**Установка Apache Spark**

**Установка Apache** Для простоты установки будет использоваться Ubuntu. Вначале нам необходимо установить репозитарии **Java**

$ sudo apt-add-repository ppa:webupd8team/java

$ sudo apt-get update

$ sudo apt-get install oracle-java7-installer

$ sudo apt-get update

Так же нам необходимо установить **git**

$ sudo apt-get install git

На этом подготовительные работы завершены и мы можем приступать к установки Apache Spark По шагам: 1) Переходим в домашний каталог

cd ~

2) Создадим директорию, куда будет скачен дистрибутив

mkdir spark  .&& cd spark/

wget <http://mirror.tcpdiag.net/apache/spark/spark-1.2.0/spark-1.2.0.tgz>

3) После скачивания распаковываем и

gunzip -c spark-1.2.0.tgz | tar -xvf —

cd spark-1.2.0/

4) Установка и копирование на «постоянное место жительства»

sudo sbt/sbt assembly

cd ..

sudo cp -Rp spark-1.2.0 /usr/local/

 cd /usr/local/

sudo ln -s spark-1.2.0 spark

Дабы не противоречить постулатам **Unix** не будем работать от **root** и создадим специального пользователя и ключи для него

sudo addgroup spark

sudo useradd -g spark spark

sudo adduser spark sudo

sudo mkdir /home/spark

sudo chown spark:spark /home/spark

Добавим в sudo

visudo

spark ALL=(ALL) NOPASSWD:ALL

sudo chown -R spark:spark /usr/local/spark/

Логинимся нашим пользователем и генерируем ключи

sudo su spark

ssh-keygen -t rsa -P «»

cat ~/.ssh/id\_rsa.pub >> ~/.ssh/authorized\_keys

Делаем служебные папки

sudo mkdir -p /srv/spark/{logs,work,tmp,pids}

sudo chown -R spark:spark /srv/spark

sudo chmod 4755 /srv/spark/tmp

Теперь нужно настроить Spark

cd /usr/local/spark/conf/

cp -p spark-env.sh.template spark-env.sh

vim spark-env.sh

export SPARK\_WORKER\_CORES="2"

export SPARK\_WORKER\_MEMORY="1g"

export SPARK\_DRIVER\_MEMORY="1g"

export SPARK\_REPL\_MEM="2g"

export SPARK\_WORKER\_PORT=9000

export SPARK\_CONF\_DIR="/usr/local/spark/conf"

export SPARK\_TMP\_DIR="/srv/spark/tmp"

export SPARK\_PID\_DIR="/srv/spark/pids"

export SPARK\_LOG\_DIR="/srv/spark/logs"

export SPARK\_WORKER\_DIR="/srv/spark/work"

export SPARK\_LOCAL\_DIRS="/srv/spark/tmp"

export SPARK\_COMMON\_OPTS="$SPARK\_COMMON\_OPTS -Dspark.kryoserializer.buffer.mb=32 "

LOG4J="-Dlog4j.configuration=file://$SPARK\_CONF\_DIR/log4j.properties"

export SPARK\_MASTER\_OPTS=" $LOG4J -Dspark.log.file=/srv/spark/logs/master.log "

export SPARK\_WORKER\_OPTS=" $LOG4J -Dspark.log.file=/srv/spark/logs/worker.log "

export SPARK\_EXECUTOR\_OPTS=" $LOG4J -Djava.io.tmpdir=/srv/spark/tmp/executor "

export SPARK\_REPL\_OPTS=" -Djava.io.tmpdir=/srv/spark/tmp/repl/\$USER "

export SPARK\_APP\_OPTS=" -Djava.io.tmpdir=/srv/spark/tmp/app/\$USER "

export PYSPARK\_PYTHON="/usr/bin/python"

SPARK\_PUBLIC\_DNS="DNS RECORD"

export SPARK\_WORKER\_INSTANCES=2

Далее

cp -p spark-defaults.conf.template spark-defaults.conf

vim spark-defaults.conf

spark.master spark://DNS RECORD:7077

spark.executor.memory 512m

spark.eventLog.enabled true

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

На этом установка закончена и можно запускать spark

cd /usr/local/spark/sbin

sbin/start-all.sh

Веб интерфейс со статистикой будет доступен по адресу

http://DNS RECORD:8080/

Чтобы меньше было отладочной информации необходимо сделать следующее

cd /usr/local/spark/conf

cp log4j.properties.template log4j.properties

И испрвить строку с

log4j.rootCategory=INFO, console

на 

log4j.rootCategory=ERROR, console

# Установка Apache Spark и IPython

## **Загрузка и установка Apache Spark**

Сначала загрузим и распакуем архив Spark.

$ cd $HOME

# Download the pre-built Spark tarball.

$ wget http://mirror.ox.ac.uk/sites/rsync.apache.org/spark/spark-1.4.0/spark-1.4.0-bin-hadoop2.6.tgz

# Extract the contents to your HOME folder.

$ tar -xvf spark-1.4.0-bin-hadoop2.6.tgz

# Remove the tarball. $ rm spark-1.4.0-bin-hadoop2.6.tgz

Дальше запускаем:

$ cd spark-1.4.0-bin-hadoop2.6

$ ./bin/run-example SparkPi 10

Вы должны увидеть, что то вроде этого:

Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties

15/08/2110:52:56 INFO SparkContext: Running Spark version 1.4.0

...

15/08/2110:53:02 INFO Executor: Finished task 0.0 in stage 0.0 (TID 0). 736 bytes result sent to driver

15/08/2110:53:02 INFO TaskSetManager: Starting task 1.0 in stage 0.0 (TID 1, localhost, PROCESS\_LOCAL, 1446 bytes)

15/08/2110:53:02 INFO Executor: Running task 1.0 in stage 0.0 (TID 1)

15/08/2110:53:02 INFO TaskSetManager: Finished task 0.0 in stage 0.0 (TID 0) in 953 ms on localhost (1/10)

15/08/2110:53:03 INFO Executor: Finished task 1.0 in stage 0.0 (TID 1). 736 bytes result sent to driver

...

15/08/2110:53:03 INFO Executor: Running task 9.0 in stage 0.0 (TID 9)

15/08/2110:53:03 INFO Executor: Finished task 9.0 in stage 0.0 (TID 9). 736 bytes result sent to driver

15/08/2110:53:03 INFO TaskSetManager: Finished task 9.0 in stage 0.0 (TID 9) in 16 ms on localhost (10/10)

15/08/2110:53:03 INFO DAGScheduler: ResultStage 0 (reduce at SparkPi.scala:35) finished in 1.457 s

15/08/2110:53:03 INFO TaskSchedulerImpl: Removed TaskSet 0.0, whose tasks have all completed, from pool

15/08/2110:53:03 INFO DAGScheduler: Job 0 finished: reduce at SparkPi.scala:35, took 1.991463 s

Прежде чем мы продолжим и приступим к настройке PySpark, Python и IPython мы добавим несколько переменных окружения в нашей системе где установлен Spark.

Откройте ваш bash\_profile командой:

$ nano ~/.bash\_profile

и добавьте строки:

# Set the Spark Home as an environment variable.

export SPARK\_HOME="$HOME/spark-1.4.0-bin-hadoop2.6

# Define your Spark arguments for when running Spark.

export PYSPARK\_SUBMIT\_ARGS="--master local[2]"

## **Оболочка PySpark**

Теперь у нас есть рабочий Spark, мы можем использовать Python. В комплекте со Spark идет pySpark оболочка. Запустите ее, используя:

$ ./bin/pyspark

После этого вы окажитесь внутри PySpark и сможете использовать Python для работы со Spark.

>>> sc

<pyspark.context.SparkContext object at 0x7f05502a1750>

data = [1, 2, 3, 4, 5]

dataRDD = sc.parallelize(data)

>>> dataRDD

ParallelCollectionRDD[0] at parallelize at PythonRDD.scala:396

## **PySpark — Конфигурация IPython**

Сначала установим virtualenv, что позволит нам работать в виртуальной среде.

# Install pip, the python package manager.

$ sudo apt-get install python-pip

# Install virtualenv

$ sudo pip install virtualenv

# Create a virtual environment.

$ virtualenv pyEnv

Теперь мы активируем это окружение, и в дальнейшем мы сможем устанавливать наши библиотеки Python внутри виртуальной среды.

$ source pyEnv/bin/activate

Далее устанавливаем IPython в виртуальной среде pyEnv. Для установки используйте следующую команду.

$ sudo pip install "ipython[notebook]"

Теперь у нас есть IPython установленный в нашей виртуальной среде. Следующим важным этапом является, настройка IPython таким образом, что бы он работал с ядром pySpark, и мы могли начать использовать Spark внутри IPython. Мы сделаем это путем создания профиля IPython специально для Spark.

# Create a new IPython profile.

$ ipython profile create pyspark

Теперь, когда мы создали реальный профиль pyspark для IPython нам нужно будет его настроить. Большинство настроек может быть сделано в файле ipython\_notebook\_config.py. Откройте этот файл (я использую nano для редактирования):

$ nano ~/.ipython/profile\_pyspark/ipython\_notebook\_config.py

Для начала изменим следующую строку:

# c.NotebookApp.ip = 'localhost'

c.NotebookApp.ip = '\*'

Так как мы работаем на виртуальном сервере, мы не хотим, чтобы IPython открывал браузер по умолчанию. Чтобы отключить это необходимо добавить следующую строку:

# c.NotebookApp.open\_browser = True

c.NotebookApp.open\_browser = False

IPython имеет порт по умолчанию всегда открытый для связи. Мы решили использовать другой порт. Вы можете изменить порт, добавив следующую строку:

# c.NotebookApp.port = 8888

c.NotebookApp.port = 8001

Конфигурация IPython закончена, но мы должны убедиться, что он на самом деле создает SparkContext при старте. Мы можем сделать это, изменив настройки файла IPython: 00-pyspark-setup.py. Откройте этот файл в текстовом редакторе:

$ nano ~/.ipython/profile\_pyspark/startup/00-pyspark-setup.py

и вставьте следующий скрипт на Python. Затем сохраните содержимое.

# Configure the necessary Spark environment

import os

import sys

# Set the spark\_home variable

SPARKHOME = os.environ.get('SPARK\_HOME', None)

sys.path.insert(0, SPARKHOME + "/python")

# Add the py4j to the path.

# You may need to change the version number to match your install

sys.path.insert(0, os.path.join(SPARKHOME, 'python/lib/py4j-0.8.2.1-src.zip'))

# Initialize PySpark to predefine the SparkContext variable 'sc'

execfile(os.path.join(SPARKHOME, 'python/pyspark/shell.py'))

Чтобы запустить IPython и он мог использовать Spark мы должны воспользоваться достаточно длинной командой, поэтому мы создадим псевдоним для нее в нашем .bash\_profile. Откройте профиль с помощью команды:

$ nano ~/.bash\_profile

и добавить следующие две строки:

# IPython alias for the use with SPARK.

alias IPYSPARK='PYSPARK\_DRIVER\_PYTHON=ipython PYSPARK\_DRIVER\_PYTHON\_OPTS="notebook --profile=pyspark --ip=0.0.0.0" $SPARK\_HOME/bin/pyspark'

После сохранения и закрытия убедитесь, что вы перезагрузили профиль, используя:

$ source ~/.bash\_profile

Теперь мы можем запустить IPython (использующий оболочку Spark), используя только что созданный псевдоним:

$ cd $HOME

$ IPYSPARK

# Конфигурирование Spark на YARN

* У Spark есть 3 варианта работать на кластере: standalone, с использованием Mesos и с использованием YARN. Мы решили выбрать третий вариант, потому что для нас он был логичен. У нас уже есть hadoop-кластер. Наши участники хорошо уже знакомы с его архитектурой. Давайте юзать YARN.

spark.master=yarn

Далее интереснее. У каждого из этих 3 вариантов развертывания есть 2 варианта деплоя: client и cluster. Исходя из [документации](http://spark.apache.org/docs/latest/running-on-yarn.html) и разных ссылок в интернете, можно сделать вывод, что client подходит для интерактивной работы — например, через jupyter notebook, а cluster больше подходит для production-решений. В нашем случае нас интересовала интерактивная работа, поэтому:

spark.deploy-mode=client

В общем-то с этого момента Spark уже будет как-то работать на YARN, но нам этого не было достаточно. Поскольку у нас программа про большие данные, то порой участникам не хватало того, что получалось в рамках равномерной нарезки ресурсов. И тут мы нашли интересную вещь — динамическую аллокацию ресурсов. Если коротко, то суть в следующем: если у вас тяжелая задача и кластер свободен (например, с утра), то при помощи этой опции Spark вам может выдать дополнительные ресурсы. Необходимость считается там по хитрой формуле. Вдаваться в подробности не будем — она неплохо работает.

spark.dynamicAllocation.enabled=true

Мы поставили этот параметр, и при запуске Spark ругнулся и не запустился. Правильно, потому что надо было читать [документацию](http://spark.apache.org/docs/latest/configuration.html) внимательнее. Там указано, что для того, чтобы все было ок, нужно еще включить дополнительный параметр.

spark.shuffle.service.enabled=true

Зачем он нужен? Когда наш джоб больше не требует такого количества ресурсов, то Spark должен вернуть их в общий пул. Самая трудозатратная стадия почти в любой MapReduce задаче — это стадия Shuffle. Этот параметр позволяет сохранять данные, которые образуются на этой стадии и соответственно освобождать executors. А executor — это процесс, который на воркере всё обсчитывает. У него есть какое-то количество процессорных ядер и какое-то количество памяти.  
  
Добавили этот параметр. Всё вроде бы заработало. Стало заметно, что участникам реально стало выдаваться больше ресурсов, когда им было нужно. Но возникла другая проблема — в какой-то момент другие участники просыпались и тоже хотели использовать Spark, а там всё занято, и они были недовольны. Их можно понять. Стали смотреть в документацию. Там оказалось, что есть еще какое-то количество параметров, при помощи которых можно повлиять на процесс. Например, если executor находится в режиме ожидания — через какое время у него можно забрать ресурсы?

spark.dynamicAllocation.executorIdleTimeout=120s

В нашем случае — если ваши executors ничего не делают в течение двух минут, то, будьте добры, верните их в общий пул. Но и этого параметра не всегда хватало. Было видно, что человек уже давно ничего не делает, а ресурсы не освобождаются. Оказалось, что есть еще специальный параметр — по прошествии какого времени отбирать executors, которые содержат закэшированные данные. По дефолту этот параметр стоял — infinity! Мы его поправили.

spark.dynamicAllocation.cachedExecutorIdleTimeout=600s

То есть если в течение 5 минут ваши executors ничего не делают, отдайте-ка их в общий пул. В таком режиме скорость освобождения и выдачи ресурсов для большого количества пользователей стала достойной. Количество недовольства сократилось. Но мы решили пойти дальше и ограничить максимальное количество executors на один application — по сути на одного участника программы.

spark.dynamicAllocation.maxExecutors=19

Теперь, конечно, появились недовольные с другой стороны — “кластер простаивает, а у меня всего лишь 19 executors”, но что поделать — нужен какой-то правильный баланс. Всех сделать счастливыми не получится.  
  
И еще одна небольшая история, связанная со спецификой нашего кейса. Как-то на практическое занятие опоздали несколько человек, и у них Spark почему-то не стартовал. Мы посмотрели на количество свободных ресурсов — вроде бы есть. Spark должен стартовать. Благо, что к тому моменту документация уже где-то записалась на подкорку, и мы вспомнили, что при запуске Spark ищет себе порт, на котором стартовать. Если первый порт из диапазона занят, то он переходит к следующему по порядку. Если он свободен, то захватывает. И есть параметр, который указывает на максимальное количество попыток для этого. По умолчанию — это 16. Число меньше, чем людей в нашей группе на занятии. Соответственно, после 16 попыток Spark бросал это дело и говорил, что не могу стартануть. Мы поправили этот параметр.

spark.port.maxRetries=50

Дальше расскажу о некоторых настройках, уже не сильно связанных со спецификой нашего кейса.  
  
Для более быстрого старта Spark есть рекомендация папку jars, лежащую в домашней директории SPARK\_HOME, заархивировать и положить на HDFS. Тогда он не будет тратить времени на загрузку этих джарников по воркерам.

spark.yarn.archive=hdfs:///tmp/spark-archive.zip

Также для более быстрой работы рекомендуется в качестве сериалайзера использовать kryo. Он более оптимизированный, чем тот, что по умолчанию.

spark.serializer=org.apache.spark.serializer.KryoSerializer

И есть еще давняя проблема Spark, что он часто валится по памяти. Часто это происходит в тот момент, когда воркеры всё посчитали и отправляют результат на драйвер. Мы сделали себе этот параметр побольше. По умолчанию, он 1Гб, мы сделали — 3.

spark.driver.maxResultSize=3072

И последнее, в качестве десерта. Как обновить Spark до версии 2.1 на HortonWorks дистрибутиве — HDP 2.5.3.0. Эта версия HDP содержит в себе предустановленную версию 2.0, но мы как-то однажды для себя решили, что Spark довольно активно развивается, и каждая новая версия фиксит какие-то баги плюс дает дополнительные возможности, в том числе и для python API, поэтому решили, что нужно делать апдейт.  
  
Скачали версию с официального сайта под Hadoop 2.7. Разархивировали, закинули в папку с HDP. Поставили симлинки как надо. Запускаем — не стартует. Пишет очень непонятную ошибку.

java.lang.NoClassDefFoundError: com/sun/jersey/api/client/config/ClientConfig

Погуглив, выяснили, что Spark решил не ждать пока Hadoop разродится, и решили использовать новую версию jersey. Они сами там друг с другом ругаются на эту тему в JIRA. Решением было — скачать [jersey версии 1.17.1](https://mvnrepository.com/artifact/com.sun.jersey/jersey-bundle/1.17.1). Закинуть это в папку jars в SPARK\_HOME, снова сделать zip и закинуть на HDFS.  
  
Эту ошибку мы обошли, но возникла новая и довольно-таки обтекаемая.

org.apache.spark.SparkException: Yarn application has already ended! It might have been killed or unable to launch application master

При этом пробуем запускать версию 2.0 — всё ок. Попробуй догадайся, в чем дело. Мы залезли в логи этого application и увидели что-то такое:

/usr/hdp/${hdp.version}/hadoop/lib/hadoop-lzo-0.6.0.${hdp.version}.jar

В общем, по каким-то причинам hdp.version не резолвилась. Погуглив, нашли решение. Нужно в Ambari зайти в настройки YARN и добавить там параметр в custom yarn-site:

hdp.version=2.5.3.0-37

Эта магия помогла, и Spark взлетел. Протестили несколько наших jupyter-ноутбуков. Всё работает. К первому занятию по Spark в субботу (уже завтра) мы готовы!  
  
**UPD**. На занятии выяснилась еще одна проблема. В какой-то момент YARN перестал выдавать контейнеры для Spark. В YARN нужно было поправить параметр, который по дефолту стоял 0.2:

yarn.scheduler.capacity.maximum-am-resource-percent=0.8

То есть только 20% ресурсов участвовали в раздаче ресурсов. Поменяв параметры, перезагрузили YARN. Проблема была решена, и остальные участники тоже смогли запустить spark context.