Automatic Adjustment of Read Consistency Level of Distributed Key-value Storage by a Replica Selection Approach

Thazin Nwe[1], *Tin Tin Yee*[1], *Myat Pwint Phyu*[1], Ei Chaw Htoon[2], Junya Nakamura[3]
University of Information Technology, Myanmar[1], *Computer University (Kyaing Tone), Myanmar*[2]
Toyohashi University of Technology, Japan[3]
{thazin.nwe,tintinyee, myatpwintphyu}@uit.edu.mm[1], *eichawhtoon@uit.edu.mm*[2], junya@imc.tut.ac.jp[3]

Abstract

In distributed key-value storage systems, Apache Cassandra is known for its scalability and fault tolerance. In such systems, Cassandra is a peer-to-peer architecture which any user can connect to any node in any data center and can read and write data anywhere. Most of the systems usually select a fixed number of replicas for read/write requests in key-value storage. When the more replicas a read request chooses, it may increase the response time and reduce the system performance. In this paper, a consistent replica selection approach is proposed to automatically select number of consistent replicas by defining the read and write consistency level. This approach searches the nearest replicas and selects the consistent replicas depending on the current time, nearest arrival time, read/write latency and version for each read request. The proposed approach tends to achieve the read/write performance of client requests for key-value storage system by reducing the read/write execution time, latency cost and storage cost.

Keywords- Consistent Replicas, Consistency Level, Key-value storage

1. Introduction

Replication is a widely used technology in distributed key-value storage systems to achieve data availability, durability, fault tolerance and recovery. In these systems, maintaining data consistency of replication becomes a significant challenge. Although many applications benefit from strong consistency, latency sensitive applications such as shopping carts on e-commerce websites chooses eventual consistency. Eventual consistency is a weak consistency that does not guarantee to return the last updated value [5]. Eventually consistent systems are high operation latencies and thus in bad performance.

Achieving high throughput and low latency of responses to client requests is a difficult problem for cloud services. To fix these issues, a replica selection process needs to include mechanisms for filtering and estimating the latency when processing requests. The replica selection process is inherently complicated.

Therefore, this paper proposes a replica selection approach for read access in distributed key-value storage systems. A key-value store is a simple database that uses an associative array as the fundamental data model where each key is associated with one and only one value in a collection. This relationship is referred to as a key-value pair.

This approach can determine the minimal number of replicas for reading request needs to contact in real time by defining the consistency levels (one, two, quorum, local quorum, etc.). Depending on these consistency levels, the system can choose the nearest consistent replicas using replica selection algorithms. By using these algorithms, the system will improve the read/write execution time on defining the consistency levels and reduce the read/write latency cost on choosing the nearest consistent replicas.

2. Related Works

Geo-distributed storage systems tend to forward client's requests towards the "close" replicas to minimize network delay and to provide the best performance. This task commonly occurs, e.g., in self organizing overlays. One of the primary tasks is to correctly compute or estimate the distance between the nodes; various systems has tackled this problem. Meridian et al. [4] is a decentralized, lightweight overlay network that can estimate the distance to a node in the network by performing a set of pings that are spaced logarithmically from the target. Kirill Bogdanov et al. [2] demonstrate the need for dynamic replica selection within a Geo-distributed environment in a public cloud. Second, they propose a novel technique of combining symbolic execution with lightweight modeling to generate a sequential set of latency inputs that can demonstrate weaknesses in replica selection algorithms. The sequential set of latency inputs is the consecutive latency that describes the network conditions.

According to [6, 7], there are two traditional mechanisms that can generally be used as how to implement consistency management in large scale

systems: an optimistic mechanism which does not immediately propagates changes and therefore tolerates replica content divergence, and Pessimistic mechanism prevents conflicts by blocking or aborting operations as necessary.

Harmony [1] is a system that can dynamically adjust replica consistency according to the application requirements. It proposes an estimation model to predict the stale read. By collecting read/write access frequency, network latency, most recent read/write access time and other information, it can predict the stale read ratio in real time and achieve the required consistency level with relatively good performance of elastically increase or decrease the number of replicas involved in each read request. Harmony uses a White box model, which decides the replicas numbers of each request by using mathematical formula derivation. To compute the number of replicas to be involved in a read operation necessary, this model finds the stale read rate smaller or equal to the defined threshold value. However, since there are so many factors that can impact the result and lots of those factors change in real time, such white box analysis may not get precise results. Besides, Harmony assumes the request access pattern meets Poisson process, however, different application' access patterns are different, which means Harmony has its usage limitation.

In most systems, it defines the rate of stale read that can be tolerated, and then try to improve system performance as much as possible while still not exceed such stale read rate. However ZHU, Y et al. [8] takes another mechanism, the longest response time is defined that it can tolerate and try to enhance the consistency level as much as possible within this time. The read/write access is broken into 6 steps: reception, transmission, coordination, execution, compaction and acquisition, and each of which can further break into smaller steps. Then a linear regression is used to predict the execution time and latency of the next request for each step. When a request comes, it maximizes the number of steps this request covers within the tolerated time, thus achieves the maximize consistency. However, the stale read rate of this system is unpredictable.

P.Bailis et al. [3] introduces Probabilistically Bounded Staleness (PBS) consistency. PBS describes two ways to estimate the staleness of data: version based and time based. Firstly, a closed-form solution is derived for version based data staleness. Then it models time based staleness and applies it in Dynamo style systems [11]. PBS uses Monte Carlo simulation to describe the time based data staleness. The paper is inspired by PBS and uses the same Monte Carlo simulation to estimate the minimal replica number a read request needs to contact in order to get a specific fresh data rate. An adaptive replica selection algorithm [9] determines minimal number of replicas for each read request needs to select in order to achieve a specific consistency level by estimating the time interval between current read request and nearest write request. However, this algorithm doesn't consider the version based staleness. W.golab et.al [10], proposed the methods of quantifying the consistency in eventually consistent storage systems. That paper described the comparisons of the staleness methods for the stale read problems and issues of Probability of Bounded Staleness (PBS) that does not consider workloads where writes overlap in time with reads.

3. Read/Write consistency level of Cassandra

Cassandra offers tunable data consistency across a database cluster. This means a developer or administrator can decide exactly how strong (e.g., all nodes must respond) or eventual (e.g., just one node responds, with others being updated eventually).

This tunable data consistency is supported across single or multiple data centers, and a developer or administrator has many different consistency options from which to choose. Moreover, consistency can be handled on a per operation basis, meaning a developer can decide how strong or eventual consistency should be per SELECT, INSERT, UPDATE, and DELETE operation.

Cassandra provides automatic data distribution across all nodes that participate in a "ring" or database cluster. There is nothing programmatic that a developer or administrator needs to do or code to distribute data across a cluster. The data is transparently partitioned across all nodes in either a randomized or ordered fashion, with random being the default. Cassandra also provides built-in and customizable replication, which stores redundant copies of data across nodes that participate in a Cassandra ring. This means that if any node in a cluster goes down, one or more copies of that node's data are available on other machines in the cluster.

Unlike complicated replication schemes in various RDBMSs or other NoSQL databases, replication in Cassandra is extremely easy to configure. A developer or administrator simply indicates how many data copies are desired, and Cassandra takes care of the rest. Replication options are also provided that allow for data to be automatically stored in different physical racks (thus ensuring extra safety in case of a full rack

hardware failure), multiple data centers, and cloud platforms [14].

Data consistency is the synchronization of data on all its replicas in the cluster. The number of replicas that need to acknowledge the write request to the client application is determined by write consistency level and the number of replicas that must respond read request before returning data to the client application is specified on the reading consistency level. Consistency levels can be set globally or on a per-operation basis. Few of the most used consistency levels are stated below:

• ONE

A response from one of the replica nodes is sufficient.

• Quorum

A response from a quorum of replicas from any data center. The quorum value is found from the replication factor by using the formula. Quorum = (Replication Factor/2).

• All

All nodes play equal roles; with node communicative with each other equally. There is no master node so there is no single point of failure and all the data has copies in other nodes which secures the data stored. It is capable of handling large amounts of data and thousands of concurrent users or operations per second across multiple data centers.

4. Proposed System

In this architecture, a client writes a file to the replicas as the write consistency level. On Cassandra, the read and write consistency levels (e.g., one, two, quorum, local quorum; etc.) can be defined. In a cluster with a replication factor of three, and the read/ write the consistency level of quorum, the two of the three replicas have to respond to read/write requests. Consistency level describes the behavior seen by the client. Writing and reading at quorum level allows strong consistency.

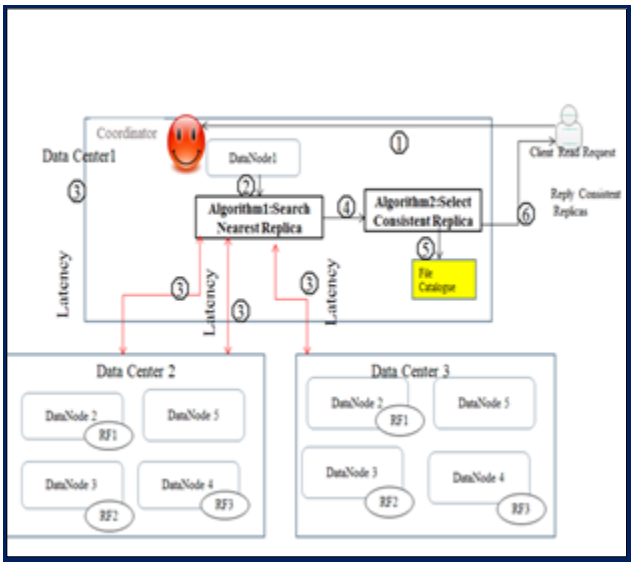

Figure1. Consistent Replica Selection Architecture for reading request in key-value storage

Erasure coding (EC) is a method of data protection in which data is broken into fragments and encoded with redundant data pieces and stored across a set of different locations or storage media. The goal of erasure coding is to enable data that become corrupted at some point in the disk storage process to be reconstructed by using information about the data that's stored elsewhere in the array.

In figure1, the write requests are incoming to the coordinator node. The coordinator node performs the erasure-encoding that divides the data block into m fragments and encode them into n fragments. For example, the incoming data is 5MB; it is split into same size for each MB. And two more 1MB parity pieces are added for redundancy. Therefore, the original block is divided into five fragments and then stored on five separate Cassandra key-value storage nodes and two redundant nodes. In this case, the proposed system writes totally 7*7 (write consistency level * fragments) into a cluster. The fragments created are saved by consistent hashing [12] on different quorum nodes. The acknowledgement of successful writes is sent to the coordinator node.

Secondly, when the client reads a file, it sends a read request to the coordinator Node. The Coordinator Node collects the list of DataNodes that it can retrieve data by using the replica selection algorithm described in the next section. When sufficient fragments have been obtained, the Coordinator Node decodes the data and supplies it to the read application request. In this case

where a client reads from the cluster the file with the read consistency level of five. Therefore, the coordinator of the read request retrieves 5*5 (read consistency level *fragments) from data nodes. If two of five nodes fail, the data cannot be lost.

5. Algorithm Definition

The replica selection algorithm has two parts. It includes (i) searching nearest replica and (ii) selecting consistent replica. In algorithm_1, the coordinator node sends the request message to each replica and latencies of different replicas are listed in the read latency map. And it chooses the lowest latency of replica from this map.

In algorithm_2, the replica selection algorithm in the coordinator node chooses the consistent replica from nearest replicas.

1. **Input**: Replicas RF= { RF_1, RF_2,….RF_n}
2. **Output**: Nearest Replica NR
3. **Set** latencyCost= MAX_VALUE;
4. **Set** lowestLC []=null; //Initialize return lowest latency replica
5. **For** each r in RF // RF=Replicas
6. **Begin**
7. **Set** latencyCost=getLatencyCost(RF_r, job);
8. **If**(latencyCost<=MAX_VALUE)Then// MAX_VALUE =threshold values
9. MAX_VALUE =latencyCost;
10. lowestLC.add (RF_r);
11. **End**
12. **End for**
13. **Return** lowest LC //nearest replica NR

Algorithm 1: Search nearest Replica

Search nearest Replica part executes in two stages. First, all replicas are sorted based on their physical location, so that all replicas in the same rack and then the same datacenter as the source are at the top of the list. Second, the latencies are computed from the local node (originator of the query) to all other nodes. If the latency cost is greater than a threshold of the closest node, then all replicas are sorted based on their latency costs. Finally, the top replicas from the list are chosen.

Firstly, total numbers of replica are listed as input (line1). The threshold value is set at the latency cost of line3. In line8, the coordinate node contacts every other replica with request messages. The round trip time it takes from the request until the reply is passed through T_{total} =$RTT_{requsest}/_2$ $T_{processing}$+$RTT_{reply}/_2$. T_{total} is used to get the latency cost of computing data nodes in algorithm1. These costs are used when the local node needs to forward client requests to other replicas.

And then total times taken from different replicas are listed in latencyCost (line8). Finally algorithm1 returns the list of lowest latency cost of the replicas in lowestLC as output to client. (line14).

1. **Input**: Nearest Replica NR= {NR_1,NR_2,…NR_n}
2. **Output**: Consistent Replicas
3. **For** each Nearest Replica NR_i
4. **Begin**
5. **Set** RCL=2//ConsistencyLevel.QUORUM
6. **Set** noOfConsitentRead=0
7. **While**(noOfConsistentRead<=RCL)
8. **If**(stalerate<=maxStalteRate)**Then**
9. consistentRead.add(NR_i)
10. noOfConsistentRead++;
11. **Return** consistentRead;
12. End for
13. End

Algorithm 2: A Consistent Replica Selection

In algorithm_2, the set of the nearest replicas is collected as input that comes from output of algorithm_1 by computing latency costs. And then algorithm_2 sets the read consistency level (RC) that the client will need the most up-to-date information. Read/Write latencies of different replicas are listed in history file on the coordinator node.

This algorithm determines the number of consistent replica nodes, one read request should select in real-time, according to calculate arrival times of nearest update request and the processing order of read request and write request in different replicas.

For computing stale rate of algorithm_2, a quorum system obeys PBS k-staleness consistency if with probability 1-p_{sk}; at least one value in any read quorum has been committed within k versions of the latest committed version when the read.

$$p_{sk} = \left(\frac{\binom{N-W}{R}}{\binom{N}{R}} \right)^k \quad (1)$$

In eq. 1, When N=3, R=W=1, this means that the probability of returning a version within 2 versions is $0.\bar{5}$, within 3 versions is $0.\overline{703}$, within 5 versions is > 0.868, and within 10 versions is > $0.\overline{98}$.

When N=3, R=1, W=2 (or, equivalently, R=2, W=1), these probabilities increase: k=1 -> 0.6, k=2 -> 0.8, and k=5 > 0.995.

A quorum system obeys PBS (k,t)-staleness consistency if, with probability 1 - p_{skt}, at least one value in any read quorum will be within k versions of the latest committed version when the read begins, provided the read begins t units of time after the previous k versions commit. A quorum system obeys PBS k-staleness consistency with probability 1 $-p_{sk}$ where p_{sk} is the probability of non-intersection with one of the last k independent quorums.

$$p_{st} = \frac{\binom{N-W}{N}}{\binom{N}{R}} + \sum_{c \in (W,N]} \frac{\binom{N-c}{N}}{\binom{N}{R}} \cdot [P_w(c+1,t) - P_w(c,t)]$$

(2)

The above equation makes several assumptions. Reads occur instantly and writes commit immediately after W replicas have the version. T-staleness in real systems depends on write latency and propagation speeds.

6. Analysis of read/writes execution time

The read/write execution time of consistency level is tested by using Cassandra cluster on VMware Ubuntu 14.04 LTS i386. The processor is Intel(R) Core(TM) i7-4770 CPU @ 3.40 GHz. Installed memory (RAM) is 4.00GB as shown in table1.

Table1. Hardware Specification and Virtual Environment

Operating System	VMware Ubuntu 14.04 LTS i386
RAM	4.00GB
Hard-disk	195GB
Processor	Intel(R) Core(TM) i7-4770 CPU @ 3.40 GHz
Cassandra	version: 1.0.6

The staff data from Ministry of Higher Education is used on Cassandra cluster. Staff information is described by Unicode in "staff.csv".

When importing data from the csv to Cassandra, java hector code truncate the input csv data with a comma (",") line by line. And then the output csv data are exported on Cassandra.

Figure2 shows Cassandra supports Unicode, but, Hbase does not support it. Therefore, staff data can be tested on Cassandra cluster. Unicode is the international accepted standard by the World Wide Web Consortium, the main international standards organization for the World Wide Web. And it also makes that it is extremely easy to translate the Wikipedia's interface. And Unicode fonts support 11 languages that use the Myanmar script: Burmese, 2 liturgical languages: Pali and Sanskrit, 8 minority languages: Mon, Shan, Kayah, four Karen languages and Rumai Palaung [13]. It was officially released by Myanmar Natural Language Processing (NLP) Research Center joining existing Myanmar Unicode 5.1.

Figure2. Unicode on Cassandra Cluster

Ubuntu 14.04 LTS is installed on three servers and one client by Cassandra clusters. There are 203 rows and 40 columns from csv file are inserted into one of Cassandra servers and replicate it interconnected other servers. And Replication Factor (RF), Read Consistency Level (RCL) and Write Consistency Level (WCL) are defined by changing the consistency level (one, two and quorum) and tested by Java hector code. Write execution time of servers and read time of the client according to consistency level are shown in figure3. According to this figure, the read/write execution time of consistency level (quorum) is better than consistency level (one and all).

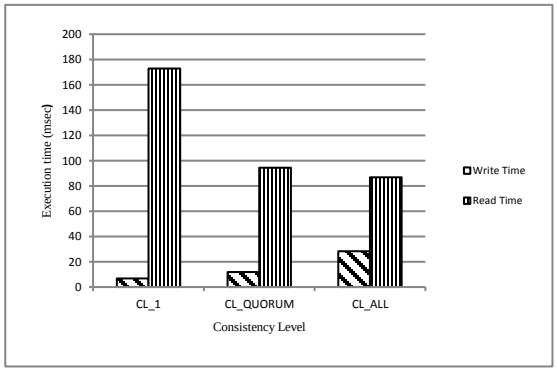

Figure3. Read/Writes execution time of consistency level (one, two and quorum)

7. Conclusion

The paper presents the performance of the consistency level (one, all and quorum) for read/write requests in Cassandra key-value data storage. In defining these consistency levels, a replica selection

approach is proposed for choosing the consistent replicas in different clusters by searching the nearest replica and selecting the consistent replica. For a specific application, its read/write access pattern, network latency and system load always change dynamically. Therefore, at different time, to reach the same consistency level, the impact on system performance is different. In this approach, the arrival time of read/writes request latencies, timestamp and versions are used to choose the consistent replicas near the clients. And this approach can determine the minimal number of replicas for reading request needs to contact in real time and thus improve the system performance as a result of reduced read/write execution time, latency cost and disk storage cost.

8. Future Work

In the future, the proposed algorithms will be validated on Cassandra clusters. And predicted t-visibility and latency will be compared with measured values and will compute the stale read rate for consistent replicas by adding more nodes and more dataset size on Cassandra and MongoDB distributed key-value data storage. And read/writes execution time, latency cost and storage cost of the system will be compared with the existing system.

9. References

[1] H. Chihoub, S. Ibrahim, G. Antoniu and M. S. Perez, "Harmony: Towards Automated Self-Adaptive Consistency in Cloud Storage", IEEE International Conference on Cluster Computing, September 24-28; Beijing, China , 2012.

[2] K. Bogdanov, M. Pe´on-Quir´os, Gerald Q. Maguire Jr.Dejan Kosti´c, "The Nearest Replica Can Be Farther Than You Think", ACM 978-1-4503-3651-2/15/08, 2015.

[3] P. Bailis, S. Venkataraman, J. M. Hellerstein, M. Franklin and I. Stoica. "Probabilistically Bounded Staleness for Practical Partial Quorums", Proceedings of the VLDB Endowment. 5, 8 , 2012.

[4] B. Wong, A. Slivkins, SIRER and E. G. Meridian, "A lightweight network location service without virtual coordinates", in ACM SIGCOMM Computer Communication Review, vol. 35, ACM, pp. 85–96, 2005.

[5] W. Vogels, "Eventually consistent", CACM, 52:40–44, 2009.

[6] Y. Saito and H. M. Levy, "Optimistic replication for internet data services", in International Symposium on Distributed *Computing*, pages 297–314, 2000.

[7] Y. Saito and M. Shapir, " Optimistic replication", ACM Comput. Surv., 37(1):42–81, 2005.

[8] Y. Zhu and J. Wang. Malleable, "Flow for Time-Bounded Replica Consistency Control", OSDI Poster, October 8-10; Hollywood, USA , 2012.

[9] Z.Ye and Weijian, "An Adaptive Replica Selection Algorithm for Quorum based Distributed Storage System", International Journal of Grid and Distributed Computing Vol.9, No.5,2016.

[10] W.golab, Muntasir R. Rahman. et.al, "Eventually Consistent: Not What You Were Expecting?", Volume 12 Issue
1, January2014CM.

[11] G. DeCandia, D. Hastorun, M.Jampani, G.Kakulapati, A.Lakshman, A.Pilchin, S.Sivasubramanian, P.Vosshall and W.Vogels, "Dynamo: Amazon's Highly Available Key-value Store", ACM 978-1-59593-591-5/07/0010, 2007.

[12] P. Garefalakis , P. Papadopoulos, I. Manousakis, and K.Magoutis, "Strengthening Consistency in the Cassandra Distributed Key-Value Store", International Federation for Information Processing 2013.

[13] https://wikivisually.com/wiki/Burmese_ (language)

[14] A. Basith, "Introuduction to Apache Cassandra", April 22, 2017.