations are to read tags
C—Read only	Read: 100%	Zipfian	User profile cache, where profiles are constructed elsewhere (e.g., Hadoop)
D—Read latest	Read: 95% Insert: 5%	Latest	User status updates; people want to read the latest statuses
E—Short ranges	Scan: 95% Insert: 5%	Zipfian/Uniform*	Threaded conversations, where each scan is for the posts in a given thread (assumed to be clustered by thread id)

Figure 4 Workloads

We use the predefined workloads in the core package of YCSB by assigning different distributions to the two main choices we must make: which operation to perform, and which record to read or write. The various combinations are shown in Table. Although we do not attempt to model complex applications precisely (as discussed above), we list a sample application that generally has the characteristics of the workload. [34]Loading the database is likely to take longer than any individual experiment, while we run each experiment (e.g., a particular workload at a particular target throughput against a particular database). All the core package workloads use the same dataset, so it is possible to load the database once and then run all the workloads. However, workloads A and B modify records, and D and E insert records. If database writes are likely to impact the operation of other workloads (e.g., by fragmenting the on-disk representation) it may be necessary to re-load the database. We do not plan to prescribe a particular database loading strategy in our benchmark, since different database systems have different loading mechanisms (including some that have no special bulk load facility at all).

VI. The Design

The nodetool utility in Cassandra allows collecting Cassandra performance statics. Also commands like TOP and SAR are useful to collect system statistics. As there is built in support of performance counters that provide information about how system is doing. Recoding information from this raw data is more for troubleshooting in development of the application.

The performances parameters that need to be evaluated are memory utilization, thread pool statistics, read and write statistics for column families (CF), read and write statistics for keyspaces. Hence we aim to write shell scripts to collect statistics repeatedly after some time interval and store it in a file (log file). We have to write a program which will raed this file and display the statistics graphically. The aim is to build an interactive UI that can test the Cassandra cluster in real time and show the results.

VII. CONCLUSION

We have presented the strategy of developing a tool for benchmarking Cassandra. This strategy will be used to achieve reports displaying the performance of Cassandra against various parameters. This will help application developers determine whether Cassandra is suitable for their application. It will also aim at achieving compaction and compression techniques on the database.

REFERENCES

[1]Prashant Malik, AvinashLakshman. "Cassandra - a decentralized structured storage system" The 3rd ACM SIGOPS International Workshop on Large Scale DistributedSystems and Middleware (LADIS 09), October 2009.

[2] Bingwei Wang, Si Peng, Xiaomeng Zhang, Mark Bownes, Rob Paton and FarshidGolkarihagh "Cassandra as used by Facebook" December 15,2010

[3]NabeelAhamedAkheel "Cassandra"

[4]DietrichFeatherston "Cassandra: Principles and Application"

[5]PrasannaBagade, Ashish Chandra, Aditya Dhende "Performance Monitoring Tool for NoSQL Column Oriented Distributed Database (Cassandra)"