빅데이터 마스터과정
(DAM)

하석재
CEO, 2HCUBE
sjha72@gmail.com

스파크 프레임워크

스파크(Spark)란?

- **Apache Spark™**
  - **a fast and general engine for large-scale data processing** (이전)
  - a unified analytics engine for large-scale data processing (현재)

스파크(Spark)란?

- **Apache Spark™**
  - a fast and general engine for large-scale data processing (이전)
  - **a unified analytics engine for large-scale data processing** (현재)

스파크(Spark)란?

- **Apache Spark™**
  - a fast and general engine for large-scale data processing (이전)
  - **a unified analytics engine for large-scale data processing** (현재)

단순 빅데이터 처리 -> 본격 분석기술로의 발전

스파크(Spark)란?

- 스파크 = 빅데이터처리 + 데이터분석 + SQL

스파크(Spark)란?

- **스파크 = 빅데이터처리 + 데이터분석 + SQL**

- 빅데이터처리(하둡과 유사한 기술)
  - 배치처리 / 실시간처리(스트리밍) 지원

스파크(Spark)란?

- **스파크 = 빅데이터처리 + 데이터분석 + SQL**

- 빅데이터처리(하둡과 유사한 기술)
  - 배치처리 / 실시간처리(스트리밍) 지원
- 데이터 분석(R / 파이썬 데이터 사이언스)
  - 머신러닝(딥러닝은 별도의 라이브러리 필요)
  - https://www.nextobe.com/single-post/2017/08/01/상위-10가지-딥러닝-프레임워크

스파크(Spark)란?

- **스파크 = 빅데이터처리 + 데이터분석 + SQL**

- 빅데이터처리(하둡과 유사한 기술)
  - 배치처리 / 실시간처리(스트리밍) 지원
- 데이터 분석(R / 파이썬 데이터 사이언스)
  - 머신러닝(딥러닝은 별도의 라이브러리 필요)
  - https://www.nextobe.com/single-post/2017/08/01/상위-10가지-딥러닝-프레임워크

- 정형데이터처리(SQL)

스파크(Spark)란?

- **스파크 = 빅데이터처리 + 데이터분석 + SQL**

- 빅데이터처리(하둡과 유사한 기술)
  - 배치처리 / 실시간처리(스트리밍) 지원
- 데이터 분석(R / 파이썬 데이터 사이언스)
  - 머신러닝(딥러닝은 별도의 라이브러리 필요)
  - https://www.nextobe.com/single-post/2017/08/01/상위-10가지-딥러닝-프레임워크

- 정형데이터처리(SQL)
- 그래프관련 처리(GraphX)

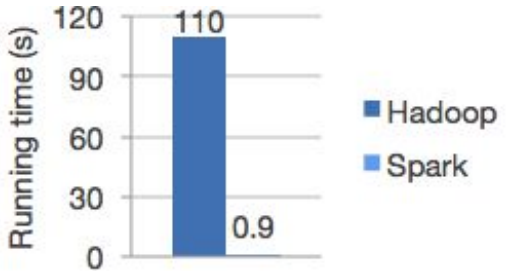
스파크(Spark)란?

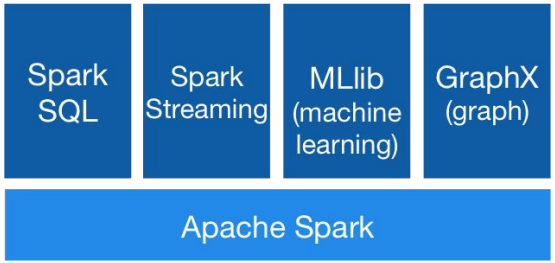
- 속도
  - 하둡보다 100배 이상 빠름(메모리 위주로 처리)
- 사용편의성
  - 자바, **스칼라**, 파이썬, R, SQL지원

```
df = spark.read.json("logs.json")

df.where("age > 21").select("name.first").show()
```

- SQL, 스트리밍, 분석(머신러닝) 지원
  - 자체 지원
- Scalable 서비스 지원
  - 하둡, 쿠버네티스, 클라우드 지원





스파크(Spark)와 하둡의 비교

- 하둡과의 비교
  - 하둡은 분산파일시스템(HDFS)와 분산처리시스템(MapReduce)의 조합

- **스파크는 별도의 파일시스템 없이 분산처리시스템으로 존재**
  - 하둡 관점에서는 HDFS와 같은 파일시스템 위에서 MapReduce 프레임워크를 대체
  - 스파크는 HDFS상에서 동작할 수도 있고 없어도 동작가능

- 하둡은 디스크기반으로 설계
  - 스파크는 **RDD라는 메모리기반 자료구조**를 사용

하둡과 스파크 비교

Word Count Example

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

수십 라인의 Java 코드를 단 3줄로 만들어 버림

데이터분석의 괴로움 해결

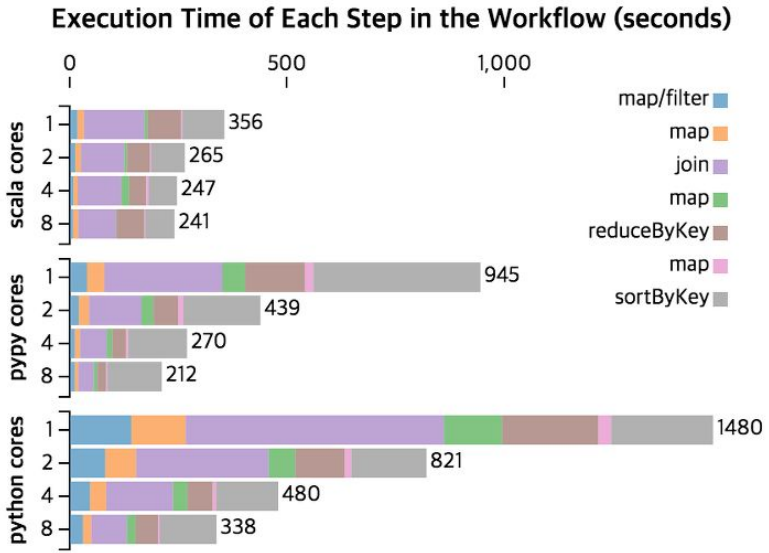
스파크(Spark)란?

- **스칼라(scala) 언어**로 만들어짐(JVM필요)

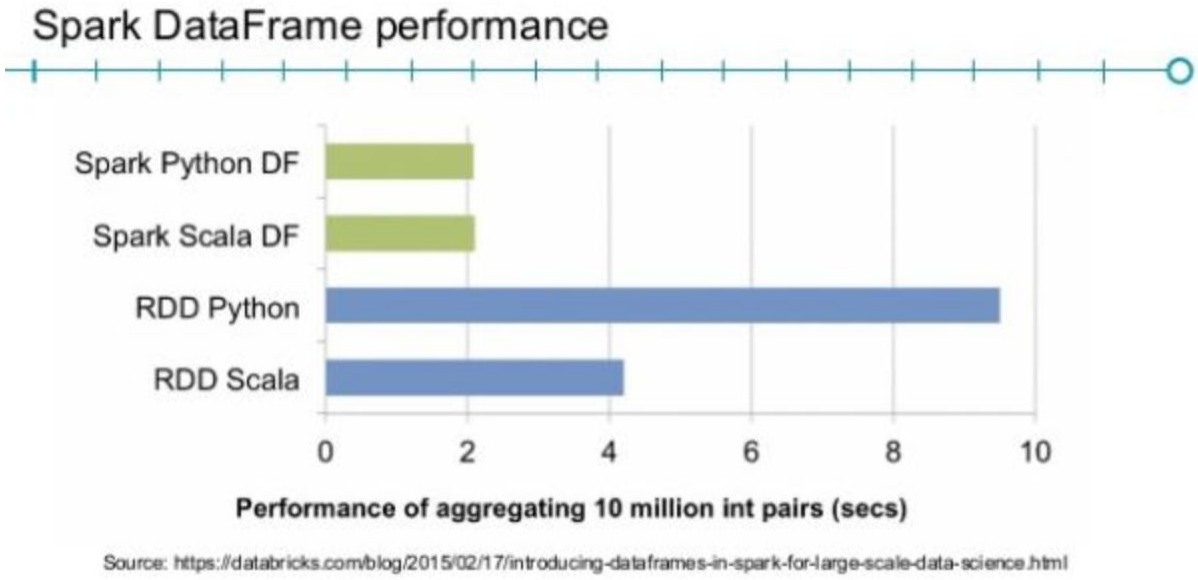
- 자바/파이썬/R 지원
  - **성능은 조금씩 차이가 남**
  - 스칼라보다 파이썬이 느림

- 쉘(Shell)
  - spark-shell(scala), pyspark(python)
  - 자바는 쉘이 없음
  - 한 줄 씩 실행모드

Scala vs. Python(Spark 1.x)



Scala vs. Python(Spark 2.x)



http://man
8%EC%84

Source: https://databricks.com/blog/2015/02/17/introducing-dataframes-in-spark-for-large-scale-data-science.html

스파크(Apache Spark)

- UC 버클리의 AMPlab에서 만든 경량 오픈소스 분산처리 프레임워크
  - Spark: Cluster Computing with Working Sets
- 단순 맵리듀스 외에 SQL/스트리밍/머신러닝이 묶인 구조
- **메모리**를 최대로 활용해 반복작업에 높은 효율
  - 하둡은 디스크기반
- **스칼라(Scala) 언어**로 되어 있슴
  - 자바, 파이썬 지원
  - Spark SQL에서 Language-Integrated queries는 스칼라만,
  - Spark Streaming은 스칼라와 자바,
  - MLlib의 각종 Matrix는 스칼라와 자바에서 지원
  - 셸은 스칼라와 파이썬만 지원

RDD(함수)

- ‘Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing’
  - UC Berkeley, 2012년
- **Read Only**
  - 데이터를 수정가능하면 데이터 유실 시 복구가 어려워진다
- **Fault Tolerant**
  - 데이터 유실 시 복구가능
  - Mapped RDD 중에 일부 데이터가 손실되면 데이터 도출 경로**(Lineage)**를 통해 재계산을 해서 해당 데이터를 다시 복구
- **Lazy Loading**
  - 성능개선을 위해 적용, 실제 사용하기 직전에 로딩

RDD

- Resilient Distributed Dataset
- **메모리(RAM)**에 **읽기 전용**, **파티션**되어 있는 자료
  - Immutable, partitioned collections of records
- 부모로 부터 생성된 **이력(lineage)**만 기록
- **Fault Tolerant** 특성 지원

변환함수/액션함수

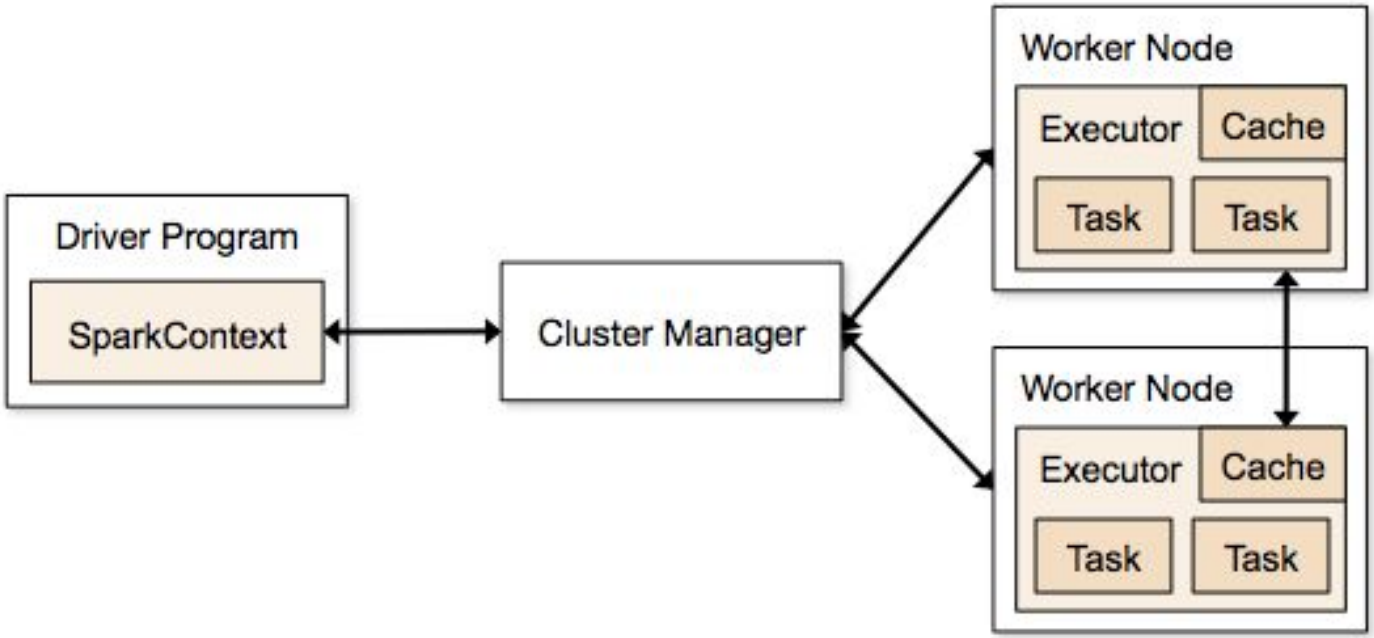
- **Transformation**
  - 바로 실행되지 않고 큐잉(대기)되는 함수
    - 거의 대부분 함수
  - 액션함수가 실행되야 하는 상황 전에 실행됨
- **Action**
  - 호출되면 바로 실행이 되는 함수
  - reduce(*func*)_func_
  - collect()
  - count()
  - take(*n*)
  - saveAsTextFile(*path*)
  - countByKey()
  - foreach(*func*)

스파크 아키텍처

- **드라이버 프로그램(Driver Program)**
  - **SparkContext**가 포함된 메인 프로그램
  - RDDs의 transform이나 조작을 정의해 놓고, 스파크 마스터에 요청해서 클러스터 매니저를 통해 워커 노드에서 실제적인 처리가 되도록 한다

- **워커노드(Worker Node)**
  - 실제로 데이터를 처리하는 노드
  - 프로그램의 요청이 오면 각각 Executor를 실행->
  - 내부에서 태스크와 캐시를 생성->
  - 프로그램의 코드를 받아 실제적인 데이터처리 한 후 결과값을 저장

스파크 아키텍처


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

분산 클러스터 매니저

- 메소스(mesos)
  - 웹에 최적화되지 않은 일반 애플리케이션을 관리하는 분산 클러스터 관리 플랫폼

- 얀(Yarn)
  - 리소스(Resource) 관리 전문 플랫폼
  - 각 노드의 utilization 관리
  - 하둡2부터 기본 리소스 관리용으로 도입

Scala 언어

스칼라 언어의 특징

- **JVM 상에서 구동** / 자바와 연동가능

- 언어별 평균연봉 1위(2018)
  - 미국기준, 전세계는 7위
  - StackOverflow기준

스칼라 언어의 특징

- 마틴 오더스키(Martin Odersky)가 개발
  - EPFL(스위스 로잔 연방 공과대학교)
- 다중 패러다임 언어
  - 함수형 언어 / 객체지향언어

- **람다 함수**
  - **간단한 함수**
  - **정의코드를 절약**

- **함수형 언어**
  - **함수의 인자/리턴값으로 코드를 넘긴다**
    - **변수와 함수의 구분을 없앰**
  - **코드를 절약 / 개념은 낯섬**

스칼라 언어의 특징

- 변수
  - **val -** immutable
    - 수정불가능한 변수, **스파크에서는 val(RDD)을 써라**
  - **var** - mutable (일반적인 변수)
- 불가능

  val arr = Array(1, 2, 3, 4, 5)

  arr = Array(1, 2, 3)
- 가능

  var arr = Array(1, 2, 3, 4, 5)

  arr(0) = 0

람다식 함수정의(스칼라)

- 일반적인 함수 정의(C/자바계열)

```
int sum(int a, int b) {
   return a+b;
}

print sum(10,20);
```

- 람다식 함수 정의(코드가 짧아짐)

```
(a,b) => (a+b)        입력=>출력
_+_ 도 동일한 결과

print ((a,b)=>(a+b))(10,20)    cf.print((_+_)(10,20))
```

람다정의(파이썬)

- 일반적인 함수 정의

```
def sum(x, y):
...  return x + y
...
>>> sum(10, 20)
30
```

- 람다식 정의

```
(lambda x,y: x + y)(10, 20)
```

함수**형**언어(스칼라)

- 변수 != 함수
  - 함수의 인자/리턴값 => 변수
    C/C++/Java(8이전)
- 변수 = 함수
    Scala/Modern Java(8이후)/Javascript/Python

- 인자/리턴값으로 함수를 넣을 수 있음 return { ... };

```
.flatMap(line => line.split(" "))  # 함수의 인자값으로 함수를 넘겨줌
.map(word => (word, 1))
reduceByKey(_+_)
```

- 결과 코드가 짧아짐(가독성은 낮아짐)

map() 함수의 동작

- 리스트(배열)

  입력RDD [1,2,3,4,5] -> map(a=>(a*a)) -> 출력RDD [1,4,9,16,25]

- 배열 요소별로 인자 넘겨진 함수를 적용한다(**inner loop**)

  scala> val a = Array(1,2,3,4,5)

  a: Array[Int] = Array(1, 2, 3, 4, 5)

  scala> **a.map((a)=>(a*a))**

  res4: Array[Int] = Array(1, 4, 9, 16, 25)

함수형언어(파이썬)

- 함수형 언어
  - 변수의 자리(인자, 리턴값)에 함수를 넣을 수 있슴
  - 변수 != 함수(C/C++/Java)
  - 변수 = 함수(scala/python/modern java)

```
list(map(lambda x: x ** 2, range(5)))
```

  - 리스트의 각 요소값에 함수를 적용하라

- 코드는 짧아지고 가독성은 낮아짐

클로저

- 함수의 리턴값으로 코드를 리턴
- 이 코드 안에 로컬변수가 들어있슴
- 리턴된 커드를 실행 -> 유호

```
def calc():
    a = 3
    b = 5
    return lambda x: a * x + b  # 람다 표현식을 반환
c = calc()
 print(c(1), c(2), c(3), c(4), c(5))
```

전역변수를 써야 하나?

- SW 위기
  - 소프트웨어 엔지니어링
- 전역변수 남발(주요 원인)
  - 캡슐화(OOP의 시작)

- 대안 -> 클로저(closure)

도커설치

- docker.com에서 docker desktop 다운로드 및 설치

- 도커는 리눅스 기술
  - **$ sudo apt install docker.io**
  - 다른 환경에서는 가상화를 사용해서 설치
  - 윈도우 버전은 최근 wsl2기반으로 변경됨

스파크 설치(Docker) + 주피터 노트북

$ docker run -it -p 8888:8888 jupyter/all-spark-notebook

- 토큰을 복사해 웹브라우저로 접속
  - 127.0.0.1:8888?token=.....

- Jupyter Notebook(옛, IPython)을 통해 실습

감사합니다