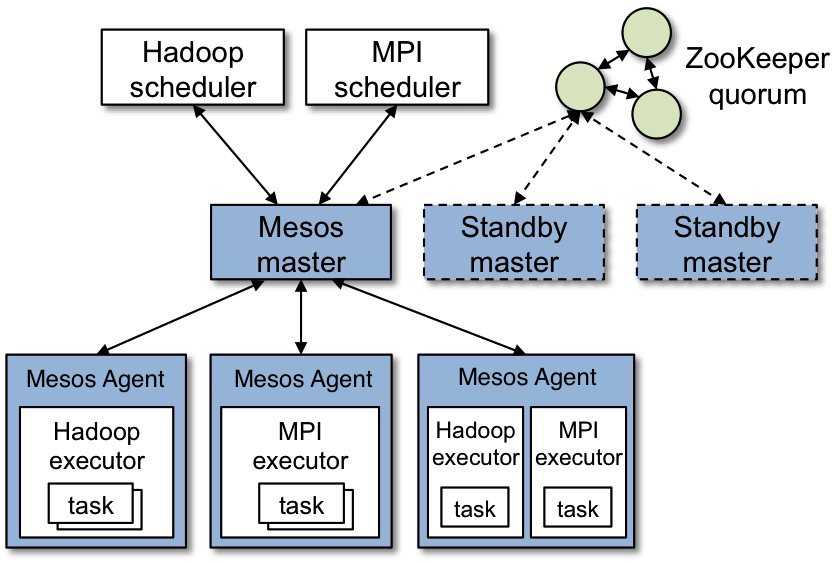
作业二

学号：1300013022

姓名：武守北

一、用自己的语言描述Mesos的组成结构，指出它们在源码中的具体位置，简单描述一下它们的工作流程：

Mesos的组成结构图：



Mesos架构主要由四部分组成：master、slave(agent)、Framework的scheduler和executor。外加一个故障恢复的zookeeper。

**1、master**

master 代码位于 /src/master 目录下。

master 在 Mesos 架构中居于核心位置。master 负责管理每一个集群节点上运行的 agent 守护进程，以及 agent 上执行具体任务的 frameworks. master 进行对 agent 的管理的重要手段是统筹管理集群资源。master 利用 resource offer 机制实现这一点。resource offer 记录每个节点上有哪些处于空闲状态的资源（包括 CPU、内存、磁盘、网络等），资源列表的维护和更新由 master 完成。master 会根据这一列表中的信息、利用某种分配策略决定下一步为各个节点分配哪些资源。master 的资源分配机制可能需要根据不同运行状况和节点需求而有所改变。为了适应多种分配机制的需求，master 使用了一种模块化的体系结构来使得添加新的分配策略变得更为容易。在实际应用中，常常会有多个 master 同时存在的情况，它们互为备份，防止由于某一个 master 终止运行造成整个系统意外停止。Mesos 使用 Zookeeper 管理多个 master，并选择其中一个作为主节点执行各项功能。

**2、agent**

agent 在一些旧的文档中被称为slave，代码位于 /src/sched 和 /src/slave 目录下。agent 一方面向处于运行状态的 master 报告目前节点上的空闲资源，从而更新 resource offer 的资源列表，另一方面接收 master 关于分配资源和执行任务的指令，将资源分配给具体 framework 的 executor.

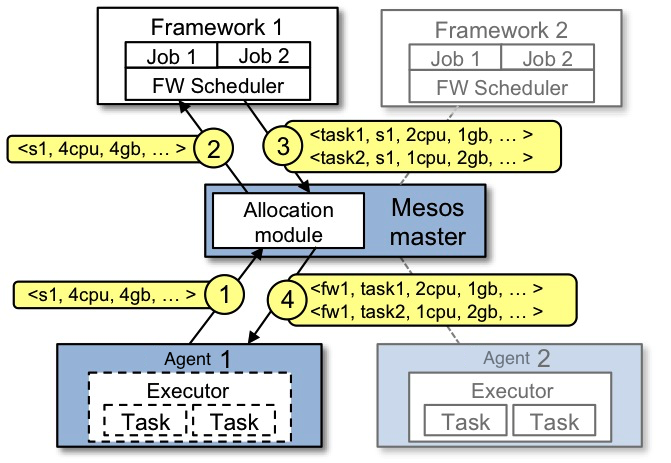
**3、frameworks**

frameworks 分为 scheduler 和 executor 两部分，负责具体执行某一个任务时的资源调度和执行工作，代码分别位于 /src/scheduler 和 /src/executor 目录下。 scheduler 负责与 master 交流目前 framework 运行需要哪些资源，以及 master 能够提供哪些资源。scheduler 向 master 注册框架信息后，master 会不断告知 scheduler 目前有哪些资源可用，由 scheduler 决定是否接受。若是，scheduler 还需要在接收资源并在节点内部进行分配后，告知 master 各项资源的具体分配信息。 executor 负责在接收资源后具体执行任务。新的框架加入集群时也需要 executor 启动框架。

**4、Zookeeper**

Zookeeper 是一个 Apache 顶级项目。它是一个针对大型应用的数据管理、提供应用程序协调服务的分布式服务框架，提供的功能包括：配置维护、统一命名服务、状态同步服务、集群管理等。在生产环境中， Zookeeper 能够通过同时监控多个 master 在前台或后台运行或挂起，为 Mesos 提供一致性服务。代码位于 /src/zookeeper 目录下。

下图是一个简单的Mesos工作流程：



1、当出现以下几种事件中的一种时，会触发资源分配行为：新框架注册、框架注销、增加节点、出现空闲资源等；

2、Mesos Master中的Allocator模块为某个框架分配资源，并将资源封装到ResourceOffersMessage（Protocal Buffer Message）中，通过网络传输给SchedulerProcess；

3、SchedulerProcess调用用户编写的Scheduler中的resourceOffers函数（不能版本可能有变动），告之有新资源可用；

4、用户的Scheduler调用MesosSchedulerDriver中的launchTasks()函数，告之将要启动的任务；

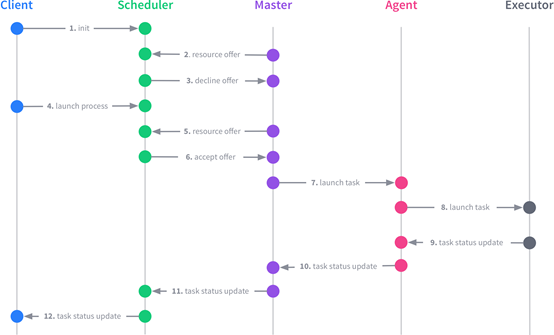
5、SchedulerProcess将待启动的任务封装到LaunchTasksMessage（Protocal Buffer Message）中，通过网络传输给Mesos Master；

6、Mesos Master将待启动的任务封装成RunTaskMessage发送给各个Mesos Slave；

7、Mesos Slave收到RunTaskMessage消息后，将之进一步发送给对应的ExecutorProcess；

8、ExecutorProcess收到消息后，进行资源本地化，并准备任务运行环境，最终调用用户编写的Executor中的launchTask启动任务（如果Executor尚未启动，则先要启动Executor。

总的来说Mesos是一个二级调度机制，第一级是向框架提供总的资源，第二级由框架自身进行二次调度然后将结果返回给Mesos。



Master部分在 /path/to/mesos/src/master 下，main.cpp是入口程序，其内部会生成一个master对象，该对象开始监听信息。

Slave部分在 /path/to/mesos/src/slave 下，同样main.cpp是slave的入口程序，在处理完若干参数后会生成一个slave对象，该对象开始监听信息并发送状态给master对象。

MesosSchedulerDriver的启动模块在 /path/to/mesos/src/sched/sched.cpp下，它创建一个scheduler的进程等待framework通过http的方式来注册，相当于给外部framework提供了一个接口。

MesosExecutorDriver的启动模块在 /path/to/mesos/src/exec/exec.cpp下，同理它创建了一个executor进程等待framework通过http的方式注册。

二、用自己的语言描述框架（如Spark On Mesos）在Mesos上的运行过程，并与在传统操作系统上运行程序进行对比：



将上图的Cluster Manager 替换为Mesos，就得到了Spark on Mesos的结构图。其工作流程为：

1、Spark 启动后像Mesos注册，成为Mesos Framework中的一个；

2、用户向Driver Program提交Task；

3、Mesos对Worker Nodes进行调度，来决定在哪个Slave(Worker Nodes)上运行哪一个Task；

4、Slave上的executor接受SparkContext，从而运行特定的task；

与传统操作系统的对比：

相同点：

1、Mesos和传统操作系统都提供了一个基本性的功能：对硬件资源（CPU、内存）进行抽象，将硬件接口等底层实现进行屏蔽，从而对上层的结构（如应用）提供资源的抽象与分配管理。

2、Mesos和传统操作系统都实现了隔离功能：传统操作系统实现了进程与进程之间的隔离；而Mesos实现了框架与框架之间、任务与任务之间资源的隔离。

不同点：

1、Mesos管理的是集群中的资源，一般拥有多台设备；而传统操作系统一般针对单台设备。

2、Mesos对IO设备的速度要求高于传统操作系统。为了调度与协调不同slave之间的工作，slave节点之间、master与slave节点之间的通信速度有较高要求。

3、Mesos分配资源时，先对某个框架中的任务提供可用资源的数量，任务可以选择性的接受资源，与选择性的拒绝资源，然后Mesos再返回被接受的资源。在传统操作系统中一般是进程要求一定数目的资源，操作系统就相应地分配所需要的资源。

三、叙述master和slave的初始化过程：

1、Master

Master的入口是mesos-1.1.0/src/master/main.cpp

开一个master::Flags记录flags。

flags用到了stout库，主要是其中的option和flag。

flags涉及到了LIBPROCESS\_IP等环境变量。

进行libprocess库的process的初始化。

进行日志logging的初始化。

将warning写入日志中。

新建一个VersionProcess线程用于返回http请求的版本号。

初始化防火墙。

初始化模块。

初始化hooks（暂时不知道有什么作用）。

新建一个分配器的实例。

新建用于state的空间。

创建State实例。

创建Registrar实例。

创建MasterContender实例。

创建MasterDetector实例。

初始化Authorizer相关内容。

初始化SlaveRemovalLimiter相关内容。

创建master实例，创建master线程以监听请求。

等待master结束。

垃圾回收。

2、Slave

Slave的入口是mesos-1.1.0/src/slave/main.cpp

开一个slave::Flags进行flags的chuli。

向Master提供资源，每隔"disk\_watch\_interval"的时间就调用一次Slave::checkDiskUsage。

输出版本号。

利用libprocess生成一个slave的ID。

进行libprocess库的process的初始化。

进行日志logging的初始化。

将warning写入日志中。

新建一个VersionProcess线程用于返回http请求的版本号。

初始化防火墙。

初始化模块。

创建containerizer。

创建detector。

Authorizer管理。

创建gc、StatusUpdateManager、ResourceEstimator。

创建slave实例，创建slave线程。

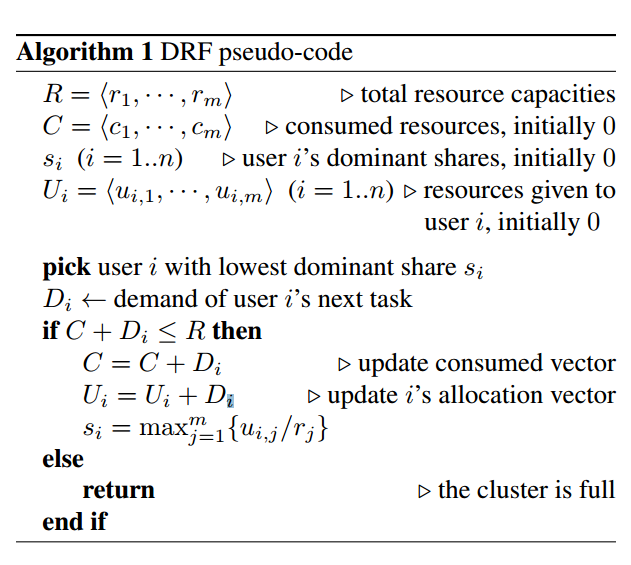
等待slave结束。

垃圾回收。

四、查找资料，简述Mesos的资源调度算法，指出在源代码中的具体位置并阅读，说说你对它的看法：

Mesos使用的是Dominant Resource Fairness算法(DRF)。

其目标是确保每一个用户，即Mesos中的Framework能够接收到其最需资源的公平份额。首先定义主导资源(domainant resource)和主导份额。 主导资源为一个Framework的某个资源，Framework所需除以Master所有，大于其它所有资源。 Framework所需除以Master所有，即为这个Framework的主导份额。DRF算法会解方程，尽量让每一个Framework的主导份额相等，除非某个Framework不需要那么多的资源。如果是带权重的DRF算法，只需将权重归一化再执行DRF算法即可。核心算法如图所示：



算法的主要部分位于 /src/master/allocator/mesos/hierarchical.cpp中的HierarchicallAllocatorProcess::allocate()；

五、写一个完成简单工作的框架(语言自选，需要同时实现scheduler和executor)并在Mesos上运行，在报告中对源码进行说明并附上源码。

主要内容包括两个文件：

scheduler.py：GetSumScheduler类的定义和整个framework的入口函数

executor.py：GetSumExecutor类的定义

Scheduler部分代码执行过程：

1、初始化exeuctor信息，包括名称、执行路径和资源信息等。

2、初始化framework信息，包括名称、用户信息和Host。

3、初始化scheduler驱动，这个类封装在pymesos包中，使程序员摆脱了底层相关的事情，直接调用API即可。

4、增加信号处理函数，Ctrl + C。

5、开启运行的线程，然后用while等待线程。

6、进入GetSumScheduler类，其构造函数首先会读文件，然后将数据平均分成10份，然后创建10个task，每个task执行一份数据。定义一个frameworkMessage方法来接收从executor执行回来的结果。

7、在updateStatus中判断是否10个tasks的任务执行完毕，若完毕则可以结束scheduler。

Executor部分代码执行过程：

1、先发送当前的状态信息，初始化时状态为RUNNING。

2、接着执行核心的部分。

3、执行结束后会再发送FINISHED的信息。 （注意如果executor代码发生异常，则可能会卡在RUNNING和FINISHED之间，这里需要再处理）

运行结果如下：

