Systemtechnik Theorie

5DHIT 2024/25

Protokoll Zlabinger 5DHIT

Theorieprotokoll

Christof Zlabinger 5DHIT

14. Januar 2025

Bewertung:		Version:	1.0
Betreuer:	Michael Borko	Begonnen:	05.09.2024
		Beendet:	XX.XX.XXXX

Inhaltsverzeichnis

1 Distributed Computing

1.1 ACID and BASE

1.1.1 ACID

Transactions are used to ensure data is consitent. These transactions consist of multiple commands that are either all executed or are not executed at all.

Atomicity
Atomicity means that either the entire transaction is executed or it is not executed at all. This prevents some statements to be executed while others aren't, which could lead to errors.

Consistency
Consistency means that before and after a transaction is executed all the rules or contraints are met.

Isolation
This means that transactions should be seperated from another but still be executed at the same time and should give the same result if they were run one after the other. There are different isolation levels. Each level isolates some part of or the entire database from other transactions with the drawback that transactions can't be concurrent.

Durability
This means that the data is stored persistently.

1.1.2 BASE

BASE is an alternative to ACID.

Basically-Available
Transactions are atomically safed in a queue and are executed sequentially.

Soft-State
After a specific time that a transaction has been in the queue it will be timeouted.

Eventually Consistent
After all transactions have been executed the database is consistent.

1.2 CAP

It is only possible to achieve two out of the three following goals.

1.2.1 Consistency

All locations have the same data.

1.2.2 Availability

The system is available.

1.2.3 Partition Tolerance

This describes the system to be able to function even if some parts of it are cut off network wise.

1.3 DiffSync

Differential Syncronisation is an algorithm designed to syncronize documents across multiple devices. There must be an diff and patch algorithm for the type of data that should synced.

1.3.1 Introduction

DiffSync enables collaboration in real time. The syncronization algorithm in an application is hard to change. DiffSync can be used in existing applications.

Suited for:

- unreliable/slow networks
- identical code on server/client
- No history required
- Scalable

Use cases:

- Pair programming on different systems
- Remote debugging

No problem when collaborating with someone or yourself with autosave feature because all versions are kept in sync.

1.4 Alternatives

1.4.1 Pessemistic

Only edit by one user at a time Whole file is locked for other users Refined version would be to only lock subsections but created problems for small files

- Has to be supported by the application

Not good for unreliable networks

1.4.2 Edit-Based

Capture *all* edits (Typing, cut, paste, formatting, ...) and mirror them across all devices If an edit is lost it will create a fork -> two different files

1.4.3 Three way merge

Client sends changes to server -> server performs 3 way merge -> merge gets send to the clients If someone changes something while someone else is changing something it will result in a conflict Not scalable

1.5 Dual shadow method

Client text and server shadow must be identical every half syncronization or server text and client shadow

Checksums after patch must be identical otherwise entire body must be sent again

1.6 Topology

Multiple clients connected to a single server. The number of clients can become an issue -> load balanced servers with shared DB Or server to server communication (two or more servers are connected with eachother). Servers can be added without a problem but can only be disconnected if no client is connected. Latency may become an issue -> Can lead to complicated collisions. Can be reduced by avoiding long server chains and by increasing the sync time between servers

1.7 Diff/Patch

Diff/Patch algorithms are crucial. Efficiency improvements in these algorithms boost system performance. Can be used for plaintext and many more. Diff updates server shadows with client changes. It needs to differentiate minimal edits from semantic intent ensuring meaningful updates. Compares identical texts efficiently. Identifies shared starting/ending strings to speed up comparisons. Detects if a change is an addition or removal bypassing complex calculations.

Patches apply accurately, even if texts aren't exact matches, using the Bitap algorithm for efficient near-match location. Merging non-text content may use a "last user winsäpproach to prevent incorrect combinations. Handling Collisions: Systems can address patch errors either by frequent, silent synchronization or user intervention on failed patches, balancing usability with data accuracy.

1.8 Adaptive timing

Client update frequency is critical for system responsiveness. Infrequent updates lead to costly operations and edit conflicts, while overly frequent updates strain the network. The Guaranteed Delivery Method reduces resource use by batching diffs for transmission. An adaptive system adjusts update frequency based on user activity, keeping updates efficient and responsive by shortening intervals during high activity and lengthening them when activity is low.

1.9 Future work

The fuzzy patch operation uses a three-way merge comparing Server Text and two versions of the Server Shadow. It only supports single synchronization packets limiting performance in high-latency connections. Additionally edits from all users are blended making attribution and individual rollbacks difficult.

1.10 JavaScript implementation

1.10.1 Which implementation of diff-patch algorithms are used? How is it copied?

It uses json-fast-patch.

1.10.2 Where are the documents and shadows?

The shadow and the backup are saved in ‘container.shadow‘‘ or ‘container.backup‘ and the document in mainText. It can be copied by using ‘jsonpatch.deepClone$mainText$‘.

1.10.3 How and why can we ajust the sync-cycle? What are the advantage/disadvantages?

We can adjust them to either have lower network traffic if we lower the cycles or faster syncronization if we increase the cycles.

1.10.4 Where/How is the edit stack implemented?

It is implemented as an array that is saved in the container as ‘container.edits‘.

1.10.5 It is possible to deploy a peer to peer version?

Yes since the code of the client and server are nearly identical.

1.10.6 How is it possible to use the API in other JS projects?

1.10.7 Are the JSON documents interchangable with other kinds of documents?

It is possible but it would require the change from the json-fast-patch package to another one.

1.10.8 How are conflicts handled?

It just gets discarded.

1.11 Firebase

1.11.1 Cloning the Repository

```
git clone https://github.com/firebase/friendlychat
```

1.11.2 Creating flake.nix for NixOS Specific Behavior

```
# flake.nix
{
  inputs = {
    nixpkgs.url = "github:NixOS/nixpkgs/nixpkgs-unstable";
    systems.url = "github:nix-systems/default";
  };
  outputs =
    { systems, nixpkgs, ... }@inputs:
    let
    eachSystem = f: nixpkgs.lib.genAttrs (import systems) (system: f nixpkgs.legacyPackages.${sy
    in
    {
      devShells = eachSystem (pkgs: {
        default = pkgs.mkShell {
          buildInputs = [
            pkgs.nodejs
            pkgs.nodePackages.firebase-tools
          ];
        };
      });
    };
}
```

This is required to use dynamically linked libraries on NixOS systems.
Execute in the directory of the flake:

```
nix develop
```

1.11.3 Start

1.11.4 Login to Account

```
firebase login
```

1.11.5 Authorize Firebase CLI

```
firebase use --add
```

1.11.6 Deploy App

```
firebase deploy --except functions
```

1.12 Install Dependencies

Run:

```
npm i
```

to install all dependencies.

1.13 Setup in the Web Interface

In the Firebase web interface, the storage had to be created and the project set up.

1.14 Implement Functions

1.14.1 Enable Functions in Cloud Console

```
// Import the Firebase SDK for Google Cloud Functions.
const functions = require('firebase-functions');
// Import and initialize the Firebase Admin SDK.
const admin = require('firebase-admin');
admin.initializeApp();

// TODO(DEVELOPER): Write the blurImages Function here.

// TODO(DEVELOPER): Write the sendNotification Function here.
```

1.14.2 Welcome Message

```
// Adds a message that welcomes new users into the chat.
exports.addWelcomeMessages = functions.auth.user().onCreate(async (user) => {
  functions.logger.log('A new user signed in for the first time.');
  const fullName = user.displayName || 'Anonymous';

  // Saves the new welcome message into the database
  // which then displays it in the FriendlyChat clients.
  await admin.firestore().collection('messages').add({
    name: 'Firebase Bot',
    profilePicUrl: '/images/firebase-logo.png', // Firebase logo
    text: `${fullName} signed in for the first time! Welcome!`,
    timestamp: admin.firestore.FieldValue.serverTimestamp(),
  });
  functions.logger.log('Welcome message written to database.');
});
```

Deploy the newly created functions:

```
firebase deploy --only functions
```

1.14.3 Exception During Deployment

This command throws an exception: `No targets in firebase.json match '-only functions'`

Attempted solutions:

- `firebase deploy functions`: No error message, only a note about `firebase [command] -help`.
- `firebase deploy`: Does not throw an error, but the function does not appear in the web interface and is not executed.
- `firebase deploy functions -add`: `unknown option -add`.

2 Embedded Systems

2.1 Pico Development Environment for NixOS

Repository: https://github.com/czlabinger/NixPicoEnv/

Open in VSCode, install and start the DevPods extension. SSH into the created container. The template is located in `src`, and everything outside of `src` can be ignored. Modify `CMakeLists.txt` to import the desired projects.

Building:
'sh src/build.sh'

2.2 Setup pico SDK

'git clone https://github.com/raspberrypi/pico-sdk.git?submodules=1'
'cd pico-sdk'
'echo 'export PICO_SDK_PATH=/pico-sdk' » /.bashrc'

2.3 Blink

The blink delay in the 'blink.h' file had to be set to 25 to achieve the targeted frequency.

While pressing the BOOTSEL button on the pi while plugging it in it can be mounted via:

'sudo dmesg | tail' this should show a new drive sdX. Now mount this drive using 'sudo mount /dev/sdX /mnt'. Now the generated uf2 file can be copied onto the pi. Lastly execute 'sudo sync' for the pi to acknowlege the changes.

2.4 CAN

2.4.1 Operation-Modes MCP2515

- **Normal Mode:** Standard mode in which the controller can send and receive messages.
- **Sleep Mode:** Power-saving mode in which the controller consumes minimal current and does not process messages. An external wake-up can reactivate the controller.
- **Loopback Mode:** Internal test mode where sent messages are directly received without leaving the CAN bus. Ideal for tests without an external CAN network.
- **Listen-Only Mode (Default):** Receive-only mode where the controller reads received messages without sending ACK bits. Often used for bus diagnostics.
- **Configuration Mode:** Configuration mode where settings such as bitrate and filters of the controller can be configured.

2.4.2 Pins

MCP2515	Pico
VCC1(5V)	VBUS
3,3V(OUT)	VCC(3,3V)
PIN_MISO	GP4
PIN_CS_A	GP5
PIN_SCK	GP6
PIN_MOSI	GP7

2.4.3 Code

After connecting the pins as described above the code from https://github.com/a-kipp/RP2040-CAN-Driver and changing the pins and setting the mode to normal mode it is now possible to connect two picos over a logic analyzer where one is connected to CANH and one to CANL.

2.5 Hello USB

Using ‘minicom -D /dev/ttyACM0 -b 115200‘ it is possible to read the output from printf calls on the another device that is connected via usb.