ID2210 Project: Decentralized Resource Management

Andreas Hallberg, Zhengyang Zhao

May 30, 2014

1 Introduction

Resource manager is a relatively new infrastructure of datacenter. Its function is to allocate resources (CPUs, memory, network bandwidth, etc) to competing applications according some scheduling policies. Resource managers on large clusters can perform concurrent scheduling of millions of tasks per second, with low latency and high availability. Works of this scale is not achievable by centralized solutions. A recent trend has been to decentralize resource managers to make them scale to even larger clusters. Sparrow is one of the strategies to achieve this goal.
In the project, we built a decentralized resource scheduler that implements Sparrow, and improved the performance of the scheduler with Gradient overlay network implemented by TMan topology management. The evaluation shows that our implementation can scale to at least hundreds of nodes, and TMan dramatically improves the performance of Sparrow.

2 System Design

The Resource Manager consists of 3 main components: Cyclon, Sparrow and TMan.

2.1 Cyclon

Cyclon is a fundamental component in the Resource Manager. It periodically gossips with other peers to get random samples and provides the samples to Sparrow and TMan.

2.2 Sparrow

Sparrow is built upon Cyclon. It uses the random samples from Cyclon to discover available resources within a well-defined set of nodes. After

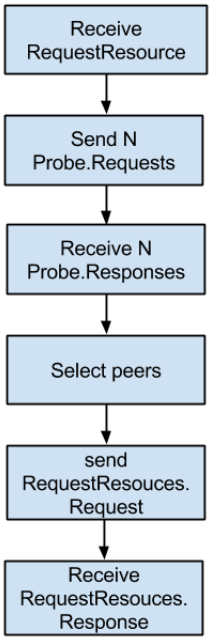

Figure 1: Resource Manager allocates resource request to Workers.

detecting appropriate peers with desirable resources, Sparrow allocates tasks to the peers.

2.3 TMan

TMan is also built on top of Cyclon. A node in the system uses TMan to periodically gossips with other peers to exchange and update its TMan view. The view is used to provide TMan samples to Sparrow.

3 Implementation

3.1 Sparrow

We used batch sampling in the implementation of Sparrow. When a resource manager (RM) receives a job that requests X machines, each with Y amount of memory and Z CPUs, the RM sends $N \cdot X$ probes to $N \cdot X$ peers selected from its current samples, where N is probe ratio. The procedure is shown in figure 1.
When a Worker receives an allocating request, it will immediately allocate resources for the request if there are enough free resources, otherwise put

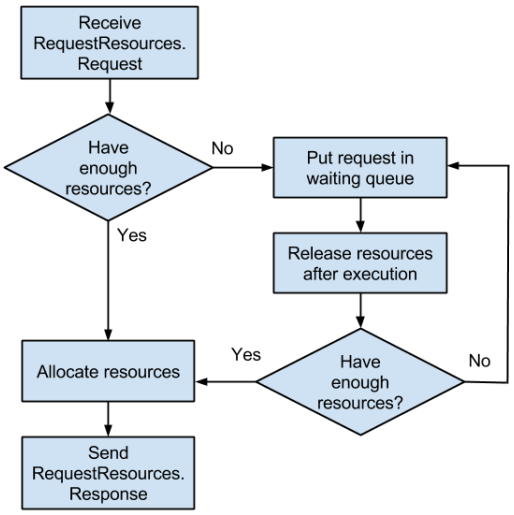

Figure 2: Worker allocates free resources to request.

the request to waiting queue until enough resources are released after some time. The procedure is shown in figure 2.

3.2 TMan

TMan is a protocol that can construct and maintain topology with the help of a ranking function. Its procedure is similar to Cyclon. The ranking function orders a set of nodes according to their desirability to be neighbors of a given node. We tried two kinds of strategies to implement the ranking function. One strategy is using Gradient preference function. The other strategy is just picking the peer with highest utility but not the upper closest one. From the test we found that Gradient has a better performance.
For the heterogeneity of peers, we do not maintain three topologies to track their utilities. Instead, We use a combination utility policy. A node's utility is defined as follows:

$$FreeCPU/MAX\_CPU \times FreeMemory/MAX\_Memory$$

4 Evaluation

4.1 Experimental setup

There are a lot of ways to measure the performance of the system, many different scenarios and configurations will yield different results. We have made the following decisions:

- The "time to find a resource", which is the main property we're measuring, is the time it takes for the resource manager to receive a request from the application, to the moment it receives an allocation response from another RMworker.
- To limit the time it takes to perform the tests, the number of nodes in most of the tests is set to 50, and the number of tasks is set to 3000. We have made some measurements with thousands of nodes, but the time they required was too much to do repeated tests, due to our testing hardware. For most of the tests, an allocation request is: 2 CPUs, 2000 MB memory, 10 seconds, 2 machines.
- For the majority of tests, the cyclon update period is 1000ms, and the TMan update period is 100ms. Requests are sent every 100 ms, and the Resource Manager is using 2 probes.
- Not all tests were run on the same machine, although all simulations for each particular test were.

To do the measurements, the Resource Manager will output the start- and end times of each request to a log file. After the program is finished, we are running a python script (test.py and request_times.py) which will gather all the start- and end times, sort them, and output the mean, median and 99th percentile. As well as a plot.

4.2 Results

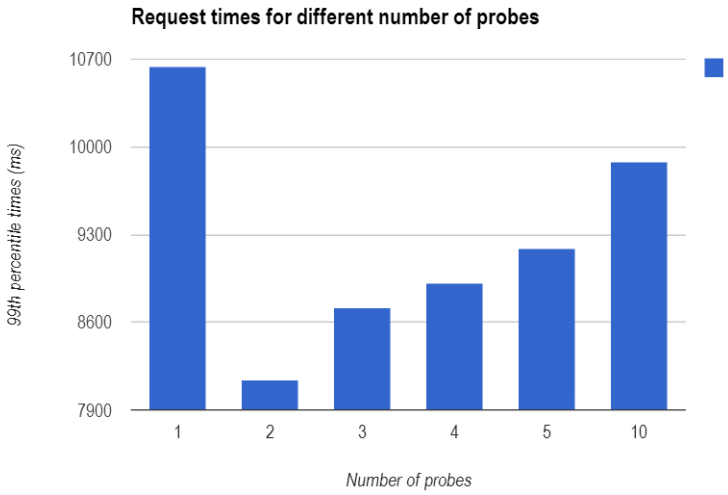

Figure 3: The effects of changing the number of probes for each task (Using Sparrow). Using only 2 probes seems to be the most efficient.

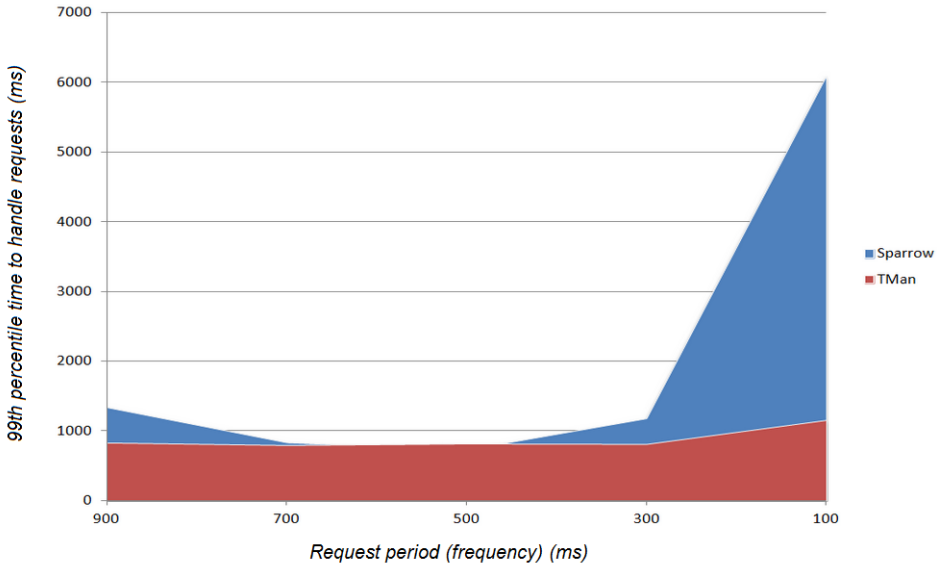

Figure 4: Displays what happens when we try to saturate the system by requesting resources faster than they can be released. The blue area shows the performance of our basic Sparrow implementation. The red area represents the improved implementation using TMan.

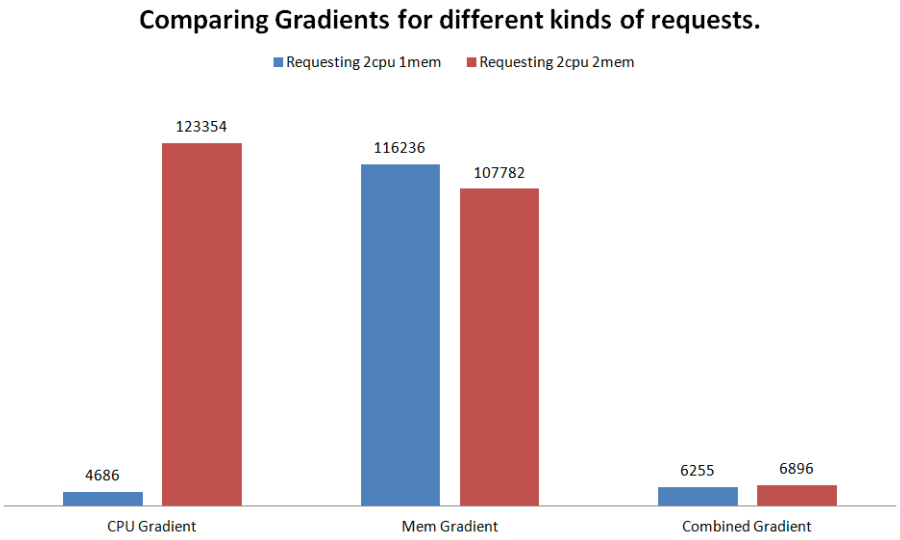

Figure 5: Using different kinds of gradients for different kinds of requests. The blue areas represent the 99th percentile time to handle requests which requires more CPU than memory. The red areas represent the time to handle requests which neither prefers CPU nor memory.

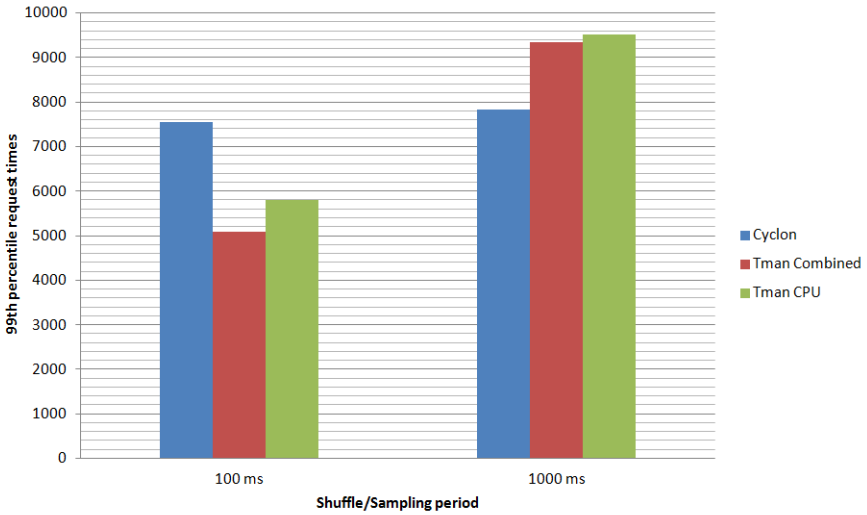

Figure 6: Comparing the performance using samples from TMan or Cyclon. It's notable that using TMan with a 100ms sampling period increases the performance, while TMan with a 1000ms sampling period decreases it.

5 Conclusions

We have managed to implement a decentralized resource management system with some basic scheduling features. The first implementation has most of the features seen in Sparrow, but not all of them. Some exceptions include late binding and failure handling. We have shown through simulations (Figure.3) that the scheduler can utilize the *power of two choices load balance technique*, to effectively distribute load among the nodes in the system.

We have also made improvements to Sparrow by building it on top of a gradient overlay network, instead of using random sampling. The Gradient overlay was implemented using TMan, which is bootstrapped by Cyclon. Our simulations (Figure 4, 6) indicate that the improved implementation actually makes a significant impact on performance, at least in the simulated scenarios we used for the evaluation.