A Scalable Replica Management Method in Peer-to-Peer Distributed Storage System

Jing Zhou, Yijie Wang, and Sikun Li

School of Computer Science, National University of Defense Technology
Room 307, National Key Laboratory for Parallel and Distributed Processing, NUDT
Changsha, China
jingle77@126.com

Abstract

Large numbers of replicas in peer-to-peer distributed storage systems deteriorate inconsistency and load imbalance. According to those data management problems, a scalable replicas management method based on decentralized and unstructured peer-to-peer network is proposed. Replicas are partitioned into different hierarchies and clusters according to single replica replication, and then replicas are coded and managed based on the user-defined hierarchy-coding rule. After that, replicas are organized with centralization in local and peer to peer in wide area, and the cost of reconciling consistency can be greatly depressed combining with defined propagation-time-plot. The simulation results show it is an effective multi-replica management method, achieving good scalability, and adapting well to applications with frequent updates.

1. Introduction

Peer-to-peer distributed storage systems have received much attention recently for sharing information among a large number of participants. As for it, data replication has been the subject of a great deal of recent interest [1,2,3]. More recently, the focus has shifted to sharing information in a large-scale decentralized setting.

Systems that address information sharing typically have to deal with information that changes over time. In a decentralized setting, this means dealing with consistency of mutable replicated data. Large numbers of replicas increase the update propagation delay and update conflicts, and sometimes create a load imbalance.

The solutions to manage large number of replicas can be divided into several types: client-server [4], hierarchy [5], peer-to-peer [6], and topology-based [2]. An unreliable multicast protocol [7] is employed to efficiently distribute updates in the general case. Additionally, optimistic consistency [8] relaxes the requirement of strong consistency, and improves performance and availability.

A distributed service faces two inherently conflicting challenges: high availability and strong data consistency. To address these challenges, a Scalable Replicas Management method in Peer-to-peer distributed storage systems (*SRMP*) is proposed, which is based on decentralized and unstructured peer-to-peer network architecture.

The remainder of this paper is organized as follows. Section 2 introduces the basic conceptions and definitions, and we describe our algorithm in detail in Section 3. We evaluate the design from several aspects experimentally in Section 4, and conclude in Section 5.

2. Basic conceptions and definitions

Large-scale distributed systems adjust placement of replicas according to dynamic external accesses or system requirement, to create new replica or to remove old one.

Definition 1. **Active Replica**: There are one or several copies of the same data that have uniform logical identifiers but are placed on different nodes. The copies are called replicas of that data. Replica that could be accessed at least by one user is an *active replica*. S_R denotes the replicas set including all the active replicas in system.

Now replication in peer-to-peer distributed storage systems is mostly based on single data replica.

Definition 2. **Replication Origin**: As for single-replica replication, the copy of some one replica r' is created and transferred to another node, and then a new replica r is created. Here, replica r' is the replication source of replica r, denoted as $O(r) = r'$.

3. Multi-replica clustering management

In *SRMP* method, replicas are partitioned into different hierarchies and clusters, and then replicas are coded and

International Conference on Management of Data
COMAD 2006, Delhi, India, December 14-16, 2006

managed based on the user-defined hierarchy-coding rule, which can also dispose the alteration of clusters caused by dynamic replica addition or replica removal. The cost of reconciling consistency can be greatly depressed combining with defined propagation-time-plot.

3.1 Replicas clustering based on replication origin

SRMP partitions replicas into clusters as followings.

- Replica and its replication origin are partitioned into the same cluster. The replication origin in each cluster is head, and others are general. Suppose r is the head of some cluster, and then it is denoted as $C(r)$.
- Suppose replica r_k is created by replicating replica r_j. If the cluster $C(r_j)$ exists, replica r_k joins it; otherwise one new cluster $C(r_j)$ is created.
- All the active replicas in system are partitioned into nested different clusters. As for the outer clusters, its nested cluster is regarded as a single active replica.
- The number of replicas in one cluster is finite. When the number excesses the threshold, the cluster overflows, and then it needs to be disposed carefully.

$H(r)$ or $G(r)$ represents that replica r is head or general. $C(r,r')$ represents that r' is in cluster $C(r)$. The number of general replicas in $C(r)$ is got from expression:

$$num_{GR}(C(r)) = \sum_{\forall r' \in S_R} (C(r,r') \wedge (r' \neq r)) \ ?\ 1\ :\ 0 \tag{1}$$

3.2 Hierarchy-coding rule

In *SRMP*, replicas are partitioned into clusters and clustering management model is established by encoding. Each replica is assigned a unique binary character string (identifier) when it joins system and the codes are reclaimed when replica failure or replica removal occurs.

The codes comprises of three segments: Replica-Level Codes (*RLC*), Replica-Sequence Codes (*RSC*), and Inheriting cluster-Head Codes (*IHC*). *RLC* shows the nested relation between replicas, *RSC* shows the sequence in its cluster and *IHC* inherits segmental codes from its replication origin and shows the inherited relation between replica and its replication origin.

We devise that replica identifier has fixed number of bits, *RLC* and *RSC* are of same length for any replicas, *IHC* adjusts length to local conditions, and the lack bits for outer replicas are all set as 0.

Suppose sequence codes and level codes are e and l bits respectively, the maximal number of replicas is:

$$sum = 2^e + 2^e \cdot 2^e + \cdots + 2^e \cdot \left(2^e \cdot (\cdots)\right) = \sum_{i=1}^{2^l} \left(2^e\right)^i \tag{2}$$

When $l = 2$ and $e = 4$, the number of replicas according to code rule approximates 2^{16}, e.g. 65,536, which can meet system scalability adequately.

AddReplica (Node N)
Select one best replica r_1 for node N from S_R;
A copy of r_1 is transferred to node N and new replica r is created;
For replica r_1
// Suppose *RHC* of r_1 are $h'_{k-1} \cdots h'_0$, where $k = 2^l$ and l is the length of *RHC*.
$j \leftarrow 0$;
For $i = 0$ **to** $k-1$ **If** $h'_i = 0$ **Then** $j \leftarrow 1$; **break**;
If $j = 0$
Then $h_{k-1} \cdots h_0 \leftarrow h'_{k-1} \cdots h'_0 - 1$;
If $C(r_2, r_1)$ **Then return** ($ClusterHeadIs(r_2)$);
// The head of the cluster that replica r will join is replica r_2.
Else $h_{k-1} \cdots h_0 \leftarrow h'_{k-1} \cdots h'_0$; **return** ($ClusterHeadIs(r_1)$);
For replica r
Get head r_{head} from returned message, and $r_{head} = r_2$ or $r_{head} = r_1$;
Replica r sends *joining* request to r_{head};
While $H(r_{head}) \wedge num_{GR}(C(r_{head})) = 2^e$ **Do**
$h_{k-1} \cdots h_0 \leftarrow h_{k-1} \cdots h_0 - 1$;
If $C(r'_{head}, r_{head})$ **Then** $ClusterOverflow(r_{head}, r'_{head}, r)$;
$r_{temp} \leftarrow r_{head}$; $r_{head} \leftarrow r'_{head}$;
For replica r_{head}
r_{head} assigns codes to r, and records the information about it; // See ①
If $(h_{k-1} \cdots h_0 = h'_{k-1} \cdots h'_0) \vee (h_{k-1} \cdots h_0 = h'_{k-1} \cdots h'_0 - 1)$
Then $G(r)$; // See ②
Else $H(r)$; $S_{temp} \leftarrow \varnothing$;
For $\forall r_x \left(r_x \in \{r_t \mid C(r_{temp}, r_t)\}\right)$ // Suppose *RSC* of r_x are $s_{e-1} \cdots s_0$.
If $\left(\sum_{i=0}^{i=e-1} 2^{s_i} > 2^{e-1}\right)$ **Then** $PartitionGeneralReplica(r, r_x)$;
$S_{temp} \leftarrow S_{temp} \cup \{r_x\}$; $NotifyNewHead(S_{temp}, r)$; // See ③④

Figure 1. Replicas addition algorithm—*A-SRMP*.

3.3 Replicas addition algorithm – *A-SRMP*

SRMP uses single-replica replication to create new replica. The algorithm *A-SRMP* of replica addition in *SRMP* is shown in Figure 1.

- The codes are invalid and reclaimed when replica removal or failure occurs. Therefore cluster-head replica searches all the codes in its own cluster to find the codes that have not been assigned.
- Replica r is general if it joins system by its true replication origin $O(r)$ or by the replica $r_1 \mid C(r_1, O(r))$; otherwise it is the cluster-head.
- $PartitionGeneralReplica(r, r_x)$: It is because that several clusters in existent management structure is full when r is added. If these clusters are not partitioned, all the following added replicas will repeat all the operations that r experiences.
- $NotifyNewHead(S_{temp}, r)$: Replica addition may bring on the alteration of cluster-head for some replicas, and the new cluster-head r employs a broadcast to inform the alteration to replicas in S_{temp}.

3.4 Replicas removal algorithm– *R-SRMP*

SRMP needs to make a difference between the removals of general replicas and cluster-heads. The algorithm *R-SRMP* of replica removal in *SRMP* is shown in Figure 2.

- Replica r_{n_h} is selected as the new cluster-head

RemovalReplica(Replica r)
If $\forall_{r' \in S_R}(\neg C(r,r'))$ // Replica r is not head for any replica r',
Then $G(r) \wedge \neg H(r)$; //and it is only general and it cannot be cluster-head.
If $\exists_{r_h \in S_R}(C(r_h,r))$ // r_h is the head of the cluster that replica r falls into.
Then Replica r_h deletes and reclaims the information of replica r;
Else $r'' = r$; $S_{temp} \leftarrow \{r_{temp} | r_{temp} \in S_R \wedge C(r'', r_{temp})\}$;
While $(S_{temp} \neq \varnothing)$ **do**
Select replica r_{n_h} from S_{temp} as new head; //See ①
If $\exists_{r_h' \in S_R}(C(r_h', r''))$ **Then** r_h records the information of r_{n_h}; //See ②
Replica r_{n_h} replicates management information from r'';
$r'' = r_{n\_h}$; $S'_{temp} \leftarrow \{r_t | r_t \in S_R \wedge C(r'', r_t)\}$;
r_{n_h} broadcasts the alteration of head to replicas in $S_{temp} - \{r_{new\_h}\}$;
$S_{temp} = S_{temp}'$; //See ③

Figure 2. Replica removal algorithm—*R-SRMP*.

from S_{temp} based on replica activity or replica physical location etc. We will not discuss it in detail here.

- When the removed replica r is a cluster-head, its cluster-head is just to modify information (e.g. the physical location) corresponding to the codes of replica r_{n_h} but not to add new information.
- The replicas in $S_{temp} - \{r_{n\_h}\}$ need not modify their own codes when they receive the broadcast from replica r_{n_h}, and they are only be informed that the physical location of cluster-head is altered.

3.5 Consistency based on *propagation-time-slot*

We introduce the idea of *Propagation-Time-Slot* (PTS) to optimize update commitment and sorting. For any active replicas r_1 and r_2, PTS is the minimal time to propagate an update issued by replica r_1 to replica r_2 in an idealized network environment without congestion or losing messages.

The reconciling course in *SRMP* is as follows, which is shown in Figure 3.

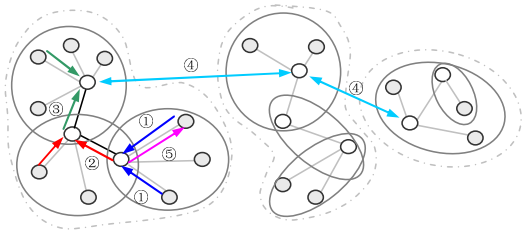

Figure 3. An illustration of the reconciling course.

- The updates issued by replica r are always propagated to its cluster-head replica r'.
- Replica r' merges all the updates into a new update during PTS, and then sends it to its cluster-head r''.
- The rest may be deduced by analogy until r'' is the outermost, that is, replica-level codes of r'' are $00\cdots0$.
- The peer to peer replicas in the outermost cluster use *ack vectors* proposed by Golding [9] or *casual history* method [3] to order updates, and distribute updates by *anti-entropy* [10].
- Replica r gets all the unseen updates stored on r' when it propagates updates to r'. Additionally, once replica r' receives new updates from r'', it propagates them to r initiatively.

Introducing PTS has the following advantages: It can earliest detect conflicting updates for local replicas to reduce update-conflict-rate. Merging multi un-conflicting updates cannot destroy consistency, but can reduce the number of delivered updates, and then depress the cost of keeping consistency.

4. Simulation

OptorSim [11] was developed to evaluate the effectiveness of replica optimization algorithms. In our simulations, we utilize OptorSim to construct a distributed environment with 1,000 sites. Each data object is a 10GBytes file, and the size of data object set is 150GBytes. Initially, each file only has a master replica and is distributed over the sites random. We use a replication strategy with replica addition according to frequency of access and least accessed replica deleted.

There are five read access types and five write access types. The size of file accessed by each read access and the probability (Prob.) of each are summarized in Table 1 (a), and that of each write access are shown in Table 1 (b).

Table 1. The schedules of read and write accesses.

(a)

Type	Size(GB)	Prob.
R_1	10	20%
R_2	20	20%
R_3	30	35%
R_4	40	20%
R_5	50	5%

(b)

Type	Size(MB)	Prob.
W_1	1	20%
W_2	10	25%
W_3	100	30%
W_4	1000	22%
W_5	10,000	3%

(1) Setting Propagation-time-slot

The propagation-time-slot is the basis of simulation, which should be set first. We simulate the alteration of PTS in different network bandwidth configuration, as is shown in Figure 4.

For different network configuration, PTS is not under 0.1 second in idealized state. However, in practice, the time is excessive longer than that due to network latency. So in the following simulations, we set that PTS is 0.1s.

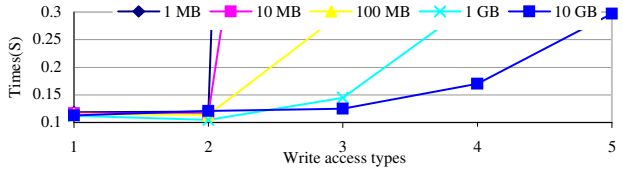

Figure 4. The alteration of PTS in different network bandwidth configuration.

(2) Effects of Number of Replicas

We configure several size of storage space of each site, and then submitting 10,000 external read accesses. Suppose there are no occurrences with node failures or network partitions during simulator running. Replicas of each file is partitioned into clusters and coded according to the code rule with $l=2$、 $e=4$. After the end of run,

the distributions of maximal number and minimal number of replicas are shown in Figure 5.

Because of our employing replication strategy, the results are influenced by storage capacity. We just select extremes of numbers of replicas for all the files in file set.

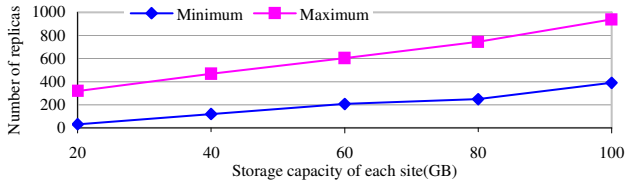

Figure 5. The distribution of the number of replicas of one file.

The average propagation time of each update is shown in Figure 6. We compare our algorithm with others, such as replica autonomy (Uniform) and policies organizing replicas into a regular graph (Ring, Tree, and Mesh). The simulator is run until 5,000 updates had been processed. For compared policies, two replicas exchange updates in an *anti-entropy* session, and the *anti-entropy* rate is half of the update rate.

The results show that consistency depends on the number of replicas. For small numbers of replicas, the Uniform policy performs quite well. For large numbers of replicas, our method has an obvious advantage and there are slight variations when the number is large to a certain extent. The policies that simulate a fixed topology have the worst performance and scaling.

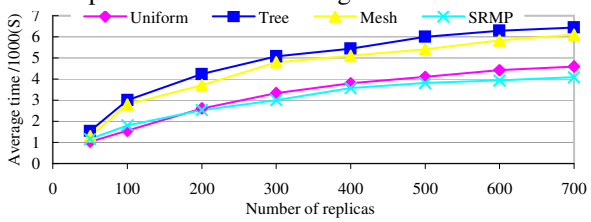

Figure 6. Effects of the number of replicas on the performance.

(3) Effects of the Update Rate

Figure 7 shows the effect of varying the update rate on different method. Suppose there are 500 replicas and the *anti-entropy* rate is 1.

The update rate is higher, the number of update exchanged in each session is more, so the time to achieve consistent is shorter. *SRMP* is more predominant when the update rate is higher because there are few replicas (clusters) to exchange updates by sessions in a broad range after clustering and our method merges updates during PTS .

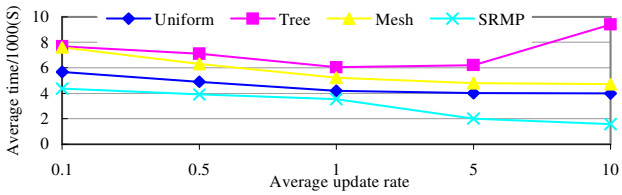

Figure 7. Effects of the update rate on the performance.

5. Conclusions

SRMP replica management method is proposed to the question of complicated resource management brought by large numbers of replicas in a large-scale peer-to-peer distributed environment. It manages replicas by clustering based on the user-defined hierarchy-coding and takes dynamic replica addition, replica removal, and consistency into account adequately.

The simulation results also show that *SRMP* is an efficient way to manage a large number of replicas, achieving good scalability, and adapting well to applications with frequent updates.

6. References

1. M. Dahlin, L. Gao, A. Nayate, A. Venkataramani, P. Yalagandula, and J. Zheng. PRACTI Replication for Large-scale Systems. Technical Report, TR-04-28, Austin: University of Texas at Austin, 2004.
2. Yasushi Saito, Christos Karamanoli, Magnus Karlsson and Mallik Mahalingam. Taming Aggressive Replication in the Pangaea Wide-area File System. In: Proceedings of the fifth symposium on Operating systems design and implementation. New York: ACM Press, 2002. 15-30.
3. Brent ByungHoon Kang, S2D2: A framework for scalable and secure optimistic replication [Ph.D. Thesis]. Berkeley: University of California, 2004.
4. J. Kistler and M. Satyanarayanan. Disconnected operation in the Coda file system. ACM Transaction on Computer Systems, February 1992, 10(1): 3–25.
5. J Kubiatowicz, D Bindel, Y Chen, et al. OceanStore: an architecture for global-scale persistent storage. ACM SIGARCH Computer Architecture News, 2000, 28(5): 190~201.
6. Clarke, I., Sandberg, O., Wiley, B. and Hong, T.W. Freenet: a distributed anonymous information storage and retrieval system. In ICSI Workshop on Design Issues in Anonymity and Unobservability, Berkeley, California, 2000. 25-31.
7. R. A. Golding and Darrell D. E. Long. Design choices for weak-consistency group communication. Technical report, UCSC–CRL–92–45, Santa Cruz: University of California, September 1992.
8. J. Gray, P. Helland, P. E. O'Neil, and D. Shasha. The dangers of replication and a solution. In: Proceedings of ACM SIGMOD International Conference on Management of Data. Montreal, Canada, June 1996. 173-182.
9. R. A. Golding. Modeling replica divergence in a weak-consistency protocol for global-scale distributed data bases. Technical Report, UCSC-CRL-93-09, Santa Cruz: University of California, February 1993.
10. K. Petersen, M. J. Spreitzer, D. B. Terry. Flexible update propagation for weakly consistent replication. In: 16th ACM Symposium on Operating Systems Principles. New York: ACM Press, October 1997. 288~301.
11. W. H. Bell, D. G. Cameron, L. Capozza, P. Millar, K. Stockinger, and F. Zini. Optorsim: a grid simulator for studying dynamic data replication strategies. International Journal of High Performance Computing Applications, 2003, 17(4): 403-416.