GROUP COMMUNICATION SYSTEM SPECIFICATION AND DESIGN FOR NON-REPLICATED SERVICE

Ruiyong Jia, Yanyuan Zhang, Yong Feng
School of Computer Science
Northwestern Polytechnical University
Mailbox 755, No.127 West Youyi Road, Xi' An City (710072), P.R.China
E-mail: {jiary,zhangyy,fengyong}@co-think.com

KEYWORDS
Group Communication System, Storage Area Network, Non-Replicated Service

ABSTRACT

With the advent and popularization of high-speed storage area network, non-replicated services are applied in more and more high performance distributed storage systems. In this paper, we define the specification of group communication system for non-replicated service and design a novel group communication system GCSLight according to the specification. GCSLight is novel in the following points: (1) it is lightweight in that it removes those complex and extraneous features, such as multicast and virtual synchrony, which exist in many other group communication systems aiming at replicated service. (2) it may obtain optimized performance in failure-free runs by adopting a lazy failure detection protocol. Finally, the preliminary performance of GCSlight is presented.

1. INTRODUCTION

Traditionally, server-attached storage (SAS) model dominates in distributed computing systems. Because SAS model limits direct storage sharing among servers, reliability has to be obtained by replicating critical information among multiple clustered servers. In order to support such replicated service, Group Communication System (GCS) middleware (Gregory C. et al. 2001) was advanced. Utilizing the infrastructure, replication-based clustered servers may be built in a two-layers architecture (Y. Amir. 1995):

- Clustered Servers Layer (CSL)

CSL is composed of a group of server processes that deal with application field affairs only, such as serving the query and update requests from clients, balancing the workload among themselves and recovering from possible failure conditions.

- GCS Layer (GCSL)

GCSL underlies CSL and provides the overlying layer with notifications of cluster membership changes, which will trigger the actions of recovery and rebalancing in CSL. Also, some multicast primitives are provided for disseminating messages to total server processes with required Quality of Service (QoS).

Such a layered architecture has advantages as follows:

- Modularity: Make it possible to separately reason about the guarantees of each layer and the correctness of its implementation.
- Simplicity: With GCS as a toolkit, it will release CSL from those complex tasks, such as detecting various failures and reporting them in a consistent manner, delivering messages to multiple sites with various safety and consistency guarantees. Therefore, the design and implementation of CSL can be simplified greatly.
- Scalability: With the membership service of GCS, CSL will reconfigure automatically, in case of failing, joining, and leaving of cluster members.

In recent years, the advent and popularization of high-speed storage area network has made possible non-replicated clustered servers. The non-replicated service obtains reliability and fails over by a direct take-over manner. For example, suppose that cluster member X is suspected to have crashed, cluster member Y will take over the shared disk of X and do recovery by analyzing the journaling existing on the shared disk. And then Y can take on the work completely charged by X .

The non-replicated service may be applied in many high performance distributed storage systems (J.Menon et al. 2003; Peter J. B. 2004) for shared storage environment. In these systems, a Metadata Server Cluster (MDS Cluster) provides all metadata and locks to clients and the clients access directly the data on shared storage. The size of MDS Cluster is usually very small. For example, a MDS Cluster with ten members is enough for a petabyte-scale distributed storage system (Sage A W. et al 2004).

The non-replicated service has different demands for underlying GCS, compared to replication-based service (R. Golding and O.Rodeh. 2003). In this work, we define a GCS specification suitable for non-replicated service and design a lightweight GCS: GCSLight. It differs from those existing group communication systems in the following points: (1) Simple design. It provides only the necessary GCS functions for non-replicated service. (2) Optimized performance. It eliminates the communication overhead completely in

Proceedings 19th European Conference on Modelling and Simulation
Yuri Merkuryev, Richard Zobel, Eugène Kerckhoffs © ECMS, 2005
ISBN 1-84233-112-4 (Set) / ISBN 1-84233-113-2 (CD)

failure-free runs by employing a lazy failure detection mechanism.

The rest of the paper is structured as follows: Section 2 defines the GCS specification for non-replicated service. Section 3 presents the design and implementation of GCSLight. Preliminary performance of GCSLight is given in section 4. The related work is described in section 5. Section 6 concludes the paper.

2. THE SPECIFICATION

Non-replicated service can implement failover by a direct take-over manner. So it is not necessary to replicate information among all cluster members. The only information needing to be kept globally consistent is the group membership. This can be achieved by a strong group membership protocol (Gregory C. et al. 2001) itself.

The group membership problem may be decomposed into two sub-problems: the processor group membership and the server group membership. The first problem is how to achieve agreement on the identity of all correctly functioning processors (or hosts, their incarnations are GCS daemons) that can execute server processes. The latter is how to maintain agreement on the global state of server process group, when server process joins, leaves or crashes.

2.1 Assumptions about the Environment

We all processors to use a physical local area network as the communication network; namely, no bridge elements may lie between two processors, such as hubs or routers. The processes communicate by exchanging messages and have access to private hardware clocks whose drift rate are bounded. Neither message delays nor computing speeds can be bounded with certainty. The asynchronous system model allows for the following failures: processes have crash and performance failure semantics.

In addition, we also assume that the system contains a hidden communication channel, namely the storage area network, which allows concurrent access to shared disks by multiple processes.

2.2 Failure Detection

Failure detection is fundamental to the adaptation of the processor group membership protocol. It defines the basic means to access availability status of processors and provide inputs to the processor group membership protocol.

We suggest a lazy failure detection protocol for GCS. A processor only checks the availability status of other processors on explicit requests. For example, when a client experiences a communication failure with a cluster member in CSL, it will notify some other available cluster member about the potential failure. At the exact moment, the available cluster member will explicitly require its GCS module to initiate the processor membership protocol to exclude the processor suspected to have failed. Once one processor is suspected to have failed, it will be deleted from the processor group. This is so-called single site suspicion (M. Hiltunen and R. Schlichting. 1995).

2.3 Processor Group Membership Protocol

A processor membership protocol is an agreement protocol for achieving a consistent system-wide view of the operational processors in the presence of member departure, member join and communication failure.

Processor membership protocols could be divided into primary-partition or partitionable membership protocols (Gregory C. et al. 2001). We suggest adopting a partitionable membership protocol, which allows the existence of multiple parallel processor groups at the same point in read-time as a sequence of network partition. It gives applications developers the flexibility of determining how to react when network partitions.

Formally, the processor group membership protocol of GCSLight has properties as follows:

Definitions:

joined (p,g) : holds true if processor p joins group g

LVp (g) : the local view of p about the members of group g.

Historyp : an infinite sequence of LV {LVp1, LVp2……, LVpk,……}

Historyp≡Historyq :

$$\forall \; j \; (LVpj = LVqj) \; \vee (LVpj \cap LVqj = \Phi)$$

(Property 1) Self-Inclusion

If a processor p joins group g, then p is a member of g. Formally: joined (p,g) → p∈LVp (g).

(Property 2) Agreement on Group Membership

If processor p and q joins the same group g, both p and p see the same members in the g. Formally:

$$\text{joined}(p,g) \wedge \text{joined}(q,g) \rightarrow LVp(g) = LVq(g)$$

(Property 3) Agreement on Group History

All processors have the same history. Formally:

$$\forall \; p,q \; (Historyp \equiv Historyq)$$

3. IMPLEMENTATION

3.1 Architecture and Interfaces

We implement a GCS (GCSLight) for non-replicated service according to the specification. The GCSLight is based on a daemon-client model where generally long-running daemons provide membership services to clustered servers in CSL. Servers linked with a small library GL_Lib must connect to the daemon resided on the same processor in order to gain access to the services of GCSLight.

The daemon architecture of GCSlight is presented in the Figure 1. The implementation of GCSlight is inspired by Spread (Yair A. and Jonathan. 1998) and Transis (D. Dolev and D. Malki. 1996).

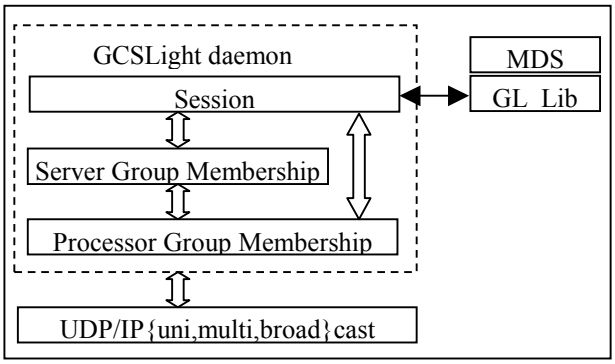

Figure 1: GCSLight Architecture

The GL_Lib provides the entire client interfaces as depicted in the Figure 2. The connection between the GL_Lib and the daemon is implemented by IPC mechanism. The Session and Server Group Membership modules manage user connections, server group memberships, and translate processor group membership changes into server group membership changes.

When one server fails, other user application components may call GL_viewcheck() to update the server group membership consistently. GL_viewrecv() may be used to receive the changes of server group membership.

```
GL_connect (const char* GCSLight_name,
                mailbox * mbox);
GL_disconnect (mailbox * mbox);
GL_join (mailbox * mbox, char * groupname);
GL_leave (mailbox * mbox, char * groupname);
GL_viewcheck (mailbox * mbox, char * error_report);
GL_viewrecv (mailbox * mbox, int max_mess_len,
                char mess*);
GL_error (int error);
```

Figure 2: GCSLight Application Programming Interfaces

3.2 Algorithm for Processor Group Membership

The processor group membership algorithm is based on jahanian's work (Jahanian F. et al. 1993). What different from jahanian's work is that we adopt a lazy failure detection mechanism in the protocol. The updating of new membership is done by a 2-phase protocol: Firstly, the leader of the group sends a "NEW_MEMBERSHIP" message to all other members; Secondly, the members except the leader will acknowledge the message; Finally, after collecting all "ACK" messages, the leader will send a "COMMIT" messages to all members. The leader of the processor group is the processor, which creates the group. The algorithm is intuitionally depicted as follows:

(1) Handling joins

When a processor starts, it will broadcast a message "JOIN_CLUSTER" to all possible processors in the network. If the processor can't discover any other active cluster nodes, it will assume that it is the first processor to start and other processors will join later, and form a singleton group and set itself as the leader. Otherwise, the leader of the existing cluster will accept the new comer and update the membership change to all cluster members including the new comer.

(2) Handling leaves

When a processor leaves the cluster normally, it will broadcast a "LEAVE_CLUSTER" message. If the processor is a leader, the rest of the cluster will elect a new leader and reform the cluster. Otherwise, the leader of the cluster will update the membership change to all other cluster members.

(3) Handling faults

The possible faults of a processor include performance failure (too slow), crash failure or communication failure (message omit or network partition). When one of these failures occurs, the fault will be reported to the corresponding GCSLight module. Then GCSLight will initiate the agreement protocol about the processor group membership. If the processor suspected to have failed is a leader, the rest of the cluster will elect a new leader and reform the cluster. Otherwise, the leader of the cluster will update the membership change to all other cluster members.

Because all processors agree on the sequence in which they join the group, the new leader may be selected locally when the current leader fails. Namely, the processor next to the current leader in the processor group membership is the leader candidate.

The pseudo-codes of the above algorithm are given in the appendix.

3.3 Algorithm for Server Group Membership

GCSLight provides the method of managing multiple servers like MDSes as a logical server group. The object of the server group membership protocol is to delivery the server group membership changes consistently to servers, whenever a server joins, leaves or fails.

The server group membership algorithm is triggered when one of the following events appears:

(1) User applications call GL_join, GL_leave;

(2) The processor group membership changes;

(3) Servers crashes. GCSLight monitors the status of servers by the services provided by operating system. For example, GCSLight will poll the socket (select()) for possible data, and then call recv() to receive the data. If recv() return 0, this will indicate a crash error of server.

The new server group membership is also promulgated in a 2-phase protocol alike in the processor group membership protocol.

4. PRELIMINARY PERFORMANCE

For non-replicated service oriented GCS like GCSLight, the performance metrics only include those of processor group membership protocol. The main metrics of processor group membership protocol (Oliver S. and Flavin C. 1998) are:

(1) Stability: the number of spurious membership changes per time unit;
(2) Join processing time: the time between the start of a server p and the moment a new group that includes p is formed;
(3) Failure detection time: the time between the crash or the disconnection of a member server p and the moment a new group that exclude p from its membership is formed.

Because the processor group membership protocol of GCSLight is based on lazy failure detection, the suspected failures are just reported by CSL. GCSLight itself will not cause spurious membership changes. Therefore, the stability of GCSLight is dependant on foreign failure detector, so does failure detection time. We only test the join processing time of GCSLight.

The tests were conducted on 10 Pentium II 350Mhz workstations interconnected by a 10 Megabit/sec Ethernet local area network. The workstations are equipped with Windows 2000 Server as operating system. Our measurements approximate the join processing time as the time between the broadcast of the "JOIN_CLUSTER" message by p and the moment p receives the "COMMIT" message.

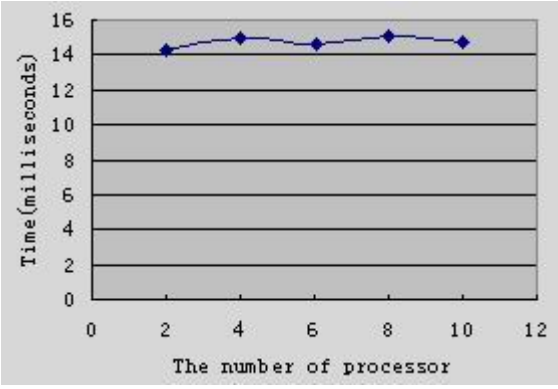

Figure 3: Join Processing Time

Because of the broadcast capability of Ethernet, the "JOIN_CLUSTER" message and "COMMIT" message are delivered to all processors at virtual the same time. So we can see that the joining processing time is almost constant independent of the number of processor.

The test of GCSLight is just preliminary. In further work, we will compare the joining processing time of GCSLight with those of other GCSes in the same test conditions.

5. RELATED OUR WORK TO OTHERS

Most of the existing GCSes were designed toward the end of supporting replicated service. Some of the leading GCSes are: ISIS (Birman, K. P. 1986), Phoenix (Malloth, C. P. et al. 1995), Transis (D. Dolev and D. Malki. 1996), Spread (Yair A. and Jonathan. 1998), etc. we call these GCSes replicated service oriented GCSes.

Replicated service oriented GCSes also provide multicast services and the programming model of virtual synchrony besides membership service (Y. Amir. 1995). The multicast services have different levels of ordering and reliability, such Safe Delivery, FIFO Order, Causal Order, Total Order. Virtual Synchrony orders membership messages with regard to application messages, so that all active members observe a same messages flow. Both multicast services and virtual synchrony are for ordering and reliability of replication. Therefore, they are redundant for non-replicated service.

To the best of our knowledge, the processor group membership protocols of existing GCSes are based on positive failure detection. For example, a processor sends "are-you-live" messages to other processors periodically and other processors acknowledge the messages. When a processor p misses a certain number of "ack" messages from q, p will suspect that q has failed. The shortage of the method is its messages overhead even in the failure-free runs.

We suggest a lazy failure detection-based processor group membership protocol in GCSLight. Lazy failure detection may eliminate the message overhead completely in failure-free runs. Considering that GCSLight is designed to support small clustered servers running on a physical network and the server platforms tend to become more and more reliable today, the failure conditions will appear rarely. Therefore, we argue that lazy failure detection will lead to better average performance.

6. CONCLUSION

In this paper, we define the specification of GCS for non-replicated service and design a novel group communication system GCSLight according to the specification. GCSLight is novel in the following points: (1) it provides only the necessary GCS functions for non-replicated service. So it is very lightweight. (2) by adopting a lazy failure detection-based processor group membership protocol, GCSLight itself does not incur any communication overhead in failure-free runs.

Although non-replicated service can also be built on some existing GCSes, like Transis and Spread, these systems are very complicated and hard to understand. Through the practice of designing and implementing the GCSLight, we think that it is worthy of paying out the price of developing a new, yet very simple and efficient GCS, when considering the cost of maintain and testing a very complex GCS with many unrelated features.

REFERENCES

A. Bartoli. 2004. "Implementing a Replicated Service with Group Communication", *Journal of Systems Architecture*, Elsevier, Volume 50, Issue 8(Aug), 45-519.

Birman, K. P. 1986. "ISIS: A System for Fault-Tolerant Distributed Computing". Technical Report TR86-744 (Apr), Cornell University, Department of Computer Science.

D. Dolev and D. Malki. 1996. "The Transis approach to high availability cluster communication", *Communications of the ACM* 39, No.4 (Apr), 64-70.

Gregory C. et al. 2001. "Group Communication Specification: A Comprehensive Study", *ACM Computing Surveys* 33,

No.4 (Dec),1-43.
Jahanian F. et al. 1993. "Processor Group Membership Protocols: Specification, Design and Implementation ". *In Proceedings of the IEEE Symposium on Reliable and Distributed Systems (Oct)*.
J.Menon et al. 2003. "IBM Storage Tank—A heterogeneous scalable SAN file system". *IBM Systems Journal,* Volume 42, Number 2.
M. Hiltunen and R. Schlichting. 1995. "Properties of membership services", *In IEEE International Symposium on Autonomous Decentralized Systems (Apr)* .
Malloth, C. P. et al. 1995. "Phoenix: A toolkit for building fault-tolerant, distributed applications in large scale". *In Workshop on Parallel and Distributed Platforms in Industrial Products (Oct)*.
Oliver S. and Flavin C. 1998. "Evaluating the Performance of Group Membership Protocols**".** *In Proceeding of Fourth International Conference on Engineering Complex Computer Systems (Aug)*, Monterey**,** California
P. Felber et al. 1999. "Failure detectors as first class objects". In *Proceeding of 1st IEEE Intl. Symp.on Distributed Objects and Applications (DOA'99)*, Edinburgh, Scotland, Sep., 132–141.
Peter J. B. 2004. "The Lustre Storage Architecture". http://www.lustre.org/docs/lustre.pdf, 2004.
R. Golding and O.Rodeh. 2003. "Group Communication still complex after all these years". http://www.haifa. il.ibm .com/projects/storage/zFS/public.html.
Sage A W. et al 2004. "Intelligent Metadata Management for a Petabyte-scale File System". *Second Intelligent Storage Workshop*, University of Minnesota (Apr).
Yair A. and Jonathan. 1998. "The Spread Wide Area Group Communication System", Technical Report 98-4, Center for Networking and Distributed Systems, Johns Hopkins University (Jul).
Y. Amir. 1995. "Replication Using Group Communication Over a Partitioned Network", PhD thesis, Institute of Computer Science, The Hebrew University of Jerusalem, Israel.

Appendix：The Pseudo-Codes of Algorithm in Section 3.2

/* variable definition */

myid: the processor id. It may be retrieved from the stable storage, its instance number is incremented by one after every restart.

CLV: current local view of a processor, which is a set of processor ids. Initially Φ;

SLV: suggested new local view by the processor group membership protocol. Initially Φ;

number: the sequence number in the CLV or SLV.

membership :Boolean initially false; holds true, if the processor group membership is in execution.

protocol_stage: when waiting for COMMIT message, set it to WAIT_COMMIT; Else, IDLE;

/* initialization */

```
myid←getmyid(); myid<-myid+1; setmyid();
broadcast (joinMes<JOIN_CLUSTER,myid>);
set join_timer;
```

/*main loop */

```
loop
 wait (message or timer)
 switch (message.type)
   case JOIN_CLUSTER:
     do_member_join(&message);
     membership←true;
   case LEAVE_CLUSTER:
   case VIEW_CHECK:
     do_member_leave(&message);
     membership←true;
   case PREPARE_COMMIT:
     do_ack (&message);
   case COMMIT:
     do_delivery (&message);
     membership←false;
   case ACK:
     do_ack_check(&message) ;
   ……
 endswitch
if Join_Timer.expire
  CLV←CLV∪{myid};
  number←get_number (CLV);
  Deliver CLV to the layer of server group membership.
fi
if Wait_PreCommit_Timer.expire
   SLV←CLV- {processor ids whose number is larger than that of myid.};
   broadcast(PreCommitMesM
              <NEW_MEMBERSHIP,myid,SLV>);
   set Collect_Ack_timer(timeout_value);
 fi
 if Collect_Ack_timer.expire
    SLV←CLV-{those members which do not send ACK message};
   broadcast(PreCommitMesM
              <NEW_MEMBERSHIP,myid,SLV>);
   set Collect_Ack_timer(timeout_value);
 fi
if Wait_Commit_Timer.expire   /*leader dies*/
  SLV←CLV-{ processor ids whose number is larger than that of myid};
  Broadcast(PreCommitMes
              <NEW_MEMBERSHIP,myid,SLV>);
end loop

procedure do_member_join(&message)≡
 if membership is false
   SLV←CLV∪{message.sender};
   if number =1 /*is I am leader*/
     if message.sender is not in CLV
       broadcast(PreCommitMes
              <NEW_MEMBERSHIP,myid,SLV>);
       set Collect_Ack_timer(timeout_value);
     fi
   else
     set Wait_PreCommit_ Timer (
                  number*timeout_value);
   fi
 fi
```

```
procedure do_member_leave(&message) ≡
 if membership is false
    if number =1
       if message.failed_member is in CLV
          SLV←CLV-{message.failed_member};
          broadcast(PreCommitMes
                    <NEW_MEMBERSHIP,myid,LV>);
       fi
    else
       set Wait_PreCommit_timer (
                                  number*timeout_value);
    fi
 fi

procedure do_ack (&message) ≡
   if myid is in message.SLV
      if message.sender is the leader in CLV
         unicast(AckMes<ACK,myid>);
         protocol_stage←WAIT_COMMIT;
         unset Wait_PreCommit_Timer;
         set Wait_Commit_Timer(
                         number*timeout_value);
      else if protocol_stage is WAIT_COMMIT
         unicast(AckMes<ACK,myid>);
         number←get_number(message.SLV);
         set Wait_Commit_Timer(
                          number*timeout_value);
      fi
   fi

procedure do_ack_check(&message) ≡
   if receive all "ACK" messages from every member
      broadcast (CommitMes<COMMIT,myid>);
      unset Collect_Ack_timer;
   fi

procedure do_delivery (&message) ≡
/*according to the change of processor group
membership, check and report the server group
membership to user applications.*/
   if protocol_stage is WAIT_COMMIT
   unset Wait_Commit_Timer.
   CLV ←SLV; number←get_number (CLV);
   protocol_stage←IDLE;
   Deliver CLV to the layer of server group
   membership;
   Fi
```

AUTHOR BIOGRAPHIES

Ruiyong Jia was born in 1976. He received the B.S. Degree in Computer Science from Henan University, China in 1999 and the M.S. Degree in Computer Science from Northwestern Polytechnical University, China in 2002. He is currently a Doctor Candidate at School of Computer Science, Northwestern Polytechnical University. His research interests include distributed algorithms and fault tolerant systems, distributed storage systems.

Yanyuan Zhang was born in 1954. He is a Full Professor and a head of Software Engineering Institute in School of Computer Science, Northwestern Polytechnical University. He received his Master Degree from Northwestern Polytechnical University in 1987 with specialty of computer He is a member of the Committee of Information Storage Technology, China Computer Federation. He has published four books, over 20 Journal papers. His research interests cover software engineering, storage management, fault tolerance, and distributed parallel processing.

Yong Feng was born in 1977. He is a Doctor Candidate at School of Computer Science, Northwestern Polytechnical University. He received his Master Degree from Northwestern Polytechnical University in 2002 with specialty of computer software theory. His research interests are in high performance network storage system and parallel I/O, fault tolerance, parallel and distributed computing, performance evaluation and benchmark.