1. Introduction

We implemented an asynchronous request-reply model between servers and clients to build a distributed file sharing system 'EZShare'. The server side provided to clients with the functions of publishing a public resource, sharing an existing resource on server, removing a stored resource, querying a resource by different attributes across servers, fetching a shared resource and informing the server of a list of other servers.The server itself was facilitated with the function of periodically exchanging server list with a random selected server and the function of preventing an IP address from frequently accessing the server within a short interval, while the client was facilitated with the function of timeout in case of connection unstable and long waiting time.

One of the technical challenges we met during the development was the connection sometimes closed before the delayed incoming data stream was read. So, we let the receiving side to sleep for one second and then start to read. This simple implementation solved our problem.

Another technical challenge we were faced with was how to let the server do periodically exchange when the server still listened for clients' connections. At last, we decided to create a new thread for this function to run separately.

In this report, we elaborated many other challenges like scalability, concurrency, consistency and failure modelwhich we either dealt with or had a deep discussion.

2. Scalability

For a file sharing system like EZShare, scalability was reflected by ensuring the effectiveness of the system while amount of resources and amount of real-time traffic were increasing. We considered scalability issues from two aspects: vertical and horizontal scalability.

2.1 Vertical Scalability

We identified resource storage as an issue that constrained the scalability of each single node inthe system. Currently, we used a HashMap to store all the resources in memory, which limited the amount of resources that could be maintained by a server. And once the server was closed, the resources saved by this server were lost.

Thus, we chose NoSQL database as a solution to this problem. For one thing, all the resources were in form of JSON, which was a perfect fit for NoSQL database like MongoDB, where one JSON could be saved as one record. It would be much harder for us to query or remove a resource if we applied relational database because according to the normalization principles, we would have to separate the data for one resource into several tables. For another, NoSQL database would be easier to apply for partitions and distributed designs, while this would be harder for relational databases.

2.2 Horizontal Scalability

We identified two important issues caused by large amount oftraffic between servers that may affect theperformance of the system. Firstly, if the real-time traffic was of high load, for example all the servers tried to query others for resources, the locks used to ensure the concurrency of shared resources would result in unacceptable waiting time, which was a factor that limited the scalability of our system. Secondly, if the number of servers was getting higher and higher, the communication among servers to ensure consistency and to detect node failure would become more expensive. For example, the periodically exchange method used to exchange server information and detect ineffective node may become less efficient when the number of servers got extremely large.

Possible solution for this issue was to sacrificesome degree of consistency by constraining functions of auto-exchanging and querying between servers. For example, if the real-time traffic was of extremely

high load, we could stop the auto-exchange function for a moment and restart when there was less traffic. This mechanism sacrificed the real-time consistency under high load and tried to ensureeventual consistency.

Discussions regarding tradeoff among scalability, concurrency and consistency would be further made in following parts of this essay.

3. Concurrency

3.1 Concurrency Issues

We identified two major concurrency issues for our EZShare System: write-write conflicts and read-write conflicts.

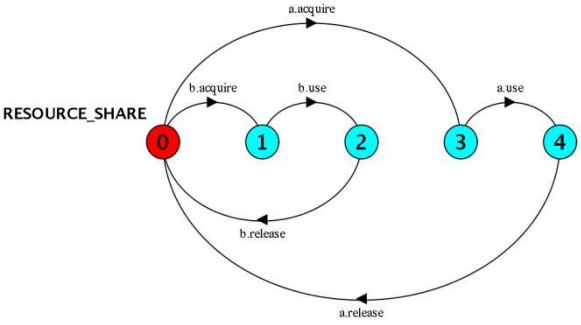
Write-write conflicts represented for the situation where two clients tried to publish, share, remove, or modify the same resource at the same time. For example, if two clients wanted to publish a resource with the same uri and same channel but a different owner, they would possibly both get a successful response, which was not supposed to be allowed by the server. This was because when both threads called the publish function and checked the existing resources, they did not find any duplication with the resource to be published, so they would both publish the same resource as a legitimate resource.

Read-write conflicts represented for the situation where one client queried the server with read authority, while the other client modified the queried resource right after the query, which let the previous query no longer effective. For example, if one client used query command and obtained a resource uri as 'file://dog.png' and tried to fetch the file later, while the other client finished updating the uri of this file to 'file://puppy.png' right after the query, the fetch command with uri 'file://dog.png' from the former client would fail.

3.2 Potential Solutions

A straight-forward solution to concurrency issues was to lock the shared resources each time it was used, no matter for reading or writing. Although this would thoroughly solve the above issues, there would be an unacceptable waiting time under high traffic load and the availability of the server was in some way sacrificed, and as we mentioned before, the scalability could not be ensured either.

An alternative solution was to lock the shared resources only for writing operations such as publish, share and remove. This method would avoid the 'write-write' conflicts but still have some problems. On one hand, the 'read-write' conflicts still existed and might cause the inaccuracy of query. On the other, the waiting time could not be shortened if lots of clients tried to use the writing commands.
Therefore, a better way to solve the concurrency problem as well as to ensure the accuracy of query and acceptability of waiting time would be locking on a smaller basis. We may partition the shared resources to several parts, and block each part only when the operation required using this part. This way, we may only need to block one part at a time for a client, while the other parts were free to be used. The smaller one part was, the less waiting time client may need to spend. However, there was another tradeoff between the size of resource chunks and storage overhead or query overhead, and we could not put each resource in a new table. Furthermore, we may add a load balancing function for utilizing different parts of the resources. For instance, if function A wanted to query resources part I, which was locked by function B, then instead of waiting in queue for that part, the load balancing function would recommend function A to use resources part II first.



Graph 3.2.1 **Ideal Resource Sharing Process**

4. Other Challenges

4.1 Consistency

Consistency was one the issues that our system did not handle very well. In other words, we did not ensure the consistency of application logic and the consistency of resources across different servers. For instance, the server should not allow the same resource to be published twice, but if a client published a resource on one serverand anotherclient published the same resource on another server with the same primary key, both operations would succeed because the servers did not check the resources saved on other servers while publishing. However, when the client tried to query across different servers, they would get the duplicated results.

One solution to this problem was to keep a centralizeddatabase that all servers could get access to. But a centralized database design was not easy to extend, and we may sacrifice scalability and partition tolerance.

According to the CAP theorem, we ensured the availability and partition tolerance for our system by storing the resources in a distributed manner, it would be impossible to ensure consistency at the same time. Nevertheless, we may ask servers to exchange their resources when there was less traffic from clients to check whether there were any resources breaking the logic rules and to make revision. By doing so, the resources saved on all servers would eventually become consistent with each other.

4.2 Reliability and Failure Model

Although we had considered the situation of server failure and had implemented the timeout mechanism on client side and the periodically exchange function on server side, there were still some challenges regarding system failure modeling.

One of challenges was that we could not distinguish a server failure from the network crash. For instance, when server A periodically exchanged server list with server B and found server B could not be connected, server A would consider server B as disabled and removed server B from server list, while in fact the connection failure was due to the instability of network. A similar example would be the case wherea client tried to get response from the server but failed, the client side could consider the timeout as a server failure, but it could also be a network failure, or just a slow response.

A possible solution for this problem on server list exchange was to test more times later after one-time failure, which meant we would not consider a server to be broken only after one time of failure. We may keep the unreachable server on server list until we tested three times at different time with the same failure result.

It was a bit different when we considered the solution for timeout on client side, because firstly, the automatically retry from all clients might lead to the overload of server. If the timeout was caused by a slow response, the retry function would make the problem even worse. Secondly, we may also need to deal with the duplicated request problem if we implemented a retry function after timeout on client side.

Considering the above problem, we would like to use a different implementation on client side to deal with the failure. We may ask the client to regard the server as fail after timeout and to try a different server. Or, we may introduce a caching function on server to memorize the results of last few requests and to mitigate the workload brought by retry after timeout.

5. Limitation

Besides the topics discussed above, we also considered challenges and possible extensions in area of security, backup, thread pool, queue management and etc. However, considering the word limit on this report, we could not elaborate.

6. Conclusion

To sum up, we discussed issues and possible solutions regarding scalability, concurrency, availability and failure to extend our Distributed File Sharing System EZShare. Nevertheless, we suggested considering the tradeoffs and side effect for any solutions thoroughly, because it could be hard to have the best for both worlds.

7. Reference

[1]Coulouris, G. F., Dollimore, J., & Kindberg, T. (2005). *Distributed systems: concepts and design*. pearson education.

[2]Kondo, D., Javadi, B., Iosup, A., & Epema, D. (2010, May). The failure trace archive: Enabling comparative analysis of failures in diverse distributed systems. In *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on* (pp. 398-407). IEEE.

[3]Bailis, P., & Ghodsi, A. (2013). Eventual consistency today: Limitations, extensions, and beyond. *Communications of the ACM*, *56*(5), 55-63.

[4]D'Ippolito, N., Braberman, V., Kramer, J., Magee, J., Sykes, D., & Uchitel, S. (2014, May). Hope for the best, prepare for the worst: multi-tier control for adaptive systems. In *Proceedings of the 36th International Conference on Software Engineering* (pp. 688-699). ACM.

[5] Subject Slides and Lecture Recording

Appendix - Meeting Minutes

Date: Mar 27, 2017
Place: Union House
Team: Dr. Stranger
Attendants: Jiacheng CHEN, Jiahuan HE, Jiayu WANG, Yiming JIANG

Brief: The team established an overall understanding of the problem, exchanged ideas about the timeline and targets at each stage, broke the tasks down and allocated the workload for each member for the next seven days.

1. **Timeline**
   The whole period for development would be 4 weeks. We all agreed on the first week to work out a couple of elements that has lower dependencies, and then implemented all required functions on a working product by end of the second week. For week three, we planned to perform detailed testing on all aspects of the system and finally, writing reports in week four.

2. **Tools and Communications**
   We decided to use Git to perform version control and created repository on Github. Besides, we would use our WeChat group as preferred communication tool, and we would hold a weekly meeting on every Monday.

3. **Project Decomposed**
   We discussed the basic client-server model of the system, and decided to start with a simple client, which would deal with basic command and server connection. As for the server, we planned to start with the six basic functions, which is PUBLISH, SHARE, QUERY, EXCHANGE, FETCH, REMOVE.

4. **Task Allocation**
   Jiacheng CHEN : PUBLISH and SHARE function
   Jiahuan HE : FETCH and REMOVE function
   Jiayu WANG : QUERY and EXCHANGE function
   Yiming JIANG : Client Side and Parsing function

Date: April 3, 2017
Place: A Table near Stop 1
Team: Dr. Stranger
Attendants: Jiacheng CHEN, Jiahuan HE, Jiayu WANG, Yiming JIANG

Brief: Our team went through what had been achieved by last week, agreed on a few admin issues, made overview of requirements again, and specified the tasks assigned to individuals.

1. **Review Previous Tasks**
   First of all, we reviewed the tasks done in last week. We implemented a Resource class that can be shared

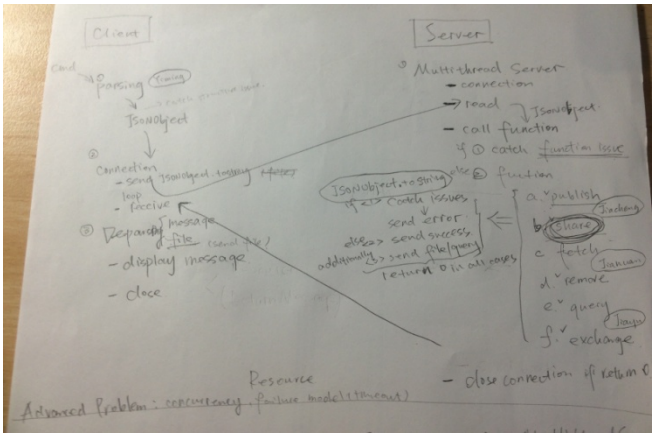
by all our functions, and then we completed two of the functions on the server side.

2. **Admin**

   We had decided to assign the task of meeting minutes to Jiayu Wang, because she was experienced in summarizing and displaying information. We had emphasized on creating new git branch while working concurrently and pull the most updated resources in time.

3. **Requirements Overview**

   We went through the whole process from typing in commands to delivering final messages together, clarified anything unclear, and discussed the expected input and output of each function. We attached one of the discussing drafts below.



Pic 3.1 **Discussion Draft**

4. **Implementation Specification**

   In addition, we discussed several implementation details for a few functions. For instance, the transmission problem of share and fetch function. We decided to let each function to deal with their own errors and transmission separately instead of centrally.

5. **Task Allocation**

   We had agreed on the importance of submitting jobs on time, and made the following allocation.
   Jiacheng CHEN : PUBLISH and SHARE function
   Jiahuan HE : FETCH and REMOVE function
   Jiayu WANG : Multi-thread Server
   Yiming JIANG : Dealing, sending and receiving messages on client side

---

Date: April 10, 2017
Place: Project Room 1, ERC
Team: Dr. Stranger
Attendants: Jiacheng CHEN, Jiahuan HE, Jiayu WANG, Yiming JIANG

Brief: Our team celebrated the milestone that had been achieved, confirmed the time for next meeting, planned individual tasks for the coming week, and discussed a few advanced features for the project.

1. **Celebrate the achievements**

   Prior to the meeting, we celebrated what had been achieved so far, which was a working system with all

the basic functions.

2. **Arrange time for next meeting**

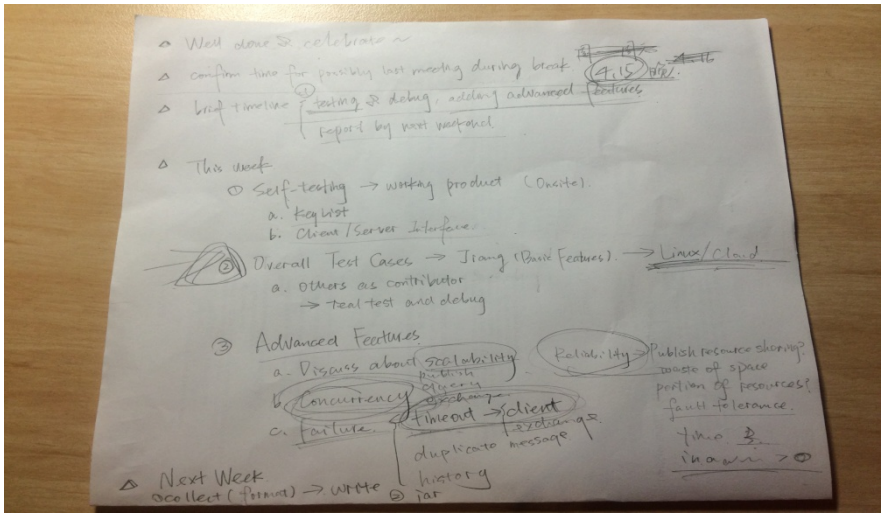
   Since next week is the break week, we rearranged the meeting to this Saturday (April 15).

3. **Review Previous Tasks**

   We reviewed tasks had been done in last week, including a client side with marshalling function and a server side with all six required functions. The client side and server side could establish connection and deal with basic commands.

4. **Task Allocation**

   For this week, we would start by self-testing. Then, we would establish a full list of test cases and start an overall testing. After that, we may try to discuss and implement advanced features like scalability, concurrency, reliability, and failure models.



Pic 4.1 **Agenda and Task Allocation Draft**

We had made the following specified allocation:
Jiacheng CHEN : self-testing PUBLISH and SHARE function and testing according to test cases
Jiahuan HE : self-testing FETCH and REMOVE function and testing according to test cases
Jiayu WANG : adding timer and secret to server, testing QUERY&EXCHANGE functions, consult tutors
Yiming JIANG : generate test cases, add timeout to client, debug whole system, and consult tutors

---

Date: April 15, 2017
Place: Project Room 5, ERC
Team: Dr. Stranger
Attendants: Jiacheng CHEN, Jiahuan HE, Jiayu WANG, Yiming JIANG

Brief: Our team completed FETCH function on client and server side, tested and debugged all functions together, and assigned tasks for the next week.

1. **Completed the fetch function**

   First of all, we completed the FETCH function one both Server and Client side, debugging it to make it

work.

2. **Test and debug all the functions**
   We testedthe whole project all together, and fixed most of bugs that were found during self-testing last week.
3. **Discussion about report**
   We discussed a few features of the project, and the feasibility of these features on our system.
4. **Arrange time for next meeting**
   Since next week is the last week before the due date of this project and is a break week, we arranged an online meeting on next weekend (April 22).
5. **Task Allocation**
   For next week, we would write the report for the project. According to the project specification, the report will contain 5 parts including Introduction, Scalability, Concurrency, Other Challenges and conclusion.

   We had made the following specified allocation:
   Jiacheng CHEN : Draft Concurrency part of the report
   Jiahuan HE : Draft Other Challenges part
   Jiayu WANG : Draft Introduction, Limitation and Conclusion parts. Also integrate the whole report.
   Yiming JIANG : Draft Scalability part

Date: April 24, 2017
Place: Project Room 3, ERC
Team: Dr. Stranger
Attendants: Jiacheng CHEN, Jiahuan HE, Jiayu WANG, Yiming JIANG

Brief: We tested our project by running several servers, and improved the report together.

1. **Testing the project**
   We testedthe whole project all together, and fixed most of bugs that were found during server testing.
2. **Improving the report**
   We discussed the imperfection of the report and improved it.
3. **Task Allocation**
   For next week, a few finishing touches are needed to make the project completed. And we had made the following specified allocation:
   Jiacheng CHEN : Constructing the jar package
   Jiahuan HE : Testing the project
   Jiayu WANG : Testing the project and finishing the report.
   Yiming JIANG : Testing the project