Lab 1 - Part 2 - Design Document

Overview and Objectives:

The design document outlines the creation of an online Toy Store using gRPC, featuring a dynamic server thread pool for efficient request handling. The server will implement two primary functions: Query, to return item costs and stock levels, and Query, to manage stock reductions upon purchase. The catalog includes four toys, catering to various enthusiasts. The system utilizes protobuf for message structuring, ensuring robust communication. Emphasizing concurrency, it allows multiple clients to perform queries and purchases simultaneously, showcasing the thread pool's capability to handle parallel requests seamlessly.

Design:

System Architecture and API:

A client-server architecture where each client request is handled using a distinct socket connection. After the server responds to a request, the connection is terminated. For any new request from the client, a fresh socket connection is established, ensuring that each interaction is isolated and managed independently. Thread-per-request model was used.

- **ToyStoreServer:** Hosts a toy catalog and provides two primary RPC services: **itemQuery** for querying the details of a toy (such as stock and cost) based on its name, and **itemBuy** for processing the purchase of a toy, updating stock accordingly. The server utilizes a **ConcurrentHashMap** for storing toy data, ensuring thread-safe access and modifications. It operates with a dynamic thread pool to efficiently manage concurrent client requests.
- **ToyStoreClient:** Consumes the server's services to allow users to query toy information and make purchases. The client sends requests to the server for either querying toy details or buying a toy, and it processes the server's responses accordingly. The ToyStore client calculates latency statistics for each toy ("animal") during buy and query operations by measuring the time taken for these operations to complete. It collects maximum, minimum, and total latencies for each operation type and toy, as well as the counts of calls made. These statistics are then used to calculate mean latencies. The client performs this for 1000 randomly chosen operations (either buy or query for one of

the predefined toys) and prints out the max, min, and mean latencies for each toy and operation type, as well as overall statistics for all calls.

Key Components

- **Toy Class:** Represents a toy in the catalog with properties like name, price, and stock.
- **ConcurrentHashMap:** Used on the server to store toy information, enabling thread-safe operations.
- **ThreadPoolExecutor:** Manages a pool of threads for handling incoming RPC calls on the server, configured with core and maximum pool sizes, keep-alive time, and a work queue capacity to ensure scalable performance.

Communication Protocol

Protobuf Schema (toyStore.proto)

- Defines the structure for request and response messages for toy queries and purchases.
- Specifies the gRPC service, including RPC methods for querying toy information (itemQuery) and purchasing toys (itemBuy).
- gRPC is used for communication between the client and server, with the server defining two services (ToyService):
  - itemQuery: Takes an ItemQueryRequest containing the name of the toy and returns an ItemQueryResponse with the cost and stock.
  - itemBuy: Takes a BuyRequest with the name of the toy to purchase and returns a BuyResponse indicating the result of the purchase attempt (success, out of stock, or item not found). The method returns 1 if the call succeeds, 0 if the item is not in stock and -1 if an invalid item name is specified.

Error Handling and Synchronization

- The system handles errors such as querying for a non-existent toy or attempting to purchase a toy that is out of stock. In such cases, the server responds with appropriate error codes or indicators (e.g., stock = -1, cost = -1.0 for queries, and result codes for purchase attempts).
- Synchronization is used in the ItemBuy method to ensure that stock updates are atomic, preventing race conditions when multiple clients attempt to buy the last toy simultaneously.
- Synchronization is used in the ItemQuery method to ensure that stock reads and writes dont overlap and cause inconsistencies.

Scalability and Performance

- The server's thread pool and concurrent data structures are designed to handle multiple concurrent client requests efficiently, ensuring scalability.

- The dynamic thread pool allows the server to adjust the number of active threads based on demand, optimizing resource utilization.

Citations:

1. https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.html
2. https://grpc.io/docs/languages/java/basics/
3. https://grpc.io/docs/what-is-grpc/introduction/