STUDY OF CONSISTENCY AND PERFORMANCE TRADE-OFF IN CASSANDRA

Kena Vyas and PM Jat

DAIICT, Gandhinagar, Gujarat, India

ABSTRACT

Cassandra is a distributed database with great scalability and performance that can manage massive amounts of data that is not structured. The experiments performed as a part of this paper analyses the Cassandra database by investigating the trade-off between data consistency andperformance. The primary objective is to track the performance for different consistency settings. The setup includes a replicated cluster deployed using VMWare. The paper shows how difference consistency settings affect Cassandra's performance under varying workloads. The results measure values for latency and throughput. Based on the results, regression formula for consistency setting is identified such that delays are minimized, performance is maximized and strong data consistency is guaranteed. One of our primary results is that by coordinating consistency settings for both read and write requests, it is possible to minimize Cassandra delays while still ensuring high data consistency.

KEYWORDS

NoSQL, Cassandra, Consistency, Latency, YCSB, and Performance.

1. INTRODUCTION

Data's relevance has skyrocketed to the point where it is now seen as a precious asset. For any organisation, data is an essential. Every day, massive amounts of data get generated. Data of various formats are seen nowadays in IoT devices such as smartwatches, smart TVs, and home assistants. Every second or minute, data of different kinds gets generated from different devices. As a result, the ability to properly store and retrieve such huge and diverse data is required.

Relational databases have typically been used to store structured data with a high level of consistency. But when it comes to working with unstructured data, they have a number of drawbacks. The rigorous schema constraints of relational databases make it challenging to store massive data, which is typically anticipated to be unstructured or loosely structured. Field lengths are limited in relational databases, which leads to improper handling of unstructured data. Because of the inadequacies of relational databases when it comes to massive data, NoSQL databases have grown in popularity.

NoSQL Databases are non-relational data management systems. It gives a way to save and retrieve data. The data is represented differently than in relational databases, where tabulated relations are used. It does not require a fixed schema. The key advantage of using a NoSQL database is for huge data with dispersed data repositories. Therefore, it's becoming more prevalent in big data and real-time online applications. NoSQL databases have the following features: Flexible schemas, High availability, and Horizontal scaling. NoSQL databases has eventual consistency and hence lacks ACID features.

David C. Wyld et al. (Eds): CST, NLMLT, DMS, CLBD, ITCS, VLSIE - 2022
pp. 61-77, 2022. CS & IT - CSCP 2022 DOI: 10.5121/csit.2022.121907

1.1. Motivation

The main motivation of this paper is to find optimal setting for Cassandra database such that it provides strong consistency and minimal latency. Understanding this trade-off is crucial for finding a database state that is consistent. The paper examines the trade-offs that NoSQL databases must make between consistency, availability, and latency. It's crucial to understand how different consistency settings affect system latency. There are many NoSQL databases available for use. various industry trends suggest that Apache Cassandra is one of the top three in use today together with MongoDB and HBase [1]. Apache Cassandra is a columnar distributed database that takes database application development forward from the point at which we encounter the limitations of traditional RDBMSs in terms of performance and scalability [2]. Cassandra is a NoSQL distributed database system that is known for managing large amounts of distributed data. It provides high availability without a single point of failure [3].

1.2. Objective

In this paper, the Cassandra database is used to provide a quantitative examination of the fundamental Big Data trade-offs between data consistency and performance. We'd like to provide practical recommendations to developers that use Cassandra as a distributed data storage system, allowing them to forecast Cassandra latency while keeping the required consistency level in mind, and to optimise the consistency settings of operations. A benchmarking approach is developed that optimizes Cassandra's performance that guarantees strong data consistency under the selected workload. A NoSQL database like Cassandra supports database replication in order to maintain availability in the case of event failure or planned maintenance events. Cassandra keeps replicas on several nodes to ensure automatic failover and durability. Depending on the replication mechanism employed, a consistency setting needs be found that maximises performance while minimising latency.

1.3. Outcomes

A benchmarking methodology is created for working with read and write workloads in different proportions. Various workload runs are executed on the deployed cluster and their results are measured. The Cassandra database is monitored for Latency and Throughput values when read and write workloads are executed on it for a varying number of threads. Various combinations of read and write workloads are considered. The outcome of this paper will help the user of the database in identifying a consistency setting that is strong and simultaneously provides sufficient throughput with minimized latency. Two experiments are performed as a part of this work that measured the performance of the Cassandra database for varying read/write workloads, changing threads, and different consistency settings. The first experiment measures the results by separating the read and write workloads. In the second experiment, various proportions of read/write workloads are considered together so that we can get all possible combinations and can measure the results accordingly. From the measured results, regression formulas are generated which can be used for prediction purposes.

1.4. Paper Organization

This paper is organised as follows. In the next section i.e., Section 2 is Cassandra and Consistency where concepts like NoSQL, replication factor and consistency levels are covered, Section 3 talks about Performance Benchmarking with YCSB along with related works, Section 4 is about experimentation, the two experiments performed as a part of this paper are explained in

detail along with their objective, setup, and results. Section 5 concludes the paper with a conclusion.

2. CASSANDRA AND CONSISTENCY

Two Facebook developers, Lakshman and Malik, released Cassandra to the Apache community in 2008. They describe Cassandra as a "distributed storage system for managing very large amounts of structured data spread across many commodity servers while providing highly available service with no single point of failure"[4]. Cassandra is a column-oriented, peer-to-peer NoSQL database that is a distributed and decentralized storage system that is open source. It oversees massive amounts of structured/unstructured/loosely structured data from all around the world. It ensures high availability, which eliminates the possibility of a system failure and provides eventual consistency [5].

Cassandra provides a familiar interface known as Cassandra Query Language (CQL). CQL offers an abstraction layer to the database where implementation specifics are hidden, and native access syntaxes are provided. The data in Cassandra is kept in keyspaces, which are similar to databases in relational database concepts. A column family in the Cassandra database is equivalent to a table in a relational database, and they can be represented as a collection of rows. Rows are formed of columns and their values, which are represented as key-value pairs [6]. The Replication Factor and Strategy can be defined at the time of keyspace creation.

2.1. Cassandra Data Model

The Wikipedia page of Cassandra mentions that the Cassandra data model is “designed for distributed data on a very large scale” [7]. Cassandra runs in main memory and makes asynchronous disc writes on a regular basis. Cassandra comprises ACID properties in order to increase availability and performance. The structure of the Cassandra model is quite different from the relational model.

A Cassandra cluster is a storage unit in the database. It consists of multiple keyspaces. A level of Column families exists beneath the keyspace level. A column family is a logically arranged collection of one or more columns depending on database design. There will be one or more column(s) inside a column family. Within the Cassandra data paradigm, a column is the simplest data structure and is at the lowest level. A column has 3 different attributes namely name, value, and timestamp. The name attribute is used to identify a column. Value attribute stores the actual value related to the name attribute and timestamp is the time when the column is stored, it is mainly used during data replication.

A "row" is similar to a relational database row which is a collection of values linked together. However, there is a difference between the two. The row in the Cassandra model is dynamic and can have a varying number of columns. One of the advantages of Cassandra is the flexibility ofwhat may be stored and the fact that no space is allocated for columns that are not part of the current data set.

2.2. NoSQL

NoSQL is often referred to as "non-SQL" or "non-relational". Eben Hewitt has his own explanation of what NoSQL is all about in his book Cassandra: The Definite Guide [8]. "Comparing NoSQL to relational is basically a shell game," Hewitt argues. Eben Hewitt implies that NoSQL cannot be directly compared to a relational database because it encompasses a wide

range of non-relational database types. Most NoSQL databases provide some level of balance among consistency, availability, partition tolerance, and latency. Although a few databases have made ACID (Atomicity, Consistency, Isolation, Durability) transactions core to their architecture, most NoSQL stores lack these [9].

NoSQL systems can be classified into categories according to their data model. There are four different types of NoSQL databases: Column-oriented, Graph, Document, and Key-value databases. Cassandra, MongoDB, Couchbase, HBase, and Redis are some of the most popular NoSQL databases. Cassandra offers a range of unique features which makes it a good choice for us. Cassandra has no single point of failure because of its peer-to-peer architecture. Scalability is another advantage that Cassandra provides for scaling up or down. It is highly available and fault tolerant because of the data replication it provides. Such benefits provided by Cassandra makes it a great choice.

2.3. Replication and Consistency

Data replication is the process of storing several copies of data in multiple nodes. The replication approach ensures that the same data is available in other nodes if one node fails. Cassandra supports replication in the database to ensure availability in the event of failure or other predefined activity. The process of replicating data from one location to another is known as replication. The replication method for each keyspace determines the nodes where replicas are placed. Cassandra keeps replicas on several nodes to ensure fault tolerance and reliability. The replication factor refers to the total number of replicas in the cluster. A replication factor of one means that each row in the Cassandra cluster has only one copy. At the time of keyspace generation, the Replication Factor can be specified. The replication factor should not be more than the total cluster nodes.

The minimal number of Cassandra nodes that must recognize a read or write operation before it may be declared successful is known as the Cassandra consistency level. Different Edge keyspaces can have different consistency levels allocated to them. When the consistency option is one, it indicates that for a read/write operation to succeed, at least one of the Cassandra nodes in the datacentre must react. Depending on the replication mechanism employed, a consistency setting can be found that maximizes performance while minimizing latency. Cassandra's consistency settings can be set to balance data accuracy and availability. Consistency can be set for a session or for each read or write operation individually.

Hewitt explains three different levels of consistency in his book about Cassandra [8].

Strong Consistency - All data received from the database must be the most current information available. A mechanism for a global timer will be necessary to put a time stamp on the data and actions done to the system. String consistency is essential in areas like financial institutions, e-commerce websites, etc at all times. Strict consistency ensures that the data returned will be consistent and valid. However, one disadvantage is that performance will be degraded because the system will have to verify data with multiple nodes before returning the results.

Most NoSQL systems use the concept of R, W, N where R is the number of nodes from which data is read, W is the number of nodes where data is written and N is the replication factor and when we have R+W>N then, strong consistency can be achieved.

Eventual Consistency - Context here is we have partitioned and replicated data. Any update to such a database needs to be propagated to all replicas. Any read request for a data item following its write should get the last updated value irrespective of a replica from which value is being read.

Eventual consistency is weaker than strong consistency. Whenever eventual consistency is used and a request for data is made, then it may provide data which is one version older than the current one. However, eventual consistency makes sure that the most recent data is available to the user after a certain period of time.

When we make a change to a distributed database, eventual consistency ensures that the change is mirrored across all nodes that store the data, ensuring that we get the same response every time query is made. Eventual consistency offers low latency. Because changes take time to reach replicas throughout a database cluster, early results of eventual consistency data queries may not have the most current updates. The database system guarantees that if no new updates are made to the object, eventually all accesses will return the last updated value [11].

Weak Consistency - Another type of consistency is weak consistency which gives no guarantee that all nodes will have same data at any given time. From time to time, updates are exchanged among nodes such that all nodes have updated data. After a certain period of time, the data in the nodes will reach a consistent state.

2.3.1. Consistency Level (CL) on Write

The number of replica nodes that must acknowledge before the coordinator can report back to the client is determined by the consistency level for write operations. The number of nodes that acknowledge (for a given consistency level) and the number of nodes that store replicas (for a certain replication factor) are almost always different. For e.g., even when only one replica node recognizes a successful write operation with consistency level ONE and RF = 3, Cassandra concurrently replicates the data to two other nodes in the background. Below are write consistency levels that are used in the paper:

Table 1. Consistency levels for Write operation

Level	Description
ONE	It only requires one replica node to recognise it. Because only one copy needs to acknowledge the write operation, it is faster.
QUORUM	It requires 51 percent or a majority of replica nodes across all datacentres to acknowledge it.
ALL	It requires confirmation from all replica nodes. Because all replica nodes must acknowledge the write operation, it is the slowest. Furthermore, if one of the replica nodes fails during the write operation, the write operation will fail, and availability will degrade. As a result, it's advisable not to use this option in production deployment.

2.3.2. Consistency level (CL) on Read

The consistency level for read operations determines how many replica nodes must respond with the most recent consistent data before the coordinator can deliver the data back to the client successfully. Below are read consistency levels that are used in the paper:

Table 2. Consistency levels for Read operation

Level	Description
ONE	Only one replica node returns the data at consistency level ONE. In this scenario, data retrieval is the quickest.
QUORUM	It signifies that 51 percent of replica nodes in all datacentres have responded. The data is then returned to the client via the coordinator.
ALL	It requires confirmation from all replica nodes. The read operation is the slowest in this situation since all replica nodes must acknowledge.

Quorum Calculation - The QUORUM level works with the number of quorum nodes. The following is how a quorum is computed and then rounded down to a whole number:

quorum = floor((sum_of_replication_factors / 2) + 1)

In a cluster of 3 nodes, a quorum is 2 nodes. In a cluster of 6 nodes, a quorum is 4 nodes.

There are mainly 2 ways for setting consistency in a cluster:

1st Way

To set the consistency level for all queries in the current cqlsh session, use CONSISTENCY in cqlsh.

Syntax:

CONSISTENCY [Level]

Example: CONSISTENCY ONE

2nd Way

For setting the consistency level individually for each operation, the consistency can be set in the command line argument (CLI).

-p cassandra.readconsistencylevel=[Level] -p cassandra.writeconsistencylevel=[Level]

Example:-p cassandra.readconsistencylevel=[ONE] -p cassandra.writeconsistencylevel=[ONE]

2.4. CAP Theorem

Being ACID compliance is one of the strengths of relational databases. However, it is hard to achieve serializability in distributed and replicated environment and may leads to delays that are beyond acceptable limits. NoSQL systems have compromised ACID properties in order to achieve better performance when working with large data sets. Because of that, NoSQL systems need to follow some other set of rules that fit the NoSQL criteria. A scientist called Eric Brewer established a theorem called Brewer's CAP theorem. Brewer et.al. [6] realizes this and presents CAP theorem which states that any distributed data store can only provide two of the three (i.e., consistency, availability and partition tolerance) guarantees.

Brewer's CAP theorem categorizes database systems according to their capabilities. The CAP theorem was created to put the different NoSQL solutions together because the bulk of them was obliged to compromise the ACID guarantee in order to focus on more critical aspects for their specific needs. CAP is an acronym that stands for [13]:

- Consistency - At the same moment, all connected nodes see the same data.
- Availability - Even if a request is unsuccessful, it is guaranteed that a response will be received if it is delivered to the database.
- Partition tolerance - There is no single point of failure in the system. If one node fails, the data can still be accessed by another node, and the system will continue to function normally.

Hewitt states in his book about Cassandra that "Brewer's theorem is that in any given system, you can strongly support only two of the three" [8]. The definition says that a database system cannot provide all three properties at the same time. When a system is spread across numerous nodes, it cannot be 100% consistent and available at any given time. When the state of a database is changed (new data added or data updated) due to various reasons it will take a few milliseconds or seconds to propagate the changes to other nodes because of which the system is called eventually consistent.

3. PERFORMANCE BENCHMARKING WITH YCSB

YCSB is an abbreviation for Yahoo cloud serving benchmark. YCSB is a program suite for computing the execution of NoSQL systems. It is used to evaluate/compare the working of different NoSQL systems based on several parameters. YCSB Benchmark is a collection of workloads. It can collect the performance metrics of a system under a specific, pre-defined workload. It makes it easier to compare the performance of the next generation of data serving systems [8]. The YCSB framework is a standard benchmark for evaluating the operation of NoSQL databases such as Redis, MongoDB, HBase, Cassandra, and others. The YCSB framework is made up of a client that generates a workload and a set of basic predefined workloads that cover various aspects of performance. YCSB provides five different workloads. Each workload is a unique combination of read/write queries and data sizes. The operations in the workload are Insert, Update, Read and Scan. The vital feature of the YCSB framework is its extensibility. The workload generating client is extensible which supports the benchmarking of different databases. The workloads are [8]:

Table 3. YCSB default workloads

Workload	Read Weightage	Update Weightage	Insert Weightage	Scan Weightage
A-Update Heavy	50%	50%	0%	0%
B-Read Mostly	95%	5%	0%	0%
C-Read Only	100%	0%	0%	0%
D-Read Latest	95%	0%	5%	0%
E-Short Ranges	0%	0%	5%	95%

3.1. Related Works

Relational databases have been the choice for majority of systems due to their rich set of features. However, they are not suitable for handling huge data. NoSQL databases have gained popularity as they efficiently work with big data [13]. The paper "NoSQL Databases: MongoDB vs Cassandra" talks mainly about NoSQL databases along with their types and also briefs about CAP/ACID theorems. YCSB benchmark is used for the experimentation. The performance parameter which signifies the execution time is taken into consideration for comparing the two databases i.e., MongoDB and Cassandra. In the experiments, six different YCSB workloads are used for testing both the databases. The results indicate that as the data size increased, MongoDB started to reduce performance [13]. However, Cassandra became faster as data size increased.

Yahoo cloud serving benchmark framework is presented in the paper titled "Benchmarking Cloud Serving Systems with YCSB", that facilitates performance comparisons of data serving systems. Four widely used databases like Cassandra, HBase, Yahoo!'s PNUTS, and a simple sharded MySQL implementation are used in the paper for benchmarking. The papers use core workload of YCSB for measuring performance and scalability of the databases. The results show that Cassandra and HBase have higher read latency on a read heavy workload and lower update latency on write heavy workload [14]. Along with that, Cassandra and PNUTS showed better scalability. The paper also explains in details the core workloads provided by YCSB. The paper also talks about the workload generating client that comes with YCSB using which new workloads can be defined.

Our paper focuses on the consistency and latency trade-off aspect mainly. To identify the best setting of threads and read/write workloads such that strong consistency can be obtained. The paper "Consistency Trade-offs in Modern Distributed Database System Design" explains in detail the consistency/latency trade-off. The paper gives a good introduction about CAP theorem. According to CAP, the system must choose between high availability and consistency [10].

The paper "Interplaying Cassandra NoSQL Consistency and Performance: A Benchmarking Approach" puts light on the trade-off between data consistency and performance. The main aim of the paper is to allow the developers to predict the delay in Cassandra by considering the required consistency level. The paper proposes a benchmarking approach for optimising performance of Cassandra such that strong consistency is ensured [12]. In the paper, a Cassandra database is deployed and executed in a real production environment. YCSB benchmark is modified to execute application specific queries. The Cassandra database is benchmarked for various conditions such as different workloads, different consistency settings, etc. After that, regression functions are generated that interpolate the average read/write latency with precision. The paper identifies optimal consistency setting by using regression functions which will help the developers to find out settings such that required consistency level is obtained.

Our presented work shows how different consistency setting affect the Cassandra response time and throughput. Because Cassandra provides the feature of tuneable consistency, it is possible to achieve strong consistency by finding optimal settings. By monitoring various parameters of Cassandra database while different combinations of workload, threads and consistency settings are executed, we try to find certain consistency setting that provides the minimum latency.

4. EXPERIMENTATION

4.1. Experiment Objective

To describe a methodology for benchmarking the performance of Cassandra. To extract experimental results, show how different consistency settings influence the latency and throughput. To understand the relationship between the parameters and generate a regression equation for predicting the parameters. The experiment extracts results based on two scenarios:

1. When the read and write operations are executed individually.
2. When mixed read and write workload are executed.

To narrow down the available options for consistency setting based on results obtained. The objective also includes generating a data set for finding multiple regression equations which can be used to perform predictive analysis and to find an optimal setting such that strong data consistency is guaranteed.

4.2. Cassandra Cluster Setup

A Cassandra cluster of 3 nodes with different IP addresses is deployed on VMware. All the nodes are connected in a cluster by installing Cassandra in all of them and configuring them. A replication factor of 3 is configured for ALL consistency to be applied. The data in the nodes is 3-replicated which means a row in a table has 3 copies in the cluster. The VMware virtual machine uses CentOS operating system that is based on Linux. YCSB benchmark is used in order to evaluate the performance of databases under different workloads. The YCSB Client is a Java program that generates data for database loading and runs the loaded workloads.

Three nodes with IPs: 192.168.29.143, 192.168.29.144, and 192.168.29.145 are deployed in a single cluster such that they are connected and Cassandra is installed on each.

4.3. Results

4.3.1. Experiment 1

In the experiment 1 where the performance of Cassandra is measured by considering the read and write workload individually, the configuration is made as follows. A Cassandra cluster of 3 nodes is deployed on the VMware. In our study, the focus is on examining the dynamic features of Cassandra's performance in various consistency settings. We investigate how the current workload affects database latency and throughput. The following configuration is made for experiment 1.

- A replication factor of 3 is configured.
- Nodes have a Keyspace YCSB and table USERTABLE for experimentation purposes.
- YCSB workload c [read] and workload a [write] parameterized to execute only write operations are used.
- 25,000 records are used for loading and execution
- The results are calculated with
    - a Varying number of threads from 10 to 1000.
    - 3 consistency settings: ONE, QUORUM, and ALL.
- Latency and Throughput for all the combinations are measured for further analyses.
- Regression equations

Read Write Latencies and Throughput Measurements

The tables below show the results of the Cassandra performance benchmarking. The average latency and throughput for read and write requests are shown. For each request, the results are calculated using 25000 records. We may use a mix of average delay and throughput to look at how average read and write delays are affected by the current workload.

Table 4. Cassandra READ latency statistics

Threads	Average latency, us			Throughput,ops/s		
	ONE	QUORUM	ALL	ONE	QUORUM	ALL
10	22705	23645	44586	400	386	174
50	58040	59129	70685	736	724	605
100	90431	94485	108955	888	872	762
200	150734	154128	167620	1084	1025	938
300	199428	200451	219143	1140	1108	1002
400	251888	253844	272117	1150	1113	1103
500	267875	298233	350792	1203	1166	1049
600	366471	376034	448648	1197	1130	940
700	426929	396847	484428	1186	1235	811
800	481751	500140	571021	1154	1161	946
900	505123	532567	580480	1193	1184	1075
1000	596130	611603	631444	1140	1120	1107

Table 5. Cassandra WRITE latency statistics

Threads	Average latency, us			Throughput,ops/s		
	ONE	QUORUM	ALL	ONE	QUORUM	ALL
10	31100	32908	45034	300	275	196
50	55668	64989	68692	750	666	593
100	78148	75230	91153	1008	1058	884
200	138283	162299	84844	974	983	768
300	172168	192620	218842	1203	1137	988
400	235078	203891	271080	1156	1274	993
500	280063	356982	392521	1208	1192	952
600	308273	312009	359718	1371	1293	1157
700	356290	362819	375992	1372	1322	1218
800	409974	429026	503939	1220	1250	1096
900	449739	466700	504209	1419	1378	1128
1000	531026	544698	620622	1437	1390	1262

Latency graphs

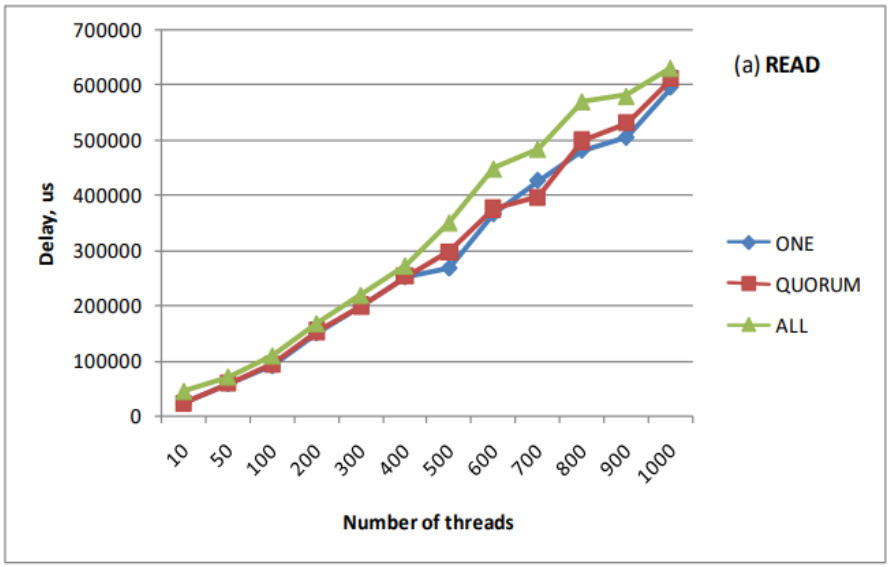

Figure 1. Average Cassandra delay depending on the current workload: reads

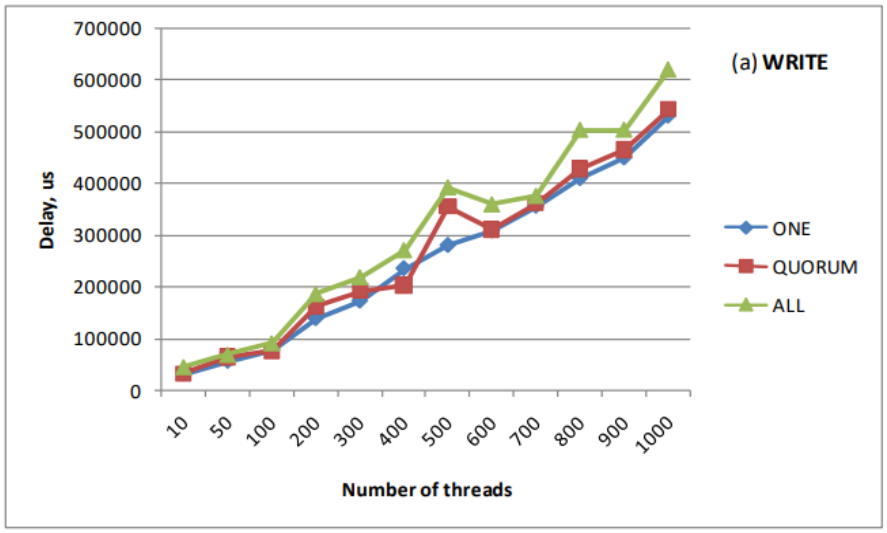

Figure 2. Average Cassandra delay depending on the current workload: writes

Throughput graphs

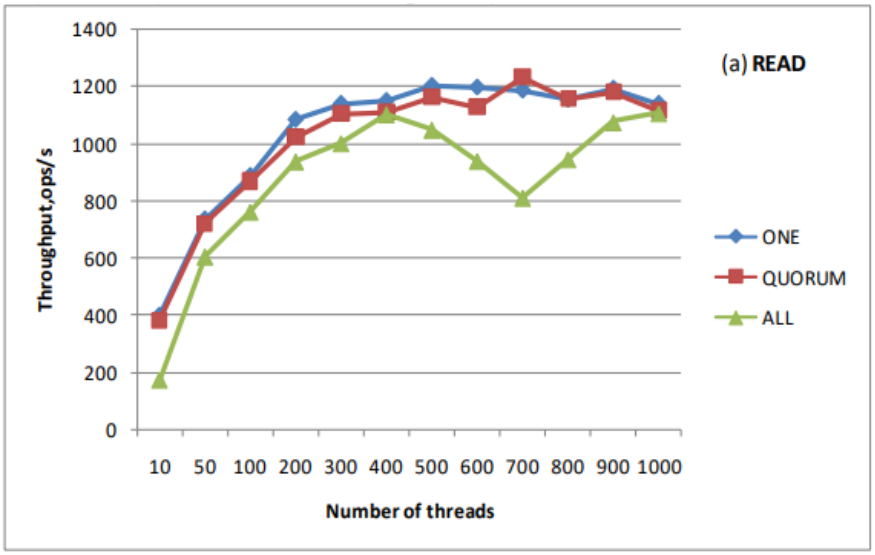

Figure 3. Cassandra Throughput depending on the current workload: reads

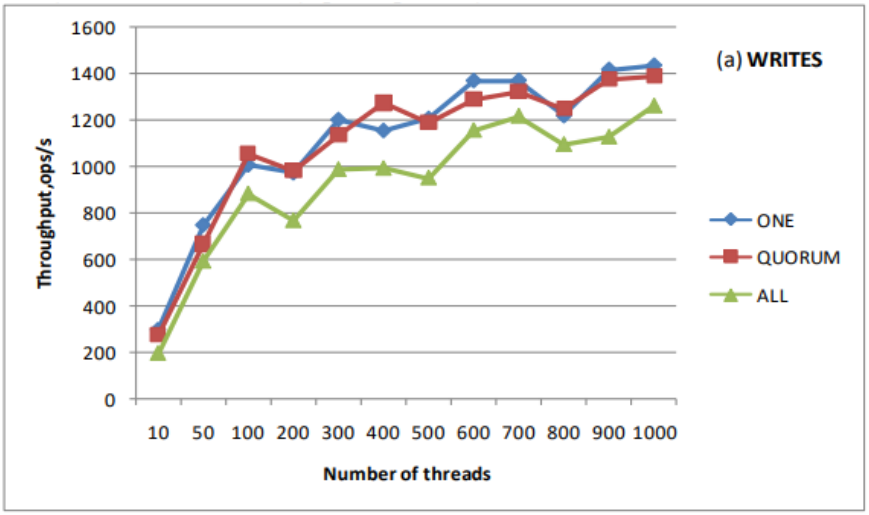

Figure 4. Cassandra Throughput depending on the current workload: writes

Experimental Results

Cassandra reads with the ONE consistency level achieve a maximum throughput of 1203 requests per second, as shown in Table 4. It varies between 1240 and 110 requests per second for the QUORUM and ALL consistency levels. For writes, it is 1437 for ONE consistency level and it fluctuates around 1400 and 1250 for QUORUM and ALL consistency setting respectively.

The graphs in Figure 1 and 2 show the delay experienced for read and write operations individually. The X-axis represents the number of threads running and the Y-axis represents the delay in microseconds. The three lines denote the average latency for ONE, QUORUM, and ALL consistency settings. The average latency for ALL consistency settings is the highest compared

with ONE and QUORUM. However, as shown in Figure 3 and 4, the throughput for ALL consistency settings is the lowest for both read and write operations.

4.3.2. Experiment 2

As already discussed, if the overall number of written and read replicas is more than the factor of replication, the Cassandra database can ensure the maximum data consistency model. This means for a 3-replicated system there are six different read/write consistency settings that can be used to provide high data consistency. They are

- 1R-3W: One read-All write
- 2R-2W: Quorum read-Quorum write
- 3R-1W: All read-One write
- 2R-3W: Quorum read-All write
- 3R-2W: All read-Quorum write
- 3R-3W: All read-All write

Besides, the two settings: 1R-3W and 2R-1W provide the 66.6% of consistency. Finally, the 1R-1W setting can guarantee only the 33.3% of consistency [12]. Whenever a smaller number of replicas are invoked read/write operations in Cassandra executes faster. Hence, in real life experiments, the following consistency should be chosen: 1R-3W, 2R-2W and 3R-1W. All the three combinations follow the rule:

$$(nodes\_written + nodes\_read) > replication\_factor$$

As all the three consistency settings provide strong consistency, a system developer may want to know the performance of those settings for different read/write load proportions and different read/write consistency settings.

Read/Write Latency measurements

For this experiment, 5 different read/write load proportions are taken into consideration: Read/Write-10/90%, Read/Write-30/70%, Read/Write-50/50%, Read/Write70/30% and Read/Write-90/10%. For each of these 5 proportions, read and write latency are measured for 3 consistency settings such as 1) ‘Read ONE – Write ALL’ (1R-3W) 2) ‘Read QUORUM – Write QUORUM’ (2R-2W) 3) ‘Read ALL – Write ONE’ (3R-1W). Table 6 to 10 shows the measured results. The consistency setting that fetches the lowest latencies is highlighted. The tables below show some estimations of Cassandra latency for various configurations, ensuring good consistency in a mixed read/write workload.

Table 6. READ and WRITE latency for ratio: 10/90%

Threads	Read/write	10/90%				
	1R-3W		2R-2W		3R-1W	
	read latency	write latency	read latency	write latency	read latency	write latency
10	30331	31031	191277	50288	42065	31306
50	67304	80219	175507	84587	80445	67491
100	108690	113468	231514	121740	144463	101449
200	149067	153665	308020	198210	187619	154400
300	245266	258348	400140	282352	239309	213881
400	290760	360503	415130	319530	337565	258517
500	372124	481837	455951	366129	431158	323984
600	394041	499268	495030	446155	456912	402001
700	503725	523590	569443	515119	477099	443565
800	612811	669392	760502	593398	637259	611418
900	634603	690514	774205	667434	675742	612999
1000	693981	710275	829634	781465	804184	765775

Table 7. READ and WRITE latency for ratio: 30/70%

Threads	Read/write	30/70%				
	1R-3W		2R-2W		3R-1W	
	read latency	write latency	read latency	write latency	read latency	write latency
10	32741	45848	41353	30100	42690	39421
50	75024	95098	62709	60774	72801	69542
100	111084	138188	148051	121493	246692	224263
200	187287	232319	352954	296963	319321	249553
300	233918	290167	372207	346133	455106	317409
400	256812	301175	404357	370283	517383	363734
500	357096	363032	512833	437759	605245	410471
600	416477	481412	553125	522874	651345	619337
700	533552	597259	638761	603694	677148	618079
800	628928	658895	712611	682823	732824	641235
900	656188	698454	795098	732968	741928	706249
1000	703524	739317	796232	748007	769981	732211

Table 8. READ and WRITE latency for ratio: 50/50%

Threads	Read/write	50/50%				
	1R-3W		2R-2W		3R-1W	
	read latency	write latency	read latency	write latency	read latency	write latency
10	33895	56687	36924	34661	46568	39165
50	73229	95830	89636	87565	99267	89607
100	114747	156710	142052	134773	156400	141365
200	204456	249069	240831	219370	308926	252583
300	253925	297516	328745	299773	353231	301648
400	291299	381609	387702	334432	437584	363444
500	353225	434046	496390	397122	543788	458692
600	427640	501726	534005	461660	550183	472276
700	648993	740161	787573	658604	759187	668920
800	756730	842811	887630	749299	895984	767130
900	783249	879269	892225	792315	907415	857217
1000	853586	964892	970729	885770	973234	871643

Table 9. READ and WRITE latency for ratio: 70/30%

Threads	Read/write	70/30%				
	1R-3W		2R-2W		3R-1W	
	read latency	write latency	read latency	write latency	read latency	write latency
10	24600	31393	30218	29344	39219	32329
50	68325	81604	35040	71423	86173	73635
100	102237	127953	117078	113457	134405	111624
200	180272	235217	199588	184491	230216	184480
300	253253	329891	257724	328285	569271	518260
400	280630	342402	331645	304551	532370	411360
500	406752	529736	408194	360940	763241	486193
600	470798	585201	466305	436402	714981	521906
700	463361	551020	532713	538365	908323	727998
800	499385	589462	669009	629421	819209	669959
900	544131	638540	719122	671458	942113	788877
1000	612822	701998	918065	845165	968210	812331

Table 10. READ and WRITE latency for ratio: 90/10%

Threads	Read/write	90/10%				
	1R-3W		2R-2W		3R-1W	
	read latency	write latency	read latency	write latency	read latency	write latency
10	21729	27576	40283	38839	45563	33537
50	58921	79251	98406	93869	122738	97894
100	93010	138732	151495	142770	167254	129479
200	195954	246562	243233	228524	278509	213629
300	207471	309810	330112	311948	390988	299755
400	286479	366534	408961	300051	381964	538683
500	342750	400440	475445	445930	738210	620529
600	617369	732549	627722	581649	792640	659901
700	644989	726347	675910	632609	811068	690821
800	760282	871117	750107	715626	833562	713990
900	807910	895528	849403	786800	876981	844081
1000	869993	953896	911467	888315	921107	753156

Experimental Results

The 1R-3W configuration delivers the lowest consistency for threads till 200 when the read load proportion is less than 30%. For threads from 200, the 3R-1W setting shows optimal latency among others. When the read load proportion increases, it can be observed that, regardless of the current workload, the 1R-3W option delivers the best latency readings when compared to others. For a read and write proportion of 90/10 %, the 2R-2W setting shows the lowest consistency for a greater number of threads. As the number of requests per second and the fraction of read requests increases, the 2R-2W and specifically the 3R-1W arrangements becomes extremely wasteful. When the percentage of read requests is around 10%, the 3R-1W design still provides the shortest delay in high write-heavy workloads.

4.4. Correlational Analysis

To generalize our results, a multiple regression equation is generated such that it identifies the optimal write consistency factor for the given workload. Syntax of multiple regression equation:

$$Y = \text{Constant } C0 + C1*(X1) + C2*(X2) + C3*(X3) + C4*(X4) \quad (1)$$

The dependent variable Y is the write consistency measure needed to provide strong consistency. There are 4 independent variables: X1-read latency, X2-write latency, X3-threads, and X4-proportion of write workload. To make all of the parameters on the same scale, they are compressed. The following multiple regression formula is created based on the 200 records measured in our experiment:

Table 11. Multiple regression equation static

	Coefficients	*Standard Error*	*P-value*
Intercept	**0.5173**	0.0653	2.57E-13
X1	**-3.876**	0.2967	1.166E-27
X2	**3.5528**	0.4448	1.777E-13
X3	**0.3473**	0.2889	0.231
X4	**0.0739**	0.0853	0.3872

$$Y=0.5173-3.876*X1+3.5528*X2+0.3473*X3+0.0739*X4 \quad (2)$$

Multiple Regression for Read Latency

Table 12. Multiple regression equation for read latency

	Coefficients	*Standard Error*	*P-value*
Intercept	**0.1598**	0.016	6.39E-19
x1	**-0.1257**	0.0142	7.73E-16
x2	**0.8071**	0.0177	2.7E-99
X3	**-0.0708**	0.0204	0.0007

$$Y=0.1598-0.1257*X1+0.8071*X2-0.0708*X3 \quad (3)$$

Here the parameter Y is the read latency measured for various read and write combinations.

Multiple Regression for Write Latency

Table 13. Multiple regression equation for write latency

	Coefficients	*Standard Error*	*P-value*
Intercept	**0.1**	0.0128	5.396E-13
x1	**0.0018**	0.0114	0.8721
X2	**0.7827**	0.0142	7.98E-113
X3	**-0.0981**	0.0164	1.227E-08

$$Y=0.1+0.0018*X1+0.7827*X2-0.0981*X3 \quad (4)$$

Here the parameter Y is the write latency measured for various read and write combinations

5. CONCLUSIONS

To measure Cassandra's latency and performance, we used benchmarking approach. The benchmarking is performed to assess system performance in order to establish how well the system can handle a mixed workload when different consistency settings are employed.

Our research focuses on the relationship between multiple settings for consistency and the performance of the Cassandra column-oriented database. The findings suggest that consistency settings have a considerable impact on Cassandra's response time and throughput, which must be taken into account during system development and monitoring. The Cassandra database gives programmers the ability to fine-tune the consistency setting for each read and write operation request. Software developers can assure strong consistency for their setup by managing the consistency setting by ensuring that the sum of nodes written to and read from is more than the replication factor. In our research, the aim is to choose optimal consistency setting such that strong consistency is provided along with lower latency for our experiment-specific setup.

REFERENCES

[1] Github: Benchmarking Cassandra and other NoSQL databases with YCSB. https://github. com/cloudius-systems/osv/wiki/Benchmarking-Cassandra-and-other-NoSQL-databaseswith-YCSB.
[2] Mishra, V. (2014), Beginning apache Cassandra development. Apress [E-book].
[3] P. Bagade, A. Chandra and A. B. Dhende, "Designing performance monitoring tool for NoSQL Cassandra distributed database," International Conference on Education and e-Learning Innovations, 2012, pp. 1-5, doi: 10.1109/ICEELI.2012.6360579. Eben Hewitt. Cassandra: The Definitive Guide. O'Reilly Media, Inc., 1 edition, 2010.
[4] Datamodel - cassandra wiki. http://wiki.apache.org/cassandra/DataModel.
[5] Daniel Bartholomew. Sql vs. nosql. Linux J., 2010.
[6] Lourenço, J.R., Abramova, V., Vieira, M., Cabral, B., Bernardino, J. (2015). NoSQL Databases: A Software Engineering Perspective. In: Rocha, A., Correia, A., Costanzo, S., Reis, L. (eds) New Contributions in Information Systems and Technologies. Advances in Intelligent Systems and Computing, vol 353. Springer, Cham. https://doi.org/10.1007/978-3-319-16486-1_73.
[7] Abramova, Veronika & Bernardino, Jorge & Furtado, Pedro. (2014). Evaluating Cassandra Scalability with YCSB. 8645. 199-207. 10.1007/978-3-319-10085-2_18.
[8] Eben Hewitt. Cassandra: The Definitive Guide. O'Reilly Media, Inc., 1 edition, 2010.
[9] Pritchett, Dan. (2008). Base an acid alternative. ACM Queue. 6. 48-55. 10.1145/1394127.1394128.
[10] D. Abadi, "Consistency Tradeoffs in Modern Distributed Database System Design: CAP is Only Part of the Story," in Computer, vol. 45, no. 2, pp. 37-42, Feb. 2012, doi: 10.1109/MC.2012.33.
[11] Lakshman, Avinash & Malik, Prashant. (2010). Cassandra — A Decentralized Structured Storage System. Operating Systems Review. 44. 35-40. 10.1145/1773912.1773922.
[12] Gorbenko, A and Romanovsky, A and Tarasyuk, O (2020) Interplaying Cassandra NoSQL Consistency and Performance: A Benchmarking Approach. Dependable Computing - EDCC 2020 Workshops. EDCC 2020. Communications in Computer and Information Science., 1279. pp. 168-184. ISSN 1865-0929 DOI: https://doi.org/10.1007/978-3-030-58462-7_14.
[13] Abramova, Veronika & Bernardino, Jorge. (2013). NoSQL databases: MongoDB vs cassandra. Proceedings of the International C* Conference on Computer Science and Software Engineering. 14-22. 10.1145/2494444.2494447.
[14] Cooper, Brian & Silberstein, Adam & Tam, Erwin & Ramakrishnan, Raghu & Sears, Russell. (2010). Benchmarking cloud serving systems with YCSB. Proceedings of the 1st ACM Symposium on Cloud Computing, SoCC '10. 143-154. 10.1145/1807128.1807152.

© 2022 By AIRCC Publishing Corporation. This article is published under the Creative Commons Attribution (CC BY) license.