Apache Spark的部署

目前Apache Spark支持三种分布式部署方式，分别是standalone,Spark on Mesos和Spark On Yarn。其中第一种类似于MR 1.0采用的模式，内部实现了容错性和资源管理，后两种则是未来的发展趋势，部署容错性和资源管理交由资源管理系统完成。让Spark运行在一个通用的资源管理系统之上，这样可以与其它计算框架，如MapReduce共用一个集群资源，降低运维成本和提高资源利用率。

# 1.Spark Standalone

Standalone模式，即独立部署模式，自带完整的服务，可单独部署到一个集群中，无需依赖任务其它资源管理系统。从一定程度上说，该模式是其他两种的基础。借鉴Spark开发模式，我们可以得到一种新型计算框架的一般思路：先设计出它的standalone模式，为了快速开发，起初不需要考虑服务的容错性，之后再开发相应的wrapper，将standalone模式下的服务原封不到的部署到资源管理系统YARN或者Mesos上，由资源管理系统负责服务本身的容错。

## 安装

通过手动的启动master和worker进程来启动Spark集群，步骤如下：

1. 下载安装包

*wget* [*https://d3kbcqa49mib13.cloudfront.net/spark-2.1.1-bin-hadoop2.6.tgz*](https://d3kbcqa49mib13.cloudfront.net/spark-2.1.1-bin-hadoop2.6.tgz)

*tar -zxf spark-2.1.1-bin-hadoop2.6.tgz*

1. 启动master

*./sbin/start-master.sh*

启动进程信息如下所示:

*[spark@fys1 spark-2.1.1-bin-hadoop2.6]$ jps*

*11210 Master*

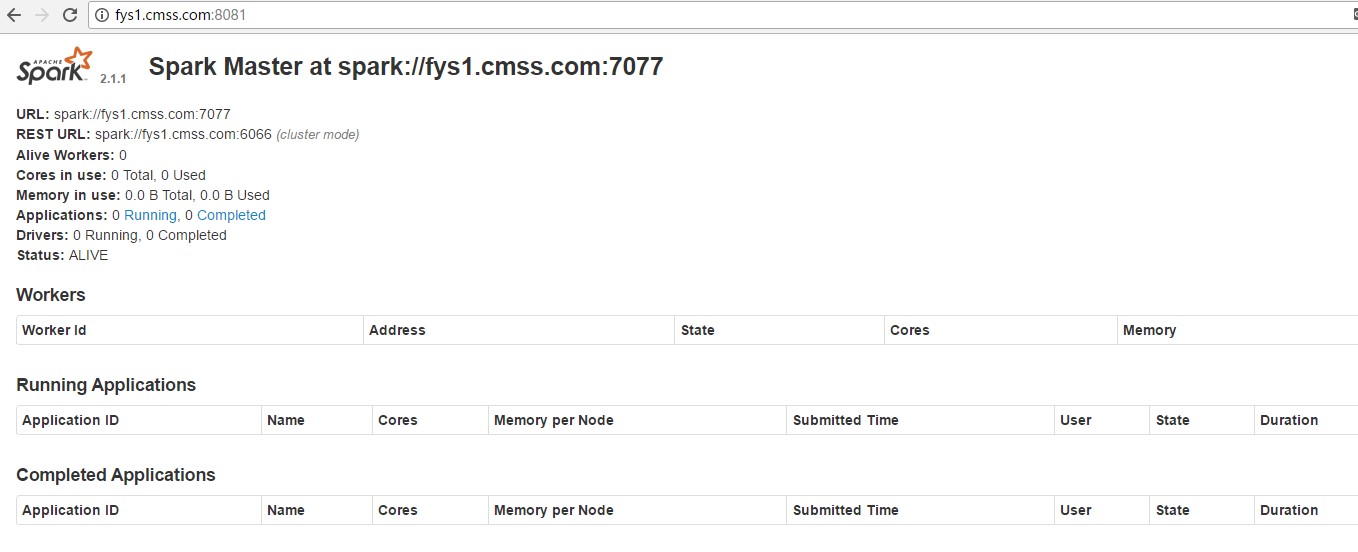
*[spark@fys1 spark-2.1.1-bin-hadoop2.6]$ netstat -anp|grep 11210*

*tcp 0 0 ::ffff:10.139.9.123:7077 :::\* LISTEN 11210/java*

*tcp 0 0 :::8081 :::\* LISTEN 11210/java*

*tcp 0 0 ::ffff:10.139.9.123:6066 :::\* LISTEN 11210/java*

启动后，master使用Spark://HOST:PORT作为其URL，使用该URL与Worker交互，SparkContext中使用该URL作为参数。Master的默认Web UI为<http://localhost:8080>（端口被占用，则重新分配一个端口），页面如下所示：



1. 启动slave

*$sbin/start-slave.sh spark://fys1.cmss.com:7077*

在命令行中指定spark://MASTER:PORT，启动信息如下：

*[spark@fys1 spark-2.1.1-bin-hadoop2.6]$ jps*

*11210 Master*

*11675 Worker*

*[spark@fys1 spark-2.1.1-bin-hadoop2.6]$ netstat -anp|grep 11675*

*tcp 0 0 :::8082 :::\* LISTEN 11675/java*

启动worker后，可以在master的web ui上看到新启动的worker node信息，同时会列出cpu和memory。

1. 配置文件详解

Spark中有两个配置文件，spark-env.sh和spark-defaults.conf，下面进行详述：

1. spark-defaults.conf，控制应用程序绝大部分属性，并且可以通过Spark Conf对象或者Java系统属性进行设置，Spark Conf对象可以传递给SparkContext，这些属性可以传递给set方法任意key-value对。如下所示：

*val conf = new SparkConf()*

*.setMaster("local")*

*.setAppName("CountingSheep")*

*.set("spark.executor.memory", "1g")*

*val sc = new SparkContext(conf)*

在一些场景中，避免在代码中将SparkConf对象的属性进行设置死，比如可能想在不同的master护着不同的内存容量运行应用程序，可以在运行程序的时候配置。

Spark Shell和Spark-submit工具支持两种方式来动态加载配置属性，第一种是命令行方式，比如—master；spark-submit工具可以通过—conf标记接收任何的spark属性。Spark-submit工具也会从conf/spark-defaults.conf配置文件中读取配置选项，如下所示:

*spark.master            spark://iteblog.com:7077*

*spark.executor.memory   512m*

*spark.eventLog.enabled  true*

*spark.serializer        org.apache.spark.serializer.KryoSerializer*

每个值将作为flags传递给应用中并将SparkConf对象中的属性进行合并，通过SparkConf对象配置的属性优先级最高，其次是对spark-submit或spark-shell通过flags配置，最后是spark-defaults.conf文件中的配置。

在应用的Web UI界面http://<driver>:4040界面上的Environment tab上可以看到Spark配置。应用的配置如下：

|  |  |  |
| --- | --- | --- |
| 参数名 | 默认值 | 含义 |
| spark.app.name | None | 应用程序的名字，将在UI和日志数据中出现 |
| spark.driver.cores | 1 | Driver程序运行所需要的cpu内核数 |
| spark.driver.maxResultSize | 1g | 每个Spark action所有分区的序列化结构的总大小限制，设置值至少1M，设置0时无限制。如果总大小超过这个值，程序将会终止。该值设置过大会造成Driver的OOM问题，依赖spark.driver.memory和JVM中对象的消耗 |
| spark.driver.memory | 512m | Driver进程使用的内存大小 |
| spark.executor.memory | 512m | 每个executor进程使用的内存数，和JVM内存相同的格式 |
| spark.extralisteners | None | 注册监听器，需要实现SparkListener |
| spark.local.dir | /tmp | Spark暂存空间的使用目录，包括map的输出文件和存储到磁盘的RDD，可以配置多目录，用逗号隔开。在Spark 1.0之后，该参数被SPARK\_LOCAL\_DIRS(standalone和mesos)及LOCAL\_DIRS(YARN)环境变量所覆盖，由ClusterManager覆盖。 |
| spark.logConf | False | 当SparkContext启动后，将有效的SparkConf记录为INFO |
| spark.master | None | Cluster Manager连接的url |
| spark.submit.deployMode | None | Spark Driver的部署模式，Client或者Cluster，client是将driver程序启动在本地，cluster模式是 启动在集群中另外一个主机上。 |

运行时参数如下所示：

|  |  |  |
| --- | --- | --- |
| 参数名 | 默认值 | 含义 |
|  |  |  |

http://spark.apache.org/docs/latest/configuration.html

http://blog.csdn.net/guohecang/article/details/52088117

1. spark-env.sh，配置环境变量，可以分别对每台机器进行相应的配置，比如IP等

通过环境变量配置确定的Spark设置，环境变量从Spark安装目录下的conf/spark-env.sh脚本中读取，在独立的或者mesos模式下，这个文件可以给机器确定的信息，如主机名。当运行本地应用程序或者提交脚本时，也起作用，可以在spark-env.sh中设置的变量如下所示：

|  |  |
| --- | --- |
| 参数名 | 含义 |
| JAVA\_HOME | Java安装的路径 |
| PYSPARK\_PYTHON | PySpark用到的Python二进制执行文件路径 |
| SPARK\_LOCAL\_IP | 主机绑定的Ip地址 |
| SPARK\_PUBLIC\_DNS | Spark应用程序通知给其他机器的主机名 |

除了以上这些，Spark Standalone cluster 脚本也可以设置一些选项，例如每台机器使用的核数以及最大内存。

1. 日志，所有的日志相关属性可以在log4j.properties文件中进行设置

Spark用log4j logging，可以通过在conf目录下添加log4j.properties文件来设置。

http://spark.apache.org/docs/latest/spark-standalone.html#starting-a-cluster-manually

## 在集群中运行程序

## 1.2.1 Spark Shell

在Spark Cluster中运行应用，将master的spark://ip:port参数传递到SparkContext构造器。交互式Spark Shell的执行命令如下：

*./bin/spark-shell --master spark://IP:PORT*

同时也可以传如total-executor-cores参数，来配置Spark Shell可以使用的集群cores的数目。

## 1.2.2 Spark Submit

可以通过spark-submit脚本来向集群中提交Spark 应用。在Standalone模式下，Spark支持两种部署模式:

1. Client，driver程序和提交客户端是同一个进程
2. Cluster，Driver启动在另外一个Worker的进程中，客户端提交程序后不等待应用结束就退出

如果应用程序通过spark submit提交程序，application的jar包会自动分发到worker节点上，如果app jar依赖additional jars，可以通过—jars标识来指定。应用的配置和执行环境，可以在Spark Configuration中使用。

如果应用程序异常退出，Spark会重启应用，该特性通过在spark-submit提交程序通过supervise标识来设置，应用程序的kill通过下面的命令：

*./bin/spark-class org.apache.spark.deploy.Client kill <master url> <driver ID>*

可以在Master UI页面上找到对应的Driver ID。

## 资源调度

Standalone Cluster模式下仅支持FIFO的调度。但是，为了支持多个用户，可以控制每个应用可以使用的最大资源。默认情况下，配置集群中的所有cores，也就是集群中一次仅能运行一个应用。可以在Spark Conf中配置spark.cores.max，例如：

*val conf = new SparkConf()*

*.setMaster(...)*

*.setAppName(...)*

*.set("spark.cores.max", "10")*

*val sc = new SparkContext(conf)*

而且，可以在master进程的启动参数spark.deploy.defaultCores进行配置，也可以在spark-env.sh中通过配置：

*export SPARK\_MASTER\_OPTS="-Dspark.deploy.defaultCores=<value>"*

## 监控和日志

Spark Standalone 模式提供Web-based用户接口来监控集群，master和worker进程都会有Web UI来展示集群和Job统计信息。默认情况下，Master的Web UI使用端口8080，在配置文件或者命令行中均可以指定。

默认情况下Job的log输出默认是SPARK\_HOME/work目录。

## Runnig Alongside Hadoop

Spark Standalone模式下启动Spark Cluster，该Cluster可以直接使用Hadoop中的服务。在Spark中访问Hadoop数据，通过hdfs://URL（例如，hdfs://<namenode>:9000/path），可以在Hadoop NameNode的Web UI上可以获取正确的hdfs路径。

当Spark和Hadoop不共用节点，访问HDFS会因为跨节点的问题，造成性能问题，但是如果是同一网络可以忽略这个问题。

# 2.Spark On YARN

限于YARN自身的发展，目前仅支持粗粒度模式，在YARN中Container的资源时不可以动态伸缩的，一旦container启动之后，使用的资源不再发生变化。将Spark运行在YARN上，Spark自身只提供了作业管理功能，资源调度依托于YARN。Spark on Yarn支持两种模式：

1. yarn-cluster，适用于生产环境
2. yarn-client，适用于交互、调试并且希望立即看到应用的输出。

yarn-cluster和yarn-client的区别在于yarn appMaster，每个yarn app实例有一个appMaster进程，是为app启动的第一个container，负责从RM请求资源，获取到资源后，告诉NM为其启动container。yarn-client和yarn-cluster的模式内部实现有很多区别。

Spark on YARN依靠yarn来调度Spark Job，不用维护Spark Master和Worker进程，部署时保证环境变量HADOOP\_CONF\_DIR和YARN\_CONF\_DIR指向Hadoop集群的配置目录即可，在spark-env.sh中设置如下：

*export JAVA\_HOME=/usr/jdk64/jdk1.7.0\_67*

*export SCALA\_HOME=/home/spark/scala-2.12.0*

*export SPARK\_MASTE\_IP=10.139.9.123*

*export SPARK\_WORKER\_MEMORY=4G*

*export HADOOP\_CONF\_DIR=/cmss/bch/bc1.3.5/hadoop/etc/hadoop*

*export YARN\_CONF\_DIR=/cmss/bch/bc1.3.5/hadoop/etc/hadoop*

*export SPARK\_MASTER\_PORT=7077*

*export MASTER=spark://${SPARK\_MASTER\_IP}:${SPARK\_MASTER\_PORT}*

在YARN中运行Spark程序有两种方式，通过—master参数来指定，命令格式如下所示：

*$ ./bin/spark-submit --class path.to.your.Class --master yarn --deploy-mode cluster [options] <app jar> [app options]*

1. cluster 模式，spark driver运行在yarn am 进程中，提交后客户端不等待结果，直接退出

*$ ./bin/spark-submit --class org.apache.spark.examples.SparkPi \*

*--master yarn \*

*--deploy-mode cluster \*

*--driver-memory 2g \*

*--executor-memory 1g \*

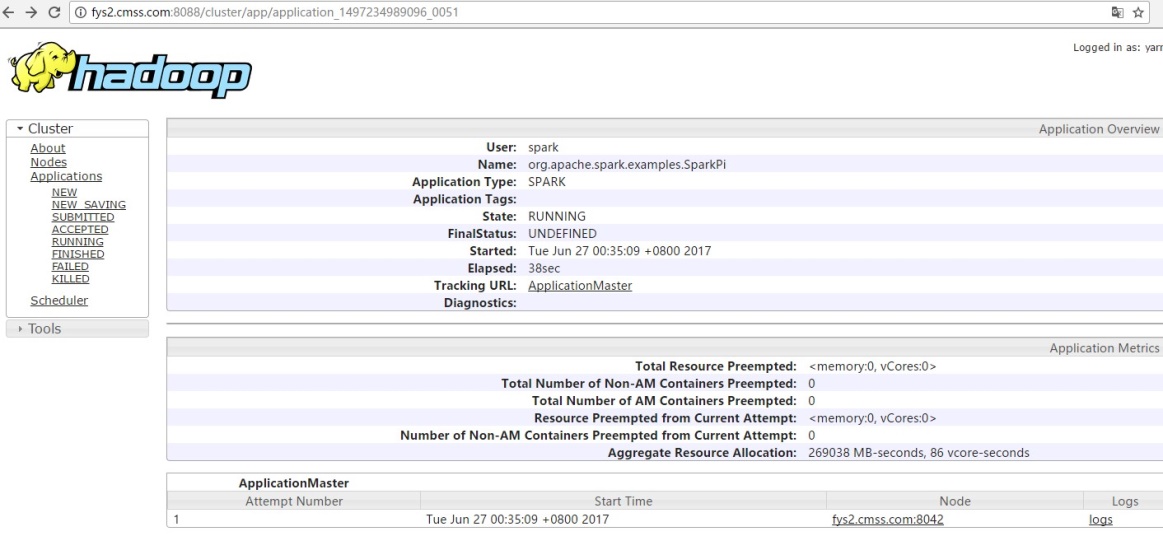
*--executor-cores 1 \*

*--queue root.default \*

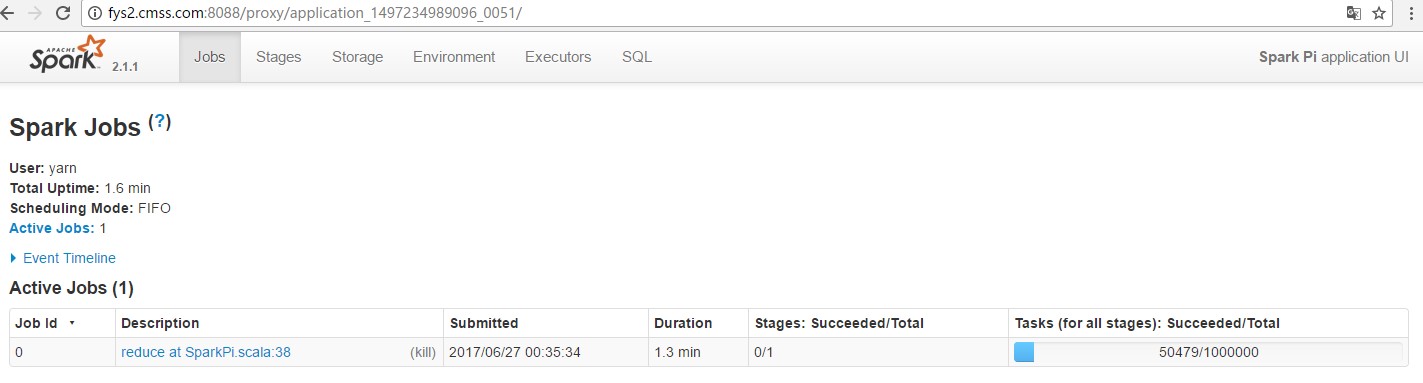
*examples/jars/spark-examples\*.jar \*

*10*

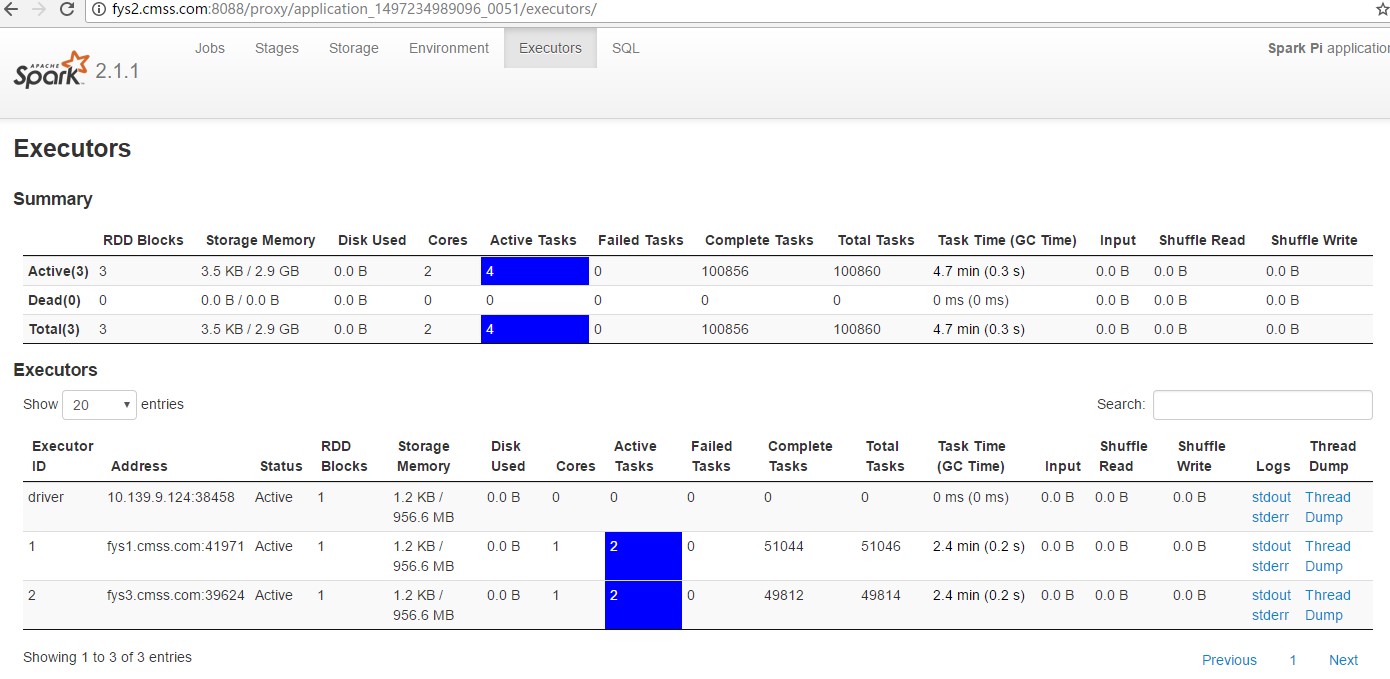
提交后，在yarn的Web UI上可以看到app，如下所示：



进入Spark AM Master，页面如下所示：



可以查到到具体的执行信息，如stages,环境变量，Executors等执行信息，其中Executors的监控页面如下：



可以看到driver启动节点（am），其执行executor的启动信息。

1. client模式，driver启动在客户端进程中，am仅用于向yarn申请资源，命令如下：

*$ ./bin/spark-submit --class org.apache.spark.examples.SparkPi \*

*--master yarn \*

*--deploy-mode cluster \*

*--driver-memory 2g \*

*--executor-memory 1g \*

*--executor-cores 1 \*

*--queue root.default \*

*examples/jars/spark-examples\*.jar \*

*10*

在客户端可以看到driver中的调度信息，相关输出如下：

*17/06/27 00:46:06 INFO util.Utils: Successfully started service 'SparkUI' on port 4040.*

*17/06/27 00:46:06 INFO ui.SparkUI: Bound SparkUI to 0.0.0.0, and started at http://10.139.9.123:4040*

*17/06/27 00:46:37 INFO spark.SparkContext: Starting job: reduce at SparkPi.scala:38*

*17/06/27 00:46:37 INFO scheduler.DAGScheduler: Got job 0 (reduce at SparkPi.scala:38) with 10 output partitions*

*17/06/27 00:46:37 INFO scheduler.DAGScheduler: Final stage: ResultStage 0 (reduce at SparkPi.scala:38)*

*17/06/27 00:46:37 INFO scheduler.DAGScheduler: Parents of final stage: List()*

*17/06/27 00:46:37 INFO scheduler.DAGScheduler: Missing parents: List()*

*17/06/27 00:46:37 INFO scheduler.DAGScheduler: Submitting ResultStage 0 (MapPartitionsRDD[1] at map at SparkPi.scala:34), which has no missing parents*

*17/06/27 00:46:38 INFO cluster.YarnSchedulerBackend$YarnDriverEndpoint: Registered executor NettyRpcEndpointRef(null) (10.139.9.124:38744) with ID 2*

*17/06/27 00:46:38 INFO scheduler.DAGScheduler: Submitting 10 missing tasks from ResultStage 0 (MapPartitionsRDD[1] at map at SparkPi.scala:34)*

*17/06/27 00:46:38 INFO cluster.YarnScheduler: Adding task set 0.0 with 10 tasks*

*17/06/27 00:46:38 INFO scheduler.TaskSetManager: Starting task 0.0 in stage 0.0 (TID 0,*

*ge 0.0 (TID 1) in 948 ms on fys1.cmss.com (executor 1) (1/10)*

*17/06/27 00:46:39 INFO scheduler.TaskSetManager: Starting task 3.0 in stage 0.0 (TID 3, fys1.cmss.com, executor 1, partition 3, PROCESS\_LOCAL, 6035 bytes)*

*17/06/27 00:46:39 INFO scheduler.TaskSetManager: Finished task 2.0 in stage 0.0 (TID 2) in 41 ms on fys1.cmss.com (executor 1) (2/10)*

*17/06/27 00:46:39 INFO scheduler.TaskSetManager: Starting task 4.0 in stage 0.0 (TID 4, fys1.cmss.com, executor 1, partition 4, PROCESS\_LOCAL, 6035 bytes)*

*17/06/27 00:46:39 INFO scheduler.TaskSetManager: Finished task 3.0 in stage 0.0 (TID 3) in 32 ms on fys1.cmss.com (executor 1) (3/10)*

*…………….*

*17/06/27 00:46:39 INFO scheduler.TaskSetManager: Finished task 9.0 in stage 0.0 (TID 9) in 30 ms on fys1.cmss.com (executor 1) (10/10)*

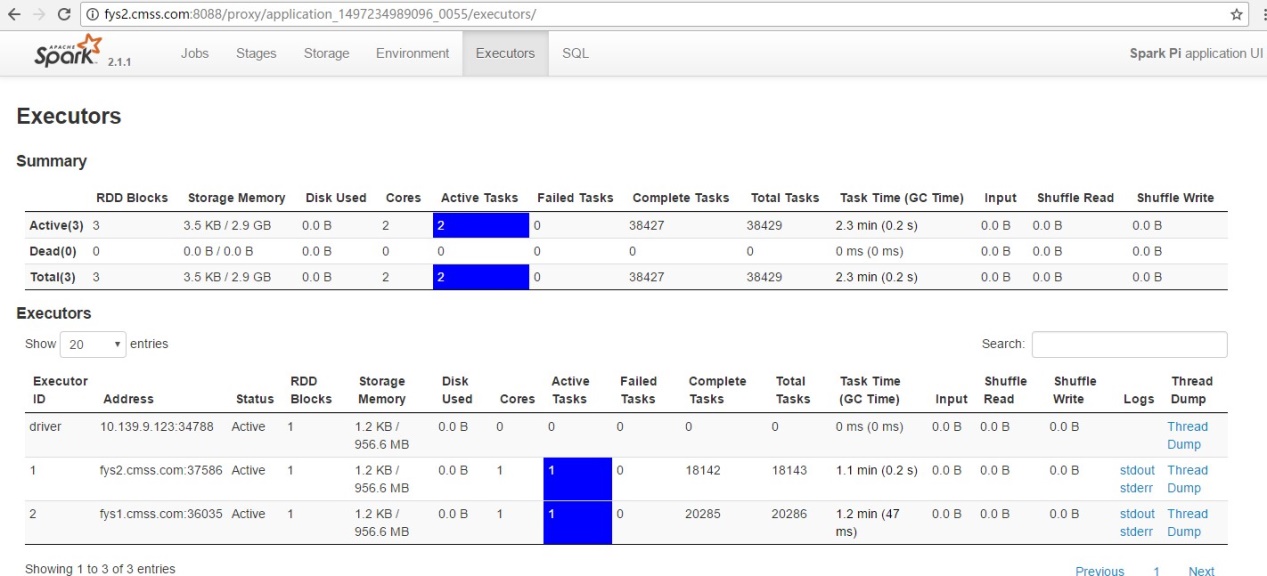
*17/06/27 00:46:39 INFO cluster.YarnScheduler: Removed TaskSet 0.0, whose tasks have all completed, from pool*

*17/06/27 00:46:39 INFO scheduler.DAGScheduler: ResultStage 0 (reduce at SparkPi.scala:38) finished in 1.203 s*

*17/06/27 00:46:39 INFO scheduler.DAGScheduler: Job 0 finished: reduce at SparkPi.scala:38, took 2.486423 s*

*Pi is roughly 3.141095141095141*

打开driver:4040，页面如下所示，跳转到yarn web ui：



Driver启动在提交的节点上。

# 3. Spark On Mesos

Spark之初考虑到支持Mesos，因此目前官方推荐这种模式，而且Spark运行在Mesos上会比运行在YARN上更加灵活，更加自然。目前Spark On Mesos环境中，用于可以选择两种调度模式之一运行自己的应用程序：

1. 粗粒度模式，每个应用程序的运行环境由一个Driver和若干个Executor组成，其中，每个Executor占用若干资源，内部可运行多个Task，对应多个Slot，应用程序在各个任务正式执行之前，需要将运行环境中的资源全部申请好，且运行过程中要一直占用这个资源，即使不用，最后程序运行结束后，回收这种资源。
2. 细粒度模式，鉴于粗粒度模式会造成大量资源浪费，这种模式类似于云计算，思想是按需分配

该部分内容不再介绍。

参考链接：

http://www.oschina.net/translate/spark-standalone?cmp

Spark独立部署：<http://spark.apache.org/docs/latest/spark-standalone.html>

http://spark.apache.org/docs/latest/spark-standalone.html