15-418 Assignment 4

Ziyuan Gong, Hongyu Li
ziyuang, hongyul@andrew.cmu.edu
April 5, 2017

1 Score Table

1.1 grading_wisdom.txt

```
*** The results are correct! ***

Avg request latency:  1877.83 ms
Total test time:      19.64 sec
Workers booted:       1
Compute used:         19.88 sec

----------------------------------------------------------------
Grade: 12 of 12 points
         + 2 points for correctness
         + 10 points for perf
         - 0 points for worker usage

         100.0% requests met the 2500 ms latency requirement
----------------------------------------------------------------
```

1.2 grading_tellmenow.txt

```
*** The results are correct! ***

Avg request latency:  1130.78 ms
Total test time:      13.84 sec
Workers booted:       1
Compute used:         14.09 sec

----------------------------------------------------------------
Grade: 12 of 12 points
         + 2 points for correctness
         + 10 points for perf
         - 0 points for worker usage

         100.0% of tellmenow requests met the 150 ms latency requirement
         94.2% of all other requests met the 2500 ms latency requirement
----------------------------------------------------------------
```

1.3 grading_compareprimes.txt

```
*** The results are correct! ***
```

```
Avg request latency:  869.65 ms
Total test time:      7.92 sec
Workers booted:       1
Compute used:         8.18 sec

-----------------------------------------------------------------
Grade: 12 of 12 points
         + 2 points for correctness
         + 10 points for perf
         - 0 points for worker usage

         100.0% requests met the 2000 ms latency requirement
-----------------------------------------------------------------
```

1.4 grading_uniform1.txt

```
*** The results are correct! ***

Avg request latency:  613.53 ms
Total test time:      21.89 sec
Workers booted:       1
Compute used:         22.13 sec

-----------------------------------------------------------------
Grade: 6 of 6 points
         + 2 points for correctness
         + 4 points for perf
         - 0 points for worker usage

         100.0% requests met the 2400 ms latency requirement
-----------------------------------------------------------------
```

1.5 grading_nonuniform1.txt

```
*** The results are correct! ***

Avg request latency:  1615.00 ms
Total test time:      68.52 sec
Workers booted:       4
Compute used:         136.86 sec

-----------------------------------------------------------------
Grade: 12 of 12 points
         + 2 points for correctness
         + 10 points for perf
```

```
        - 0 points for worker usage

        97.6% requests met the 2500 ms latency requirement
----------------------------------------------------------------
```

1.6 grading_nonuniform2.txt

```
*** The results are correct! ***

Avg request latency:  1946.98 ms
Total test time:      42.45 sec
Workers booted:       4
Compute used:         96.66 sec

----------------------------------------------------------------
Grade: 12 of 12 points
        + 2 points for correctness
        + 10 points for perf
        - 0 points for worker usage

        100.0% of project idea requests met the 4100 ms latency requirement
        91.3% of all other requests met the 2500 ms latency requirement
----------------------------------------------------------------
```

1.7 grading_nonuniform3.txt

```
*** The results are correct! ***

Avg request latency:  1417.37 ms
Total test time:      61.86 sec
Workers booted:       6
Compute used:         149.45 sec

----------------------------------------------------------------
Grade: 12 of 12 points
        + 2 points for correctness
        + 10 points for perf
        - 0 points for worker usage

        97.8% of project idea requests met the 4100 ms latency requirement
        99.6% of tellmenow requests met the 150 ms latency requirement
        97.9% of all other requests met the 2500 ms latency requirement
----------------------------------------------------------------
```

2 Master Design

2.1 Request Handling

- `compareprimes`: the master will convert the `compareprimes` request into 4 separate `countprimes` request, and schedule them as normal requests. After receving all the responses, the master will do the remaining work of comparison and send the result to the original request client.
- `tellmenow`: In order to meet the strict response latency requirement, we dedicate a single thread specifically for `tellmenow` request on the first worker (since the first worker will never be shut down).
- `projectidea`: In order to cope with the `projectidea` request, each worker is only allowed to process two `projectidea` requests at the same time. By using the `pthread_setaffinity_np` function, we set the 1_{st} and 2_{nd} thread to run on different processors so that there won't be conflicts on the L3 cache.

2.2 Scheduling

Our implementation for the master server can assign each request to a worker on a specific thread. We encode the thread id into the `tag` field of the request/response by setting the last two digits of the `tag` to the thread id we want to assign to. For example, if we want to assign the job of tag 4 on the thread 15, we will set the `tag` argument to be 415 in the `Request_msg` object.

The scheduleing algorithm is rather simple. Upon receiving a request from the client, we find whether there is any thread on each workers that is not processing any request. If yes, we assign to it directly, otherwise we will append it to our job queue. (`projectidea` job will only be assigned to thread 1 and 2).

Upon receiving response from the worker, we then know that one of the thread on this worker is now available to process a new job from the queue. Using this method, we can keep the threads busy since each thread will immediately work on a new job after finishing the previous one.

2.3 Scaling

Scalue up: In order to add more workers when the amount of requests is high. We adapted two queues to monitor the pending requests for `projectideas` and other requests except `tellmenow`. When the queue reaches a certain limit(1 for `projectideas` and 17 for common requests), we would add more workers to accommodate for the surging requests.

Scale down: Upon each `handle_tick()` call, we will loop through the workers and shut down the idle worker with no active threads. In the meantime we ensure that at least one worker is active all the time.

3 Worker Design

During the process of initializing each worker, our software will create 24 threads to process requests sent by the master. Each thread will have its own job queue (since our master can assign request to specific thread), and the thread routine is constantly poping jobs from its job queue and execute it.