SYSTEM DESIGN DOCUMENT

for

Mini Trading Application

COMPSCI 677 Lab 1

Prepared by: Mudit Chaudhary (32607978)
Girish Baviskar (33976648)

Contents

1 Introduction **3**

1.1 Purpose . . . 3

1.2 Intended Audience and Reading Suggestions . . . 3

1.3 Tech Stack . . . 3

2 System Design **4**

2.1 Part 1 . . . 4

2.1.1 RequestRunner . . . 4

2.1.2 SinglePollingThread . . . 4

2.1.3 ProducerConsumerQueue . . . 4

2.1.4 ThreadPool . . . 4

2.1.5 Client . . . 5

2.2 Part 2 . . . 5

2.2.1 Protobuf . . . 5

2.2.2 StockDatabase . . . 5

2.2.3 TradingServer . . . 5

2.2.4 Clients . . . 6

1 Introduction

1.1 Purpose

This document specifies the system design for Part 1 (Socket-based with thread pooling) and Part 2 (gRPC application). This project corresponds to the CS677 Lab 1.

1.2 Intended Audience and Reading Suggestions

This document is intended for developers, testers, documentation writers, and the grading staff for the project. This document does not specify the technical details, evaluation results, and user manual.

1.3 Tech Stack

We use the following tech stack for EleNa:

- Java
- Maven
- gRPC

2 System Design

2.1 Part 1

Fig. 2.1 shows the system flow diagram of our thread pool implementation. We implement a queue to accept incoming requests, which are then handled by one of the idle threads in the thread pool. We initially looked some implementations[1] of thread pool online and then made our own design choices to develop the threadpool application from scratch.

In the subsequent sections, we explain the components of our system.

2.1.1 RequestRunner

We provide RequestRunner which implements the Runnable interface. It runs the received request (lookup) on an idle thread. Each RequestRunner handles a single request. We queue the RequestRunner objects in our queue, which are then picked up by idle threads in the thread pool.

2.1.2 SinglePollingThread

It represents a single thread in the thread pool. If a RequestRunner object is available in the queue, it consumes it and runs it.

2.1.3 ProducerConsumerQueue

It is our implementation of a thread-safe request queue, which stores Runnable objects (RequestRunners). We used synchronization to make it thread-safe during adding and consumption of RequestRunners. We provide a *take* method, which the consumers (idle threads) can use to consume a task in the queue. If no item is available, it puts the thread on *wait* until it is notified of an item in the queue. When any item is added in the queue, we notify an arbitrary waiting thread to wake up and consume the item. We notify an arbitrary thread instead of notifying all threads to avoid contention.

2.1.4 ThreadPool

This is our implementation of threadpool, which consists of multiple SinglePollingThreads and a ProducerConsumer queue. It provides methods to add requests to the queue and initializing idle threads.

[1] https://blog.caffinc.com/2016/03/simple-threadpool/

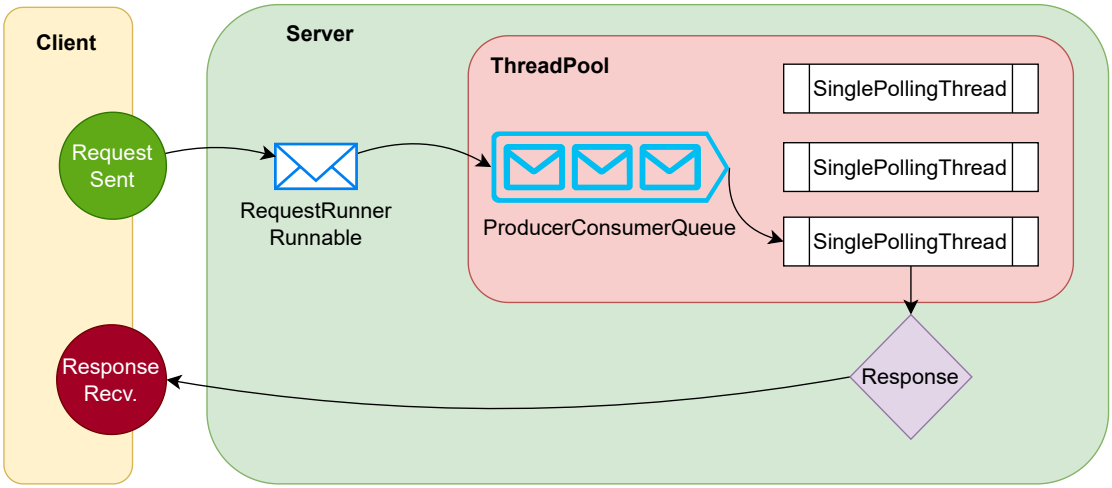

Figure 2.1: Part 1 Threadpool System Design

2.1.5 Client

We provide a simple client that provides the functionality to automatically send random valid requests to the server. We use this to perform load testing.

2.2 Part 2

Part 2 uses gRPC to communicate between the clients and the server.

We explain the components of our system in the sections below:

2.2.1 Protobuf

We create a protobuf to define the request messages and other datatypes. To make our request and response codes easier to understand, we create two enums – REQUEST_STATUS and TRADE_TYPE. We define request, response messages, and services for lookup, trade, update.

2.2.2 StockDatabase

This is an in-memory stock database to store stock details. We use ConcurrentHashMap to achieve thread-safety. It also uses separate locks for read and write, which makes it more efficient to use with multiple requests.

2.2.3 TradingServer

The TradingServer is responsible for creating a gRPC server and adds the trading service to the server. For the threadpool, we create a dynamic cached thread pool with maximum 10 threads. We have a timeout of 100 seconds for each thread. We use a

SynchronousQueue, which proactively hands out all the requests to an idle threads or creates a new thread if no idle thread is available.

2.2.4 Clients

We provide two types of clients – 1) Customer Client, 2) Updating Client. Customer client can perform lookup and trade operations. We provide the functionality of automatically sending random valid requests to the gRPC server and use it for load testing. Updating Client performs update requests at random intervals.