

2019

2019年4月13-14日 · 北京

Python开发者日

让开发者紧跟技术潮流

CSDN

Spark With Python应用



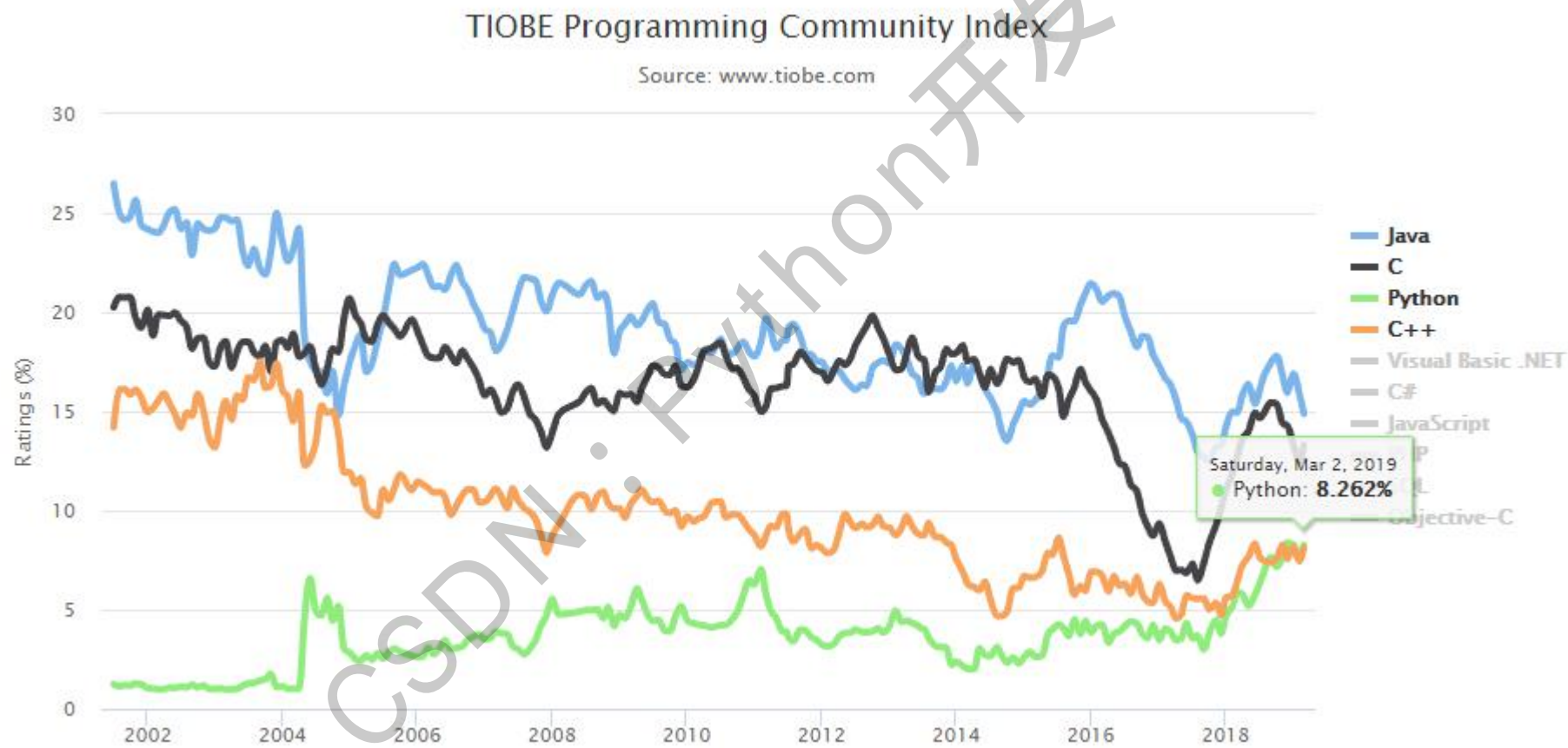
流行的Python
了解PySpark
PySpark Application运行原理
PySpark的缺点
YARN集群配置Python环境
使用 PySpark Shell
.....

流行的 Python

TIOBE社区编程语言排名（2019.03）

Mar 2019	Mar 2018	Change	Programming Language	Ratings	Change
1	1		Java	14.880%	-0.06%
2	2		C	13.305%	+0.55%
3	4	^	Python	8.262%	+2.39%
4	3	v	C++	8.126%	+1.67%
5	6	^	Visual Basic .NET	6.429%	+2.34%
6	5	v	C#	3.267%	-1.80%
7	8	^	JavaScript	2.426%	-1.49%
8	7	v	PHP	2.420%	-1.59%
9	10	^	SQL	1.926%	-0.76%
10	14	^^	Objective-C	1.681%	-0.09%

排名前4的编程语言增长趋势



在哪里使用Python?

- 多年来，Python一直处于各种流行编程语言排名的前列。该语言几乎可以用于任何事情。它旨在提高程序员的生产力，而不是他们编写的代码本身。
- Python可以用于Web、桌面应用程序开发、自动化脚本、复杂计算系统、科学计算、生命支持管理系统、物联网、游戏、机器人、自然语言处理等等。
- 近20年来，c、c++ 和 java 一直排在前三名，远远领先于其他部分。python 现在正在加入这三种语言，成为大型编程语言的一部分。
- 不管你喜欢与否，Python一直是最流行的编程语言之一。

Python的流行趋势

- Python 最流行的 3 个应用场景是：数据处理、网络应用的后端编写和自动化脚本。
- 在科学计算和数据科学领域，人们更喜欢Python工具，许多基于MATLAB、R或Mathematica的传统应用都迁移到Python之上了。
- Python虽然是解释型语言，大家会觉得效率低，但是很多需要大量计算的复杂运算都是通过库来完成的，Python只是完成调用，所以所谓的效率并不能阻碍Python的流行。
- Python可以便捷地和C/C++进行交互，这样人们就可以使用C/C++的高性能工具包。
- 由于Python极高的开发效率，逐渐得到了IT企业的青睐，会有越来越多的公司选用Python进行网站Web、搜索引擎（Google）、云计算（OpenStack）、大数据、人工智能（AlphaGo）、科学计算等方向的开发。

Python数据处理实用工具

- **numpy/scipy/matplotlib**
 - 这三个工具提供了 MATLAB 的典型功能，包括快速矩阵运算、科学计算函数、还提供了绘图工具、这些绘图工具被广泛使用，其思想也源于 MATLAB。
- **pandas**
 - 该工具的功能和R的data.frame类似，但启动的效率往往要比data.frame高不少。
- **scikit-learn/statsmodels**
 - 这两个工具提供了高质量的机器学习算法（分类/回归、聚类、矩阵分解等）的实现和统计模型实现。
- **nltk**
 - 一个深受欢迎的自然语言工具。



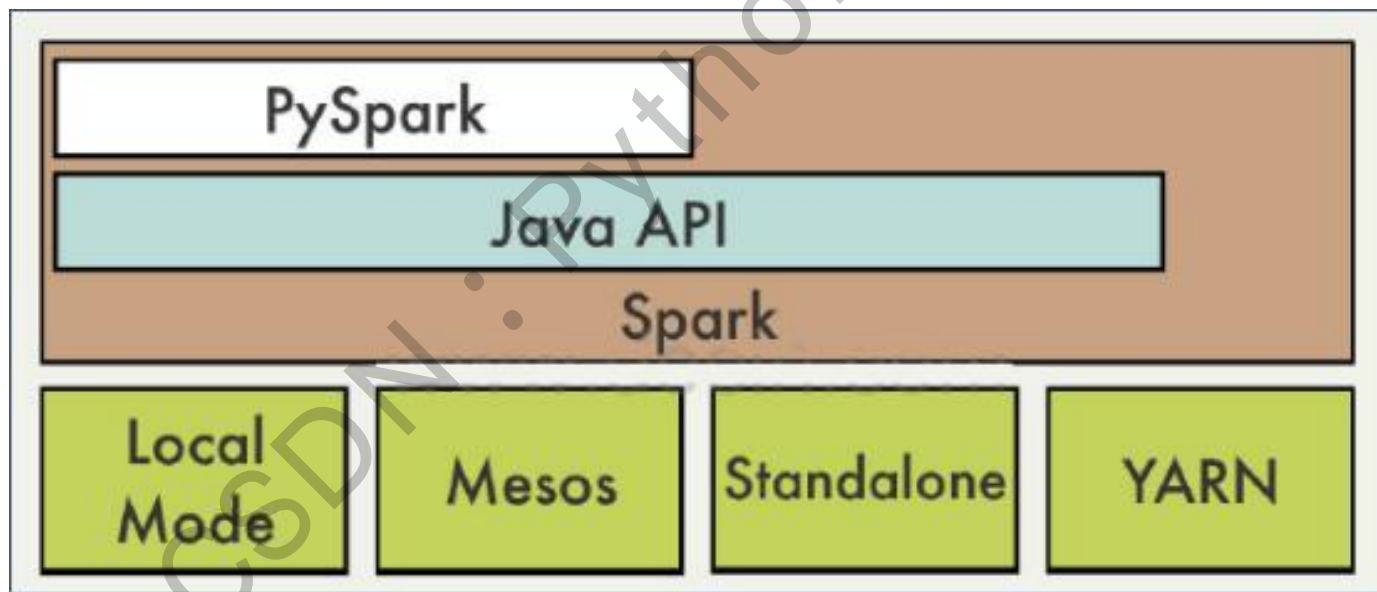
了解 PySpark

什么是PySpark?

AI、大数据、云计算，这三者在如今的互联网时代无人不知无人不晓，火爆程度不言而喻。Spark 是什么，相信也不用做过多的介绍，不过有几点还是要提一下，Spark是使用 Scala 语言实现，而Scala建立在JAVA之上。不过，Spark支持Java，Python和Scala API，支持交互式Python和Scala的shell。

- PySpark 是 Spark 为使用 Python 程序编写的 Spark Application 而实现的客户端库。
- 通过 PySpark 也可以编写 Spark Application 并在 Spark 集群上运行。
- 为了能够充分利用Python那些高效的模块，很多机器学习程序都会使用 Python 实现，同时也希望能够在 Spark 集群上运行。

- 官方对PySpark的释义为：“PySpark is the Python API for Spark”。也就是说PySpark为Spark提供的Python编程接口。
- Spark使用py4j来实现python与java的互操作，从而实现使用python编写Spark程序。Spark也同样提供了pyspark，一个Spark的python shell，可以以交互式的方式使用Python编写Spark程序。



什么是Py4J?

- 简单点说，py4J 是 Python 和 Java 之前的桥梁。
- Py4J允许在Python解释器中运行的Python程序动态访问Java虚拟机中的Java对象。调用方法就好像Java对象驻留在Python解释器中，并且可以通过标准Python集合方法访问Java集合。Py4J还使Java程序能够回调Python对象。

```
>>> from py4j.java_gateway import JavaGateway
>>> gateway = JavaGateway() # connect to the JVM
>>> random = gateway.jvm.java.util.Random() # create a java.util.Random instance
>>> number1 = random.nextInt(10) # call the Random.nextInt method
>>> number2 = random.nextInt(10)
>>> print(number1, number2)
(2, 7)
>>> addition_app = gateway.entry_point # get the AdditionApplication instance
>>> value = addition_app.addition(number1, number2) # call the addition method
>>> print(value)
9
```


使用Py4J编写一个Python程序

现在，您将编写访问Java程序的python程序。启动Python解释器，并确保Py4J位于PYTHONPATH中。

1. 第一步是导入必要的Py4J类：

```
>>> from py4j.java_gateway import JavaGateway
```

2. 接下来，初始化JavaGateway。通常情况下，默认参数就足够了。当您创建JavaGateway时，Python尝试使用网关(端口25333上的本地主机)连接到JVM。

```
>>> gateway = JavaGateway()
```

3. 从网关对象中，我们可以通过引用入口点的entry_point成员来访问入口点：

```
>>> stack = gateway.entry_point.getStack()
```

使用Py4J编写一个Python程序

4. 堆栈变量现在包含一个堆栈。尝试推动和弹出一些元素：

```
>>> stack.push("First %s" % ('item'))
>>> stack.push("Second item")
>>> stack.pop()
u'Second item'
>>> stack.pop()
u'First item'
>>> stack.pop()
u'Initial Item'
```

5. 您会得到一个Py4JJavaError，因为JVM端有一个异常。此外，您还可以看到在Java端抛出的异常类型及其堆栈跟踪。

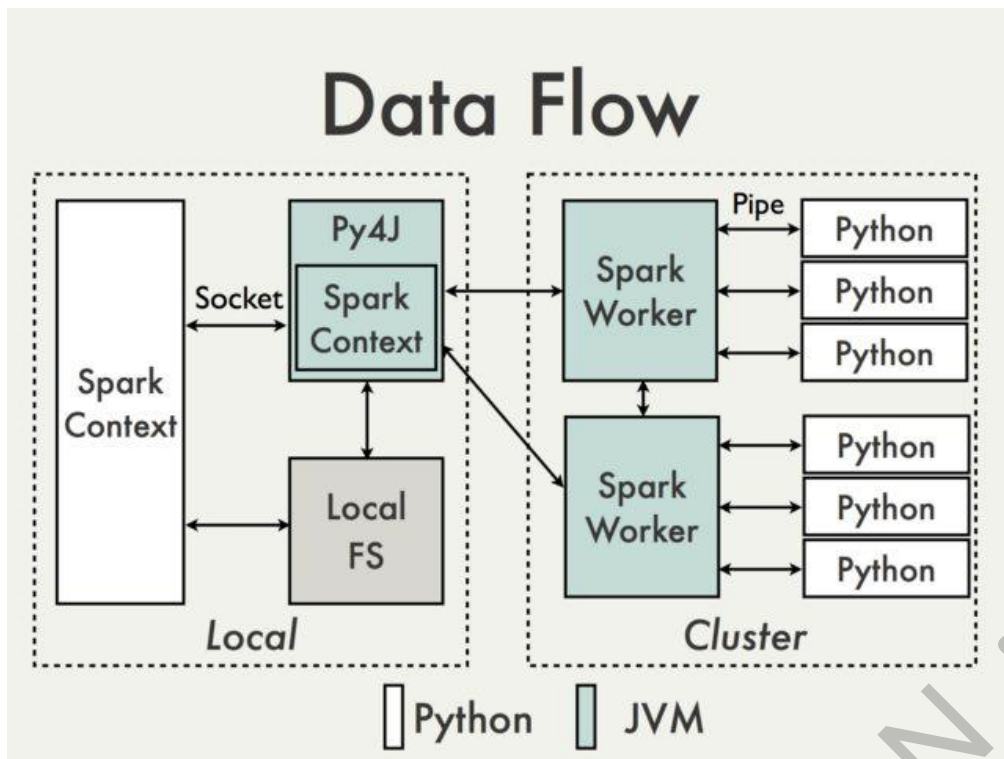
```
>>> stack.pop()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "py4j/java_gateway.py", line 346, in __call__
    return_value = get_return_value(answer, self.gateway_client, self.target_id, self.name)
  File "py4j/java_gateway.py", line 228, in get_return_value
    raise Py4JJavaError("An error occurred while calling %s%s%s" % (target_id, '.', name))
py4j.java_gateway.Py4JJavaError: An error occurred while calling o0.pop.
java.lang.IndexOutOfBoundsException: Index: 0, Size: 0
    at java.util.LinkedList.entry(LinkedList.java:382)
    at java.util.LinkedList.remove(LinkedList.java:374)
    at py4j.examples.Stack.pop(Stack.java:42)
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:57)
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
    at java.lang.reflect.Method.invoke(Method.java:616)
    at py4j.reflection.MethodInvoker.invoke(MethodInvoker.java:119)
    at py4j.reflection.ReflectionEngine.invoke(ReflectionEngine.java:392)
    at py4j.Gateway.invoke(Gateway.java:255)
    at py4j.commands.AbstractCommand.invokeMethod(AbstractCommand.java:125)
    at py4j.commands.CallCommand.execute(CallCommand.java:81)
    at py4j.GatewayConnection.run(GatewayConnection.java:175)
    at java.lang.Thread.run(Thread.java:636)
```

使用 PySpark 的缺点

1. 性能开销大，为了能在基于 JVM 的语言（比如 Scala）上运行用Python编写的算法，我们必须在不同环境中传递代码和数据，这会付出代价，而且在转换的过程中信息有时会丢失。
2. 不能用上最新的版本和最好的功能，Spark的机器学习、流处理和图分析库全都是Scala写的，而新的功能对Python绑定支持可能要慢得多。
3. 对你理解Spark的原理不能提供过多的帮助，即使在Python种调用Spark，API仍然反映了底层计算原理，但它却是Spark从其开发语言Scala继承过来的。



PySpark Application 运行原理



数据处理流程图

PySpark 的 Python 解释器在启动时会同时启动一个 JVM, Python 解释器与 JVM 进程之间通过套接字保持通信。PySpark 利用 Py4j 项目来处理 Python 解释器和 JVM 之间的通信。

JVM 作为实际的 Spark 驱动程序会加载一个 JavaSparkContext, JavaSparkContext 和集群中的 Spark 执行器通信。

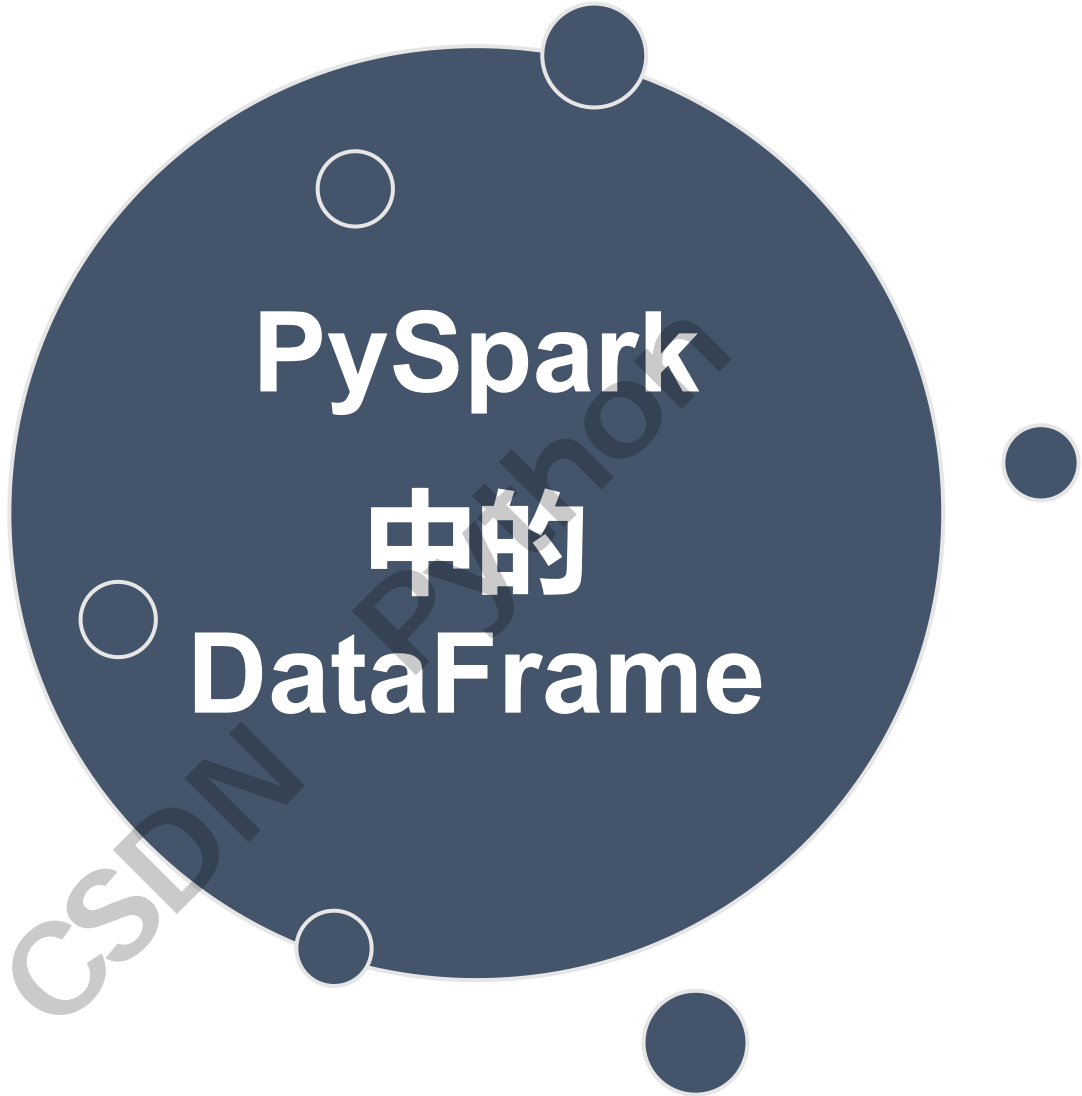
对 SparkContext 对象 Python API 调用会被翻译为对 JavaSparkContext 对象的 Java API 的调用。

集群上的 Spark 执行器为每个 CPU 核启动一个 Python 解释器, 并在需要执行用户代码时通过 Unix 管道 stdin 以及 stdout 与这个解释器进行数据通信。

在 Python RDD 上调用 API, 所有相关代码将通过 cloudpickle 进行序列化并分发到执行器上。接着, 代码的序列化数据由 Java 对象转化成 Python 兼容的表现形式 (即 pickle 对象) 并通过一个管道流的方式传给执行器相关的 Python 解释器。

举个例子

1. PySpark 的 `sc.textFile()` 实现将调用分派给 `JavaSparkContext` 的 `.textFile` 方法；
2. 在Python 解释器中运行的 `sc.textFile()` 将会调用 `JavaSparkContext` 的 `textFile` 方法；
3. 该方法会把集群中的数据加载为 Java String 对象；
4. 类似的，用 `newAPIHadoopFile` 加载一个 Parquet/Avro 文件会把对象加载为 Java Avro 对象



PySpark 中的 DataFrame

DataFrame 概述

在Apache Spark中，DataFrame 是命名列下的分布式行集合。简单地说，它与关系数据库中的表或具有列标题的Excel表相同。它和RDD还有一些共同的特点：

- 本质上是不可变的：我们可以创建一次 DataFrame/RDD，但是不能更改它。我们可以在应用转换之后转换 DataFrame/RDD。
- 延迟计算：这意味着在执行操作之前不会执行任务。
- 分布式：RDD 和 DataFrame本质上都是分布式的。

为什么 DataFrame 是有用的？

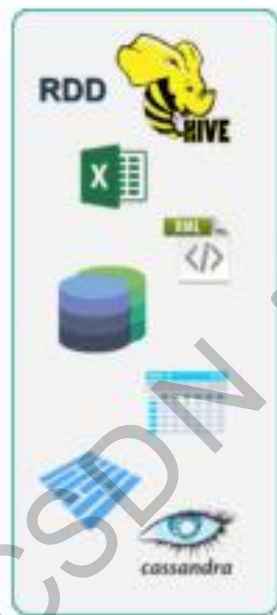
DataFrame API 受到R和Python (Pandas) 中的数据框架的启发，但是从底层开始设计以支持现代大数据和数据科学应用程序。作为现有RDD API的扩展，DataFrame具有以下功能：

- 支持多种数据格式和数据源。
- 能够从单台笔记本电脑上的千字节数据扩展到大型群集上的PB级数据。
- 通过Spark SQL Catalyst优化器实现最先进的优化和代码生成。
- 通过Spark无缝集成所有大数据工具和基础架构。
- Python , Java , Scala和R的API。
- 通过 DataFrame 与 Catalyst 优化器，现有的Spark程序迁移到 DataFrame 时性能得到改善。由于优化器生成用于执行的JVM字节码，因此Python用户将体验到与Scala和Java用户相同的高性能。

DataFrames数据来源

它可以使用不同的数据格式创建。例如，从JSON、CSV加载数据。

- 从现有的RDD加载数据。
- 以编程方式指定模式



DATAFRAME				
DESCRIPTION	COLUMN ONE	COLUMN TWO	COLUMN THREE	COLUMN FOUR
First Feature	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>
Second Feature	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Third Feature	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>
Fourth Feature	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Fifth Feature	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>

Pandas vs PySpark DataFrame

Pandas 和 PySpark DataFrame之间的几个区别是:

- Pyspark DataFrame上的操作在集群中的不同节点上并行运行，但是 Pandas 情况下，这是不可能的。
- PySpark DataFrame中的操作本质上是惰性的，但是对于Pandas，我们只要应用任何操作就会得到结果。
- 在PySpark DataFrame 中，由于 DataFrame 是不可变的属性，我们不能更改它，我们需要对它进行转换。但在 Pandas 却不是这样。
- Pandas API 比 PySpark DataFrame 支持更多的操作。 Pandas API比Spark更强大。
- 在 Pandas 中执行复杂的操作比 Pyspark DataFrame 更容易。
- 除了以上几点之外，Pandas 和 Pyspark DataFrame 还有一些基本的区别，比如列的选择、过滤、添加列等等，这些我在这里没有介绍。



YARN集群 配置 Python环 境

应用场景

1. 初始安装YARN，Spark集群，并考虑到了当前应用需要支持Python程序运行。

安装

1. 准备好对应Python程序包、依赖模块，在YARN集群中的每个节点上进行安装。
2. 这样，YARN集群的每个NodeManager上都有Python环境，目前流行的是使用安装Python虚拟环境，使用Anaconda等软件，可以极大的简化Python环境管理的工作。

缺点

1. 后续增加新的Python依赖模块，需要在集群的每个节点上都进行该新增模块的安装
2. 如果依赖Python版本，还要管理不同版本的Python环境。因为提交 PySpark Application 运行，具体是在哪些 NodeManager 上运行，是由YARN调度器决定的，必须保证每个 NodeManager 上都有Python环境（基础环境+依赖模块）。

应用场景

1. 已经安装了规模较大的YARN集群，并在开始使用时并未考虑后续会用到 Python。
2. 并且不想再YARN集群的 NodeManager 上安装 Python 基础环境及其依赖模块。

安装

1. 在任意一个Linux OS的节点上，安装Anaconda软件。
2. 通过 Anaconda 创建虚拟 Python 环境。
3. 在创建好的 Python 环境中安装依赖的 Python 模块。
4. 将整个Python 环境达成 zip 包。
5. 提交 PySpark Application 时，通过 `-archives` 选项指定 zip 包路径。

下面进行详细说明

首先我们在 centos7 系统上，基于 Python 2.7，安装了 Anaconda 软件。Anaconda 的安装路径为 /root/anaconda2，创建一个虚拟环境。

```
1 | conda create -n mlp_env --copy -y -q python=2 numpy pandas scipy
```

上述命令创建了一个名称为 mlp_env 的Python环境，--copy 选项将对应的软件包都安装到该环境中，包括一些C的动态链接库文件。同时，下载 numpy、pandas、scipy 这三个依赖模块到该环境中。

接着，将Python环境打包。

```
1 | cd /root/anaconda2/envs  
2 | zip -r mlp_env.zip mlp_env
```

将该zip压缩包拷贝到指定目录中，方便后续提交PySpark Application：

```
1 | cp mlp_env.zip /tmp/
```

最后，我们可以提交我们的PySpark Application，执行如下命令：

```
1 PYSPARK_PYTHON=./ANACONDA/mlpy_env/bin/python spark-submit \  
2 --conf spark.yarn.appMasterEnv.PYSPARK_PYTHON=./ANACONDA/mlpy_env/bin/python \  
3 --master yarn-cluster \  
4 --archives /tmp/mlpy_env.zip#ANACONDA \  
5 /var/lib/hadoop-hdfs/pyspark/test pyspark dependencies.py
```

上面的test_pyspark_dependencies.py文件中，使用了numpy、pandas、scipy这三个依赖包的函数，通过上面提到的YARN集群的cluster模式可以运行在Spark集群上。

可以看到，上面的依赖zip压缩包将整个Python的运行环境都包含在里面，在提交PySpark Application时会将该环境zip包上传到运行Application的所在的每个节点上，并解压缩后为Python代码提供运行时环境。如果不想每次都从客户端将该环境文件上传到集群中运行PySpark Application的节点上，也可以将zip包上传到HDFS上，并修改--archives参数的值为hdfs:///tmp/mlpy_env.zip#ANACONDA，也是可以的。

另外，需要说明的是，如果我们开发的/var/lib/hadoop-hdfs/pyspark/test_pyspark_dependencies.py文件中，也依赖的一些我们自己实现的处理函数，具有多个Python依赖的文件，想要通过上面的方式运行，必须将这些依赖的Python文件拷贝到我们创建的环境中，对应的目录为mlpy_env/lib/python2.7/site-packages/下面。

缺点

1. 不在YARN集群上安装Python环境的方案，会使提交的Python环境zip包在YARN集群中传输带来一定开销。
2. 每次提交一个PySpark Application都需要打包一个环境zip文件，如果有大量的Python实现的PySpark Application需要在Spark集群上运行，开销会越来越大。
3. 如果PySpark应用程序修改，可能需要重新打包环境。但是这样做确实不在需要考虑YARN集群集群节点上的Python环境了，任何版本Python编写的PySpark Application都可以使用集群资源运行。

使用 PySpark Shell

- 前提：已经有一个正在使用的集群并已经配置了Python环境
- 设置变量：
 - `export PYSPARK_PYTHON="/opt/anaconda3/bin/python"` # Spark2.2.0对Python3.6的兼容性已经很好了
 - `export PYSPARK_DRIVER_PYTHON="/opt/anaconda3/bin/ipython"` # 使用IPython驱动
- 提交任务的命令：
 - `/opt/spark-2.2.0/bin/pyspark --master yarn \`
 - `--driver-memory 2g \`
 - `--executor-memory 2g \`
 - `--executor-cores 2`

这样就可以使用Ipython进行编辑了，比默认PySpark Shell 要好用太多。

```
[hadoop@master1 pyspark_run]$ /opt/spark-2.2.0/bin/pyspark --master local[*]
Python 3.6.5 [Anaconda, Inc.] (default, Apr 29 2018, 16:14:56)
Type 'copyright', 'credits' or 'license' for more information
IPython 6.4.0 -- An enhanced Interactive Python. Type '?' for help.
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
19/03/29 16:43:03 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java-only versions to prevent automatic fallback!
19/03/29 16:43:03 WARN SparkConf: In Spark 1.0 and later spark.local.dir will be overridden by the value of spark.local.dir
19/03/29 16:43:04 WARN Utils: Service 'SparkUI' could not bind on port 4044. Attempting port 4045.
19/03/29 16:43:04 WARN Utils: Service 'SparkUI' could not bind on port 4045. Attempting port 4046.
19/03/29 16:43:04 WARN Utils: Service 'SparkUI' could not bind on port 4046. Attempting port 4047.
19/03/29 16:43:04 WARN Utils: Service 'SparkUI' could not bind on port 4047. Attempting port 4048.
19/03/29 16:43:30 WARN ObjectStore: Version information not found in metastore. hive.metastore.schema.verification is not enabled so recording the schema change
19/03/29 16:43:30 WARN ObjectStore: Failed to get database default, returning NoSuchObjectException
19/03/29 16:43:33 WARN ObjectStore: Failed to get database global_temp, returning NoSuchObjectException
Welcome to

      _/ _ \| | | | _/_/
     / _ \| | | | |/_/
    / ___| | | | |___/
   /___|_|_|_|_|___/
version 2.2.0

Using Python version 3.6.5 (default, Apr 29 2018 16:14:56)
SparkSession available as 'spark'.

In [1]: sc
Out[1]: <SparkContext master=local[*] appName=PySparkShell>

In [2]: from pyspark import *
```

Accumulator	HiveContext	RDD	SparkContext	StatusTracker	broadcaster
AccumulatorParam	MarshalSerializer	Row	SparkFiles	StorageLevel	cloud
BasicProfiler	PickleSerializer	SQLContext	SparkJobInfo	TaskContext	conf
Broadcast	Profiler	SparkConf	SparkStageInfo	accumulators	cont

在 Jupyter Notebook 中使用PySpark

- 在小黑窗里面的PySpark功能还是太少了，下面我们配置JupyterNotebook使用PySpark。
- `export PYSARK_PYTHON="/opt/anaconda3/bin/python"` # PySpark 执行代码的环境
- `export PYSARK_DRIVER_PYTHON="/opt/anaconda3/bin/jupyter"` # 这里使用 jupyter 启动
- `export PYSARK_DRIVER_PYTHON_OPTS='notebook --config=/data/jupyter_notebook_config.py'` # Notebook需要一些配置才会更好用
- 启动命令：
- `/opt/spark-2.2.0/bin/pyspark --master yarn \`
- `--driver-memory 2g \`
- `--executor-memory 2g \`
- `--executor-cores 2`

Notebook配置文件：

jupyter_notebook_config.py

- `c.NotebookApp.ip = '0.0.0.0'`
- `c.NotebookApp.notebook_dir = "/path/to/pyspark_run/notebook_dir"`
- `c.NotebookApp.open_browser = False`
- `c.NotebookApp.port = 58888`
- `c.NotebookApp.token = ''`

注：如果每次都要手动敲那么多命令，那比较麻烦，所以把上面的一堆命令可以写成一个脚本，每次使用的时候只需要启动脚本，然后去浏览器打开 ip:58888 就可以使用PySpark 了

Shell 脚本：

```
[hadoop@master1 pyspark_run]$ cat startup.sh
#!/usr/bin/env bash

set -ex

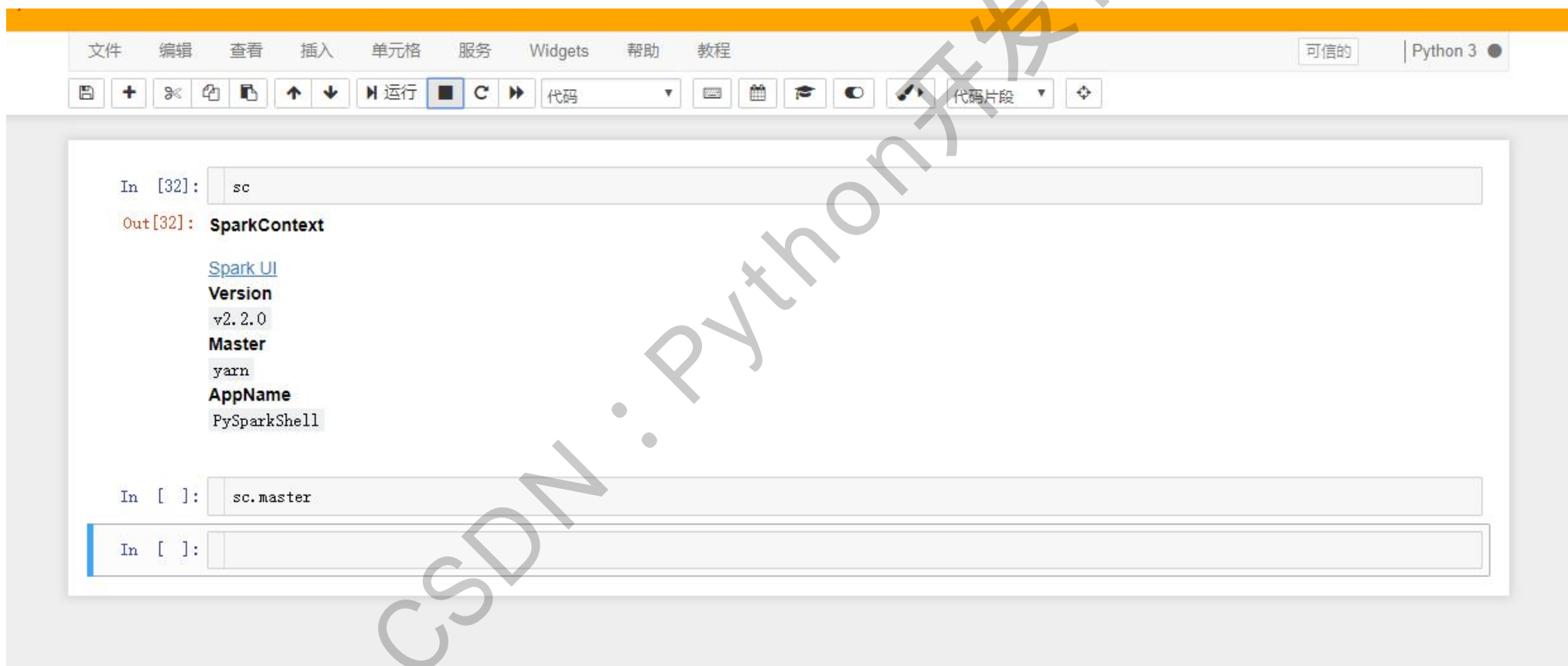
cd `dirname $0`

MASTER=${1:-"yarn"}

export SPARK_HOME="/opt/spark-2.2.0"
export PYSARK_PYTHON="/opt/anaconda3/bin/python"
export PYSARK_DRIVER_PYTHON="/opt/anaconda3/bin/jupyter"
export PYSARK_DRIVER_PYTHON_OPTS='notebook --config=/data/hadoop/liuchy/pyspark_run/jupyter_notebook_config.py'

/opt/spark-2.2.0/bin/pyspark --master ${MASTER} \
                             --driver-memory 2g \
                             --executor-memory 2g \
                             --executor-cores 2 > notebook.log 2>&1 &
```

进入Home页以后，创建一个Python3，即可自动提交任务到集群上。



多用户的 PySpark

多用户Notebook

- 前面提到的都是单用户向集群上提交任务，大家也都知道，这种随便使用集群资源的情况是很少的，所以这里会讲到如何搭建一个多用户的Notebook环境。

- 搭建这个工具需要用到的软件：

JupyterHub + JupyterNotebook

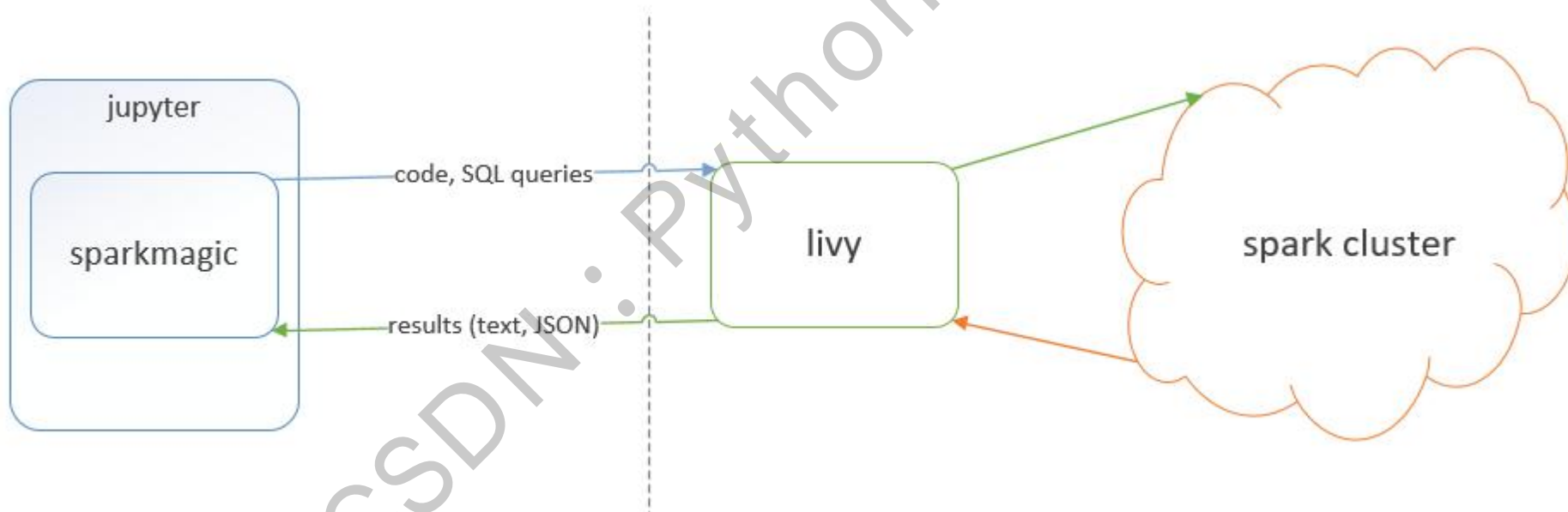
- 这样只是实现了一个多用户的Notebook环境。但是这样只是实现了多用户的需求，我们想要提交任务到集群还需要下面这个工具：

sparkmagic

- sparkmagic是一组工具，通过Livy(一个Spark REST服务器)在Jupyter笔记本上与远程Spark集群交互工作。

SparkMagic 通信原理图

- Sparkmagic使用Spark的REST服务器Livy来远程执行所有用户代码。然后，库会自动将代码输出收集为纯文本或JSON文档，并将结果显示为格式化文本或适当的Pandas数据帧。



SparkMagic的好处

- 完全远程运行Spark代码，无需在部署Notebook的服务器上安装Spark组件。
- 多语言支持，Python、Python3、Scala和R内核都具有同样丰富的特性，添加对更多语言的支持将很容易，
- 支持多个端点，您可以使用一个笔记本启动多个Spark作业，这些作业使用不同的语言和针对不同的远程集群。
- 易于与任何Python库集成，用于数据科学或可视化。

SparkMagic的限制

- 通过Livy发送所有代码和输出会增加一些开销
- 由于所有代码都是通过Livy在远程驱动程序上运行的，所以所有结构化数据都必须序列化为JSON，并由Sparkmagic库进行解析，以便在客户端对其进行操作和可视化。实际上，这意味着必须使用Python以`%%local`模式对客户端数据进行操作。

2019
Python开发者日
让开发者紧跟
技术潮流

谢谢观看