BENCHMARKING CASSANDRA

Prof B.B.Gite[1], Megha Shah[2], Poonam Pany[3], Priyanka Makhija[4]
[1,2,3,4]Dept. of Computer Engineering, Sinhgad Academy of Engineering, University of Pune, Pune, India

Abstract—**Today, 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. The popularity of NoSQL databases (especially Cassandra) has been increasing day by day. Now, as many companies are developing Cassandra applications, they may need new tools to monitor database performance efficiently. Developers have difficulty optimizing something they can't see. When problems related to performance occur and proper analysis is needed, the statistical data generated by monitoring tool will be of a lot help. To optimize NoSQL applications, developers need to have an idea about how the database is behaving in different working scenarios. Cassandra is easy to configure, but for the proper performance tuning it is necessary to study the performance requirements for a particular application. This can be judged by monitoring tool. The paper describes the design of such monitoring tool and the results generated ie. statistics and graphs. The tool will be used primarily for low end machines as they are cost effective.**

Keywords: **Distributed databases, NoSQL databases, Database Performance, Parameters of performance.**

I. INTRODUCTION

Cassandra is NoSQL distributed database system which is known for managing large amount of distributed data. It provides high availability without single point of failure, the reason behind this is that it treats failure of node as norm rather than exception. It is also famous for high write throughput without harming read efficiency. As it is distributed database it replicates data to keep search latency small. Every keyspace when created, it is assigned a replication factor. Cassandra provides replication polices namely rack aware, rack unaware and data center aware [1].The data model of Cassandra is column oriented, columns together form column family. Column family is nothing but collection of columns associated with the key. Column has a name, value and timestamp. Different rows in the same column family may not have same number/type of columns. Super- column family is like column family within a column family. Every super column family is the collection of similar/related columns [3]. Cassandra dynamically partitions the data across the cluster and it provides different partitioning methods like random partitioner and order preserving partitioner. Cassandra is much easier to configure compared to other distributed database. It also allows fine performance tuning as per changing requirements. Mainly Cassandra system can be contains three layers - core layer, middle layerand top layer. The top layer is allows efficient, consistent reads and writes using a simple API. Cassandra provides simple queries insert, get & delete. The Cassandra API is made up of simple getter and setter methods and has no reference to the database distributed nature. Hinted hand-off is also the part of top layer. This occurs when a node goes down - the successor node becomes a coordinator and temporarily receives and stores write activities (hint) for downed node. When downed node becomes live, this information is given(handed) by coordinator node to live node. The middle layer contains functions for handling the data that is being written into the database. Compaction is the process which tries to combine keys and columns to increase the performance of the system by freeing the memory. The different ways of storing data such as Memtable and SSTable are also handled here[3]. The core layer deals with the distributed nature of the database, and contains functions for communication between nodes, the state of the cluster as a whole (including failure detection) and replication between nodes

Core	Middle	Top
Messaging service Failure detection Cluster state Partitioner Replication	Indexes Compaction Commit log Memtable SSTable	Hinted handoff Read repair Monitoring Admin tools

Table 1:- Cassandra Layers

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)

Table 2:- YCSB Workloads

II. ROLE OF YCSB

We have tested the core set of YCSB predefined workloads to evaluate different aspects of a system's performance. We 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.

While the core package examines several interesting performance axes, 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.

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. 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.

III. DESIGN

The nodetool utility in Cassandra allows to collect Cassandra performance statistics. Using this functionality we can extract the performance data onto a file called as the LOG file. The goal is to highlight the performance of the 2 Cassandra versions 1.2.13 and 2.0.4 against each other i.e. benchmarking. Also commands like TOP, SAR are useful to collect statistics. As there is built in support of performance counters that provides information about how system is doing. Recoding the information from these
Counters is very much necessary for troubleshooting in the development phase of application. The main performance parameters we should consider here are Throughput, Runtime, Average latencies for 1000 and 100000 numbers of records for both Cassandra versions against 5 core YCSB workloads.

So here we have to write total five shell scripts to collect the statistics repeatedly after some interval of time and store it in the file. We have to write a program which will read the files and display statistics graphically.

IV. GUI

The GUI has been designed in JSP in order to display the output of the tool in the form of graphs for better understanding of the benchmarking results. The GUI is easy

to use and user friendly and it shows various scenarios, namely:
1000ClusterRecords
1000ClusterRecords (Avg Latency/Records) single node and cluster
1000ClusterRecords (Throughput and Runtime)
1000ClusterRecords (Avg Latency) Parameterized workload single node and cluster
1000Records (1.2.13 vs 2.0.4) Throughput and Runtime
1000Records (1.2.13 vs 2.0.4) Avg Latency/record
1000Records (1.2.13 vs 2.0.4) Parameterized workload Throughput and Runtime
1000Records (1.2.13 vs 2.0.4) Parameterized workload Avg Latency/record

Similarly for 100000 records

V. RESULTS

Some of the screenshots of the GUI showcasing the benchmarking results are as below:

Figure 1:-1000ClusterRecords (Throughput and Runtime)

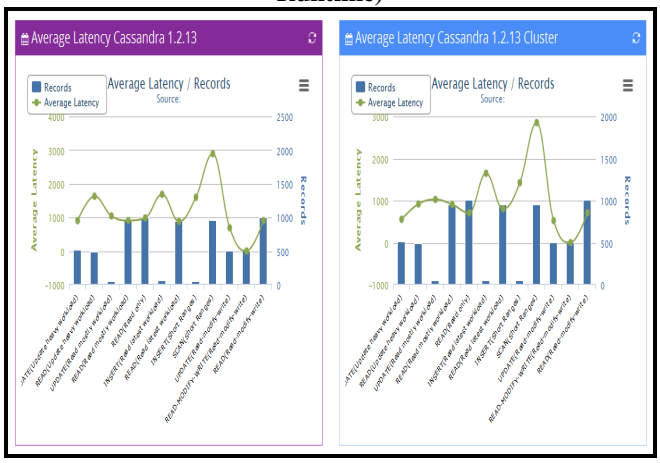

Figure 2:-1000ClusterRecords (Avg Latency/Records) single node and cluster

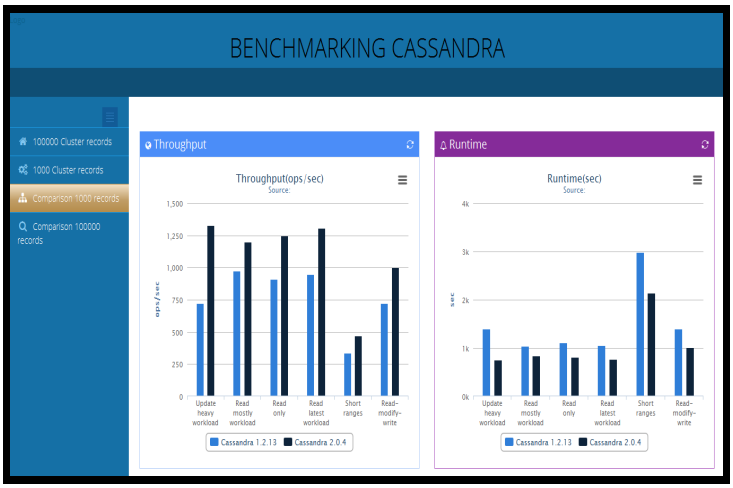

Figure 3:-1000Records (1.2.13 vs 2.0.4) Throughput and Runtime

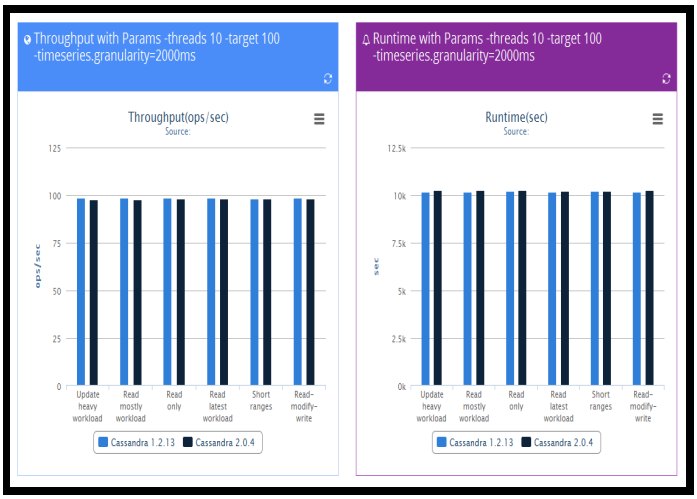

Figure 4:-1000Records (1.2.13 vs 2.0.4) Parameterized workload Throughput and Runtime

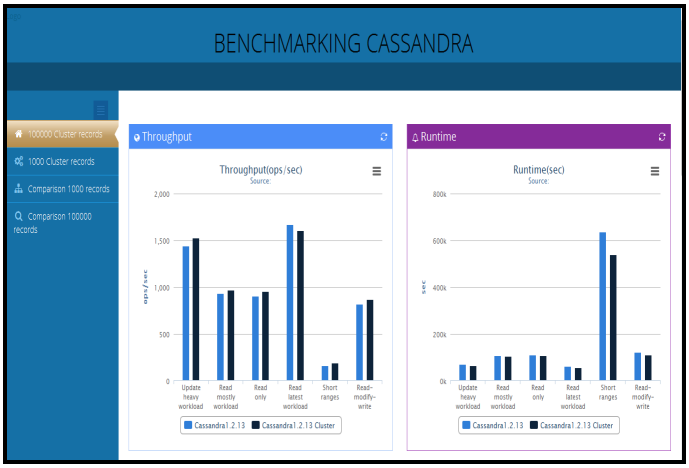

Figure 5:-100000ClusterRecords (Avg Latency/Records) single node and cluster

Figure 6:-100000Records (1.2.13 vs 2.0.4) Throughput and Runtime

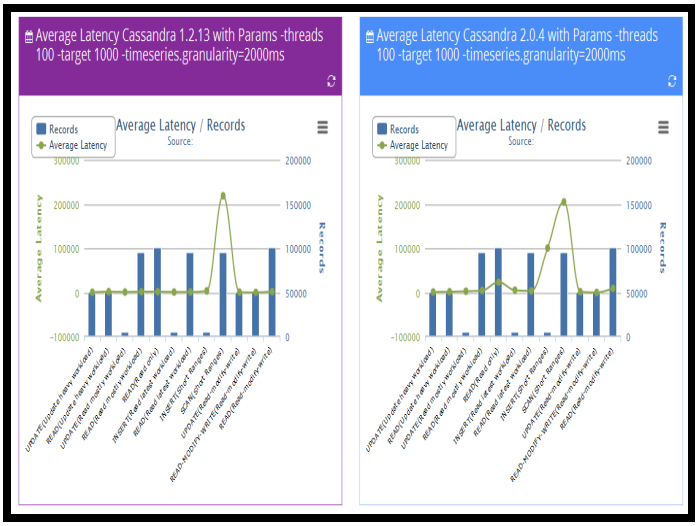

Figure 7:-100000Records (1.2.13 vs 2.0.4) Parameterized workload Avg Latency/record

VI. LIMITATIONS

The tool developed is limited to two Cassandra versions (1.2.13 and the latest 2.0.4) and can perform benchmarking for the same. The system may fail in following scenarios

1) Network failure: The network failure stands for breakage or saturation of link between two nodes and if the node goes down due to power failure.

2) System Crash: The System may crash due to excessive load on a specific machine.

VII. CONCLUSION

We have developed a tool for benchmarking Cassandra(v 1.2.13 vs 2.0.4) under 5 core workloads and parameterized workloads on a single node machine as well as on a Cassandra cluster with the desired replication factor. We have developed a user friendly and easy to understand GUI for showcasing these comparisons in the forms of various graphs. The tool showcases the Cassandra performance under various performance parameters such as throughput, latency, runtime. This will help application developers determine whether Cassandra is suitable for their application.

VIII. REFERENCES

[1] Amazon SimpleDB. http://aws.amazon.com/simpledb/.
[2]ApacheCassandra.http://incubator.apache.org/cassandra
[3] Apache CouchDB. http://couchdb.apache.org/.
[4] Apache HBase. http://hadoop.apache.org/hbase/.
[5]Dynomite Framework. ttp://wiki.github.com/cliffmoon/-dynomite/dynomite-framework.
[6] Google App Engine. http://appengine.google.com.
[7] Hypertable. http://www.hypertable.org/.
[8] mongodb. http://www.mongodb.org/.
[9] Project Voldemort. http://project-voldemort.com/.
[10] Solaris FileBench. http://www.solarisinternals.com/wiki/index.php/FileBench.
[11] SQL Data Services/Azure Services Platform. http://www.microsoft.com/azure/data.mspx.
[12] Storage Performance Council. http://www.storageperformance.org/home.
[13]Yahoo!QueryLanguage.http://developer.yahoo.com/yql
[14] A. Arasu et al. Linear Road: a stream data management benchmark. In VLDB, 2004.
[15] F. C. Botelho, D. Belazzougui, and M. Dietzfelbinger. Compress, hash and displace. In Proc. of the 17th European Symposium on Algorithms, 2009.
[16] F. Chang et al. Bigtable: A distributed storage system for structured data. In OSDI, 2006.
[17] B. F. Cooper et al. PNUTS: Yahoo!'s hosted data serving platform. In VLDB, 2008.
[18] G. DeCandia et al. Dynamo: Amazon's highly available key-value store. In SOSP, 2007.
[19] D. J. DeWitt. The Wisconsin Benchmark: Past, present and future. In J. Gray, editor, The Benchmark Handbook. Morgan Kaufmann, 1993.
[20] I. Eure. Looking to the future with Cassandra. http://blog.digg.com/?p=966.
[21] S. Gilbert and N. Lynch. Brewer's conjecture and the Feasibility of consistent, available, partition-tolerant web services. ACM SIGACT News, 33(2):51–59, 2002.
[22] J. Gray, editor. The Benchmark Handbook For Database and Transaction Processing Systems. Morgan Kaufmann, 1993.
[23] J. Gray et al. Quickly generating billion-record synthetic databases. In SIGMOD, 1994.

[24] A. Lakshman, P. Malik, and K. Ranganathan. Cassandra: A structured storage system on a P2P network. In SIGMOD, 2008.
[25] B. C. Ooi and S. Parthasarathy. Special issue on data Management on cloud computing platforms. IEEE Data Engineering Bulletin, vol. 32, 2009.
[26] A. Pavlo et al. A comparison of approaches to large-scale data analysis. In SIGMOD, 2009.
[27] R. Rawson. HBase intro. In NoSQL Oakland, 2009.
[28] A. Schmidt et al. Xmark: A benchmark for XML data Management. In VLDB, 2002.
[29] R. Sears, M. Callaghan, and E. Brewer. Rose: Compressed, log-structured replication. In VLDB, 2008.
[30] M. Seltzer, D. Krinsky, K. A. Smith, and X. Zhang. The case for application-specific benchmarking. In Proc. HotOS, 1999.
[31] P. Shivam et al. Cutting corners: Workbench automation for server benchmarking. In Proc. USENIX Annual Technical Conference, 2008.
[32] M. Stonebraker et al. C-store: a column-oriented DBMS. In VLDB, 2005.
[33] B. White et al. An integrated experimental environment for distributed systems and networks. In OSDI, 2002.
[34] K. Yocum et al. Scalability and accuracy in a large-scale network emulator. In OSDI, 2002.