LOAD BALANCING IN DISTRIBUTED SYSTEMS

Project by :

Sivov, Svyatoslav

Soni, Shikha

Vyshnyvetskyi, Oleg

Overview

The developing and fast growing computing world has caused a tremendous increase in the server client interactions taking place under various applications, where each of the interactions carry a large amount of tasks. Now, if all these tasks run on a single processor, the time taken to complete them will increase causing inefficiency. If these centralized units stop working, the applications will face a single point of failure. These large amount of tasks need to be distributed across various resources which include computers, processors, clusters or drivers. Implementing this kind of distribution will allow the system to work faster, process data faster and handle large number of tasks at the same time. Load balancing in itself explains the concept of balancing the load across the processors. It aims at optimizing the resource usage, maximizing the throughput and reducing the redundancy. In a load balancing system, there are three entities; worker, server and the client. The client orders tasks to the server, while the server distributes the tasks to the processors depending on their capacity and many more aspects. Sharing the amount of work is the main idea behind load balancing. Each user uses the applications and resources with different goals and objectives and hence management of resources is a difficult and complex task. Load balancing ensures that each processor gets work according to parameters which are predefined based on the system needs. One way to deal with the management of distributed systems is to have an unique decision maker that will lead the system to its optimum.

Problems faced and solutions:

· The basic problem that we faced was how to **analyze** the given tasks in order to distribute them such that each worker has been loaded equally.

· The initial consideration was to somehow figure out the **complexity** of a given task. But there was no basis or parameters which could efficiently find out how complex would the task be, or how will it load the worker.

· Unless and until we didn't know the tasks, it was **difficult to determine** the amount of **computations** and **execution time**. Since the server receives tremendous amount of work to be performed, the division of work had to have an algorithm, to figure out balancing of load.

· We **predefined** three tasks in order to let the client choose and assign to the worker with different amount of data. As the client assigns these tasks, we created **profiles** for each of them, which stored the information of how much **CPU load was consumed,** while how long did it take to execute the tasks. This information stored is used when later the same task is assigned.

· The CPU load parameter can be **predicted** **for future** and in turn assigned to the worker who has an unused CPU for that task. The system or rather the server builds a profile for each task for the future use. As the task comes in to be performed, its profile is referred and worker with the capability in terms of empty queue and available CPU is assigned.

Methodology:

Load balancing as described above aims at dividing the work equally across the workers. We create a client-server-worker infrastructure, where the clients assign tasks server accepts them and gives it to the most ideal worker, that is the one who has enough resources left to run the task. Our project has two clients, a general client that uses the system to get his work done, while the second one that runs multiple tasks on the system to test the load balancing algorithms, basically called as Benchmark testing. We have tried implementing 5 different types of algorithms and then tried observing the one which is the most efficient. The following are the algorithms:

1. Round Robin Fashion:

Here the workers are assigned tasks one after the other assuming that the first worker which was assigned the task first, will complete the task first with least execution time. After the last worker is given the task, the server again comes back to the first worker for the next task.

2. Min queue length:

As simple as the first one, in this algorithm, the worker as well as the server maintains a copy of the queue of the tasks that are assigned and have to be performed next as soon as any of the threads from the worker pool is become free after the task execution; that if the processor has returned the result. The queue of each of the worker is compared and the one with the least number of tasks in the queue, is given the next task.

3. Fitting CPU share:

This particular method the server is given the worker information in terms of the CPU load each task occupies. The CPU load for each worker is sent to the server after every task. This information is used to build a profile for the 3 tasks and thus make the server aware of the future load it will put on a system. When a task comes in, on the basis of the profile the server knows the CPU load it will occupy and thus gives it to the worker that has enough CPU to run the tasks.

4. Min CPU share :

This particular method the server is given the worker information in terms of the CPU load each task occupies. The CPU load for each worker is sent to the server after every task. This information is used to build a profile for the 3 tasks and thus make the server aware of the future load it will put on a system. When a task comes in, on the basis of the profile the server knows the CPU load it will occupy and thus gives it to the worker that has the lowest current CPU load.

5. Min CPU load queue balancer:

This is by far the most efficient way of balancing the tasks amongst the workers. This method calculates a total CPU share required by all of the tasks already in the queue of each worker (using CPU consumption profile of each task type). The task is given to the worker with the lowest score. Default value of 0.5 is used for the tasks with unknown CPU consumption profile. Each task reports its CPU consumption upon completion; therefore, the default value is used if none of the tasks of a particular type have been completed yet.

We used the tasks of 3 type to test our load balancing algorithms:

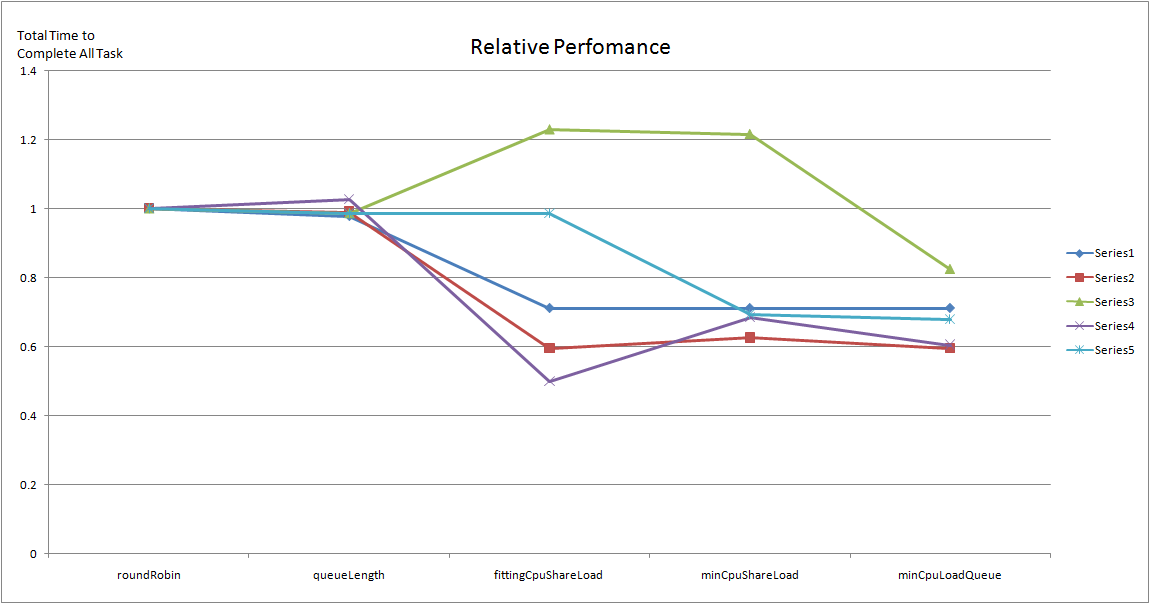
a. Password Brute Force (PasswordBruteforce2.java)

b. Timer (Time2.java)

c. Prime Calculator (PrimeCalculator2.java)

Outcomes:

We tested each of our algorithms against the same set or tasks. Our goal was to simulate a scenario where the server needs to handle a mix of IO-intensive and CPU-intensive tasks. We used round robin algorithm as a base of comparison. We found Min Queue Length algorithm performance was similar to the Round Robin in many cases. When the algorithms were put to use and analyzed, the min CPU load queue balance came out with great results and up to 40% improvement in the performance in some cases. While the other algorithms (3 and 4) gave similar performance to algorithm 5 for some datasets. However, algorithm 5 was the only one that produced the highest performance gains across all datasets used. We attribute this to the fact that algorithm 5 takes into consideration both the node-specific metrics (node’s current queue) as well as task-specific metrics (CPU intensity of the task’s type) . The below chart represents our findings with Round Robin algorithm total running time being 1:



Lessons learnt:

* “Intelligent” load balancer can dramatically improve performance of distributed system
* The nature of the tasks (specifically, resources used) affects performance of the system
* The system performance improves over time as the server obtains profiles of the tasks
* There is no “silver bullet” and an effective load balancer requires substantial amount of resources on the server (both memory and computation)
* Task profiling proved to be an effective technique in load balancing the tasks among the multiple workers
* Real-time performance reports from the workers proved to be an effective tool in optimal load balancing among multiple workers in the distributed system