Peer-to-Peer Collaborative Photo Editor

Yajie (Angus) Zhao, Zezhi (Herry) Wang, and Shikun (Jason) Wang

May 8, 2018

Abstract

Collaborative photo editor aims to allow multiple users to edit a single picture simultaneously. In this paper, we propose a possible implementation of such a photo editor, which is based on Peer-To-Peer(P2P) network protocol. This implementation of photo editor also makes use of several distributed system protocols and algorithms including heartbeat failure detector, quorum system, and ABD algorithm. Moreover, we implement this design and verify its robustness and efficiency. Lastly, we also discuss the future work that can further improve the functionality, efficiency and stability of our photo editor.

1 Introduction

As an essential infrastructure today, Internet has the capability to connect everyone in the world, regardless one's location, or what hardware terminal one has. Innumerous tasks now can be done on the Internet by an arbitrary number of people efficiently. For instances, Google Docs allows many users to edit a same document, and GitHub enables programmers around the world to contribute to a single project. With many successes of collaborative tools in other fields, we propose an idea of collaborative photo editor since there isn't known related work done in this field.

There are many observed situations where one is reluctant to photoshop everyone in a group picture, a time-consuming task, or where one is not satisfied with how he/she is photoshopped by others. Collaborative Photo Editor is an application that dedicates to enabling multiple users to edit photos simultaneously and smoothly. The backend architecture of this application is based on decentralized peer-to-peer system. We also implement modified ABD algorithm and failure detector to ensure data consistency and fault-tolerance. Using peer-to-peer network system enables great scalability and efficiency. In this project, we will be exploring different such as ABD, to achieve data consistency fault-tolerant.

2 Related Works

In the field of collaborative work tools, there are two successful applications: Google Docs and GitHub. Google Docs allows collaborators to simultaneously edit a text document.

Google achieves this synchronization through centralized servers, i.e. all the changes were sent to servers at Google's data center and it is servers' responsibility to broadcast these changes to other collaborators.

Unlike Google Docs, GitHub does not support simultaneous collaboration but is rather a platform for version control of projects. More specifically, users can work on a project separately, and publish their newer versions of work to other group members. GitHub also develops a sophisticated mechanism to resolve conflicts that occur among different collaborators' newer versions.

Since there isn't any known related work done in the area of collaborative photo editing, we decide to be the pioneer in this field. Having centralized servers may better achieve consistency among collaborators but is very costly and susceptible to single point of failure. On the contrary, a P2P architecture seems to be a more practical and scalable solution if we can solve the issue of consistency.

In the industry of cloud computing, people often utilize quorum system to solve data consistency among replica in a cluster. Essentially any modification of the data in the group has been agreed upon by other nodes' votes [1]. One implementation of the quorum system is ABD algorithm [2] [3]. In this way, under asynchronous system, we can ensure the consistency of every read operation. Also, in the world of database, two-phase commit and three-phase commit protocols are widely used to ensure data consistency. Whenever a transaction need to commit, it has to let other nodes know and receive enough votes in order to safely commit [4].

In the following sections, we will discuss our designs that help solve the problem of data consistency in our peer-to-peer architecture by incorporating and modifying the protocols above.

3 System Design

After evaluating the advantages and the disadvantages of the existing technology, we design a system that ensures strong consistency in log files of photo modification and fault-tolerance under crash failure assumption. However, the network condition is assumed to be asynchronous, therefore our system can't achieve consensus among all the nodes. For the following sections, please refer to table 1 for the abbreviation of each term.

Term	Definition
N	Number of nodes
ACK	Acknowledgement
DEC	Decline
HB	Heartbeat
PB	Piggy-back
FD	Failure Detector

Table 1: Terms and Definition

3.1 Module Design

The back-end system supports reliable P2P communication and is designed to ensure data consistency in photo editing. It consists of quorum system, heartbeat failure detector, UDP communication, and file transfer system based on TCP connection. It sets up the connection between nodes in one group.

The front-end system is responsible for interacting with the users. Our photo editing functionalities are bounded with the front-end, which gives user instant responses as they perform operations on the photo.

Both back-end and front-end systems are located in one machine and are connected through localhost network, since in our design there is not a centralized system that serves the back-end servers.

3.2 Process Design

3.2.1 Initiation Process

We design a communication protocol that allows any node to invite other nodes to join the group, illustrated in figure 1.

Algorithm 1: Group Initialization

```
Initialization by node i:
Node i sends invitation to node j with membership list
Upon received ACK from node j:
send image to node j

Node j upon receiving an invitation:
Send ACK to the inviter and receive image file from it
broadcast join group message to all members
```

To initialize a collaborative photo editing session, the owner of the photo plays an initiator and creates a group. Then the initiator can invite other nodes by sending invitation message to them. For simplicity, we assume that the initiator is able to obtain the IP addresses of others. When the initiator receives acknowledgement from a node it invites, it connects to that node via TCP connection and transfers the image file. On the other hand, upon the invited node receives invitation message, it can choose to accept it by sending ACK back to the initiator and open a TCP socket to receive image file from the initiator. The detailed algorithm in shown in algorithm 1.

The file transfer is done via TCP connection since the reliability of TCP can ensure the completeness of each picture file. Although establishing TCP connections has large overhead, this can be ignored as the photo transfer only needs to be done once for every new group member.

3.2.2 Photo Editing Process

For the actual photo editing stage, we design a quorum-like system to ensure strong consistency in the changes made to the photo(Figure 2). Since photo editing is equivalent to

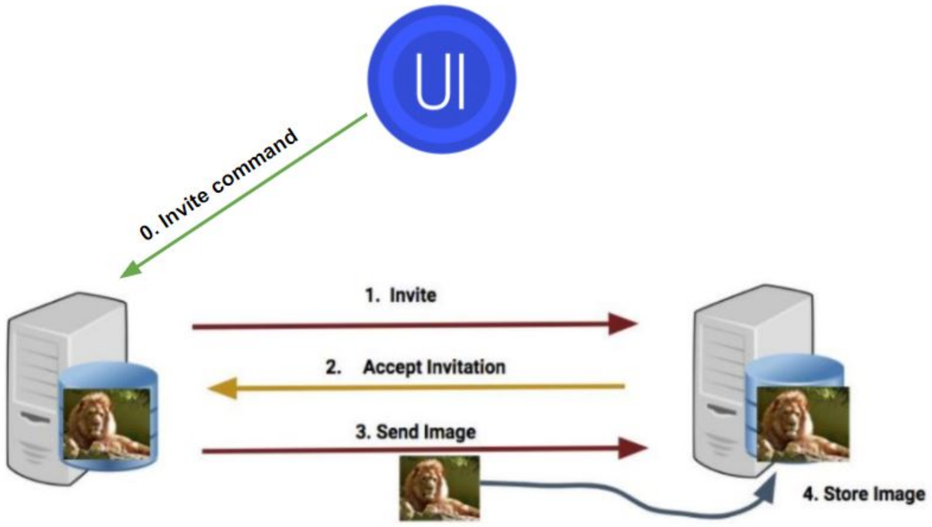

Figure 1: Process of Initiation

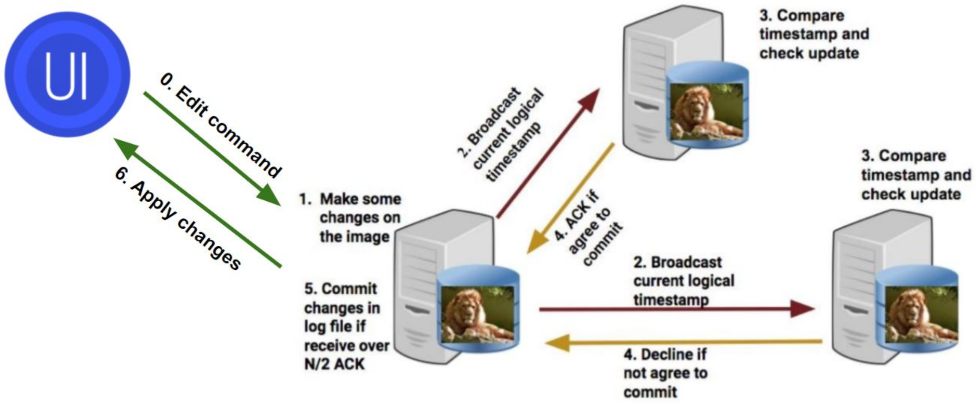
UI
0. Edit command
6. Apply changes
1. Make some changes on the image
5. Commit changes in log file if receive over N/2 ACK
2. Broadcast current logical timestamp
3. Compare timestamp and check update
4. ACK if agree to commit
3. Compare timestamp and check update
2. Broadcast current logical timestamp
4. Decline if not agree to commit

Figure 2: Photo Editing Process

keeping track of an ordered list of log files, our system is able to ensure strong consistency in the log files. The strong consistency in our context means that if a node commits a change on the photo, other nodes should also see the change and commit it. To achieve strong consistency in an asynchronous system, we combine the majority quorum system and heartbeat failure detector(Section 3.2.3). A node can only commit a change if it receives majority of acknowledgements from active nodes in the group (Algorithm 2).

In particular, the node j commits the changes made by node i only when node i has the up-to-date log file record comparing to node j's log files. With the majority quorum system, nodes are guaranteed to have overlapping write quorums, which ensures data consistency even in an asynchronous system. However, this majority quorum may not work correctly when the size of group is an even number, so we add a timeout mechanism to prevent nodes from waiting forever.

In addition, every node has a boolean value *voted* that tracks whether this node has

Algorithm 2: Quorum based 2-phase commit protocol

```
Upon Node i making a change:
Node i broadcast UPDATE message
if Upon until received N/2 + 1 ACK within T then
    Commit changes and apply to the photo
    Broadcast COMMIT message
else
    Abort operation within T
end

Node j upon receiving UPDATE msg from node i:
$T_i$ := Timestamp of node i
if Node j has voted or $T_j > T_i$ then
    send DEC
else
    Set voted := True
    Store local time of receiving msg in hold-back entry time
    if $T_j < T_i$ then
        Send REQUEST for newest log files
    end
    Send ACK
    if not receive COMMIT in time T then
        set voted := False
end
```

voted or not. This helps to avoid repeated voting during the process. After voting, nodes will wait for node i's notification of whether to commit or not.

This protocol is based on the two-phase commit protocols in database system where when a transaction needs to commit, the coordinator first sends a proposal to all the participant nodes, and commits it when it receives ACK from all of them(reference). Instead of electing a designated coordinator, every node who wishes to make a change is the coordinator, and it only needs to receive votes from the majority of the group to commit.

3.2.3 Heartbeat Process

Since our system model is under asynchronous network condition, we need to design a mechanism that ensures nodes can catch up the newest log file record. In our design, we implement heartbeat failure detector both for checking newest log file version and for detecting failed nodes.

The reason we need a Cleanup time and a Failure time is to avoid adding inactive node back into membership list. The way we merge two membership list is that if there is a new node appearing in the membership list received from node j, node i has to include that new node in its own membership list. If an entry of node i's membership list has smaller timestamps than that of the received membership list, node i has to set the timestamps

Algorithm 3: Heartbeat Failure Detector

```
Upon sending HB periodically
while True do
    HB++
    Broadcast((HB, Membership))
    for peer i in membership list do
        if peer i's HB time - current time > CLEANUP then
            Delete peer i from membership
        else if peer i's HB time - current time > FAIL then
            Deactivate the Node
    end
end

Upon node i receiving (HB, Membership) message from node j
Merge node i's Membership List with node j's
Update node j's HB time to current local time
Check Log Entry
if Node i has smaller log entry index then
    Send UPDATE request to node j for newest log entries
end
```

equal to the received ones.

3.2.4 Snapshot Process

We take a snapshot of the current version of the image every time 20 log entries are applied, by writing the current image to disk. By doing so we are able to keep track of versions of image throughout the editing process and make it possible to undo some changes by starting from previous versions.

4 Implementation

All the back-end communication protocols are implemented in Golang, which provides powerful tools such as channel that optimizes inter-thread communication. The front-end photo rendering process is implemented with Python opencv package and Tkinter UI package. The codes are in the following git repository: https://github.com/JasontheMonster/P2P-Photo-Editor.

5 Future Works

While our photo editor receives many commendations, there are much work that can be done to improve the efficiency and user experiences. As the result of time constraint, we only implement our photo editor to support local area network. One obvious feature to add is the support for communication over the Internet.

Another important feature will be the support of concurrent edits on different regions of the image. This will allow the commits to be done more efficiently.

The next feature will be developing snapshot process and minimize the amount of time required for a new member to catch up, and save memory space by deleting applied log entries.

6 Conclusions

Utilizing what we've learned from class, we successfully met our milestones of building a decentralized peer-to-peer photo editor. Although the front end is not yet well-established, we analyzed and designed out own efficient back-end algorithm to avoid concurrent writes and achieve strong consistency among group members.

From the poster session, we received several helpful suggestions. One of them is to use UDP instead of TCP for the communication between front-end and back-end. It is reasonable since UDP does not have the issue of message loss on a localhost but is much faster and has less overhead than TCP.

From this project we learned about designing algorithm and validating the claim of our algorithm. Even some slight change in it can affect the outcome a lot. In addition, we learned several coding styles that can help us debug.

We enjoy the poster session pretty much. And we think it might be better if we can have the poster session outside of CS department, and somehow share to more people.

References

[1] Cachin, Christian, *Principles of Distributed Computing*, ETHZ, Summer 2003.

[2] Attiya, Hagit. Bar-Noy, Amotz and Dolev, Danny. ”Sharing Memory Robustly in Message-Passing Systems”. *Laboratory for Computer Science, MIT*. 1990.

[3] Aguilera, Marcos K et al. ”Reconfiguring Replicated Atomic Storage: A Tutorial”. *European Association for Theoretical Computer Science*. October, 2010.

[4] Kumar, Manoj. ”Commit Protocols in Distributed Database System: A Comparison”. International Journal for Innovative Research in Science Technology. May 2016