Web Interface for Secure Decentralized Collaboration Platform

Rehan Mulakhel

School of Computer and Communication Sciences

Decentralized and Distributed Systems lab

Semester Project

June 2017

Responsible
Prof. Bryan Ford
EPFL / DEDIS

Supervisor
Kirill Nikitin
EPFL / DEDIS

Contents

Chapter 1

Introduction

The internet is the biggest network ever developed by the humans. One of the primary goals was to established a reliable infrastructure for the communications. That's why it was designed as a decentralized system. However, the applications developed for the platform were centralized. Companies like Facebook, Google or Netflix monopolized the data flow on the net.

Online real-time collaborative services, such as Google Docs and Etherpad, rise in popularity thanks to their global accessibility and convenience of use. However, the users of such systems have to fully trust service providers for data protection and preserving privacy. Potential server's compromise or pressure from state agencies can result into data leakage and fail in this trust.

A decentralized peer-to-peer collaboration can be a solution to these concerns. In fact, data are not concentrated anywhere, nor is the flow. This structure prevents attacks such denial-of-service, making it more robust thus reliable. The state agencies cannot look at the bottleneck since there is no center, making it impossible to scan easily the exchanges of the users.

BitTorrent appeared a long time ago. Tests were carried out to destroy it. But people always improved it and made it impossible to block it. More recently, the blockchain technology came. Tests were also carried out to destroy it. But no one succeeded. The evolution is moving to this direction. This lets us think that peer-to-peer architecture are better than server one in terms of security.

The application is divided into three main processes: the main one being the management which works like a local server. It generates the html and other static files to the browser and is responsible for the communication with the outside world. There is also one process for the ABTU algorithm. This one generates the operation on the document based on the remote information it received through the management(s) before sending them to the front end process. The ABTU is another project on which Damien Aymon works. The last process is the graphical interface which is the object

this paper.

The goal of this project is to design and create a web interface for real-time peer-to-peer collaboration, a text editor to start with. The challenges include alleviating the impact of potentially higher communication latency on the user interface, adapting to the serverless architecture and incorporating functionality for distributed access control.

Most of the new applications developed for the desktop are based on HTML5. The main reason behind is the usability: users do not need to download anything.

From a developer point of view, there are frameworks and open-source solutions for javascript which can be used to avoid building from scratch. In particular, we will take Ace[1] as a building block.

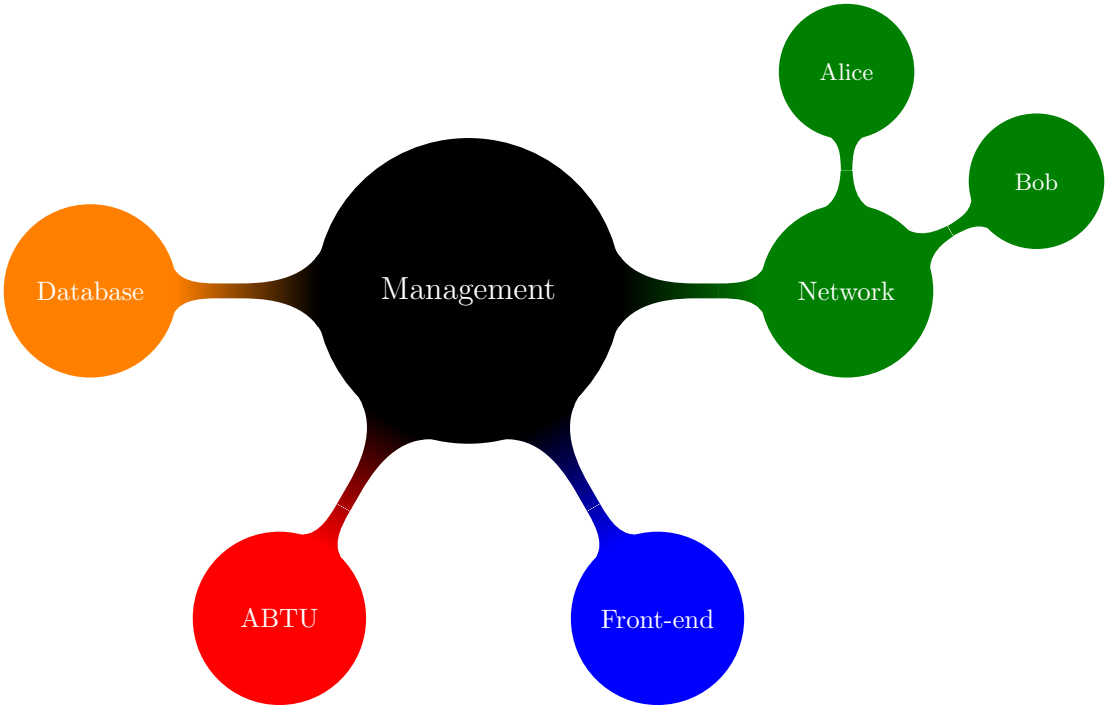
[1] `https://ace.c9.io/`

Chapter 2

Architecture

2.1 High level Architecture

The first version of P2PTE (P2P Text Editing) is not like it should ideally be. Here is the description of what the second version will look like.



The current version puts the database as a leaf of the interface (front-end).

2.1.1 The structure of the front-end

The interface is split into a controller, edit functions and collaborator functions. The master is obviously the controller which receives and sends the

message to the management.

Chapter 3

The functionalities

A collaborative platform for document editing requires some basics features like any application. For the first version, it is possible to store files and folders, and share documents with peers.

3.1 Access control

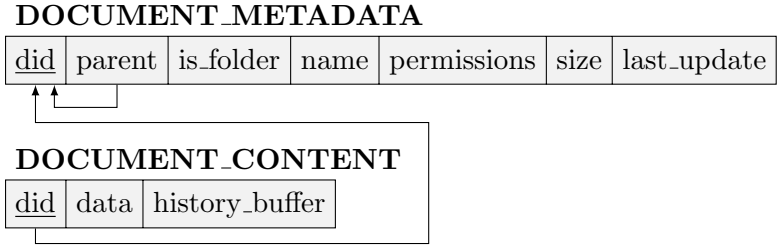
The command `ls -l` displays the the folders and the files where the user is. It also includes the permissions of each elements. There are three levels of rights: read, write, execute. Each of these can be seen as boolean with value 1 or 0. If a file can only be read, then we get: $100_2 = 4_{10}$. If a file can be written and read, then we get $110_2 = 6_{10}$. If a file can be written, read and executed, then we get $111_2 = 7_{10}$.

The same logic applies for P2PTE.

Unix	Rights	P2PTE
Read	4	Read
Read + Write	6	Read + Write
Read + Write + Execute	7	Read + Write + Share

3.2 File system

The file system is a set of structures which can be a folder or a file.



When the value of the parent field of document is `Null`, then it means its position is at the root level.

3.3 Content editing

The interface only lets the user the possibility to work with text files while being shared. The supported extensions include java, python, scala, tex and more languages. The auto completion is available by the `ctrl + space` keys combination.

It is also possible to create document with the extensions `doc`, `docx` or `odt`. This will open a WYSIWYG[1] box to edit the content. Unfortunately, the engine used does not provide basic functions like inserting or removing a char at a specific index. This makes the editing impossible by many people. Only one person can use it but it is better to use word or open office in this case... for the moment. This feature will be implemented in future versions.

3.4 Missing functionalities

The drag & drop, one of the most intuitive feature of a user friendly interface is missing.

There is not a table which makes the many-to-many relation between the collaborators and the documents. A contact directory needs to be done. This will create a better interface since only the email field of the peer is required, except the first time.

3.5 Known bugs

Since the system is decentralized, each field needs to have a time vector to maintain the whole data consistent among the distributed system.

From the interface, the permissions of the peers can be changed, the name of the documents can changed too. But these changes do not have a time vector. Therefore the changes should not be applied to user's permissions even if the interface does have this function.

One solution to fix this would be to insert those data, including the file name, into the content, inside a special tag which are invisible for the user point of view.

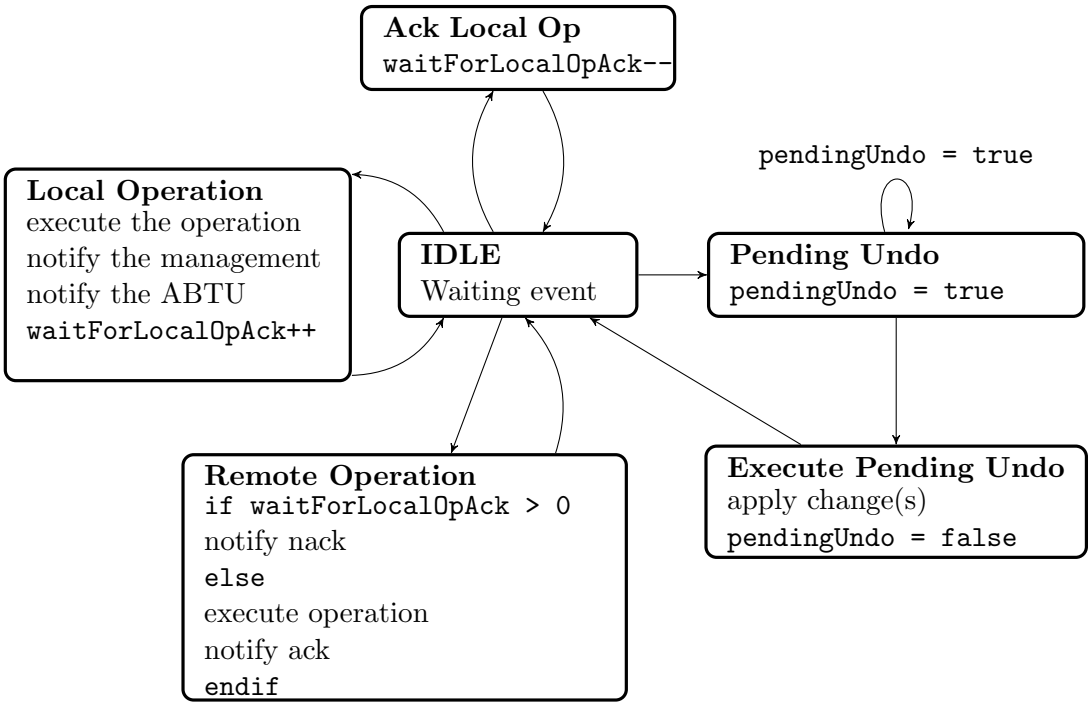
[1] It is an acronym which stands for "what-you-see-is-what-you-get"

Chapter 4

Protocols of communications

4.1 Protocol for the front-end

The javascript language cannot run two functions at the same time. This means no function will run until one is finished.[1]



[1] `https://groups.google.com/forum/#!topic/ace-discuss/cIYsOw8Vl5g`, visited on June the 8th, 2017

4.2 Protocol with the management

There is only one socket from the interface point of view. It receives/sends messages from/to the management through the port 5050. It uses json message encoded in bytes.

The structure of the messages are: {`Event: ..., Content: ...`}

The Event can be: "`ABTU`", "`AccessControl`" or "`Cursor`".

The Content itself is split into:

- Bytes {Type: "collaboratorOperation", Content: CollaboratorAction}
- Bytes {Type: "localOperation", Content: SimpleOperation}
- Bytes {Type: "cursorOperation", Content: Cursor}
- Bytes {Type: "ackRemoteOperation", Content: Boolean}
- Bytes {Type: "ackLocalOperation", Content: true}
- Bytes {Type: "remoteOperation", Content: SimpleOperation}

The `CollaboratorAction` has this form: `Bytes {Type: Action, DocId: Number, Email: byte[], PublicKey: byte[], IP: byte[], Rights: RightsEnum}`.

The actions are number from zero to four: share, receive, login, logout, update.

The rights are number with the following mapping: (RO: 4), (RW: 6), (AD: 7).

The `SimpleOperation` has this structure: `Bytes {OpType: OpType, Character: byte[], Position: number}`

The `OpType` has only two possible values: `INSERT: 1, DEL: 0`.

The `Cusor` has three numbers: `Bytes {uid: number, From: number, To: number}`

Chapter 5

Future work

In order to merge the interface to the ABTU process, changes have to be introduced.

5.1 Database

For the moment, the database can only be accessed through the browser. The ultimate project needs to communicate with other peers. Nor the interface, nor the ABTU can do it. That's why a management, who needs to distribute the data to every processes and to other peers, is placed in the center of the architecture.

The introduction of the manager was done in the last weeks. Therefore the database was not moved.

Replacing a NoSQL by an SQL will simplify the deletion part. It is worth to precise the interface will need adjustment, specially the collaborators storage and invitation part. The commands to create the table are written in the appendix.

5.2 Update the engine for the WYSIWYG mode

There are many solutions available to integrate a WYSIWYG editor in a web page. The minimum requirement is to have these functions and events:

```
void insertChar(c, i); // insert char c at index i
void removeChar(i); // remove char at index i
---
event changeCursorPosition
event changeContent
```

The current implementation uses bootstrap-wysiwyg. Most of the different projects that are based on it are not maintain anymore. Sometimes

their `README` file points to other project (which redirects to other project). It almost always ends up in 404 error.

It exists firepad which takes ace or codemirror as a back end. This is probably this one that will replace the bootstrap-wysiwyg.

5.3 Notifications

A top banner is placed at the top of the page. The click event works but the box that it opens is not used. This can be used for the notifications for example.

Appendix A

Software

The minimum requirement includes a modern browser and a database. Javascript needs to be activated which is generally the case by default. We assume that the browser is a modern one which can use sockets, feature introduced in HTML5.

A.1 Download

This project uses Couchdb, a NoSQL database which runs in most operating systems. The link to the installation of it and its dependencies is available here: `https://wiki.apache.org/couchdb/Installation`.

We will only describe in detail the steps for Fedora. As a root, enter this command: `[root@localhost]# dnf install couchdb`

There is one command to know which may take three different parameters: `service couchdb [start | status | stop]`. These do not need to be described... Starting is required each time the computer is shut down or the process is killed.

A.2 Verify the installation

Start Couchdb if not already running: `service couchdb start`. To check if everything is going right, enter: `curl http://127.0.0.1:5984/` in the terminal. The reply should look like:

```
{
    "couchdb": "Welcome",
    "uuid": "85fb71bf700c17267fef77535820e371",
    "version": "1.4.0",
    "vendor": {
        "version": "1.4.0",
        "name": "The Apache Software Foundation"
```

```
    }
}
```

The project's files contains a script which will create automatically all the 'tables'[1] if necessary.

A.3 Enabling CORS

According to the official documentation:[2]

> Cross-Origin Request Blocked: The Same Origin Policy disallows reading the remote resource at http://[couchDBIP]:[couchDBPort]/[dbname]/?_nonce=[request hash]. This can be fixed by moving the resource to the same domain or enabling CORS

We cannot move the html files (file:///home/user/my/path/my_project/) under the same domain as the database (localhost) because the files are static.

This constraint leads us to enable CORS. This can be done under this link: `http://localhost:5984/_utils/config.html`. It is needed to also allow all http methods.

A.4 Repository

The source code is available here:

- `https://github.com/Ito-Pakito/peer-to-peer-doc-editing`
- `https://github.com/DamienAy/epflDedisABTU`.

To make it run, first start the database. Second, execute: `go run main.go frontend=somewhere/peer-to-peer-doc-editing`.

A.5 Notes

The project started from scratch for the interface part and for the implementation of the ABTU algorithm. This lead the development of each part to be independent from the other (at the beginning). For an architecture point of view, a management had to be inserted and the database will be moved and changed from a NoSQL to SQL. The steps to follow for the ABTU part should be described in the ABTU algorithm implementation paper.

[1] This is an SQL concept whose equivalent does not exist in Couchdb.

[2] `https://pouchdb.com/errors.html`, visited on June the 7th, 2017

Appendix B

The SQL commands

```
CREATE TABLE DOCUMENT_METADATA(
    did INTEGER,
    parent INTEGER,
    is_folder BOOLEAN,
    name VARCHAR(63),
    permissions INTEGER,
    size INTEGER,
    last_update DATETIME,
    PRIMARY KEY(did),
    UNIQUE(did),
    FOREIGN KEY (parent) REFERENCES DOCUMENT_METADATA ON DELETE SET NULL
)

CREATE TABLE DOCUMENT_CONTENT(
    did INTEGER,
    data TEXT,
    history_buffer TEXT,
    PRIMARY KEY(did),
    UNIQUE(did),
    FOREIGN KEY (did) REFERENCES DOCUMENT_METADATA ON DELETE SET NULL
)

CREATE TABLE CONTACT(
    cid INTEGER,
    name VARCHAR(63),
    email VARCHAR(63),
    public_key VARCHAR(1024),
    PRIMARY KEY(cid),
    UNIQUE(email)
)
```

```
CREATE TABLE IP_ADDRESS(
    identifier VARCHAR(128),
    cid INTEGER,
    PRIMARY KEY(identifier, cid),
    FOREIGN KEY (cid) REFERENCES CONTACT ON DELETE CASCADE
)

CREATE TABLE PUBLIC_KEY(
    key VARCHAR(1024),
    cid INTEGER,
    PRIMARY KEY(key, cid),
    FOREIGN KEY (cid) REFERENCES CONTACT ON DELETE CASCADE
)

CREATE TABLE MAPPING_DOCUMENT_CONTACT(
    did INTEGER,
    cid INTEGER,
    PRIMARY KEY(did, cid),
    FOREIGN KEY (did) REFERENCES DOCUMENT_METADATA ON DELETE CASCADE,
    FOREIGN KEY (cid) REFERENCES CONTACT ON DELETE CASCADE
)
```