# EZshare system

## Group: As We Can

### Group members:

Shaohong TIAN shaohont@student.unimelb.edu.au shaohongt

Fangfang Huang fangfangh@student.unimelb.edu.au fangfangh

Ziqian Qiao zqiao1@student.unimelb.edu.au zqiao1

Zhe xu zxu5@student.unimelb.edu.au zxu5

# 1. introduction

In this project, we build a distributed file sharing system, which is composed of servers and clients called 'EZShare'. Multiple functions are implemented on the client side:

QUERY: get resources across servers

FETCH: download any type of files from severs along with URL

PUBLISH: create a public resource

SHARE: give authority to share resources which are already on servers

REMOVE: remove a stored resource

The server side has serval functions: Maintains a list of server records; Auto-exchange the server record list in certain time(the default setting is 10 minutes) and relay(when server recive a QUERY with relay set true, then it will wend QUERY to all servers sotred in record lists).

During development, we had trouble with how to store the resources through PUBLISH. The solution is to use arrays on server side which can be traversed and printed out through initiator. Another technical challenge is how to handle with the JSONObject during PUBLISH. In the end we set client-side directly default to null, while server-side directly storage.

All the basic functions on client side are well achieved. But there are still a few features are not implemented, like get debug information. In the report, we will further analysis the EZShare system in scalability, concurrency and some other challenges.

# 2. Scalability

Analysis of EZShare system scalability is particularly important. As the scale of the system grows, the performance requirements for the underlying storage system are increasing. The requirements for system storage capacity and performance are also increasing. So there are several scalability issues to consider.

## 2.1 Load scalability

Load scalability refers to the system can smoothly transport information under changing workloads by adjusting the resources. In the adjustment, process does not produce more than the specified limit of time delay and resource consumption. As can see, when there is a high load jam between the servers, it will cause unnecessary waiting time and congestion. For example, all other servers want to QUERY the same resources from 'server-A' then they will wait for the query in proper order. Thus, the scalability of system is limited. Another example is when a cient QUERY with relay, they the server must send this QUERY to all servers recorded in the lists. This cost a lot unnesscay actions.

## 2.2 Spatial scalability

Spatial scalability is a limitation of the system's regulatory resources. That is, the maximum amount of resources (memory, disk space, etc.) and the load grows linearly in the case of increased load, which prohibits the increase of resources through infinity load the practice. We use JSON to send resources and all of them are stored in the server-side array. Although doing this is light and fast, it will limited the amount of resources that could be maintained by the server. Even worse, once the server is closed or broken down, the whole resources will get lost. In this way, EZShare system is more suitable for short-term storage of small groups. As the size of the user grows, the servers have to be stable and work all the time.

### 2.3 Structural scalability

Structural scalability refers to the realisation of the structure can be easily carried out on the expansion and adjustment of resources. Since the system does not have a core or a centre node, expansion nodes are easily to join in by using EXCHANGE.

However, this factor may cause another problem: it will cost a lot to maintain the consistency between severs and to detect node failure. It is fine when the servers amount are keep under hundred, but when the number of nodes expanded to thousand or million, the efficiency of the entire system will be reduced to a very low extend. The solution is to control the auto-exchange function when the traffic load is high between servers. For example, We can set up a traffic monitor, when the load reaches the threshold, turn off the auto function to ensure that the normal communication between the server. This solution is not perfect because it may cause varying degrees of delay and disrupt consistency.

## 3. concurrency

With the multi-user simultaneous use the EZShare system, concurrency issues become very important. The two main issues are read-write synchronization and write-write synchronisation[1].

### 3.1 Read-write synchronization

This problem happens when the client wants to QUERY or FETCH a resource and other clients just finished PUBLISH, SHARE or REMOVE. For example, client-A wants to QUERY a resource with the URI 'file://test.txt' but immediately, client-B just finished changing the file URI into 'file://folder/test.txt' or changing the file to a JPG document. Then client-A will not got any response because the previous file does not exist anymore.

### 3.2 Write-write synchronisation

This situation may occur when clients using the command below: PUBLISH, SHARE and REMOVE. Some will report a system error and some will be accepted but get a wrong resource in the end. For example, if client-A wants to share resource-A but at the same time, client-B just remove the resource-B. So client-A will receive a system error '{ "response" : "error", "errorMessage" : "missing resource and\ /or secret"}. Another example is that, if two clients want to publish the same resource(which have the same name, same URI, same channel but different owner), technically speaking, the system will not allow this happen. The fact is, when both clients are processing the PUBLISH function, they will pass the condition review. The reason is there is no same resource exits at that time, so the server will failed in checking duplicated resources[2].

### 3.3 Solution-Zookeeper lock

The most easy but useful way is to lock the shared resources when it is using. Thus, duplication and unexpected can be eliminated. For example using Zookeeper lock.

The general idea is that when each client locks a function, a unique instantaneous node is generated in the directory of the specified node corresponding to the function on the zookeeper. Determining whether to obtain the lock is very simple, just let the node be the smallest one. The instantaneous node can be simply deleted when the lock is released[3]. At the same time, it can avoid service downtime caused by the lock cannot be released, for instance

deadlock problems. But there is a drawback, when the traffic load is enormous, like thousands of threads are waiting for processing the shared resource, the waiting time will also be immeasurable.

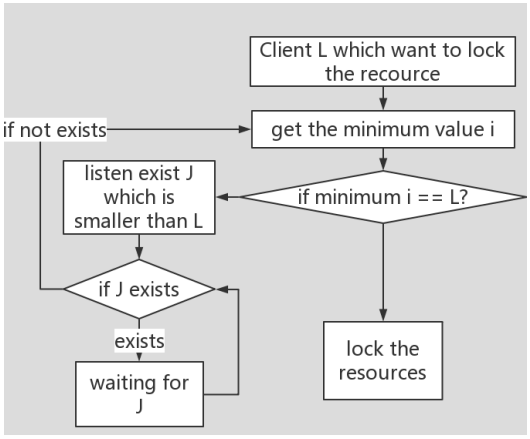


**Figure 3.2 flow chart of Zookeeper lock**

# 4. Other Distributed System Challenges

## 4.1 Independent failure

The EZShare system is consisted of many independent clients and servers, So any one of the individual is damaged may affect the entire system. The best way to handle demaged node is to abandon it. For example, if server-A is doing the auto-exchange with server-B, but A fail to get record lists from B. So B can be regarded as a broken node and then all the records from B which are already stored in A should be deleted.

## 4.2 Security

With no supervisors or any administrators, the security of the system is difficult to ensure. Any clients can PUBLISH resources without permit, therefore, there will be users to malicious release to occupy the system and network resources. One way to handle this issue is to limit the threads within a certain period of time. We can set up a monitor to watch for commands from the client. If one client send too much command lines during a short time, then a process will be executed to prevent the new command from this client.

The other issue is we find the SHARE functionality exists for design vulnerabilities. For example, a recourse has never been executed SHARE, but users can still FETCH it if they know the file path. So bascily, SHARE becomes useless when a file path is known.

## 4.3 Tolerating failure

Attempting and processing each failure that occurs is sometimes impractical, and sometimes it is best to tolerate them, such as a web server that is not available, try again later. As we mentioned at independent failures(4.1), if server-A can not exchange record lists with server-B, we treat B as a damaged server. In fact, the reasons are more complicated, such as unstable network, long time delay and other network failures. An improved approach is first hide this error, and give server-B a few chance to attempt, such as 5 times. After all the 5 attempts are failure, we will report the error and then make B a broken server.

## 4.4 Redundancy

An effective method to tolerate failures is to use redundant components or backup the resources. Multiple servers that provide the same service is a good example. However, as all resources are stored at distrebuted servers, backup the whole resources contributes less to the sharing system compared with it cost. So we may sacrifice some reliability to make sure the basic functions work.

### 4.5 Consistency

Ideally, the data can be consistent between multiple servers in a distributed environment, and if the change of the data item is successful, all users can read their latest values. However, EZShare system cannot achieve this goal due to design defects. For example, clients may get the duplicated results when they try to query across different servers. The client-A publishes a resource on server-A along with the client-B publishes a different resource on server-B(A and B are in the same primary key) at the same time, both operation will be succeeded. Because the servers A and B could not detect the same resource when the PUBLISH is processing.

There is a theorem called CAP(Consistency, Availability, Partition tolerance), which believe among these three aspects, only two of them can be meet at the same time[4]. For example, to achieve CA, we can use 2pc two-phase transaction submission to ensure. Obviously, the shortcoming is to abandon the partition fault tolerance. However, once an operation fails, the entire system will be crashed down without tolerate.

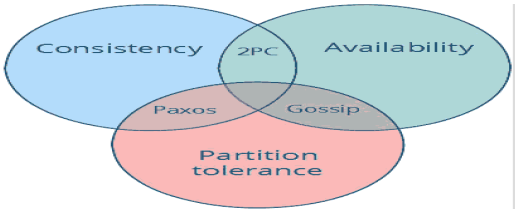


**figure 4.5 relation between CAP**

## 5. Conclusion

The EZshare system is a small and fast file exchange system and most of the functions are achieved. However, challenges are still need to be achieved, such as when the client sends a new command line, the server will create a new thread for it.

## 6. Reference

[1] Solved Problems, Unsolved Problems and Problems in Concurrency, Leslie Lamport, 1983

[2]LIU, Y. and LIN, Z. (2011). Concurrency control in distributed database system with score-based method. Journal of Computer Applications, 31(5), pp.1404-1408.

[3] IEEE Transactions on Multi-Scale Computing Systems House Advertisement. (2016). *IEEE Internet Computing*, 20(1), pp.77-77.

[4]Brewer, E. A. 2000. Towards robust distributed systems. In Proceedings of the 19th Annual ACM Symposium on Principles of Distributed Computing (July 16-19, Portland, Oregon)