サービスベースシステムにおけるトランザクションの並行制御

7Q−7

何 千山　田中英彦
東京大学工学部

1 Introduction

We are implementing service base systems (SBS) based on an object-oriented approach [1]. SBS provides extensible as well as integrated environments for users and application programs to access and combine distributed resources conveniently. Current version of SBS is implemented by C++ and works serially. Reliability and concurrency control are two important issues which should be solved in SBS. This paper presents our design considerations to deal with these issues.

2 Motivation

SBS is a distributed system with the following features: a service base in every node is constructed and cooperates with each other by invoking service requests. Information about data or programs in computers are abstracted as objects and registered in SBS. Basing on these information, SBS combines distributed data and programs to provide services to users or application programs.

To implement SBS, two important problems should be solved: reliability and concurrency. A service may performs operations on data scattered across networks. There is the possibility that part of the system fails while the rest continues to operate. To execute a service, it is necessary to ensure that system failing will not cause unconsistency: either updates on data are done, or updates on data is not done when crashes occur. Concurrency makes SBS more efficient: operations in a service may be executed in several nodes concurrently; An users may invoke several service requests at the some time. Without concurrency control, e.g., SBS will make a service request to a remote node and block for results.

To solve the problems above, we are considering to introduce distributed atomic transactions to SBS. Atomic transactions have two important properties. First, each transaction is recoverable: either it runs to completion and commits, or a failure occurs and it aborts. In other words, either all or none of the effects of a transaction occur, partial executions are not possible. Second, transactions are serializable: transactions are allowed to execute concurrently, but the results will be the same as if the transactions executed serially.

To implement a transaction system, we have the flexibility in choosing the granularity of transactions. A finer granularity of transactions may lead to greater concurrency, by typically at the expense of a more complicated program.

3 Transactions and Concurrency Control

We divide transactions in SBS into three levels:

Level 1: Execution of One Service In SBS, a service can be described as: "An operation is imposed on data and certain results are produced":

$$method(results, input1, input2, ...)$$

The *inputs*, *results* are data which can be files in OS or data in a database etc. The *method* is the operation imposed on these data. A method can be a OS's command or a program etc. For each method or data, SBS contains its attribute information which are abstracted as an object and are stored in a database. These attribute information tell about where this data or program is located, what formate this data is, etc.

On accepting a service request from an user or application program, SBS works in the following steps:

1. From the database, get objects which contain attributes of the data or the method.
2. Check where each data or method is located.
3. If a data is not in the node where method is located, fetch the data to the node where method is located.
4. Semantic checking: to see whether the method can be imposed on these data. If not, transform the data to a suitable formate.
5. Execute the service in the node where the method is located.

Concurrency Control of Transactions in Service Base System

Qianshan He and Hidehiko Tanaka
University of Tokyo

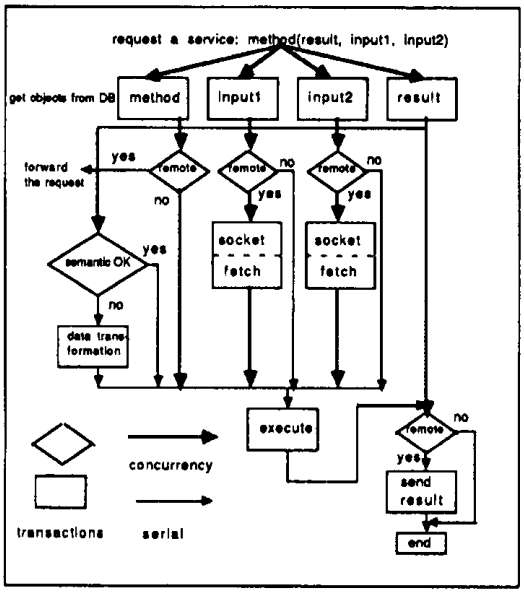

図 1: Transactions and Concurrency in Level 1

6. Send the results to a node where the user specified.

Each step can be seen as a transaction. For example, during step 6, this transaction should abort if there is a communication failure or node crash, and data in that remote node should not be changed partly. We can also find that there are many concurrency among these transactions. For example, in step 3, the transaction to fetch a remote data means to open a communication port (to connect a server socket in our implementation) and to get the data stream. When it is necessary to fetch several data which scatter in different nodes, step 3 can be divided into several transactions and these transactions can be executed concurrently. Figure 1 shows transactions and concurrency during the execution of one service. We are considering to use the two-phase commitment protocol in each transaction, because it is easy to be implemented. We are also considering to use two-phase locking to implement concurrency control among transactions.

Level 2: Combination of Services SBS provides facilities to combine services which are already registered in SBS. This is an important characteristic of SBS: providing a way to extend computer functions. Combination of services can be specified by an user or an application program, or done by SBS automaticly when it is necessary. Concurrent execution of services in a combination must consider synchronization between services: outputs of a service may become inputs of another service. Using parallel logic language FLENG++ [2] can solve with this problem easily. Suppose that an user specifies to combine three services: service1 is independent of service2 and service3, the output of service2 is the input of service3. With FLENG++, the service request can be expressed as the following three goals:

$$?-service1(in1), service2(in2, out2), service3(out2).$$

FLENG++ will first try to execute these three goals concurrently. In the execution of service3, because out2 is not available, service3 suspends until the execution of service2 completes and out2 is available.

In this level, we consider each service is a transaction. That is, either each service commits or aborts. Node crashing will not cause unconsistency during the execution of a service. The transaction commitment protocol and concurrency control in this level use the same one as in level 1 described above.

Level 3: Multi-users The process structure of SBS uses one SBS_backend_process per user or application program. There will be one SBS_demon process per machine, and it will be started at machine initiation time. An user runs a SBS interface (which runs on X window environment), and the interface is connected to SBS_demon through a stream socket port. After SBS_demon forks one SBS_backend_process for an user, the SBS_backend_proc is responsible for accepting service requests from this user and forwarding requests to remote nodes. Backend processes work concurrently.

4 Future Works

We have discussed how to divide transactions and how to manage concurrency control of transactions in SBS. We believe that it is easy to implement transactions and concurrency control in SBS by parallel language FLENG++. FLENG++ is also suitable to be a query language of SBS for users to make concurrent service requests. A future work is to use FLENG++ to implement the designs described in this paper.

参考文献

[1] Q. He and H. Tanaka, "An Object-Oriented Distributed System Integrating Multimedia Resources", Proc. of the First International Conference on Systems Integration, New Jersey, April 1990.

[2] H. Nakamura and H. Tanaka, "Object-Oriented Programming Language FLENG++ Based on Parallel Logic Language FLENG", WOOC'89, Tokyo, 1989.