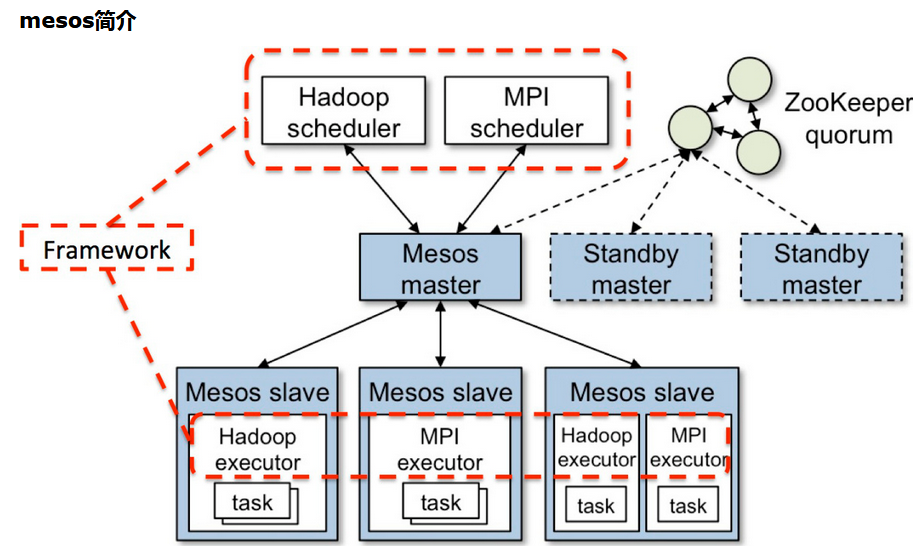
**Mesos**

## 一、基础知识

**1 简单介绍**

Apache Mesos为了简化设计，也是采用了master/slave结构，为了解决master单点故障，将master做得尽可能地轻量级，其上面所有的元数据可以通过各个slave重新注册而进行重构，故很容易通过zookeeper解决该单点故障问题。



**2 yarn和mesos不同点**

1）、最大的不同点在于他们所采用的scheduler：mesos让framework决定mesos提供的这个资源是否适合该job，从而接受或者拒绝这个资源。而对于yarn来说，决定权在于yarn，是yarn本身（自行替应用程序作主）决定这个资源是否适合该job，对于各种各样的应用程序来说或许这就是个错误的决定（这就是现代人为什么拒绝父母之命媒妁之言而选择自由婚姻的缘故吧）。所以从scaling的角度来说，mesos更scalable。

2）、yarn是MapReduce进化的产物，yarn从诞生之日起就是为hadoopjobs管理资源的（yarn也开始朝着mesos涉及的领域进军），yarn只为hadoop jobs提供了一个static partitioning。而mesos的设计目标是为各个框架（hadoop、spark、web services等）提供dynamical partitioning，让各个集群框架共用数据中心机器。

**3 总体架构**

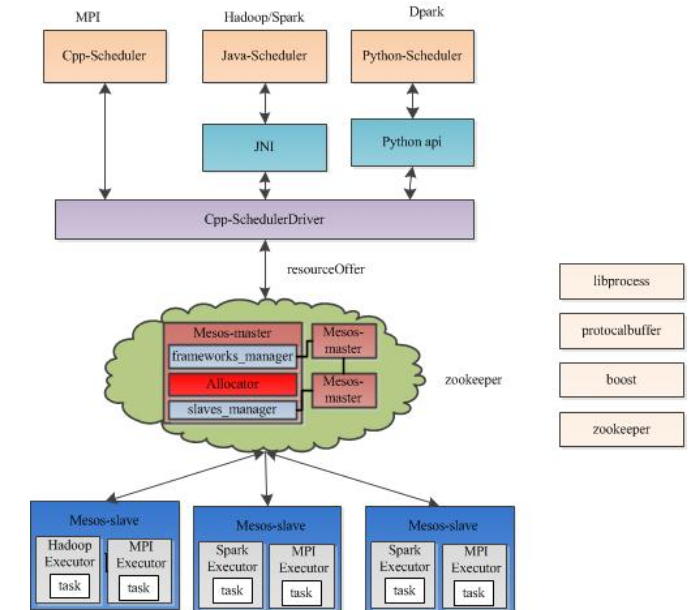
**Apache Mesos由四个组件组成，分别是Mesos-master，mesos-slave，framework和executor。**

（1）Mesos-master是整个系统的核心，负责管理接入mesos的各个framework（由frameworks\_manager管理）和slave（由slaves\_manager管理），并将slave上的资源按照某种策略分配给framework（由独立插拔模块Allocator管理）。

（2）Mesos-slave负责接收并执行来自mesos-master的命令、管理节点上的mesos-task，并为各个task分配资源。mesos-slave将自己的资源量发送给mesos-master，由mesos-master中的Allocator模块决定将资源分配给哪个framework，当前考虑的资源有CPU和内存两种，也就是说，mesos-slave会将CPU个数和内存量发送给mesos-master，而用户提交作业时，需要指定每个任务需要的CPU个数和内存量，这样，当任务运行时，mesos-slave会将任务放到包含固定资源的linux container中运行，以达到资源隔离的效果。很明显，master存在单点故障问题，为此，mesos采用了zookeeper解决该问题。

（3）Framework是指外部的计算框架，如Hadoop，Mesos等，这些计算框架可通过注册的方式接入mesos，以便mesos进行统一管理和资源分配。Mesos要求可接入的框架必须有一个调度器模块，该调度器负责框架内部的任务调度。当一个framework想要接入mesos时，需要修改自己的调度器，以便向mesos注册，并获取mesos分配给自己的资源， 这样再由自己的调度器将这些资源分配给框架中的任务，也就是说，整个mesos系统采用了双层调度框架：第一层，由mesos将资源分配给框架；第二层，框架自己的调度器将资源分配给自己内部的任务。当前Mesos支持三种语言编写的调度器，分别是C++，java和python，为了向各种调度器提供统一的接入方式，Mesos内部采用C++实现了一个MesosSchedulerDriver（调度器驱动器），framework的调度器可调用该driver中的接口与Mesos-master交互，完成一系列功能（如注册，资源分配等）。

（4）Executor主要用于启动框架内部的task。由于不同的框架，启动task的接口或者方式不同，当一个新的框架要接入mesos时，需要编写一个executor，告诉mesos如何启动该框架中的task。为了向各种框架提供统一的执行器编写方式，Mesos内部采用C++实现了一个MesosExecutorDiver（执行器驱动器），framework可通过该驱动器的相关接口告诉mesos启动task的方法。



**Scheduler**

public abstract interface org.apache.mesos.Scheduler {

// 向SchedulerDriver注册Scheduler

public abstract void registered(org.apache.mesos.SchedulerDriver arg0, org.apache.mesos.Protos.FrameworkID arg1, org.apache.mesos.Protos.MasterInfo arg2);

// 注册成功后

public abstract void reregistered(org.apache.mesos.SchedulerDriver arg0, org.apache.mesos.Protos.MasterInfo arg1);

// 处理来自各slave的Resource offer，以决定将task分配到哪个slave上

public abstract void resourceOffers(org.apache.mesos.SchedulerDriver arg0, java.util.List arg1);

// resource offer撤回

public abstract void offerRescinded(org.apache.mesos.SchedulerDriver arg0, org.apache.mesos.Protos.OfferID arg1);

// 某个task的状态发生改变

public abstract void statusUpdate(org.apache.mesos.SchedulerDriver arg0, org.apache.mesos.Protos.TaskStatus arg1);

// 收到来自某个Executor的消息

public abstract void frameworkMessage(org.apache.mesos.SchedulerDriver arg0, org.apache.mesos.Protos.ExecutorID arg1, org.apache.mesos.Protos.SlaveID arg2, byte[] arg3);

// "失联"后

public abstract void disconnected(org.apache.mesos.SchedulerDriver arg0);

// 某个slave"失联"后

public abstract void slaveLost(org.apache.mesos.SchedulerDriver arg0, org.apache.mesos.Protos.SlaveID arg1);

// 某个executor"失联"后

public abstract void executorLost(org.apache.mesos.SchedulerDriver arg0, org.apache.mesos.Protos.ExecutorID arg1, org.apache.mesos.Protos.SlaveID arg2, int arg3);

// 运行过程中发生错误时

public abstract void error(org.apache.mesos.SchedulerDriver arg0, java.lang.String arg1);

}

**Executor**

public abstract interface org.apache.mesos.Executor {

// 向ExecutorDriver注册Executor

public abstract void registered(org.apache.mesos.ExecutorDriver arg0, org.apache.mesos.Protos.ExecutorInfo arg1, org.apache.mesos.Protos.FrameworkInfo arg2, org.apache.mesos.Protos.SlaveInfo arg3);

// 估计是注册成功后

public abstract void reregistered(org.apache.mesos.ExecutorDriver arg0, org.apache.mesos.Protos.SlaveInfo arg1);

// “失联”后

public abstract void disconnected(org.apache.mesos.ExecutorDriver arg0);

// 运行任务

public abstract void launchTask(org.apache.mesos.ExecutorDriver arg0, org.apache.mesos.Protos.TaskInfo arg1);

// 收到scheduler杀死task的命令

public abstract void killTask(org.apache.mesos.ExecutorDriver arg0, org.apache.mesos.Protos.TaskID arg1);

// 接受scheduler发送的消息

public abstract void frameworkMessage(org.apache.mesos.ExecutorDriver arg0, byte[] arg1);

// 收到scheduler shutdown的命令

public abstract void shutdown(org.apache.mesos.ExecutorDriver arg0);

// 运行过程中出错

public abstract void error(org.apache.mesos.ExecutorDriver arg0, java.lang.String arg1);

}