Project Report

Chenran Li chl278

Kaiyuan Tang kat137

1. Reader/writer solution

Every server will make a copy of the original database as its local file when it is started. In this case, original file is initially empty. When a reader comes in, it will read the server's own copy, hence server can respond to reader very quickly without accessing the remote database. When a writer wants to make update to its connected server, all of the servers will know the update information and apply it to its own copy. There is a primary server who is responsible for writing to the remote database after each writer's request no matter to which server the writer is writing.

The server broadcasts each request it receives from the client to every other server, and add this operation to the proper position in the operation queue according to the timestamp.

(1). If it is a writing operation, it will only be executed when that operation is at the head of the operation queue and server has received an allow message for that operation from each other server. After processing the write, server removes it from the queue together with its corresponding allow messages and send release message to other servers.

(2). If it is a read operation, it will be executed after two requirements are matched: **[1]** server has received an corresponding allow message from each other server, **[2]** there is no write operation with smaller timestamp than this read operation in the operation queue. In other words, the read operation can be executed even when it is not at the head of the queue. We thereby achieve concurrent multi-reading.

(3). Server will also execute write operation sent by other server. When there is a write operation at the head of the queue and it is sent by other server, server will run it anyway to its own copy file and wait for sender's release message for that write operation.

The reason why we choose to implement this protocol is that it keeps data held by every server consistent while gives reader a very fast respond. By only letting one server updating remote database saves the updating time compared with all the server trying to write to the original file and is less error-prone.

2. Performance Analysis

(1) Advantages

a. This protocol is very efficient when there are a lot of reads and few writes. Each read will have a short response time by avoiding server accessing remote database to fetch data and allowing multi-read happening concurrently.

b. Only letting one server have the right to write to shared file avoids database dealing with concurrently writing, hence simplifying implementation of database.

c. Every server having a copy of data makes the system more robust and fault tolerant. Even though only the primary server can write update to shared file, this is not an one point of failure. Since once primary server is down, all the other servers are still connected with each other and have consistent data. We can delete all the information of down server and randomly choose one of working servers to be the new primary server and the whole system will still work.

(2) Disadvantages

a. Having a copy at each server will cost a lot of memory space if the data size is very huge.

b. Writing to remote database is time-consuming, when primary server is updating the shared file, all the other server have to wait to guarantee consistency. This reduces the whole system performance when there are a lot of writes.

c. Since a server has to communicate with all other servers in order to do write or read, once some server is down, then the whole system blocks until either the broken server resume working or the requests of down server are remove from each server's operation queue and the down server itself is removed from each server's server list which stores servers' information. Therefore we need some other mechanism to suspecting each server to see if they are working correctly.

(3)Scalability

Depending on what application this protocol is applied, it may or may not be efficient.

(a) read/write ratio

When most operations are read, then the system has a good performance similar to caching. When write/read ratio is high, compared with normal distributed mutual exclusion protocol, the difference is that only primary server communicate with shared data. If primary is located nearer than average case to the database, then the performance is better. If primary server is more powerful than ordinary server, which means working faster, then the performance is also better. Therefore we can deliberately choose some specific server as primary server to achieve better performance.

(b) Shared data size

If the shared data size is very small, then this protocol works well. But it may cost too much memory space when we have to store a large amount of data.

(c) Adding/Removing server

Since every server holds information about other server, it is complicated when we want to remove or add some server. We have to block every already existing server to update their server list and operation queue in order to get the whole system go ahead.

(d) Server crash

When some server break down, we need some special mechanism to realize that and remove that server or fix it. We can solve this by set a timeout on each request, when there is no allow message or release message from some server, we assume it is dead, then remove all its information and broadcast that information to all other server. If after a while we find this was a mistake, simply add that server again as a new server.

3. Code

(NOTICE: All java files are in the java package mutualExclusion.)

(1) start servers

```
C:\eclipse\eclipse-workspace\OS 2510 project1\src>java mutualExclusion/Server

Usage: enter integer port number:
```

[1]start three terminals

[2]each run Server java file,

[3]enter port number: 9999, 10000, 10001,

[4]enter corresponding server ID: 0, 1, 2

```
C:\eclipse\eclipse-workspace\OS 2510 project1\src>java mutualExclusion/Server

Usage: enter integer port number:
9999

Please enter server ID:
0
server 0 is waiting for server 2 to connet:
```

```
C:\eclipse\eclipse-workspace\OS 2510 project1\src>java mutualExclusion/Server

Usage: enter integer port number:
1000

Please enter server ID:
1
server 1 is waiting for server 2 to connet:
```

```
C:\eclipse\eclipse-workspace\OS 2510 project1\src>java mutualExclusion/Server

Usage: enter integer port number:
10001

Please enter server ID:
2
Waiting to send connection to server 1 .
Please enter IP address and port number of server 1 :
```

[5] Start with server 2, first enter IP address and port number of server 1(0.0.0.0 10000), click ENTER, then enter IP address and port number of server 0(0.0.0.0 9999), click Enter

```
C:\eclipse\eclipse-workspace\OS 2510 project1\src>java mutualExclusion/Server

Usage: enter integer port number:
10001

Please enter server ID:
2
Waiting to send connection to server 1 .
Please enter IP address and port number of server 1 :
0.0.0.0 10000
true
have sent connection to server 1

Waiting to send connection to server 0 .
Please enter IP address and port number of server 0 :
0.0.0.0 9999
true
have sent connection to server 0


We have bind server 2 with all other server!
```

[6] For server 1, enter server 0's IP address and port number(0.0.0.0 9999)

```
Please enter server ID:
1
server 1 is waiting for server 2 to connet:
server 2 binded


Waiting to send connection to server 0 .
Please enter IP address and port number of server 0 :
0.0.0.0 9999
true
have sent connection to server 0


We have bind server 1 with all other server!
```

```
server 0 is waiting for server 2 to connet:
server 2 binded


server 0 is waiting for server 1 to connet:
server 1 binded



We have bind server 0 with all other server!
```

[7] Server 0 automatically receives connections from server 1 and server 2. Now we have bound each pair of servers.

(2) start clients

There are two ways to start clients.

(a). Each client is a different thread.

Just for proving the correctness of code and saving time, you can run Client1.java file where each client is a thread not a process. In this case, we only have 3 writer and 3 reader since it causes extremely high CPU utilization.

```
C:\eclipse\eclipse-workspace\OS 2510 project1\src>java mutualExclusion/Client1 "filepath"
```

(b). Each client is a difference process.

According to requirement.

For convenience, only start 8 clients, it works well with 10 or more clients.

[1] Starts 8 terminals, type the command line (a script file name and a file path) for one server

```
C:\eclipse\eclipse-workspace\OS 2510 project1\src>client0 distributed_me_requests.txt
```

```
C:\eclipse\eclipse-workspace\OS 2510 project1\src>client1 distributed_me_requests.txt
C:\eclipse\eclipse-workspace\OS 2510 project1\src>client2 distributed_me_requests.txt
C:\eclipse\eclipse-workspace\OS 2510 project1\src>client3 distributed_me_requests.txt
C:\eclipse\eclipse-workspace\OS 2510 project1\src>client4 distributed_me_requests.txt
C:\eclipse\eclipse-workspace\OS 2510 project1\src>client5 distributed_me_requests.txt
C:\eclipse\eclipse-workspace\OS 2510 project1\src>client6 distributed_me_requests.txt
C:\eclipse\eclipse-workspace\OS 2510 project1\src>client7 distributed_me_requests.txt
```

[2] Create 6 writers and 2 readers

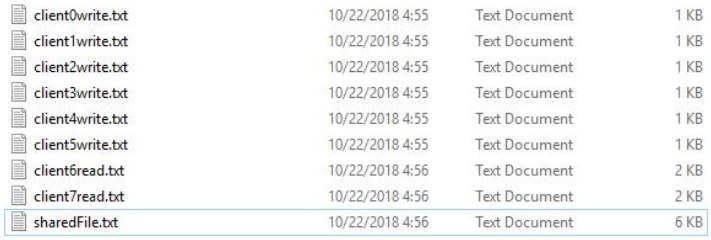
(**Notice**: you should change "distributed_me_requests.txt" to the path of input file.)

(3) Run

Run each clients as soon as possible to insure concurrency.

(4) Output

Each clients will write a file and server 0 which is primary server will write the sharedfile.txt.



client0write.txt	10/22/2018 4:55	Text Document	1 KB
client1write.txt	10/22/2018 4:55	Text Document	1 KB
client2write.txt	10/22/2018 4:55	Text Document	1 KB
client3write.txt	10/22/2018 4:55	Text Document	1 KB
client4write.txt	10/22/2018 4:55	Text Document	1 KB
client5write.txt	10/22/2018 4:55	Text Document	1 KB
client6read.txt	10/22/2018 4:56	Text Document	2 KB
client7read.txt	10/22/2018 4:56	Text Document	2 KB
sharedFile.txt	10/22/2018 4:56	Text Document	6 KB

For writer, it is their input. For reader, it contains what they read from server.