Consistent Replica Selection Mechanism for Cloud Data Storage

Thazin Nwe, Tin Tin Yee, Myat Pwint Phyu, Ei Chaw Htoon
University of Information Technology, Yangon, Myanmar
tintinyee@uit.edu.mm, myatpwitphyu@uit.edu.mm, eichawhtoon@uit.edu.mm

Abstract

Cloud computing is becoming increasingly popular and cloud storage services attract more attentions for their high security and availability with a low cost. Cloud storage is expected to become the main force of the future storage market. To achieve high data availability, cloud storage services rely on replication. Data replication is a widely used technology in quorum based data center distributed storage system. In such systems, it is important to choose a closest set of replicas to service a client request for increasing performance and reliability. Static replica selection would be suboptimal even over a dedicated infrastructure. Suboptimal replica choices result in reduced performance as a result of increased response latency. In this paper, a consistent replica selection mechanism is proposed to automatically select number of consistent replicas by searching nearest replicas and selecting consistent replicas depends on current time, nearest arrival time, read/write latency and version for each read request. This mechanism tends to achieve the performance of client requests for cloud services by reducing latency cost.

1. Introduction

Cloud computing is the new computing paradigm which provides large pool of dynamical scalable and virtual resources as a service on demand. Nowadays many of these applications are data-intensive: companies like Google, Amazon, and Facebook deal with terabytes of data every day. In this context, storage management and performance within clouds is extremely important. Storage systems often rely on replication to achieve availability, data durability, fault tolerance, disaster recovery.

However, with the use of replication comes the issue of consistency. Insuring data consistency at all times by means of synchronous replication results in very high operation latencies and thus in bad performance. Moreover, cloud storage systems are deployed on a wide area scale and data are replicated over geographically distant areas.

Achieving high throughput and low latency of responses to client requests is a difficult problem for cloud services. Depending on replication policies, the consistency model of a service and the current network state, clients have to choose which replica or set of replicas they will access, using so-called replica selection algorithms.

The replica selection process is inherently hard. To try to compensate for these issues, a replica selection process needs to include mechanisms for filtering and estimating the latency when processing requests.

Therefore, the paper proposes a replica selection mechanism for read access in cloud storage. This mechanism can determine minimal number of replicas for read request needs to contact in real time and thus improve system performance.

The reminder of the paper is organized as follows. The literature review is presented in Section 2. Issues related to replication, Cassandra that use the replication, MogoDB that use the replication are discussed in Section 3. The proposed system is described in Section 4. Finally in Section 5, the system describes the conclusion.

2. Related Work

Geo-distributed storage systems tend to forward client's requests towards the "closest" 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 among the nodes; various systems have tackled this problem. Meridian [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 on a public cloud. Second, it proposes 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.

According to [6, 7], there are two traditional mechanism that can generally be used as to how implement consistency management in large scale systems: an optimistic mechanism which does not immediately propagates changes and therefore tolerates replica content divergence, and a pessimistic mechanism that blocks access to a replica as long as this later is not up to date.

Harmony [1] is a system that can dynamically adjust replica consistency according to the requirement. 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 by elastically increase or decrease the number of replicas involved in each read request. However, Harmony is a white box model, which decides the replicas number of each request by using mathematical formula derivation. 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. Besides, Harmony assumes the request access pattern meets Poisson process, however, different applications' access patterns are different, which means Harmony has its usage limitation.

In most system, it defines the rate of stale read 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, it defines the longest response time it can tolerate and try to enhance consistency level as much as possible within this time. It breaks the read/write access into 6 steps: reception, transmission, coordination, execution, compaction and acquisition, each of which can further breaks into more small steps. Then it uses linear regression to predict the cost time of 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 in this system is unpredictable.

In [3], P. Bailis, S. Venkataraman.et.al introduces Probabilistically Bounded Staleness (PBS) consistency. PBS describes two ways to estimate the staleness of data: version based and time based. Firstly, it derives a closed-form solution for version based data staleness. Then it models time based staleness and applies it in Dynamo style systems. 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.

2.1. Issues Related to Replication and Eventual consistent system

A simple replication-based mechanism has been used to achieve high data reliability of Distributed File System. However, replication based mechanisms have high degree of disk storage requirement since it makes copies of full block without consideration of storage size. Erasure coding mechanism can provide more storage space when used as an alternative to replication. It divides the data into fragments and encodes them into data fragments. It can increase write throughput compared to replication mechanism.

Modern distributed data stores need to be scalable, highly available, and fast. These systems typically replicate data across different machines and often across datacenters for two reasons: first, to provide high availability when components fail and, second, to provide improved performance by serving requests from multiple replicas. In order to provide predictably low read and write latency, systems often eschew protocols guaranteeing consistency of reads and instead opt for eventually consistent. However, eventually consistent systems make no guarantees on the staleness of data items returned except that the system will "eventually" return the most recent version in the absence of new writes [5]. Weakly consistent systems may return stale or out-of-order data without bound..

2.2. Cassandra that use Replica Selection

The system provides background on quorum systems. Cassandra is a highly configurable NoSQL distributed database, designed to work with large datasets in local and geo-distributed environments. Unlike many other distributed databases, Cassandra

can perform read operations with a per request variable consistency level, depending on the client's requirements. The client communicates with one of the replicas (presumably the one closest to the client). This node selects a subset of closest replicas (including itself), forwards the client's request to them and waits for their replies. Once enough replicas have replied, a single reply is sent back to the client. The consistency level can vary from LOCAL - i.e., reads from a single node, QUORUM (i.e., N=2+1 replicas), to ALL where all nodes have to respond prior to returning an answer to the client. Generally, a configurable number of nodes will be used to produce an answer, thus allowing a tradeoff between consistency, availability, and performance – as acceptable to the client.

2.3. MongoDB that use Replica Selection

MongoDB is a popular distributed document management database. It supports atomic, strongly consistent write operations on a per document basis and either strong or eventually consistent read operations depending on the client's preferences. MongoDB implements a master-slave replication strategy, with one primary and a number of secondary nodes. All write operations are directed towards the primary node and eventually propagated to the secondary replicas. Replication increases redundancy, data availability, and read throughput for clients that accept eventual consistency semantics. By default, a client's read operations are directed to the primary machine and return strongly consistent data, although the client has an option to use secondary replicas, or to choose the closest node regardless of its current status.

2.4. Latency Estimation and Consistency in Replica Selection

A well-known example from statistics is a group of Weighted Moving Average (WMA) and Exponentially Weighted Moving Averages (EWMA) functions. This class of functions is commonly used during distance estimation in replica selection algorithms and applied to sampled data, such as RTT or system load.

Latency estimation in MongoDB is responsibility of the client's driver, which chooses the closest server to connect to. MongoDB has multiple clients for compatibility with many programming languages, and currently there are more than a dozen different client's drivers (including open source community drivers). Naturally many of them have been implemented by different developers, thus their implementations have significant differences in peer selection.

Replica selection algorithm Cassandra implements a module called Snitch to help each node to choose the best set of replicas to which to direct read operations. There are several types of Snitches that allow administrators to tailor the logic to the deployment environment.

3. Proposed System

The system proposes a replica selection mechanism to automatically select number of consistent replicas for each read request to achieve the performance of client request for cloud services as shown in figure 1.In this architecture, client writes a file to the replicas as the consistency level. Consistency level describes the behavior seen by the client.

Firstly, 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. The fragments created are saved by consistent hashing on different quorum nodes. The acknowledgement of successful write is sent to the coordinator node.

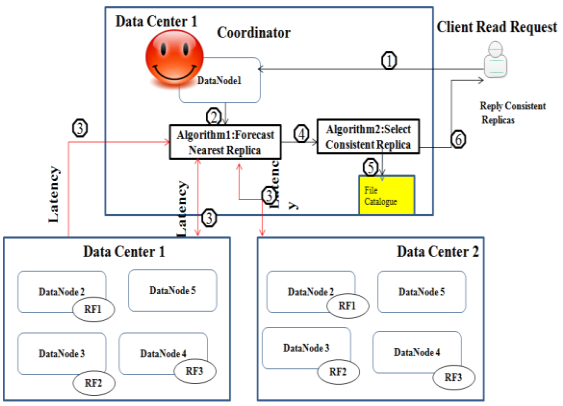

Figure1. Consistent Replica Selection Architecture for read request in cloud storage

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 becomes 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.

An erasure code provides redundancy without the overhead of strict replication. Erasure codes divide an object into m fragments and recode them into n fragments, where n>m. This calls r=m/n<1 the rate of encoding. A rate r code increases the storage cost by a factor of 1/r . The key property of erasure codes is that the original object can be reconstructed from any m fragments. For example, using an r=1/4 encoding on a block divides the block into m=16 fragments and encodes the original m fragments into n=64 fragments; increasing the storage cost by a factor of four. Erasure codes are a superset of replicated and RAID systems. For example, a system that creates four replicas for each block can be described by an (m=1, n=4) erasure code. RAID level 1, 4, and 5 can be described by an (m=1, n=2), (m=4, n=5), and (m=4, n=5) erasure code, respectfully.

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 a replica selection algorithm. When sufficient fragments have been obtained, the Coordinator Node decodes the data and supply it to the read application request.

3.1. Algorithm Definition

A replica selection algorithm has two parts. It includes (i)searching nearest replica and (ii)selecting consistent replica. In Algorithm1: Search nearest replica, the coordinator node sends the request message to each replica and latencies of different replicas are listed in read latency map. And it chooses the lowest latency of replica from this map.

In Algorithm2: A Consistent Replica Selection Algorithm, the replica selection algorithm in coordinator node chooses the consistent replica from nearest replicas.

Replica algorithm executes in two stages. First, all replicas are sorted based on their physical location. Second, the latencies is 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 number of replicas are listed as input(line1). Threshold value is set in latencyCost of line3. Client requests are incoming in jobQueue[ ] in line 5. In line8, the coordinate node contacts every other replica with request messages. The 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.

1. **Input**: Replicas RF= { RF1, RF2,….RFn}
2. **Output**: Nearest Replica NR
3. **Set** latencyCost= MAX_VALUE;
4. **Set** lowestLC[]=null; //Initialize return lowest latency replica
5. **Set** jobQueue[]=null;//incoming client requests
6. **For** each RF // RF=Replicas
7. **Begin**
8. **Set** latencyCost=getLatencyCost(RF,job);
9. **If**(latencyCost<=MAX_VALUE)Then// MAX_VALUE =threshold values
10. MAX_VALUE =latencyCost;
11. lowestLC.add(RF_i);
12. End
13. End for
14. **Return** lowest LC

Algorithm 1: Search Nearest Replica

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

1. **Input**: Nearest Replica NR= {NR1,NR2,…NRn}
2. **Output**: Consistent Replicas
3. **For** each Nearest Replica NR_i
4. **Set** RC=readConsistencyLevel
5. **Set** noOfConsitentRead=0
6. **While**(noOfConsistentRead<=RC)
7. **If**(stalerate<=maxStalteRate)**Then**
8. consistentRead.add(NR_i)
9. noOfConsistentRead++;
10. **Return** consistentRead;

Algorithm 2: A Consistent Replica Selection

In algorithm2: A Consistent Replica Selection Algorithm, nearest replicas are collected as input that come from output of algorithm1 by computing latency costs. And then algorithm2 sets the read consistency level (RC) that the client will need the most up-to-date information. Partial quorum systems can lower latency by requiring fewer replicas to respond. With partial quorums, sets of replicas

written to and read from need not overlap: given N replicas and read and write quorum sizes R and W, partial quorums imply R + W ≤ N. However lowering latency has a consistency cost: contacting fewer replicas for each request typically weakens the guarantees on returned data.

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

3.1.1. Probabilistically Bounded Staleness

The side effect of weakening consistency is increased staleness. Staleness describes the age of a value returned by a read relative to the last updated value, and hence quantifies how badly a system's behavior deviates from the gold standard.

Version-based staleness defines age by counting versions of an object (for example, a read returns the k^{th}-latest version).

Time-based staleness defines age in terms of wall-clock time (for example, a read returns a value t time units older than the last updated value).To make sense of eventual consistency, we turn to relaxed consistency properties−k-atomicity and Δ-atomicity− which give precise meaning to the notions of version-based and time-based staleness, respectively.

The PBS model estimates the probability of <k, t> staleness−the condition that the read, which begins time units after the end of the write, returns the value assigned by one of the last k writes. This condition is similar to k-atomicity but considers only a single read at a fixed distance t from the write. When k = 1, it captures the probability that the read returns the latest value (that is, is not stale).

In PBS k-staleness, 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 begins.

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

When N=3, R=W=1, this means that the probability of returning a version within 2 versions is 0:5, within 3 versions, 0:703, 5. versions, > 0:868, and 10 versions, > 0:98. 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.

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

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

3.1.2. Write Requests

The number of write requests per second on the coordinator nodes, analogous to client writes. Monitoring the number of requests over a given time period reveals system write workload and usage patterns.

3.1.3. Write Request Latency

The average response times (in milliseconds) of a client write. The time period starts when a node receives a client write request, and ends when the node responds back to the client. Depending on consistency level and replicas, this may include the network latency from writing to the replicas.

3.1.4. Read Requests

The number of read requests per second on the coordinator nodes, analogous to client reads. Monitoring the number of requests over a given time period reveals system read workload and usage patterns.

3.1.5. Read Request Latency

The response time (in milliseconds) for successful read requests. The time period starts when a node receives a client read request, and ends when the node responds back to the client. Optimal or acceptable levels of read latency vary widely according to hardware, network, and the nature of application read patterns.

The value is also picked from history records. The consistent replicas are chosen from nearest replica list by using current time, nearest arrival

time, read latency and write latency and version until reaching the read consistency level.

3.2. Expected Result

The proposed system describes a replica selection mechanism is proposed to automatically select number of consistent replicas. For client write request, the coordinator node writes the data fragments to data nodes by using the erasure-encoding. These fragments created are saved by consistent hashing on different quorum nodes. This fact will reduce the cloud storage size. And then for read request, consistent replica selection algorithm will reduce the cost of latency by searching nearest replicas and selecting consistent replica depends on nearest arrival time, read/write request and versions.

4. Future Work

To validate these algorithms, we test them on OptorSim simulation. In the future, we will compare our predicted t-visibility and latency with measured values observed in a commercially available, open source Dynamo style data store by using Cassandra and MongoDB.

5. Conclusion

The paper presents a replica selection mechanism for read request in different replicas by searching nearest replica and selecting 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 mechanism, arrival time of read/write request latencies and versions are used to choose the consistent replicas near the clients. The mechanism makes the clients to get the most up-to-date information by defining consistency level. And this mechanism can determine minimal number of consistent replicas for read request needs to contact in real time and thus improve system performance as a result of reduced response latency .

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] Kirill Bogdanov, Miguel 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. . . $15.00, 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] WONG, B., SLIVKINS, A., AND SIRER, 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] Zhen Ye and Weijian Hu , "An Adaptive Replica Selection Algorithm for Quorum based Distributed Storage System" , International Journal of Grid and Distributed Computing Vol. 9, No. 5, pp.55-68, 2016.