2010 Ninth International Conference on Grid and Cloud Computing

An Application-Based Adaptive Replica Consistency for Cloud Storage

Ximei Wang
School of Computer Science and Technology, Univ. of Science and Technology of China
Hefei Anhui
wxm11008@mail.ustc.edu.cn

Shoubao Yang
School of Computer Science and Technology, Univ. of Science and Technology of China
Hefei Anhui
syang@ustc.edu.cn

Shuling Wang
School of Computer Science and Technology, Univ. of Science and Technology of China
Hefei Anhui
slwang@mail.ustc.edu.cn

Xianlong Niu
School of Computer Science and Technology, Univ. of Science and Technology of China
Hefei Anhui
nxllao@mail.ustc.edu.cn

Jing Xu
School of Computer Science and Technology, Univ. of Science and Technology of China
Hefei Anhui
jingxu@mail.ustc.edu.cn

***Abstract*—The intrinsic characteristic heterogeneous of cloud applications makes their consistency requirements different. Furthermore, the consistency requirement of certain application changes continuously at runtime, so a fixed consistency strategy is not enough. An application-based adaptive mechanism of replica consistency is proposed in this paper. We divide the consistency of applications into four categories according to their read frequencies and update frequencies, and then design corresponding consistency strategies. Applications select the most suitable strategy automatically at runtime to achieve a dynamic balance between consistency, availability, and performance. Evaluation results show that the proposed mechanism decreases the amount of operations significantly while guaranteeing the application's consistency requirement.**

***Keywords*—adaptive; replica consistency; update frequency; read frequency; cloud storage**

I. INTRODUCTION

As cloud computing is becoming increasingly popular, 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. As a key technology of cloud computing, replication faces new challenges, especially replica consistency.

The research of replica consistency devotes to solve the synchronization between two replicas or more. The most popular are strong consistency and eventual consistency[9]. Strong consistency ensures all of the replicas to be updated immediately. There is no difference between replicas. Therefore, every access to replicas will get fresh data. But the maintaining cost increases significantly, which cuts down the availability of replicas and system's performance. Eventual consistency doesn't update all the replicas immediately, so it tolerates replica divergence. But it promises all the replicas to be consistent at a specific time. For instance, Amazon's S3 service provides eventual consistency[9]. In this system, the transaction cost is low but it may results in high penalty caused by false operations. Therefore, there is a non-trivial trade-off between consistency, availability and performance.

Strong consistency and eventual consistency are both only suitable for particular scenes. In the environment of cloud computing, the customers of cloud storage is multifarious. Not all the applications need strong consistency or eventual consistency. In addition, the consistency requirement of an application is variable at runtime. Take auction system for an example, at the beginning, the data is not so important to the final deal, so the requirement of consistency is low. As the deadline approaching, the customers rely more and more on the latest data to make next bid. Eventually, the data should be always up-to-date and modified under strong consistency.

The strategies for replica consistency should be suitable for different applications. Therefore, adaptive replica consitency will be the best choice. Adaptive consistency is based on such a fact that if the requirement of application is satisfied, the consistency level can be relaxed aptly, and the update of replicas is allowed a little degree of delays. Adjusting of the replica consistency strategy dynamically according to the need of customers and application can satisfy the application requirements and minimize the transaction cost at the same time. The importance degree of a data can be mainly represented by read frequency, while the transaction cost is mainly represented by update frequency. Therefore, we take these two frequencies as the guideline of the application's consistency requirement. During the runtime of an application, these two metrics will change continuously, so consistency level will change accordingly, and the system will get a right balance between consistency, availablity and performance.

The remainder of this paper is organized as follows: Section 2 discusses related works. Section 3 describes the model structure and the adaptive consistency mechanism. Evaluations are given in section 4 and conclusion is in the final section.

II. RELEATED WORKS

Strong consistency is expensive not just in the transaction cost, but also in terms of replicas' availablity and system's performance. What's more, not all applications need strong consistency guarantees. However, eventual consistency may result in high penalty cost caused by false operations. Therefore, researchers pay attentions to the balance between consistency, availablity and performance.

Kraska proposes a strategy that system can switch the level of consistency between serializability and session consistency dynamically according to running condition in the cloud[2]. It

978-0-7695-4313-0/10 $26.00 © 2010 IEEE
DOI 10.1109/GCC.2010.16

divides the data into three categories, and treats each category differently depending on the consistency level provided. The consistency level will be changed accordingly while the data's impact changes continuously at runtime.

Ruay-Shiuang proposes an adaptive replica consistency service for data grid[3]. The strategy treats replicas differently according to the access frequency during the initializing process. The original replica and first level replicas can be updated immediately. If access frequency exceeds a predefined threshold, the second level replicas are updated immediately, too. If not, the replica is only updated when it is accessed. Dongmei Cao proposes an adaptive consistency model for grid according to access frequency[1]. Compared to [3], the most important improvement is allowing system to switch consistency level automatically at runtime.

HaiFeng Yu designs a continuous consistency model for replicated services[4]. It is based on the concept of consistency unit, and capturing the consistency spectrum using three metrics, numerical error, order error and staleness.

Ghalem compares pessimistic consistency with optimistic consistency, and combines these two existing approaches[8]. It divides replicas into several sites. Optimistic principals are used to ensure replica consistency within each site. Whereas, global consistency is covered by the application of algorithms inspired from the pessimistic approach.

Some of the above researches don't allow the system to change consistency level automatically at runtime, so they can't achieve the dynamic balance between consistency, availability and performance. Some partition the consistency level continuously, so the switch transaction cost is high. And in some works, the metric is selected unilaterally, so it can't be the very representative for an application. In order to avoid the above problems, the adaptive consistency mechanism proposed in this paper is based on read frequency and update frequency. System can select a suitable strategy dynamically according to these two metrics at runtime.

III. ADAPTIVE CONSISTENCY MECHANISM

A. Model Structure

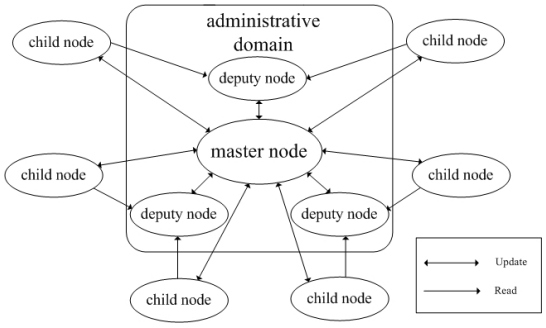

Figure 1. Model structure

The management of replicas in this paper adapts a centralized structure shown in Fig. 1. The administrative domain is the core of the whole structure, it includes one master node and three deputy nodes.

The master node is the most important part of administrative domain. It maintains the information of all nodes, including the geography position, update frequency and so on. Simultaneously, it is responsible for confirming all the update requests, and then excutes a corresponding operation according to the strategy of replica consistency.

To ensure the reliability and availability, deputy nodes are brought in as backups of the master node. The information they maintained are the same with the master node's. One of the deputy nodes will act as the new master node when the previous one breaks down.

Child nodes are the replicas holders distributing everywhere. Their object is to resolve problems such as network congestion and low availability. One of them will change status to deputy node when the number of deputy nodes is less than three.

During the initial phase of the system, the original replica node is selected as the master node. Deputy nodes are selected from child nodes according to their geography positions.

In this model, all nodes can receive updates from terminal customer, but only the master node can deal with these updates. If child nodes or deputy nodes receive update requests, they must send the update to the master node. When the master node accepts an update, it records update frequency of the original node where the update arised.

When a child node receives a read request from terminal customer, it records the read frequency, and then obtains the required data either from itself or the nearest deputy node as shown in Fig. 1.

The consistency requirements of various applications are different, and update frequency and read frequency are main metrics to reflect their characteristic. Recording the update frequency and read frequency aims to judges the replica consistency strategy the application needed.

B. Adaptive Consistency Mechanism

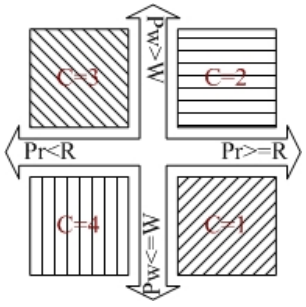

Figure 2. Categorizes of consistency mechanism

The adaptive consistency mechanism divides the consistency level into four categories according to statistical read frequency and update frequency. As shown in Fig. 2, for each checkpoint interval τ, the system once more judges the consistency category that the application needs according to statistical update frequency *Pw* and read frequency *Pr* in the interval. If the category is not same as the current one, it will be changed in the next interval. During the development process,

the first kind of strategy for replica consistency, namely strong consistency is provided to guarantee the requirement of the application. In the following we will discuss the four consistency strategies respectively. *C* indicates the kind of the strategy of replica consistency.

***C*=1:** read frequency is high and update frequency is low. The data should be treated with strong consistency. When read frequency is high, the data should update immediately in order to make the probability that customer read the up-to-date data as high as possible. When update frequency is low, transaction cost of pushing all the updates to all nodes is not too high. When the master node accepts the update, it pushes the update to all the other nodes immediately. When a node receives an access, it accepts the access immediately.

***C*=2:** read frequency is high and update frequency is high. Here, continual update will result in higher cost and lower performance, and the high read frequency makes that a larger percentage of access should read the latest data. So we should choose a trade-off between cost and consistency to maximize the benefits of both sides.

The data updates as follows: when the master node receives an update, it examines time T_{i-1} of the latest pushing, if (1) is satisfied, pushing the update to all the other nodes, otherwise the update is only propagated to deputy nodes.

$$T_i - T_{i-1} \geq k \qquad (1)$$

When a node receives an access, it accepts the access immediately. Although this strategy can't guarantee that all the access will read the latest data, it reduces the number of transactions for the high update frequency. So the system achieves a trade-off between consistency and performance.

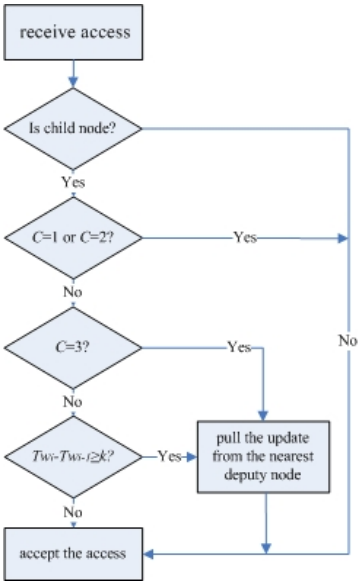

Figure 3. Access process

***C*=3:** read frequency is low and update frequency is high. Because of the low read frequency, continual updates immediately will waste bandwidth. On the other hand, frequent and immediate update will result in high cost. Consequently, the strategy here chooses a weaker consistency. When a child node is accessed, it will pull the latest data from the nearest deputy node. When the master node accepts an update request from terminal customer or other nodes, it only propagates the update to deputy nodes.

***C*=4:** read frequency is low and update frequency is low. The way to propagate update in this category is a combination of the second strategy and the third one: the update the master node accepted is only propagated to deputy nodes. when a child node receives an access, it examines its latest update time Tw_{i-1}, then pulls the update from the nearest deputy node firstly if (2) is satisfied:

$$Tw_i - Tw_{i-1} \geq k \qquad (2)$$

If not, it will accept the access directly.

The access and update processes of the whole system are shown in Fig. 3 and Fig. 4：

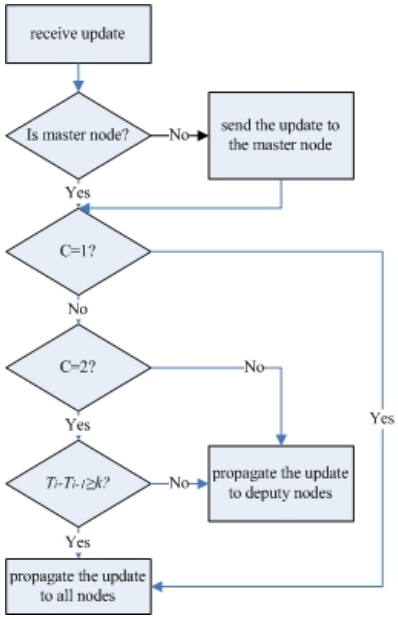

Figure 4. Update process

Based on the previous description, we can get the amount of operations in the system's process as (3) shows:

$$\begin{aligned} Nw(T) = & \underbrace{\frac{Pw_1}{\tau} \cdot P_1 \cdot T \cdot n}_{(i)} \\ & + \underbrace{\frac{Pw_2}{\tau} \cdot P_2 \cdot T \times 4 + \frac{\lfloor \tau / k \rfloor}{\tau} \cdot P_2 \cdot T \cdot (n-4)}_{(ii)} \\ & + \underbrace{\frac{Pw_3}{\tau} \cdot P_3 \cdot T \times 4 + P_3 \cdot T \cdot \frac{Pr_3}{\tau} \cdot \frac{n-4}{n}}_{(iii)} \\ & + \underbrace{\frac{Pw_4}{\tau} \cdot P_4 \cdot T \times 4 + P_4 \cdot T \cdot \frac{Pr_4}{\tau} \cdot \frac{n-4}{n} \cdot \frac{\lfloor \tau / k \rfloor}{\tau}}_{(iv)} \end{aligned} \qquad (3)$$

Here, $Pw_1 \le W$, $Pw_2 > W$, $Pw_3 > W$, $Pw_4 \le W$, $Pr_3 < R$, $Pr_4 < R$.

P_i indicates the percentage that the system implements the ith consistency strategy in the total time T. It relates to concrete applications. And P_i satisfies $\sum_{i=1}^{4} P_i = 1$, so the time that system implements the ith consistency strategy is $P_i \cdot T$. Pw_i indicates the update frequency in the interval τ when system implements the ith consistency strategy, and Pr_i indicates the read frequency accordingly.

The first part(i) of the equation calculates the total number of updates propagated to all nodes in the time that system implements the first consistency strategy. $\frac{Pw_2}{\tau} \cdot P_2 \cdot T \times 4$ in the second part(ii) of the equation indicates the number of updates propagated to master node and deputy nodes, and $\lfloor \tau / k \rfloor$ indicates the number of updates propagated to child nodes each interval τ, here, $(n-4)$ is the number of child nodes. $\frac{Pr_3}{\tau} \cdot \frac{n-4}{n}$ in the third part(iii) of the equation indicates the number of accesses received by all of the child nodes every second, therefore, $P_3 \cdot T \cdot \frac{Pr_3}{\tau} \cdot \frac{n-4}{n}$ indicates the number of updates accepted by all child nodes in the time that system implements the third consistency strategy. In the same way, $\frac{Pr_4}{\tau} \cdot \frac{n-4}{n}$ in the forth part(iv) of the equation indicates the number of accesses received by all of the child nodes every second, and $\frac{\lfloor \tau / k \rfloor}{\tau}$ indicates the average probability that a child node needs to pull an update when it receives an access.

The number of updates in the T time that system implements strong consistency strategy:

$$Nw'(T) = \frac{Pw}{\tau} \cdot T \cdot n \qquad (4)$$

Compared to the strategy of strong consistency, the number of updates when system implements the mechanism proposed in this paper decreases significantly, that is to say, the performance of the system improves significantly. In the meantime, because the operations on replicas are not frequent, the availablity of replica improves, too. The replica consistency strategy can be switched automatically according to the requirement of application at runtime, so the probability that false operation arises because of inconsistent is low. Therefore, system achieves a right balance between consistency, performance, and availability at low cost.

IV. EVALUATION

Experiments here are based on OptorSim[10] simulator. OptorSim is a package of simulation written in Java which is used to modeling the interactions of the individual components of distributed environment and its basic architecture is presented in Fig. 5. We modify it to satisfy our demand, and then compile our consistency mechanism on it.

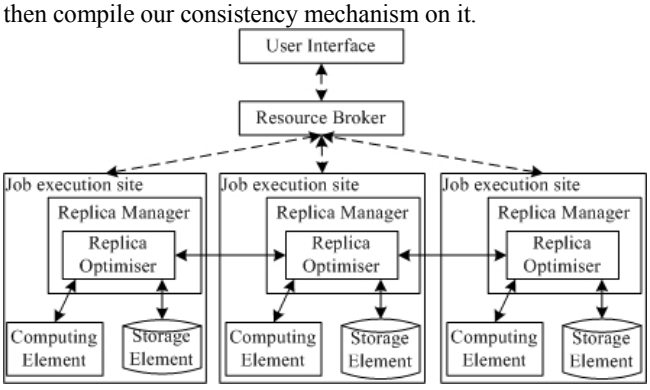

Figure 5. Basic architecture of OptorSim

One of the main efforts of this paper is to decrease the transaction cost while guaranteeing the consistency requirement of application. In order to study the consistency and transaction cost, we choose to compare our mechanism with strong consistency and eventual consistency. We take the percentage that the application accesses up-to-date data in time interval τ to be the representative of consistency requirement of an application, and the overall update amount to be the representative of transaction cost. Values of interval τ and threshold k relate to specific application and can be modified to get the most suitable ones. Since the jobs are simple, so the value of τ is 10 minutes, and the k is 2.5 minutes. The results of simulation are shown in Fig. 6 and 7.

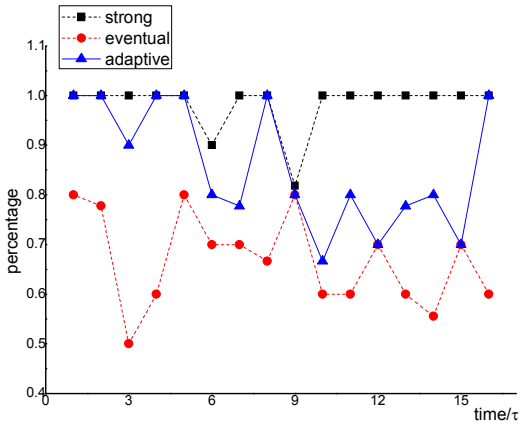

Figure 6. The percentage of accessing the up-to-date data

Fig. 6 shows the comparison of the percentage that read a latest data every interval τ between strong consitency, eventual consistency and our consistency mechanism. Strong consistency almost guarantees that every access read the latest data. Eventual consistency guarantees weaker consistency, so

the percentage is lower than strong consistency obviously. Our mechanism provides different consistency strategy at runtime, so the difference between our mechanism and strong consitency changes continually. When the first consistency strategy is provided, the diffence is small or even vanish. If the third consistency strategy is provided, the diffence between our mechanism and eventual consistency is small or even vanish. If the consistency strategy provided is the second one or the forth one, the percentage changes dramatically among strong consistency and eventual consistency.

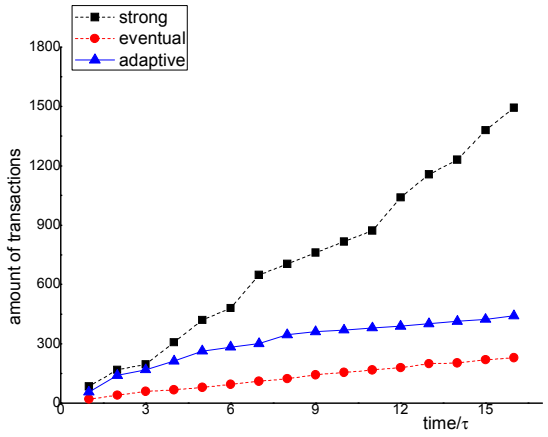

Figure 7. The amount of transactions

In Fig. 7, we compare the overall amount of update for strong consistency, eventual consistency and our mechanism. As runtime goes by, the amount of transactions of strong consistency increases quickly. For eventual consistency, the amount increases slowly. The amount of transactions of our mechanism exceeds the amount for eventual consistency slightly. Compared to strong consistency, the amount of transactions of our mechanism decreases significantly.

Because the consistency level the system provided relates to specific applications, the curve of our mechanism will vary evidently when the application changes. Every point of the curve is suitable for the application at runtime, while strong consistency and eventual consistency can't change their strategy when application needs.

Compared to strong consistency, the adaptive consistency mechanism proposed decreases transaction cost significantly while the needs of application for consistency are mainly satisfied. And our consistency mechanism guarantees higher consistency level than eventual consistency while the transaction cost slightly increases. Consequently, we get a better banlance between consistency, availability and performance.

V. CONCLUSION

In this paper we propose an application-based adaptive mechanism for replica consistency aiming to satify different requirement of applications in cloud computing storage services. The mechanism allows system to automatically switch consistency strategies according to update frequency and read frequency at runtime. The results of evaluation show that the adaptive mechanism proposed in this paper decreases the amount of operations significantly while mainly guaranteeing the application's requirement of consistency. Therefore, it is more suitable than existing consistency strategies for diversiform application environments such as cloud storage.

Future work will focus on the standard for adjusting parameters in our mechanism, and consider more factors for switching strategy.

ACKNOWLEDGEMENT

This work is supported by the National Natural Science Foundation of P. R. China (No. 60673172&6027304) and National 863 High Technology Research Program of P. R. China (No. 2006AA01A110).

REFERENCES

[1] Cao DongMei. The Research of Replica selection and consistency for Grid[D]. WuHan: Huazhong University of Science and Technology, 2007.

[2] Kraska, T, M. Hentschel, et al. Consistency Rationing in the Cloud: Pay only when it matters. Proceedings of the VLDB Endowment, 2009, 2(1): 253-264.

[3] Ruay-Shiung Chang, Jih-Sheng Chang. Adaptable Replica Consistency Service for Data Grid. Third International Conference on Information Technology: New Generations(ITNG'06). 2006.

[4] Haifeng, Y. and V. Amin. Design and Evaluation of a Conit-Based Continuous Consistency Model for Replicated Services. ACM Transactions on Computer Systems Vol. 20, No. 3, August 2002: 239–282.

[5] Haifeng, Y. and V. Amin. The costs and limits of availability for replicated services. ACM Trans. Comput. Syst. 2006, 24(1): 70-113.

[6] Susarla, S, J. Carter. Flexible consistency for wide area peer replication[A]. In: Proceedings - International Conference on Distributed Computing Systems[C], Columbus, OH, United states: Institute of Electrical and Electronics Engineers Inc, 2005: 199-208.

[7] M. Brantner, D. Florescu, D. A. Graf, et al. Building a database on S3. In Proc. of ACM SIGMOD, 2008: 251–264.

[8] Ghalem, B, S. Yahya. A Hybrid Approach for Consistency Management in Large Scale Systems. Proceedings of the International conference on Networking and Services, IEEE Computer Society.

[9] Werner Vogels. Eventually consistent. Amazon.com.2008

[10] OptorSim: A Replica Optimiser Simulation. http://edg-wp2.web.cern.ch/edg-wp2/optimization/optorsim.html.

[11] Zhou, X., X. Lu, et al. A dynamic distributed replica management mechanism based on accessing frequency detecting. SIGOPS Oper. Syst. Rev. 2004, 38(3): 26-34.

[12] Ghalem, B. and S. Yahya. Consistency Management for Data Grid in OptorSim Simulator. Proceedings of the 2007 International Conference on Multimedia and Ubiquitous Engineering, IEEE Computer Society. 2007.

[13] Abdelkrim Beloued, J.-M. G., Maria-Teresa Segarra, Francoise Andre. Dynamic Data Replication and Consistency in Mobile Environments. ACM International Conference Proceeding Series 114. 2009: 1-5.