Atomic Commitment in Grid Database Systems

Sushant Goel[1], Hema Sharda[1], and David Taniar[2]

[1] School of Electrical and Computer Systems Engineering, Royal Melbourne Institute of Technology, Australia
s2013070@student.rmit.edu.au
hema.sharda@rmit.edu.au
[2] School of Business Systems, Monash University, Australia
David.Taniar@infotech.monash.edu.au

Abstract. Atomic Commitment Protocol (ACP) is an important part for any distributed transaction. ACPs have been proposed for homogeneous and heterogeneous distributed database management systems (DBMS). ACPs designed for these DBMS do not meet the requirement of Grid databases. Homogeneous DBMS are synchronous and tightly coupled while heterogeneous DBMS, like multidatabase systems, requires a top layer of multidatabase management system to manage distributed transactions. These ACPs either become too restrictive or need some changes in participating DBMS, which may not be acceptable in Grid Environment. In this paper we identify requirements for Grid database systems and then propose an ACP for grid databases, *Grid-Atomic Commitment Protocol (Grid-ACP)*.

1 Introduction

Atomic commitment is one of the important requirements for transactions executing in distributed environments. Among Atomicity, Consistency, Isolation and Durability (ACID) [4] properties of a transaction, Atomic Commitment Protocols (ACP) preserves the atomicity of transaction running in distributed environment. *Two-phase Commit* (2PC) and its variants are widely accepted as ACP for transactions running in distributed data repositories [2,3,14]. These data repositories are considered to be homogeneous, tightly integrated and synchronous.

Grid infrastructure [7,8], a new and evolving computing infrastructure promises to support collaborative, autonomously evolved, heterogeneous, data intensive applications. Grid databases would access distributed resources in general and distributed data repositories in particular. Thus, protocols developed for homogeneous distributed architecture will not work in the Grid infrastructure. Hence classical approaches of data management need to be revisited to address challenges of grid databases.

Transaction management is critical in any data based application, be it simple file management system or structured Database Management Systems (DBMS). Transaction management is responsible to manage concurrency control and reliability protocols. Many applications will not need transactional support, i.e. ACID properties, while executing on Grids e.g. Business Activities [12]. Our earlier work

H. Jin et al. (Eds.): NPC 2004, LNCS 3222, pp. 22-29, 2004.
© IFIP International Federation for Information Processing 2004

was focused on concurrency control in Grid environment [17]. In this paper we particularly focus on ACP in Grid environment.

Grid databases [1,2] are expected to store large data from scientific experimentations viz. astronomical analysis, high-energy physics [16], weather forecasting, earth movement etc. These experiments generate huge volume of data daily. Particle physics experiments, e.g. *Babar*, may need to store up to 500 GB of data each day and is arguably world's largest database that stores approx. 895 TB of data as of today (Mar '04) [15]. Wider research community is interested in generic data collected at various data collecting sites [1,10,13,15]. Distributed access to data raises many issues like security, integrity constraints, manageability, accounting, replication etc. But, here we will be mainly concerned with managing the transaction in Grids and its requirement of atomic commitment. In this paper distributed database is used in a broader sense to cover distributed/federated/multidatabase systems, since all these accesses data located at physically distributed sites, unless otherwise stated.

The remainder of the paper is organized as follows. Section 2 explains the background work in distributed DBMS. Section-3 explains the working model and identifies the problem in applying existing ACP in the Grid model. We propose the Grid-ACP to meet Grid requirement for ACPs in section-4 along with proof of correctness of the protocol. Section-5 concludes the work and explains future work.

2 Background

Atomic commitment is an important requirement of transactions running in distributed environment. All cohort of distributed transaction should either commit or abort to maintain the *atomicity* property of the transaction and thus consequently maintain the correctness of stored data. We broadly classify distributed DB systems in two categories: (a) Homogeneous and (b) Heterogeneous distributed DBMS. Detailed classification can be found in [14].

2.1 Homogeneous Distributed Database

2PC [4] is the simplest and most popular ACP proposed in the literature to achieve atomicity in homogeneous DBMS [3,14]. We briefly discuss 2PC from the literature to help our further discussion. The site where the transaction originates acts as *coordinator* for that transaction; all other sites where data is accessed are *participants*. 2PC works as follows [4]:

The coordinator sends *vote_request* to all the participating sites. After receiving a request the site responds by sending its vote, *yes* or *no*. If the participant voted *yes*, it enters in *prepared* (or *ready*) state and waits for final decision from the coordinator. If the vote was *no*, the participant can abort its part of the transaction. After collecting all the votes, if all of them including the coordinator's vote are *yes* then the coordinator decides to *commit* and send the message accordingly to all the sites. Even if, one of the votes is *no* the coordinator decides to abort the whole transaction. After receiving *commit* or *abort* decision from the coordinator, the participant commits or

aborts accordingly from prepared state. While the participant is in prepared state it is *uncertain* of the final decision from the coordinator. Hence 2PC is called as a *blocking protocol*.

2.2 Heterogeneous Distributed Database

Multidatabase systems assume heterogeneous environment [5,9] for transaction execution. They typically execute a top layer of multidatabase management system for transaction management. These systems are designed for certain application specific requirements and mostly for short and synchronous transactions. Due to high autonomy (design and execution) requirements in multidatabase systems, the ACPs are not designed for replicated data. Thus these protocols are not suitable for Grid environment. In literature [9] following major strategies are discussed for atomic commitment of distributed transaction in heterogeneous database environment: (1) Redo (2) Retry (3) Compensate.

Since all sites may not support prepare-to-commit state and thus even if global transaction decides to commit, some local sub-transaction may decide to abort while others may decide to commit. Hence, transactions that decided to abort must *redo* the write operation, and commit, to reach consistent global decision [9]. Another approach to deal with above problem is the *retry* approach, as discussed in [9]. In retry approach, the whole subtransaction is retried rather than *redo*ing only the write operations. Inherent limitation of this approach is that the subtransaction must be *retriable*. A subtransaction is retriable only if the top layer of multidatabase system has saved the execution state of the aborted subtransaction. If the global decision is to abort and any local subtransaction has already committed, then *compensating* transactions can be executed [9]. Compensating transactions also need to access information stored in global DBMSs.

3 Grid Database Model and Problem Identification

In this section we first discuss the general model and terminology that we use in our study. Then we discuss the problem in implementing standard ACPs in this model.

3.1 Model

The Grid middleware will join geographically separate computing and data resources. Concept of virtual organization (VO) [7] has been coined for integrating organizations over network. Grid infrastructure is expected to support and make use of *web-services* for specialized purposes. We focus on the collaborative, data intensive work that need to access data from geographically separated sites. The general model is shown below:

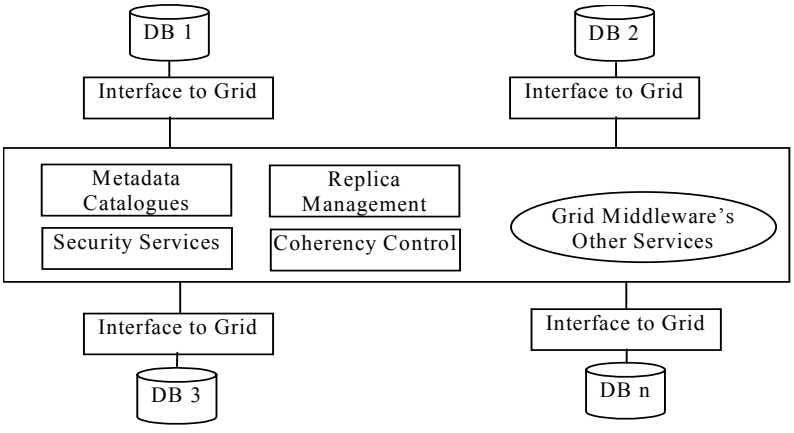

Fig. 1. General model of Grid database system

All individual database systems are autonomously evolved and hence heterogeneous in nature. These database systems may join and leave the Grid as per their convenience. A transaction is termed as global transaction if it originates at any site and need to access data from other sites, in other terms if the transaction has to access data from more than one site it is a global transaction. The division of the global transaction at individual sites are called subtransactions.

3.2 Problem Identification

2PC is the most widely accepted ACP in distributed databases. 2PC is a consensus-based protocol that asks all the participating sites to vote whether *subtransactions* running at that site can *commit*. After collecting and analyzing all votes, the coordinator decides the fortune of the distributed transaction. It involves two phases, *voting phase* and *decision phase*, of communication messages before terminating the transaction atomically, hence the name *two-phase commit*.

Many variations and optimizations have been proposed to increase the performance of 2PC. But, homogeneity between sites is the basic assumption behind the originally proposed 2PC for distributed databases. Multi/federated database systems are heterogeneous but the nature of transactions and applications these heterogeneous database systems are studied, designed and optimized are much different than their counterparts in Grid databases, e.g. for short, synchronized, non-collaborative transactions, to name few of them. These systems have a leverage of a top level layer, known as multidatabase management system that assists in making decision but Grids may not enjoy this facility due to distributed nature of database systems. Multidatabase employs redo, retry and compensate approach for ACP. These requirements may not be implemented in absence of top-layer management system and at the same time may be too restrictive [6]. Grid databases need to operate in a loosely coupled *service-oriented* architecture. Apart from data consistency perspective Grid databases will be expected to access data from via WWW [11,12]. Most of the distributed DBMSs are not designed to operate in WWW environment.

4 Proposed Protocol

As discussed earlier, requirements of Grid DB systems cannot be satisfied by existing distributed DBMS. In this section we propose an ACP to meet these requirements.

4.1 Grid Atomic Commitment Protocol (Grid-ACP)

Before we proceed with the protocol we would like to remind that executing compensating transactions don't result in standard atomicity of transaction. The notion is referred as *semantic atomicity* [9].

Figure-2 shows the state diagram of proposed Grid-Atomic Commitment Protocol (Grid-ATC). We introduce a new state and call it *sleep state*. The sub-transaction will enter in *sleep* state, when it finishes execution and is ready to release all acquired resources. *Sleep* state is an indication to transaction managers that the local sub-transaction of global transaction has committed. But it is still waiting for decision from the originator of the transaction. If any of the other participating sites aborts the subtransaction, the coordinator informs all the *sleep*ing sites to *compensate* the changes made by the transaction.

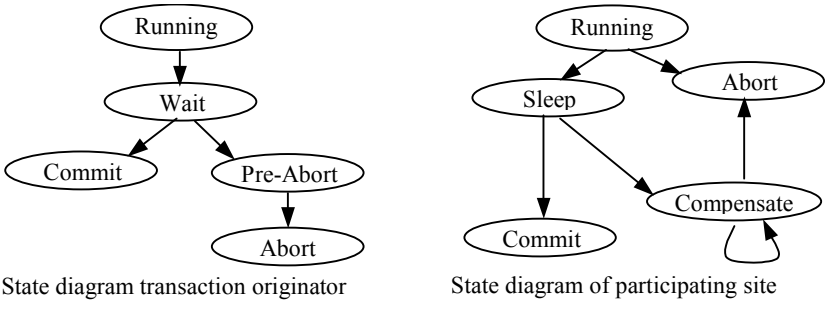

Fig. 2. State diagram of Grid-ATC

The Grid-ATC algorithm is explained as follows:

1. The transaction originator splits the transaction based on the information at *Grid-middleware service* and submits to participating database systems.
2. Respecting the autonomy of participating sites, they execute their portion of sub-transaction and goes to *sleep* state, after logging all the necessary compensating information in the stable storage. The site then informs the outcome of the sub-transaction execution to the originator.
3. The originator, after collecting response from all participants, then decides whether to commit or to abort. If all participants decided to go in sleep state the decision is to commit else the decision is to abort. If the decision is to abort, message is sent only to those participants who are in sleep state. If the decision is to commit, it is sent to all participants.

4a. If the local site decided to commit and is in sleep state and the global decision is also to commit, the transaction can directly go to commit state. As everything went as expected by the local site.

Grid-ACP: Originator's Algorithm

```
submit sub-transactions to participants;
wait for response from all participants;
if all response to sleep then begin
      write commit record in log;
      send global_commit to all participants;
end if
else begin
      write abort record in log;
      send global_abort to participants who decided to commit
      wait for response from these participants;
end
return
```

Grid-ACP: Participant's Algorithm

```
received sub-transaction from originator
if participant decides to commit then begin
      write sleep in log
      send commit decision to originator
      wait for decision from originator
      if decision is commit then
           write commit in log
      end if
end if
else if decision is abort then begin
      start compensating transaction for this transaction
line 10:  if compensating transaction aborts then begin
           restart compensating transaction until it commits
           write commit for compensating transaction
      end if
      else
           write commit for compensating transaction
      end
end if
else if participant decides to abort then begin
      write abort in log
      send abort decision to originator
end if
return
```

4b. If the local site decided to commit and is in sleep state but the global decision is to abort the transaction, then the local transaction must be aborted. But as mentioned earlier when the local site enters the sleep state it releases all locks on data items as well as all acquired resources. This makes abortion of transaction impossible. Hence, a compensating transaction must be executed to revert all the changes, using *compensation rules*, to restore the semantics of database before executing the original subtransaction, thus achieving *semantic atomicity*. If the compensating transaction fails, it is resubmitted. We are not defining the compensation rules as they are out of scope of the paper.

Maintaining autonomy of local sites is primary in Grid environment. Considering that, different sites may employ different protocols for serializability as well. Some sites may employ *locking protocols* while others may employ *timestamping* or *optimistic* concurrency control strategy at local sites. Thus, in presence of such an

autonomous and heterogeneous environment in Grids and absence of a top-layer management system it may be impossible to avoid cascading aborts. The proposed *sleep* state restricts the number of cascading aborts. We would also like to highlight that the *sleep* state does not interfere with the autonomy of the local sites. Implementing this state does not need any modification in local *transaction manager* module. Whenever the site decides to join the Grid, the *sleep* state may be defined in the interface and hence no changes are required in any local modules.

We briefly discuss the time and message complexity of the proposed algorithm. Grid-ACP needs 2 rounds (*time complexity*) of message under normal conditions: (1) after the local sites decide to commit/abort (2) the decision from the originator. Maximum number of messages required is 2*n* (*message complexity*) to reach a consistent decision under normal conditions i.e. without failure. Where *n* is the number of participants in ACP. Considering that originator sends the final decision to all the sites, the number of messages in each round is *n*.

4.2 Correctness of Proposed Protocol

We show the correctness of our ACP by following lemma:

Lemma 1: All participating sites reach the same final decision.

Proof*:* We prove this lemma in two parts, part-I for consistent commit and part-II for consistent abort.

Part I: In this part we show that when the global decision is to commit, all participant commits. From step-2 of the algorithm it is clear that the participants execute autonomously. If local decision is to commit, the information is logged in the stable storage and the subtransaction goes in *sleep* state after sending a message to the originator. If the originator of the transaction finds all commit decision in response, it sends the final *commit* to all participants. In this case the participant is not required to do any action as all resources were already released when the participant entered the *sleep* state. Participant just has to mark the migration of state from *sleep* to *commit*.

Part II: The participants have to do more to achieve this part. In this part we show that if the global decision is abort all participants decides to abort. All participants that decided to commit now receives abort decision from the originator. Those participants decided to abort have already decided to abort unilaterally. Those subtransactions that decided to commit, have already released locks on data items and cannot be aborted. Hence, compensating transactions are constructed using the *event-condition-action* or the *compensation rules*. These compensating transactions are then executed to achieve the semantic atomicity (step-4b of the algorithm). To achieve semantic atomicity the compensating transaction must commit. If the compensating transaction aborts for some reason it is re-executed until it commits. The compensating transaction has to eventually commit, as it is a logical inverse of a committed transaction. This is shown in the state diagram by self-referring compensate state and *line-10* of the participant's algorithm. Though the compensating transaction commits, the semantic of the subtransaction is abort. Thus all participants terminate with consistent decision. ■

5 Conclusion

We have seen that ACP proposed or homogeneous DBMS e.g. 2PC is not suitable for autonomously evolved heterogeneous Grid databases. Strategies for traditional heterogeneous DBMS like multidatabase management system are too restrictive and need a global management system. We have proposed an ACP to meet Grid database requirements that uses *sleep* state for participating sites. The proposed *sleep* state will also help in putting a cap on the number of aborting transactions. We also demonstrated correctness of the proposed protocol. In future we intend to quantify and optimize the capping values of the protocol.

References

[1] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, S. Tuecke, "The Data Grid: Towards an architecture for the Distributed Management and Analysis of Large Scientific Datasets", *Journal of Network and Computer Applications*, vol. 23, pp 187-200, '01

[2] H. Stockinger, "Distributed Database Management Systems and the Data Grid", *18th IEEE Symposium on Mass Storage Systems* '01.

[3] T,Ozsu, P.Valduriez, "*Distributed and Parallel Database Systems*", *ACM Computing Surveys,* vol.28, no.1, pp 125-128, March '96.

[4] P. A. Bernstein, V. Hadzilacos, N. Goodman, *Concurrency Control and Recovery in Database Systems*, Addision-Wesley, 1987.

[5] K. Barker, "Transaction Management on Multidatabase Systems", PhD thesis, Department of Computer Science, The university of Alberta, Canada, 1990.

[6] P. Muth, T. C. Rakow, "Atomic Commitment for Integrated Database Systems", *Proceedings of IEEE, 7th Intl. Conference on Data Engineering*, pp 296-304, 1991.

[7] I. Foster, C. Kesselman, S.Tuecke, "The Anatomy of the Grid", *International Journal of Supercomputer Applications*, vol. 15, no. 3, 2001.

[8] I. Foster, C. Kesselman, J. M. Nick, S.Tuecke, "The Physiology of the Grid", http://www.globus.org/research/papers/ogsa.pdf.

[9] Y. Breitbart, H. Garcia-Molina, and A. Silberschatz. "*Overview of multidatabase transaction management*". VLDB Journal, vol. 1, no. 2, pp.181-240, 1992.

[10] P. Watson, "*Databases and the Grid*", Technical Report, CS-TR-755, Uiversity of New Castle, 2002

[11] IBM, Microsoft, BEA, "*Web Services Coordination*" ftp://www6.software.ibm.com/software/developer/library/ws-coordination.pdf, Sept. 2003.

[12] IBM, Microsoft, BEA, "*Web Services Transaction*" http://www.ibm.com/developerworks/library/ws-transpec/, August 2002.

[13] M. P. Atkinson, V. Dialani, L. Guy, I. Narang, N. W. Paton, D. Pearson, T. Storey, P. Watson, "Grid Database Access and Integration: Requirements and Functionalities" *Global Grid Forum, DAIS-Working Group*, Informational Document, 2003.

[14] M.T. Ozsu and P. Valduriez, editors. *Principles of Distributed Database Systems* (Second Edition). Prentice-Hall, 1999.

[15] Baber Home Page, http://www.slac.stanford.edu/BFROOT/

[16] http://www.griphyn.org/

[17] S. Goel, H. Sharda, D. Taniar, "Preserving Data Consistency in Grid Databases with Multiple Transactions", *2nd International Workshop on Grid and Cooperative Computing (GCC '03), Lecture Notes in Computer Science*, Springer-Verlag, China, Dec. 2003.