Client-Server Application System Using RMI

Phyu Pyar Khin Lay
University of Computer Studies, Mandalay
phyulay4568@gmail.com

Abstract

Most modern network programming is based on a client/server model. The Remote method Invocation API lets java objects on different hosts communicate with each other. A remote object lives on a server. In RMI application, Resources may be accessed by many clients concurrently and managed by server. Concurrency control is atomic execution of transaction on shared resources by controlling the interleaving of concurrent accesses. This paper implements a light weight Client/Server application for multiple clients by using java RMI mechanism. The system controls the concurrency based on 2-Phase Locking Algorithm and multi-threading technology. To show the experimental results, it uses the fashion shop as an application which has TRI, Pim, STEP, Giorano and Adidas transactions. Customers can buy the same type (such as TRI) simultaneously from different sites.

1. Introduction

Rapidly evolving network and computer technology is coupled with network programming which is based on a client server model. Basic network programming requires all the involved parties to communicate via application-level protocol. With no single application model, it can be difficult for teams to communicate application requirements effectively and productively. Another complicating factor in application development is that the design of application-level protocol is complex and can be error-prone. RMI represents a higher-level view of networking; a programmer might only use the RMI protocol. So, a difficult and error prone programming situation is avoided.

Concurrency control mechanism plays an important role in multiple client applications; its main motivation is sharing the resources. This paper uses 2-Phase Locking algorithm to control the interleaving of concurrent transactions. This algorithm is implemented by using the multi-threading technology.

In this paper, the main database is stored on server machine and it accessed from multiple clients via the communication network. The sever side JVM involves the main database, server code and skeleton. The client involves stub and the client code. The server code uses 2-phase locking algorithm in java thread to control the concurrent access of transaction from client JVM.

2. Client-server model

In Client/Server model, the client initiates a conversation, while a server waits for clients to start conversation with it. Some servers process and analyze the data before sending the result to the clients. Such servers are often referred to as the application server to distinguish them from the more common file servers and database servers. A file or database server will retrieve information and send it to a client, but it won't process that information. An application server is not a server that serves files that happen to be applications. In this system the server program is an application server.

Associated with every transaction is a coordinator, which is responsible for governing the outcome to the transaction. The coordinator may be implemented as a separate service or may be co-located with the user for improved performance-that is, within the same Java Virtual machine (JVM). It communicates with participants enlisted with its transaction to inform them of the desired termination requirements within the scope of the transaction.

3. Concurrency control

The concurrency control is to improve the performance of the transactions. The goal concurrency control must accomplish, is to produce serializable schedules for multiple transactions [2]. It allows several transactions to be executing simultaneously such that collection of manipulated data items is left in a consistent state. Concurrency control achieves consistency by ensuring data items

are accessed in a specific order. The sharing of resources is a main motivation for constructing the systems. Resources may be managed by server and accessed by clients. Concurrency control is atomic execution of transactions on shared data by controlling the interleaving of concurrent accesses [5]. To satisfy concurrency control, methods are:

- Locking
- Timestamp Ordering
- Optimistic Concurrency Control

3.1. Two-phase locking

In order to ensure serializability of parallel executed transactions elaborated different methods of concurrency control. One of these methods is locking method. There are different forms of locking method. In this paper the system uses one of forms of locking method, namely two-phase locking method. Structure of transaction and method of managing, which satisfy principle of two phase locking, is significant that process of transaction execution can be divided into two periods: period of gradation and degradation. In first period, transactions lock all necessary resources and in this period number of resources seized by transaction monotonically increase. In second period all locked resources simultaneously unlocked [1]. The 2-phase locking class of schedules is the intersection of the 2PL class. To comply with the 2-phase locking protocol, a transaction needs to comply with 2-phase locking protocol and release its write locks only after it has ended. On the other hand, read locks are released regularly during phase 2. Typically end of phase 1 is safely determined when a transaction has entered its ready state in all its process. If several processes are involved, then a synchronization point among them is needed to determined end of phase-1 for all of them. This is usually too costly and end of phase-1 is usually postponed to be merged with transaction end. This system imply the 2-phase locking protocol with the ExeCoordinator class (a thread).

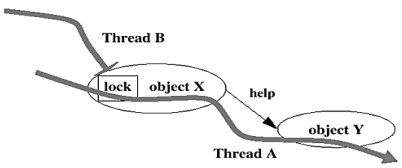

Figure 1. Locking between threads

In Figure 1, before each read operation, the service places a read lock on an object, and before each write operation, it places a write lock on an object. This way, the read and write operations are covered by locks. There are a few simple rules for managing locks on objects. First, locks must be held until the transaction commits or aborts. Recall that an abort corresponds to a write operation, replacing the new value with the old one. In case of an abort, therefore, after that final write restores the old value, the write lock releases on the modified object. A transaction is well formed if each of its operations is covered by locks, and if all locks release at the transaction's completion. Second, since an object with a read lock is not modified by a transaction, other transactions can read that object, even if the first transaction is still in progress. For this reason, read locks on an object can be shared between transactions, they are shared locks. On the other hand, if an object has a write lock, that implies that the lock-owning transaction modified the object. A write lock is exclusive to a transaction that owns it , other transactions cannot share it. If an object has an exclusive lock on an object, other transactions must wait before they can either read or write that object. In addition, a transaction cannot lock an object if it does not intend to perform a subsequent read or write on that object. And finally, a transaction can upgrade a shared lock to an exclusive lock, but cannot downgrade an exclusive lock to a shared lock [5].

4. Multithreaded program

Multithreaded programs extend the idea of multitasking by taking it one level lower: individual programs will appear to do multiple tasks at the same time. Each task is usually called a thread which is short for thread of control. Programs that can run more than one thread at once are said to be multithreaded. The essential difference between multiple processes and multiple threads is that while each process has a complete set of its own variables, threads share the same data. However, shared variables make communication between threads more efficient and easier to program than inter-process communication. Moreover, on some operating systems, threads are more “lightweight” than processes it takes less overhead to create and destroy individual threads than it does to launch new processes [4].

4.1. Communications in Java RMI

The Java Remote Method Invocation (RMI) is a three-tier communications protocol. Remote objects are defined through a remote interface, which can be exported to remote clients. Objects are passed between the client and server by marshaling the object into a serialized byte stream. This serialization process can be performed dynamically

using Java's reflective capabilities, or it can be tuned by the implementer for optimal results [3]. RMI (Remote Method Invocation) is a java-based high-level communication mechanism. RMI enables us to invoke methods on remote machines (or processes) and get the invocation results. In RMI, a Remote interface is defined, and a Remote object (sometimes referred as the server) implements it. Since it is defined as a 'remote interface', any Java process (sometimes referred as the client) may invoke methods on the Remote object. After the remote object is created, it should bind itself to a Name Server (rmiregistry), so the clients can look it up. The name server acts as a phone guide: each remote object binds itself with a unique name, and the clients can connect to the name server and ask for that remote object. The name server returns a stub for the remote object; the stub is an implementation of the remote interface which is automatically created and takes care of the communication between the client and the remote object.

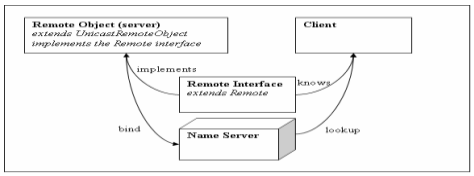

Figure 2. A typical life cycle of a RMI system

In Figure 2, the name server (rmiregistry) is started on the server's machine. The server starts, creates a remote object and binds it to the name server. The client starts, connects to the name server and asks for the server's remote object. In order to do so, the client must know the address of the name server (host and port), and the name of the remote object.

5. System overview

This paper follows the popular three layers application architecture used for fashion shop sales center. It consists of three independent workstations and controls concurrency to satisfy customers. It will be used in fashion shop center. In this center, TRI, Pim, Giorano, Adidas, STEP exist. If customers buy many items simultaneously from the sales center at only one server, concurrency occurs.

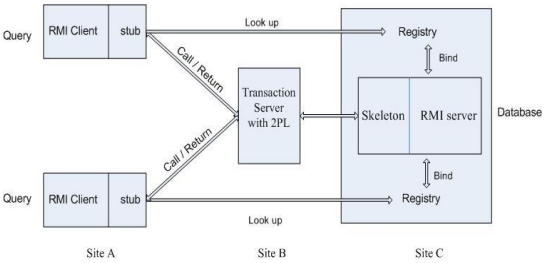

Figure 3. Structure of the system

In Figuer 3, several RMI Clients are in charge of interaction with users at site A. If the first customer (client) buys Pim shirt and the second customer (client) buys Giorano shirt from the database at the single server, the service requests are transported to the site B, where multiple transaction requests are granted locks to resources according to 2-phase locking algorithm. The serialized requests are then send to site C and the RMI server at site C will fetch via site A and execute it remotely at site C. After the transactions finish, site C's RMI server will return the result can in turn feedback to the user at site A.

As multiple clients can submit their transactions simultaneously from physically different workstations, the transaction coordinator must grantee the consistency of the database behind it. Two-phase locking a[illegible]s used here for this [illegible]. This class parses [illegible]e format of the data coming fr[illegible] Time [illegible]en the transaction is decomposed i[illegible] RMI request/reply [illegible]st. Then multiple requests a[illegible] RMI reuest/reply serialized and sent to s[illegible]

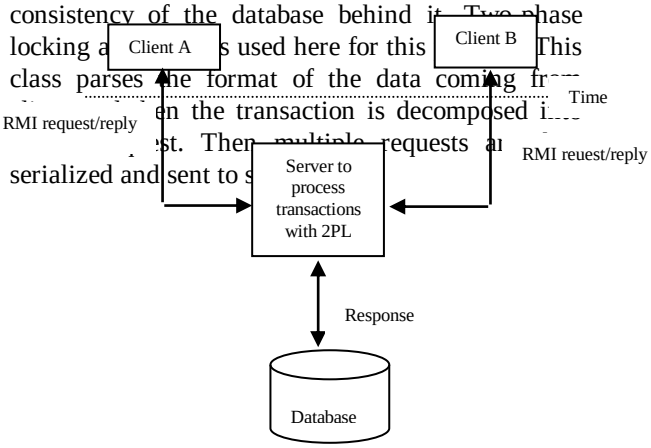

Figure 4. Flow of the system

The implementation of this system includes two sites (Client and Server). In this paper, many clients can request to a single server. The Client requests the remote procedure it needs to access (read

from/write to database) to the server and the server performs the requested procedures for the client.

Two clients concurrently connected to a single server to read/write the data (such as Pim, Giorano) from the database. The clients request the remote procedure and it needs to access data (read from database). As a demo, the first client buys the 'PIM' shirt and the second client buys the 'Giorano' shirt respectively. The first client will send the request that read the price, balance of the 'PIM' shirt and the second client will send the request that read the price and balance of the 'Giorano' shirt respectively by calling the read procedure on the server. After gathering the require information for each item, the client will perform the sale process and it also need to write back the balance to the server again. Each client will request the procedure that writes back the data to database as similar as the read operation as shown in the figure. In this case, if the two clients send the request at the same time, the server needs to control the concurrency and performs the procedure in serial order. To control concurrency the system will use 2-phase locking at server. According to 2-phase locking, transactions are serialized to send at database. The database replies transactions to the corresponding users.

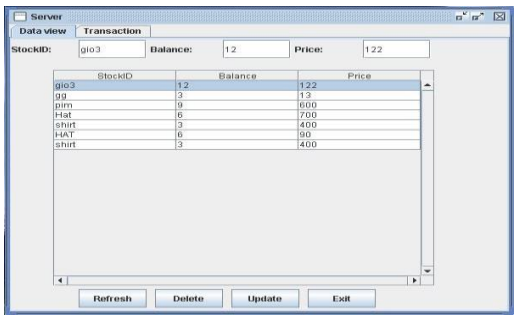

Figure 5. Server site data access form

The detail processing of each transaction is displayed on the server site as shown in Figure 5. The server can also watch the information of the items. In the server site forms, the data (such as Pim, Giorano) written by different client site forms are serialized according to two phase locking to control concurrency. Data view in the server site form displays the read or write transactions by the different client sites. If the data is updated, select the data record and enter the data which is update in StockID, Balance and Price.

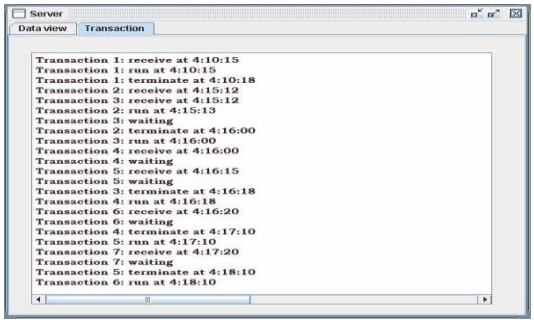

Figure 6. Server site transaction access form

Figure 6 is the server site and it provides the administrator to access the server's transaction. If many clients read many data existing in the database, read transaction is showed in transaction at the server site. If many clients write many data into the database at the single server, transactions are serialized according to two phase locking. Transaction 1 is defined on it, earliest received time and the later received transactions are transaction 2, transaction 3 and so on. When transaction 1 writes (Transaction 1, 4:10:15, Client 1), transaction 3 writes (Transaction 2, 4:10:15, Client 2), transaction 3 writes (Transaction 3, 4:10:15, Client 3), the concurrency of three transactions is occurred. The system uses two phase locking to control concurrency. While transaction 1 is running (Transaction 1, 4:10:15, client 1), transaction 2 and transaction 3 must be waiting till transaction 1 terminates. After transaction 1 terminates (Transaction 1, 4:10:18, client 1), transaction 2 is running (Transaction 2, 4:10:18, client 2). After transaction 2 terminates (Transaction 2, 4:16:00, client 2), transaction 3 is running (Transaction 3, 4:16:00, client 3). In this way, Many transactions are read/written at the server site by many clients according to two phase locking.

6. Conclusion

The primary goal of the system is to implement two Phase Locking Algorithm and Java RMI to construct a light weighted client/server application and to connect the analysis of applicability of architectural decisions. All work was carried out as the basis creation of prototype for the Internet Shopping Application. This system can also be used as a frame work for some developer who relies on only programming language (java) to develop client/ server application without having to worry about the underlined operating system and hardware because of the platform independent property of the java RMI

mechanism. Therefore, the system is used for the development and debugging of code Java RMI and all Internet Shopping Applications. Although the client/server system can use many clients, the paper uses many clients to perform the application. It can be used in the light weighted client/server environment, but it cannot give grantee for the heavy weighted application. Since it used the two phase locking protocol for concurrency control, it cannot give grantee for deadlock condition.

7. References

[1] Agil A. ALIYEV, "Two phase locking protocol in distributed database".

[2] Gjermund Hanssen, "Concurrency Control in Distributed Geographical Database Systems".

[3] Katrina E. Kerry Falkner, Paul D. Coddington and Michael J, "Implementing Asynchronous Remote Method Invocation in Java".

[4] "Multithreading in Java"

[5] "Operating System Concepts," Sixth Edition.

[6] Wei Chen, "Prototype of Distributed Database System With 2-phase Locking Algorithm," April 29, 1999.