Parallel Computer Architecture and Programming

Assignment 4 - A Simple Elastic Web Server

Che-Yuan Liang
cheyuanl@andrew.cmu.edu
Mao Yu
ymao1@andrew.cmu.edu

We summarized our work into 3 stage:

1. Correctness
2. Extending Functionality
3. Parameter Tuning

Overall speaking, we optimize the performance to get full performance credit first, then adding the elastic scheduler.

1. Correctness

+ Allow master process with multiple request at the same time by adding a queue result:

2. Extending Functionality

+ Adding a priority queue, dispatch the priority queue whenever master got worker response, then dispatch the normal queue.

+ Initializing number of worker to 4 without elastic

+ Adding pthread worker pool in worker.cpp, supporting 24 threads.

+ Divided compare prime task for 4 threads in the worker.cpp

	wisdom	compare prime	tellmenow	non-uniform1	non-uniform2	non-uniform3	uniform1
performance	10	10	10	10	3	7	3
worker usage	-5	-5	-5	-5	-3	-5	-2
					5.0% of project idea	19.6% of project idea	78.3% requests met

+ Mutual exclusive project idea, every worker <= 2 project idea

	wisdom	compare prime	tellmenow	non-uniform1	non-uniform2	non-uniform3	uniform1
performance	10	10	10	10	**8**	7	3
worker usage	-5	-5	-5	-5	-3	-5	-2
					67.5% of project idea	52.2% of project idea	

+ Cache for all request: we use a std::map to store all of the request/response

	wisdom	compare prime	tellmenow	non-uniform1	non-uniform2	non-uniform3	uniform1
performance	10	10	10	10	8	7	**4**
worker usage	-5	-5	-5	-5	-3	-5	-2

+ All worker node reserves 1 slot for project idea

	wisdom	compare prime	tellmenow	non-uniform1	non-uniform2	non-uniform3	uniform1
performance	10	10	**6**	10	**9**	**10**	4
worker usage	-5	**-3**	-5	**0**	**-2**	-5	-2
			50.0% of tellmenow		87.5% of project idea		

+ Let tellmenow use the reserved space (tellmenow is quick)

	wisdom	compare prime	tellmenow	non-uniform1	non-uniform2	non-uniform3	uniform1
performance	10	10	**9**	10	**10**	10	4
worker usage	-5	-3	-5	0	-2	-5	-2
			85.0% of tellmenow		97.5% of project idea		

Now we almost get full credit on performance. However, the performance decreased. By observing the logger we find the server takes 2 seconds to fire up a worker. which is bad for the spiky request that require low latency.

+ add simple elastic:

if queue = 0 and a worker node has no task -> delete node
if queue > 0 request a new node

	wisdom	compare prime	tellmenow	non-uniform1	non-uniform2	non-uniform3	uniform1
performance	10	10	**5**	10	**4**	**8**	4
worker usage	**-1**	**0**	-2	0	-2	0	0
			35.8% of tellmenow		35.0% of project idea	45.7% of project idea	

We switch to smart working assignment, also changed the context or worker node to be 48.

+ **Smarter elastic**:

If the current workload (num of running task on a node) is larger than a threshold, then fire up a new node. Previous, we fire up the node only when the queue is growing, which is too late for the spike request that need low latency.

+ Increase max context

As suggested in the piazza. More context might squeeze more computation from the operating system. So use choose to spawn **1 + 47 pthreads** in worker.cpp.

+ Two level assignment

If number of running task is grater than 24, than search for another node. If all workers have more than 24 tasks, than assign more job till 48 context are filled. Since the margin effect of adding task after 24 context decreases.

There = 0.7	wisdom	compare prime	tellmenow	non-uniform1	non-uniform2	non-uniform3	uniform1
performance	10	10	**10**	10	**3**	8	**3**
worker usage	**0**	0	**0**	0	0	0	0
			35.8% of tellmenow		17.5% of project idea	45.7% of project idea	86.7%

- Threshold might be too large.
- Current strategy shares two slot for project idea and tellmenow, if project idea has occupy all of the reserved slots, tellmenow no longer has priority.

+ Preserving a thread and a project idea queue exclusively in charge of projected in the worker node

We find that there are too many project idea in non-uniform2, and they need priority, so we add a thread exclusively for project idea.

+ Decrease the worker node fire up threshold to 0.6

In order to response to the spiky requests earlier.

There = 0.6	wisdom	compare prime	tellmenow	non-uniform1	non-uniform2	non-uniform3	uniform1
performance	12	10	**10**	10	**8**	**10**	**3**
worker usage	**0**	0	**0**	0	0	0	0
					77.5% of project idea		86.7% request met

Summary

- **Describe the basics of a worker. How is work received by the worker assigned to threads? How many threads did you create and why? etc.**

The worker has **48 pthreads**, one of them is exclusively work for project idea. Theoretically, the worker node contains two, six-core Xeon e5-2620 v3 processors (2.4 GHz, 15MB L3 cache, hyper-threading, AVX2 instruction support) thus the total working context number is 2 * 6 * 2 = 24. However, this configuration won't achieve the maximum throughput of the CPU since the operating system is involved. Empirically, 48 pthread will squeeze the CPU to 100 usage.

- **How does the master decide how to assign requests to workers?**

The assignment is handled in the "handle_client_request". We *monitor the number of tasks* (i.e., tellmenow, projectidea…) in each worker node. Each worker node has a preserved slot for prior tasks such as tellmenow and projectidea. The *maximum capability of a worker node for projectidea is 1*, otherwise the project idea might be sent to the same processor of a worker, which will break the cache since the project idea needs almost all L3 cahch.

Request handle

Every time when the "handle_client_request" is called, the master scan through the available slots from the beginning of the node, send the request as soon as it find the suitable slots. In this way, we will *push the tasks to the front worker node and leave the empty node at the end for scalability.*

Recycle

If the request is not assigned by "handle_client_request", the tasks are push into the queue or vip queue. Every time the the worker send the response back, the master node would traverse the vip queue first and normal queue second.

- **How does the master decide how to scale up or down the number of workers?**

Scale up

We monitor the **current number of worker**, and the **sum of tasks** in each worker node Those status sampled for each time step using the tick handler. When the queue is non zero or the number of task exceed by certain threshold, the master node will send a request to fire up the new worker.

Scale down

Since our request sending policy has already try to push the request tasks to the same workers. We only have to detect if currently there is a empty node and the size of queue is zero, then we can shut down the worker node immediately.

- **Did you use any other optimizations? Caching, etc.**

  We add a std:map at the master node to record all of the request/response pair, all of the request will check the cache first, if hit, the master directly return the response in the cache.

  We also preserve some slots in worker and master. for vip tasks and open a vip queue exclusive for latency-demeaned tasks.