Replicating Sparrow, a distributed low-latency scheduling tool

Term project progress report

Andrew Hollenbach, Akshay Srivatsa

Goal

For our project, we aim to replicate some of the novel features introduced by Sparrow, a low-latency scheduler, and present a live demo of a scheduler implementing the batch + late binding scheduling paradigm.

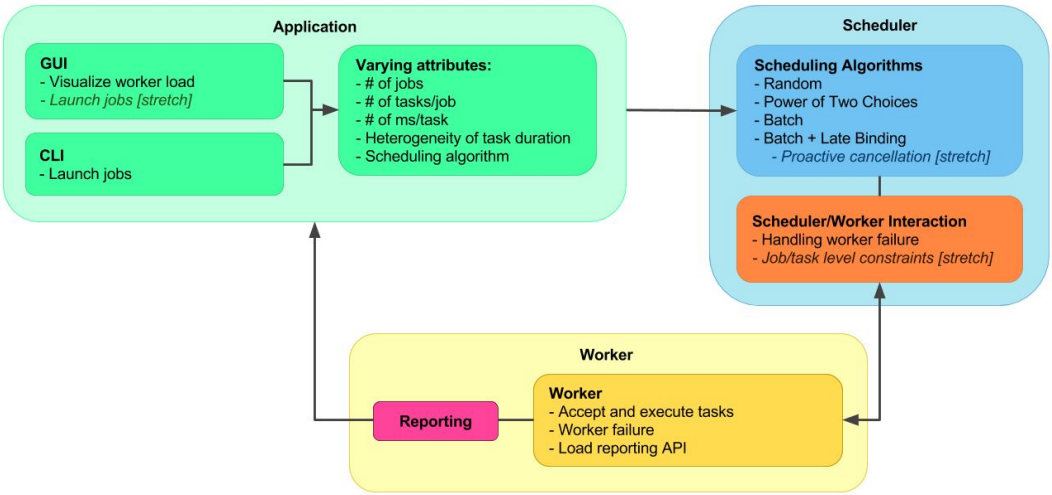
Challenges

Sparrow was originally built to solve several challenges, including achieving millisecond latency, quality placement, fault tolerance, and high throughput when scheduling sets of tasks from jobs in a large distributed system. The main focus was on reducing latency while still achieving quality placement (such that no worker nodes would become disproportionately overloaded), as many existing scheduling systems either focus on one or the other.

Proposed Solution

We propose a Python-based scheduler with three core modules, an application layer for interacting with the scheduler and viewing worker status, a scheduler which accepts jobs/tasks and distributes them, and a worker, which accepts tasks and executes them, and reports its status to an API. The following is a diagrammatic representation of the architecture of our solution and planned features. Our initial plan is to test the system on a network of at least 10 CS worker machines (hopefully more), although this may change as we get further along in development.

Design Diagram



Note**:** Items marked [Stretch] are stretch goals, which are unlikely to be implemented, but may be implemented in the event that all other features are completed first.

Progress

We have identified and understood the main challenges of the project and come up the main architecture for combating these problems. We have a clearly defined set of tasks to accomplish in order to successfully complete the project. We have kept stretch goals, which we will attempt to implement if we are able to meet our core goals. We have also allocated tasks to each individual member of the group.

Currently, the tasks that are in progress are creating a worker to accept and execute tasks, creating the load reporting API and implementing how the scheduler handles worker failure. We have not yet run the system full-cycle.

Demo

We aim to demonstrate the scheduling capabilities of our system by launching a series of jobs and tasks and we aim to provide a visualization of the status of worker load, as the scheduler assigns tasks.