15-618: Assignment 4
A Simple Elastic Web Server

Eshan Verma
Carnegie Mellon University
everma@andrew.cmu.edu

Karandeep Johar
Carnegie Mellon University
kjohar@andrew.cmu.edu

1. The Master: The Evenstar

Our master is a serial implementation acting as the front-end to all clients. There is no special modification done for any particular trace and implementation is general and should scale to any trace. The master is responsible for handling client requests, forwarding it to a worker, load balancing, elasticity as well as caching of requests. Low level details of each of these is explained in the following sections:

1.1. Handling Requests & Caching

The master adds the request tag to the client request and forwards to the master. The master is also responsible for correctly routing the worker responses to the correct client.
For compare primes, instead of handling the comparison in the worker, the request is broken in to 4 parts, and each request is independently handled. This has the advantage of exploiting the cache and avoiding overloading of one worker.
For cache hits, a unique strategy implemented by us is to queue up pending cache updates. For count primes, some numbers might be repeated, however, before the cache is populated, the number might be requested again. To avoid sending this request to the worker and having duplication of work, we queue up the requests and serve from cache once cache is populated.
Upon receiving a response from worker, the cache is populated and various queues are drained. One of the queues was mentioned above, the other queues will be explained in the next section.

1.2. Scheduling

The scheduling and elasticity policies implemented are a function of the configuration of the workers. We employ and exhaustion-based scheduling to have a chance for a worker to be killed and utilize the minimum amount of workers.
We have multiple queues in the master for handling various requests. A queue maintains generic requests like wisdom and count primes. A separate queue exists for project ideas and tell me now requests are never queued. The scheduler decides which worker to route the request to based on the following heuristic:

- Fill up one worker always till NUM_THREADS

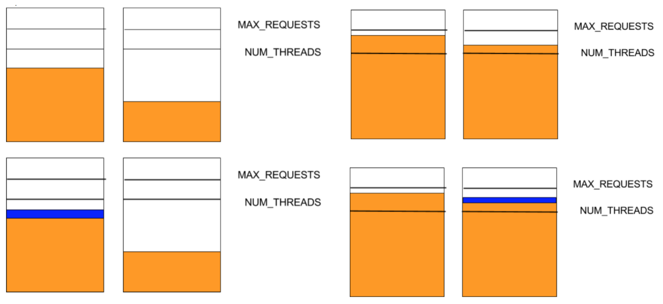


Figure 1: This is just fantasy

- Once all workers are at NUM_THREADS, fill evenly till the next threshold

This is illustrated in the diagrams. This strategy is for two reasons:

- To allow workers become idle and be shut down
- Utilize the present worker the fullest

Highest priority is given to the project ideas queue.

Upon receiving a response the queues are drained to reload the worker with minimal overhead.

1.3. Elasticity

Scaling-Up

- Whenever there is an outstanding project ideas request or whenever one worker gets NUM_THREADS amount of work.
- This is done while ensuring that while one worker is booting, another is not booted.

Scaling-Down

- Tick is called every 1 seconds and kills at most one worker if the the worker is idle.
- Idle Worker is a necessity to avoid any lost requests.

2. The Worker - Threadpool

The worker is responsible for executing the request and generating the response. The worker has a fixed thread pool and this pool grabs work from a shared queue.
The worker maintains three queues for different types of request. One for generic requests, one for project ideas and one for serving the fast tellmenow requests.
To accelerate performance one thread from this thread pool has affinity set to core 0, while all other threads are set to cores 1-23. The thread on core 0 serves generic requests until a project request comes, in which case it exclusively serves project requests. Once project requests are done, the thread can go back to serving generic requests. This is done to:

- Ensure no L3 thrashing due to multiple project idea requests
- Ensure all cores are available for generic requests

3. Special Cases: “You spin me right round”

Some optimizations we did based on the input request type. No trace-specific changes were done in the code.

3.1. Core 0 Affinity

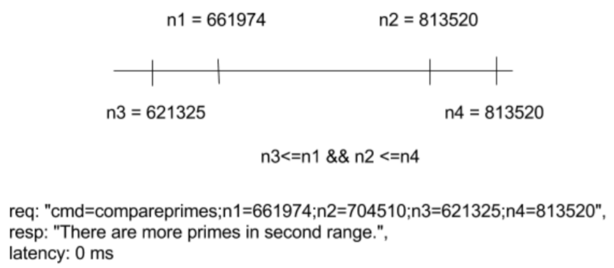
All core 0’s in our worker implementation have only one thread pinned on them. This is done to:

- Have all cores available for generic requests
- Serve projectideas with no thrashing

The thread can switch between serving either of these requests. We have implemented this by modifying the work-queue implementation to provide a thread safe method to poll the size of the queue, without the polling thread going to sleep. This allowed to have a thread worker implementation which could easily switch between either requests by simply polling this attribute and not being put to sleep.

3.2. Compareprimes

- We could parallelize the computation of a single compareprime task into 4 count primes task in the master to take a better advantage of the parallelism and to even out the workload across worker machines. We transferred this part of the logic from the worker to the master.
- Trivial return of results. Let's say the arguments for this command are $n1 = 3$ $n2 = 50000000$ $n3 = 2$ $n4 = 10000000000$. Then without sending the request to the worker we can conclude that there more primes between $n3$ and $n4$ than between $n1$ and $n2$. That is because $n3 \leq n1$ and $n2 \leq n4$. So because the range of $n3$ and $n4$ overlaps $n1$ and $n2$ we are guaranteed that there will be at least as many primes in the second range as there are in the first range. We also took care of the symmetric case and lessened the load on the worker nodes.



```
req: "cmd=compareprimes;n1=661974;n2=704510;n3=621325;n4=813520",
resp: "There are more primes in second range.",
latency: 0 ms
```

Figure 2: Breaking down compare-primes

3.3. Projectidea

- Separate pipeline for projectidea on master and worker(Ensure that only worker processes a projectidea task at a time.)
- Smart implementation to ensure that it always run on core 0.
- While no project ideas are pending, the thread serves generic requests.

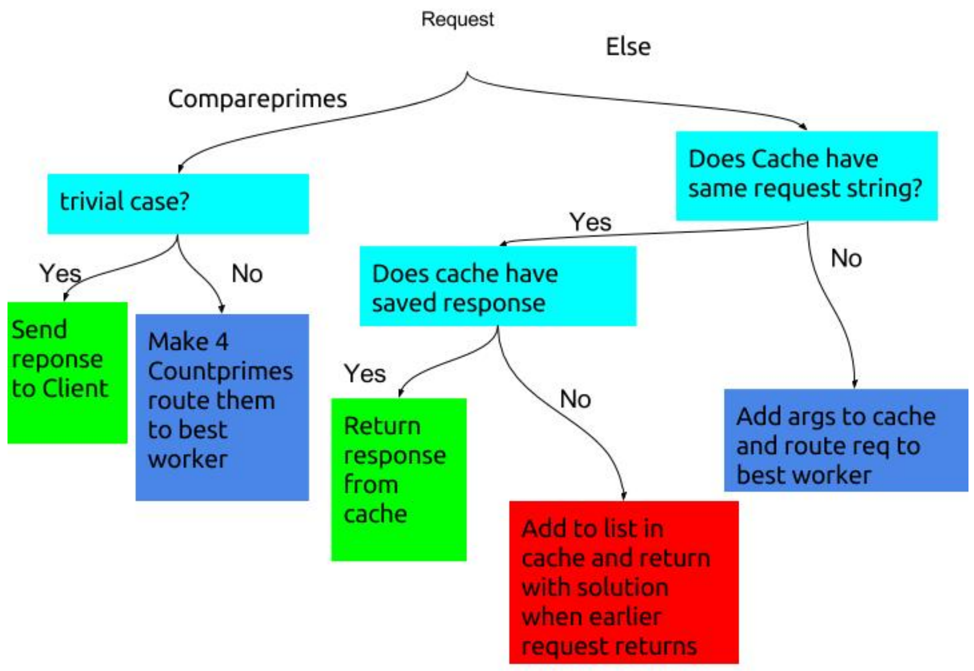
3.4. TellMeNow

We just kept a floating thread which could take any of the cores from 1-23 (0 indexed) to meet pretty strict latency requirement.

3.5. Caching

We observed in uniform and a couple of nonuniform tests that they were requests strings that repeated. So if we could store this result in a map we would not have to resend the later requests to the workers. So a naive implementation would be to check if the result for a particular request string is in the cache If present return the result else send the request to the worker and store the results in the cache.
But in uniform we saw that multiple requests for counting primes for a number like "$cmd = countprimes$;$n = 779069$" came in quick succession. That is even before the first request for "$cmd = countprimes$;$n = 779069$" could return a new request for the same argument came in. In a naive implementation we would send this request to the worker. But in a smart implementation we could keep all the subsequent requests for "$cmd = countprimes$;$n = 779069$" in a data structure and return the request responses when the first request returns a response. So in the naive implementation we would send in two requests for the same arguments. In the better implementation we would send in a single request to reduce load on the worker in lieu of some additional bookkeeping. Better caching of requests and responses.

4. Logic Flows



Request
Else
Compareprimes
trivial case?
Does Cache have same request string?
Yes
No
Yes
No
Send reponse to Client
Make 4 Countprimes route them to best worker
Does cache have saved response
Yes
No
Return response from cache
Add to list in cache and return with solution when earlier request returns
Add args to cache and route req to best worker

Figure 3: Handling Client Request

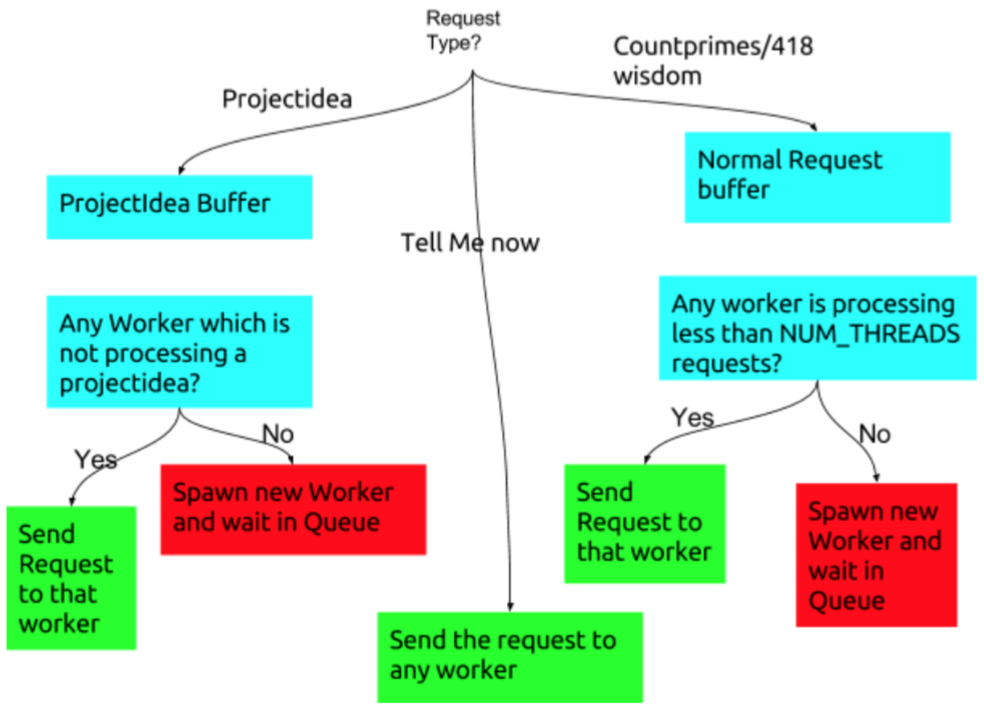

Request Type?
Projectidea
Countprimes/418 wisdom
Tell Me now
ProjectIdea Buffer
Normal Request buffer
Any Worker which is not processing a projectidea?
Yes
No
Send Request to that worker
Spawn new Worker and wait in Queue
Send the request to any worker
Any worker is processing less than NUM_THREADS requests?
Yes
No
Send Request to that worker
Spawn new Worker and wait in Queue

Figure 4: Handling Client Request-2

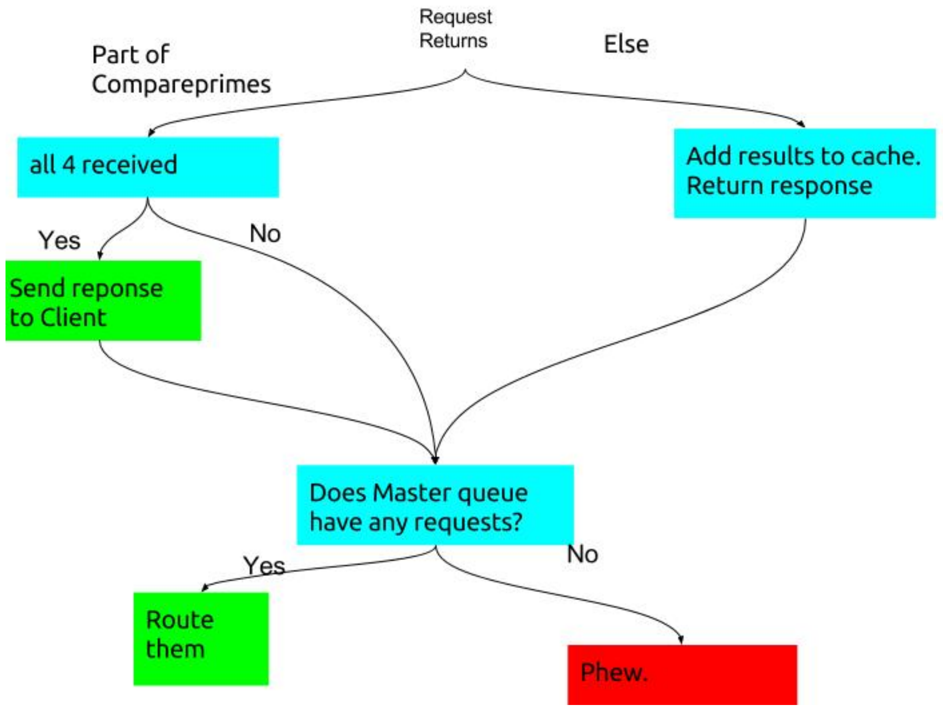


Figure 5: Handling Worker Response

5. Results

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

Avg request latency:  1458.37 ms
Total test time:      27.58 sec
Workers booted:       1
Compute used:         28.04 sec

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

         99.2% of tellmenow requests met the 150 ms latency requirement
         82.5% of all other requests met the 2500 ms latency requirement
----------------------------------------------------------------
NON_UNIFORM 2
*** The results are correct! ***

Avg request latency:  1909.74 ms
Total test time:      47.65 sec
Workers booted:       10
Compute used:         140.87 sec

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

         90.0% of project idea requests met the 4100 ms latency requirement
         90.0% of all other requests met the 2500 ms latency requirement
----------------------------------------------------------------
NON_UNIFORM1
*** The results are correct! ***

Avg request latency:  1811.10 ms
Total test time:      121.03 sec
Workers booted:       5
Compute used:         232.25 sec

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

         95.4% requests met the 2500 ms latency requirement
----------------------------------------------------------------
NON_UNIFORM 3
*** The results are correct! ***
```

```
Avg request latency:  1165.35 ms
Total test time:      121.92 sec
Workers booted:       27
Compute used:         318.84 sec

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

         94.6% of project idea requests met the 4100 ms latency requirement
         98.6% of tellmenow requests met the 150 ms latency requirement
         96.4% of all other requests met the 2500 ms latency requirement
----------------------------------------------------------------
COMPARE_PRIMES
*** The results are correct! ***

Avg request latency:  1223.82 ms
Total test time:      18.82 sec
Workers booted:       1
Compute used:         19.30 sec

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

         91.4% requests met the 2000 ms latency requirement
----------------------------------------------------------------
UNIFORM1
*** The results are correct! ***

Avg request latency:  644.11 ms
Total test time:      42.07 sec
Workers booted:       1
Compute used:         42.21 sec

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

         98.6% requests met the 2000 ms latency requirement
----------------------------------------------------------------
WISDOM
----------------------------------------------------------------
Grade: 12 of 12 points
         + 2 points for correctness
         + 10 points for perf
```

```
        - 0 points for worker usage

        94.8% requests met the 2500 ms latency requirement
-----------------------------------------------------------------
```