NexRes

Exploratory Systems Lab | Dec 06, 2023

ResView- Visualizing Resilient DB

What is ResView?ResView is a tool for visualizing the PBFT process in Resilient DB. ResView is built on top of Resilient DB and gathers statistics from transactions to display in graphical form. C...

What is ResView?

ResView is a tool for visualizing the PBFT process in Resilient DB. ResView is built on top of Resilient DB and gathers statistics from transactions to display in graphical form. Currently, ResView works on local instances of Resilient DB, but can be used for deployed instances with some changes to Websockets and connections. The target userbase of ResView is people learning about ResilientDB who want more understanding of how the PBFT consensus protocol works and the general flow of blockchain consensus, as well as developers who want to observe the status of ResilientDB and the rate at which various ResilientDB processes are occurring, such as message collection.

Below is a diagram showing the workflow of ResView and the high-level structure:

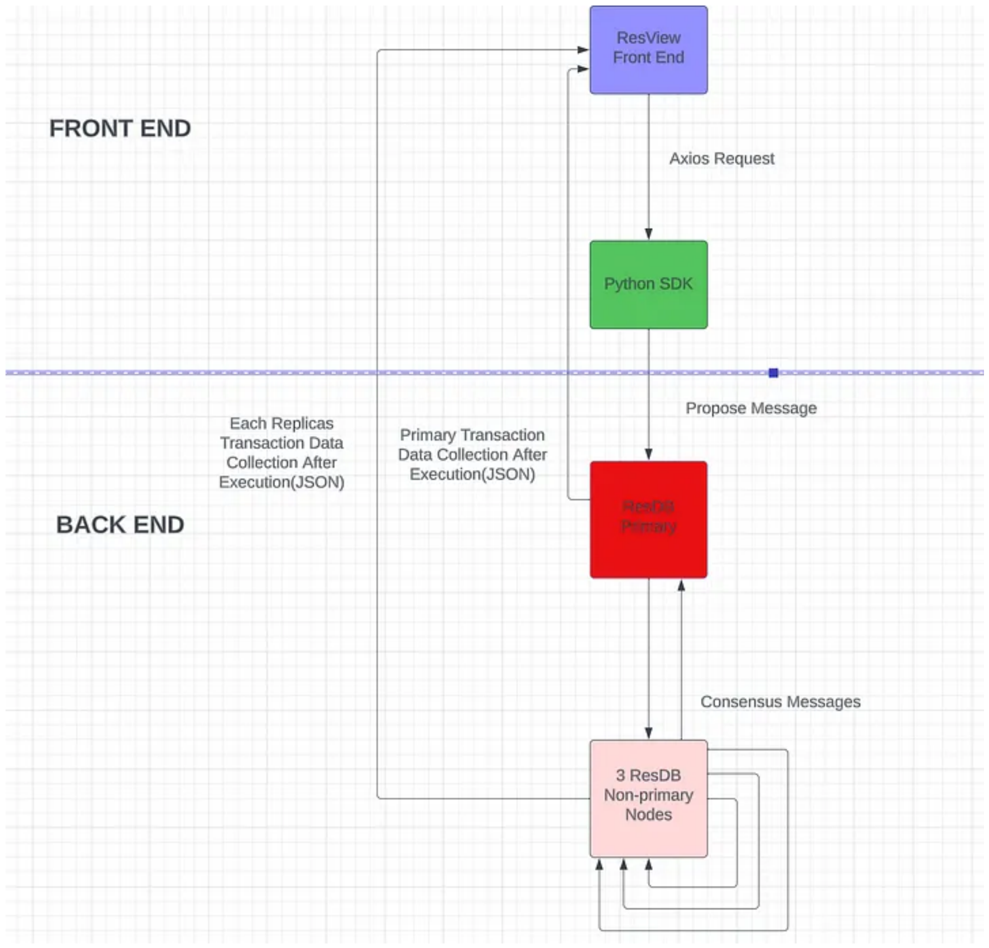
ResView Front End
FRONT END
Axios Request
Python SDK
Propose Message
Each Replicas Transaction Data Collection After Execution(JSON)
Primary Transaction Data Collection After Execution(JSON)
BACK END
ResDB Primary
Consensus Messages
3 ResDB Non-primary Nodes

Figure 1. Diagram displaying the structure of ResView and how data is passed between the different services.

Interfacing with ResilientDB

ResView offers both ways to interact with the ResilientDB architecture, as well as ways to visualize what is occurring in ResilientDB. Users are able to interact with ResilientDB through the application using the transaction forms and “make a replica faulty” buttons. Within the transaction forms, users select whether they want to perform a SET or GET transaction and input the data fields accordingly, key & value or just key. Once this transaction is confirmed, it is sent via Axios to the sdk, which verifies the transaction and forwards it to the ReslientDB backend, where it is then processed and executed.

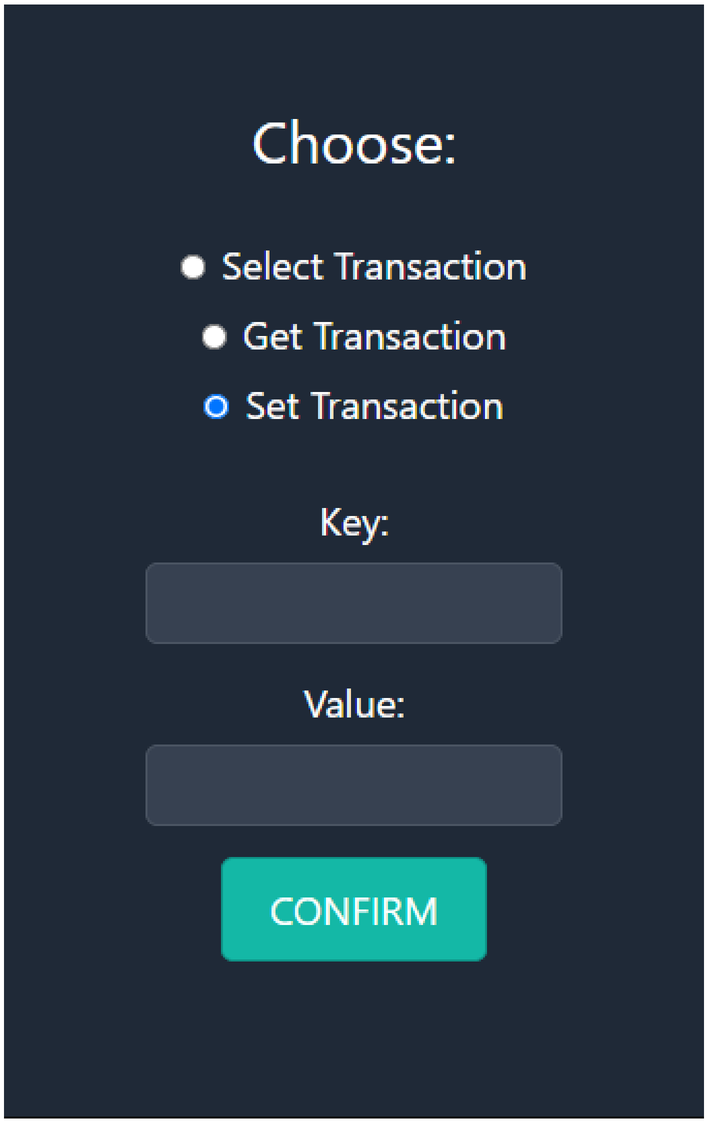

Figure 2. Image of the Transaction Form.

For the faulty buttons, when these are pressed, they send a signal to the corresponding the replica informing it to reject any transactions it receives, simulating a replica unable to receive messages. Before this addition, there was no method of forcing a replica to be faulty, making this a valuable addition for testing various cases and potential situations ResilientDB could face.

Data Collection

ResView collects the data for visualization directly from the ResilientDB application, as while transactions are being ran, a statistics data structure collects the contents of the transaction, timestamps of when each state was reached, and timestamps of when messages were received. These statistics are then grouped as a JSON, which is then sent out to the front end application. The reason the data is collected this way is to enable ResView to produce visualizations even without access to log files, as well as gather information regardless of who initiated the transaction.

Viewing the Data

On the application, users are able to select which transaction's data they want to see diagrams of, as all data is stored as a history while the application is running. Once a transaction is selected, there are 3 viewable graphs: PBFT diagram, prepare messages vs time, and commit messages vs time.

The PBFT diagram is dynamically constructed using the data collected from the replicas. The main data points utilized are the primary id, which signifies in the diagram where the client should send its request to, the timestamps of each state, as they identify that the replica reached that state and how long it took, and the existence of each replica's data, as faulty replicas do not send their data to the front end. This allows the PBFT diagram to work with real time data and accurately convey the state of ResilientDB to the user, as well as the flow of the consensus protocol.

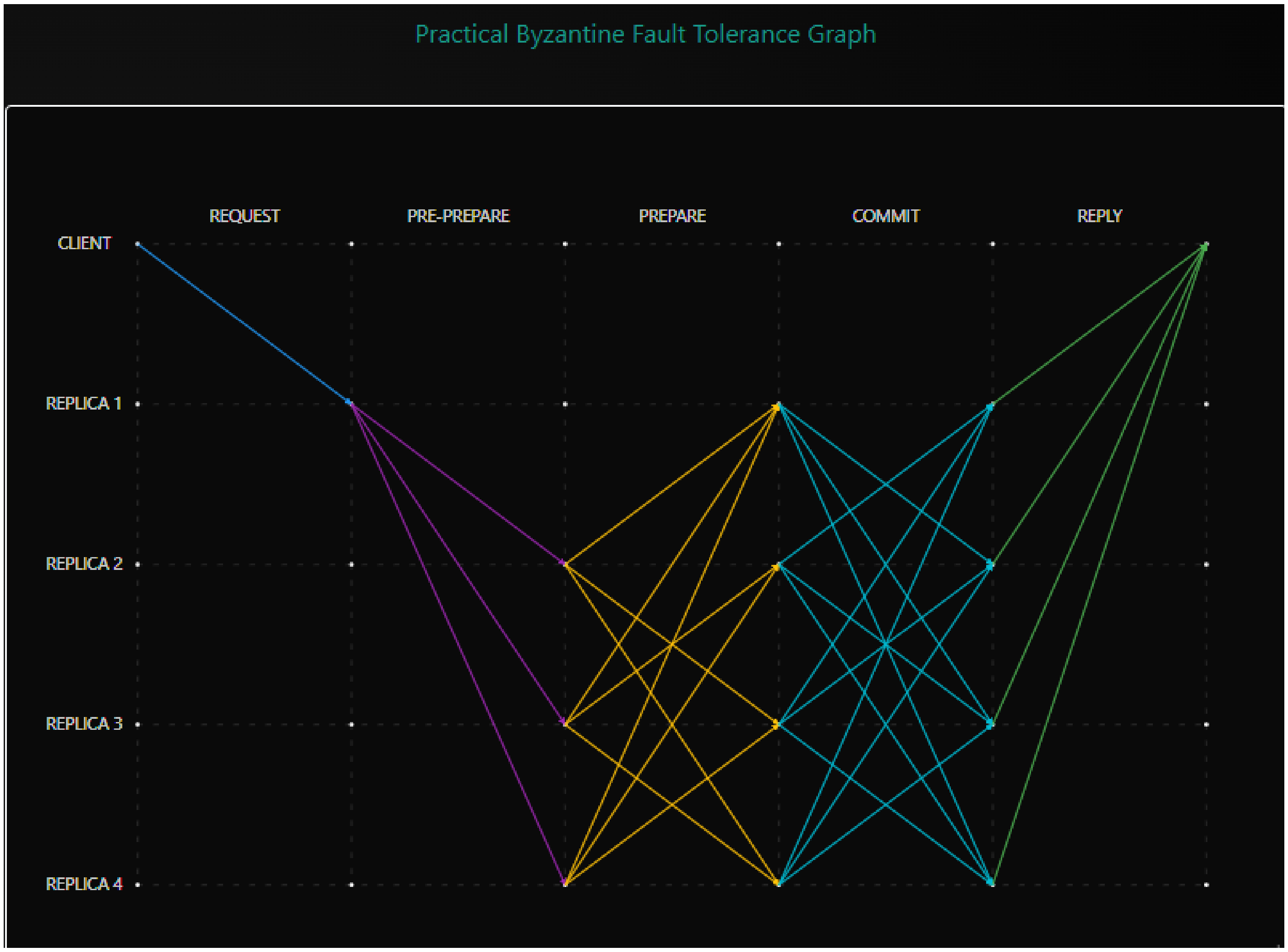

Figure 3. PBFT Diagram.

Both the prepare message vs time and commit message vs time graphs use a line graph containing a line of each replica's message collection. The timestamps start from when a replica is first able to start sending messages, which the message collection times are relative to. The purpose of displaying lines from each replica is to easily compare the rates at which replicas collected messages to understand the relationship between them, as well as identify any potential issues with message collection. In order to focus on specific replicas, the user is also able to toggle the lines of any replica which reduces the amount of data of the graph being displayed.

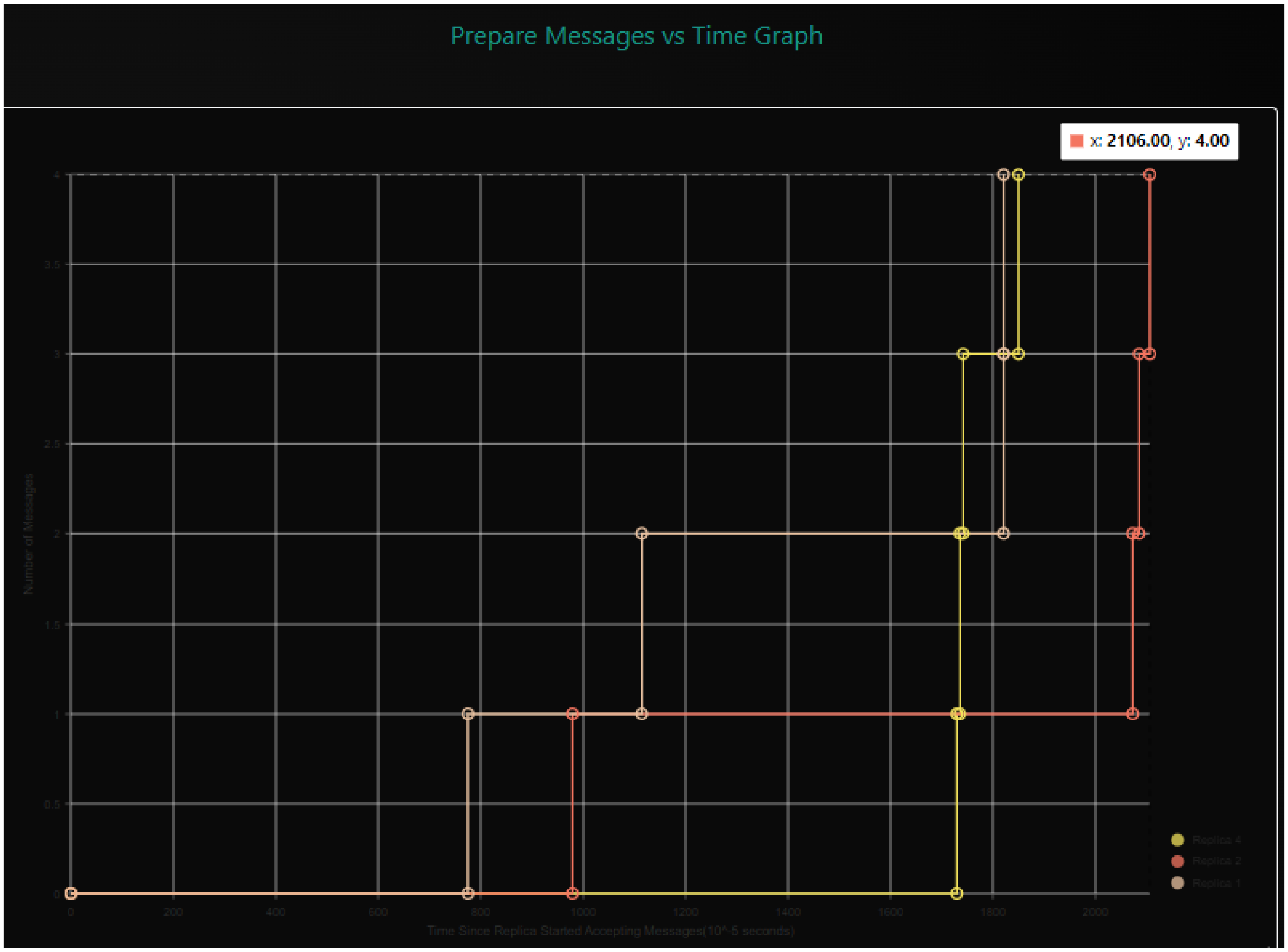

Figure 4. Prepare Message Graph.

Demo Video

Running the Application

Prerequisites

Before running the ResView application, you need to start kv service on the ResDB backend and the sdk.

resilientdb

Git clone the ResView backend repository and follow the instructions to set it up:

```
git clone https://github.com/Saipranav-Kotamreddy/ResView
```

Setup KV Service:

```
./service/tools/kv/server_tools/start_kv_service.sh
```

sdk

Git clone the GraphQL Repository and follow the instructions on the ReadMe to set it up:

Install GraphQL:

```
git clone https://github.com/ResilientApp/ResilientDB-GraphQL
```

Setup SDK:

```
bazel build service/http_server:crow_service_main

bazel-bin/service/http_server/crow_service_main service/tools/config/interface/service.config service/http_server/server_config
```

With these 2 services running, the ResView front end can now send transactions to the ResDB framework

Running the ResView Application

Clone the repo and open in a new folder.

```
npm install
```

Run the below code to start the app and load the script.

```
npm start
```

Using the ResView Application

Once ResView has been started, go to http://localhost:3000/pages/visualizer

Load KV_Service on resilient db once on the visualizer page, the web sockets should connect and log the "OPEN" message into the console 4 times.

Once on the visualizer page, there are 3 tabs: Transaction Form; PBFT Diagram; and Messages vs Time. Use the transaction form to send transactions to the sdk, whether SET or GET transactions. Once the transaction's data has been sent back, click on the various graphs to view the transaction data in a more comprehensive way. In order to select previous transactions, go to the transaction form's choose transaction section and type the transaction number you wish to view.

Future Work

Fix a bug with View Change which is inconsistent on the backend, as setting multiple replicas to be faulty simultaneously causes resilientdb to start producing errors.

Add a diagram to see view changes and how they function

Enable users to see transaction's command, key, and value before selecting which one to view

Allow users to dynamically scale resilientdb to different replica amounts to simulate larger networks

Set up ResView on Cloud Instance to retrieve far more transaction data

Store transaction information between application instances to avoid data loss

Source Code Repositories:

https://github.com/aunshx/dds_265_prj
https://github.com/Saipranav-Kotamreddy/ResView
https://github.com/ResilientApp/ResilientDB-GraphQL

Contributions:

Saipranav: Designed project architecture and workflow, coded the backend data collection, the faulty replica toggles, configured all websockets, wrote JSON data parsing code, created Message vs Time graphs and line toggles, and setup transaction sending and sdk incorporation.
Aunsh: Designed the front end UI/UX and architecture, skeleton, contexts, data-stores and pages, developed main PBFT diagram, setup graph component skeletons and general styling, look and functionality of the entire application.
Nikita/Madhumitha: Designed UI elements for forms and buttons, implemented styling for the homepage contents, and wrote the code for the homepage.
Harshini: Wrote contents for the home page, integrated them to the front end and worked on the code and styling.