Replication of Processes and Objects in Heterogeneous Groups

1G−02

Kenichi Hori and Makoto Takizawa
Tokyo Denki University
{hori, taki}@takilab.k.dendai.ac.jp

1 Introduction

In order to make a system fault-tolerant, a component of the system is replicated. A unit of system component is an *object*. A database server is a stereo type of object. A system is realized by the 3-tier architecture. A user in a client initiates a transaction in an application server. The transaction issues requests to object servers. On receipt of a request from a transaction, the request is performed on objects in an object server and then a response is sent back to the transaction. Only if the transaction successfully manipulates objects in object servers, the transaction commits. Objects are encapsulations of data and methods for manipulating the data. Object state may be changed by performing some methods like *write*. Object state may not be changed even if a method like *read* is performed on the object. There are many discussions on how to replicate database servers. In this paper, not only object servers but also application servers are replicated in order to realize fault-tolerant 3-tier client-server applications, and replicas of object servers and transactions in application servers are distributed in various types of computers interconnected with various types of networks like local area networks, the Internet, and mobile networks.

Replicas of objects like database systems are manipulated in the two-phase locking protocol. Here, one replica is locked for a read request and all the replicas are locked for a write request. In the quorum-based protocol [2], quorum numbers N_r and N_w of replicas are locked for read and write requests, respectability. Here, $N_r + N_w > m$ and $N_w + N_w > m$ for total number m of replicas. Objects support more abstract level of methods than *read* and *write*. A paper extends the quorum concept for *read* and *write* to abstract methods supported by objects. In order to maintain mutual consistency among replicas, it is critical to discuss in which order conflicting methods like *deposit* and *balance* on an *account* object to be performed on each replica, i.e. *serializability*. Objects are state-full while processes, i.e. transactions on application servers are state-less.

Object and application servers are distributed on different types of computers which are interconnected by various types of networks. A heterogeneous group is a collection of servers which are realized in different types of computers with different level of reliability. Computers are characterized by computation speed of each request and reliability level. More number of requests can be performed in faster servers than slower servers. The more number of requests are issued to slower servers, the longer the waiting quenes of the servers are getting. If requests waiting, not to be performed, are removed from the quenes, the slower replicas can follow the faster replicas. [5]

In section 2, we present a system model. In section 3, we discuss a protocol.

2 System Model

A system is realized in 3-tier client server architecture, which is composed of object servers, application servers, and clients. Transactions are initiated in an application server by a user in a client and issue request messages to object servers manipulating objects. Objects in object servers are manipulated and then responses are sent back to the transactions. In order to make the system fault-tolerant, object servers and application servers are replicated. A collection of replicas of a server is referred to as a *cluster* of the servers. Figure 1 shows a cluster which is composed of replicas $O_1 \ldots O_m$ $(m > 1)$ of an object server O and a cluster of application servers $A_1 \ldots A_l (l > 1)$ where a transaction T is performed. In addition, mobile clients like Oracle Lite can have a database where replicas of objects are stored.

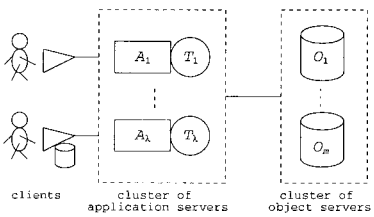

Figure 1: Replicas.

3 Replication

3.1 Replication of process

Transactions are realized in processes. Processes are state-less. There are following ways for replicating a process; *active* replication [3], *passive* replication [1] and *hybrid* replication [4]. Let $p_1, \ldots, p_n$ be replicas of a process p. In the active replication, every replica receives same messages in a same sequence, same computation is performed on every replica, and same sequence of output is sent back. Here, the process p is required to be deterministic. The process is operational as long as at least one replica is operational.

In the passive replication, there is one *primary* replica, say p_1, and the other replicas $p_2, \ldots, p_n$ are *secondary* ones. Messages are sent to only the primary replica p_1 and the computation is performed on only the primary replica p_1. No computation is performed on any secondary replica. A checkpoint is eventually taken at the primary replica p_1. A state of the primary replica p_1 taken at the checkpoint is sent to all the secondary replicas. Here, state of each secondary replica is changed with the state sent by the primary replica. The passive replication can be adopted for a non-deterministic process. If the primary replica is faulty, some secondary replica takes over the primary one. The secondary replica starts as a new primary

replica. That is, the process rolls back to the checkpoint and then is restarted. Since it takes a longer time to recover from the fault of the primary replica, the passive replication supports less availability than the active one.

The hybrid replication is same as the active one except that messages are sent to not only the primary replica but also the secondary replicas. The checkpoint is taken at the primary replica and is transmitted to all the secondary replicas. The secondary replicas can restart from the checkpoint and the state is restored to the current one by performing massages received.

3.2 Replication of object

Objects are state full differently from transactions. It is critical to keep state of every replica mutually consistent. In order to do so, conflicting methods like *read* and *write* are required to be performed so as to satisfy the *serialization* constraint.

Wiesamann *et al.* [5] classifies ways for replicating a database system. First, there are following types of replications with respect to when replicas are updated; eager replication and lazy replication. In the eager type, every replica is updated by methods as soon as the methods are delivered. In the lazy type, some replicas are delayed to be updated after the methods are delivered.

Next, replication ways are classified with respect to which object replicas are updated; primary, everywhere, and quorum-based.

In the primary way, there is one primary replica. Every method is performed on the primary replica. In the everywhere way, every method is performed on every replica. In the quorum-based way [2], each method is performed on some number of replicas. A subset of replicas where a method is performed *quorum* of the method. Let Q_t and Q_u be a pair of quorums of methods t and u, respectively. Let m be the total number of replicas. $|Q_t| + |Q_u| > m$ if one of methods t and u is an update type. For example, $Q_w + Q_r > m$ for *read* and *write* quorums.

There are following combinations of replications; eager primary, eager update everywhere, lazy primary, lazy update everywhere, eager quorum-based, and lazy quorum-based ways.

The famous two-phase locking protocol is a type of eager update everywhere since *write* is performed on every replica. In the primary replication, every request is issued to a primary replica. In the eager type, every other replica is updated before a transaction commits. In the lazy type, other replicas are updated after a transaction commits.

3.3 Replication of transactions and object servers

Let $T_1, \ldots, T_m$ be replicas of a transaction $T(m \geq 1)$. Let $O_1, \ldots, O_n$ be replicas of an object server $O(n \geq 1)$. As discussed here, there are three ways for replicating a transaction and six ways for replicating an object server. Totally eighteen combinations of transaction and object server replications.

First, let us consider a passive replication of a transaction. Let T_1 be a primary transaction replica and the others $T_2, \ldots, T_m$ are secondary. Only the primary replica T_1 issues requests to replicas of the object server O. The response of the request is sent back to the primary transaction replica T_1. A save point is taken during the primary transaction replica T_1 is being performed. A state of T_1 taken at a save point is sent to all the replicas. On receipt of the save point, the state of each transaction replica is changed at the state sent from T_1. One-to-one and one-to-many communications from transactions replicas to replicas object servers are done.

Next, let us consider a hybrid replication of a transaction. Let T_1 be a primary replica and the others be secondary. The hybrid replication is same as the passive replication except that replicas on which requests are performed send responses to all the secondary transaction replicas. Thus, every transaction replica receives responses from object replicas while only the primary transaction replica issues requests to the object replicas. One-to-one and one-to-many communications from transactions replicas to replicas object servers are done.

Lastly, let us consider a case a transaction is actively replicated. Every transaction replica T_i issues requests to replicas of the object server. Here, many-to-many communication between transaction replicas and object replicas is done.

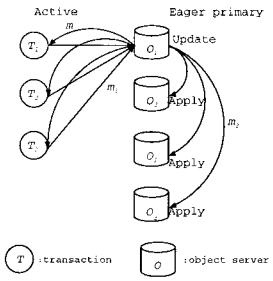

Figure 2: Active eager primary replication.

4 Concluding Remarks

We discussed how to replicate transactions and objects in client-server systems. Transactions are state-less while objects are state-full. Transactions and objects are replicated in different ways. We are now evaluating architectures for replicating transactions and object servers.

References

[1] Budhiraja, N., Marzullo, K., Schneider, B. F., and Toueg, S., "The Primary-Backup Approach," *ACM Press*, 1994, pp.199–221.

[2] Garcia-Molina, H. and Barbara, D., "How to Assign Votes in a Distributed System," *JACM*, Vol 32, No.4, 1985, pp.841-860.

[3] Schneider, B. F., "Replication Management using the State-Machine Approach," *Distributed Computing Systems, ACM Press*, 1993, pp.169–197.

[4] Thomas, L. C. and Mukesh, S., "The Delta4 Distributed Fault-Tolerant Architecture," *Distributed Computing Systems, IEEE CS Press*, 1990, pp.223–247

[5] Wiesmann, M., et al., "Understanding Replication in Databases and Distributed Systems," *Proc. of IEEE ICDCS-2000*, 2000, pp.264–274.