Preserving Data Consistency in Grid Databases with Multiple Transactions

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

[1] School of Electrical and Computer 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. High performance Grid computing provides an infrastructure for access and processing of large volume, terabyte or even petabytes, of distributed data. Research in data grid has focused on security issues, resource ownership, infrastructure development and replication issues assuming presence of single transaction in the system. In this paper we highlight that grid infrastructure comes with new set of problems in maintaining the consistency of databases in presence of multiple transactions. Traditional distributed data management techniques may not meet the requirements of databases in grid environment. We first show the circumstances where grid infrastructure may produce incorrect results and then propose a correctness condition – *Grid Serializability Criterion* that preserves the consistency of data in data-grids.

1 Introduction

Grid computing can be defined as a type of parallel and distributed system that enables the sharing, selection and aggregation of geographically distributed autonomous resources dynamically at runtime depending on their availability, performance, cost and users Quality of Service [9]. Following the grid infrastructure the need for data grid [3] was realized. Intension behind developing data grid was to access and process geographically distributed data in computationally effective way. Handling distributed data has many research issues like scheduling of transactions, query execution, maintaining consistency of data etc.

Recent research in data grid has focused mainly on file structures and read-only transactions [2,5,8]. Various replication strategies [1,3,5,6] have also been proposed to maintain the consistency of the data in these Network File System (NFS) and Distributed File System (DFS). Many applications need to modify the distant data, but with the replication strategy it may be difficult to maintain the correctness of the data in the data repository.

Sometimes data consistency is referred as keeping the replicated data items synchronized [5]. This definition of data consistency assumes a single transaction in the system. We model the scenario where multiple transactions are in action and the transactions can be any combinations of read/write operations. In this paper we always refer consistency as the correctness of data in the data repository, we also use correctness and consistency interchangeably. Thus, we focus on interleaving of multiple transactions; the concept of interleaving of transactions is known as serializability [10] in database systems. Based on centralized serializability, we name the correctness criterion in grid environment as *Grid Serializability Criterion* (GSC). The motive behind modelling multiple transactions with any read/write combination would be clear in the following sections where we use an example to demonstrate that under this scenario the data grid may produce incorrect result.

Large-scale science and engineering problems need to access geographically separated data. Weather forecast, astronomical predictions, earth observation and applications that need collaborative analysis will have to retrieve data from different data sources. Traditional file systems like NFS and DFS, or Database Management Systems (DBMS) [11] have either of two strategies to manage transactions [9] (a) *Central scheduling scheme* (b) *Decentralized consensus-based scheme*. The central scheduler may become performance bottleneck due to central overloaded site by using these two scheduling strategy.

Motivation behind this research of decentralized processing of data is that the enormous amount of data that today's application has to handle, it is difficult to ship the data between sites and at the same time it is computationally very expensive to have a centralized scheduler. Secondly, centralized scheme and decentralized consensus based scheme may be suitable for read-only transactions. But, if transaction of any application has to modify any data source in the distributed data repository then the central scheduler can become a performance bottleneck.

We show that new set of problems come into existence with introduction of distributed schedulers in grid environment. One of the major issues in decentralizing the scheduling responsibilities is the threat to maintain the consistency of the data. This paper looks at the future data-intensive grid applications from data correctness point of view. Section 2 gives an overview of data-grids. Section 3 discusses the new dimension of problem that grid infrastructure introduces in multi-transaction environment. Section 4 explains Grid Serializability Criterion, section 5 shows the correctness of the algorithm. Section 6 concludes the work with future directions.

2 Database on Grid: An Overview

Most of the work done in data grid infrastructure assumes the existence of file systems like NFS and DFS for data storage, they also assume the execution of single transaction at a given time [6,7,5,8]. NFS and DFS are not designed to meet the stringent requirements of high performance data intensive applications [7].

2.1 Data-Grid

With increasing diversity of scientific disciplines, amount of data collected is increasing. In domains as diverse as global climate change, high-energy physics and computational genomics the volume of data is already being measured in terabytes and will soon be measured in petabytes [3]. It becomes increasingly difficult to handle this volume of data if the data itself is geographically distributed. Grids will enable ubiquitous access to data and data-collection systems spread across the globe.

Two major challenges for data-intensive applications would be to adapt the distributed data-scheduling environment and to keep the number of messages in the system to minimum while maintaining the correctness of data. Under these circumstances we need to integrate end systems in a more effective manner using new hardware and software infrastructure.

Several design questions have been identified to meet the requirements for this architecture like, How to provide metadata information that describes data set's location? What is the effect of caching of data in performance of the application? How replication affects the performance? What happens if the total amount of data is larger than the transmission capacity of the grid? How to synchronise the remote copies of data? What are the interfacing techniques between different databases?

Past research [1,4,5,6,8] has focused on these issues. These issues assume only single transaction is executing in the system which is read-only and the data is stored in file structure. Our main concern in this paper is to deal with transactions that tend to modify the data items in data repository with multiple transaction execution.

2.2 Data-Grid Architecture

Fig. 1. shows a general architecture of data intensive application where data is collected and stored at multiple remote sites. Initial research in grid computing was focused on management of computational resources. Most of the research was limited to file I/O operations, management of file caches, security and replication issues [4,12,6]. Lately researchers realized the importance of integrating data sites with grid infrastructure [3]. Replication and cache management issue assumes the read-only environment in the data intensive systems [3], which is not always true.

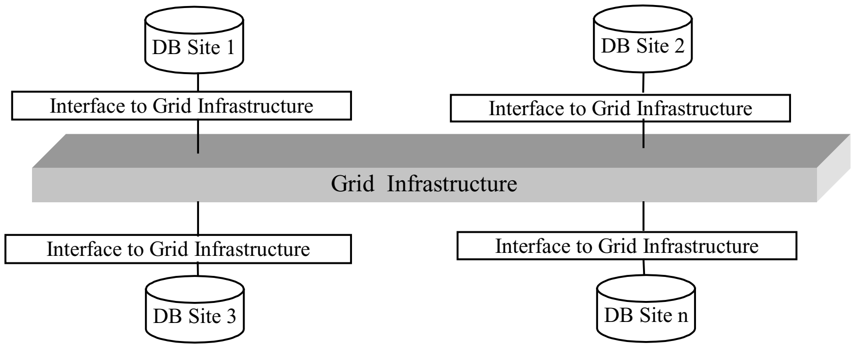
DB Site 1
DB Site 2
Interface to Grid Infrastructure
Interface to Grid Infrastructure
Grid Infrastructure
Interface to Grid Infrastructure
Interface to Grid Infrastructure
DB Site 3
DB Site n

Fig. 1. Geographically separate databases using grid infrastructure

Traditional NFS provides access to remote data with uniform data namespace but because of lack of replication and batch I/O cannot have good performance. Parallel database systems like bubba, gamma [11] can provide collective I / O but are not designed to meet the distributed nature of grid computing [7]. Present database systems need central (or, consensus based decentralized) coordinator to achieve correctness of data, widely known and accepted as database serializability [10]. The centralized serializability criterion has to be extended and should evolve to such an extent that it could be implemented in grid infrastructure. With the database applications we are expecting to evolve in near future and the amount of data that will be generated, petabytes of data per year, it is impossible to use any of the present serializability criterions that could maintain the correctness of data in grid database systems environment.

3 Data Consistency in Multi-transaction Environment

Major research interest in grid computing has been towards developing fast-interconnected networks, high throughput applications, moving data efficiently between sites, developing fast and scalable applications [1,3,5,6,8,]. Accepting the importance of these infrastructural developments, in this section we show the importance of maintaining the consistency of the database under multi-transaction environment that intend to modify the data.

Correctness criterion for read-only transactions is different than transactions that modify the data. With application size spanning to global level it becomes difficult to manage central coordinator and possibilities of mutually independent scheduling techniques have to be explored. Distributing the scheduling responsibilities to local database sites may pose a threat to the consistency of data. We explain the problem by following example.

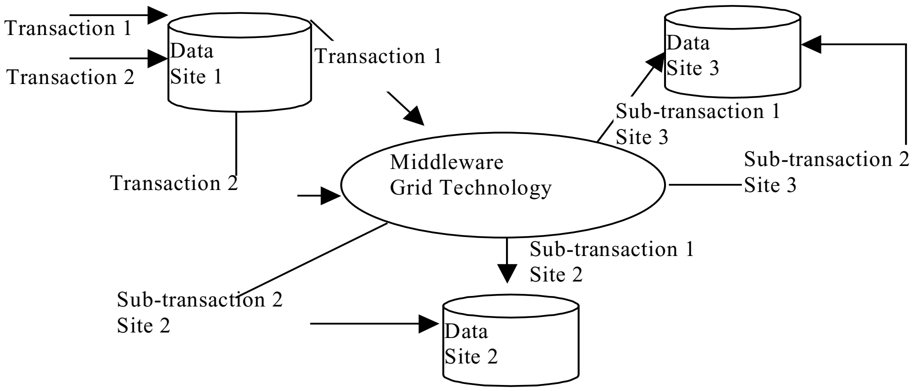

Fig. 2. Consistency of data in distributed data set

Without loss of generality we assume a scenario where an application has to access data from three different data sites – data-site 1, data-site 2 and data-site 3 (see fig 2). These three sites are connected by the state-of-the-art grid technology middleware and can communicate between themselves by high bandwidth network connections.

Let us assume that two *transactions*, *transaction 1 and transaction 2,* are submitted to *data-site 1* which needs to access data from other two data-sites as well, *data-site 2* and *data-site 3*. By using the metadata information the database system at site 1 can form two subtransactions for each of the transaction i.e. *Sub-transaction 1 Site 2, Sub-transaction 1 Site 3* and *Sub-transaction 2 Site 2, Sub-transaction 2 Site 3.*

And then submit the subtransactions to the respective data-sites. If the transaction is read-only transaction it does not pose any threat to consistency but if the transaction intend to modify any data item it must synchronize the access to the data items by using semaphores or locks. By synchronization we mean, if *Sub-transaction 1 Site 2* precedes its execution to *Sub-transaction 2 Site 2* at site 2 then *Sub-transaction 1 Site 3* must also precede its execution to *Sub-transaction 2 Site 3* at site 3.

Under these circumstances, delegating scheduling responsibilities to individual data-sites may produce incorrect interleaving. Thus we see that there is a genuine problem of scheduling that may be menace to correctness and consistency of the data.

4 Proposed Grid Serializability Criterion

In this section we propose a *Grid Serializability Criterion* (GSC) that enforces a *total-order* in the schedule to ensure correctness of data in multi-transaction environment. We name the correctness criteria with the word serializability in between to correlate with the correctness criteria already existing in single DBMSs, conflict serializability [10]. Total-order is required only for those transactions (T_i) that accesses more than one data-site simultaneously and is implemented by using a unique timestamp value.

4.1 GSC Algorithms

Following functions are used to demonstrate the algorithm: *Split_trans*(T_i) returns set of subtransactions that accesses different data-sites, *Site_accessed*(T_i) returns the set of data-sites where the subtransactions for T_i are to be executed. *Cardinality()* returns the number of elements in the set. *Append_TS(Subtransaction)* appends *timestamp* to the subtransaction.

We assume that architecture is capable of producing unique timestamp values.

Phase I: The transaction starts execution (algorithm 1).

1. As soon as the transaction arrives at the local database, *split_trans*(T_i) splits the transaction into multiple subtransactions according to the allocation of data.
2. If there is only one subtransaction required by the transaction, the transaction can be submitted to the data-site immediately without any delay. The transaction with one subtransaction will not conflict with other transactions.
3. If multiple subtransactions are required by the transaction, the grid infrastructure appends a timestamp with every subtransaction before submitting it to the corresponding data-site.
4. If there is a transaction that access more than one data-site then the subtransactions are submitted to the data-site's local scheduler. The subtransactions from the scheduler are executed strictly according to the *ts*.
5. Subtransactions from *step-2* can be assumed to have lowest timestamp value e.g. 0 and can be scheduled immediately.
6. When all subtransactions of any transaction T_i, complete the execution at all the sites, only then T_i commits (algorithm 2).

Working of the algorithm is explained below:

Algorithm for Grid Serializability Criterion during submission of transaction

```
begin
    input $T_i$ : Transaction
    generate timestamp $ts$ : unique timestamp is generated
    $Split\_trans(T_i)$
    $Site\_accessed(T_i)$ $\leftarrow$ set of sites accessed
    if $Cardinality(Site\_accessed(T_i)) \leq 1$ then
        begin
            submit subtransaction to PE
        end
    else begin
        for each subtransaction of $T_i$
            $Append\_TS(Subtransaction)$
        end for
        for each $site_k \in site\_accessed(T_i)$
            begin
                $Active\_Trans(Site_k) \leftarrow Active\_Trans(Site_k) \bigcup T_i$
                submit subtransaction to $site_k$ timestamp queue.
            end
        end for
    end
    end if
end
```

Phase II: Termination condition for transaction.

1. When any subtransaction finishes execution, it reports to the originating site.
2. The originating site checks whether the subtransaction is the last subtransaction to terminate.

2a. If the subtransaction is not the last to terminate, then that site is removed from the *site_accessed()* set.

2b. If the subtransaction is the last subtransaction of the transaction to terminate, then that transaction is removed from the *Active_Trans()* set.

We intend to achieve two major advantages by using the Global Serializability Criterion that uses a decentralized scheduling approach:

Reducing the load from the originating site: Cental scheduling scheme and decentralized consensus bases policies intend to delegate the originating site of the transaction as the coordinator. The proposed criterion delegates the scheduling responsibility to the respective sites where the data resides and thus avoids the originating site becoming the bottleneck.

Reducing the number of messages in the inter-network: Centralized and consensus based decentralized scheduling schemes need to communicate with the coordinator to achieve correct schedules. The communication increases number of messages in the system. Messages are one of the most expensive things to handle in any distributed infrastructure. The algorithm intends to reduce the number of messages in the system.

5 Correctness of the Algorithm

We assume that the local databases are capable of guarantying correctness of data, stated in proposition 1, but as discussed earlier there is a need for additional strictness in grid infrastructure to maintain the correctness of data. We achieve this additional strictness by applying the concept of total-order, which in turn is achieved by using timestamps. In this section we show that GSC preserves the correctness and consistency of data in the grid.

Proposition 1: All local database sites always schedule all transactions in correct order. ■

To prove the correctness of the proposed algorithm we show that the additional criterion enforced by the total-order will guarantee GSC in grid systems. The total-order enforced only determines the way subtransactions are submitted to the sites. If any transaction accesses more than one geographically separated database then the subtransactions cannot be scheduled immediately and thus have to be submitted to *wait_queue* (Proposition-2). Each site's *queue* schedules the subtransactions according to their timestamp and thus guarantying the total-order of the transaction from global perspective.

Proposition 2: The originator site submits the subtransactions to the *wait_queue* of site's being accessed by that transaction, if transactions access more than one mutually exclusive database. ■

Transactions that access more than one local databases can produce incorrect scheduling of subtransactions, Proposition-2 ensures that subtransactions are executed according to total-order and thus guarantee correctness of data in data-grids.

Lemma 1: For any two transactions T_i, T_j that follows the Grid Serializability Criterion, either all of T_i*'s* subtransactions are executed before T_j at all the sites where the subtransactions execute or vice versa.

Proof: Following two cases are to be considered:

Case 1) *A transaction T_i requires only single subtransaction:* This situation is shown by ***if*** condition of the algorithm. The subtransaction is submitted immediately as shown in the algorithm flow chart. From Poposition-1 it follows that any other subtransaction $\in T_j$ either precedes or follows the subtransaction of T_i. As we have seen earlier that the transaction with only one subtransaction cannot produce incorrect interleaving.

Case 2) *A transaction T_i splits into multiple subtransactions:* This situation is shown by ***else*** condition of the algorithm. Under this condition schedulers at sites may schedule the transactions that can produce incorrect interleaving. Hence, transactions in the sites are submitted in the *wait_queue* instead of scheduling it immediately (Proposition-2), which executes transactions strictly according to the timestamp order.

Say, transaction T_i has two subtransactions T_{i1} and T_{i2} already executing at $site_1$ and $site_2$. When T_j arrives and it also has T_{j1} and T_{j2}. Then, ***if*** condition of the algorithm fails and the subtransactions are submitted to the *wait_queue* at each PE.

Assume that the timestamp of T_i, TS(T_i) $\prec$ TS(T_j). A unique timestamp is appended to the subtransactions of T_i and T_j. Then T_i will precede T_j at both the sites because the transactions are scheduled strictly according to the timestamp value avoiding execution of incorrect schedules and thus ensuring the total-order of transactions. Thus GSC avoids any incorrect interleaving of subtransactions. ■

6 Conclusion

We have seen that despite a lot of infrastructural development for deploying grid computing and making it a reality, data-intensive applications and maintaining correctness of data in such applications have been neglected. In this paper we first show that high performance grid infrastructure may pose threat to the consistency of data in data-intensive applications that implement existing correctness criterions. We then propose a correctness criterion based on total-order of subtransactions that ensures the correctness of data. Finally, we demonstrate the correctness of the proposed criterion. Present work does not take failure of sites into consideration. Future work would focus in extending the algorithm to encounter local site failures.

References

[1] W. Hoschek, J. J., Martinez, A. S. Samar, H. Stockinger, and K. Stockinger. "Data management in an international data grid project." *ACM Workshop on Grid Computing (GRID-00)*, 17-20 Dec., pp 77-90, India, '00

[2] B. Allcock, I. Foster, V. Nefedova, A. Chervenak, E. Deelman, C. Kesselman, J. Leigh, A. Sim, A. Shoshani, B. Drach, D. Williams, "High-Performance Remote Access to Climate Simulation Data: A Challenge Problem for Data Grid Technologies." *Proc. of SC,* '01.

[3] 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

[4] C. Baru, R. Moore, A. Rajasekar, M. Wan, "The SDSC storage Resource broker", *in Proceedings of CASCON '98 Conference,* '98.

[5] D. Dullmann, W. Hosckek, J. Jaen-Martinez, and B. Segal, A.Samar, H. Stockinger, K. Stockinger, "Models for Replica Synchronization and Consistency in a Data Grid," *Proc. IEEE Symposium. on High Performance on Distr. Computing*, '01.

[6] I. Foster, C. Kesselman, C. Lee, R. Lindell, K. Nahrstedt, A. Roy, "A Distributed Resource Management Architecture that Supports Advance Reservations and Co-allocation", *in Proceedings of the International Workshop on Quality of Service,* pp 27-36, '99.

[7] I. Foster and C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit", *International Journal of Supercomputer Applications*, vol 11(2), pp 115-128, '97.

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

[9] R. Buyya, "Economic-based Distributed Resource Management and Scheduling for Grid Computing", PhD thesis, Monash University, Australia, '02.

[10] P. A. Bernstein, V. Hadzilacos, N. Goodman, *Concurrency Control and Recovery in Database Systems*, Addision-Wesley, '87.

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

[12] I. Foster, C.Kesselman, G.Tsudik, S.Tuecke, "A Security Architecture for Computational Grids", *ACM Conf. on Computers and Security,* pp 83-91, ACM Press, '98.