

# 1. Spark-on-YARN

yarn 是 hadoop 中的一个组件，统一的资源调度平台。

spark 任务集群运行的一种方式

spark 程序运行在 yarn 上，standalone 是否需要启动？

不需要部署启动 standalone 集群，只需要配置 jdk，yarn 的配置路径。

官方文档：

<http://spark.apache.org/docs/latest/running-on-yarn.html>

## 1.1. 配置安装

### 1.1.1. 安装 hadoop 环境

1.安装 hadoop: 需要安装 HDFS 模块和 YARN 模块，HDFS 必须安装，spark 运行时要把 jar 包存放到 HDFS 上。

补充配置一：

yarn 默认情况下，只根据内存调度资源，所以 spark on yarn 运行的时候，即使通过 --executor-cores 指定 vcore 个数为 N，但是在 yarn 的资源管理页面上看到使用的 vcore 个数还是 1。相关配置在 **capacity-scheduler.xml** 文件：

```
<property>
<name>yarn.scheduler.capacity.resource-calculator</name>
<!-- <value>org.apache.hadoop.yarn.util.resource.DefaultResourceCalculator</value> -->
<value>org.apache.hadoop.yarn.util.resource.DominantResourceCalculator</value>
</property>
```

补充配置二：

修改所有 yarn 节点的 yarn-site.xml，在该文件中添加如下配置

```
<property>
  <name>yarn.nodemanager.pmem-check-enabled</name>
  <value>false</value>
</property>
<property>
  <name>yarn.nodemanager.vmem-check-enabled</name>
  <value>false</value>
</property>
```

如果不配置这两个选项，在 spark-on-yarn 的 client 模式下，如果内存不足，报错如下：

```
18/01/26 20:17:41 ERROR client.TransportClient: Failed to send RPC 6505294471618682346 to /192.168.1.8.11:48756: java.nio.channels.ClosedChannelException
java.nio.channels.ClosedChannelException
    at io.netty.channel.AbstractChannel$AbstractUnsafe.write(...) (Unknown Source)
18/01/26 20:17:41 ERROR cluster.YarnSchedulerBackend$YarnSchedulerEndpoint: Sending RequestExecutors(0,0,Map(),Set()) to AM was unsuccessful
java.io.IOException: Failed to send RPC 6505294471618682346 to /192.168.1.8.11:48756: java.nio.channels.ClosedChannelException
    at org.apache.spark.network.client.TransportClient.lambda$sendRpc$2(TransportClient.java:237)
```

分析原因：内存溢出

参数说明：

yarn.nodemanager.pmem-check-enabled

是否启动一个线程检查每个任务正使用的物理内存量,如果任务超出分配值,则直接将其杀掉,默认是 true。

yarn.nodemanager.vmem-check-enabled

是否启动一个线程检查每个任务正使用的虚拟内存量,如果任务超出分配值,则直接将其杀掉,默认是 true。

把配置分发到各台节点上:

```
# cd /root/apps/hadoop/etc/hadoop
[root@hdp-01 hadoop]# for i in 2 3 ;do scp capacity-scheduler.xml yarn-site.xml
hdp-0$i:`pwd`;done
```

### 1.1.2. 同步系统时间:

1, 手动设置同步时间: `date -s 2018-03-09`    `date -s 09:30:20`

2, 利用命令同步时间    `ntpdate` 时间服务器

`ntpdate` 命令没有, 就使用 `yum -y install ntpdate` 来安装该命令。

```
ntpdate us.pool.ntp.org                      该命令作用: 利用 ntpdate 同步标准时间
1.cn.pool.ntp.org
2.cn.pool.ntp.org
3.cn.pool.ntp.org
0.cn.pool.ntp.org
cn.pool.ntp.org
tw.pool.ntp.org
0.tw.pool.ntp.org
1.tw.pool.ntp.org
2.tw.pool.ntp.org
3.tw.pool.ntp.org
```

### 1.1.3. spark 配置

**安装 Spark:** 解压 Spark 安装程序到一台服务器上, 修改 spark-env.sh 配置文件, spark 程序将作为 YARN 的客户端用于提交任务

在 spark-env.sh 中配置

```
export JAVA_HOME=/usr/local/jdk1.8.0_131
export HADOOP_CONF_DIR=/root/apps/hadoop/etc/hadoop
```

**注意:** 该配置文件的作用, 就是告知 spark 程序 yarn 的位置。

可以使用的是 HADOOP\_CONF\_DIR 或者 YARN\_CONF\_DIR, 如果不配置该选项, 就会报错如下:

```
Exception in thread "main" java.lang.Exception: When running with master 'yarn' either HADOOP_CONF_DIR or YARN_CONF_DIR must be set in the environment.
```

启动 HDFS 和 YARN 后, 就可以提交任务到 yarn 集群中。

## 1.2. 运行模式 (cluster 模式和 client 模式)

任务提交到 yarn 集群中, 没有分配到资源时的状态。

```
18/03/09 09:44:31 INFO yarn.Client: Application report for application_1520559300927_0001 (state: ACCEPTED)
18/03/09 09:44:32 INFO yarn.Client: Application report for application_1520559300927_0001 (state: ACCEPTED)
```

spark1.6 中的提交方式: --master yarn-client yarn-cluster

```
[root@hdp-01 ~]# spark-submit --master yarn-cluster --class org.apache.spark.examples.SparkPi \
/root/apps/spark-2.2.0-bin-hadoop2.7/examples/jars/spark-examples_2.11-2.2.0.jar 100
Warning: Master yarn-cluster is deprecated since 2.0. Please use master "yarn" with specific deploy mode instead.
```

### 1.cluster 模式

```
./bin/spark-submit --class org.apache.spark.examples.SparkPi \
--master yarn \
--deploy-mode cluster \
--driver-memory 1g \
--executor-memory 1g \
--executor-cores 2 \
--queue default \
lib/spark-examples*.jar \
100
```

```
spark-submit --master yarn --deploy-mode cluster --class org.apache.spark.examples.SparkPi \
/root/apps/spark-2.2.0-bin-hadoop2.7/examples/jars/spark-examples_2.11-2.2.0.jar 1000
```

cluster 模式下的进程信息:

```
[root@hdp-03 ~]# jps
5123 Jps
2435 DataNode
2551 NodeManager
4940 ApplicationMaster
5039 CoarseGrainedExecutorBackend
[root@hdp-03 ~]#

[root@hdp-01 ~]# jps
6400 ResourceManager
5985 NameNode
8584 CoarseGrainedExecutorBackend
6526 NodeManager
7903 SparkSubmit
6127 DataNode
```

sparkSubmit: 仅仅是负责提交任务，提交完任务，功能结束。（standalone 中，driver 和 sparksubmit 在一起）

之前说的 dag, stage, 生成 task, 都是由 AppMaster 完成的，（yarn-cluster 模式下，driver 是在 AppMaster 中）

资源可以 DIY:

```
spark-submit --master yarn --deploy-mode cluster --driver-memory 1g --driver-cores 2
--executor-cores 2 --executor-memory 1g --num-executors 100 --class
org.apache.spark.examples.SparkPi
/root/apps/spark-2.2.0-bin-hadoop2.7/examples/jars/spark-examples_2.11-2.2.0.jar 1000
```

--num-executors: 总的 executor 的个数

没有 standalone 模式，依然可以查看 spark 任务的 DAG 图

Cluster

- About
- Nodes
- Node Labels
- Applications
  - NEW
  - NEW SAVING
  - SUBMITTED
  - ACCEPTED
  - RUNNING
  - FINISHED
  - FAILED
  - KILLED
- Scheduler

Tools

Kill Application

User:	root
Name:	org.apache.spark.examples.SparkPi
Application Type:	SPARK
Application Tags:	
Application Priority:	0 (Higher Integer value indicates higher priority)
YarnApplicationState:	RUNNING: AM has registered with the cluster
Queue:	default
FinalStatus Reported by AM:	Application has not completed yet
Started:	Fri Feb 02 22:56:54 +0800 2018
Elapsed:	15sec
Tracking URL:	ApplicationMaster
Log Aggregation Status:	DISABLED

## 2.client 模式

```
./bin/spark-submit --class org.apache.spark.examples.SparkPi \
--master yarn \
```

```
--deploy-mode client \  
--driver-memory 1g \  
--executor-memory 1g \  
--executor-cores 2 \  
--queue default \  
lib/spark-examples*.jar \  
10
```

client 模式下，没有了 AppMaster，替代的是 ExecutorLauncher，功能类似于 AppMaster

```
[root@hdp-03 ~]# jps  
2435 DataNode  
2551 NodeManager  
6730 Jps  
6682 CoarseGrainedExecutorBackend  
6591 ExecutorLauncher  
[root@hdp-03 ~]#
```

```
[root@hdp-01 ~]# jps  
6400 ResourceManager  
5985 NameNode  
10497 CoarseGrainedExecutorBackend  
10498 CoarseGrainedExecutorBackend  
9738 SparkSubmit  
6526 NodeManager  
6127 DataNode  
10527 Jps
```

```
[root@hdp-01 ~]# spark-shell --master yarn --deploy-mode cluster  
--cores 2 --executor-cores 2 --executor-memory 1g  
Error: Cluster deploy mode is not applicable to Spark shells.  
Run with --help for usage help or --verbose for debug output  
[root@hdp-01 ~]#
```

spark-shell 必须使用 client 模式

```
./bin/spark-shell --master yarn --deploy-mode client
```

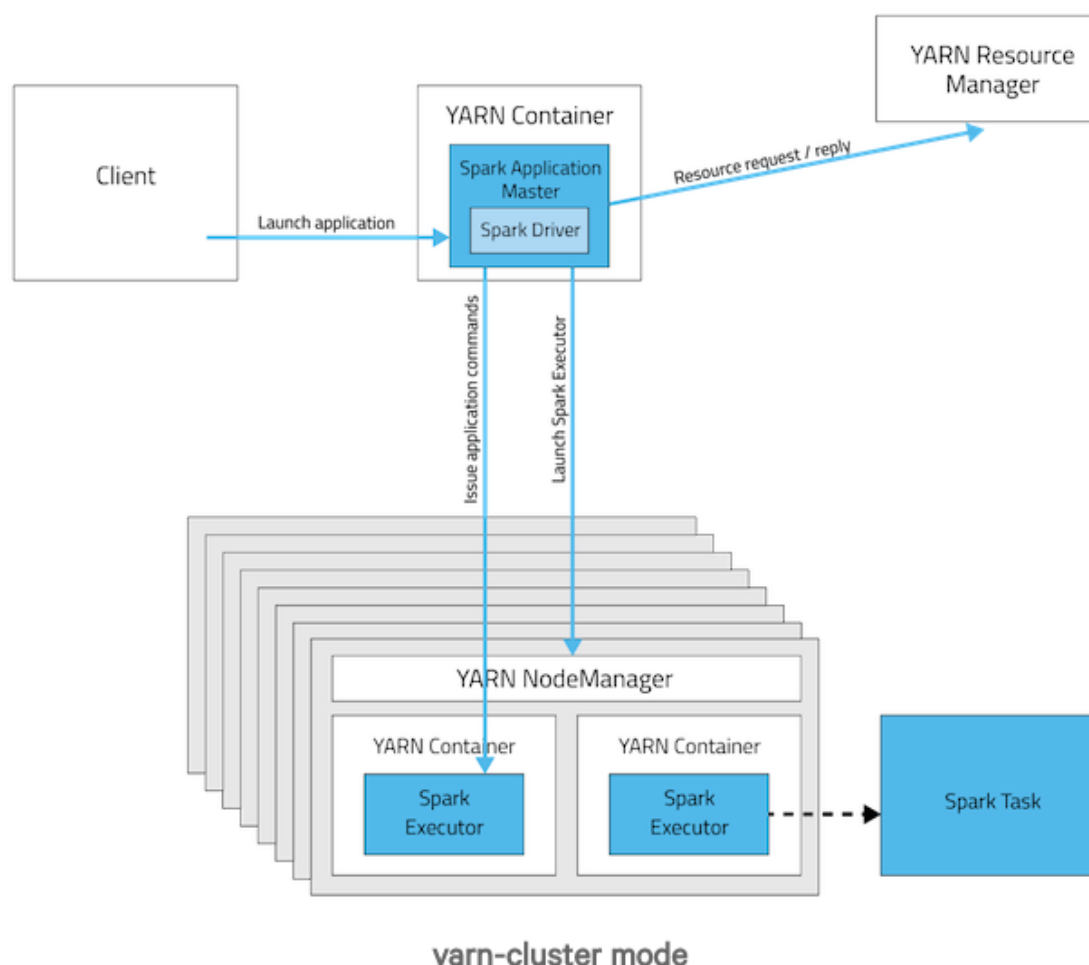
### 3.两种模式的区别

**cluster 模式：**Driver 程序在 YARN 中运行，应用的运行结果不能在客户端显示，所以最好运行那些将结果最终保存在外部存储介质（如 HDFS、Redis、Mysql）而非 stdout 输出的应用程序，客户端的终端显示的仅是作为 YARN 的 job 的简单运行状况。

**client 模式：**Driver 运行在 Client 上，应用程序运行结果会在客户端显示，所以适合运行结果有输出的应用程序（如 spark-shell）

## 1.3. 原理

cluster 模式:



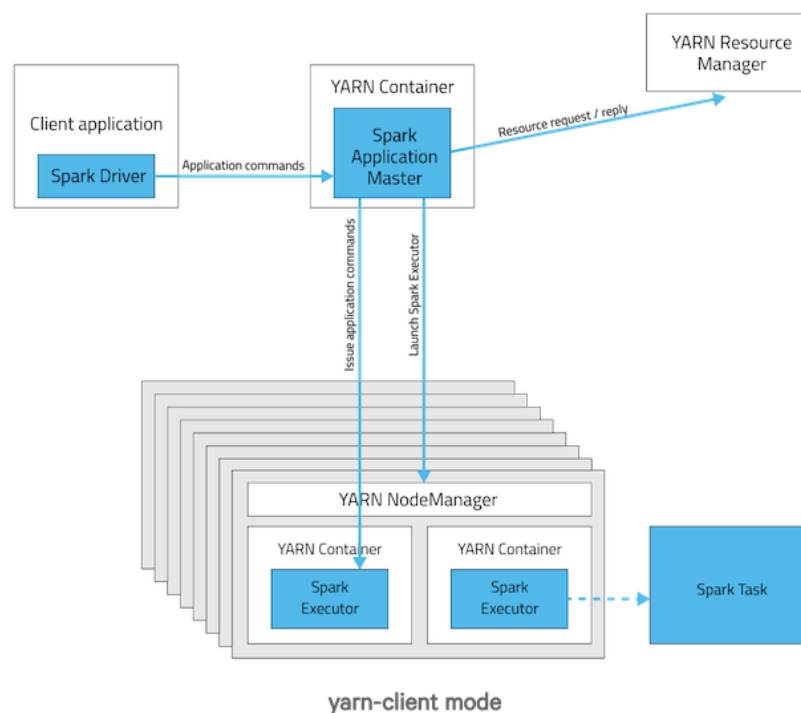
Spark Driver 首先作为一个 ApplicationMaster 在 YARN 集群中启动，客户端提交给 ResourceManager 的每一个 job 都会在集群的 NodeManager 节点上分配一个唯一的 ApplicationMaster，由该 ApplicationMaster 管理全生命周期的应用。具体过程：

1. 由 client 向 ResourceManager 提交请求，并上传 jar 到 HDFS 上  
这期间包括四个步骤：
  - a). 连接到 RM
  - b). 从 RM 的 ASM (ApplicationsManager) 中获得 metric、queue 和 resource 等信息。
  - c). upload app jar 等 jar
  - d). 设置运行环境和 container 上下文 (launch-container.sh 等脚本)
2. ResourceManager 向 NodeManager 申请资源，创建 Spark ApplicationMaster (每个 SparkContext 都有一个 ApplicationMaster)
3. NodeManager 启动 ApplicationMaster，并向 ResourceManager AsM 注册
4. ApplicationMaster 从 HDFS 中找到 jar 文件，启动 SparkContext、DAGscheduler 和 YARN Cluster

## Scheduler

5. ResourceManager 向 ResourceManager AsM 注册申请 container 资源
6. ResourceManager 通知 NodeManager 分配 Container, 这时可以收到来自 ASM 关于 container 的报告。(每个 container 对应一个 executor)
7. Spark ApplicationMaster 直接和 container (executor) 进行交互, 完成这个分布式任务。

### client 模式:



在 client 模式下, Driver 运行在 Client 上, 通过 ApplicationMaster 向 RM 获取资源。本地 Driver 负责与所有的 executor container 进行交互, 并将最后的结果汇总。结束掉终端, 相当于 kill 掉这个 spark 应用。一般来说, 如果运行的结果仅仅返回到 terminal 上时需要配置这个。

客户端的 Driver 将应用提交给 Yarn 后, Yarn 会先后启动 ApplicationMaster 和 executor, 另外 ApplicationMaster 和 executor 都是装载在 container 里运行, container 默认的内存是 1G, ApplicationMaster 分配的内存是 driver-memory, executor 分配的内存是 executor-memory。同时, 因为 Driver 在客户端, 所以程序的运行结果可以在客户端显示, Driver 以进程名为 SparkSubmit 的形式存在。

如果使用 spark on yarn 提交任务, 一般情况, 优先使用 cluster 模式, 该模式, Driver 运行在集群中, 其实就是运行在 ApplicationMaster 这个进程中, 如果该进程出现问题, yarn 会重启 ApplicationMaster (Driver), SparkSubmit 的功能就是为了提交任务。

如果使用交互式的命令行，必须用 **Client** 模式，该模式，Driver 是运行在 SparkSubmit 进程中，因为收集的结果，必须返回到命令行（即启动命令的那台机器上），该模式，一般测试，或者运行 **spark-shell**、**spark-sql** 这个交互式命令行时使用

怎么查看 spark-on-yarn 的任务：

可以查看所有的 yarn application 命令的参数

```
ERROR: Could not find or load main class application
[root@hdp-01 ~]# yarn application
18/02/02 23:01:53 INFO client.RMProxy: Connecting
8032
18/02/02 23:01:53 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Invalid Command Usage :
usage: application
  -appId <Application ID>          Specify Application ID
  -appStates <States>              Works with -list
                                     based on input command
                                     application state
                                     state can be one of
                                     ALL,NEW,NEW_SAVING,
                                     FINISHED,FAILED
  -appTypes <Types>                Works with -list
```

如果杀死一个 application

先通过 `yarn application -list` 查看当前 yarn 上的所有任务

然后通过 `yarn application -kill appId` 杀死一个任务。

`yarn application -kill application_1517581724315_0007`

## 1.4. sparkOnyarn 的资源配置

yarn 中，对每一个容器都有资源限定，目前的默认值都是 1024m

申请资源时，除了指定的资源 `-executor-memory 1g` + 堆外内存的资源 最低的限定是 384M

一个 executor 应该真实的资源是  $(1g + 384M) = 2g$

yarn 中的容器的最小资源分配：

<http://hadoop.apache.org/docs/r2.8.4/hadoop-yarn/hadoop-yarn-common/yarn-default.xml>

<code>yarn.scheduler.minimum-allocation-mb</code>
---

1024
------

yarn 中每一个容器的资源，必须是这个值的倍数。

spark-on-yarn 的配置中：

cluster 模式下的资源分配：

driver, executor 还是 Applicationmaster,除了通过参数分配的资源之外，都有一个堆外资源需要使用：这个资源是 分配资源\*0.10 的值，最小值是 384mb



spark.yarn.executor.memoryOverhead	executorMemory * 0.10, with minimum of 384	The amount of off-heap memory (in megabytes) to be allocated per executor that accounts for things like VM overheads, interned strings overheads, etc. This tends to grow with the executor size (typically 6
spark.yarn.driver.memoryOverhead	driverMemory * 0.10, with minimum of 384	The amount of off-heap memory (in megabytes) to be allocated per mode. This is memory that accounts for things like VM overheads, in native overheads, etc. This tends to grow with the container size (ty
spark.yarn.am.memoryOverhead	AM memory * 0.10, with minimum of 384	Same as spark.yarn.driver.memoryOverhead, but for the YARN A client mode.

如果指定 `--executor-memory = 1g`

实际上 申请的资源是 `1g + 384m`,

再启动容器额时候, 会占用 `1408m` 如果 yarn 的默认值是 `1024m`, 该容器的实际占用资源是 `2g`; 如果 yarn 的默认值是 `512m`, 那么该容器实际占用的资源是 `1.5g`

```

ApplicationMaster.scala x YarnRMClient.scala x YarnAllocator.scala x
YarnAllocator
27 private var numUnexpectedContainerRelease = 0L
28 private val containerIdToExecutorId = new HashMap[ContainerId, String]
29
30 // Executor memory in MB. 通过参数设置的executor的资源
31 protected val executorMemory = sparkConf.get(EXECUTOR_MEMORY).toInt
32 // Additional memory overhead.
33 protected val memoryOverhead: Int = sparkConf.get(EXECUTOR_MEMORY_OVERHEAD).getOrElse(
34 // math.max((0.10 * --executor-memory), 384) 默认最小是384M
35 math.max((MEMORY_OVERHEAD_FACTOR * executorMemory).toInt, MEMORY_OVERHEAD_MIN)).toInt
36 // Number of cores per executor.
37 protected val executorCores = sparkConf.get(EXECUTOR_CORES)
38 // Resource capability requested for each executors
39 // 每一个executor占用的内存资源为: (executor-memory + memoryOverhead 资源)
40 private[yarn] val resource = Resource.newInstance(executorMemory + memoryOverhead, executorCore
41

```

cluster 模式下, driver 的配置是生效的。

但是在 client 模式下, driver 的配置参数是无效的。

验证:

```

spark-submit --master yarn --deploy-mode cluster --driver-memory 2g --executor-memory 500m
--class org.apache.spark.examples.SparkPi
/root/apps/spark-2.2.0-bin-hadoop2.7/examples/jars/spark-examples_2.11-2.2.0.jar 1000

```

如果报错如下: 直接临时添加一个环境变量即可。

```

Exception in thread "main" java.lang.Exception: When running with master 'yarn'
either HADOOP_CONF_DIR or YARN_CONF_DIR must be set in the environment.
at org.apache.spark.deploy.SparkSubmitArguments.validateSubmitArgument
s(SparkSubmitArguments.scala:263)
at org.apache.spark.deploy.SparkSubmitArguments.validateArguments(Spar
kSubmitArguments.scala:240)
at org.apache.spark.deploy.SparkSubmitArguments.<init>(SparkSubmitArgu
ments.scala:116)
at org.apache.spark.deploy.SparkSubmit$.main(SparkSubmit.scala:112)
at org.apache.spark.deploy.SparkSubmit.main(SparkSubmit.scala)
[root@hdp-03 ~]# export HADOOP_CONF_DIR=/root/apps/hadoop/etc/hadoop

```

ApplicationMaster:

`2g + 384mb` 向上取整 `3g`

Node HTTP Address	Last health-update	Health-report	Containers	Mem Used	Mem Avail	VCores Used	VCores Avail
<a href="#">hdp-01:8042</a>	Tue Jun 12 17:40:20 +0800 2018		1	3 GB	1 GB	1	3

每一个 executor: 指定的资源是 500m + 384m 向上取整 1g

- 7089	<a href="#">hdp-01:8042</a>	Tue Jun 12 17:40:20 +0800 2018	1	3 GB	1 GB	1	3
- 9624	<a href="#">hdp-02:8042</a>	Tue Jun 12 17:40:19 +0800 2018	1	1 GB	3 GB	1	3
- 2274	<a href="#">hdp-03:8042</a>	Tue Jun 12 17:40:19 +0800 2018	1	1 GB	3 GB	1	3

验证 client 模式的配置:

```
spark-submit --master yarn --deploy-mode client --driver-memory 2g --executor-memory 500m
--class org.apache.spark.examples.SparkPi
/root/apps/spark-2.2.0-bin-hadoop2.7/examples/jars/spark-examples_2.11-2.2.0.jar 1000
```

在 client 模式下, driver-memory 的配置没有生效。

实际上 ApplicationMaster(executorLauncher) 的容器使用的资源是 1g

512m + 384m 向上取整

Node HTTP Address	Last health-update	Health-report	Containers	Mem Used	Mem Avail	VCores Used	VCores Avail
<a href="#">hdp-01:8042</a>	Tue Jun 12 17:44:20 +0800 2018		1	1 GB	3 GB	1	3

512 和 384 都是从配置文件中得到的:

<http://spark.apache.org/docs/2.2.0/running-on-yarn.html>

spark.yarn.am.memory	512m	Amount of memory to use for the YARN Application Master in client mode, in the same format as JVM memory strings (e.g. 512m, 2g). In cluster mode, use spark.driver.memory instead. Use lower-case suffixes, e.g. k, m, g, t, and p, for kibi-, mebi-, gibi-, tebi-, and pebibytes, respectively.
spark.yarn.am.memoryoverhead	AM memory * 0.10, with minimum of 384	Same as spark.yarn.driver.memoryoverhead, but for the YARN Application Master in client mode.

通过 **--conf spark.yarn.am.memory=2g** 指定 client 模式下, ApplicationMaster 的资源分配

```
spark-submit --master yarn --deploy-mode client --conf spark.yarn.am.memory=2g
--executor-memory 500m --class org.apache.spark.examples.SparkPi
/root/apps/spark-2.2.0-bin-hadoop2.7/examples/jars/spark-examples_2.11-2.2.0.jar 1000
```

当指定 am.memory=2g 的时候, yarn 的容器的最小资源是默认值 1024Mb

启动容器所占用的资源为: 2g + 384m 取整 3g

<a href="#">hdp-03:8042</a>	Tue Jun 12 17:52:19 +0800 2018		1	3 GB	1 GB	1	3
-----------------------------	--------------------------------	--	---	------	------	---	---

yarn 的任务的调度:  
FIFO 已被废弃  
Capacity Scheduler  
公平调度器

spark standalone: application 的调用 FIFO

## 1.5. 总结 SparkOnYarn

1, spark 的任务运行在 yarn 资源调度平台上。

spark-submit --master yarn --deploy-mode cluster client yarn-client yarn-cluster

3, yarn 集群的配置, spark 的配置 (HADOOP\_CONF\_DIR/YARN\_CONF\_DIR)

3, 两种模式的区别: cluster, driver 端 (sparkContext 的初始化, dag 的生成, stage 的切分, task 的生成) cluster, driver 运行在 yarn 集群的某一个 container 中 (ApplicationMaster) client 模式下, driver 运行在 client 上。

实际工作中, 优先 cluster 模式, 自己测试, client 模式

4, 任务的资源可以自定义, --num-executors 可以指定任务使用的所有的 cores。 (可通过 spark-submit 来查看)

5, spark-shell, 只能运行在 client 模式下

## 2. Spark-HA 集群

secondaryname namenode edits (元素的日志文件) 合并的工作

HA: high available 高可用 双机热备 7\*24H 应用不能出问题

Master 节点存在单点故障, 要解决此问题, 就要借助 zookeeper, 并且启动至少两个 Master

节点来实现高可靠, 配置方式比较简单:

Spark 集群重新规划: (新增一台 master 节点)

hdp-01, hdp-02 是 Master;

hdp-02 hdp-03 hdp-04 起 Worker

安装配置 zk 集群，并启动 zk 集群,验证 zk 集群

```
[root@hdp-02 ~]# zkServer.sh status
JMX enabled by default
Using config: /root/apps/zookeeper-3.4.6/bin/../conf/zoo.cfg
Mode: follower
[root@hdp-02 ~]#
```

停止 spark 所有服务，在 master 节点上执行 (hdp-01)

```
# stop-all.sh
```

修改配置文件 spark-env.sh，在该配置文件中删掉 SPARK\_MASTER\_HOST 并添加如下配置

```
# vim /root/apps/spark/conf/spark-env.sh
```

```
export
```

```
SPARK_DAEMON_JAVA_OPTS="-Dspark.deploy.recoveryMode=ZOOKEEPER
```

```
-Dspark.deploy.zookeeper.url=hdp-01:2181,hdp-02:2181,hdp-03:2181
```

```
-Dspark.deploy.zookeeper.dir=/spark"
```

```
export JAVA_HOME=/usr/local/jdk
#export SPARK_MASTER_HOST=hdp-01
#export SPARK_MASTER_PORT=7077
export SPARK_DAEMON_JAVA_OPTS="-Dspark.deploy.recoveryMode=ZOOKEEPER -Dspark.deploy.zookeeper.url=hdp-01:2181,hdp-02:2181,hdp-03:2181 -Dspark.deploy.zookeeper.dir=/spark"
```

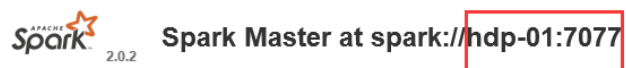
1. 在 hdp-01 节点上修改 slaves 配置文件内容指定 worker 节点

```
# vim /root/apps/spark/conf/slaves
```

```
# A spark worker will be started on each
hdp-01
hdp-02
hdp-03
hdp-04
hdp-05
```

2. 在 hdp-01 上执行 **start-all.sh** 脚本，然后在 hdp-02 上执行 **start-master.sh** 启动

## 第二个 Master



URL: spark://hdp-01:7077  
REST URL: spark://hdp-01:6066 (cluster mode)  
Alive Workers: 5  
Cores in use: 5 Total, 0 Used  
Memory in use: 13.6 GB Total, 0.0 B Used  
Applications: 0 Running, 0 Completed  
Drivers: 0 Running, 0 Completed  
Status: ALIVE

### Workers

Worker Id
worker-20171008030801-192.168.8.13-38105
worker-20171008030806-192.168.8.12-35019
worker-20171008040126-192.168.8.11-39825
worker-20171011104905-192.168.8.14-40103
worker-20171011112240-192.168.8.15-59582



URL: spark://hdp-02:7077  
REST URL: spark://hdp-02:6066 (cluster mode)  
Alive Workers: 0  
Cores in use: 0 Total, 0 Used  
Memory in use: 0.0 B Total, 0.0 B Used  
Applications: 0 Running, 0 Completed  
Drivers: 0 Running, 0 Completed  
Status: STANDBY

连接 spark 集群的时候，指定多台 master 即可。

```
spark-shell --master spark://hdp-01:7077,hdp-02:7077
```

部署：

zookeeper

修改 spark 的配置文件，改成连接 zookeeper 的地址

正常启动 spark 集群。在其他的一台机器上，单独启动 Master。（start-master.sh）

当 alive 状态的 master 挂掉之后，就会自动切换到 standby 状态的 master。

连接时，只需要指定多个 master 地址就可以了。

## 3.zookeeper

### 3.1.什么是 zookeeper

zookeeper 是一个分布式系统的基础协调服务

具体来说，zookeeper 的核心功能有如下几个：

- 1) 帮客户端管理数据（少量协调所需的数据——状态信息）
- 2) 帮客户端监听数据（如果数据发生变化，zookeeper 负责通知监听者）
- 3) 检测客户端是否失联，并可以自动删除失联者所注册的数据

### 3.2.zookeeper 的应用场景举例

hbase: master 如何动态感知各 regionserver 的上下线及负载状态；

spark: 两个 Master 节点的状态切换

### 3.3.zookeeper 自身的工作机制

zookeeper 一般是有奇数台服务器的，本身不存在单点故障。（3 台，5 台就能完全胜任成百上千台 hadoop，spark 集群的使用。）

会有一个 leader 选举机制，半数以上的票，就选举 leader 成功。

- 1) zookeeper 自身也是一个分布式系统，它会将用户的数据冗余存储在多台 zookeeper 的服务器中（磁盘、内存）
- 2) zookeeper 对内部数据一致性的保证，是通过 leader--> followers 模式同步
  - zookeeper 中有一台服务器是 leader 状态
  - 其他服务器是 follower 状态
  - 任何数据更新操作，都是先由 leader 实施，然后同步给其他 follower
- 3) zookeeper 的可靠性和可用性非常高，它不存在单点故障问题；
  - leader 是动态选举产生的；
  - （每台 zk 服务器，都有一个自己的 myid 号（1,2,3）

**模拟一下 zk 集群初次启动：**

id=1 的服务器启动了，它向局域网发送组播寻找 leader（端口 2888）。发现没有 leader，这台服务器就进入选举状态（3888 端口），并向局域网组播投票（投自己）；

id=2 的服务器启动了，它向局域网寻找 leader，发现没有，进入投票状态（3888 端口），收到服务器 1 所投的票；然后发现  $1 < 2$ ，投票（投 2）；此时，服务器 1 也会收到 2 的投票，发现  $2 > 1$ ，重新投（投 2）；此时，服务器 2 会收到两票；然后通过配置文件获知集群总共有 3 台机器，从而知道自己已经得多数票（当选）；服务器 2 就切换到 leader 状态（2888）；

id=3 的服务器启动了，它向局域网寻找 leader，发现服务器 2 是 leader，自己主动进入 follower 状态；

)

2181: 客户端连接的端口

2888: 寻找 leader 的端口

3888: 选举端口

### 3.4.zookeeper 的集群部署

1、上传安装包到集群服务器

2、解压

3、修改配置文件

进入 zookeeper 的安装目录的 conf 目录

```
# cp zoo_sample.cfg zoo.cfg
```

```
# vi zoo.cfg
```

```
# The number of milliseconds of each tick
tickTime=2000
initLimit=10
syncLimit=5
dataDir=/root/zkdata
clientPort=2181

#autopurge.purgeInterval=1
server.1=hdp-01:2888:3888
server.2=hdp-02:2888:3888
server.3=hdp-03:2888:3888
```

在 3 台节点上，都创建目录 `mkdir /root/zkdata`

在 3 台节点，在工作目录(/root/zkdata)中生成 myid 文件，但内容要分别为各自的 id: 1,2,3

**注意：1 2 3 和>之间有空格**

hdp-01 上: `echo 1 > /root/zkdata/myid`

hdp-02 上: `echo 2 > /root/zkdata/myid`

hdp-03 上: `echo 3 > /root/zkdata/myid`

可以在 hdp-01 上 执行命令, 在 hdp-02 和 hdp-03 上生成相应的配置文件

```
[root@hdp-01 ~]# ssh hdp-02 "mkdir /root/zkdata"
```

```
[root@hdp-01 ~]# ssh hdp-03 "mkdir /root/zkdata"
```

```
[root@hdp-01 ~]# ssh hdp-02 "echo 2 > /root/zkdata/myid"
```

```
[root@hdp-01 ~]# ssh hdp-03 "echo 3 > /root/zkdata/myid"
```

4、从 hdp-01 上 scp 安装目录到其他两个节点

```
cd /root/apps
```

```
scp -r zookeeper-3.4.6/ hdp-02:$PWD
```

```
scp -r zookeeper-3.4.6/ hdp-03:$PWD
```

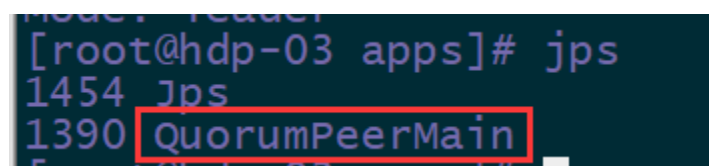
5、启动 zookeeper 集群

zookeeper 没有提供自动批量启动脚本, 需要手动一台一台地起 zookeeper 进程

在每一台节点上, 运行命令:

```
bin/zkServer.sh start
```

启动后, 用 jps 应该能看到一个进程: QuorumPeerMain



```
[root@hdp-03 apps]# jps
1454 jps
1390 QuorumPeerMain
```

但是, 光有进程不代表 zk 已经正常服务, 需要用命令检查状态:

```
bin/zkServer.sh status
```

能看到角色模式: 为 leader 或 follower, 即正常了。



```
Starting zookeeper ... already running a
[root@hdp-01 ~]# zkServer.sh status
JMX enabled by default
Using config: /root/apps/zookeeper-3.4.6
Mode: follower
[root@hdp-01 ~]#

Starting zookeeper ... already running a
[root@hdp-03 apps]# zkServer.sh status
JMX enabled by default
Using config: /root/apps/zookeeper-3.4.6
Mode: leader
[root@hdp-03 apps]#
```

### 3.5.zookeeper 的数据存储机制

#### 数据存储形式

zookeeper 中对用户的数据采用 kv 形式存储

只是 zk 有点特别:

key: 是以路径的形式表示的, 那就意味着, 各 key 之间有父子关系, 比如

/ 是顶层 key

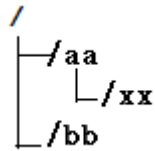
用户建的 key 只能在/ 下作为子节点, 比如建一个 key: /aa 这个 key 可以带 value 数据

也可以建一个 key: /bb

也可以建 key: /aa/xx

zookeeper 中, 对每一个数据 key, 称作一个 znode

综上所述, zk 中的数据存储形式如下:



## zookeeper 的命令行客户端操作

### 数据管理功能:

bin/zkCli.sh 命令进入交互式命令行

```
[root@hdp-03 ~]#  
[root@hdp-03 ~]# zkCli.sh
```

创建节点: `create /aaa 'ppppp'`

查看节点下的子节点: `ls /aaa`

获取节点的 value: `get /aaa`

修改节点的 value: `set /aaa 'mmmmm'`

删除节点: `rmr /aaa`

zookeeper 集群各机器时间必须同步:

`# ntpdate us.pool.ntp.org`      该命令作用: 利用 ntpdate 同步标准时间