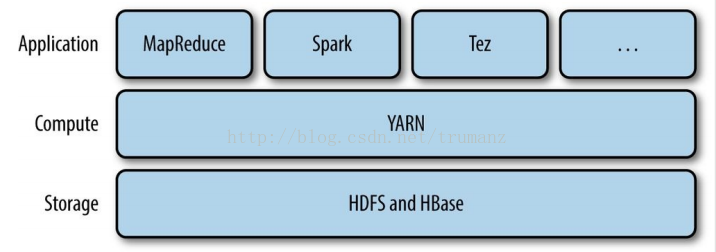
**通过编写一个YARN Application 来了解Hadoop YARN**

YARN是Hadoop中的一个cluster资源管理系统。 MapReduce 就是泡在YARN 之上的，其关系如下图， MapReduce 通过YARN 在cluster内申请内存和cpu资源，YARN执行MapReduce Task.



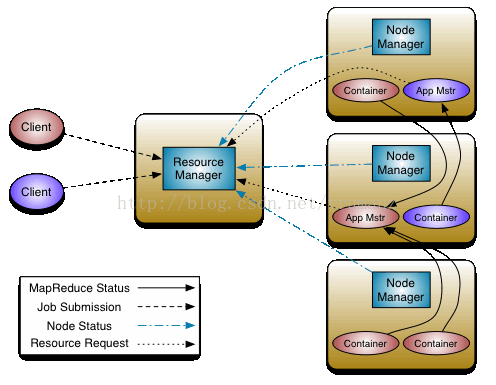
最近学习了YARN，

从部署的角度YARN支持三种调度器:FIFO, Capacity 和 Fair。

从开发应用程序的角度，实现一个YAN应用程序，需要编写三个程序1. Client  2. Application Master 3. Container Application， 也就是下图中椭圆形的三种程序， 对于MapReduce 其提供了Application Master 和  Client 一些丰富的Library 并且提供了 Container Application 的基础框架，所以我们基本上只需要做配置和编写Container Application 就可以。

这里将以一个简单的代码例子（不考虑错误）来看YARN提供的功能

[https://github.com/trumanz/hadoop\_the\_definitive\_guide/tree/master/ch04-yarn/yarnExample](https://github.com/trumanz/hadoop_the_definitive_guide/tree/master/ch04-yarn/yarnExample" \t "https://blog.csdn.net/trumanz/article/details/_blank)



# ****YARN  Client编写****

****1. 创建一个Application****

        YarnClientApplication app = yarnClient.createApplication();

    2.  设置Application的名字

         app.getApplicationSubmissionContext().setApplicationName( "truman.ApplicationMaster");

****3.  设置Applicaton 的 内存 和 cpu 需求 以及 优先级和 Queue 信息， YARN将根据这些信息来调度App Master****

         app.getApplicationSubmissionContext().setResource(Resource.newInstance(100, 1));

         app.getApplicationSubmissionContext().setPriority(Priority.newInstance(0));  
         app.getApplicationSubmissionContext().setQueue("default");

****4.  设置ContainerLaunchContext, 这一步， amContainer中包含了App Master 执行需要的资源文件，环境变量 和 启动命令，****这里将资源文件上传到了HDFS，这样在NODE Manager 就可以通过 HDFS取得这些文件

         app.getApplicationSubmissionContext().setAMContainerSpec(amContainer);

****5.  提交应用****

         ApplicationId appId = yarnClient.submitApplication(app

                    .getApplicationSubmissionContext());

对于Client 的编写还是比较简答的，不需要维护状态，只需要提交相应的消息给Resource Manager 就可以。

# ****YARN  App Mstr编写****

App Mstr的编写比较复杂，其需要通Resource Manager 和 Node Manager 交互，

通过Resource Manager： 申请Container， 并接收Resource Manager 的一些消息，如 可用的Container， 结束的Container等。

通过Node Manage : 启动Container ， 并接收Node Manage 的一些消息 ，如 Container 的状态变化 以及Node 状态变化。

****1. 创建一个AMRMClientAsync对象，这个对象负责与Resource Manager 交互****

        amRMClient = AMRMClientAsync.createAMRMClientAsync( 1000, new RMCallbackHandler());

这里的RMCallbackHandler 是我们编写的继承自AMRMClientAsync.CallbackHandler 的一个类，其功能是处理由Resource Manager收到的消息,

       其需要实现的方法由如下

       public void onContainersCompleted(List<ContainerStatus> statuses);

          public void onContainersAllocated(List<Container> containers) ;  
          public void onShutdownRequest() ;  
          public void onNodesUpdated(List<NodeReport> updatedNodes) ;  
          public void onError(Throwable e) ;

       这里不考虑异常的情况下，只写onContainersAllocated， onContainersCompleted 这两个既可以， 一个是当有新的Container 可以使用， 一个是Container 运行结束。

       在onContainersAllocated 我们需要编写 启动container 的代码，amNMClient.startContainerAsync(container, ctx); 这里的ctx 同Yarn Client 中第4步中的amContainer 是同一个类型， 即这个container 执行的一些资源，环境变量与命令等， 因为这是在回调函数中，为了保证时效性，这个操作最好放在线程池中异步操作。

       在onContainersCompleted 中，如果是失败的Container，我们需要重新申请并启动Container，（这一点有可能是YARN的 Fair Schedule 中会强制退出某些Container 以释放资源） 成功的将做记录既可以。

****2. 创建一个NMClientAsyncImpl对象，这个对象负责与Node Manager 交互****

      amNMClient = new NMClientAsyncImpl(new NMCallbackHandler());

      这里NMCallbackHandler 使我们需要编写的继承自NMClientAsync.CallbackHandler 的对象，其功能是处理由Node Manager 收到的消息

       public void onContainerStarted(ContainerId containerId,  Map<String, ByteBuffer> allServiceResponse);

          public void onContainerStatusReceived(ContainerId containerId,  ContainerStatus containerStatus);  
          public void onContainerStopped(ContainerId containerId) ;  
          public void onStartContainerError(ContainerId containerId, Throwable t);  
          public void onGetContainerStatusError(ContainerId containerId,  Throwable t) ;  
          public void onStopContainerError(ContainerId containerId, Throwable t);

      这里简单的不考虑异常的情况下，这些函数可以写一个空函数体，忽略掉处理

****3. 将自己注册到Resource Manager 上****

RegisterApplicationMasterResponse response = amRMClient

                    .registerApplicationMaster(NetUtils.getHostname(), -1, "");

        这个函数将自己注册到RM上，这里没有提供RPC port 和TrackURL.

****4. 向RM 申请Container****

ContainerRequest containerAsk = new ContainerRequest(  
                         //100\*10M + 1vcpu  
                         Resource.newInstance(100, 1), null, null,  
                         Priority.newInstance(0));

      amRMClient.addContainerRequest(containerAsk);

     这里一个containerAsk   表示申请一个 Container， 这里的对nodes和rasks 设置为NULL，猜测MapReduce应该由参数来尝试申请靠近HDFS block的container的

****5. 等待Container 执行完毕，清理退出****

我的代码如下， 循环等待container 执行完毕，并上报执行结果

void waitComplete() throws YarnException, IOException{

          while(numTotalContainers.get() != numCompletedConatiners.get()){  
               try{  
                    Thread.sleep(1000);  
                    LOG.info("waitComplete" +   
                         ", numTotalContainers=" + numTotalContainers.get() +  
                         ", numCompletedConatiners=" + numCompletedConatiners.get());  
               } catch (InterruptedException ex){}  
          }  
          exeService.shutdown()；  
          amNMClient.stop();  
          amRMClient.unregisterApplicationMaster(FinalApplicationStatus.SUCCEEDED, "dummy Message", null)；  
          amRMClient.stop();  
     }

# YARN  Container Application

   真正处理数据的是由App Mstr 由amNMClient.startContainerAsync(container, ctx)提交的 Container application,  然后这这个应用并不需要特殊编写，任何程序通过提交相应的运行信息都可以在这些Node中的某个Container 中执行， 所以这个程序可以是一个复杂的MapReduce  Task 或者 是一个简单的脚本。

总结：

YARN 提供了对cluster 资源管理 和 作业调度的功能。

编写一个应用运行在YARN 之上，比较复杂的是App Mstr 的编写，其需要维护container 的状态并能共做一些错误恢复，重启应用的操作。 比较简答的是Client的编写，只需要提交必须的信息既可以，不需要维护状态。 真正运行处理数据的是Container Application ，这个程序可以不需要针对YARN做代码编写。

# 参考

<https://blog.csdn.net/trumanz/article/details/46627875>