Automatic Adjustment of Consistency Level by Predicting Staleness Rate for Distributed Key-Value Storage System

Thazin Nwe [1], Junya Nakamura [2], Ei Chaw Htoon [1], Tin Tin Yee [1]
[1] *University of Information Technology, Myanmar*
[2] *Toyohashi University of Technology, Japan*
thazin.nwe@uit.edu.mm, junya@imc.tut.ac.jp, eichawhtoon@uit.edu.mm, tintinyee@uit.edu.mm

Abstract

Nowadays, Distributed Key-Value storage is extremely useful in almost every large system. Most of the data management systems use Distributed Key-Value database in order to manage and process large volume of data in real time. In such databases, Apache 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 these systems usually select a fixed number of replicas of 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. Consistency in Key-Value data storage systems requires any read operation to return the most recent written version of the content. The system provides transaction services which provide NoSQL databases with enhanced consistency by varying the read and write quorum size. The proposed approach tends to automatically select the minimum number of consistent replicas by predicting the staleness rate on adjusting the consistency level.

Keywords- Distributed Key-Value Storage, Consistency Level, Quorum, Apache Cassandra, Staleness rate

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 choose eventual consistency [6]. Eventual consistency is a weak consistency that does not guarantee to return the last updated value [8]. 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 consistent replica selection process needs to include mechanisms for estimating the latency when processing requests.

Therefore, this system proposes a consistent replica selection approach to read/write access to distributed key-value storage systems by encoding and decoding data onto Distriuted Hash Table (DHT).

This approach can determine the minimal number of replicas for reading request needs to contact in real time by defining the consistency level (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 of defining the consistency levels and reduce the read/write latency cost of choosing the nearest consistent replicas.

There are two parts in the system. These are read and write operations for automatic adjustment of the Consistency Level of Distributed Key-value Storage by a Replica Selection Approach. For the write execution time, data is distributed among the nodes in the cluster using Distributed Hash Tables (DHT) that the atomic ring maintenance mechanism over lookup consistency. DHT is mentioned in Section 5. To get the consistent data for the read performance, two algorithms are proposed to Section 4.

2. Related Work

Performance and reliability of quorum based distributed Key-Value storage systems [1, 5] are proposed in the literature. H. Chihoub et al. [3] proposed an estimation model to predict the stale read and the system adjusted replica consistency according to the application requirements. Harmony uses a White box model uses mathematical formula derivation to choose the replicas numbers of each request. To select 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 result and the system did not consider the performance of read/write operations. Harmony assumes the request

access pattern meets Poisson process. However, Harmony has usage limitation because different application access patterns are different.

In most systems, it defines the rate of stale read that can be tolerated, and then try to improve the system performance as much as possible while still not exceed such stale read rate. However ZHU Y et al. [9] 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 cover within the tolerated time, thus achieves the maximize consistency. However, the stale read rate of this system is unpredictable.Tlili et al. [7] proposed that a master peer is assigned by the lookup service of the DHT. The master node holds the last update timestamp to avoid reading from a stale replica. However, this system cannot get the precise result of the consistent data and read/write execution time will reduce compared with encoded data on DHT.

3. Proposed Architecture

In our architecture, a client writes data onto the replicas as the write consistency level, i.e., quorum. In the proposed system, the read and write consistency levels (e.g., one, two, quorum, local quorum; etc.) can be defined. For example, if the replication factor of a cluster (N) is defined as five, the sum of read consistency level (R) and write consistency level (W) is greater than a replication factor of a cluster (N). Consistency level describes the behavior seen by the client. Writing and reading at quorum level allows strong consistency.

Data Encoding is a method of data protection in which data is encoded with redundant data pieces and stored across a set of different locations or storage media. The goal of data encoding 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. The encoded data of all the transactions are stored in the block.

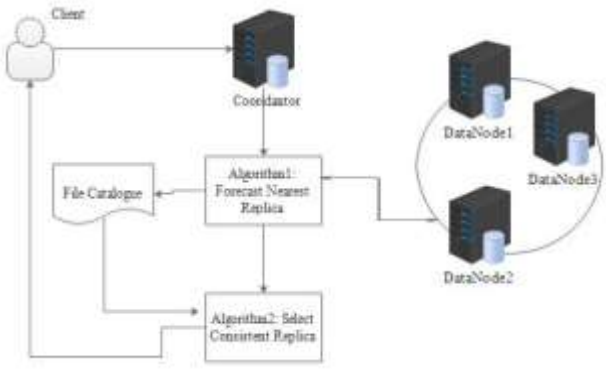

Figure 1. Consistent Replica Selection Architecture for reading request in Key-Value storage

The system has two parts such as read/write requests. The write requests are incoming to the Coordinator Node. The Coordinator Node performs the encoding operation that divides the data block into m fragments and encode them into n fragments. The fragments created are saved by consistent hashing [10] on different quorum nodes. This means that a coordinator must contact to multiple nodes to decode an original replica. For example, the coordinator has to contact 2 (read consistency level) * 5 (fragments) nodes if RCL=2 and data encoding divides original data into 5 fragments and 2 parities. The acknowledgement of successful write request is sent to the Coordinator Node.

Secondly, when the client reads data, 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 request. In this case where a client reads from the cluster the file with the read consistency level of five. 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.

Read and Write operations of the distributed Key-Value storage system to dynamically adjust the consistency level are mentioned in this section. In the write part of the system, Data is distributed among the nodes in the cluster using Consistent Hashing based Function. Consistent Hashing is a widely used technology in distributed key-value storage system. It is a good solution when the number of nodes changes dynamically. And when the virtual node is combined, the load balancing problem will also be solved. Consistent hashing is the algorithm the helps to figure out which node has the

key. The algorithm guarantees that a minimal number of keys need to be remapped in case of a cluster sizes change. DHTs characteristically emphasize the following properties:

- Autonomy and decentralization: the nodes collectively form the system without any central coordination.
- Fault tolerance: the system should be reliable (in some sense) even with nodes continuously joining, leaving, and failing.
- Scalability: the system should function efficiently even with thousands or millions of nodes.

In figure 2, when the write requests are incoming to the coordinator node. The coordinator node performs the encoding input data is saved for the Cassandra cluster by consistent hash on different quorum nodes.

When the client reads a file, it sends a read request for 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.

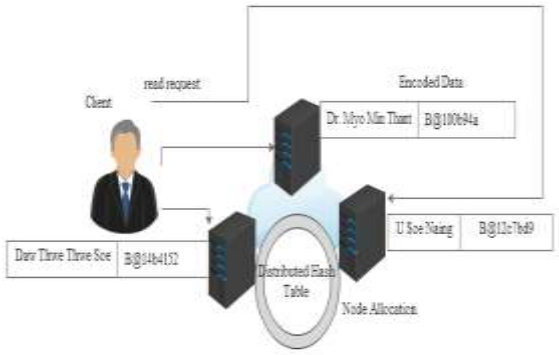

Figure 2. System Architecture

3.1 Storing history Data in Blockchain Architecture

A proof of concept Blockchain system that holds history data containing their timestamps, versions and log data of read/write request time from Key-Value storage cluster.

Blockchain technology promises network-distributed, decentralized and immutable data storage and transaction conduction. The information contained within the blocks make up a database where adding or changing information in that database comes in the form of appending new blocks to the blockchain. A block consists of a header, containing mostly metadata, and transactions that hold the actual blockchain information making up the database.

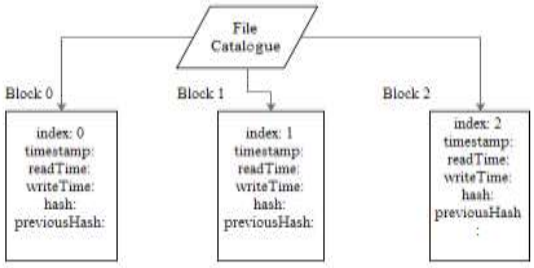

Figure 3. Storage of file catalogue in Blockchain Architecture

4. Algorithm Definition

In algorithm-1, the nodes are mapped to a circle (the value is $0 \sim 2^{32}$) by a specific hash function. There are three nodes (n1, n2, n3), and this algorithm use Consistent Hashing Approach to map three nodes to a circle.

The replica selection algorithm has two parts- (i) searching the nearest replica and (ii) selecting consistent replicas. In algorithm-2, the coordinator node sends the request message to each node who has one or more replicas and latencies of different replicas are listed in the read latency map. And it chooses the lowest latency of replicas of this map.

The Coordinator node executes the replica selection algorithm (algorithm 3) and chooses the consistent replica of the nearest replicas.

Input: The nodes of Cassandra Cluster
Output: Distributed Hash Table Nodes
For each n in Cassandra Cluster
Begin
For r=1,2,…n do
Process the hash function for each node in DHT
Return DHT nodes
End for
End

Algorithm 1: Distributed Hash Table Nodes

Set lowestLC [] =null; //Initialize return lowest latency replica
For each r in DHT
Begin
Set latencyCost=getLatencyCost (RF_r, job);

```
        If
          (latencyCost<=MAX_VALUE)
        Then
          MAX_VALUE=latencyCost;
  //MAX_VALUE =threshold values
          LowestLC. add (RF_r);
          Return lowest LC //nearest replica NR
    End
  End for
```

Algorithm 2: Search the nearest Replica

Search the nearest Replica part is divided into two stages. First, all replicas are sorted based on their physical location, so that all replicas of the same rack and then the same data center 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 of the list are chosen.

Firstly, total numbers of replicas are listed as input. The threshold value is set at the latency cost. The coordinator node contacts every other replica with request messages. The round tripped to the time it takes from the request until the reply is passed through the following equation (1).

$$T_{total} = \frac{RTT\ request}{2} * Tprocessing + \frac{RTT\ reply}{2} \quad \textbf{(1)}$$

T_{total} is used to get the latency cost of computing, data nodes in algorithm2. These latency costs are used when the local node needs to forward the client request to other replicas.

And then total times taken from different replicas are listed in latencyCost. Finally algorithm2 returns the list of lowest latency cost of the replicas as output to client.

```
Input: Nearest Replica NR= {NR_1, NR_2,… NR_n}
Output: Consistent Replicas
For each Nearest Replica NR_i
  Begin
      Set RCL=2//ConsistencyLevel.QUORUM
      Set noOfConsistentRead = 0
   While (noOfConsistentRead <= RCL)
     If (stalerate <= maxStaleRate) Then
       consistentRead.add (NR_i)
       noOfConsistentRead++;
     Return consistentRead;
  End
End for
```

Algorithm 3: Consistent Replica Selection

In algorithm-3, the set of the nearest replicas is collected as input that comes from the output of algorithm 1 by computing latency costs. And then algorithm 2 sets the read consistency level (RCL) 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 calculated arrival times of nearest update request and the processing order of the read request and write request in different replicas.

For computing stale rate of algorithm 3, In this section, we will use PBS to simulate more cases of replica number n, read consistency level r, and write consistency level w to prove the conclusion. Bailis et al proposed PBS to predict the consistency [2]. They believed the quorum size impacts the consistency significantly and given a formula to calculate the probability of k-staleness [2]. In staleness estimation, a client estimates the staleness of each node in the quorum, based on the last write date values provided in server, and select only those secondaries whose staleness is less than or equal to maximum staleness rate. The calculation of staleness is the following equation (2).

$$p_{st} = \frac{\binom{N-W}{N}}{\binom{N}{R}} + \sum_{c \in (W,N]} \frac{\binom{N-c}{N}}{\binom{N}{R}} \cdot [P_w(c+1,t) - P_w(c,t)] \quad \textbf{(2)}$$

For quorum system, consider N replicas with randomly chosen read and write quorums of sizes R and W. The system calculates the probability that the read quorum does not contain the last written version. This probability is the number of the quorums of size R composed of nodes that were not written to in the write quorum divided by the number of possible read quorums.

4.1 PBS <k, t > staleness

PBS <k, t > staleness combines both versions and real-time staleness metrics to determine the probability that a read will return a value no older than k versions stale if the last write committed at least t seconds ago:

A quorum system obeys PBS <k, t> staleness consistency, if, with probability 1 − pskt, 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.

The probability of inconsistency is calculated by the following equation (3):

$$P_{skt}=\frac{\binom{N-W}{R}}{\binom{N}{R}}+\sum_{c\in(W,N)}\frac{\binom{N-c}{R}}{\binom{N}{R}}.[P_w(c+1,t)-Pw(c,t)])k \quad \textbf{(3)}$$

The staleness result of this equation is calculated in section 5.

5. Experiment and performance evaluation

We evaluate the proposed algorithms and read/write execution time of consistency level by using a Cassandra cluster of 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.

Table 1. 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 Exchange rate currency is used in Cassandra cluster. Exchange rate currency information is described by Unicode in "currency.csv". When importing data onto the csv to Cassandra, Java hector code truncates the input csv data with a comma (",") lined by line. And then the output csv data are exported on Cassandra.

Ubuntu14.04 LTS is installed on three servers and one client of the Cassandra clusters. There are 203 rows and 40 columns from csv file are inserted into Cassandra clusters with Consistent Hashing DHT (Distributed Hash Table).

DHT is one of the fundamental algorithms used in distributed scalable systems. DHT deals with the problem of locating data that are distributed among a collection of machines. In the general case, a lookup service may involve full-content searching or a directory-services or structured database query approach of finding data record that matches multiple attributes. Lookup service similar to a hash table: (Key, Value) pairs are stored in a DHT, and any participating node can efficiently retrieve the value associated with a given key. Responsibility for maintaining the mapping from keys to values is distributed among the nodes [4].

The encoded data by writing execution time for distributed DHT nodes are better than simple DHT [10]. DHT is a widely-used solution to search the nearest neighbor node that transforms the data to obtain a short code consisting of a sequence bits. Each experiment is repeated five times and the average result is considered for performance evaluation. The staleness rate for consistency label is mentioned in figure 4, 5. If client reads with QUORUM, then 2 nodes will get fresh data, and when the system reads with QUORUM, then the system calculates the staleness rate and choose the replica with minimum staleness rate to get fresh data. The following results are calculated by Probability of Bounded Staleness based on time and version is mentioned in section 4.1.

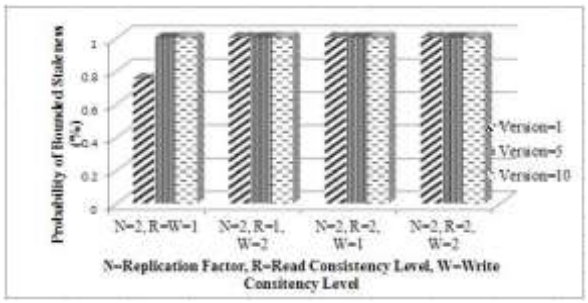

Figure 4. Probability of Staleness rate for Consistency Level on Replication Factor=2

When N=3, R=W=1, this means that the probability of a staleness rate within 1 version is 0.75, within 3 versions, 3.38 ends, 5versions, 7.59. When N=3, R=2, W=2, these probabilities decreases that within 2 versions is 1.44, within 3 versions is 1.75, and within 5 versions is 2.49.

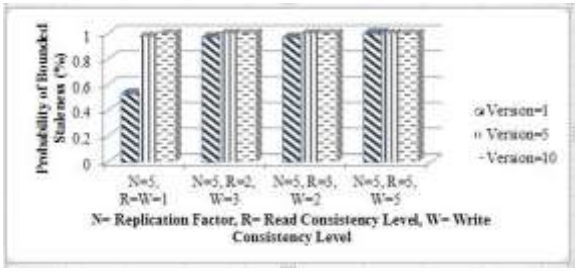

Figure 5. Probability of Staleness rate for Consistency Level on Replication Factor=5

When N=5, R=3, W=5, this means that the probability of a staleness rate within 2 versions is 2.37, within 3 versions, 3.64 and, 5 versions, 8.60. When N=5, R=3, W=8, these probability increases that within 2 versions is 1.71, within 3 versions is 2.24, and within 5 versions is 3.83. The system defines the decision how to define the read/write consistency level by predicting the staleness rate.

For QUORUM of both read & write requests, the system gets strong consistency per following equation

R + W > N, where R - read replica count, W - write replica count, and N - replication factor.

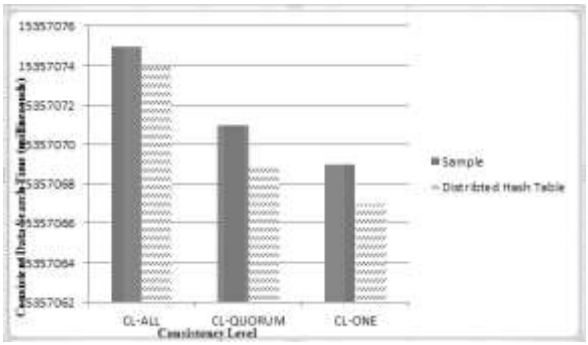

Figure 6. Search Time for consistent data in DHT

In figure 6, the system compares the search time of consistent data onto DHT and sample search. By using DHT, the closest node to the client is looking for the data with the identifier. If the data doesn't exist, this node which sends it to other nodes which send it back to sender node. Therefore, the system reduces the size of the jump at each step and are faster than sample DHT.

6. Future Work

In the future, the proposed algorithms will be validated with more datasets in distributed Key-Vale storage system and the performance of the system is measured by storing log data in Blockchain Architecture.

7. Conclusion

The proposed system presents the prediction of the staleness rate for Key-Value data storage of financial data onto DHT by adjusting the consistency level. In defining these consistency levels, two algorithms are proposed to choosing the consistent replicas of different clusters by searching the nearest replica and selecting the consistent replica. The proposed algorithms can determine the minimal number of consistent replicas of reading request needs to contact in real time and thus improve the system performance as a result of reduced read/write execution time. The system examined a real-time data replicated storage system in which content updates are replicated and stored in a quorum systems. Either a write or read request goes to all the nodes in the system, and it is considered complete once there are at least w or r responses. Assuming that the write delay dominates the latency, the freshness of the replicated storage system is measured by the average age of the content returned by a read at any time t.

8. References

[1] Agrawal, D., El Abbadi, A.: The generalized tree quorum protocol: An efficient approach for managing replicated data. ACM Trans. Database Syst. 17(4), 689.

[2] Bailis P, Venkataraman S, Franklin MJ, Hellerstein JM, Stoica I. Probabilistically bounded staleness for practical partial quorums. Proc VLDB Endowment. 2012;5(8):776–787.

[3] 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.

[4] Jingdong Wang, Ting Zhang, Jingkuan Song, Nicu Sebe, and Heng Tao Shen, "A Survey on Learning to Hash", IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, VOL. 13, NO. 9, APRIL 2017.

[5] Malkhi, D., Reiter, M.: Byzantine quorum systems. pp. 569_578. TOC'97,ACM(1997),http://doi.acm.org/10.1145/ 258533.258650.

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

[7] Tlili, M., Akbarinia, R., Pacitti, E., Valduriez, P.:Scalllable P2P reconciliation infrastructure for collaborative text editing, Second International Conference on Advances in Database Knowledge and Data Applications (DBKDA), pp. 155_164, April 2010.

[8] W. Vogels, "Eventually consistent", CACM, 52:40–44, 2009.I. S. Jacobs and C. P. Bean, "Fine particles, thin films and exchange anisotropy," in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.

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

[10] Thazin Nwe, Tin Tin Yee, Ei Chaw Htoon, Automatic Adjustment of Read Consistency Level of Distributed Key-Value Storage by a Replica Selection Approach, ITC-CSCC Bangkok, vol.33, pp. 740-741, 2018.