A Replication-Based Fault Tolerance Protocol Using Group Communication for the Grid

Kayhan Erciyes

Izmir Institute of Technology
Computer Eng. Dept., Urla, Izmir 35430, Turkey
kayhanerciyes@iyte.edu.tr

Abstract. We describe a replication-based protocol that uses group communication for fault tolerance in the Computational Grid. The Grid is partitioned into a number of clusters and each cluster has a designated coordinator that manages the states of the replicas within its cluster. The coordinators belong to a process group and the proposed protocol ensures the correct sequence of message deliveries to the replicas by the coordinators. Any failing node of the Grid is replaced by an active replica to provide correct continuation of the operation of the application. We show the theoretical framework along with illustrations of the replication protocol and its implementation results and analyze its performance and scalability.

1 Introduction

Computational Grids consist of heterogenous computational resources, possibly with different users, and provide them with remote access to these resources [1], [2]. The Grid has attracted researchers as an alternative to supercomputers for high performance computing. One important advantage of Grid computing is the provision of resources to the users that are locally unavailable. Since there are multitude of resources in a Grid environment, convenient utilization of resources in a Grid provides improved overall system performance and decreased turn-around times for user jobs. Users of the Grid submit jobs at random times with significant turnaround times and failure of a node of the Grid would halt the execution of the application necessiating the need for fault tolerance in the Grid. Furthermore, the difficulty and the cost of recovering from faults in the Grid environment may be higher than the normal applications. Fault tolerance schemes in the Grid environment can be classified as the application specific fault tolerance based on middleware fault detection; task and data replication at middleware and transport levels; WAN, MAN and LAN resilience schemes at Internet/Network Level [3].

In this study, we propose a model and a protocol to perform task replication at middleware level for fault tolerance in the Grid using the process groups. A process group is a logical name for a set of computing elements whose membership may change with time. Replication using process groups for fault tolerance has attracted many researchers for many years [8][9][10][11]. There are several

systems which provide fault tolerant group communication such as Transis [7], Horus [16] and Totem [6]. Moshe [14] extends these services to a WAN. The common goal of these projects is to provide a reliable multicast communication for process groups. Total Order Multicast is the basic paradigm to provide message ordering in fault tolerant systems that use active replication [15]. It has been studied extensively and many protocols have been proposed. A detailed survey is given in [12].

An important component of the replication mechanism is the *Total Order Multicast (TOM)* protocol which ensures that all of the replicas receive the multicast messages destined to the replica group in the same order so that all of the replica finite state machines are in identical states. To achieve TOM in the Grid, we assume that the Grid is partitioned into clusters by a suitable algorithm or manually. Each cluster is controlled by a cluster head called the *coordinator*. These coordinators are the cluster heads and interface points for the ordinary nodes to the network. They perform TOM on behalf of the ordinary nodes they represent. The rest of the paper is organized as follows. Section 2 provides the background on group communication and TOM. In Section 3, the proposed protocol including the coordinator and the node algorithms is described. In Section 4, the operation of the protocol is illustrated and Section 5 provides the analysis of the algorithms. The implementation results obtained from the tests are given in Section 6 and Section 7 contains the concluding remarks along with discussions.

2   Background

2.1   Group Membership

Replication is a common approach to achieve fault tolerance in a distributed system such that replicas provide redundancy in case of a failure of a server. Two main classes of replication are the *active* and *passive* replications. In passive replication, client deals only with one replica and the primary sends messages to the secondaries to update their views. A client sends a message to all of the replicas in active replication and the states of the replicas are maintained as identical, in general, using finite state machines. To ensure consistency of the replicas, a group communication primitive called the Total Order Multicast may be used which guarantees that the requests by the clients are received by all replicas in the same order.

A group membership service manages a group of processes and is based on the *view* which is the list of processes belonging to a group. *View change* should be notified to all members. There are three basic operations needed to manage group membership effectively; *join, leave and exclude. Join* is executed by a process p and upon acceptance of it, all of the processes update their view. More importantly, the state of the group needs to be transferred to the new member p. A process will be removed from a group by *exclusion* if its crash is detected by a member of a group and *exit* is a voluntarily release of a process from a group by itself. The group management module should also provide the two primitives;

send_multicast to send a message to all members and *receive_multicast* to receive a message sent by a member of the group. These two primitives can be realised using various approaches such as *reliable broadcast, reliable FIFO broadcast and total order multicast. Reliable Broadcast* of a message in a group ensures that messages are delivered by all processes or none.

2.2   Total Order Multicast

Total Order Multicast (TOM) ensures that no pair of messages are delivered to the members of a group in a different order. TOM can be specified in terms of the following properties :

- *Validity* : If a correct process broadcasts a message m, then some correct process in its group will eventually deliver it.
- *Uniform Agreement* : If a process delivers a message m, then all correct processes in its group will eventually deliver it.
- *Uniform Integrity* : Every correct process in the sender's group delivers m at most once and only if m was previously broadcast.
- *Uniform Total Order* : If two correct processes deliver two messages m_1 and m_2, they do it in the same order.

Atomic broadcast is a special case of total order multicast where a TOM message is delivered to all of the group members or none. In other words, Atomic Broadcast obeys TOM and Reliable Broadcast. Atomic Broadcast or Reliable TOM protocols can be symmetric or asymmetric depending on whether some nodes are privileged in the system exist or not. Most of the symmetric protocols such as Isis [8] impose total order from the casual order relation between the messages. The static sequencer protocols such as in Amoeba [13] assume a sequencer where messages are first transmitted to this sequencer which multicasts them in order. One disadvantage of the central sequencer type of asymmetric TOM protocols is the message bottleneck around this component and having a single point of failure in the system.

3   Replication Protocol

3.1   The Model

We assume that the clusters of the Grid are already formed. For TOM in the Grid, we propose the architecture shown in Fig. 1 where replicas form clusters and each cluster is represented by a coordinator. Each replica cluster is a process group called the *replica group* and furthermore, coordinators of the clusters form a single outer group called the *coordinator group*. Election of a new coordinator is provided as in [5] if it crashes. Coordinators perform multicast communication in both groups they belong but their main function is to represent their cluster replicas in the outer coordinator group.
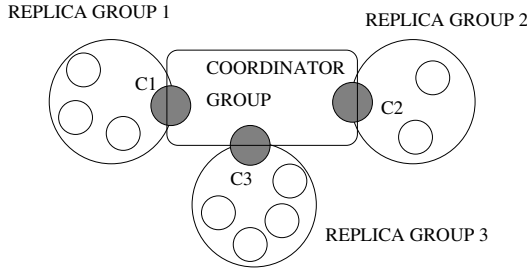
Fig. 1. The Replication Model for the Grid

3.2   The TOM Protocol

The TOM protocol proposed for the hierarhical groups use the *Static Token Algorithm (STA)* which employs similar data structures for the Token as in Suzuki-Kasami [17] algorithm for distributed mutual exclusion. A process that acquires the system wide unique token has the right to send a TOM message. The token data structure is as follows :

- Token_Sequence_Number (TSN) : integer;
- Token_Request_Queue (TRQ) : Queue of nodes;

Also every node has a *Local Sequence Number* (LSN) which shows the last sequence number of any TOM_Msg that node has received. Any node that requires to send a TOM_Msg, sends a request (Node_Req) to its coordinator as shown in the state machine diagram of Fig. 2. The request by the node is converted to a Tok_Req by the coordinator which is circulated in the ring. The coordinator sets its state to *Wait Token* (WTTK) and changes to *Hold Token* (HLTK) when it receives the token and then forwards it to the requesting node. Once a node receives the token from the coordinator, it incerements TSN and stamps the TOM message with this TSN and sends it to the coordinator. Since there is a unique Token with a unique TSN, any TOM message from any node will have a unique sequence number which provides the total ordering of the messages. The FSM of the coordinator also depicts the sequence for atomicity. When the coordinator receives the TOM_Msg from the node, it broadcasts this to the ring and upon reply, it sends TOM_Chk message to check the acknowledgements. If all nodes in the group have received the TOM_msg, the operation is succesful and a final TOM_Set message is sent to all coordinators to allow the final delivery to the nodes. Any node receiving the TOM_Set message checks whether LSN+1=TSN, that is, whether this message has arrived in sequence. If so, TOM_Msg is delivered to the application, otherwise it is delayed until prior messages arrive. At any state, a coordinator may receive a remote TOM_Msg which is not shown in Fig.2 for simplicity. In this case, the coordinator braodcasts the TOM_msg in its cluster and also receives replies from each node.
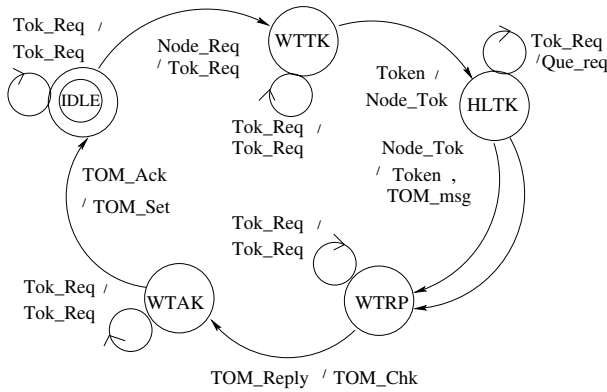
Fig. 2. The Coordinator for Static Token Algorithm

4   Illustration of STA

Fig. 3 shows an example scenario for STA. Initially, Token is held in the replica cluster 2 by coordinator C_2. For simplicity, it is assumed that coordinators are directly connected to the nodes and to each other. The following is the sequence of events :

1. Node n_{13} in cluster 1 requests Token from its coordinator C_1 by *Node_Req*.
2. C_1, does not have the Token, hence broadcasts this request by R_{13} in the coordinator group.
3. C_2 has the token which is not being used and has TSN as 0 and its queue is empty. C_2 sets the destination of the token as n_{13} and sends this token to C_1.
4. C_1 receives the Token and sends it to n_{13} which is received by it. These first four steps are depicted in Fig. 3.(a).
5. While n_{13} is holding the Token, nodes n_{21} and n_{32} in clusters 2 and 3 make requests consecutively to their coordinators for the Token which in turn send requests R_{21} and R_{32} tothe coordinator group.
6. R_{21} and then R_{32} reach C_1 consecutively. C_1 queues these requests.
7. When n_{13} receives the token, it increments TSN of the Token to 1 and sends TOM message with this sequence number to C_1 along with the Token. Steps 5-6-7 are depicted in Fig. 3.(b).
8. C_1 receives the Token and the TOM message. It broadcasts TOM on the coordinator group. C_1 also checks its local Coordinator Token Request Queue (CRQ). It appends the nodes in its local queue to the Token Queue, removes the first node from the TRQ (n_{31}) and sends the token to C_2.
9. C_3 and C_2, pass an acknowledgement message (TOM_Ack) to the source. Steps 8 and 9 are depicted in Fig. 3.(c).
10. If C_1 receives (TOM_Ack) acknowledgements from every node in the group, the TOM is succesful. In this case, C_1 issues a TOM_Set message to finally

initate the actual delivery of the TOM message to the application. The replica node however, checks its LSN with TSN to conclude TOM delivery as described above.

11. When C_2 receives the Token from C_1, it proceeds similar to 7-8-9-10 above and when n_{21} finishes with the Token, C_2 sends it to C_3. Steps 10 and 11 are depicted in Fig. 3.(d). Note that this is performed in parallel with the TOM_Msg delivery of n_{13}.
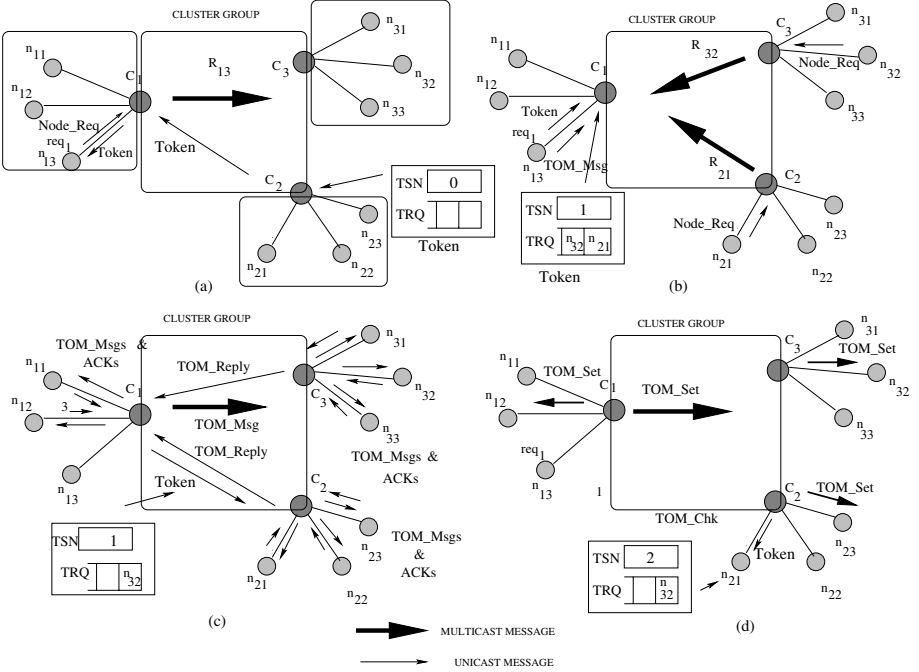

Fig. 3. Operation of the STA Algorithm

5 Analysis of STA

Assuming k, m, n and d are upperbounds on the number of clusters, nodes in a cluster in the network, nodes in the ring of coordinators and the diameter of a cluster respectively, the sending and atomically reception of the TOM message by all nodes in the group requires the following steps :

1. Request by the node : $O(d)$
2. Circulation of Tok_Req and reception of token : $O(k)$ as this is *All to All Communication* in a unidirectional ring or again $O(k)$ this is k unicast messages to implement a multicast message in case of a different architecture.
3. Sending of Token to the Node : $O(d)$

4. Sending of TOM message by the node to the coordinator : $O(d)$
5. Circulation of TOM message by the coordinator : $O(k)$
6. Local broadcast of TOM by each coordinator in its cluster $O(m)$
7. Collection of the acknowledgements from the nodes by each coordinator : $O(m)$
8. Circulation of acknowledgement message (TOM_Ack) by the coordinator : $O(k)$
9. Set operation by the source cordinator for atomicity : $O(k)$
10. Set operation by the coordinators in local clusters for atomicity : $O(m)$

Theorem 1. *The total time per TOM using STA Algorithm is $O_{TOMT}(m)$*

Proof. The total time required for TOM delivery is the sum of all of the 9 steps above which can be evaluated as follows

$$O_{TOMT} = 4k + 3m + 3d = O_{TOMT}(m) \tag{1}$$

assuming $k{=}m$ and d is negligible.

Theorem 2. *The total number of messages per TOM using STA Algorithm is $O_{TOMM}(m^2)$*

Proof. The total number of messages required for TOM delivery can be found similarly by calculating the sum of messages in transit at each step of operation above except for local broadcast operations (steps 6,7 and 10) where the number of messages sent are k*m.

$$O_{TOMM} = 4k + 3km + 3d = O_{TOMM}(m^2) \tag{2}$$

assuming $k{=}m$ and d is negligible.

Corollary 1. *For a network of N nodes, the total time per TOM using STA Algorithm is $O_{TOMT}(\sqrt{N})$ and the total number of messages required per TOM using STA is $O_{TOMT}(N)$.*

Proof. It was shown by theorems 2 and 3 that $O_{TOMT}(m)$ and $O_{TOMM}(m^2)$. Since total number of nodes in a network N is equal to the total number of nodes in the model network which is $km{=}m^2$ assuming k=m, $O_{TOMT}(\sqrt{N})$ and $O_{TOMT}(N)$.

An algorithm like Suzuki-Kasami [17] would require 0 or N messages ($N-1$ for requests and 1 for Token). By Corollary 1, we can conclude that STA provides and order of magnitude decrease in TOM execution time including obtaining the Token with respect to an algorithm like Suzuki-Kasami.

5.1   Verification of Total Order Multicast Properties by STA

The TOM properties can be verified for STA operation follows :

- *Validity* : When a correct process broadcasts a message *msg*, then all of the correct *TO_Node* processes will deliver this message as directed by their cluster coordinators in TOM_Set message.
- *Uniform Agreement* : The delivery of the messages to the application by all of the *TO_Node* processes occur in the same cycle and the messages are delivered by all of the correct *TO_Node* processes in the cluster identities of which are supervised by the local coordinator.
- *Uniform Integrity* : The messages at a node are delivered at most once by the TOM_Set message issued by the coordinator.
- *Uniform Total Order* : The messages are delivered in the same order as there is only one sequence number (TSN) kept at Token and only the holder of the Token can send a TOM message. Each node has a local sequence number (LSN) and will not deliver a message that is larger than its LSN+1 which means any out of order messages wil be delayed until the TOM_Msg with the correct sequence arrives.

6   Implementation Results Using MPI

We implemented the architecture shown in Fig. 1 using the MPI (Message Passing Interface) [4] over a cluster of 20 processors. The end-to-end run times of the STA Algorithm for a single request, from the time of the request until the actual delivery of the TOM message to the application are measured against varying number of clusters and the size of clusters as shown in Fig. 4.
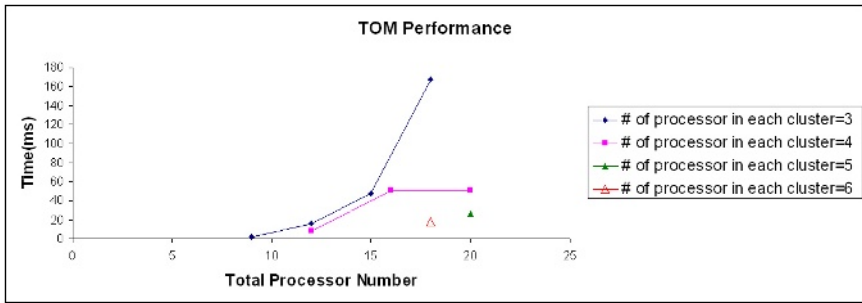

Fig. 4. TOM Run Time Results against Cluster Size and Numbers

MPI multicast message facility was used for group communication within a cluster. As shown in the figure, the run time of STA increases linearly with the number of clusters and also the count of processors in each cluster. The performance of the proposed architecture in terms of the size of the single message delivered to the application is depicted in Fig. 5. It can be observed that the delivery times are almost stable with respect to cluster numbers and sizes.
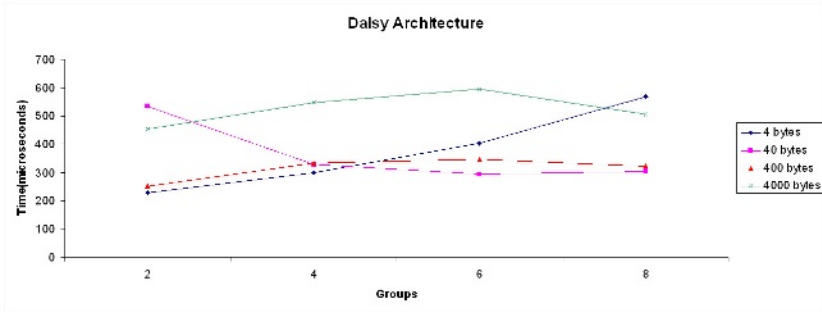
Fig. 5. Daisy Architecture Run Time Results against Message Size

7   Discussions and Conclusions

We proposed a framework and a protocol with two algorithms to implement TOM in the Grid to provide fault tolerance by replication to the application. We showed that the proposed algorithms provide significant gains in the number of messages and the time to deliver TOM messages theoretically with respect to a flat architecture without any hierarchies. The preliminary results indicated that the protocol is scalable in terms of run times and the sizes of the messages delivered. The coordinators have an important role and they may fail. New coordinators may be elected and any failed node member can be excluded from the cluster which is an improvement over classical algorithms as they do not provide recovery for a crashed node in general. The recovery procedures can be implemented using algorithms as in [5] which is not discussed here. One other advantage of the proposed model is the pre-processing of the requests of the nodes by the coordinators are performed independently resulting in improved performance. We are looking into implementing this protocol in a Grid environment with larger cluster sizes and counts and measure the performance of the whole protocol including the clustering algorithm.

Acknowledgements

I would like to thank Orhan Dagdeviren for his help in the experimental setup of the tests in Section 6.

References

1. Foster, I., Kesselman, C., Tuecke, S.: The Anatomy of the Grid: Enabling Scalable Virtual Organizations. Int. Journal of High Performance Computing Applications, 15(3), (2001), 200-222.
2. Foster, I.: What is the Grid ? A Three Point Checklist, Grid Today, 1(6), (2002).

3. Valcarenghi, L et al. : QoS Aware Fault Tolerance in Grid Computing, Workshop on Reliability and Robustness in Grid Computing Systems, GGF16, Feb. 13-16, 2006, Athens, Greece.
4. MPICH-G2: A Grid-enabled Implementation of the Message Passing Interface, Journal of Parallel and Distributed Computing, 63(5), (2003), 551 - 563.
5. Tunali, T, Erciyes,K., Soysert, Z.: A Hierarchical Fault-Tolerant Ring Protocol For A Distributed Real-Time System, Special issue of Parallel and Distributed Computing Practices on Parallel and Distributed Real-Time Systems, 2(1), (2000), 33-44.
6. Y.Amir et all, The TOTEM Single Ring Ordering and membership Protocol, ACM Trans. Comp. Systems., 13(4),1995.
7. Y.Amir et all, Transis: A communication subsystem for high availability. Proc. of 22nd IEEE Int'l Symp. on Fault-Tolerant Computing, IEEE Press, NJ, pp. 76-84.
8. Birman K. P., van Renesse, R., Reliable Distributed Computing with the Isis Toolkit, IEEE Computer Society Press, Los Alamitos, Ca., 1994.
9. Birman K. P., The Process Group Approach to Reliable Distributed Computing, Communications of the ACM, 36(12), December 1993.
10. Chockler, G, Keidar, I., Vitenberg, R., Group communication specifications: a comprehensive study, ACM Computing Surveys, 33 (4), December 2001, pp. 427 - 469.
11. Cristian F., Synchronous and Asynchronous Communication, Communications of the ACM. Special Section on Group Communication, 39(4), April 1996.
12. Defago, X., Agreement Related Problem : From semi-passive replication to Totally Ordered Broadcast. Ph.D. thesis, Ecole Polytechnique Lausanne, Switzerland, Aug. 2000.
13. Kaashoek, M. F., Tanenbaum, A. S., Group Communication in the Amoeba distributed operating system, Proc. of the 11th IEEE International Conf. on Distributed Computing Systems, IEEE Computer Society press, pages 436-447
14. Keidar, I. et al, Moshe: A group membership service for WANs, ACM Transactions on Computer Systems (TOCS) 20 (3) , August 2002, 191-238.
15. Schenider, F., Replication management using the state-machine approach, *Duistributed Systems*, ACM Press, 169-198.
16. Van Renesse R., Birman K. P.,Maffeis S., Horus : A Flexible Group communication System, Communications of the ACM, Special section on Group Communication, 39(4), April 1996
17. Susuki, I., Kasami, T. : A Distributed Mutual Exclusion Algorithm, ACM Trans. Computer Systems, Vol. 3(4), (1985), 344-349