National College of Ireland

Ensemble Fault Tolerance Technique Configuration Manual

MSc Research Project
Cloud Computing

Erel Ozturk
Student ID: 21245312

School of Computing
National College of Ireland

Supervisor: Vikas Sahni

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Erel Ozturk
Student ID:	21245312
Programme:	Cloud Computing
Year:	2023
Module:	MSc Research Project
Supervisor:	Vikas Sahni
Submission Due Date:	14/08/2023
Project Title:	Ensemble Fault Tolerance Technique Configuration Manual
Word Count:	1644
Page Count:	6

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Erel Ozturk
Date:	11th August 2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Ensemble Fault Tolerance Technique Configuration Manual

Erel Ozturk
21245312

1 Prequisites

This document provides the reader with the necessary setups to conduct the experiments described in the thesis paper. The purpose of these experiments is to determine the best one out of three techniques, checkpointing, replication, and a combination of checkpointing and replication together. The manual will first walk the readers through the necessary software setups and specifications of the instances that the experiments will be conducted on as well as the necessary configurations on these instances. After that, it will be discussed how to download and run the source code of the experiments. The paper will finally provide how to run the experiments and capture the metrics.

To set up the client-side of the application, Android Studio is required. It will be necessary to make use of two components of it, Android Virtual Device (AVD) and Profiler. After installing Android Studio, using AVD, a new virtual device should be created with the specifications given in Table 1.

Table 1: Virtual mobile device specification

Metric	**Value**
OS	Android 11.0
CPU	ARM64 (4 cores)
RAM	1536 MB
Storage	800 MB
VM Heap	256 MB

To get the client application up and running, the Flutter framework with Dart programming language is required. Please follow the installation instructions in Flutter's official website[1] to install **Flutter version 3.10.5**.

To set up the server-side of the application, two identical *t2-micro* EC2 instances are required. The specifications of the servers are given in Table 2.

2 Environment

The client application is run on the local machine. Specifications of the author's local machine are given in Fig 1.

[1] https://flutter.dev/

Table 2: EC2 instance specification

Region	OS	CPU	RAM	Storage
eu-west-1	Ubuntu 22.04.2 LTS	Intel(R) Xeon(R) CPU E5-2676 v3 @ 2.40GHz (single core)	957 MB	7.6 GB

Figure 1: Specifications of the local machine

Server-side of the application is run on AWS EC2 instances as mentioned in the previous section.

3 Installation

This section describes how to download the source code for the client and the servers. To download the source code and ensure libraries are correctly installed, please follow the instructions described below:

```
$ git clone https://github.com/erel98/NCI_thesisProject_client.git
$ cd NCI_thesisProject_client
$ flutter pub get
```

Note: The `$ flutter pub get` command will install the specified versions of the libraries. A list of the versions of libraries that are used in this application can be found in *pubspec.yaml* file.

For both of the servers, Python 3.7.5 and pip installations are required. The installation process is straightforward, therefore not described. After `ssh`'ing to the main server installing Python 3.7.5 and pip, follow the instructions below to complete the necessary installations.

```
$ sudo apt-get rsync
$ git clone https://github.com/erel98/NCI_thesisProject_mainServer.git
$ cd NCI_thesisProject_mainServer
$ chmod +x dataReplication.sh
$ python3 -m venv env
$ source env/bin/activate
$ pip install pip --upgrade
$ pip install -r requirements.txt
```

To enable service replication in background:

```
$ chmod +x serviceReplication.sh
$ nohup ./serviceReplication.sh &
```

Do the same for the backup server with the following instructions:

```
$ git clone https://github.com/erel98/NCI_thesisProject_backupServer.git
$ cd NCI_thesisProject_mainServer
$ python3 -m venv env
$ source env/bin/activate
$ pip install pip --upgrade
$ pip install -r requirements.txt
```

4 Configuration

Do the same for the backup server with the following instructions: This section instructs readers on how to configure their EC2 instances to accept custom TCP connections on 8000 port. Please follow the step-by-step instructions below:

1. Login to your AWS account and go to the EC2 dashboard.
2. Make sure you are in Dublin (eu-west-1) region and locate your main server instance. Click on the instance name.
3. Copy the public IP address of the instance and save it somewhere for future usage.
4. From the tabs below, switch to the Security tab.
5. Click on the security group name.
6. In the Inbound rules table, find and click the "Edit inbound rules" button.
7. Add a new rule with the details given in Table 3 and save it.
8. Open the directory where you downloaded the Flutter app from your local machine.

9. Open *consts.dart* and paste the copied IP address from step 3 to *mainServer* constant.

10. Repeat all the steps for the backup server as well and paste the public IP address to the *backupServer* constant in *consts.dart*.

Table 3: Details of new inbound rule

Type	**Protocol**	**Port range**	**Source**	**IP**
Custom TCP	TCP	8000	Custom	0.0.0.0/0

Please note that the public IP addresses of the EC2 instances are subject to change if the instance is stopped or restarted. In case they change, make sure to update the *consts.dart* with the latest IP addresses in order for the client to be able to connect to the servers.

5 Running the experiments

This section contains information about how to execute experiments and monitor the metrics of the client. As a first step, it should be decided which scenario is wished to be executed. Based on the choice, in the main server, the *helper.py* file must be edited to let the system know what is being simulated. After `ssh`'ing to the main server, do the following to edit the file.

```
$ cd fastapi
$ nano helper.py
edit the scenario variable
```

- **For checkpointing:** Set the *scenario* variables to *checkpoint*.
- **For replication:** Set the *scenario* variables to *replication*.
- **For combined:** Set the *scenario* variables to *combined*.

Similarly, in the Flutter app, navigate to *consts.dart* file and locate the *isCombined* constant. Set it to *true* only if the ensemble scenario is being simulated.

After letting the main server know the scenario, the servers should be started with the following command:

```
$ uvicorn main:app --host 0.0.0.0 --port 8000 --reload
```

It's worth noting that the main server will raise an intentional exception at 5th iteration while executing replication and combined scenarios. If desired, it can be adjusted from the source code of *main.py*, line: 58.

Now that the servers are up and running, the client application should be started. Before starting the emulator, make sure to locate the Profiler button in Android Studio as shown in Fig 2. From Android Studio, start the AVD that was configured in the previous sections. Now, it should be possible to see the emulator when clicked on the plus button in Profiler tab. From Android Studio, run *main.dart* to start the application. Once the

application is up and running in the emulator, in Profiler tab, it should be possible to see the application with a name like *com.example.faulttolerance* under the emulator when clicked on the plus button. Locate and click it to monitor the metrics of the emulator during the experiments. It can be clicked on the desired metric to have a closer look (i.e. network).

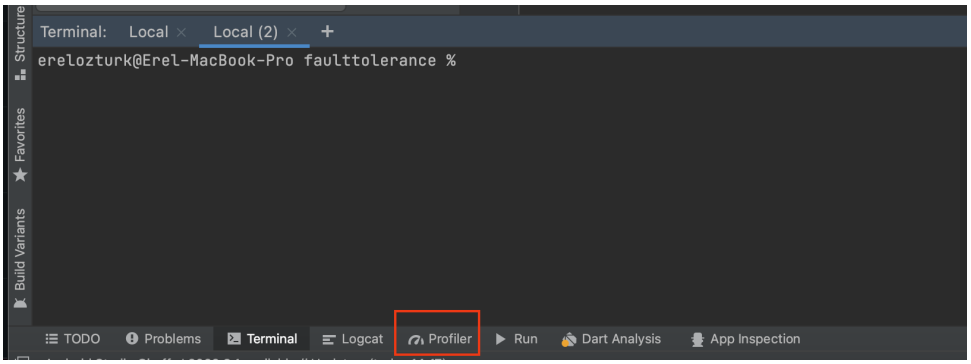

Figure 2: Profiler button in Android Studio

By default, when the application is started, the client will establish a connection with the main server initially. There are two buttons at the screen:

- **Start simulation**: Emits the message to the server and initiates execution.
- **Connect**: Connects to the server manually if the connection is closed (when execution is completed, the connection between the server and the client is automatically closed).

To proceed with the rest of the experiment, please see the relevant section from below according to the choice in *helper.py* file in the main server.

5.1 Checkpointing

This scenario involves a network disconnectivity to see how the system recovers from the disconnectivity state. To be able to simulate this scenario, the *scenario* variable should have been set to *checkpoint* in *helper.py*. After starting the application, click the *Start simulation* button on the screen. In the console, some outputs will be seen that count from 0. By default, the client application is configured to send 15 to the server, which means there will be 15 iterations in total. If desired, this value can be changed by editing the *messageToBeEmitted* variable in *MainScreen.dart*.

When the emulator is wanted to be disconnected from the internet to simulate network disconnectivity, swipe down from the top of the screen to open the notification panel. Scroll again to expand the menu. Swipe left to switch to the second screen from the menu and locate Airplane mode. When the Airplane button is clicked, the emulator will lose network connection and the execution will pause. When desired, it can be clicked again on the same button to restore the network connection of the emulator. It will be seen that the execution will continue from the last iteration after the emulator restores its network connection. It's possible to follow the process from the console in Android Studio. When the execution is completed, the total execution time will be prompted in the console.

While the application is running, metrics of the emulator (such as network data usage, memory utilization, etc.) can be tracked from the Profiler tab. To be able to see both console outputs and the Profiler activity, it is recommended to change the view mode of the Profiler tab (i.e., windowed) from the setting button at the top right-hand side of the Profiler tab.

5.2 Replication

This scenario involves an intentional server-side failure at 5th iteration of the execution. To be able to simulate this scenario, *scenario* variable should have been set to *replication* in *helper.py*. Unlike checkpointing scenario, this scenario does not involve any user interaction other than clicking on the "Start simulation" button. Once the simulation is started, in the console, some outputs counting from 0 will be seen again. Once it reaches the 5th iteration, the main server will raise an exception and connection to the main server will be closed. The client will then automatically connect to the backup server and emit the same message. Thus, the execution will be initiated from the start.

5.3 Combined

This scenario involves an intentional server-side failure at 5th iteration of the execution similar to the replication scenario. To be able to simulate this scenario, the *scenario* variable should have been set to *combined* in *helper.py*. The procedure is exactly the same until the execution reaches to 5th iteration. After the main server raises an intentional exception on the 5th iteration, the main server will synchronize the *checkpoint.json* file with the backup server using the *dataReplication.sh* script. After that, the client will close the connection with the main server and connect to the backup server. However, instead of starting from the start, the execution will continue from the 5th iteration. Therefore, the latest snapshot of the system is synchronized to the backup server so that when the client connects to the backup server, the execution can continue from where it was left at.