Apache Spark

https://spark.apache.org/

Spark – это

Инструмент для «молниеносных кластерных вычислений»
Быстрая и универсальная платформа для обработки данных

Лаконичность:

```
sparkContext.textFile("hdfs://...")
    .flatMap(line => line.split(" "))
    .map(word => (word, 1)).reduceByKey(_ + _)
    .saveAsTextFile("hdfs://...")
```

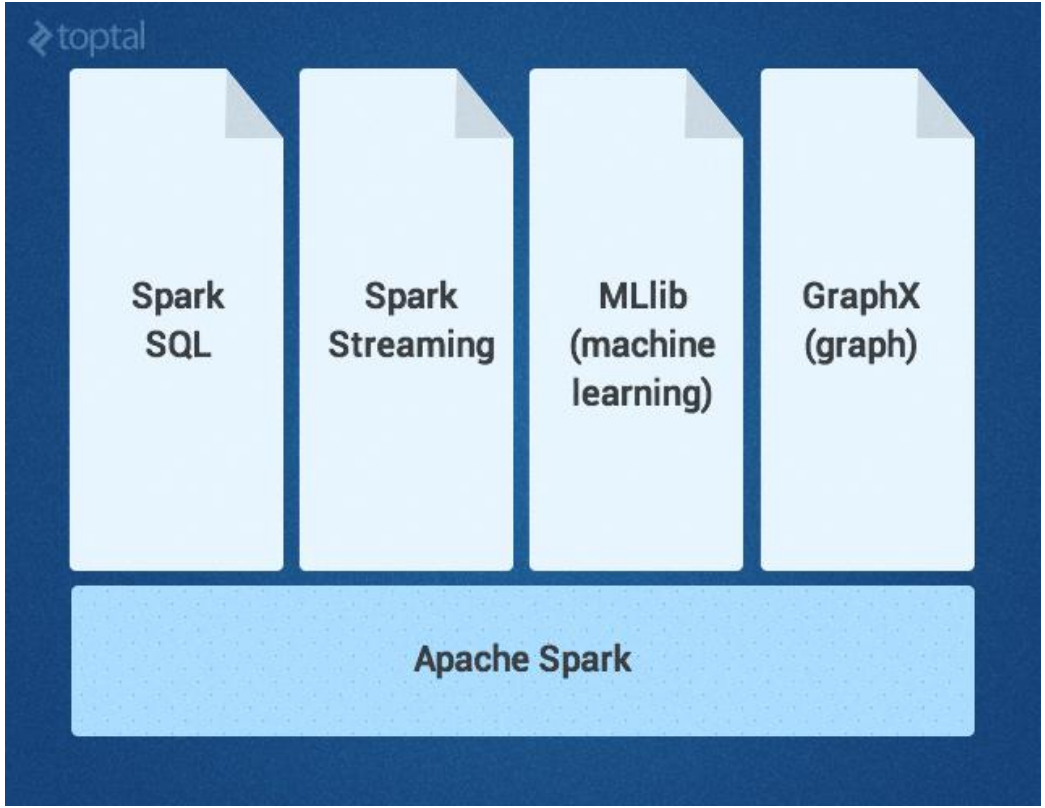
API: Scala, Java и Python

Источники данных: HDFS, Amazon S3, Hive, HBase, Cassandra, etc.

Может работать на кластерах или автономно

Дополнения:
SparkSQL,
Spark Streaming,
MLlib (машинное обучение)
GraphX

Spark – это


toptal
Spark SQL
Spark Streaming
MLlib (machine learning)
GraphX (graph)
Apache Spark

Основные понятия

Драйвер - основа приложения Spark:

- Сохраняет и обрабатывает информацию о состоянии приложения
- Отвечает на запросы пользовательских программ
- Анализирует и распределяет задачи между **исполнителями** и порядок их выполнения.

Исполнители взамен выполняют **задачи** и отчитываются об их выполнении и свое состояние **драйвера**. **Драйвер** и **исполнители** являются обычными процессами,

Spark DataFrame

- Содержит таблицу состоящая из строк и колонок
- Список колонок и их типов называют схеме
- Аналогичные структуры данных является в языке **R** и библиотеке **pandas**, но в Spark он **распределенный** на несколько **разделов** и **неизменяемый**.

Раздел - это набор строк DataFrame на одной физической машине кластера

Основные понятия

DataFrame в Spark - неизменны, но к нему можно применить преобразования, чтобы создать новый DataFrame:

```
evenRows = myData.where("number % 2 = 0")
```

Преобразование ленивы и не выполняются, а только добавляются в план вычислений до тех пор, пока пользователь не попросит выполнить действие (расчет, сохранение данных и т.д.)

Действие запускает **задачи** (англ. Job), которые выполняют все необходимые преобразования и действия согласно оптимизированного плана вычислений.

Процесс выполнения задачи можно мониторить с веб-интерфейса Spark, который находится по адресу http://localhost:4040 / в локальном режиме, или на узле кластера, на котором запущен драйвер.

RDD - устойчивый распределенный набор данных

Разделенная на части коллекция записей, доступная только для чтения

Создается путем загрузки внешнего набора данных или распределения коллекции из основной программы (driver program)

поддерживаются операции

- Трансформации (отображение, фильтрация, объединение и т.д.), результат трансформации - новый RDD. Выполняются в «ленивом» режиме - только после вызова действия.
- Действия (редукция, подсчет и т.д.), возвращают значение.

Популярные трансформации

.map(function) — применяет функцию function к каждому элементу датасета

.filter(function) — возвращает все элементы датасета, на которых функция function вернула истинное значение

.distinct([numTasks]) — возвращает датасет, который содержит уникальные элементы исходного датасета

.union(otherDataset)

.intersection(otherDataset)

.cartesian(otherDataset) — новый датасет содержит в себе всевозможные пары (A,B), где первый элемент принадлежит исходному датасету, а второй — датасету-аргументу

Популярные действия

.saveAsTextFile(path) — сохраняет данные в текстовый файл (в hdfs, на локальную машину или в любую другую поддерживаемую файловую систему — полный список можно посмотреть в документации)

.collect() — возвращает элементы датасета в виде массива. Как правило, когда данных в датасете уже мало

.take(n) — возвращает в виде массива первые n элементов датасета

.count() — возвращает количество элементов в датасете

.reduce(function) — функция function (которая принимает на вход 2 аргумента возвращает одно значение) должна быть обязательно коммутативной и ассоциативной

SparkSQL

- работает поверх Spark Core
- обеспечивает абстракцию над данными - DataFrames
- поддерживает структурированные и полуструктурированные данные
- Предоставляет предметно-ориентированный язык (DSL) для манипуляции DataFramesв Scala, Java, или Python
- Добавляет поддержку языка SQL, с интерфейсом командной строки и ODBC / JDBC серверами
- начиная с Spark 2.0, Spark SQL также поддерживает строго типизированный DataSet.

```
// sc – это существующий SparkContext.
val sqlContext = new org.apache.spark.sql.hive.HiveContext(sc)

sqlContext.sql("CREATE TABLE IF NOT EXISTS src (key INT, value STRING)")
sqlContext.sql("LOAD DATA LOCAL INPATH 'examples/src/main/resources/kv1.txt' INTO TABLE src")

// Запросы формулируются на HiveQL
sqlContext.sql("FROM src SELECT key, value").collect().foreach(println)



myDataFrame.createOrReplaceTempView("название_таблицы")
spark.sql("SELECT ... FROM название_таблицы ...")
```

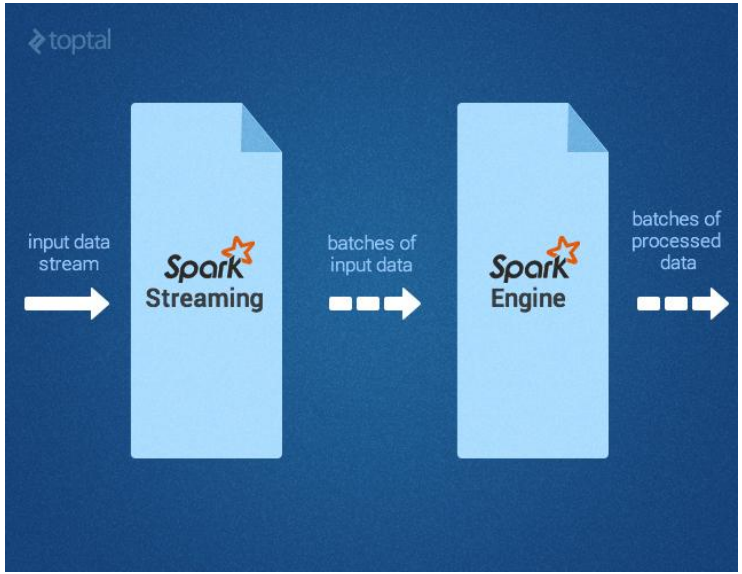
SparkSQL - пример

```
import org.apache.spark.sql.SQLContext

val url = "jdbc:mysql://yourIP:yourPort/test?user=yourUsername;password=yourPassword" // URL
для сервера бази даних.
val sqlContext = new org.apache.spark.sql.SQLContext(sc) // создать объект sql контекста

val df = sqlContext
  .read
  .format("jdbc")
  .option("url", url)
  .option("dbtable", "people")
  .load()

df.printSchema() // Показує схему DataFrame.
val countsByAge = df.groupBy("age").count() // Сосчитать количество людей каждого возраста
```

Spark Streaming

обработка потоковых данных в реальном времени

```
TwitterUtils.createStream(...)
        .filter(_.getText.contains("earthquake") || _.getText.contains("shaking"))
```



MLlib - библиотека для машинного обучения

```
// Готовим данные о твитах, касающихся землетрясения, и загружаем их в формате LIBSVM
val data = MLUtils.loadLibSVMFile(sc, "sample_earthquate_tweets.txt")

val splits = data.randomSplit(Array(0.6, 0.4), seed = 11L)
val training = splits(0).cache()
val test = splits(1)

val numIterations = 100
val model = SVMWithSGD.train(training, numIterations)

model.clearThreshold()

val scoreAndLabels = test.map { point =>
  val score = model.predict(point.features)
  (score, point.label)
}

val metrics = new BinaryClassificationMetrics(scoreAndLabels)
val auROC = metrics.areaUnderROC()

println("Area under ROC = " + auROC)
```

Разбиваем данные на тренировочные (60%) и тестовые (40%).

обучаем модель

Очищаем пороговое значение, заданное по умолчанию

Предсказываем

Показатели качества

GraphX - работа с графами

```
import org.apache.spark._
import org.apache.spark.rdd.RDD
import org.apache.spark.graphx._

val vertices=Array((1L, ("SFO")),(2L,("ORD")),(3L,("DFW")))
val vRDD= sc.parallelize(vertices)
val nowhere = "nowhere" // Defining a default vertex called nowhere

val edges = Array(Edge(1L,2L,1800),Edge(2L,3L,800),Edge(3L,1L,1400))
val eRDD= sc.parallelize(edges)

val graph = Graph(vRDD,eRDD, nowhere) // создать граф

val numairports = graph.numVertices   // сколько вершин
val numroutes = graph.numEdges        // сколько дуг

// выборка дуг
graph.edges.filter { case Edge(src, dst, prop) => prop > 1000 }

// сортировка дуг
graph.triplets.sortBy(_.attr, ascending=false).map(triplet =>
     "Distance " + triplet.attr.toString + " from " + triplet.srcAttr + " to " +
triplet.dstAttr + ".").collect.foreach(println)
```

Примеры

Способы запуска задач

Локально на 8 ядрах

```
./bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master local[8] \
  /path/to/examples.jar \
  100
```

Удаленно на кластере YARN

```
export HADOOP_CONF_DIR=XXX
./bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master yarn \
  --deploy-mode cluster \  # can be client for
client mode
  --executor-memory 20G \
  --num-executors 50 \
  /path/to/examples.jar \
  1000
```

Удаленно на кластере в режиме клиента

```
./bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master spark://207.184.161.138:7077 \
  --executor-memory 20G \
  --total-executor-cores 100 \
  /path/to/examples.jar \
  1000
```

Удаленно на кластере в режиме клиента с супервизором

```
./bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master spark://207.184.161.138:7077 \
  --deploy-mode cluster \
  --supervise \
  --executor-memory 20G \
  --total-executor-cores 100 \
  /path/to/examples.jar \
  1000
```

Быстрый старт

setup.py:

```
install_requires=[
        'pyspark=={site.SPARK_VERSION}'
]
```

SimpleApp.py:

```
from pyspark.sql import SparkSession
logFile = "YOUR_SPARK_HOME/README.md"
spark = SparkSession.builder.appName("SimpleApp").getOrCreate()
logData = spark.read.text(logFile).cache()
numAs = logData.filter(logData.value.contains('a')).count()
numBs = logData.filter(logData.value.contains('b')).count()
print("Lines with a: %i, lines with b: %i" % (numAs, numBs))
spark.stop()
```

```
$ YOUR_SPARK_HOME/bin/spark-submit --master local[4] SimpleApp.py
```

Скачать HADOOP
Распаковать в D:\bin\hadoop
Поставить HADOOP_HOME=D:\bin\hadoop

Чтение JSON

object
{"user_name":"Neutral player","x":27224,"y":79658,"user_alliance":null,"map":"tutorial"}
{"user_name":"Neutral player","x":31207,"y":71163,"user_alliance":null,"map":"tutorial"}
{"user_name":"maz2","x":6399,"y":66179,"user_alliance":"tutorial","map":"tutorial"}
{"user_name":"Neutral player","x":6113,"y":66325,"user_alliance":null,"map":"tutorial"}
{"user_name":"fuchs","x":52500,"y":32679,"unit_param_id":15,"user_alliance":"GAIA","map":"2"}

```
from pyspark.sql import SparkSession
logFile = "D:/data/proj/sandbox/spark/data/battle_reports.csv"
spark = SparkSession.builder.appName("Python Spark SQL basic example") \
        .config("spark.some.config.option", "some-value").getOrCreate()
df = spark.read.json(logFile)
df.show()
spark.stop()
```

Запросы - printSchema()

```
from pyspark.sql import SparkSession
logFile = "D:/data/proj/sandbox/spark/data/battle_reports.csv"
spark = SparkSession.builder.appName("Python Spark SQL basic example") \
    .config("spark.some.config.option", "some-value").getOrCreate()

df = spark.read.json(logFile)
df.printSchema()
```

```
#root
# |-- _corrupt_record: string (nullable = true)
# |-- ihb: boolean (nullable = true)
# |-- killer_unit_param_id: long (nullable = true)
# |-- map: string (nullable = true)
# |-- unit_param_id: long (nullable = true)
# |-- user_alliance: string (nullable = true)
# |-- user_name: string (nullable = true)
# |-- x: long (nullable = true)
# |-- y: long (nullable = true)
```

Запросы - выбранные колонки

```
from pyspark.sql import SparkSession
logFile = "D:/data/proj/sandbox/spark/data/battle_reports.csv"
spark = SparkSession.builder.appName("Basic example") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()
df = spark.read.json(logFile)
df.select(['user_name']).distinct().show()
```

```
#+------------+
#|   user_name|
#+------------+
#|       fuchs|
#|      rela41|
#|  andretest4|
#|   max-power|
#|        foxy|
#|     djbomba|
```

Запросы - distinct()

```
from pyspark.sql import SparkSession
logFile = "D:/data/proj/sandbox/spark/data/battle_reports.csv"
spark = SparkSession.builder.appName("Python Spark SQL basic example") \
    .config("spark.some.config.option", "some-value").getOrCreate()
df = spark.read.json(logFile)
df.select(df['user_alliance']).distinct().show()
```

```
#+---------------+
#|  user_alliance|
#+---------------+
#|tester-alliance|
#|     Timberclub|
#|           ANTS|
#|     Nebee_only|
#|          simon|
#|     0_tutorial|
```

Запросы - условие

```
from pyspark.sql import SparkSession
logFile = "D:/data/proj/sandbox/spark/data/battle_reports.csv"
spark = SparkSession.builder.appName("Python Spark SQL basic example") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()
df = spark.read.json(logFile)
df.filter(df['map']!='tutorial').show()
```

```
#+---------------+----+-------------------+---+------------+------------+-------------+-----+-----
+
#|_corrupt_record| ihb|killer_unit_param_id|map|unit_param_id|user_alliance|    user_name|    x|
y|
#+---------------+----+-------------------+---+------------+------------+-------------+-----+-----
+
#|           null|null|               null|  8|        null|        null|Neutral player|25715|20897|
#|           null|null|               null|  8|        null|        null|Neutral player|27297|24139|
#|           null|null|               null|  1|        null|        null|Neutral player|75234| 5666|
```

Запросы - группировка

```
from pyspark.sql import SparkSession
logFile = "D:/data/proj/sandbox/spark/data/battle_reports.csv"
spark = SparkSession.builder.appName("Basic example")\
          .config("spark.some.config.option", "some-value")\
          .getOrCreate()
df = spark.read.json(logFile)
df.filter(df['map']!='tutorial') \
  .groupBy('user_alliance').count() \
  .withColumnRenamed("count","nAttacks") \
  .orderBy(['nAttacks'], ascending=[0]).show()
```

```
# +---------------+--------+
# |  user_alliance|nAttacks|
# +---------------+--------+
# |       EROBERER|  107251|
# |           GAIA|   92094|
# |           null|   40423|
# |          GSG-9|   21550|
# |      DESTROYER|   12538|
# |            SEK|   11687|
# |           FAIR|    9835|
# |            BWT|    5915|
# |  gjDjg_ghj0jg1|    2101|
```

Запросы - join

```
val bikeStations = sqlContext.sql("SELECT * FROM sf_201508_station_data")
val tripData = sqlContext.sql("SELECT * FROM sf_201508_trip_data")

val justStations = bikeStations
  .selectExpr("float(station_id) as station_id", "name")
  .distinct()

val completeTripData = tripData
  .join(justStations, tripData("Start Station") === bikeStations("name"))
  .withColumnRenamed("station_id", "start_station_id")
  .drop("name")
  .join(justStations, tripData("End Station") === bikeStations("name"))
  .withColumnRenamed("station_id", "end_station_id")
  .drop("name")
```

RDD - примеры

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("Python Spark create RDD example") \
          .config("spark.some.config.option", "some-value").getOrCreate()

df = spark.sparkContext.parallelize([(1, 2, 3, 'a b c'),
          (4, 5, 6, 'd e f'),
          (7, 8, 9, 'g h i')]).toDF(['col1', 'col2', 'col3','col4'])

Employee = spark.createDataFrame(
[('1', 'Joe',   '70000', '1'),('2', 'Henry', '80000', '2'), ('3', 'Sam',   '60000', '2'), ('4', 'Max',   '90000', '1')],
['Id', 'Name', 'Sallary','DepartmentId'] )

df = spark.read.format('com.databricks.spark.csv').options(header='true', inferschema='true').\
      load("/home/feng/Spark/Code/data/Advertising.csv",header=True)
```

RDD - примеры

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("Python Spark create RDD example") \
        .config("spark.some.config.option", "some-value").getOrCreate()

## User information
user = 'your_username'
pw   = 'your_password'

## Database information
table_name = 'table_name'
url = 'jdbc:postgresql://##.###.###.##:5432/dataset?user='+user+'&password='+pw
properties ={'driver': 'org.postgresql.Driver', 'password': pw,'user': user}

df = spark.read.jdbc(url=url, table=table_name, properties=properties)
```

RDD - примеры

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("Python Spark create RDD example") \
        .config("spark.some.config.option", "some-value").getOrCreate()

## User information
user = 'your_username'
pw   = 'your_password'

## Database information
table_name = 'table_name'
url = 'jdbc:postgresql://##.###.###.##:5432/dataset?user='+user+'&password='+pw
properties ={'driver': 'org.postgresql.Driver', 'password': pw,'user': user}

df = spark.read.jdbc(url=url, table=table_name, properties=properties)
```

Советы для ускорения вычислений

Используйте Spark DataFrames

Если используете RDD:

Не вызывайте collect() на больших RDD / лучше take(), takeSample(), countByKey(), countByValue() или collectAsMap()

Уменьшайте RDD перед вызовом join

Избегайте groupByKey() на больших RDD / лучше reduceByKey(), combineByKey() or foldByKey()

Используйте broadcast для небольших наборов данных

Избегайте flatmap(), join() и groupBy()

Данные из SQL

```
url =  "jdbc:mysql://yourIP:yourPort/test?user=yourUsername;password=yourPassword"
df = sqlContext.read.format("jdbc").option("url", url).option("dbtable", "people").load()

# Посмотреть схему
df.printSchema()

# Сосчитать людей по возрасту
countsByAge = df.groupBy("age").count()
countsByAge.show()

# Сохранить в S3 как JSON
countsByAge.write.format("json").save("s3a://...")
```

Данные из HDFS

```
textFile = sc.textFile("hdfs://...")

# Создать DataFrame с одной колонкой "line"
df = textFile.map(lambda r: Row(r)).toDF(["line"])
errors = df.filter(col("line").like("%ERROR%"))

# Сосчитать строки
errors.count()

# Сосчитать строки с упоминанием “MySQL”
errors.filter(col("line").like("%MySQL%")).count()

# Преобразовать в массив строк
errors.filter(col("line").like("%MySQL%")).collect()
```

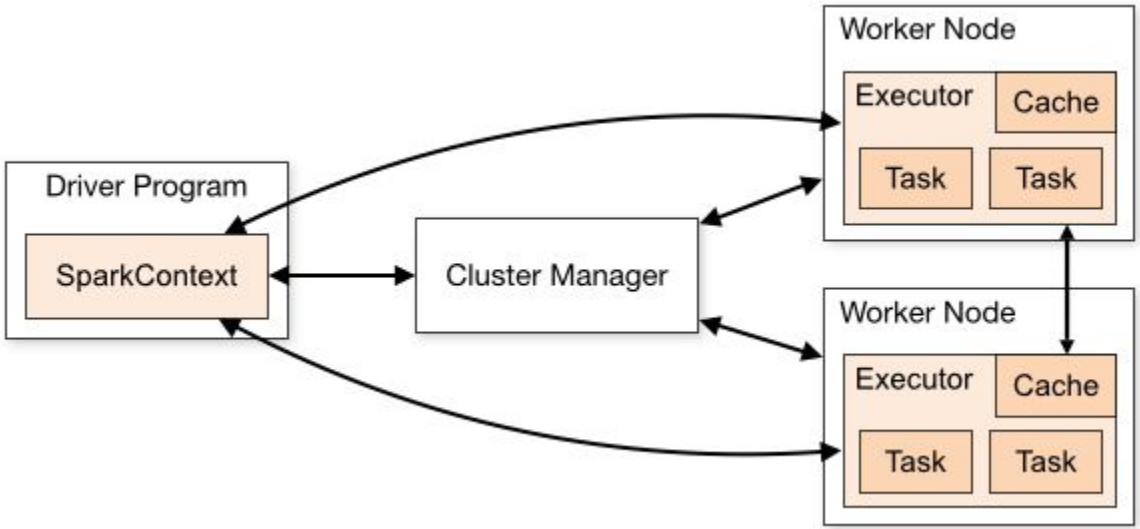
```
text_file = sc.textFile("hdfs://...")
counts = text_file.flatMap(lambda line:
line.split(" ")) \
          .map(lambda word: (word, 1)) \
          .reduceByKey(lambda a, b: a + b)
counts.saveAsTextFile("hdfs://...")
```

Настройка кластера

Компоненты

Spark приложение запускается как массив независимых процессов на кластере, координацией которых занимается объект SparkContext в вашей программе(называется она driver program).

Компоненты


Worker Node
Executor
Cache
Task
Task
Driver Program
SparkContext
Cluster Manager
Worker Node
Executor
Cache
Task
Task

Менеджеры кластеров

Выделяет ресурсы для приложений

- Автономный(Standalone) - Простой менеджер кластера входящий в Spark позволяет упростить установку кластера
- Apache Mesos - основный менеджер кластера что также может запускать Hadoop MapReduce задачи
- Hadoop YARN - ресурс менеджер в Hadoop 2

Единожды присоединившись, Spark получает исполнителей (executors) на ноде в кластере, которые обрабатывают вычисления и сохраняют данные для вашего приложения. В итоге, SparkContext отсылает задачи (tasks) к исполнителям для обработки.

Мониторинг

Каждая driver программа имеет web UI, обычно доступна по порту 4040 и отображает информацию о запущеных тасках, исполнителях, использовании хранилища.

Установка кластера

$ sudo vim /etc/hosts
<MASTER-IP> master
<SLAVE01-IP> slave01
<SLAVE02-IP> slave02

$ sudo apt-get install python-software-properties
$ sudo add-apt-repository ppa:webupd8team/java
$ sudo apt-get update
$ sudo apt-get install oracle-java7-installer
$ sudo apt-get install scala
$ sudo apt-get install openssh-server openssh-client

$ ssh-keygen -t rsa -P ""
Скопировать содержимое .ssh/id_rsa.pub (на master) в .ssh/authorized_keys (на всех slave и master) и проверить:
$ ssh slave01
$ ssh slave02

Установка кластера

$ wget http://www-us.apache.org/dist/spark/spark-2.3.0/spark-2.3.0-bin-hadoop2.7.tgz
$ tar xvf spark-2.3.0-bin-hadoop2.7.tgz
$ sudo mv spark-2.3.0-bin-hadoop2.7 /usr/local/spark
$ sudo vim ~/.bashrc
export PATH = $PATH:/usr/local/spark/bin
$ source ~/.bashrc

Установка кластера

Настроить spark-env.sh
$ cd /usr/local/spark/conf
$ cp spark-env.sh.template spark-env.sh

$ sudo vim spark-env.sh
export SPARK_MASTER_HOST='<MASTER-IP>'
export JAVA_HOME=<Path_of_JAVA_installation>

Добавить Workers
$ sudo vim /usr/local/spark/conf/slaves
И добавить строки.

master
slave01
slave02

Старт
$ cd /usr/local/spark
$ ./sbin/start-all.sh

Остановка
$ cd /usr/local/spark
$ ./sbin/stop-all.sh

Статус
$ jps

Веб-интерфейс
http://<MASTER-IP>:8080/