Building Distributed Systems with Mesos

Abhishek Tiwari

Citation: *A. Tiwari*, "Building Distributed Systems with Mesos", Abhishek Tiwari, 2014. doi:10.59350/7awbn-2mm29

Published on: February 27, 2014

Apache Mesos is a popular open source cluster manager which enables building resource-efficient distributed systems. Mesos provides efficient dynamic resources isolation and sharing across multiple distributed applications such as Hadoop, Spark, Memcache, MySQL etc on a dynamic shared pool of resources nodes. This means with Mesos you can run any distributed application which requires clustered resources.

Single compute unit

Mesos allows multiple services to scale and utilise a shared pool of servers more efficiently. The key idea behind the Mesos is to turn your data centre into one very large computer. Think in this way, if you have 5 nodes with 8 CPU Cores and 32GB RAM and 10 nodes with 4 CPU Cores and 16GB RAM - using Mesos you can run them as one single large compute unit of 320GB RAM and 80 CPU Cores. With Mesos rather than running multiple applications on separate clusters , you are efficient distributing resources across multiple application on a single large compute unit.

Who is using Mesos?

Apache Mesos has been used in production by Twitter, AirBnB and many other companies for web-scale computing. Twitter is running several key services including analytics and ads. Similarly AirBnB uses Mesos to manage their big data infrastructure.

Similar type of systems Borg and its successor Omega are used by Google. Both Omega and Mesos let you run multiple distributed systems atop the same cluster of servers.

Mesos Internals

Mesos is leveraging features in modern kernels for resource isolation, prioritisation, limiting and accounting. This is normally done by cgroups in Linux, zones in Solaris. Mesos provide resources isolation for CPU, memory, I/O, file system , etc. It is also possible to use Linux containers but current isolation support for Linux container in Mesos is limited to only CPU and memory.

Mesos Architecture

Mesos architecture[1] consists of a Mesos Master that manages Mesos Slaves and frameworks that run tasks on Mesos Slaves. Master is in soft state[2] with one or more stand-by managed by ZooKeeper to

[1] Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center

[2] the master can reconstruct completely its internal state from the periodic messages it gets from the slaves, and from the framework schedulers

implement the failover mechanism.

A framework is an application running on top of Mesos. A Mesos framework normally uses Mesos APIs (C++, Python, Ruby, JVM, Go) exposed by Mesos kernel. Using these APIs a framework interacts with Mesos Master to accept and register the resources allocation on Mesos Slaves.

A framework consists of two key components,

- a scheduler accepts and registers resources from master, and
- an executor is a process launched on slave nodes to run the framework's tasks.

Some frameworks distributed in nature such as Hadoop, Spark, etc. Distributed frameworks normally employ distributed executors.

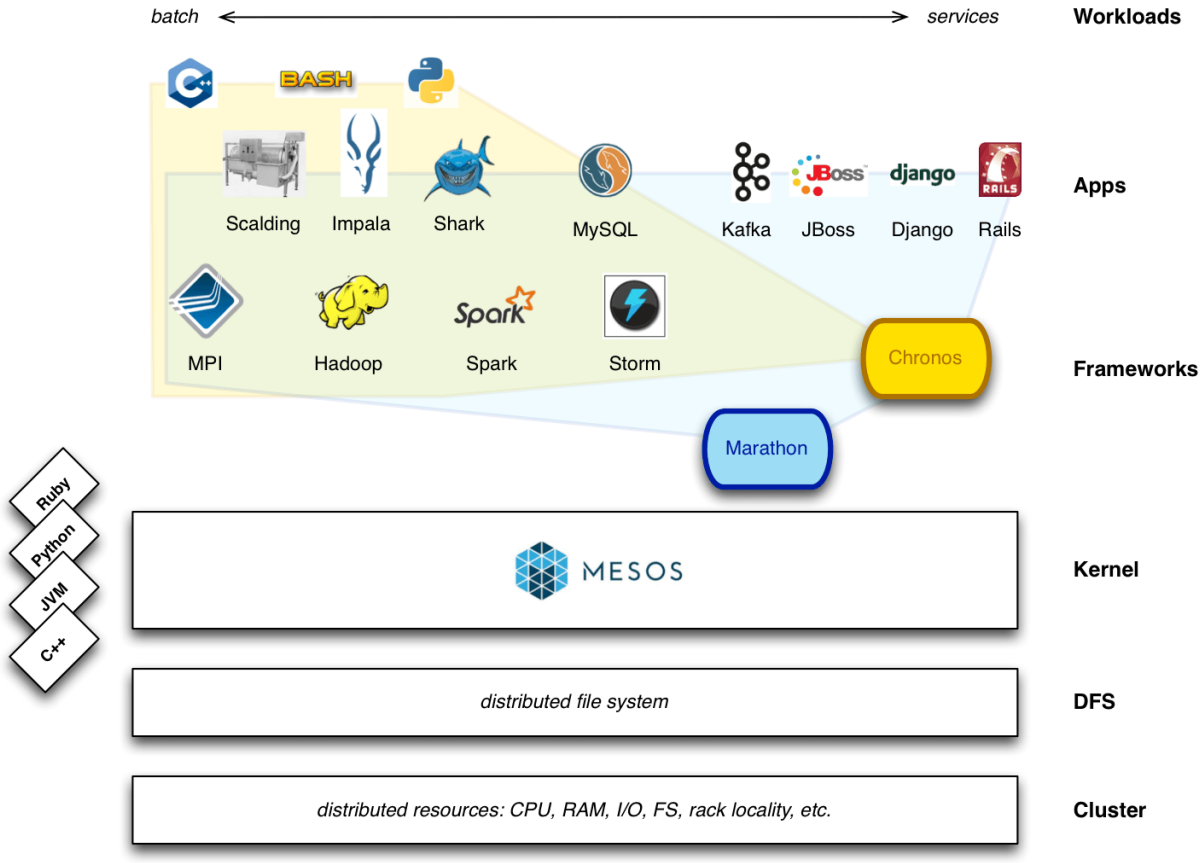

Figure 1: Mesos Architecture and key components

How Mesos Works?

1. When a Mesos Slave has free resource, it reports to Mesos Master about availability.

2. Using allocation policy, Mesos Master determines how many resources are offered to each framework.
3. Then Master send offers and frameworks' schedulers select which of the offered resources to accept.
4. When a frameworks accepts offered resources, it passes to Mesos a description of the tasks it wants to run on them.
5. In turn, Mesos Master sends the tasks to the Slave, which allocates appropriate resources to the framework's executor.
6. Finally, framework's executor launches the tasks.

Mesos Frameworks

Number of Mesos frameworks is growing very rapidly. Here is a list of popular frameworks,

Continuous Integration: Jenkins, GitLab

Big Data: Hadoop, Spark, Storm, Kafka, Cassandra, Hypertable, MPI

Python workloads: DPark, Exelixi

Meta-Frameworks / HA Services: Aurora, Marathon

Distributed Cron: Chronos

Containers: Docker

There are two notable frameworks currently available to support batch processing and long-running services: Chronos (Unix cron equivalent) and Marathon (Unix init.d equivalent). Marathon is a meta framework and it is used by most of other frameworks and application including Chronos.

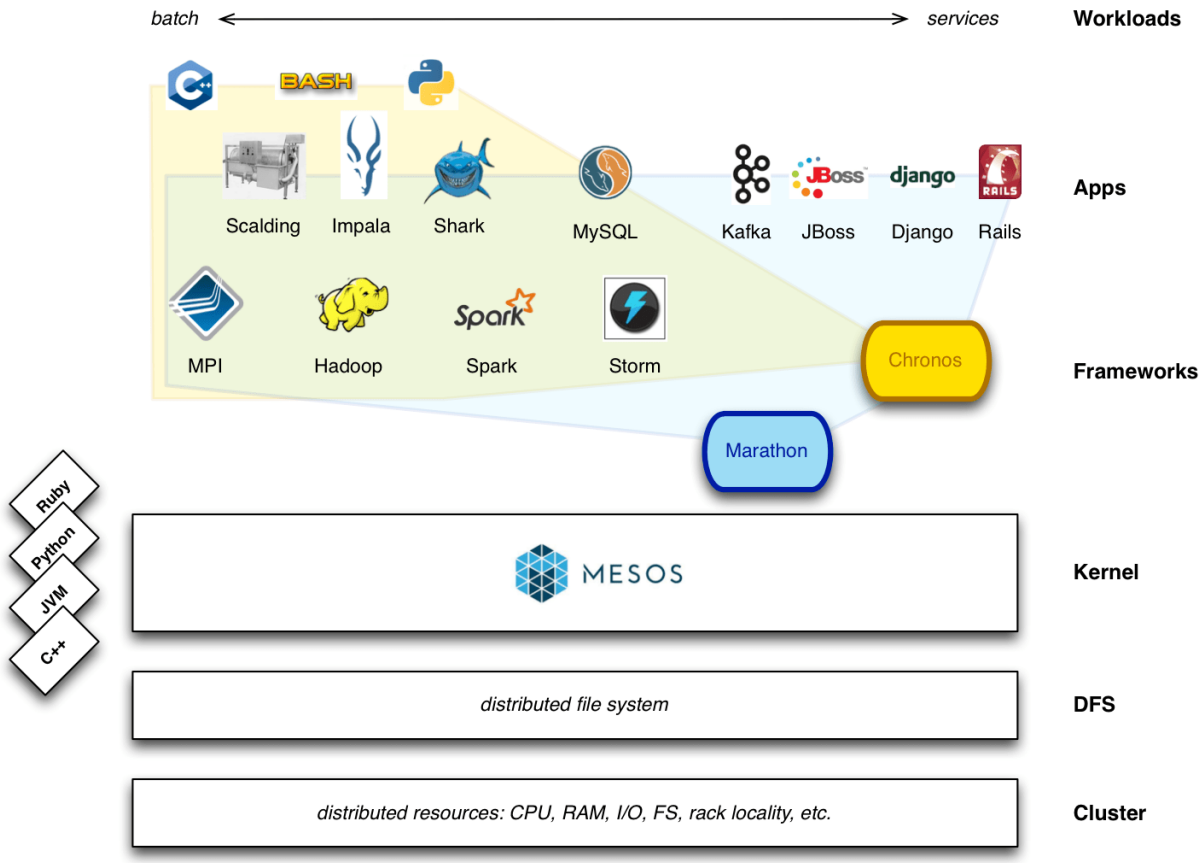

Figure 2: Mesos architecture supports both batch jobs and long-running services/apps. Image Credits Typesafe Blog.

Marathon can run web applications like Rails[3], Django and Play[4]. Marathon provides a REST API for starting, stopping, and scaling these applications (scale-out and scale-in).

Chronos is a distributed and fault-tolerant cron/job scheduler. Chronos supports custom Mesos executors as well as the default command executor `sh`. Using a RESTful JSON API over HTTP frameworks and applications can communicate with Chronos.

Writing your own framework

Writing your own framework on top of Mesos is quite easy. First you create your scheduler in C, C++, Java/Scala, or Python by inheriting from the corresponding `Scheduler` class.

```
import mesos
```

[3] Run Ruby on Rails on Mesos
[4] Run Play on Mesos

```
# Example using Python
class MyScheduler(mesos.Scheduler):
    # Override Scheduler Functions like resourceOffers, etc.
```

```
import org.apache.mesos.MesosSchedulerDriver;
import org.apache.mesos.Protos;
import org.apache.mesos.Protos.*;
import org.apache.mesos.Protos.TaskID;
import org.apache.mesos.Scheduler;
import org.apache.mesos.SchedulerDriver;

public class MyScheduler implements Scheduler {
    // Override Scheduler Functions like resourceOffers, etc.
}
```

Then you write your framework executor by inheriting from the `Executor` class.

```
import mesos
# Example using Python
class MyExecutor(mesos.Executor):
    # Override Executor Functions such as launchTask, etc.
```

```
import org.apache.mesos.Executor;
import org.apache.mesos.ExecutorDriver;
import org.apache.mesos.MesosExecutorDriver;
import org.apache.mesos.Protos.Environment.Variable;
import org.apache.mesos.Protos.*;
import org.apache.mesos.Protos.TaskID;
import org.apache.mesos.Protos.TaskStatus;

// Example using Java
public class MyExecutor implements Executor {
    // Override Executor Functions such as launchTask, etc.
}
```

Finally install your framework on Mesos cluster which requires placing your framework somewhere that all slaves on the cluster can get it from. If you are running HDFS, you can put your framework executor into HDFS.

Running Distributed Systems On Mesos

Apache Mesos can be used as an SDK for building fault-tolerant distributed frameworks on top of it[5]. You can port existing distributed systems or new distributed applications on Mesos using a custom framework wrapper in less than 100-300 lines (approach described in previous section) without involving any networking related code.

[5] Apache Mesos as an SDK for Building Distributed Frameworks

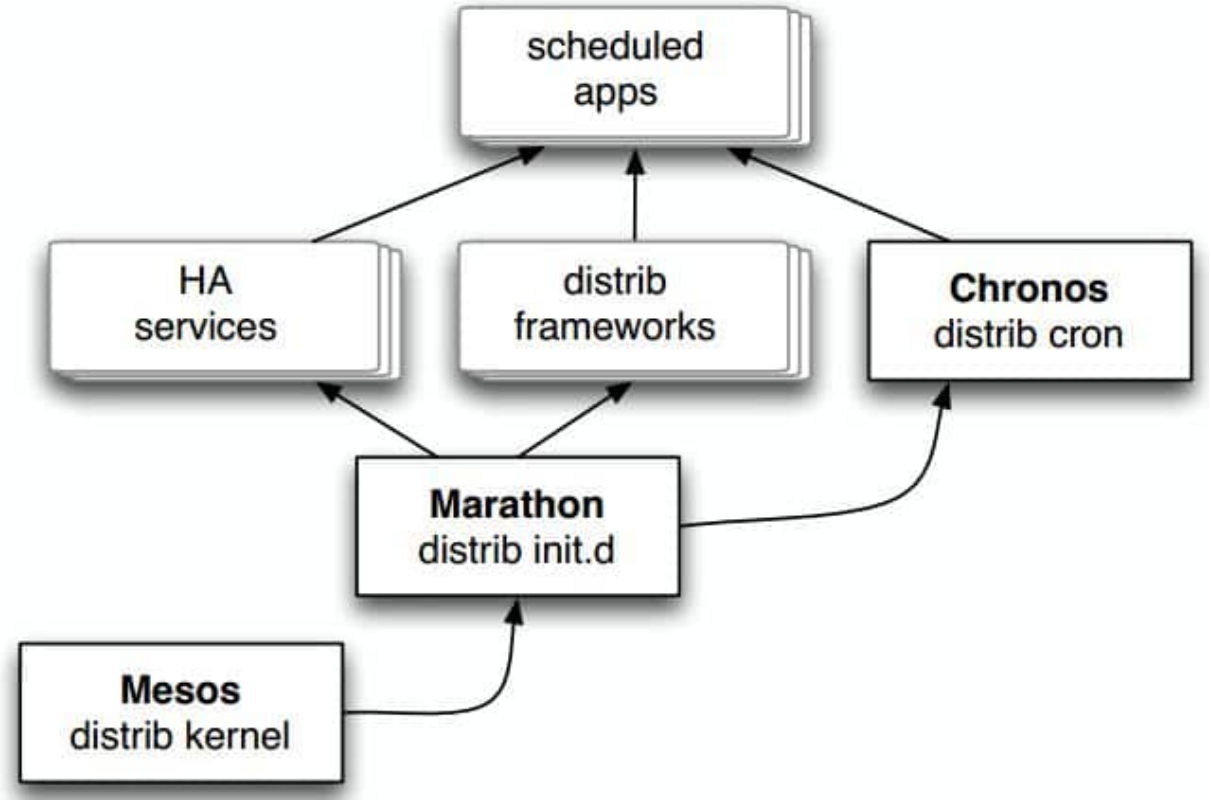

Figure 3: Mesos and Distributed Frameworks as Graph. Image Credits Paco Nathan.

When porting existing distributed system like Hadoop or Storm, challenging bit seems to be the mapping the distributed system specific logic (Distributed system graph) on `Scheduler` and `Executor` (Mesos graph). For instance when running Hadoop on Mesos [6] [7],

- Hadoop'sJobTracker represents both a scheduler and a long-running process
- Mesos Scheduler is a thin layer top of Hadoop Scheduler.
- Hadoop TaskTrackers are executors.
- Hadoop Job is collection of map and reduce tasks.
- Hadoop Task is one unit of work for a Job (map or reduce)
- Hadoop Slot is a task executor (map or reduce)
- Hadoop JobTracker launches TaskTrackers for each Job (fixed or variable slot policy)
- Hadoop Task scheduling is left to the underlying scheduler (i.e., Hadoop FairScheduler)

Similarly for Storm, we have to map Nimbus (JobTracker of Storm) and the Supervisors (TaskTrackers of Storm) on `Scheduler` and `Executor` (Mesos graph).

[6] Learning Apache Mesos
[7] Hadoop on-mesos