Design Report

Distributed Staff Management System

Design Report for COMP 6231 BB

Assignment 1

Concordia University

Second Edition

Huaqiang Kang

STU ID 40018914

2016 06 08

Table of Content

1. Introduction

In this system, I have designed and implemented a simple distributed staff management system. It can store staff information distributed. The whole system consists of three parts. The bottom layer is the distributed database, which is simulated by HashMap and Linkedist. The middle layer is the server interface. Through Java RMI, it is possible to access remote resources just like local resources. The server exposes interfaces to other servers for synchronization and provides functions to clients to store and edit employee information. The high layer is the client application, which receives the input data from manager, transforms them to proper format and transfers to server side.
In the next section, I will introduce the architecture of this system, how Java RMI is implemented to assemble each part together. And the benefit of this kind of architecture. Section 3 will include the concurrency topic, how the fine-grained locks are implemented to maximize concurrency. Section 4 talks about the data structure selection in the server side, especially the database model. Sections 5 contains the test case explanation. There will be case designs to cover the functionality and concurrency.

2.1 Architecture

This system is a standard Client-Server architecture system in request reply pattern. There three layers in this system. The manager client is the user interface, which interacts with user. User need to input its administrative ID as an identifier. A log file records all the operation this user does, like logging in status, insertion and modification content and results for audit. The clinic server is responsible the concurrency transparency. Since the sequence in of each record is universal. The server deal with the synchronization of the sequence resource. It assigns a sequence number to each newly insert record. It is also listens to the port to receive UDP message. Thus the clinic server is a multi-thread system. At least 2 threads run at the same time, one is the listener to UDP message, another is the RMI server dealing with the invocation of remote resource methods. The database, simulated by HashMap, stores the records in an efficient way. Although it does not support the full ACID, a fine grade locking is implemented.

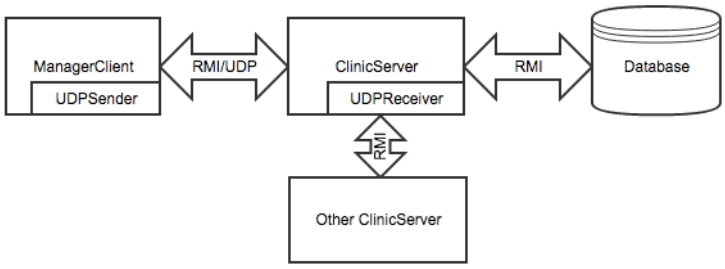


Figure 1 Architecture chart

When client starts, it retrieves the distributed server list from a default server. to create a staff record, user need to login the client software with his/her unique administration ID. After logged in, user can perform the simple management action like creating Doctor Record, creating Nurse Record, modifying a record or even get the records number of the whole distributed system.
To perform a doctor record creation task, user needs to input the information in each field. These are including first name, last name, address, phone number, specialization and location. These are all in string format. After receiving these information, the client first checks the correctness of each field. One of these constrains is that the location information must be one of these three choices. They are "mtl", "lvl" and "ddo". If not, an error will be returned. As a distributed database system, the database stores doctor records of their own clinic only. The client as a router, checks the location of the doctor record to be sent and send the record to the designated server.
When the server receives the data from the client, it assigns a unique ID to the record to be inserted, and then insert the record to the database. The server applications also communicate each other to synchronize the sequence number, so that the ID will not be duplicated.
The database, provides a container to store the staff records. It provides functions for

programmer to insert, delete and modify records. Likes traditional DBMS, it provides index for fast retrieval, record level locks and serializable isolation.
For the modification case, client does not examine the Legitimacy of input value. It only sends the record ID to be modified to each server to find out which server stores that record. And then send the field name and new value to that server for an update.
There is an UDP server running background to receive the message from client who requests for the count number in this server. To make the UDP server work properly, it is necessary to bind the Clinic Server with the UDP server for data sharing, so that the UDP server can invoke the getCount function from ClinicServer.
When a client wants to know the record count of each server. The Client first initiates a socket for sending message. And then sends a request to each server waiting for the replies. When the server receives the request from client, it invokes the getCount function to check the records number in the database and assembles the information to a message and sends back.

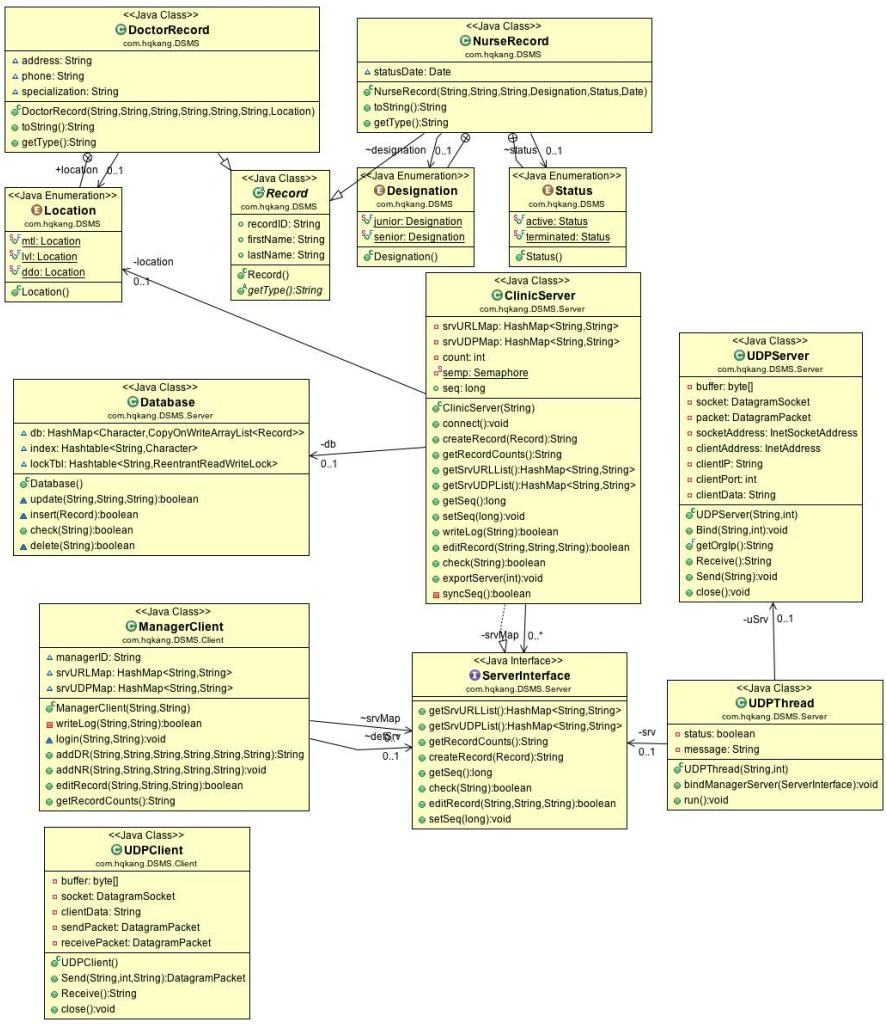
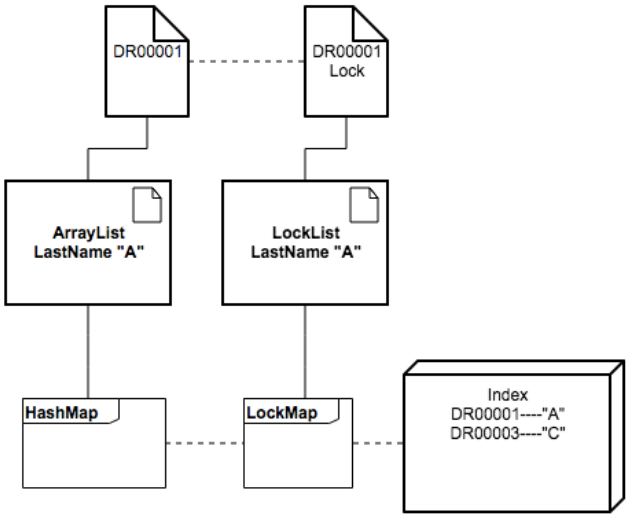


Figure 2 UML chart

2.2 Concurrency Design

In order to maximize concurrency, a fine grain level lock must be implemented when manipulate data in the database layer or server layer. For the server application, it provides a mechanism for synchronization the sequence number among each server. For the sequence resource, I set a semaphore valued 1 for this variable. Before one of the online server can read its value and increase it, the server must acquire the semaphore. If one server obtained the semaphore successfully, firstly, it communicates with other online server to get others sequence number. Secondly, after got the list of each server's sequence, it finds out the largest number and set the number back to all servers. It also assigns the largest number as the record ID to the record to be inserted. Finally, the sequence will be increased by 1 and the server can release the semaphore. At this time, the sequence may be not synchronized until an insertion of any server occurs next time.
This mechanism can make the sequence continuous but needs more than 3 times to communicate. Another proposal is that each server assigns the sequence individually. Suppose that there are 3 servers in total. Server A's sequence starts form 1 and increases by 3, while sever B's sequence starts from 2 and increases by 3. Since the sequence assignment is individually, there is no need to communicate with each other. But a configuration change will be made when adding servers.
Another place where the resource lock is used is the database layer. There is no "synchronized" token for the update function. "Synchronized" function will block the



whole system since there will be only one thread can access the object when a

synchronized method is executed. I created one reentrant ReadWrite lock for each record. In the database layer, only store operation is implemented, no static or global variable is used, so reentrant is acceptable. When a user wants to modify a record. The client first sends an existence check request to each server. Each server

Figure 3 Record Level Database Lock

examines the index to give a feedback of the record existence. During the check, the server tries to obtain a read lock of the record to be checked. If the obtaining failed, it means that another server thread is trying to modify that record. This server thread will wait until the write lock is released.

Similar write lock also applies to delete operation and insert operation, in which a write lock is used during deleting the HashMap element and index element.
To avoid deadlock, I distribute different resources in different layer. Like the sequence resource, it only exists in server layer. The ReadWrite lock only exists in database layer. This allows a safe sequence of process and allocation of resources.

2.3 Data structure

In this system, there are two kinds of record, one is Doctor record, another is Nurse record. in the server side, the functions do not distinguish the different types of record because of polymorphism. The doctor record and the nurse record all extends the class record.

In database class when user wants modify a record, he should point out the field name of the class and the new value of this field. Using reflection, a new modified object can be created more easily. But there is a mirror problem. Since the first name and last name fields are extended from father class record, these two fields cannot be found in the original class, whereas exist in the super class. A looping up through the super classes works fine without casting.

In the structure of doctor record and nurse record, enumeration type is applied to make sure there will not be any typo when user inputs information to clients.

In the database layer, a CopyOnWriteArrayList stores the records whose last names begin with the same letter. The reason for using COW is that reads does not blocked and write does not block read. Unfortunately, only one write can occur at once. And all the 26 linked list store in one HashMap. The reason for the choice of HashMap, not the HashTable is for its performance. Although HashMap is not synchronized and not thread safe, I use the explicit write lock to solve the concurrency problem.

In this case, I am mainly focusing on the concurrency capability. When an array list is full, a resize operation is needed. The cost of resizing is O(n). For the large quantity insertion scenario, it is better to choose a CopyOnWriteArrayList. When modifying an ArrayList concurrently, for example one thread is modifying, another thread is loop reading, it's not thread safe. A java ConcurrentModificationException will be thrown. Copy on Write mechanism can solve this problem.

When a modification operation invokes, user only need to specify the record id, the field and the new value. An index mapping from record id to its last name is implemented to reduce the search range, although this is not the most optimized solution.

In the database, there is a lock table stores ReadWrite locks mentioned in the last section for each record. The record ID is the key and the lock object is the value to be stored.

2.4 Test scenarios

unit tests are applied in the system.
Scenario1: When input field is illegal, an error will be returned. In the ManageClient Class, system parse the user input string to enumerate in the field of “location”, “status” and “designation”. When type casting failed, a parse exception will be thrown. I handled this exception and returns a warning to user.
Scenario2: When user logging in, the log will record this action. A “WriteLog” module is wrapped, this module can create new log for each administrator, append operation log to existed log file and record the operation time, operator and the operation returned status. Similar to client side, each server also has the log writer to record each operation received from net.
Scenario3: Create record operation. If all fields are correct, a new record can be formed in the client side and be passed to the server for storing. If successful, the record ID will be returned to client. The client can judge whether the operation is successful by detecting the returned sting.
Scenario4: Record modification operation. When a user wants to modify a record, the client passes the “record ID”, “field to be modified” and the “new value”. The server returns the Boolean value to client to give a feedback of this operation.
Scenario5: Get the record count of each server. When a user wants to know the record count of each server, the client sends a UDP message to each active server. when the server receives the message, it returns the internal value of count number in the instance by UDP package as well. If the client receives the reply, it print the result on the console.
Scenario6: Concurrency.
Since the RIM interface is already a multi-thread interface. The client side will be multi-threaded.
I created 2 clients for operation. Each client has 3 thread for concurrently inserting, editing. And finally, get the record counts of each server.
For each client, it will create 500 Doctor records and 500 Nurse Records in three threads.
After that, I created two thread “sa” and “sb” to check the availability of shared access; The edit result can be found in the administrator’s log.
Jtest is not implemented because of limited time.

3 References

[1] http://stackoverflow.com/questions/23237595/how-to-add-multi-threading-to-rmi-project
[2] http://winterbe.com/posts/2015/04/30/java8-concurrency-tutorial-synchronized-locks-examples/
[3] https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/locks/ReentrantReadWriteLock.html
[4] http://www.oracle.com/technetwork/issue-archive/2010/10-jan/o65asktom-082389.html
[5] http://stackoverflow.com/questions/16295949/get-all-fields-even-private-and-inherited-from-class
[6] http://javahungry.blogspot.com/2015/04/difference-between-arraylist-and-linkedlist-in-java-example.html
[7] http://stackoverflow.com/questions/17853112/in-what-situations-is-the-copyonwritearraylist-suitable
[8] https://en.wikipedia.org/wiki/Reentrancy_(computing)