A Transaction Model for Grid Computing

Feilong Tang, Minglu Li, and Jian Cao

Department of Computer Science & Engineering,
Shanghai Jiao Tong University
Shanghai 200030, P.R. China
{tang-fl,li-ml,cao-jian}@cs.sjtu.edu.cn

Abstract. In Grid environment, transaction must have the abilities to coordinate both short-lived operations and long-lived business activities, which can not directly be supported by traditional distributed transaction mechanisms. This paper introduces a transaction model based on agent technologies to satisfy these requirements. The model can coordinate both Atomic transaction and Compensation transaction. The agents shield users from complex transaction process. For two types of transactions, we respectively analyze the coordination mechanisms and provide corresponding algorithms. We also discuss the implementation of this model and propose a security solution.

1 Introduction

The main goal of Grid computing is sharing large-scale resources, accomplishing collaborative tasks [1], where transaction will become more and more important. To date, however, few efforts have been performed for transaction in Grid environment.

In this paper, we present a transaction model for Grid computing. The contributions mainly include: (a) the architecture and algorithms for transaction in Grid environment; (b) abilities for users to execute transaction, without knowing of process.

2 Related Work

There have been some efforts about distributed transaction. [2,3] and [4], proposed by IBM, Microsoft, BEA and OASIS respectively, describe the transaction framework for Web services. However, they can not directly support for Grid transaction.

3 Architecture of Transaction Based on Agent

The transaction model, as shown in figure 1, is based on the services provided by Globus Toolkit [5,6,7]. What a user need do is only to start up a transaction and make desirable selection because the agent can create a TransactionManager (TM), start up or cancel transaction, manage transaction and inform the user of execution results.

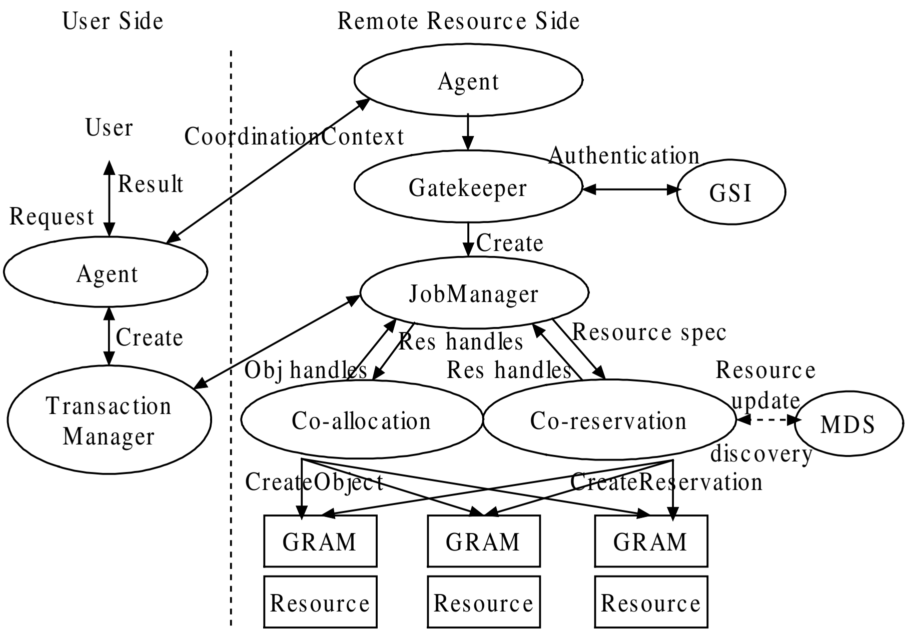

Fig. 1. An extensible transaction architecture for Grid computing

Comparing with transaction for Web services, where each service is statically created, our model adapts to the highly dynamic Grid environment. It can meet grid features and requirements because:

- It does not lock resources so as to improve the concurrency.
- It is robust in face of various failures and dynamic join and leaving of resource.
- It can coordinate both short-lived operations and long-lived business activities.

4 Coordination Mechanism

The model may handle two types of transactions, which can be selected by a user:

- Atomic transaction (AT). It is used to coordinate short-lived operations, where participants must commit synchronously and intermediate results are invisible.
- Compensation transaction (CT). It is used to coordinate long-lived business activities and allows some candidates to abort while others to commit.

4.1 Atomic Transaction Coordination

The agent responds the transaction request from a user by:

- Creating a TM, which serves as the coordinator.
- Sending CoordinationContext messages to all agents of participants.

We adopt reservation of resources to prepare for AT transaction. After receiving the Responses from participants, TM coordinates transaction in following way:

Phase1: TM sends Prepare messages to all participants P_i (i=1, 2, …N). If P_i successfully reserves, the resource handles are returned to JobManager (JM), which reports Prepared message to TM. Otherwise, the NotPrepared message is sent to TM.

Phase2: Only if TM knows that all P_i have successfully reserved, it sends Commit to all P_i, which enables all JMs to request for allocating reserved resources, record the Commit information in log, monitor the execution of tasks and report result to TM. Otherwise, TM sends Abort to all P_i, making them cancel previous reservation. Within T_2, if TM receives N Committed messages, it judges that transaction is completed and then returns final result to user. Otherwise, the TM reports failure to the user and sends Rollback to all P_i. The algorithm is as follows:

```
ActionOfParent
begin agent creates TM;
    agent sends CoordinationContext to
    all agents of participants $P_i$;
    wait for Response from JM of $P_i$;
    if timeout exit;
    send Prepare to all $P_i$;
    while ($t \le T_1$) and (n1<N)
        wait & record incoming messages;
    if (n1=N) and (all messages are Prepared)
    begin record commit in log;
       send Commit to all $P_i$;
       while ($t \le T_2$) and (n2<N)
           wait & record incoming message;
        if (n2<N) or (not N Committed)
        begin send Rollback to all $P_i$;
            exit after receiving Rollbackeds; end
    end else begin send Abort to all $P_i$;
        exit after receiving all Aborted; end
end
```

```
ActionOfChild
begin agent creates JM;
    send Response to TM;
    wait for Prepare from TM;
    if timeout  exit;
    success:=JM reserves resources;
    if (success)
    begin send Prepared to TM;
        while ($t \le T_3$) and (not Commit)
            wait for incoming message;
        if (message is Commit)
        begin allocate reserved resources;
            record transaction in log;
            commit transaction;
            send Committed to TM; end
        else begin
            cancel reservation;
            exit; end
    end
end
```

In execution of a transaction, if any P_i itself contains sub-transactions, it will apply above mechanism recursively.

4.2 Compensation Transaction Coordination

After receiving the Response, the TM coordinates transaction in following steps:

TM sends Enroll messages, which contain timestamp T, to all candidates. The candidates reserved and allocated resources successfully record operations in log, then directly commit the transaction and return Committed, which contain execution results, to the TM. The candidates failed to reserve resources return Aborted message.

According to returned results, the user may do the following by means of the TM:

- For candidates committed successfully, he selects some and cancels the others by sending Confirm and Cancel messages to them respectively within interval T.

- For candidates failed, he needn't reply them and may renew to send Enroll requests to locate new candidates until success or attempting N times.
- Within T, the candidate that has committed successfully performs either:
- In case of receiving Confirm message, it responds a Confirmed message.
- In the event of receiving Cancel message or nothing, it automatically rollbacks the taken operations according its log record, which may be implemented by performing a compensation transaction, and then returns a Cancelled message.

```
ActionOfSuperior
begin agent creates TM;
  while (transaction doesn't complete)
  begin agent sends CoordinationCont-
      ext to agent of candidate;
    wait for Response from JM;
    send Enroll to all candidates;
    while (t≤T) begin
      wait & record incoming messages;
      if (message is Committed)
        if (user selects some candidates)
        begin
          send Confirm messages to them;
          wait Confirmed messages; end
        else begin
          send Cancel messages to them;
          wait Cancelled messages; end
    end
  end
end
```

```
ActionOfInferior
begin agent creates JM;
  send Response to TM;
  wait for Enroll from TM;
  if timeout exit;
  reserve and allocate resources;
  record transaction in log;
  commit transaction;
  if (commit successfully) begin
    send Committed to TM;
    while (t≤T) begin
      wait for incoming message;
      if (message is Cancel) begin
        send Cancelled; rollback;
        release resources;
      end else
        if (message is Confirm)
        begin send Confirmed;
          release resources;  end
end end end
```

5 Security Solution

We use GSI [8] to address security issues. The security solution works like this:

- Authentication. It first creates a proxy credential signed by user's private key by using a user proxy and then mutually authenticates the identity.
- Authorization. The Gatekeeper maps the proxy credential into local user name by using text-based map file, then checks operation power of the user. If the user is authorized, Gatekeeper allocates a credential C_p used to create a process.
- Delegation. It promulgates C_p for the user to access other remote resources. By tracing back along the certificate chain to check the original user certificate, processes started on separate sites by the same user can authenticate one another, enabling the user to sign once, run anywhere.
- Encryption. Secure communication is implemented by SSL.

6 Implementation Discussions

The agent must be installed in every machine to join the transaction. Its behavior depends on the type of request. If an agent is invoked to initiate a transaction to run on remote sites, it sends CoordinationContext to the remote agents and locally creates the TM. If an agent receives a Coordination message, it creates the JM.

For participants/candidates to join a transaction, the agent sends CoordinationContext to them. The message contains the necessary information such as transaction type, transaction identifier, address of TM and timeout parameters. JM responds TM by the CoordinationContext, and passes Prepare message to the Co-reservation module. After querying the MDS, the Co-reservation gains the handles of reserved resources.

In addition, the model uses Globus Toolkit to complete low-level operations, such as discovery, reservation and allocation of resources, communication and security.

7 Conclusion and Future Work

We have described a secure transaction model for Grid computing. It can handle both short-lived operations and long-lived business activities, for which we propose coordination mechanisms and provide the algorithms respectively. In addition, we discuss some implementation strategies and propose a security solution.

The model is extensible because it may conveniently incorporate new protocols. In near future, we will develop it into a transaction middleware for Grid applications.

References

1. I.Foster, C.kesselman, and S.Tuecke. The anatomy of the grid: enabling scalable virtual organizations. Int'l J. High-Performance computing Applications. March 2001.
2. F. Cabrera, G. Copeland, T. Freund, J. Klein, D. Langworthy, D. Orchard, J. Shewchuk and T. Storey. Web Services Coordination(WS-Coordination). August, 2002.
3. F. Cabrera, G. Copeland, B. Cox, T. Freund, J. Klein, T. Storey and S. Thatte. Web Services Transaction (WS-Transaction). August, 2002.
4. A. Ceponkus, S. Dalal, T. Fletcher, P. Furniss, A. Green and B. Pope. Business Transaction Protocol V1.0. June, 2002.
5. I.Foster and C.kesselman**.** The Globus project: a status report. Heterogeneous Computing Workshop (HCW98) Proceedings. March, 1998.
6. I.Foster and C.kesselman. Globus: A Metacomputing Infrastructure Toolkit. The International Journal of Supercomputer Applications. 1997.
7. I.Foster, C.Kesselman, C.Lee, B.Lindell. K.Nahrstedt and A.Roy. A distributed resource management architecture that supports advance reservation and co-allocation. Intl Workshop on Quality of Service, 1999.
8. R. Butler, V.Welch, D. Engert, I. Foster, S.Tuecke, J. Volmer and C. Kesselman. A national-scale authentication infrastructure. Computer, December, 2000.