CSCI 53700 : Fall 2022

Assignment Number 2

Abrar Jahin

Introduction

Distributed systems may have no physically synchronous global clock (for this implementation as there are some like NTP servers or TSA servers), so a logical clock allows global ordering on events from different processes in such systems [1]. Clocks need not be synchronized absolutely if two processes do not interact; it is not necessary that their clocks be synchronized because the lack of synchronization would not be observable and thus not cause problems.

Lamport's bakery algorithm is a computer algorithm devised by computer scientist Leslie Lamport, which is intended to improve the safety in the usage of shared resources among multiple threads by means of mutual exclusion [2].

Simpler way to implement this use increments its counter before each event in that process where event could be either Send, Receive or Internal communication. When a **process object (PO)** sends a message to **Master object (MO)**, it includes its counter value with the message. Similar way on receiving message the counter of recipient is updated (if necessary) by the value send by MO, the counter is then incremented by I before the message is considered received. In our implementation, MO once receive such message from any PO it will calculate the average of all POs counter value including its counter value then calculate the offset for each and finally broadcast to all POs.

In this assignment, we supposed to implement a simple distributed system based on the principle of clock consistency, associated drifts, inter-process-communication, and the end-to-end argument. Here section 2 is briefly covered the Interaction model, Failure model and Data encryption. The implementation is covered in section 3. Finally, section 4 represents the analysis based on the simulation take place between 4 POS and MO with different event probabilities.

**Interaction Model**

* What are the main entities in the system?

4 POs which are Process Object, and a 1 MO is Master object. Each POs and MO will have its own logical clock. All POS and MO are allowed to perform three different type events, are: send, receive and internal communication with different probabilities.

* How to Interact?

By message passing in asynchronous fashion they interact with MO, where each PO maintains a message queue. POs of this system follow end-to-end principle; where one end is PO; is responsible to send its current counter value along with the message. Other end is MO which is responsible to maintain the clock consistency in the system by broadcasting the current clock value (which is basically request number) to all existing POs. Here inter-process messages get encrypted at the sending end and get decrypted at the receiving end that contains logical clock values from POS and offset value from MO.

* What are the characteristics that affect their individual and collective behavior?

By changing the probabilities of three different events inside POS and MO have the effect on both individual and collective behavior. More and More communication with MO will lead to less drift compare with others.

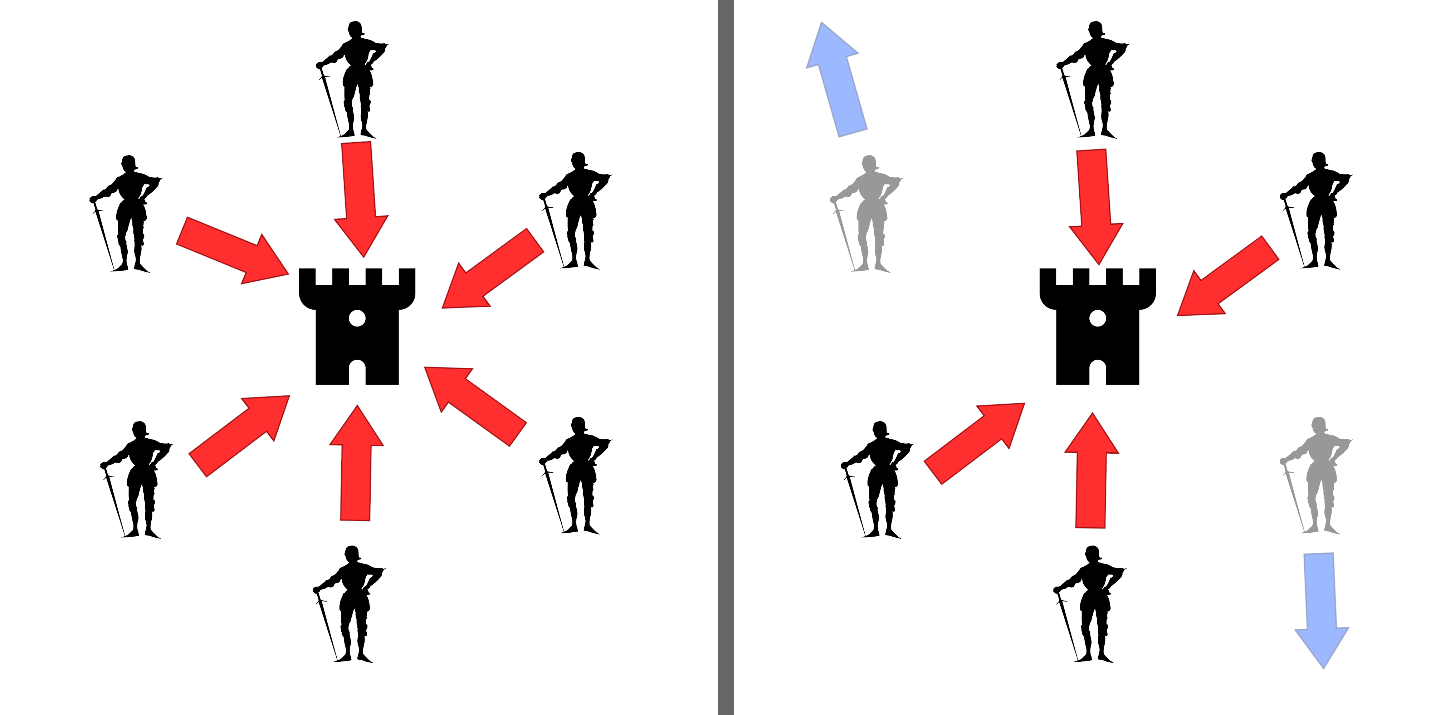


Figure 1: Byzantine fault : If all generals attack in coordination, the battle is won (left). If two generals falsely declare that they intend to attack, but instead retreat, the battle is lost (right)

**Failure Model**

* There will be arbitrary failure only during message receiving. The arbitrary behavior could be either not to update the logical clock value or to move forward the logical clock value by 100 with different user provided profanities for each event.
* Due to message buffer overflow may also result to omission failure. This situation may arise when POS send message very frequently and MO unable to handle that lead buffer become full and message can be lost.
* A Byzantine fault is any fault presenting different symptoms to different observers. A Byzantine failure is the loss of a system service due to a Byzantine fault in systems that require consensus. In a distributed system, this kind of faults can also happen.
* **Pros**: Due to asynchronous principle it is possible to achieve higher concurrency as after sending it don't need to wait for the response from MO.
* **Cons**: Here the message may get lost as there no checksum mechanism applies between two ends so no guarantee that message delivered correctly or not.

**Data Encryption:**

Data encryption is important because it allows to securely protect data that you don't want third parties to have access to. As the network is an open platform so the data that are transmitted may contain sensitive information and can be exposed to anyone else. Therefore, data encryption mechanism is also introduced in our implementation. Here data get encrypted in the sending end point and get decrypted in the receiving end.

Timeline

Description automatically generated with medium confidence

Figure 2: Data Encryption mechanism

The encryption and decryption technique are implemented in both POS and MO end point. So that when one PO send message to MO at first, it's encrypted the time stamp and send it to the MO. In the MO it has decrypt mechanism apply that on the received message. Later when MO needs to broadcast the offset to others POS it again applies the encryption mechanism. For this assignment, we don’t need to implement any encryption.

**Implementation**

I have implemented the model with a client-server model. All the functions are distributed in different classes. Let us discuss all in brief.

Client:

The client is the most straightforward part. For the client, there is mainly 2 class:

1. SocketClient.java (Which is **runnable** and runs the client)
2. ClientRequest.java (Handles the socket communication works for the client)

Server:

The server consists of mainly these classes:

1. SocketServer.java (Which is **runnable** and runs the client)
2. UserLogin.java (Responsible for user management)
3. ServerRequestHandler.java (This file handles the multi-threaded server and provides all the responses as well)

There are some standard classes as well which are used by both clients and servers. Let us have a look at the files in brief-

1. AES.java (Used for AES encryption)
2. Encryption.java (Used for encryption but was replaced by AES.java later time)
3. FailingModel.java (Providing another failing model like a server crash or network crush)
4. FileHandler.java (Responsible for file conversion to chanks and file merging to actual format from chanks)

For my implementation, I am passing username, password and file simultaneously to the server, and the server is processing the request. If the bid is successful, the server is returning a success. If failed, it produces a fail message. If anything terrible happens, the server or client will not crush; it will catch the exceptions and show the results for the exception. If the server or client is disconnected for any circumstances, they are also automatically connected if the network is fixed.

**Implementation Coverages:**

I have implemented this thing as bonus implementation-

1. Multi-threaded Server
2. Fault tolerant server
3. Fault tolerant client
4. Data passing through serialization and de-serialization
5. Fault counting through exception
6. File transferring through chanks
7. Encrypted file transfer
8. Login check, if login failed, file is not uploaded or rejected

I have implemented counter for the exceptions so that this implementation can be extended easily. And all the

**Result Analysis**

For the initial experiment we start with by setting un-equal probabilities where SEND and RECEIVE event probability is set to 85%+ and 93%+ and the result is shown in Figure 3, 4 and 5.

Figure 4: Server failure analysis: unequal probability of send and receive

In figure 4, we are seeing that data failing rate is almost consistent till 12300 clock count, after that, the failure rate is decreasing snidely. And after that, the failure rate is also becoming almost consistent.

Figure 5: Client Failure model 1 : relatively low probability of receive

Again, for a low probability (-7%) of RECEIVE event and high probability (-15%) of SEND event we collected the experimented data that represents in figure 2. If the RECEIVE event has much lower probability than SEND event, then the message queue can be out of date that may lead to serious issue regarding synchronize end up with less probability to correct the time. Therefore, it reflects that, with the changes of the probability of events will have impact on the clock synchronization process. With high probability of RECEIVE event the drift will be smaller. However, in this case the global clock value will increase as they will have more and more receiving events that might lead to Byzantine failure too. In our case Byzantine failure if occurs then it may increase the clock value by 1 unit. We are giving equal weight to all kind of failures. If we need, we can add weights on different kind of failures, but for this assignment, we didn’t need that.

In figure 5 and figure 6, we set different kind of probabilities for receiving data and sending data. And we have added clock synchronization also added in here.

Again, we take the data without any clock synchronization between POS and MO that means we didn't adjust the PO's clock value after receiving the offset from MO. Without synchronization the process to complete it takes less logical clock values; however, there is a solidly drift here.

**Conclusion**

Different probability of events may lead to the time drifts. Therefore, the synchronization is required as without synchronization the drift becomes larger that might lead to the Byzantine failure. The implemented approach will help to reduce the drift and make the clock consistent among all existing POS in a network through a single MO whose responsibility is to make the synchronization happens.

References:

1. Logical Clock: <https://en.wikipedia.org/wiki/Logical_clock>
2. Lamport's bakery algorithm: <https://en.wikipedia.org/wiki/Lamport's_bakery_algorithm>
3. Byzantine fault: <https://en.wikipedia.org/wiki/Byzantine_fault>
4. A Study on Byzantine Fault Tolerance Methods in Distributed Networks: <https://www.sciencedirect.com/science/article/pii/S1877050916304641?ref=pdf_download&fr=RR-2&rr=755cdbf4da472bc4>
5. Modes of Failure: <https://medium.com/baseds/modes-of-failure-part-2-4d050794be2f#:~:text=Arbitrary%20failures%20are%20ones%20that,the%20system%20communicate%20with%20it.>