CSE 511: Lab 1 Documentation

Team: Game of Threads

***Members:*** **Soumen Basu**, **Makesh Tarun Chandran**, **Vivek Bhasi**

1. **Problem Statement:**

* Implement a key-value store using three different styles of concurrency and IO management.
* Perform a comparative experimental performance evaluation of the three implementations.
* The three styles are:

1. MT: multi-threaded with one thread handling per client request and using blocking IO for both sockets and files.
2. EP :a single event-driven thread with polling-based event notification, and (iii)
3. ES: a single event-driven thread with signaling-based event notification

* For styles (ii) and (iii) use Non-blocking or asynchronous sockets(as appropriate). However, for file IO blocking IO calls will be used that will be made by “helper” threads that will communicate with the main event-driven thread via shared memory mechanisms and exercise appropriate care for synchronization.

1. **What we managed to do:**

We were successful in implementing the designs and also were able to experimentally evaluate Design 1 and 2. We used Python3 to implement design 1 and 2. Design 3 was implemented in C. The setup is described in detail in the “Experimentation” section of this document. The code is well documented using comments. We couldn’t complete the experimental evaluation of design 3.

1. **Issues faced:**

Implementing design 3 is *probably* not possible in Python3. **siginfo\_t** in python is a class abstraction of the C structure. But the implementation contains only a few struct variables from the C siginfo\_t and it doesn’t contain the fd. Python provides some high level libraries (asyncore/ asyncio) and conditional variables but our code inspection revealed that the underlying mechanism is *select* based polling.

1. **Populating the DB:**

Can be populated using the script *populate\_db.py*

**python populate\_db.py -n <num\_entries> -s <key-value-size> -d <db\_name>**

1. **Requirements:**
   * Python 3
   * PickleDB for persistent storage (pip install pickledb)
2. **Running Servers:**

Running the MT server:

**python multi\_threaded\_server.py <0.0.0.0> <port> <cache\_size> <db\_name>**

Running the EP server:

**python polling\_server.py <0.0.0.0> <port> <cache\_size> <db\_name> <num\_helpers>**

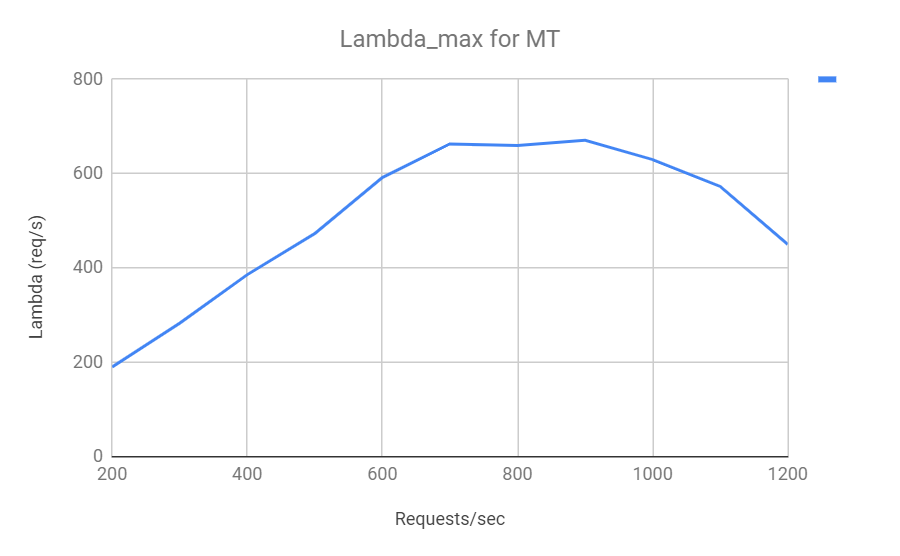
Running the ES server:

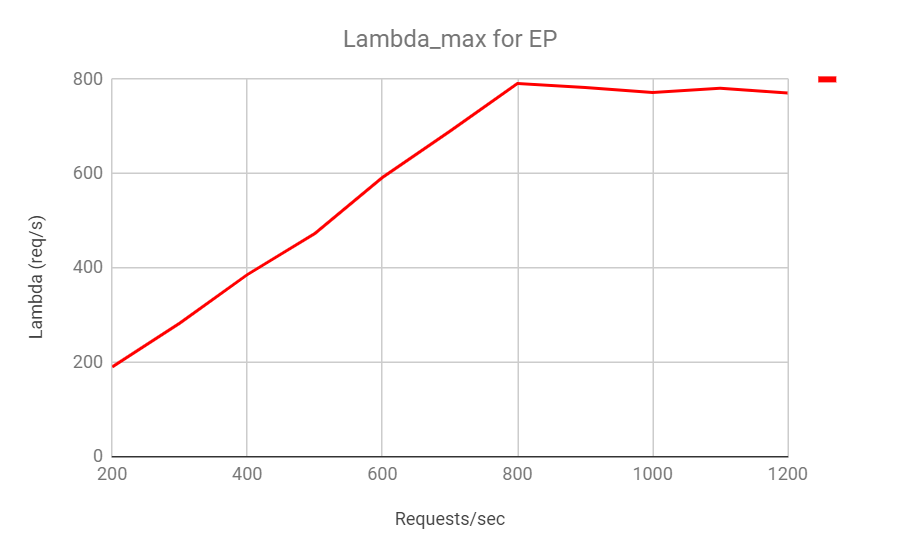
**gcc -pthread server.c -o server.o**

**./server**

**[configs for the server host, port, cache\_size, num\_helpers are in server.h]**

1. **Key-Value Pair:**
   * Write-back, Allocate on Write Cache policy
   * LRU eviction policy
   * The following APIs are released to client:
     1. Get(Key) --> Value/ -1
     2. Put(Key, Value) --> ACK/ -1
     3. Insert(Key, Value) --> ACK/ -1
     4. Delete(Key, Value) --> ACK/ -1
2. **ewfew**
3. **Experimental Setup and results:**
   * Server: We ran on a t2.small AWS EC2 instance (Ubuntu 18.04 w/ 20GB gp2)
   * Client Node: t2.xlarge (Ubuntu 18.04 w/ 20GB gp2)
   * Request Distribution is deterministic in nature (after 100s increase the number of requests by 100).
   * Skewed key popularity distribution: 10% keys make 90% requests
   * **Lambda\_max Measurement:**
     1. We started generating requests with the above stated request distribution.
     2. We spawned processes from the client node - each process generates 10 req/sec
     3. We measure the new requests in server using *Wireshark* and *tcptrace* tools (based on appropriate filtering).
     4. The trend that we observe is the lambda in server increases with the number of requests and then flattens, finally decreases from the max. (We believe this is the DoS phase).
   * Lambda\_max plots:(Next page)
     1. MT: lambda\_max = 663 (we approximated this to 650)
     2. EP: lambda\_max = 791 (approximated to 800)





**GET/ PUT Latencies for different settings:**

(get:put ratio is the first column) 50 -> 50%-ile latency, 95 -> 95%-ile latency

Key-value sizes: (100, 1000, 10000) B

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| lambda\_max | | 100 | | | | 1K | | | | 10K | | | |
| GET Latency | | PUT Latency | | GET Latency | | PUT Latency | | GET Latency | | PUT Latency | |
| 50 | 95 | 50 | 95 | 50 | 95 | 50 | 95 | 50 | 95 | 50 | 95 |
| MT | 90:10 | 63 | 556 | 300 | 1200 | 850 | 1080 | 1186 | 2109 |  |  |  |  |
| 10:90 |  |  |  |  |  |  |  |  |  |  |  |  |
| 50:50 |  |  |  |  |  |  |  |  |  |  |  |  |
| EP | 90:10 | 18.15 | 67.2 | 18.4 | 68.6 | 81 | 795 | 600 | 905 |  |  |  |  |
| 10:90 |  |  |  |  |  |  |  |  |  |  |  |  |
| 50:50 |  |  |  |  |  |  |  |  |  |  |  |  |
| ES | 90:10 |  |  |  |  |  |  |  |  |  |  |  |  |
| 10:90 |  |  |  |  |  |  |  |  |  |  |  |  |
| 50:50 |  |  |  |  |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| lambda\_max/ 2 | | 100 | | | | 1K | | | | 10K | | | |
| GET | | PUT | | GET | | PUT | | GET | | PUT | |
| 50 | 95 | 50 | 95 | 50 | 95 | 50 | 95 | 50 | 95 | 50 | 95 |
| MT | 90:10 |  |  |  |  |  |  |  |  |  |  |  |  |
| 10:90 |  |  |  |  |  |  |  |  |  |  |  |  |
| 50:50 |  |  |  |  |  |  |  |  |  |  |  |  |
| EP | 90:10 | 6 | 42.9 | 8 | 44.1 |  |  |  |  |  |  |  |  |
| 10:90 |  |  |  |  |  |  |  |  |  |  |  |  |
| 50:50 |  |  |  |  |  |  |  |  |  |  |  |  |
| ES | 90:10 |  |  |  |  |  |  |  |  |  |  |  |  |
| 10:90 |  |  |  |  |  |  |  |  |  |  |  |  |
| 50:50 |  |  |  |  |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| lambda\_max/10 | | 100 | | | | 1K | | | | 10K | | | |
| GET | | PUT | | GET | | PUT | | GET | | PUT | |
| 50 | 95 | 50 | 95 | 50 | 95 | 50 | 95 | 50 | 95 | 50 | 95 |
| MT | 90:10 |  |  |  |  |  |  |  |  |  |  |  |  |
| 10:90 |  |  |  |  |  |  |  |  |  |  |  |  |
| 50:50 |  |  |  |  |  |  |  |  |  |  |  |  |
| EP | 90:10 | 7 | 45 | 8 | 46.3 |  |  |  |  |  |  |  |  |
| 10:90 |  |  |  |  |  |  |  |  |  |  |  |  |
| 50:50 |  |  |  |  |  |  |  |  |  |  |  |  |
| ES | 90:10 |  |  |  |  |  |  |  |  |  |  |  |  |
| 10:90 |  |  |  |  |  |  |  |  |  |  |  |  |
| 50:50 |  |  |  |  |  |  |  |  |  |  |  |  |