

Learning Spark

LIGHTNING – FAST DATA ANALYSIS

CH1. LEARNING SPARK

▶ 아파치 스파크 :

- ▶ 범용적이면서도 빠른 속도로 작업을 수행할 수 있도록 설계한 클러스터용 연산 플랫폼

▶ 속도

- ▶ 다양한 연산모델을 효과적으로 지원,
- ▶ 맵리듀스 모델을 대화형 명령어 쿼리나 스트리밍 처리가 가능하도록 확장
- ▶ 메모리에서 연산수행 지원. 디스크에서 수행시도 맵리듀스보다 빠름

▶ 범용적

- ▶ Batch application, 반복 알고리즘, 대화형 쿼리, 스트리밍 같은 각각 분산된 시스템에서 돌아가던 다양한 작업타입을 동시에 커버할수 있도록 설계
- ▶ 서로 다른 형태의 작업을 수행하는 데이터 분석 파이프라인의 연계가 가능.

▶ 고수준 접근

- ▶ 파이썬, 자바, 스칼라, SQL API 및 강력한 라이브러리 내장.
- ▶ 빅데이터 tool 과의 연동
- ▶ 하둡 클러스터 위에서 실행가능

통합된 구성

- ▶ 밀접하게 연동된 여러 개의 컴포넌트로 구성
- ▶ 빠르고 범용적인 핵심엔진
- ▶ 하위 레이어의 성능향상에 의해 구성에 포함된 라이브러리 및 고수준 컴포넌트가 직접적으로 이익을 볼 수 있다.
 - ▶ 스파크 핵심엔진 최적화 -> SQL, 머신 러닝 라이브러리 속도역시 자동으로 빨라짐.
- ▶ 서로 다른 데이터 처리 모델을 합쳐서 하나의 애플리케이션으로 만들수 있다.

SPARK 구성

Spark SQL +
DataFrames
정형화된 데이터

Streaming
실시간

MLlib
Machine Learning
머신러닝

GraphX
Graph Computation
그래프처리

워크로드
구성요소

Spark Core API

코어 엔진

R

SQL

Python

Scala

Java

스파크SQL
(정형화된 데이터)

스파크 스트리밍
(실시간)

MLlib
(머신 러닝)

그래프X
(그래프 처리)

분류, 회귀, 클러스터링,
협업 필터링(추천)

스파크 코어

작업 스케줄, 메모리관리,
장애복구, 저장장치와의 연동

단독 스케줄러

안

메소스

클러스터
매니저

Spark SQL

- ▶ **Structured Data:** Spark SQL
- ▶ interactive SQL queries for exploring data
- ▶ for structured data processing.
- ▶ It provides a programming abstraction called
- ▶ DataFrames
- ▶ distributed SQL query engine
- ▶ It enables unmodified Hadoop Hive queries to run up to 100x faster on existing deployments and data.
- ▶ It also provides powerful integration with the rest of the Spark ecosystem (e.g., integrating SQL query processing with machine learning).

SPARK STREAMING

- ▶ 실시간 데이터 스트림을 처리 가능하게 해주는 스파크의 구성요소
- ▶ 실시간 데이터 스트림 : 웹서버의 로그 파일, 웹서미스 사용자들의 상태 업데이트 메시지 저장 큐 등
- ▶ 스파크 코어의 RDD API 와 거의 일치하는 형태의 조작 API 지원
- ▶ 메모리 , 디스크에 저장되어있는 데이터나 실시간으로 받는 데이터를 다루는 애플리케이션을 서로 바꿔가며 다루더라도 혼란이 없음
- ▶ 스파크 코어와 동일한 수준의 장애관리, 처리량, 확장성을 지원

MMLIB

- ▶ 분류, 회귀, 클러스터링, 협업필터링 등의 다양한 타입의 머신러닝 알고리즘 및 모델 평가 및 외부 데이터 불러오기 등의 기능을 지원
- ▶ 최적화 알고리즘 등의 몇몇 저수준 ML 핵심 기능 지원.

그래프X

- ▶ GRAPH를 다루기 위한 라이브러리, 그래프 병렬 라이브러리 수행
- ▶ https://github.com/psygrammer/ScalaML/blob/master/part3/study02/SparkGraphX/SparkGraphx_02_sejin.ipynb
- ▶ 스파크 RDD API 를 확장.
- ▶ 각 선(EDGE)나 점(VERTEX) 에 임의의 속성을 추가한 지향성 그래프 만들기
- ▶ 그래프를 다루는 다양한 메소드와 그래프 알고리즘 지원

스파크 코어

- ▶ 작업 스케줄링, 메모리 관리, 장애복구, 스토리지 연동 등 기본 기능으로 구성
- ▶ 분산 데이터 세트 RDD를 생성하고 조작하는 API 제공

클러스터 매니저

한 노드에서 수천 노드까지 효과적으로 성능 확장이 가능

유연성을 극대화 하면서 달성하기 위해, **안, 아파치 메소스, 단독 스케줄러** 등
다양한 클러스터 매니저 위에서 동작 가능

누가 무엇을 위해 사용하는가

▶ 데이터 과학

- ▶ SQL, 통계, 예측모델링(머신러닝) 경험과 함께 파이썬, 매틀랩, R 프로그래밍 기술 및 데이터 가공 기술적 경험을 가진 데이터 과학자.
- ▶ 질문에 대한 답을 찾거나 통찰을 갖기위해 데이터 분석을 하는 업무 흐름 중 단발성 분석으로 빠른 시간 내에 짧은 코드와 단순한 쿼리로 최소한의 시간 내에 결과를 보기위해 대화형 셸 사용.
- ▶ 스파크의 속도와 단순한 API, 내장된 라이브러리로 수많은 알고리즘을 즉시 꺼내쓸 수있다
- ▶ 스파크 셸 : 대화형 데이터 분석, 스파크 SQL: SQL을 통한 데이터 탐색 가능, Mlib : 머신러닝,

▶ 데이터 처리 애플리케이션

- ▶ 상용 데이터처리 애플리케이션 , 캡슐화, 인터페이스 설계, 객체 지향 프로그래밍
- ▶ 소프트웨어를 클러스터 위에서 동시에 동작, 시스템 프로그래밍, 네트워크 통신, 장애대응등의 복잡성을 숨김. 업무를 재사용가능한 라이브러리로 나누기쉽고 로컬에서 테스트 하기 쉽다.

▶ 다양한 기능, 배우고 쓰기 쉽다. 코드가 성숙, 안정

바로 시작해봅시다~!

SPARK 설치

<http://spark.apache.org/downloads.html>

분석환경이 갖추어 지지않아 Spark 를 공부할 준비가 안되셨다면~

: Let's start with Docker~!!

```
docker run -d -p 9999:8888 -e GRANT_SUDO=yes --name psy_spark jupyter/all-spark-notebook
```

jupyter노트북은 8888 port로 기본 설정되어있습니다. 자신이 사용하고 싶은 port로 연결해 줍니다. 저는 이미 다른 컨테이너에서 8888포트를 사용하고 있어 9999로 설정하였습니다.

자신의 서버 ip 또는 docker ip :9999 로 Jupyter notebook을 실행합니다.

스파크 핵심 개념


- ▶ Sc 변수 파악
- ▶ 파이썬, 스칼라 필터링 예제
- ▶ 스파크에 함수 전달하기
- ▶ 단독 애플리케이션
- ▶ SparkContext 초기화 하기
- ▶ 단독 애플리케이션 빌드하기 – 단어 세기 애플리케이션


Create Droplets


Choose an image ?


Distributions


One-click Apps



Ubuntu
14.04.4 x64


FreeBSD
Select Version


Fedora
Select Version


Debian
Select Version


CoreOS
Select Version


CentOS
Select Version

Choose a size

\$5 /mo \$0.007/hour	\$10 /mo \$0.015/hour	\$20 /mo \$0.030/hour	\$40 /mo \$0.060/hour	\$80 /mo \$0.119/hour	\$160 /mo \$0.238/hour
512 MB / 1 CPU 20 GB SSD Disk 1000 GB Transfer	1 GB / 1 CPU 30 GB SSD Disk 2 TB Transfer	2 GB / 2 CPUs 40 GB SSD Disk 3 TB Transfer	4 GB / 2 CPUs 60 GB SSD Disk 4 TB Transfer	8 GB / 4 CPUs 80 GB SSD Disk 5 TB Transfer	16 GB / 4 CPUs 160 GB SSD Disk 6 TB Transfer
\$320 /mo \$0.476/hour	\$480 /mo \$0.714/hour	\$640 /mo \$0.952/hour			
32 GB / 12 CPUs	48 GB / 16 CPUs	64 GB / 20 CPUs			


Docker 를 이용한 실습


Create Droplets


Choose an image ?


Distributions


One-click Apps


 Cassandra on 14.04


 Discourse on 14.04


 **Docker 1.11.0 on 14.04**


 Dokku v0.5.4 on 14.04


 Drupal 8.0.5 on 14.04


 Elixir on 14.04


 Ghost 0.7.9 on 14.04

 GitLab 8.7.0 CE on 14.04

 LAMP on 14.04

 LEMP on 14.04

 MEAN on 14.04

 MediaWiki on 14.04

* Documentation: <https://help.ubuntu.com/>

System information as of Tue Apr 26 07:32:54 EDT 2016

L

QS c

C

KGjQtGXP

System load: 0.0

Memory usage: 1%

Processes: 79

Usage of /: 2.0% of 78.62GB

Swap usage: 0%

Users logged in: a0 urce DataSourceDigitalOcean. Up 3.91 seconds

Graph this data and manage this system at:

<https://landscape.canonical.com/>

root@biospark:~#

root@biospark:~#

root@biospark:~# docker run -d -p 9999:8888 -e GRANT_SUDO=yes --name biospark jupyter/all-spark-notebook

docker: Error parsing reference: "jupyter/all-spark-notebook" is not a valid repository/tag.

See 'docker run --help'.

root@biospark:~# docker run -d -p 9999:8888 -e GRANT_SUDO=yes --name biospark jupyter/all-spark-notebook

Unable to find image 'jupyter/all-spark-notebook:latest' locally

[711.067397] aufs au_ops_parse:1155:docker[1232]: unknown option dirperm1

latest: Pulling from jupyter/all-spark-notebook

fdd5d7827f33: Pull complete

a3ed95caeb02: Pull complete

f08e494cf5cc: Extracting [=====>

1 406.7 MB/637 MB

6548f4aff175: Download complete

d762b5abb43e: Download complete

c841d1ad6a8e: Download complete

78ff99539390: Download complete

ca1e42ffa07b: Download complete

a2e91df52de2: Download complete

b653922de2b8: Download complete

1aff6b8f4bde: Download complete

6693b52f757b: Download complete

85562a53de6b: Download complete

40c1ea0555e5: Download complete

sudo docker run -d -p [외부포트]:[컨테이너내부포트] -e
GRANT_SUDO=yes --name [컨테이너 이름] jupyter/all-spark-notebook

출처 : <http://yujuwon.tistory.com/entry/spark-tutorial>

SPARK 대화형 셸

- ▶ 파이썬 : `bin/pyspark`
- ▶ 스칼라 : `bin/spark-shell`

Line count - scala

```
scala> var lines = sc.textFile("README.md")
```

```
lines: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[1] at textFile at  
<console>:21
```

```
scala> lines.count()
```

```
res0: Long = 98
```

```
scala> lines.first()
```

```
res1: String = # Apache Spark
```

도커로 스파크(with jupyter notebook) 시작하기

```
sudo docker run -d -p [외부포트]:[컨테이너내부포트] -e GRANT_SUDO=yes --name [컨테이너 이름] jupyter/all-spark-notebook
```

저는 biospark 라는 이름으로, 외부포트를 9999로 변경하여 실행해보겠습니다.

```
sudo docker run -d -p 9999:8888 -e GRANT_SUDO=yes --name biospark jupyter/all-spark-notebook
```

도커 웹에서 위와 같이 실행한 후,

크롬 등 인터넷 브라우저에서 ip:9999 실행하시면 jupyter notebook 환경이 뜹니다.

ip는 서버 사용하시는 서버의 ip를 사용하시면 되고

윈도우나 맥에서 docker-machine 사용하시는 경우, docker-machine ip default 로 ip 확인하시면 됩니다.

출처 : <http://yujuwon.tistory.com/entry/spark-tutorial>

초기화 하기

```
In [4]: from pyspark import SparkContext
```

```
In [5]: sc = SparkContext() # SparkContext 객체가 sc 변수에 만들어진다.
```

```
In [6]: sc # sc를 입력하여 다음과 같은 결과가 나오면 성공
```

```
Out[6]: <pyspark.context.SparkContext at 0x7fb769907750>
```

▶ 실습파일

▶ bicbio_spark_ch02example.ipynb

text.txt 파일 만들기

%%writefile 을 이용하면 쥬피터 노트북에서 쉽게 예제로 사용할 txt 파일을 만들수 있습니다.

- ▶ 실습파일
- ▶ [bicbio_spark_ch02example.ipynb](#)

```
In [7]: %%writefile text.txt
this is first line
this is second line
this is third line
this is fourth line
```

Overwriting text.txt

```
In [8]: cat text.txt
```

```
this is first line
this is second line
this is third line
this is fourth line
```

스파크에서는 연산 과정을 클러스터 전체에 걸쳐 자동으로 병렬화 해 분산 배치된 연산 작업들의 모음으로 표현

이 모음은 탄력적인 분산 데이터 셋트 혹은 RDD(Resilient Distributed Dataset) 라고 부릅니다. RDD는 분산 데이터와 연산을 위한 스파크의 핵심적인 개념!

텍스트 파일로 부터 RDD를 하나 만들어보고 간단한 분석을 해봅시다.

lines 라는 이름의 RDD를 만든다.

```
In [9]: lines = sc.textFile("text.txt")
```

파이썬으로 줄 수 세기

이 RDD의 아이템 갯수를 센다

```
In [12]: lines.count()
```

```
Out[12]: 4
```

이 RDD의 첫번째 아이템, 이 예제에서는 **text.txt** 의 첫번째 라인

```
In [13]: lines.first() # RDD의 첫번째 아이템. , text.txt 의 첫번째 라인
```

```
Out[13]: u'this is first line'
```

파이썬 필터링 예제

특정 단어만 들어있는 라인을 **filtering** 해봅시다.

```
In [10]: pythonLines = lines.filter(lambda line : "first" in line)
```

```
In [11]: pythonLines.first()
```

```
Out[11]: u'this is first line'
```

▶ 실습파일

▶ [bicbio_spark_ch02example.ipynb](#)