A Replication and Cache based Distributed Metadata Management System for Data Grid

Ke Shi
Services Computing Technology and System Lab.
Cluster and Grid Computing Lab.
School of Computer Science and Technology
Huazhong University of Science and Technology
keshi@mail.hust.edu.cn

Abstract

Metadata management is a key technique of data grid. This paper presents a replication and cache based metadata management system (RCMMS) to improve metadata management of Global Distributed Storage System (GDSS). Storage Service Provider (SSP) of GDSS has dedicated buffer to cache metadata. The design, implementation and evaluation of RCMMS are discussed in this paper. RCMMS provides efficient algorithms to manage highly dynamic metadata replicas. The evaluation demonstrates that replica plus cache metadata management system outperforms existing metadata management system of GDSS.

1. Introduction

Metadata is information that describes data. In data grids ([1], [2], [3], and [4]) metadata catalog stores all kinds of metadata in uniform structure. Whatever structure is used, metadata catalog should meet two following requirements. One is that it should be hierarchical and distributed such as LDAP ([5] and [6]) (Lightweight Directory Access Protocol). The other is that it should be compatible and interactive with existing metadata system. This paper presents a replication and cache based distributed metadata management system (RCMMS) based on the former study on data replication ([7], [8], and [9]). How to describe metadata is not in the discussing range of this paper. The system stores metadata and its replicas in LDAP servers and caches these metadata in the buffer of SSP [10]. RCMMS's replica management must achieve two goals. First, it must support multiple metadata replicas to maximize availability. Second, it needs to support dynamic addition and removal of metadata replicas even when some nodes are not available. RCMMS addresses these challenges by maintaining a sparse, yet strongly connected and randomized graph to describe the structure and relations of metadata replicas. The graph is used both to propagate updates and to discover other replicas during replica addition and removal.

Metadata servers are organized in a hierarchical structure, and the metadata servers at the same level operate in peer-to-peer mode. There is no single "master" metadata replica. Any metadata replica may be read or written at any time, and metadata replicas exchange updates among them according to the graph. Adaptive replication achieves high performance by choosing a server as close as possible to the accessing point to provide metadata service. On the base of metadata replication, cache improves metadata accessing performance further.

2. Background and related work

With the very fast development of Data grid technologies metadata management becomes more and more important. SRB [12] of American San Diego Supercomputing Center (SDSC) is one of widespread used data grid software. Current version of SRB adopts federated MCAT to provide better performance. Griddaen [13] is a data grid system designed by National University of Defense Technology in China. Metadata managing system consists of a central server and multiple local servers. Local servers are responsible for local resources and metadata management, and the central metadata server builds index of local servers and data cache. Both SRB and Griddaen store metadata in relational databases.

GDSS, as shown in Fig.1, is a storage virtualization project implementing basic functions of data grids. The

0-7695-2909-7/07 $25.00 © 2007 IEEE
DOI 10.1109/SNPD.2007.204

GDSS introduces a new component, called Storage Service Point (SSP), to play as an entry point for users to access the storage system. There are many independent and decentralized SSPs in the domains of GDSS. SSPs can join the system dynamically to meet the requirement of extensibility.

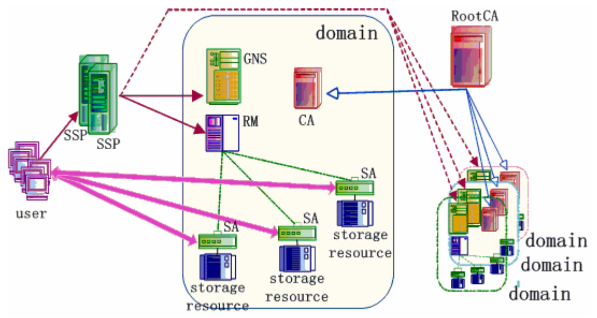

Fig.1. GDSS architecture

GNS (Global Name Server) works as a metadata server, responsible for the management of metadata. Information about users and data access control are kept in GNS. RM (Resource Manager) administrates resources requirement and scheduling. Additionally, it implements transparent duplicating strategies, which helps to achieve high reliability and load-balance. CA (Certification and Authentication Server) takes charge of certificate management and digital signature, ensuring the security of the system. SA (Storage Agent) screens heterogeneity of underlying storage resources and provides the uniform data access interfaces to users. Moreover, SA can select data transmission mode automatically, in order to achieve efficient data transmission.

In the projects mentioned above, if the metadata catalog fails during an operation, metadata service will be terminated since it does not have replicas. In RCMMS, metadata catalog is replicated to achieve high availability and avoid single points of failure. Hot metadata is cached in the buffer located at the SSPs to accelerate data discovering process.

3. RCMMS: a structural overview

RCMMS has a global metadata server and multiple local metadata servers like Griddaen. In Griddaen every domain has only one local metadata server, but in RCMMS every domain has multiple local metadata servers which communicate in hierarchical or peer-to-peer mode. Currently, our replica catalog is implemented as an LDAP directory.

RCMMS replicates metadata at the granularity of partition. Partition is a sub-tree of LDAP server's directory tree. As showed in Fig.2, a node represents a replica of a partition. Directory tree is partitioned into A, B, C, and D four partitions. A strongly connected graph is used to represent the structure and relations of the replicas. An edge represents a known connection between two replicas of a partition; partition updates propagate along the edges. The replicas of a partition and the edges between them form a strongly connected graph.

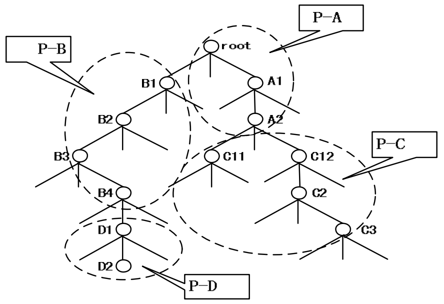

Fig.2. Directory tree and partition

RCMMS distinguishes two types of replicas: master and slave. They can both be read and written by users at any time, and they both run an identical update-propagation process. Master replicas, however, play an additional role in maintaining the hierarchical name space. First, master replicas act as starting points from which slave replicas are found during path-name traversal. Second, master replicas perform several tasks that are hard to do in a completely distributed way. In particular, they are used as pivots to keep the graph connected after a permanent node failure. Currently, in RCMMS, replicas created during initial partition construction are designated as masters and their locations are fixed unless some of them fail permanently.

RCMMS adopts decentralizing management for replica set by maintaining a strongly connected graph of replicas for each partition. The structure of this system consists of four parts as shown in Fig.3. Global Metadata Server (GMS) maintains index information of landmark metadata servers. Landmark Metadata Servers (LMS) is a kind of local metadata server hosting master replicas. Every domain has only one LMS. Replication Metadata Servers (RMS) is the other kind of local metadata server that store slave metadata replicas. Generally, every domain has multiple RMSs. SSP Cache (SC) saves hot metadata. It can improve read efficiency of metadata and occupies less storage space.

GMSs create and maintain index information of LMSs. The index information table comprises two columns, of which the first is the domain name of LMS and the second is IP address of LMS. For example, when LMS1 does not satisfy a request from an end user, LMS1 will forward it to GMS. GMS searches its index information table, finds a right LMS (say LMS3) according to domain name (domain3) and forwards the request to the LMS3. LMS3 firstly checks whether relevant metadata replicas exist, or LMS3 directly satisfies the request. If metadata replicas exist, LMS3 forwards the request to the corresponding RMS (say RMS1) which is closest to the user. The request of the end user is satisfied by RMS1 of domain3.

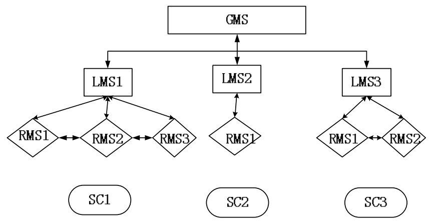

Fig.3. RCMMS architecture

The RCMMS local metadata server consists of three main modules. SSP-RCMMS protocol handler receives requests from SSP, updates local metadata replicas, and generates requests for the replication engine.

Replication engine accepts requests from the SSP-RCMMS protocol handler and other replication engine of other nodes. It creates, modifies, or removes replicas, and forwards requests to other nodes if necessary. It is the key part of the local metadata server.

Membership module maintains the status of other nodes, including their liveness, available disk space and the locations of replicas. Each node periodically sends its knowledge of nodes' status to a random node chosen from its live-node list; the recipient will merge received information with its own into a list. A stable node is designated as "landmarks" and it bootstraps newly joining nodes. The protocol can disseminate membership information quickly. A newly booted node obtains the member information from a landmark.

4. Metadata Replica Management

In RCMMS, initial metadata is created as master replica on LMS. The metadata replica is created as slave replicas on RMS through replication creating algorithm, and it is removed when a node runs out of disk space. Because these operations are frequent, they must be carried out efficiently in non-blocking mode, even when some nodes that store replicas are unavailable. Fig.4 shows the key attributes of a replica. The timestamp (*ts*) and the version vector (*vv*) record the last time the replica was modified. MasterPeers are uni-directional links to the master replicas of the partition. Peers point to the neighboring slave replicas in the partition's graph.

```
struct Replica
pid: PartitionID
ts: TimeStamp // Pair of ←physical clock, IP addr⟩.

vv: VersionVector // Maps IP addri. ↔←TimeStamp
masterPeers: Set←⇐NodeID)// Set of IP addresses

peers: Set←⇐NodeID)

←. . .
```

Fig.4. Key attributes of a replica

4.1. Metadata replica addition

The creating algorithm of metadata replicas includes how to select metadata replicas, when to create metadata replicas and where to place metadata replicas. This paper introduces a partition table to decide which partition to be replicated, uses PQ parameter principle [11] to trigger creating replicas, and place replicas on the metadata servers as close as possible.

First, we discuss how to select partition to be replicated. The algorithm dynamically partitions a directory tree into multiple partitions according to entity number. The partition information index table, including local partition root node, replication partition root node, partition size, partition response time and request numbers from SSP is built for each partition. Partition response time T_{resp} consists of partition service time T_s, waiting time T_w, communication time T_c, and $T_{resp} = T_s + T_w + T_c$. Threshold is response time threshold. When $T_{resp} > T_{threshold}$, this partition is to be replicated.

Second, PQ (P < Q) parameter principle is used to trigger replicas creating thread. In P period, the system may timely response to acutely increasing accessing requests from SSPs by creating new metadata replicas. In Q period, a more cautious creating policy is adopted to prevent the system from creating too many metadata replicas. The specific process is described as the following:

(1) $T_{resp} < T_{threshold}$, give up metadata replica creating, or else jump to (2);

(2)In P period, if $\forall \varepsilon t \in P$, $T_{resp} > T_{threshold}$ and $\frac{dT_{resp}}{dt} > 0$, then jump to (4), otherwise jump to (3);

(3) In Q period, if $\forall \varepsilon t \in Q$, $T_{resp} > T_{threshold}$, then jump to (4), otherwise give up creating a metadata replica;

(4) Triggering the thread and creating a slave replica.

Once creating time of replicas is decided, LMS selects a RMS, where the replica does not exist, to place the replica according to accessing mode. There are three kinds of accessing modes including central, uniform and random mode and two placement policies. Fastspreading policy is to create replicas on all metadata servers along the accessing path, cascading policy is to create replicas on the next level of metadata server along the accessing path and spread along the hierarchy structure.

LMS always tries to transfer data from a nearest existing replica when creating a new replica. To create a replica of partition P on RMS1, LMS first finds a replica closest to among its replica graph neighbors (say RMS2, which may be LMS itself) and forwards the request to RMS2, which in turn sends the partition to the RMS1. At this point, RMS1 will let the users start accessing the replica. The new replica must be integrated into the partition's replica graph to be able to propagate updates to and receive updates from other replicas. Thus, in the background, RMS1 chooses x existing replicas of P, adds edges to them, and requests them to add edges to the new replica in RMS1. The selection of x peers must satisfy three requirements: 1) including master replicas so that they have more choices during future replica creation, 2) including nearest replicas so that updates can flow through fast network links, 3) sufficient randomization to make the probability that the crash of nodes will catastrophically disconnect the partition's graph as possible as low.

As the graph of metadata replicas is strongly connected, RCMMS satisfies all these requirements simultaneously. RMS1 chooses three types of peers for the new replica. First, RMS1 adds an edge to the master replica on LMS1. Second, it asks the master replica to pick the replica closest to RMS1. Third, RMS1 asks LMS1 to choose $x-2$ random replicas using random walks that start from LMS1. Parameter x trades off availability and performance. A small value increases the probability of graph disconnection (i.e., the probability that a replica cannot exchange updates with other replicas) after node failures. A large value for x increases the overhead of graph maintenance and update propagation by causing duplicate update delivery.

4.2. Metadata replica removal

This section describes the protocol for removing slave replicas. Master replicas are removed only as a side effect of a permanent node loss. A slave replica is removed for two possible reasons: a node has run out of disk space, or the cost of keeping the replica outweighs its benefits. To reclaim disk space, RCMMS examines 50 random replicas kept in the node and considers their accessing numbers in a certain time interval. The replica with the minimum accessing number is evicted, and three replicas with the next-worst accessing number values are added to the candidates examined during the next round. The algorithm is repeated until it frees enough space on the disk.

To remove a replica, the server sends notices to the replica's graph neighbors. Each neighbor, in turn, initiates a random walk starting from a replica and establishes a replacement edge with another randomly live replica. Starting the walk from a live master replica ensures that the graph remains strongly connected.

5. Metadata Cache Management

To improve read and write efficiency of metadata, cache technology is adopted in the RCMMS model. Cache module is placed on the SSPs, which saves hot or being written metadata. According to accessing locality principle, hot metadata is copied from metadata servers to the buffer of SSPs. Because a small amount of metadata is kept in the buffer, cache does not consume too much memory space. Especially, metadata cache is different from metadata replicas and not registered in replication information table. Replicas keep complete information of metadata, but cache may only keep part of metadata.

The working process of metadata cache is: SSP receives an end user's read/write requests and check its own cache. If cache hits, it reads/writes cache directly, which can assure metadata returned to user is the latest. If cache does not hit, read requests are forwarded to the closest metadata servers and write requests are recorded in cache until they are updated to metadata servers at the same time. It greatly decreases network communication loads.

Metadata cache is organized by a hash table, named HashMap. Each cached object is denoted by "Attributes" and added to the HashMap. There are two kinds of metadata in cache, of which one is reading object and the other is writing object. Reading objects are organized by single linklist "R_lastAccessedList", and are sorted by the last accessing time; and the writing objects are organized by single linklist "W_ageList", and are sorted by the creating time. If one cache object has been accessed, it is added to the rear of linklist. If cache is full, cached objects in the front of "R_ lastAccessedList" will be deleted to release space. When a writing object is added to cache, it is added to the rear of "W_ageList". Once write objects have been updated to metadata servers, they are deleted from "W_ageList".

6. System Evaluations

This section evaluates system performance of RCMMS. We investigate the baseline performance and overheads of RCMMS and show that it performs competitively with GNS and is fit for data grids.

6.1. Experiment Settings

The RCMMS system is tested on a campus-wide environment. The nodes are distributed in three locations. The configuration of each node is Xeno 1G CPU, 1GB memory, 80G disk and all nodes connected with 100M Ethernet. During the testing period, these nodes implement routine work such as FTP, WEB servers and development.

6.2. Performance Evaluations

The experiment results include metadata replica storage space consumption, response time and update time under different policies and client accessing modes. Response time is the interval from SSP sending a metadata request to receiving results. All experiment results are average values. Through the comparisons, our system adopts an optimum combination of policies to manage metadata.

The following experiment has no SSP cache. When accessing mode is central, SSP frequently accesses RMS3, and metadata replicas are created according to two different policies as showed in Fig.5. Fastspreading creates replicas on RMS1 and RMS3 along the accessing path of partition A; cascading creates replicas only on the next level server RMS1.

If client accessing mode is the central mode, response time of fastspreading is 18% faster than cascading. Storage space consumption and replica update time of the former are almost as two times as the latter.If client accessing mode is the uniform node, SSP accesses partition A uniformly distributed on RMS3 and RMS4, metadata replicas are created according to two different policies as showed in Fig.6. Fastspreading creates replicas on RMS1, RMS3 and RMS4 along the accessing path of partition A, cascading creates replicas only on the next level server RMS1.

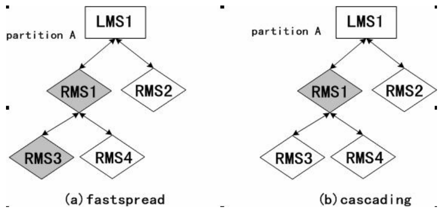

Fig.5. Two policies comparison in the central mode

Table 1. Performance metrics comparison in the central mode

Policy	Storage Space (M)	Response Time (ms)	Replica Creation Time (min)	Replicas Update Time(ms)
fastspreading	4.36	10.36	6	40.38
cascading	2.18	12.58	6	20.22

If client accessing mode is the random mode, SSP randomly accesses partition A, metadata replicas are created according to two different policies as showed in Fig.7. Fastspreading creates replicas on RMS1, RMS2, RMS3 and RMS4 along the accessing path of partition A, cascading creates replicas only on the next level servers RMS1 and RMS2.

In the random mode, response time of fastspreading is more fast 8% than one of cascading. Storage space consumption and replica update time of the former are almost as two times as ones of the latter.

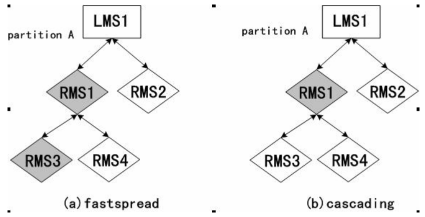

Fig.6. Two policies comparison in the uniform mode

Table 2. Performance comparison in the uniform mode

Policy	Storage Space (M)	Response Time (ms)	Replica Creation Time (min)	Replica Update Time(ms)
fastspreading	6.54	10.78	6	60.67
cascading	2.18	13.67	6	20.21

In three situations, the response time of fastspreading is faster than cascading but the former is at the cost of storage space consumption and replica update time. Therefore, when the accessing mode is central, RCMMS adopts fastspreading policy. If the accessing mode is uniform or random, RCMMS adopts cascading policy. With such a combination of policies, we compare RCMMS performance with GNS (showed as Fig.8). From Fig.8, we can draw a conclusion that response time improvement of RCMMS is remarkable at the cost of storage space consumption and replica update time. Total performance RCMMS is better than GNS. Moreover, when SC is used, the RCMMS system performance is improved about 20%, as shown in Fig.9.

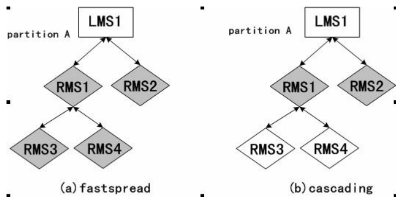

Fig.7. Two Policies Comparison in the Random Mode

Table 3. Performance comparison in the random mode

Policy	Storage Space (M)	Response Time (ms)	Replica Creation Time (min)	Replica Update Time (ms)
fastspreading	8.72	10.36	6	80.91
cascading	4.36	11.28	6	40.45

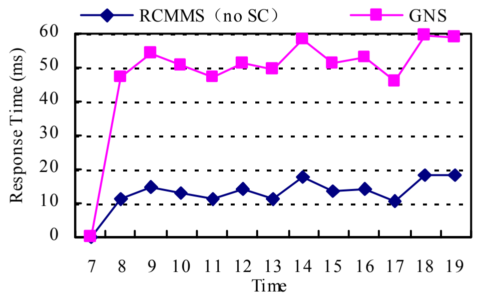

Fig.8. Response time comparison between RCMMS (no SC) and GNS Performance

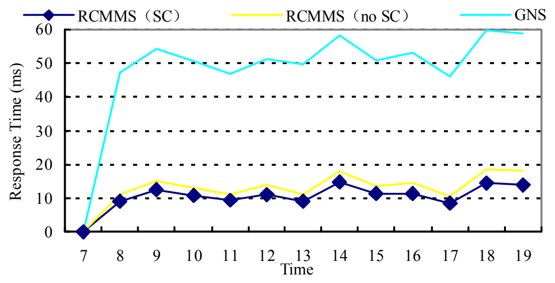

Fig.9. Response Time Comparison between RCMM S (SC) and GNS Performance

In RCMMS, metadata servers are organized in multiple domains, which are similar as virtual organizations of data grids. For data grids, every virtual organization can has a LMS and multiple RMSs. The structure of RCMMS can easily be transplanted to data grids and provide good performance and availability.

7. Conclusions and Future Work

RCMMS is a wide-area metadata management system that targets better performance for metadata accessing. The evaluation of RCMMS shows that RCMMS is faster and more efficient than GNS.

Based on the lessons learned from this design, we have undertaken a reevaluation and redesign of RCMMS. We plan to make the next implementation of RCMMS on OGSA-compliant Grid service. How to maintain consistency of metadata replicas and cache also needs further attention in RCMMS.

8. Reference

[1] Foster, I. The Grid: A New Infrastructure for 21st Century Science, Physics Today, 54 (2), 2002.
[2] A L Chervenak et al. Data management and transfer in high performance computational grid environments. Parallel Computing Journal, 28(5), 2002.
[3] Krzysztof Kaczmarski et al. Metadata in a Data Grid Construction. In Proc. of WETICE'04
[4] Srikumar Venugopal et al. A Taxonomy of Data Grids for Distributed Data Sharing, Management and Processing. ACM Computing Surveys, 38(1), 2006.
[5] Gregor von Laszewski and Ian Foster, Usage of LDAP in Globus. http://www.globus.org.
[6] Koutsonikola, et al. LDAP: framework, practices, and trends. IEEE Internet Computing, 8(5), 2004.
[7] Houda Lamehamedi. Data replication strategies in grid environments, In Proc. of CCGrid, 2002.
[8] Elizeu Santos-Neto, et al. Exploiting Replication and Data Reuse to Efficiently Schedule Data-intensive Applications on Grids. In Proc. of the 10th Workshop on Job Scheduling Strategies for Parallel Processing, 2004.
[9] Magnus Karlsson, et al. Choosing Replica Placement Heuristics for Wide-Area Systems. HP Laboratories, 2004.
[10] Chen Huang, et al. Architecture Design of Global Distributed Storage System for Data Grid. In Proc. of ICYCS 2003.
[11] Byoungdai Lee and Jon B. Weissman. An Adaptive Service Grid Architecture Using Dynamic Replica Management, GRID, 2001.
[12] Michael Wan, Arcot Rajasekar, Wayne Schroeder .An Overview of the SRB 3.0: the Federated MCAT.2003,9.
[13]LI Dong-Sheng, et al. Dynamic Self-Adaptive Replica Location Method in Data Grids. Journal of Computer Research and Development, 2003.