Apache Mesos

-Naveen Thella

Abstract:

Apache Mesos is a cluster manager that provides efficient resource isolation and sharing across distributed applications or frameworks. Mesos is a open source software originally developed at the University of California at Berkeley. It sits between the application layer and the operating system and makes it easier to deploy and manage applications in large-scale clustered environments more efficiently. It can run many applications on a dynamically shared pool of nodes. Prominent users of Mesos include Twitter, Airbnb, MediaCrossing, Xogito and Categorize.

Node Abstract:

In a similar way that a PC operating system manages access to the resources on a desktop computer, Mesos ensures applications have access to the resources they need in a cluster. Instead of setting up numerous server clusters for different parts of an application, Mesos allows you to share a pool of servers that can all run different parts of your application without them interfering with each other and with the ability to dynamically allocate resources across the cluster as needed. That means, it could easily switch resources away from framework1 (for example, doing big-data analysis) and allocate them to framework2 (for example, a web server), if there is heavy network traffic. It also reduces a lot of the manual steps in deploying applications and can shift workloads around automatically to provide fault tolerance and keep utilization rates high.

Mesos Features:

* Fault-tolerant replicated master using ZooKeeper
* Scalability to thousands of nodes
* Isolation between tasks with Linux containers
* Multi-resource scheduling (memory and CPU aware)
* Java, Python and C++ APIs for developing new parallel applications
* Web UI for viewing cluster state

Functional Features of Apache Mesos:

**Create your Framework Scheduler**

You can write a framework scheduler in C, C++, Java/Scala, or Python. Your framework scheduler should inherit from the Scheduler class (see API below). Your scheduler should create a SchedulerDriver (which will mediate communication between your scheduler and the Mesos master) and then call SchedulerDriver.run().

## Working with Executors

### Using the Mesos Command Executor

Mesos provides a simple executor that can execute shell commands and Docker containers on behalf of the framework scheduler; enough functionality for a wide variety of framework requirements.

Any scheduler can make use of the Mesos command executor by filling in the optional CommandInfo member of the TaskInfo protobuf message.

### Creating a custom Framework Executor

If your framework has special requirements, you might want to provide your own Executor implementation. For example, you may not want a 1:1 relationship between tasks and processes.

Your framework executor must inherit from the Executor class. It must override the launchTask() method. You can use the $MESOS\_HOME environment variable inside of your executor to determine where Mesos is running from.

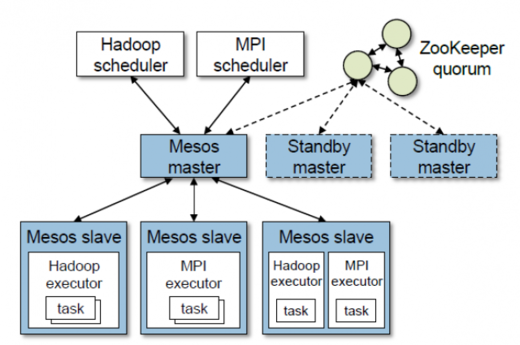
## Labels

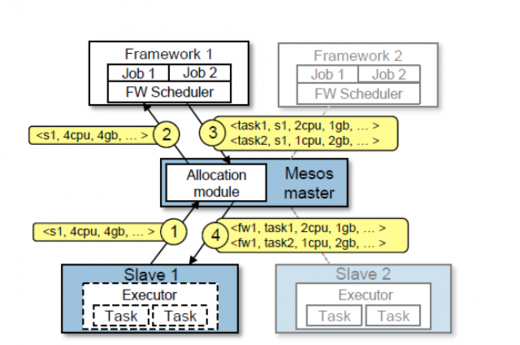
Labels can be found in the FrameworkInfo, TaskInfo, DiscoveryInfo and TaskStatus messages; framework and module writers can use Labels to tag and pass unstructured information around Mesos. Labels are free-form key-value pairs supplied by the framework scheduler or label decorator hooks. Below is the protobuf definitions of labels:

## Service discovery

When your framework registers an executor or launches a task, it can provide additional information for service discovery. This information is stored by the Mesos master along with other imporant information such as the slave currently running the task. A service discovery system can programmatically retrieve this information in order to set up DNS entries, configure proxies, or update any consistent store used for service discovery in a Mesos cluster that runs multiple frameworks and multiple tasks.

**Architecture**





Features that can be developed:

 Auto-scaling clusters (including auto reassignment of partitions) so that the resources (CPU, RAM, etc.) that brokers are using can be used elsewhere in known valleys of traffic.

 Rack-aware partition assignment for fault tolerance.

 Hooks so that producers and consumers can also be launched from the scheduler and managed with the cluster.

 Automated partition reassignment based on load and traffic.