Assignment 4

- Qiaoyu Deng (qdeng@andrew.cmu.edu)
- Changkai Zhou (zchangka@andrew.cmu.edu)

Results and Scores

1. wisdom

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

Avg request latency:  1578.02 ms
Total test time:      18.98 sec
Workers booted:       1
Compute used:         19.22 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
---------------------------------------------------------------------
```

2. compareprimes

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

Avg request latency:  868.90 ms
Total test time:      7.97 sec
Workers booted:       1
Compute used:         8.21 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
---------------------------------------------------------------
```

3. tellmenow

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

Avg request latency:  1075.33 ms
Total test time:      13.82 sec
Workers booted:       1
Compute used:         14.07 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
         100.0% of all other requests met the 2500 ms latency requirement
---------------------------------------------------------------
```

4. uniform1

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

Avg request latency:  653.15 ms
Total test time:      21.84 sec
Workers booted:       1
Compute used:         22.09 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
----------------------------------------------------------------
```

5. nonuniform1

```
Avg request latency:  1608.61 ms
Total test time:      68.46 sec
Workers booted:       3
Compute used:         128.93 sec

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

         99.9% requests met the 2500 ms latency requirement
----------------------------------------------------------------
```

6. nonuniform2

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

Avg request latency:  1685.02 ms
Total test time:      42.47 sec
Workers booted:       9
Compute used:         125.08 sec

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

         97.1% of project idea requests met the 4100 ms latency requirement
         97.4% of all other requests met the 2500 ms latency requirement
----------------------------------------------------------------
```

7. nonuniform3

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

Avg request latency:  1210.25 ms
Total test time:      62.65 sec
Workers booted:       10
Compute used:         182.17 sec

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

         82.6% of project idea requests met the 4100 ms latency requirement
         98.9% of tellmenow requests met the 150 ms latency requirement
         99.1% of all other requests met the 2500 ms latency requirement
----------------------------------------------------------------
```

Scheduling Algorithm

Master Scheduler

Scalability

1.Kick-Off

We have each worker 36 thread running at the same time, so the most number of requests that a worker can run at the same time is 36, but we want to limit this number to 24, because a worker can run 24 threads without conext switch(actualy this is not right, becasue we have kernel program and init process). We check whether should we scale out in time update process or when we assign requests to worker, and we check whether to scale in only in time update.

2.Scale-Out

Every time, when we will caculate the average number requests that assigned to all active(not include to be killed worker) workers. We have two matrix, one is CPU intensive(418wisdom or countprimes), another one is projectidea, if we find that the first one is going to be greater than 24 which indicates that worker will have to context switch to perform computation. So in that situation, we will try to find whether there exists worker is waiting to be killed, if so, we pull it back to active workers, otherwise, we will ask for new worker. Another scale out situation is inside scheduler, when we cannot find a worker with enough computing resources, we will either pull back worker or ask for a new one.

3.Scale-In

For scale in, we also check the average number of CPU intensive requests and projectidea requests, when it is lower than a threshold, we scale worker in, mark it as to be killed. However, we do not kill it immediately, because we do not want worker to be created or killed very quickly, so we wait for the killing timer go to zero and worker completes all the requests that is assigned. When worker is waiting to be killed, it will possibly be put back to active list if there are a huge amount of requests incoming.

Scheduling of Requests

Following the rules below:

- We record how many requests are send to worker in master, and each time we will look at the request type, and then find the worker with the least number of that request.
- For projectidea request, we try to assign this type of request at most two to every worker, when we cannot find a worker that has less than 2 projectidea(because this request will consume all L3, and each worker has two sockets, we have 2 L3 cache for each worker.), we will first look at whether there exists worker is waiting for being closed, if so, we assign the projectidea to this worker, and scale it back to active worker.
- Same with projectidea, we call countprimes and 418wisdom requests as CPU intensive requests, because they seldom use memory but CPU. we will want to limit the number of CPU intensive work to 24(because worker has 2 sockets, each of socket has 6 cores, with hyper threading, so we have 24 hardware context at the same time), if we cannot find a worker that has less than 22(24 - 2, reserved for) requests, we will first try to assign requests to worker that is going to be killed, and scale it back to active worker, if not possible, we just scale out to call new worker.
- We set no limit to tellmenow, because it can be respond very quickly.
- One try is that we predicate the complexity of countprimes by its n, because we believe the

greater n is, the more complex the computation will be, so we try to assign the same sum of countprimes n to each worker to get further work balance.
- For compareprimes, we just break it into 4 countprimes requests and send them to different worker in parallel. And then group them into a single response to worker.

Individual Worker

- Use *queue* to store all requests received from Master
- Use 3 *FIFO* to execute requests, one is for tellmenow, one is for projectidea, one is for CPU intensive requests. We let worker always excute tellmenow first until there are no more tellmenow, and then try to execute projectidea only when current running projectidea is less then 2. Or we will choose CPU intensive until we run out of requests.
- Create around 36 *threads* for each worker and idle *threads* will keep monitoring the *queue* to get new task when the *thread* receive the "waking-up" signal from new incoming requests.