A Real-Time Transaction Approach for Grid Services: a Model and Algorithms

Feilong Tang[1], Minglu Li[1], Joshua Zhexue Huang[2], Lei Cao[1], and Yi Wang[1]

[1] Department of Computer Science and Engineering,
Shanghai Jiao Tong University, Shanghai 200030, China
{tang-fl,li-ml}@cs.sjtu.edu.cn
[2] E-Business Technology Institute, The University of Hong Kong, China
jhuang@eti.hku.hk

Abstract. Because transactions in Grid applications often have deadlines, effectively processing real-time transactions in Grid services presents a challenging task. Although real-time transaction techniques have been well studied in databases, they can not be directly applied to the Grid applications due to the characteristics of Grid services. In this paper[3], we propose an effective model and corresponding coordination algorithms to handle real-time transactions for Grid services. The model can intelligently discover required Grid services to process specified sub-transactions at runtime, and invoke the algorithms to coordinate these services to satisfy the transactional and real-time requirements, without users involvement in the complex process. We use a Petri net to validate the model and algorithms.

1 Introduction

One objective of developing a service Grid is to provide users with transparent services and hide the complex process from them. The technology for processing a real-time transaction is a key to determine whether the service Grid can be widely accepted in commercial use because many Grid applications have time restrictions and transactional requirements. The real-time transaction for Grid services [1] differs from conventional real-time database transactions because (a) Grid services are loosely coupled, and (b) Grid services can dynamically join and leave the Grid. Therefore, it is important to investigate the real-time transaction technology in the Grid service environment.

The deadline of a real-time transaction specifies the time by which the transaction must complete or else undesirable results may occur. Based on the strictness of deadlines, real-time transactions for Grid services can be classified into three types, similar to those in the traditional distributed systems [2].

– Hard real-time transaction. This is the strictest real-time transaction. If these transactions miss their deadlines, there are catastrophic consequences.

[3] This paper is supported by 973 project of China(No.2002CB312002), and grand project of the Science and Technology Commission of Shanghai Municipality(No.03dz15027).

- Firm real-time transaction. It is of no value to complete a firm real-time transaction after its deadline but catastrophic results will not occur if a firm real-time transaction misses its deadline.
- Soft real-time transaction. Satisfaction of the deadline is primarily the performance goal. Unlike a firm real-time transaction, however, there still are some benefits for completing a soft real-time transaction after its deadline.

In this paper, we focus on the soft and firm real-time transactions. Our motivation is to provide a model, with the key component of the real-time transaction service (GridRTS) so application programmers can use GridRTS to easily materialize real-time applications.

2 Related Work

To process a transaction in a distributed environment, a common agreement is generally achieved by negotiations between a coordinator and the participants. DTP(Distributed Transaction Processing)[3] is a widely accepted model in distributed transaction processing. It defines three kinds of roles (Application Program, Transaction Manager and Resource Manager) and two kinds of interfaces (TX interface between Application Program and Transaction Manager, and XA interface between Transaction Manager and Resource Manager). However, DTP does not support the real-time transaction.

The real-time transaction schemes have heavily been researched in the database area. Abbott [4] presented a new group of algorithms for scheduling real-time transactions that produce serializable schedules. A model was proposed for scheduling transactions with deadlines on a single processor disk resident database system. The scheduling algorithms have four components: a policy for managing overloads, a policy for assigning priorities to tasks, a concurrency control mechanism, and a policy for scheduling I/O requests. Some real-time transaction scheduling algotirhms were proposed in [5], which employ a hybrid approach, i.e., a combination of both pessimistic and optimistic approaches. These protocols make use of a new conflict resolution scheme called dynamic adjustment of serialization order, which supports priority-driven scheduling, and avoids unnecessary aborts.

This paper extends these previous results to the Grid service environment by providing the GridRTS with a set of interfaces for Grid application programmers.

3 Real-Time Transaction Model

The real-time transaction model we present here is based on the Globus Toolkit 3. The core component GridRTS, as shown in Fig. 1, consists of the following:

- Service Discovery. It discovers the required Grid services that can complete the sub-tasks for a real-time transaction.

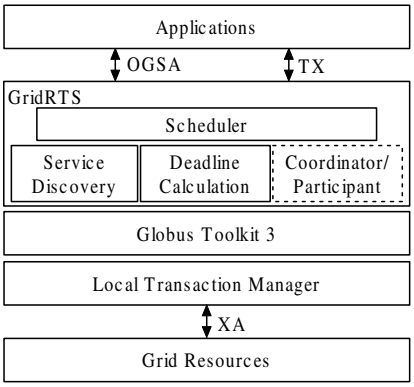


Fig. 1. The real-time Grid transaction model

- Deadline Calculation. It calculates deadlines of (sub)transactions to coordinate these activities.
- Coordinator or Participant. It is dynamically created by the scheduler of GridRTS and lives until the end of a global transaction. The scheduler of GridRTS creates a coordinator or a participant when it receives a request to initiate a global transaction or perform a sub-transaction.
- Scheduler. It takes charge of scheduling above modules.
- Interfaces. The OGSA interfaces are responsible for service management such as creating a transient Grid service instance while the TX interfaces are used to manage transactions.

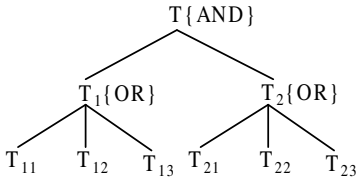


Fig. 2. A simple real-time Grid transaction

Definition 1. A real-time transaction is a 6-tuple={T, D, S, R, DL, P}, where T is the set of sub-transactions and each sub-transaction T_i is completed by several alternative functional services; D is the set of data operated by the real-time transaction; S is the state set; R is the set of relationships between (sub)transactions, defined as R={AND, OR, Before, After}; DL is the set of deadlines; and P is the priority set.

The AND relationship means that all the sub-transactions T_i of a global transaction T must be completed before their deadlines $d(T_i)$, i.e., T=T_1 AND T_2 AND... AND T_m. Each sub-transaction T_i is performed by n alternative functional services. The OR relationship means that T_i is completed if any T_{ij} finishes before the deadline of T_i, i.e., $T_i = T_{i1}$ OR T_{i2} OR ... OR T_{in} (see Fig. 2). Before and After specify the execution order between sub-transactions.

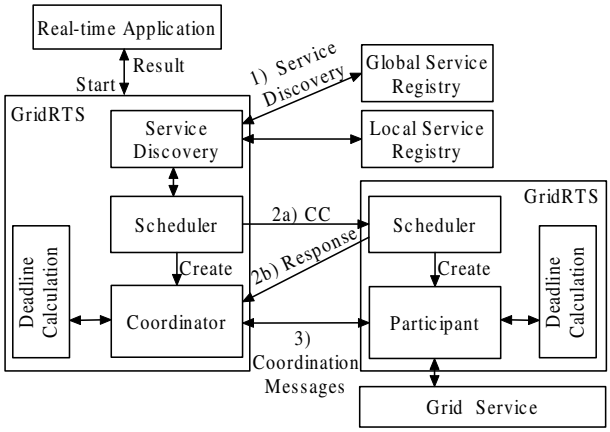


Fig. 3. The execution flow of the real-time Grid transaction.

4 Coordination of the Real-Time Grid Transaction

4.1 The Process of the Real-Time Grid Transaction

In the Grid service environment, a typical real-time transaction includes the following steps, as shown in Fig. 3.

- Step 1: GridRTS initiates a global transaction on behalf of a real-time Grid application, discovers and selects required Grid services to serve as participants, using the service discovery module as described in [6].
- Step 2: The scheduler creates a coordinator and broadcasts the Coordination-Context (CC) messages to all selected remote participants, that are created locally and return Response messages to the coordinator.
- Step 3: The created coordinator and participants interact to control the transaction execution, including the correct completion and failure recovery. The detail is described in the following subsection.

4.2 Coordination Algorithms

As described above, a sub-transaction is completed by a set of alternative functional services from an alternative functional service group (AFSG). The members of the AFSG execute the same sub-task in parallel. If one member of the AFSG can complete successfully and reports a Committable message, the AFSG is considered committable and other members are aborted.

In the preparation phase, each alternative functional service executes a specified sub-task in its private work area (PWA). On receipt of the Abort message, the service rollbacks the operations taken previously, by releasing the PWA. In the commit phase, the Commit message notifies the participants that have reported Committable messages to the coordinator. These participants are called committable participants and actually commit sub-transactions (see Fig. 4).

```
Algorithm of Coordinator
Input:service references of all functional
      alternative services S_i and d(T)
Output:result of T or failure
{
  for all S_i in all AFSGs
    send Prepare messages to them;
  end for
  while (t≤ d(T)){
    wait and record incoming messages;
    for each AFSG
      if (receive a Committable)
        send Abort to others in this AFSG;
    end for
    if (all AFSGs receive Committable)
        send Commit to committable
             Participants;
    else
        send Rollback to them;
} }
```

(a) Coordinator algorithm

```
Algorithm of Participant
Input: d(T_i)
Output: result of T_i or failure
{ while (t≤ d(T_i)) {
    wait & record incoming messages;
    if (receive a Prepare){
      execute sub-task in PWA;
      if (successfully)
        report Committable;
      else { report Uncommittable;
             rollback; } }
    Case (receive a message)
      Commit: {
        actually commit sub-transaction;
        send Committed; }
      Abort:
        rollback and send Aborted;
      Rollback:
        rollback and send Rollbacked;
    EndCase } }
```

(b) Participant algorithm

Fig. 4. Coordination algorithms of the real-time Grid transaction

Fig. 5 illustrates the state transformation diagram of the real-time Grid transaction. The solid rectangles indicate the states of both the coordinator and participants. The Dashed rectangle denotes the state of participants. The transaction enters Prepared state only when the coordinator receives a Committable message from each AFSG before the deadline d(T). Otherwise, the coordinator sends Rollback messages to undo the effect produced by the previous operations.

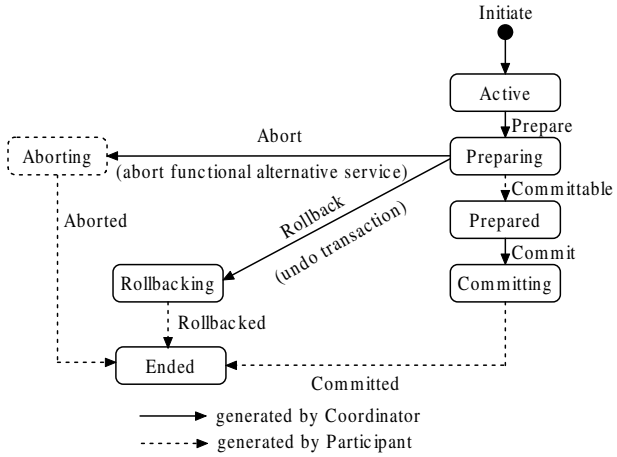


Fig. 5. The state transformation diagram of the real-time Grid transaction

5 Algorithm Validation

5.1 Modeling Algorithms with Petri Net

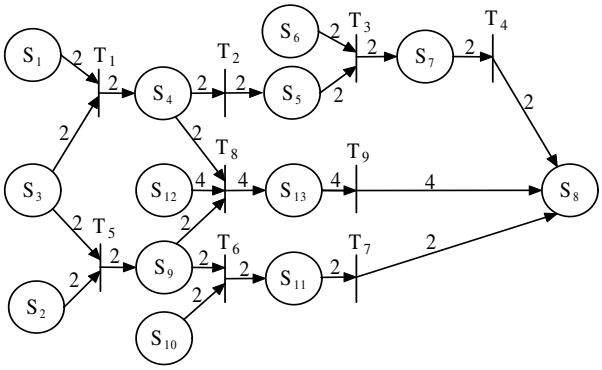
A Petri net is an abstract and formal modelling tool. It models systems' events, conditions and the relationships among them. The occurrence of these events may change the state of the system, causing some previous conditions to cease holding and other conditions to begin to hold [8]. In this work we model the coordination algorithms with a Petri net to verify their correctness. In the model, the transitions indicate actions taken by participants, and the places represent the states of the coordinator and/or participants or the receipt of the coordination messages from the coordinator.

Assume that a real-time Grid transaction consists of two sub-transactions and each sub-transaction is completed by two Grid services. We use P_{I1}, P_{I2} and P_{II1}, P_{II2} to represent the first and second AFSGs respectively. Without losing the generality, we let P_{I1} and P_{II1} first return Committable messages and finally commit while P_{I2} and P_{II2} are aborted. The Petri Net model RTPNM of this real-time transaction is depicted in Fig.6, where P_{I1} and P_{II1} are illustrated by S_1, and P_{I2} and P_{II2} by S_2. The weights of the arcs indicate the number of changed tokens whenever a firing happens (i.e. added or removed).

5.2 Analysis of the RTPNM

Let M=(M_1, M_2,..., M_{13}) be a marking, where M_i is the number of tokens in place S_i. The RTPNM has two initial markings:

– M_{0s}=(2,2,4,0,0,2,0,0,0,2,0,0,0), when P_{I1} and P_{II1} commit while P_{I2} and P_{II2} are aborted, and



S_1: $P_{I1,II1}$-Active S_2: $P_{I2,II2}$-Active S_3: C-Prepare
S_4: $P_{I1,II1}$-Preparing S_5: $P_{I1,II1}$-Prepared S_6: C-Commit
S_7: $P_{I1,II1}$-Committing S_8: P-Ended,C-Ended S_9: $P_{I2,II2}$-Preparing
S_{10}: C-Abort S_{11}: $P_{I1,II1}$-Aborting S_{12}: C-Rollback
S_{13}: $P_{I1,II1,I2,II2}$-Rollbacking T_1: $P_{I1,II1}$ prepare for commit
T_2: $P_{I1,II1}$ return Committable T_3: $P_{I1,II1}$ commit
T_4: $P_{I1,II1}$ return Committed T_5: $P_{I2,II2}$ prepare for commit
T_6: $P_{I2,II2}$ abort T_7: $P_{I2,II2}$ return Aborted
T_8: $P_{I1,I2,II1,II2}$ rollback T_9: $P_{I1,I2,II1,II2}$ return Rollbacked

Fig. 6. The Petri net model of the real-time Grid transaction(RTPNM)

– M_{0f}=(2,2,4,0,0,0,0,0,0,0,0,4,0), when at least one AFSG can not prepare for commit, resulting in all four services are rolled back.

The Petri net model can analyze the behavioral properties, which depend on the initial marking, including reachability, boundedness, liveness, coverability, reversibility, persistence and so on. For a bounded Petri net, however, the coverability tree is called the reachability tree and all above problems can be solved by the reachability tree [7]. Peterson [8] also pointed out that in Petri nets, many questions can often be reduced to the reachability problem. In this paper, we focus on the boundedness and reachability using the reachability tree, which is not illustrated here because it is too large. By analysis of the reachability tree of the RTPNM, we can draw following conclusions.

Theorem 1. RTPNM is bounded.

Proof: A Petri net (N, M_0) is said to be k-bounded or simply bounded if the number of tokens in each place does not exceed a finite number k for any marking reachable from the initial marking, i.e., $M_i \leq k$ for every place S_i and every marking $M \in R(M_0)$[7], where M_0 is an initial marking and $R(M_0)$ is the set of all possible markings reachable from the M_0. By inspection of the reachability tree of the RTPNM, we have found that ω (represent an arbitrarily large value) does not occur anywhere, and the number of tokens in each place is no more than 4. Therefore, the RTPNM is bounded and k is 4.

Theorem 2. RTPNM is L1-live.

Proof: A transition is L1-live if it can be fired at least once in some firing sequences. A Petri net is L1-live if all its transitions are L1-live. For a bounded Petri net, the reachability tree contains all possible markings. After inspecting the reachability tree of the bounded RTPNM, we have found that every marking is reachable and every transition T_i $(1 \leq i \leq 9)$ can be fired at least once from M_{0s} or M_{0f}. Therefore, the RTPNM is L1-live.

Theorem 2 indicates that the RTPNM is a deadlock-free as long as the firing starts with M_{0s} or M_{0f}. Therefore, the coordination algorithms are correct.

6 Conclusions and Future Works

We have presented a real-time Grid transaction model. Its core component GridRTS can intelligently discover required Grid services as participants to perform specified sub-transactions, and coordinate multiple Grid services to achieve real-time and transactional properties. Using the Petri net tool, moreover, we have validated the correctness of the coordination algorithms, whether a real-time transaction is successful, starting with M_{0s}, or failed, beginning with M_{0f}.

In our future work, we will add a security mechanism to enable it to adapt to the actual commercial environment.

References

1. I. Foster, C. Kesselman, J. M. Nick and S. Tuecke. The Physiology of the Grid-An Open Grid Services Architecture for Distributed Systems Integration. June, 2002. http://www.globus.org/research/papers/ogsa.pdf.
2. E. Kayan, O . Ulusoy. Real-Time Transaction Management in Mobile Computing Systems. Proceedings of the Sixth International Conference on Database Systems for Advanced Applications. April, 1999, pp. 127-134.
3. I. C. Jeong and Y. C. Lew, DCE (Distributed Computing Environment) based DTP (Distributed Transaction Processing). Proceedings of the 12th International Conference on Information Networking. January, 1998.
4. R. K. Abbott. Scheduling Real-Time Transactions: A Performance valuation. ACM Transactions on Database Systems, Vol 17. No. 3, 1992, pp. 513-560.
5. S. H. Son and J. Lee. A New Approach to Real-Time Transaction Scheduling. Proceedings of Fourth Euromicro workshop on Real-Time Systems, June , 1992, pp. 177-182.
6. F. L. Tang, M. L Li, J. Cao et al. GSPD: A Middleware That Supports Publication and Discovery of Grid Services. Proceedings of the Second International Workshop on Grid and Cooperative Computing. December, 2003, pp. 738-745.
7. T. Murata. Petri Nets: Properties, Analysis and Applications. Proceedings of the IEEE, Vol 77, No. 4, 1989, pp. 541-580.
8. J. L. Peterson. Petri Nets. Computing Surveys, Vol. 9, No. 3, September, 1977, pp. 223-252.