2017 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC)

A Hierarchical Framework with Consistency Trade-off Strategies for Big Data Management

Yingyi Yang, Yi You, Bochuan Gu
Department of Smart Grid
Electric Power Research Institute Of Guangdong Power Grid Corporation
Guangzhou, China

***Abstract*—Geo-replicated cloud storage provides good scalability, availability and fault-tolerance for managing big data of high-volume, velocity and variety nature. However, the prickly trade-off between consistency, cost and response time, which is brought in by geo-replicated cloud storage, poses great challenges to big data management. The goal of this work is to allow geo-replicated cloud storage used in big data management to dynamically switch to an appropriate consistency level at runtime in the consideration of cost and performance constraints. In this paper, we present a hierarchical consistency framework which supports the implementations of a strong protocol and a range of consistency semantics. The framework adopts a set of consistency trade-off strategies at both the data level and the transaction level. Based on a probabilistic model, a cost balance formula and a new metric Consistency-ResponseTime Efficiency defined in the trade-off strategies provides a basis to dynamically switch consistency levels by using performance records collected at runtime. Our evaluation verifies the effectiveness of our hierarchical framework and trade-off strategies.**

***Keywords*—big data management; geo-replicated cloud storage; consistency hierarchical framework; trade-off strategies; tunable consistency**

I. Introduction

With the development of technology and the improvement of demand, the data in the corporations and organizations have an explosive growth in recent years [1]. EB or even ZB data produced in this process is called big data. From an industry point of view, beside three ‘V’s of big data [2]: volume, velocity and variety, two additional dimensions, i.e. variability and complexity, are also considered.

Due to its high availability and scalability, geo-replicated cloud storage becomes an effective solution to tackle emerging challenges brought by big data. Many corporations, such as Google, Amazon and Yahoo! have deployed cloud storage systems [3-5] for big data in their large-scale cloud infrastructure. Geo-replication is one of the necessary technologies for cloud storage systems to solve problems with persistence, performance, availability and fault-tolerance. Despite its importance, geo-replication results in side effects, that is, data consistency [6].

Data consistency has been widely studied in distributed research and practical community. Different consistency guarantees, e.g. causal consistency, read-your-writes consistency, etc., had been proposed [7]. In addition to the important trade-off between consistency, availability and network partitions proposed by the CAP theory [8], there is another non-trivial trade-off between high performance, reasonable cost and some level of consistencies for geo-replicated cloud storage used in big data management. Initially, many solutions either provide strong guarantees in a limited scale [9] or trade eventual consistency [10] for performance and scalability. Though those solutions differ in the degree of consistency, there is only a single level of consistency they provide. In the era of big data, people gradually come to realize that it is difficult for a single consistency level to solve the problem once and for all due of the diversity of big data. From the perspective of application requirements, due to the diversification of big data, different degree of consistency requirements for the same data exist with respect to different types of services or even the same service at different periods of time. From an economic standpoint, users always expect to adjust the consistency level in order to control the overall cost, including consistency cost and penalty cost of inconsistencies, especially in big data management based on geo-replicated cloud storage. In terms of performance, the effect of the implementation of strong consistency semantics on performance should also be taken into account when designing geo-replicated cloud storages for big data management.

To achieve high availability and fault tolerance, data is usually stored in multiple servers not only within a data center but also across data centers in different parts of the world when using geo-replicated cloud storages. According to the above analysis, a range of factors, such as cost, response time, etc., should to be considered carefully when providing a well-designed consistency solution for big data management in this context. In general, designing geo-replicated cloud storages providing a particular consistency level is not the silver bullet for managing big data. Instead, we see it as fundamental to explore a framework that can be tuned to provide different levels of consistency at a middleware layer as well as its appropriate trade-off strategies for geo-replicated cloud storages. In this paper, the goal is to allow geo-replicated cloud storage used in big data to dynamically switch to an appropriate consistency level at runtime in the consideration of cost and performance constraints.

The contributions of this paper are as follows: (a) We introduce a consistency hierarchical framework. It supports the implementations of a strong consistency protocol, named D-Paxos [11], which has high write throughput and low average read latency under the heavy reads and writes, and a range of

This work was partially supported by China Southern Power Grid Company Limited Science and Technology Program under Grant No. GDKJQQ20161201 (036100KK52160049).

consistency semantics. (b) We define a range of consistency trade-off strategies. This set of strategies divides data into two categories requiring different consistency guarantees at the data level. They can also be used to gain a better trade-off between consistency, cost and response time at the transaction level. (c) We propose a cost balance formula and define a new metric called Consistency-ResponseTime efficiency, both of which are based on probabilistic approaches and can be used as a basis to dynamically switch guarantee levels at runtime depending on the prediction made by a probabilistic model.

The rest of paper is organized as follows. Section 2 summarizes related work. A consistency hierarchical framework is introduced in Section 3. In Section 4, we present our consistency trade-off strategies. Based on our framework, Section 5 describes the implementations of those trade-off strategies in detail. The effectiveness of our framework and its trade-off strategies rules is illustrated by simulation in Section 6. The conclusion is followed in Section 7.

II. Related Work

A new transaction paradigm, called consistency rationing is proposed in [12], which allows to automatically switch consistency guarantees at runtime. Data are divided into three categories *A*, *B* and *C*, which are treated differently depending on the consistency level they needed. In particular, some policies providing probabilistic guarantees are proposed for category *B*, which comprises all the data where the consistency requirements vary over time. Using these policies, data in the *B* category switch between session consistency and serializability at runtime. However, the approach cannot be applied with eventual consistency as weaker consistency, since inconsistencies considered in consistency rationing are due to update conflicts rather than delay of update propagation.

According to in-depth analysis of consistency costs of the most common concurrency control protocols, I. Fetai et al. [13] present a cost-based concurrency control C^3. A set of rules specified in C^3 can be used to dynamically select the most appropriate consistency level, with an aim to minimize the overall costs. With C^3, the consistency level of a transaction is determined before the transaction is effectively executed. Developers of transactional systems have to choose the most appropriate consistency level for their transactions, which is a difficult task. Moreover, the impact of factors, such as the timeliness, other than cost on consistency level is not considered.

Pileus [14] can be regarded as an extension and implementation of the adaptive framework for tunable consistency and timeliness using replication. As a replicated and scalable key-value storage system, Pileus provides the implementation of both strong and eventual consistency as well as intermediate guarantees and their interfaces. It also allows applications to declare their consistency and latency priorities via consistency-based service level agreements [15]. Related algorithm is used for selecting a proper target subSLA along with the set of nodes that can best meet this subSLA at the current time.

Harmony [16] is a novel approach that handles data consistency in cloud storage adaptively. In order to provide adequate tradeoffs between consistency and both performance and availability, Harmony adopts stale reads rate to precisely reflect the application requirement and adapt the number of replicas involved in the access operation to the requirement. Though it is an effective approach to dynamically and elastically handle consistency at run time, it does not focus on cost.

Techniques that allow more fine-grained tuning as may be required to support consistency guarantees through service level agreements (SLAs) are also be investigated [17]. A novel technique that assigns the consistency level on a per-operation basis by choosing randomly between two options (e.g. weak vs. strong consistency) with a tunable probability and a known technique that uses weak consistency and injects delays into storage operations artificially are considered in this work.

III. A Consistency Hierarchical Framework

A. Background and Motivation

In the recent years, many corporations and organizations have been migrating their ever-increasing data into geo-replicated cloud storages, in order to obtain highly scalable, highly available and low-cost storage services. However, problems with geo-replicated cloud storages are: (a) From the perspective of application requirements, data requiring strong consistency are unavoidable obviously. But in turn, it is unlikely for big data management applications to provide strong consistency guarantees for all data due to requirements from a fraction of data, which subjects to higher cost and lower performance. (b) Weaker consistency semantics avoid higher overall cost caused by more resources and time needed due to strong consistency guarantees required and may provide better response time, but it is at the risk of potential, extremely high penalty cost due to inconsistencies.

To design a consistency framework for data consistency in big data management, the side effects introduced by geo-replicated cloud storages should also be dealt with carefully. In particular, for heavy reads and writes requiring strong consistency guarantee, it is an important performance problem that how to achieve strong consistency, which has high write throughput and low average read latency in the cloud using geo-replication. Moreover, the framework should give consideration to tunable consistency in exchange for cost saving and timely response.

B. An Overview of the Framework

In order to design such a framework mentioned above, we should address the following main issues: (a) organize replicas in a reasonable way, (b) support for the implementations of different consistency semantics, and (c) develop a set of consistency trade-off rules, taking into full account influence of cost and response time. In this section, we provide our solution for the first two issues.

In the cloud using geo-replication, data are replicated over replicas across multiple data centers. Data centers, replicas and clients accessing to replicas are regarded as entities in our consistency framework. Our framework is separated into two layers, namely, the top layer consisting of all data centers and the bottom layer consisting of all replicas offering the same service, both of which are also called the data center layer and the replica layer respectively. For each data category requiring some level of consistency guarantees, all data centers in the top layer are organized logically into two groups: a primary group and a secondary group, as shown in Fig. 1. Accordingly, data centers in groups are known as primary data centers and secondary data centers respectively. Note that, data categories requiring different consistency guarantees differ in ways of logically grouping data centers in terms of their members and number. Moreover, a distinguished replica is elected in each data center, called delegator. It ensures consistent states among replicas located in the same data center and provides the only interface for messaging between data centers.

C. *Consistency in Each Data Center*

In our framework, consistency guarantees are achieved in two layers, i.e. consistency within each data center in the replica layer and consistency among data centers in the data center layer. Replicas in the same data center are often connected through local area networks of high bandwidth and low latency, which makes it reasonable to implement strong consistency semantics at lower cost in a data center. Consequently, we implement strong consistency guarantees for replicas with the same data center by adopting Paxos [18]. A delegator in each data center serves as the unique leader for Paxos instances. Since high wide-area latency among data centers is the main bottleneck for geo-replicated cloud storages, the key point is to achieve consistency among data centers more efficiently. However, achieving consistency within each data center proves to be important for implementing an effective strong consistency protocol which has high write throughput and low average read latency.

D. *Consistency among Data Centers*

Consistency among data centers consists of two parts: consistency for the primary group and consistency for the secondary group. Groups differ in the degree of consistency they implement: the primary group is used to implement strong consistency semantics, whereas the secondary group is used to implement tunable consistency semantics.

We assume that there are only two requests sent from clients: reads and writes. Writes are only handled by the primary group, whereas reads can be handled by both the primary and the secondary group. The size of the primary group can be tuned to control the cost of implementing strong consistency semantics required by writes. Since the secondary group does not handle writes, the state of the secondary data centers is brought up-to-date through lazy update propagation from the primary group after they have handled write requests. As a consequence, the secondary group can only provide weaker consistency semantics, which is usually reflected through the staleness (or degree of the lagged) of states. By providing relevant strategies, we can switch consistency guarantees in this hierarchical framework, that is, implementing tunable consistency. Obviously, reading from the primary group can obtain the most up-to-date state, but may subject to higher latency, whereas reading from the closest secondary data center enables a fast access, at the cost of staled state and potential monetary penalty. Note that a non-delegator replica from a specified data center in the primary group is selected as the lazy update publisher. Although there are many protocols can be used, we adopt D-Paxos [11] for our primary group, which guarantees all reads and writes being handled by all replicas in the same order.

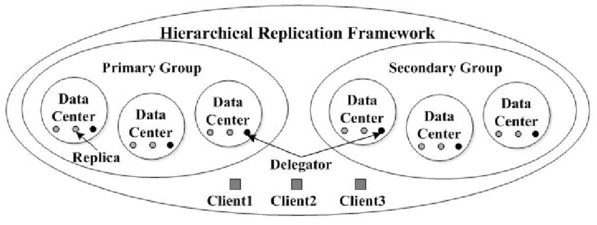

Figure 1. A Consistency Hierarchical Framework

IV. Consistency Trade-off Strategies

Despite of different choices on what degree of consistency should be provided [19, 20], we argue that providing a particular consistency level is not the silver bullet for managing big data based on geo-replicated cloud storage. In the industry, offerings such as Facebook's Cassandra [21] provide more choices of different consistency levels for users. In this paper, a set of consistency trade-off strategies at both the data level and the transaction level is developed in coordination with our consistency hierarchical framework. The purpose is to enable geo-replicated cloud storage used in big data management to dynamically switch to an appropriate consistency level at runtime in the consideration of cost and performance constraints. In simple words, implementing tunable consistency semantics when using geo-replication.

In the strategies, we first divide the data into two categories at the data level, namely, ***S*** *category* and ***T*** *category*. The ***S*** category contains data for which strong consistency semantics are required and ***T*** category contains all the data where the consistency requirements vary over time. On this basis, influences of cost (consistency cost and penalty cost of inconsistency) and response time on the consistency guarantees required by ***T*** category are further considered at the transaction level. For this purpose, a cost balance formula and a new metric Consistency-ResponseTime Efficiency are defined. Based on the prediction made by a probabilistic model, our strategies are able to gain between consistency, cost and response time. Below we present our consistency trade-off strategies in detail.

Data Level Strategy: *Preconditions:* (a) the data require strong consistency guarantees, no matter how much consistency cost need to pay, and (b) the data are provided with infinite consistency budget.

In the first precondition, consistency level is a more important factor than the monetary cost for making the decision. As for the second precondition, choices won't be restricted by the cost factor. Hence, strong consistency must be selected due to the best data quality it provides. We can get the corresponding decision as follows.

Data Level Decision: if one of the above preconditions holds, the data is classified as S data and strong semantics are required; otherwise, the data is classified as T data, the desired consistency level should be further determined through the transaction level strategy 1.

Transaction Level Strategy 1: *Precondition:* the potential penalty cost is higher than the expected savings when using tunable consistency for ***T*** category.

Unlike ***S*** category, ***T*** category may be handled with varying consistency levels because of cost concerns. From an economic perspective, it is reasonable to balance the expected cost savings and potential losses, especially when the overall cost is a major concern. The decision is given as below.

Transaction Level Decision 1: if the precondition holds, ***T*** category should be handled with strong consistency; otherwise, the desired consistency level should be further determined through the transaction level strategy 2.

The key to implement the above strategy is how to establish a formula to compare the expected savings and the potential penalty cost. In addition to the cost constraints, response time is another factor which should be taken into account. Before proposing the transaction level strategy 2, we define a new metric Consistency-ResponseTime Efficiency as below.

$$\text{Consistency-ResponseTime Efficiency} \\ = \text{ConsistencyProbability}(cl)/\text{ResponseTime} \quad (1)$$

Where *cl* is the consistency level specified by clients. ConsistencyProbability(*cl*) is the probability that the data obtained are of desired consistency level specified, and ResponseTime is the time a client takes to receive a response from the data center which provides the data. Intuitively, a high metric value indicates that one data center is able to provide with client specified consistency level in a timely manner with a high probability. Using this metric, the transaction level strategy 2 is expressed as follows.

Transaction Level Strategy 2: *Precondition:* with statistical information obtained about client specified consistency levels and response time, it is possible for clients to calculate a series of Consistency-ResponseTime Efficiency values corresponding to data centers.

Consistency-ResponseTime Efficiency is used by the transaction level strategy 2 as a basis to switch consistency levels. The object of this strategy is to enable clients to achieve as strong semantics as possible in the shortest possible response time. Consequently, the decision for this strategy is as follows.

Transaction Level Strategy 2: depending on metric values of Consistency-ResponseTime Efficiency calculated, the consistency level with the highest metric value is selected as the target consistency level for subsequent reads and the data center which provides such a metric value is selected as the target data center.

V. Design of Protocol and Strategies Based on the Consistency Hierarchical Framework

In this section, we describe the details of the strong protocol designed based on our consistency hierarchical framework and formulas used in our consistency trade-off strategies and how to obtain the relevant threshold and metric values.

A. *Obtain Strong Consistency Guarantees for Data Level Strategy*

Since writes are only handled by the primary group, strong consistency guarantees for reads and writes can only be provided by the primary group. Unlike the secondary group which updates its state through updated state received, the primary group strictly orders reads and writes indiscriminately in order to maintain the sequential relationship between them, and thus providing strong consistency guarantees. Here, we concentrate on the design of a protocol which provides strong consistency among replicas across multiple data centers in the primary group.

In geo-replicated cloud storages, the following problems should be considered carefully: (a) high latency for wide area messaging. Strong consistency protocols which rely heavily on messaging subject to wide area channel with low bandwidth and high latency, resulting in poor performance. (b) unbalanced link dependency pattern. Protocols which are leader-centric usually make most of their networks idle due to their unbalanced communication pattern. Besides, the unique sequencer in each of those protocols usually become the performance bottleneck.

Here, we propose a protocol, called D-Paxos, which provide strong consistency guarantees. It can utilize resources (network and time) in a more efficient way by means of the layered characteristics of our framework. The logical relationships between entities in the primary group adopting D-Paxos are shown in Fig. 2.

The execution stage of D-Paxos between delegators reserves some actions like those in Paxos, including suggest (if the leader proposes a non-empty proposal), skip (if the leader proposes an empty proposal) and revoke (if a recovery phase is initiated by another delegator because of the leader's failure). D-Paxos and Paxos differ in: (a) D-Paxos adopts a rotating leader scheme among delegators from data centers in the top layer. Instances executed in this layer are considered to be sequential and delegators take turns to be the leader of instances coordinated among delegators. The arrow in Fig. 2 indicates the logical sequence between delegators. (b) once serving as a leader, a delegator proposes its pre-ordered sequence of requests, which are generated by running a

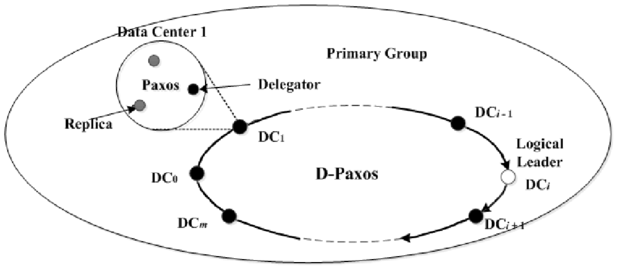

Figure 2. The logical relationships between entities in the primary group

sequence of Paxos instances in its own data center in the bottom layer. If the sequence is empty, then the leader takes skip action instead. In short, delegators are the main participants of D-Paxos and their proposals are pre-ordered sequences of requests generated in their data centers in the bottom layer. To distinguish roles delegators take in the top layer between those they take in the bottom layer, leaders of Paxos instances executed in data centers in the bottom layer are called local leaders, whereas the leaders rotated among delegators in the top layer are called rotating leaders.

Each instance of D-Paxos proceeds in two phases: (a) local consistency phase in the replica layer: before being the rotating leader, a delegator obtains a pre-ordered sequence of requests by coordinating a sequence of Paxos instances in its own data center. (b) global consistency phase in the data center layer: when a delegator becomes the rotating leader, it proposes its pre-ordered sequence of requests obtained in the local consistency phase and waits for acknowledge messages from a quorum of delegators. Once receiving acknowledge messages from a quorum of delegators, the rotating leader learns that its proposal has been chosen and notifies all other delegators. Finally, all replicas in all data centers will be synchronized.

The benefits of D-Paxos are obvious. By efficiently exploiting resources (idle time left by inherent high wide area latency among data centers and high bandwidth available in data centers) for generating pre-ordered sequences of requests in the local consistency phase and proposing them as batches in the global consistency phase, D-Paxos improves the overall throughput though batching and reduces the average latency. Moreover, with the rotating leader scheme in the data center layer, D-Paxos further improves the overall performance in a logical pipelining manner and amortizes the coordinating overhead over all delegators.

Generally, the cost of write requests can be traded off against the cost of strongly consistent read requests [15]. If strong consistency guarantees are required by geo-replicated storage used in big data management under heavy reads and writes, and clients are connected evenly to all data centers, a bigger primary group will accelerate the synchronization among data centers. When the primary group encompasses all data centers, D-Paxos provides high write throughput and low average read latency.

B. A Cost Balance Formula in Transaction Level Strategy 1

As mentioned above, the state of the secondary data centers is brought up-to-date through updated states propagated from the primary group. However, the propagation delay between groups leads to a certain degree of staleness of the secondary data centers. It's important to note that, although a fixed grouping scheme might already be determined for a certain category, indicating a relatively stable storage cost for any consistency level applied in our framework, enforcing different consistency levels still cause different number of messages and different lengths of response time, and thus different network cost and runtime cost. Our cost balance formula is established on the basis of this knowledge.

For ***T*** category, some inconsistencies are tolerable, as long as inconsistencies are still in the acceptable range of big data management applications. Therefore, it is reasonable that there is a staleness threshold specified by the client exists, where an inconsistent state that is more stale than this staleness threshold is thought to cause a penalty cost. In this paper, staleness (the number of updated states lagged) is used to reflect the consistency level of a data center (and its replicas).

In our transaction level strategy 1, a probabilistic model-based cost balance formula is established to compare the potential penalty cost of staleness which is more stale than the staleness threshold caused by tunable consistency and the expected savings of using tunable consistency. We use t to denote the time at which client issues a read request to a secondary data center, which is also used to denote the time at which the target consistency level and target data center are determined. Let $S_i(t)$ denote the staleness (which also reflect the consistency level) of the state of secondary data center i at time t, and s be a staleness threshold specified by the client. Let $P(S_i(t) \le s)$ be the probability that the staleness of the state of secondary data center i at time t is within the staleness threshold. Let $P(S(t) \ge s)$ be the probability that the staleness of the states of secondary group containing all secondary data centers at time t exceeds the staleness threshold. For $P(S(t) > s)$, a higher probability indicating a higher likelihood of all secondary data centers at time t being more stale than the staleness threshold, indicates that the potential penalty cost of staleness caused by using tunable consistency are more likely to exceed the expected savings of using tunable consistency. Let p be the probability threshold which the penalty cost is greater than the expected savings if $P(S(t) > s)$ is greater than p, or vice versa.

Let C_S be the cost of a write to a record when using strong consistency, C_T be the cost of a write to a record when using tunable consistency and C_V be the penalty cost of staleness which is more stale than the staleness threshold caused by tunable consistency. According to the precondition in the transaction level strategy 1, if the potential penalty cost of using tunable consistency $E(X)$ is higher than the expected savings of using tunable consistency, then:

$$C_S - C_T \le E(X) \tag{2}$$

If (2) holds, it is better to select strong consistency. Assuming the potential penalty cost of using tunable consistency $E(X) = P(S(t) > s)* C_V$, (2) becomes:

$$\boldsymbol{P\big(S(t) > s\big) \ge {(C_S - C_T)}/{C_V}} \tag{3}$$

Hence, the probability threshold, which is used to determine whether the potential penalty cost of using tunable consistency is greater than the expected savings of using tunable consistency or not, can be set to $\boldsymbol{(C_S - C_T)/C_V}$.

C. *Estimation of Consistency-ResponseTime Efficiency Metric in Transaction Level Strategy 2*

To estimate the Consistency-ResponseTime Efficiency, the probability that a client obtains data with specified consistency level and the response time it takes are estimated respectively.

Let G_P denote the primary group and G_S denote the secondary group. As mentioned in the previous section, S_i is the random variable denoting the staleness of the state of secondary data center *i*. *t* denotes the time at which a read request is issued. $S_i(t)$ denotes the staleness of the state of data center *i* at time *t*. In this paper, the staleness is used to reflect the consistency level. Let *cl* be the consistency level specified by the client. Hence, $P(S_i(t) \le cl)$ be the probability that the state of data center *i* at time *t* is within the consistency level *cl*. The staleness of the secondary data center *i* at time *t* is considered as the number of write requests received by the primary group since the time t_i at which the secondary data center *i* received the last lazy update. Let d_i be the duration between *t* and t_i. $S_i(t)$ is actually the number of write requests received by the primary group from clients in the duration d_i. We assume that the write requests arrival from clients follows a Poisson distribution with rate λ_w. Hence, we can obtain

$$\boldsymbol{P}\left(\boldsymbol{S}_i\left(\boldsymbol{t}\right) \le \boldsymbol{cl}\right) = \sum_{n=0}^{cl} \frac{\left(\lambda_w \boldsymbol{t}_l\right)^n \boldsymbol{e}^{-\lambda_w d_i}}{\boldsymbol{n}!} \tag{4}$$

Since the staleness is used to reflect the consistency level, the probability ConsistencyProbability(*cl*) that a client can obtain data with specified consistency level from secondary data center *i* can be calculated in (4). A smaller *cl* means a more strict consistency requirement. The greater the probability, the more likely the data center meets the consistency requirement. Formula (4) is extended to include the estimation of the staleness of state of the primary group. Since the primary group implements strong consistency semantics, obviously, the probability ConsistencyProbability(*cl*) for the primary group is ConsistencyProbability(*cl*) = 1.

Since the update propagation from lazy update publisher to any secondary data center is independent and the wide area latency is variable, we assume that the staleness of state of each secondary data center is independent. The probability $P(S(t) > s)$ in the left side of (3) is given by (5).

$$\begin{aligned} \boldsymbol{P}(\boldsymbol{S}(\boldsymbol{t}) > \boldsymbol{s}) &= 1 - \boldsymbol{P}(\boldsymbol{S}(\boldsymbol{t}) \le \boldsymbol{s}) = 1 - \prod_{i \in G_s} \boldsymbol{P}(\boldsymbol{S}_i(\boldsymbol{t}) \le \boldsymbol{s}) \\ &= 1 - \prod_{i \in G_s} \left(\sum_{n=0}^{s} {\lambda_w \boldsymbol{d}_i}^{n_e - \lambda_w d_i} \big/ \boldsymbol{n}!\right) \end{aligned} \tag{5}$$

Here we describe how to measure the values of parameters λ_w, d_i and the response time. Each delegator records the number of requests n_c it has received and handled in the duration t_c between the times at which it become the rotating leader. Once it becomes the rotating leader, it sends $< n_c, t_c >$ previously recorded to all clients. Therefore, the arrival rate λ_w can be computed as $\lambda_w = \frac{\sum_{j \in G_P} \boldsymbol{n}_c^j / \boldsymbol{t}_c^j}{\boldsymbol{P}}$, where *P* is the number of data centers in the primary group. d_i is the duration between the time at which a read request is issued and the time t_i at which the secondary data center *i* received the last lazy update, so $d_i = t - t_i$. The delegator in the secondary data center *i* sends its d_i to clients, which will then be used as input to its consistency selection algorithm. To estimate the response time, clients keep histories about getting responses from different data centers. We assume that the propagation delay T_u from a data center *u* to the client follows the Guass distribution. A sliding window with the size of *n* is used for each data center to collect samples, and the response time w_u can be computed as $w_u = \frac{1}{\boldsymbol{n}} \sum_{v=1}^{n} \boldsymbol{x}_{u,v}$, where $x_{u,v}$ is a sample value.

VI. Evaluation

A. *Simulation Settings*

We construct a simulation environment containing multiple data centers. The experimental setup is composed of four servers ***A***, ***B***, ***C*** and ***D***, each of which simulates a data center. Replicas in data centers are emulated by running concurrent threads. By default, three threads are started in each server, with one of them selected to be the delegator. All data centers are connected by an emulated wide area network. The propagation delay between ***A*** and ***B*** is 160ms, between ***C*** and ***D*** is 340ms, between C and ***A*** as well as ***B*** are both 300ms and between ***D*** and ***A*** as well as ***B*** are both 200ms. Clients which issue read and write requests are emulated by separate threads. In this scenario, the performance bottleneck are network bandwidth and latency. The size of read and write requests adopted in our experiments is 8KB.

B. *The Effectiveness of The Transaction Level Strategy 1*

In this section, we study the effect of transaction level strategy 1 on the trade-off between consistency and cost. The setting in scenario (b) described above is applied in this set of experiments. Emulated clients run in data center ***C***. To gain further insight into this strategy, the overall cost of running the workload with tunable consistency following the transaction level strategy 1 is compared with the overall cost of running the workload with strong consistency as well as the overall cost of running the workload with weak consistency. In this scenario, running the workload with strong consistency or weak consistency refers to always reading from the primary group or always reading from the secondary group respectively. Since the focus is on the effectiveness of strategy 1, a relatively reasonable pricing scheme is provided for different levels of consistency. The cost when using strong consistency is about \$0.12 per 1000 requests, the cost when using weak consistency is \$0.03 per 1000 requests, and the penalty cost of read staleness is about \$0.18.

Fig. 3 shows the trend of the overall cost with increasing number of requests. The curve for strong consistency indicates that the overall cost when using strong consistency increases linearly with the number of requests. That is because no inconsistencies (staleness) occur and no penalty cost caused when using strong consistency. Consequently, its corresponding unit cost to provide strong consistency is relatively fixed. The overall cost when using weak consistency

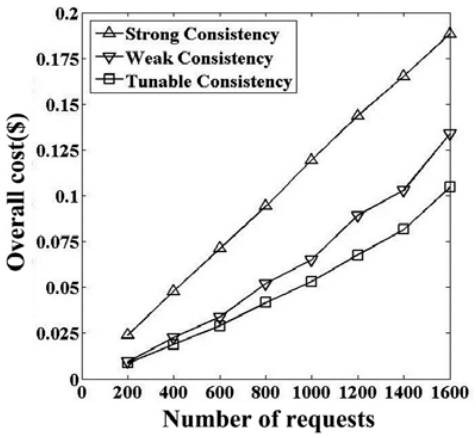

Figure 3. The overall cost when using strong consistency, weak consistency, tunable consistency, varying the number of requests

is lower compared to that when using strong consistency due to its lower unit cost. However, the overall cost when using weak consistency grows faster and faster with the increasing number of requests. The reason is that the increasing number of requests cause a busy network, which has more and more negative impact on the degree of staleness, and thus increasing the overall penalty cost. Using tunable consistency following the transaction level strategy 1 has the lowest overall cost in this scenario. When the probability that the potential penalty cost is higher than the expected savings when using tunable consistency exceeds the specified probability threshold, consistency level will be switched to strong consistency according to the transaction level strategy 1, which avoids high penalty cost. Thus, the transaction level strategy 1 finds the right balance between weak consistency (to save cost) and strong consistency (to avoid penalty cost).

The penalty cost affects the determination of the probability threshold defined in our transaction level strategy 1 and has a big influence on the overall cost. Keeping the number of requests constant, we observe the effect of the penalty cost on the overall cost. As shown in Fig. 4, the overall cost when using strong consistency is constant because all reads are handled in the primary group. No staleness occurs and no penalty cost exists. When using weak consistency, the overall cost is lower in contrast to that when using strong consistency at a penalty cost of \$0.12. However, the overall cost increases very rapidly with an increasing penalty cost and then exceeds the overall cost when using strong consistency at a penalty cost of about \$0.18. When using tunable consistency, the probability threshold decreases with the increasing penalty cost. To avoid the potential and higher total penalty cost, the transaction level strategy 1 enforces more and more reads at strong consistency, that is, reading at the primary group. In this case, the overall cost converges to the overall cost of using strong consistency.

C. The Effectiveness of The Transaction Level Strategy 2

The goal of this experiment is to verify the effectiveness of the transaction level strategy 2. To this end, the penalty cost is set to a very small value, so that rather than selecting strong consistency immediately in the transaction level strategy 1, the desired consistency level will be determined through the transaction level strategy 2. In this experiment, the primary group only contains data center ***A*** while the secondary group contains data centers ***B***, ***C*** and ***D***. Therefore, no protocol is needed to run in the primary group. Moreover, the client runs in a remote site simulated by a newly added server rather than any data centers included in the primary group or the secondary group.

We run the experiment three times, each time the average latency from each data centers to the client varies. About 200 seconds into each experiment, we collect sampling data so as to calculate the metric values of Consistency-ResponseTime efficiency provided by data centers. The experimental results are given in Fig. 5. Here, for the sake of brevity, only the maximum metric values in different cases are identified. As we can see, data center ***C*** provides the maximum value in case 1, which is about 2.51 (with the consistency probability of 0.719 and the response time 0.286s). Corresponding staleness observed subsequently is shown in Fig. 6. In case 1, some degree of staleness exists, because the secondary data center ***C*** is selected as the target. It also illustrate that, even if the only

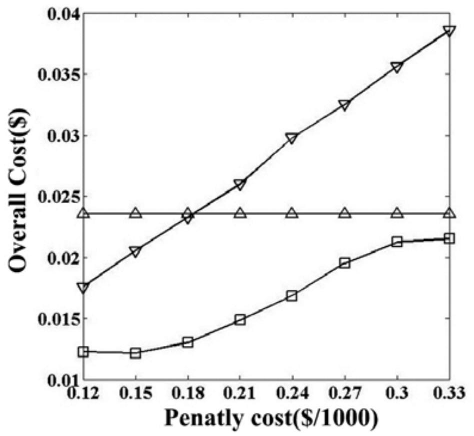

Figure 4. The overall cost when using strong consistency, weak consistency, tunable consistency, varying the penalty cost

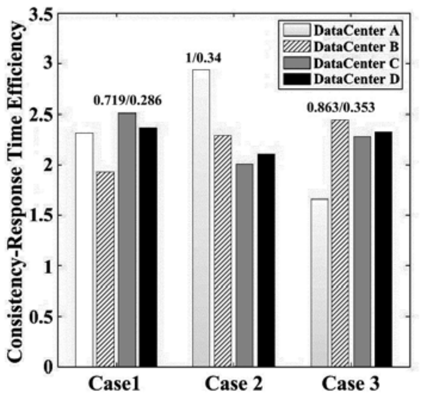

Figure 5. Under different network conditions, metric values of Consistency-ResponseTime efficiency provided by different data centers

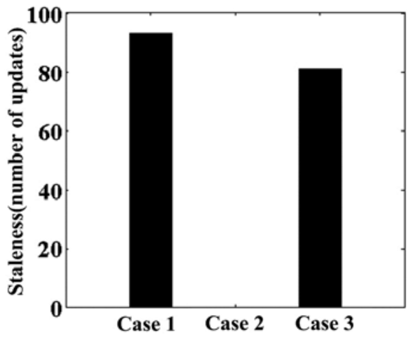

Figure 6. Under different network conditions, staleness observed in the subsequent period of time

primary data center is able to provide strong consistency guarantees, it isn't selected as the target because of the relatively lower metric value it provides due to longer response time. This is also true in case 3. As for case 2, the primary data center ***A*** is selected because of the maximum metric value it provides. Since data center ***A*** is a primary data center, the client is provided with strong consistency semantics. Accordingly, staleness 0 is shown in Fig. 6.

VII. Conclusion and Future Work

For tackling data consistency challenge in big data management based on geo-replicated cloud storage, we present a consistency hierarchical framework and a set of consistency trade-off strategies in this paper. The proposed framework provides support for the implementations of different consistency guarantees, including storing consistency and a range of tunable consistency semantics. A set of consistency trade-off strategies at both the data level and the transaction level is proposed and related protocol and formulas used in these strategies are also presented. The simulation results illustrate the effectiveness of our hierarchical framework and trade-off strategies.

For future work, our plan includes: investigating the dynamic reconfiguration of primary-secondary groups in our framework, exploring better predication models for our trade-off strategies, and having further analysis on the relationship between consistency, performance and cost involved in our framework.

References

[1] World's data will grow by 50X in next decade, IDC study predicts. Available: http://www.computerworld.com/s/article/9217988/World_s_data_will_grow_by_50X_in_next_decade_IDC_study_predicts

[2] A. V. S. S. R. Rao and R. D. Lakshmi, "A survey on challenges in integrating big data," in Proc. 2nd Intern. Conf. Intell. Comput. and Appl., Singapore, 2017, pp. 571-581.

[3] P. Gupta and N. Tyagi, "An approach towards big data — A review," in Proc. Intern. Conf. Comput., Commun. & Autom., Noida, India, 2015, pp. 118-123.

[4] G. Decandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, et al., "Dynamo: amazon's highly available key-value store," in Proc. ACM SIGOPS Symp. Oper. Syst. Princ., Stevenson, WA, 2007, pp. 205-220.

[5] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. J. Furman, et al., "Spanner: google's globally-distributed database, " in Proc. Usenix Conf. Oper. Syst. Des.and Implement., Hollywood, CA, USA, 2013, pp. 251-264.

[6] B. F. Cooper, R. Ramakrishnan, U. Srivastava, A. Silberstein, P. Bohannon, H. A. Jacobsen, et al., "PNUTS: yahoo!'s hosted data serving platform," in Proc. Vldb Endow., vol. 1, pp. 1277-1288, Aug. 2008.

[7] J. Baker, C. Bond, J. Corbett, J. J. Furman, A. Khorlin, J. Larson, et al., "Megastore: providing scalable, highly available storage for interactive services," in 5th Bienn. Conf. Innov. Data Syst. Res., Asilomar, CA, USA, 2011, pp. 223-234.

[8] L. Wu, L. Yuan, and J. You, "Survey of large-scale data management systems for big data applications," J. Comput. Sci. Technol., vol. 30, pp. 163-183, Jan. 2015.

[9] S. Gilbert and N. Lynch, "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services," ACM SIGACT News, vol. 33, pp. 51-59, Jun. 2002.

[10] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed, "Zookeeper: wait-free coordination for internet-scale systems," in Proc. the 2010 USENIX Conf. USENIX Annu. Tech. Conf., Boston, MA, USA, 2010, pp. 653-710.

[11] W. Vogels, "Eventually consistent," Commun. of the ACM, vol. 52, pp. 14-19, Oct. 2008.

[12] F. Liu and Y. Yang, "D-Paxos: building hierarchical replicated state machine for cloud environments," IEICE Trans. on Inf. & Syst., vol. E99.D, pp. 1485-1501, Jan. 2016.

[13] T. Kraska, M. Hentschel, G. Alonso, and D. Kossmann, "Consistency rationing in the cloud: Pay only when it matters," in Proc. Vldb Endow., vol. 2, pp. 253-264, Aug. 2009.

[14] I. Fetai and H. Schuldt, "Cost-based data consistency in a data-as-a-service cloud environment," in Proc. IEEE Fifth Intern.Conf. Cloud Comput., Honolulu, Hawaii, USA, 2012, pp. 526-533.

[15] R. Kotla, M. Balakrishnan, D. Terry, and M. K. Aguilera, "Transactions with consistency choices on geo-replicated cloud storage," 2013.

[16] D. B. Terry, V. Prabhakaran, R. Kotla, M. Balakrishnan, M. K. Aguilera, and H. Abu-Libdeh, "Consistency-based service level agreements for cloud storage," in Proc. 24th ACM Symp. Oper. Syst. Princ., Farminton, Pennsylvania, 2013, pp. 309-324.

[17] H. E. Chihoub, S. Ibrahim, G. Antoniu, and M. S. Perez, "Harmony: towards automated self-adaptive consistency in cloud storage," in Proc. IEEE Intern. Conf. CLUSTER Comput., Beijing, China, 2012, pp. 293-301.

[18] M. Mckenzie, H. Fan, and W. Golab, "Fine-tuning the consistency-latency trade-off in quorum-replicated distributed storage systems," in Proc. IEEE Intern. Conf. Big Data, Santa Clara, CA, USA, 2015, pp. 1708-1717.

[19] L. Lamport, "Paxos Made Simple," ACM SIGACT News, vol. 32, Jan. 2001.

[20] W. Lloyd, M. J. Freedman, M. Kaminsky, and D. G. Andersen, "Stronger semantics for low-latency geo-replicated storage," in Proc. 10th USENIX Conf. Netw. Syst. Des. and Implement., Lombard, IL, USA, 2013, pp. 313-328.

[21] C. Li, Leit, O. Jo, A. Clement, Pregui, A. Nuno, et al., "Automating the choice of consistency levels in replicated systems," in Proc. 2014 USENIX Conf. USENIX Annu. Tech. Conf., Philadelphia, PA, USA, 2014, pp. 281-292.

[22] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," ACM SIGOPS Oper. Syst. Rev., vol. 44, pp. 35-40, Apr. 2010.