Design Optimization of Computing Systems
Assignment 3: DOCS DB - A High Performance Key-Value DB

Deadline: 11:59 PM, 29th November, 2024 **Maximum Marks : 300**

General Instructions:

1. This assignment is supposed to be done in **groups** of size 1 - 3
2. The **deadline** of the assignment is set as **11:59 PM, 29/11/2024 (Hard Deadline)**
3. There are **no restrictions** in selecting **tech-stack** or **language** for this assignment
4. Implementation **modularity, and code quality** will be considered in evaluation
5. Apart from the optimizations mentioned, you are encouraged to implement any other optimizations which you think would be useful

---

DOCS DB is a key-value based high performance database. It allows you to efficiently store and retrieve data based on unique keys. DOCS DB operates as a standalone system rather than having a distributed architecture. While it lacks the benefits of distributed systems, DOCS DB provides advantages such as simpler deployment and maintenance, making it easier for developers to manage their data. DOCS DB is suitable for applications such as content management systems, agile development systems etc.

In this assignment you are going to implement DOCS DB step by step. In the first step you will be designing and testing the DOCS DB engine which can be run locally. In the second part you will be playing with Network Namespaces and Kernel Bypass libraries such as DPDK. This will help you decide your network infrastructure which you will be integrating with your database. In the third part, you will put part 1 and part 2 together to create the DOCS DB. More details below.

This assignment is supposed to give you an idea of the following things:

1. How a database is designed from scratch, to be specific, what data structures are most suitable, data storage techniques, caching heuristics etc.
2. How to test and choose a proper network stack for your task.
3. How to integrate a database with network stack and make the most out of it.

Now, we present the assignment in 3 parts below:

Part 1: Implementing a Key-Value based Storage Engine

This part of the assignment involves the development of a key-value based storage engine.

1. You will implement the **storage engine as a static library**.
2. You will implement a **simple REPL to demonstrate** the functioning of the storage engine, **in a main file**.

The goal of this assignment is to evaluate your design on its performance for `READs (GET)`, `INSERTs (SET)`, and `DELETEs (DEL)`.

1. The evaluation will involve running a set of benchmark files of queries (synchronously), containing a mixed workload. Hence it is advisable to choose your design such that no query type is too slow. You NEED TO optimize for a particular workload among the following:
    - **read heavy** (less writes),
    - **balanced** (reads, and writes are comparable),
    - **write heavy** (reads are infrequent)

2. You will be required to submit a design document describing your engineering decisions, and optimizations. This document will specify:
    - **which among the above 3 workloads** you are optimizing for,
    - **the data structures** (BTree/LSM/anything else),
    - **concurrency control** you are using,
    - **any other specific optimizations** (eg, shared-nothing architecture, caching etc.)

3. Taking advantage of multiple cores is advisable. The number of cores to be utilized should be configurable in your program. We will evaluate the programs with 4 cores.

Note 1, as mentioned above you have to make all the design decisions for the storage engine yourself.

Note 2, in further parts **we will extend the engine into a server, and the system will take on client-server architecture using the Redis protocol**. It is advisable to separate the implementation of the REPL, and the storage engine, for simpler reuse later.

The REPL only supports the following commands-

1. **`SET <key> “<value>”`**
2. **`GET <key>`**
3. **`DEL <key>`**

Part 2: Exploring Network Namespaces and Kernel Bypass Techniques

In this part, you will be exploring network virtualisation tools *i.e.* network namespaces as well as kernel bypass techniques using DPDK. The description of the task is as follows:

1. Create three namespaces *viz.* NS1, NS2 and NS3. Configure them such that NS1 and NS3 have only one *veth* interface and NS2 has 2 *veth* interfaces.

2. Use `brctl` command to make NS2 a bridge between NS1 and NS3 so that data from NS1 can be forwarded to NS3.

3. Use the `tc` command to introduce delay(100ms) and packet loss(5%) in the corresponding *veth* interface in NS3 namespace. This gives you a realistic environment to test your server, where there may be network delays and packet losses.

4. Now consider the following two cases:

a. **Case 1:** Create a simple TCP client and server application and run the client on NS1 and server on NS3.
b. **Case 2:** Create another server application using DPDK (see examples) and run the client(same as in case 1) on NS1 and server on NS3.
Note: Your server should be able to send and receive TCP packets. You are free to use a user-space TCP library built on top of DPDK such as F-Stack.

Functionality:

- The client sends a random number of bytes (between 0.5 - 1KB)
- The server replies with the length of the data sent.

5. In both the above cases, send 10,000, 100,000, 1,000,000 packets from client to the server and report the following metrics:
    a. Latency
    b. Bandwidth
    c. CPU and Memory utilization (on server side)

Part 3: Putting it all together, DOCS DB

Now we will extend the storage engine into a full fledged database, DOCS DB. To do this we will extend the storage engine, by adding a TCP server, and a connection management layer.

1. Implement a server, and a CLI client using the Redis wire protocol (RESP-2 redis-specifications/protocol/RESP2.md at master).

    a. The server will implement connection management, and asynchronously handle multiple (TCP) clients.
    b. Explain the design decisions in the connection management layer, in a design document.
    c. The client will connect to the server, and implement the REPL (from Part-1), this will demonstrate the queries.

2. We will evaluate you (partly) using `redis-benchmark` utility from the official Redis project, and you will be required to submit the results for GET, and SET queries, with different concurrencies, and RPS.
    a. You need to submit redis-benchmark results for **10,000, 100,000, and 1,000,000** concurrent (`SET`, `GET`) requests over **10, 100, and 1000 parallel connections** respectively.
    b. Do the benchmark **with and without DPDK** on the server side
    c. Hence 3 × 3 × 2 benchmark reports (concurrent request x parallel connections x with/without DPDK)

Note, given that `redis-benchmark` will be used as a part of the evaluation, make sure to correctly implement the RESP2 specification :)

Unit Tests and Documentation

Writing unit tests and documentations are one of the most fundamental parts of a software. So, in this part you will be writing unit tests and documentation.

For unit tests:

Use a proper unit test framework based on the tech stack you have chosen, for example in case of C/C++ one such well known unit test framework is GTest.

For Documentation:

For documentation, we will be using doxygen to generate documentations(in pdf as well as html) of your software. Write doxygen style comments and create a Doxyfile(doxygen config file for your project).

Note: Make sure that doxygen produces proper documentation of your project.

Deliverables

You are required to submit the following–

Note 1, Make a sub-directory for each part (`part-a`, `part-b`, and `part-c`),
Note 2, The design document, and documentation of each part should be in the `docs` subdirectory, and source code should be in the `src` subdirectory.

1) **Part-A:**
   a. The code, and a shell script/Makefile for building and running the code.
   b. A design document as detailed in the problem statement.
   c. Documentation PDF generated using Doxygen.
2) **Part-B:**
   a. A shell script which creates/deletes (and configures) the namespaces and the corresponding *veth* interfaces
   b. The code for client and servers in the two cases described
   c. A Makefile/script to run the code and generate the metrics
   d. A report of the metrics under different load situations
3) **Part-C:**
   a. The code, and a shell script/Makefile for building and running the code.
   b. A design document as detailed in the problem statement.
   c. Documentation PDF generated using Doxygen.
   d. Redis-benchmark results in a `result` directory, 3 × 3 × 2 files, named as, `result_{concurrent_requests}_{parallel_connections}_{with_dpdk}.txt`,

      for example `result_1000000_100_1.txt` for the case with 1,000,000 concurrent requests, 100 parallel connections, with DPDK enabled.

Submission Guideline

Zip all the files into a single zip file named `Asgn3_Group_<Group_No>.zip` and upload it on moodle.

Evaluation Guidelines

The total marks for the assignment = 300
The marks breakup is given below:

Parts	**Marks**
Part 1	90
Part 2	70
Part 3	100
Documentation and Unit tests	40
Total	300