COMP90015 Distributed Systems
Project 1- BitBox

Group: United Ace

Name	ID	Login	Email
Wanmin Chen	941065	WANMINC	wanminc@student.unimelb.edu.au
Junjie Huang	1016904	JUNJHUANG	junjhuang@student.unimelb.edu.au
Tong Mu	1004452	MTM	mtm@student.unimelb.edu.au
Junda Li	978005	JUNDA	junda@student.unimelb.edu.au

Introduction

In this project, we aim to develop a peer-to-peer(P2P) system called BitBox. The system consists of many peer programs located in different machines. Each peer program can monitor local file system changes and share files with other peers through P2P protocol. In addition, any changes (deletion, creation, duplication etc.) in the local file directories of one machine can be synchronized in other peers through P2P protocols.

Several technical challenges were encountered while developing the project. One of the main technical challenge is the management of sockets during socket listening. We adopt a table-driven-like approach to handle this task.

As for the outcomes, through this project, we finally develop a P2P system that support file synchronization among small number of peers. It supports synchronization of file status change including creation, deletion, and file write.

Fault detection/tolerance/recovery

In the process of system debugging, due to the careless programming, there will be various faults which might affect the operation of the system. One of them is about the protocol format. If the protocol format is not correct, it will have a great impact on the system. The format of the system processing information is fixed and cannot be as flexible as the brain. For example, the message received by the system defaults to JSON format, but if it is not designed as correct format, then the system can't recognize this message, and can't receive the Request or output the Response because the system can't get any values from this message. Similarly, if the received message is not encrypted with md5 or the encoding for the message is not UTF-8, the system will also throw an exception.

In order to solve this problem, the system will make a judgement before processing these requests or responses. First, when the system receives a message, if this message is written by JSON, the system will certainly read the different values inside. If the system does not get anything, it determines that this message does not conform to the format and sends back a "Non-compliant format" message. If the parameters are read, it is judged whether the format of the corresponding request or response is correct, whether or not md5 encryption is used, and so on. Not only can this determine and execute the corresponding request or response, but also directly avoid the failure due to the invalid protocol message.

Another problem is that when the system throws some exceptions like IO exception when writing bytes. The step for the system to face the problem now is just set the key ("status") to be false in Response and send this response to the client. We may make a revision to resend this FILE_BYTE_REQUEST automatically in limited times to make it more effective.

Scalability

A distributed system is called scalable if the system can still function properly as the number of users/ client increases. Although our P2P system runs smoothly when we tested the connection for up to 4 peers, it can be anticipated that as the number of users increases, some issue related to scalability could emerge. As the number of users increases, the number of shared files also increase since each peer should have the same copy of files. If some peers modify a file, all other peers will have the updated file. These features, however, will cause pressure on the internet bandwidth if there are multiple peers modify different files at the same time. Moreover, large P2P network may also encounters consistency problem. There may be delay in file changes in different peers. Such problem could affect the system's stability.

In addition, shared files are loaded in memory instead of hard disk (Silberschatz, Korth and Sudarshan, 2011). Since the volume of memory is finite, it is not capable to support the demand of a P2P system while there is a large number of users. To address this issue, we may need to adjust the architecture of our system. For example, we may introduce meta server. The meta server will store the data about the file distributions in the whole system.
The meta server will be visited when some peers want to access files.

Referring to our implemented system, we also identify a place that could reduce system performance as the P2P network expands. There is a class called SeverMain. Inside the SeverMain, there is a variable called peerlist that storing Sockets. Java arrayList is used to store the information of Socket of all peers and this arrayList will be iterated in the program.

The complexity of the iteration of arrayList is O(n) and this could slow down the system if the number of peers is very large.

An alternative approach is using HashMap because the time complexity for accessing a particular element in HashMap is O(1), which is more efficient than using arrayList. Such an advantage may not be obvious for a system with only four peers but it is important for a large P2P network.

Security issues

Peer to peer system is used to implement real-time network communication between different peers. A complete peer to peer system can read requests, deal with the requests and send relevant response to the sender at the same time. However, if there is an unclear third party joins the network, it will lead to some potential risks which are from the third part's malicious attacks. For example, sending a huge number of Create requests or simulate a new peer to join this network in order to achieving a large number of server sockets. The purpose of doing this is to occupy the system resources and/or prevent the system form completing the communication with other peers. Similarly, the request sent by the other party may also contain some malicious attack orders. When reading these instructions, there is also a possibility that the system will be maliciously destroyed. What's more, the third part might also intercept and capture the package by using a fake host and port number instead of other peers.

For the possibility that files may be intercepted and tampered with during transmission, the current system encrypts the transmitted documents using md5 encryption which can improve the safety of the document. For the input request, the system able to check if the file's name is safe by using the isSafePathName function provided by FileSystemManager class. In addition, for the specification of the input json message, the system uses Enum method to check it that contains the required fields of the required type or not. In addition, this system has a limitation for the number of peers that can connect to this system in the same time.

By analyzing the above-mentioned possible security issues, it is crucial to determine the identity of the source of the information. So in this case, setting up a mechanism for check the source of information and decide whether to authorize it can be considered. Check the identity of the third part who is first time sent request, if it is safe and reliable, then give a partial authority, and allow it can do some operations such as adding, deleting, checking and modifying in a definite scope , if its identity is not clear, then the connection request is rejected. But when it comes to establishing this mechanism, how to verify the security of the visitor's identity, and what method is used to verify are still need detailed discussion.

Other Challenge

Concurrency issue can occur when more than one peers within the system access same files at the same and some of the peers try to change the status of the file at the same time. For instance, peer A deletes a particular file while peer B tries to read the file and peer C is updating the file. In this scenario, peer B and peer C may encounter unexpected error messages from the system or even cause the system to be down. Hence, a P2P distributed system must be designed so that the system can still be consistent across all peers even some peers try to modify the status of the system simultaneously.

There have been several proposed approaches to handle concurrency issue. To start with, one way to do is to not allow the peers to access same resources simultaneously in the backend. That is, making access from peers sequential access and lock the resource while one peers is visiting it. In details, the system will rank the accesses based on timestamp and grant access one by one. The resource will be blocked to prevent multi-access until granted peers finished their tasks. However, the drawback of this approach is the cost of slowing down the system.

Another approach is the use of semaphore. Semaphore is one of the common methods of interposes communication(J. Cao, Y. Ran et al. 2010). It will either allow or reject access to resource. In details, semaphores are variables that are used to ensure process synchronization. The mechanism is to use two atomic operations(Meador, 2018): wait and signal. An example of semaphore is the Binary Semaphore. In Binary Semaphore, the semaphore variable can only have value 0 or 1, corresponding to locked or unlocked. The advantage of semaphore is that it not only follows mutual exclusion rule but also more efficient than sequential control.

Although semaphore is a good candidate for solving concurrency issue, it also has a drawback, which is the creation of deadlock. To reduce the risk, we must ensure the wait and signal operations function in a correct order.
Nevertheless, concurrency is not the last challenges of the P2P distributed system. There are also other challenge such as heterogeneity issue than also require attention from software developers.

References

J. CAO, Y. RAN and Z. XU, 2010. The Design and Implementation of Distributed Semaphore, *2010 International Conference on E-Product E-Service and E-Entertainment* 2010, pp. 1-4.

Meador, D. (2018). Semaphores in Operating System. [online] Tutorialspoint.com. Available at: https://www.tutorialspoint.com/semaphores-in-operating-system [Accessed 4 May 2019].

Silberschatz, A., Korth, H. and Sudarshan, S. (2011). Database system concepts. New York: McGraw-Hill.