Submitted by – Amol Dattatray Sangar

Username - asangar

REPORT – Assignment 1

Architecture

- 1 TCP Server built on NodeJS
- 1 DB which is basically multiple files (1 file per key) on backend stored in 'db' folder
- Client - supports two types of clients
    1. Normal clients connected through Socket
    2. Memcached clients (Bonus section)

Protocol

SET COMMAND –

set <key> <flags> <exptime> <bytes> [noreply]\r\n

<data block>\r\n

Ex – set testkey 0 0 5

Value

Flags and exptime are stored for future purposes

GET COMMAND –

get <key>* \r\n

- <key>* means one or more key strings separated by whitespace.

After this command, the server sends zero or more items, each of which is sent as a text line followed by a data block.

Concurrency control

Concurrent connections over server are handled by NodeJS using threads and queues internally.

Concurrency limit - In NodeJS concurrent connections depend upon a lot of factors, like CPU, ram, DB, network speeds, dataset size etc. Thus, the server can handle over **10000 connections** easily if it is involved in I/O operations alone.

- DB concurrency -

Concurrency over DB/Filesystem is handled by making use of a **lock for mutual exclusion**. Lock is created for each read/write over a single key and the other process trying to access the same key cannot work until the lock is acquired.

Since there are multiple files (1 file per key), there is no contention for queries on different keys and thus improving the performance of the system.

Furthermore, **reads are prioritized over writes** and they can skip the queue for faster processing.

Installation

Run – npm install

Server execution

Run – node server.js

Test cases

1. 1000 simultaneous connections to server (CONNECTION TEST)

Run – node test1.js

Connection details can be seen on server. Press CTRL+C to kill all processes

2. 100 connections to server and each perform 1 write and then 1 read

Run – node test2.js

Output is shown on parent process which forks all other processes and hence the output is not in the same order or formatting always if sleep is not added. Hence added 200ms sleep for better visibility of output. Press CTRL+C to kill all processes

3. 1 client writes to 1 key every 0ms and another client reads from the same key every 0ms for 100 times (Performance + Concurrency test)

Run – node test3.js

Output is shown on parent process which forks all other processes and hence the output is not in the same order or formatting always. Press CTRL+C to kill all processes.

4. 2 clients write to same key at the same time for 500 times (Concurrency test for DB and multiple requests from a single client test).

Run – node test4.js

No collision or error occurs and can be checked on the server.

Press CTRL+C to kill all processes.

5. Interactive client terminal test

Run – node interactive_client.js

Test with large data provided in the folder by the name ‘large_text.txt’ for setting value by ‘set’ command. Press CTRL+C to exit or type ‘quit’ to exit.

6. Memcached client test

Run – python memcached_client.py (PYTHON Client)

Press CTRL+C to exit or type ‘quit’ to exit.

NOTE - Each client runs on a separate process and thus have its own V8 engine instance

Key and value size limits

Key – 250 characters

Value – 1048576 characters (1MB)

(Restricted as it takes a lot of time to capture input line by line from console)

Warning – Don’t try to copy text over 10-20KB as it takes enormous time to read from console

Errors to look out for -

If server exits then its breaks all the clients’ connections as well.

Future improvements

Faster and larger input capturing through console

Taking input files from system

Multiple copies of DB/Files for performance improvement

Better validation and error handling

Better backend system like Redis