빅데이터 마스터과정
(DAM)

하석재
CEO, 2HCUBE
sjha72@gmail.com

스파크 설치(Docker) + 주피터 노트북

$ sudo apt install docker.io

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

- 다운로드 후 실행
- 포트포워딩 설정
  - 버추얼박스 - 설정 -네트워크-고급-포트포워딩-+(추가)-8888/8888 추가

- 토큰을 복사해 웹브라우저로 접속
  - 127.0.0.1:8888

- Jupyter Notebook(옛, IPython)을 통해 실습

spark-shell 및 pyspark 실행

$ docker ps -a (컨테이너명 확인)

$ docker exec -it xxxx(컨테이너명) bash

spark-shell

:quit

pyspark

quit()

CTRL-P-Q를 순서대로 입력해서 밖($)로 빠져나오기

스파크 공식 예제

- https://spark.apache.org/examples.html

- RDD
  - WordCount(MapReduce)
  - PI Estimation
- DataFrame
  - TextSearch
  - Simple Data Operations
- Machine Learning
  - Logistic Regression

sc(SparkContext) 정의

- 스파크에서는 sc를 따로 정의하지 않고 바로 쓸 수 있음(Implicit Object)

```
import org.apache.spark.SparkContext
import org.apache.spark.SparkConf

val conf = new SparkConf().setAppName("sample").setMaster("yarn")
val sc = new SparkContext(conf)
```

Apache Spark 세팅

- HDFS 파일복사

  $ hdfs dfs -mkdir /user

  $ hdfs dfs -mkdir /user/ubuntu

  $ hdfs dfs -put $SPARK_HOME/README.md /user/ubuntu/

  $ hdfs dfs -ls /user/ubuntu

스파크쉘(Scala)

```
val textFile = sc.textFile("hdfs://127.0.0.1:9000/user/ubuntu/README.md")
val textFile = sc.textFile("./README.md")

textFile.count()
textFile.first()

val lines = sc.textFile("hdfs://127.0.0.1:9000/user/ubuntu/README.md")
val lineLengths = lines.map(s => s.length)
val totalLength = lineLengths.reduce((a,b) => a+b)
```

WordCount(scala)

```
val textFile = sc.textFile("./README.md")

val counts = textFile.flatMap(line => line.split(" ")).map(word => (word, 1)).reduceByKey(_+_)

counts.saveAsTextFile("./mapreduce_result")
```

메소드 체이닝(Method Chaining)

- 함수의 연속호출
  - 함수의 리턴값을 받아(오브젝트) 다시 해당 오브젝트의 메소드를 호출하는 방식

- a.f().g().h() = ((a.f()).g()).h()

- b = a.f()
  c = b.g()
  d = c.h()

Wordcount(Phase 1)

- 원본(파일)입력
  I am a boy
  You are a girl
  …
- sc.textFile()
  ["I am a boy", "You are a girl", ...]

- map(line => line.split(" ")) 의 결과
  [["I", "am", "a", "boy"], ["You", "are", "a", "girl"]]

- flatMap(line => line.split(" "))
  ["I", "am", "a", "boy", "You", "are", "a", "girl"]

Wordcount(Phase 2)

- map(word => (word, 1)) 하둡의 map()함수의 출력
  [("I",1), ("am",1), **("a",1)**, ("boy",1), ("You",1), ("are",1), **("a",1),** ("girl",1)]

- reduceByKey(_+_) 하둡의 reduce()함수의 출력
  groupByKey()
  [("I",[1]), ("am",[1]), **("a",[1,1]),** ("boy",[1]), ("You",[1]), ("are",[1]), ("girl",[1])]

  reduce((a,b)=>(a+b))
  [("I",1), ("am",1), **("a",2),** ("boy",1), ("You",1), ("are",1), ("girl",1)]

- 최종결과
  ("I",1) ("am",1) ("a",2) ("boy",1) ("You",1) ("are",1) ("girl",1)

Wordcount 결과

- 스파크 출력(**정렬 x**, 원할경우 정렬가능)

  (uses,1)

  (SQL,2)

  (will,1)

  (information,1)

  ...

- 하둡 출력(**정렬 O**)

  For 1

  Hadoop, 1

  about 1

  and 1

  ...

스파크/하둡 결과비교

- 하둡
  - Mapper의 개수 => 입력파일의 크기를 64MB/128MB(블럭사이즈)로 나눈 값
    - Split : 19 (하둡 예전버전은 38)
  - Reducer의 개수는 프로그래밍에서 지정 한다 setReduceTasks(); 기본값 1
    part-r-00000

- 스파크
  - Worker노드의 개수 => 입력파일의 크기를 128MB로 나눈 값
    part-00000 - part-00018

WordCount(pyspark)

```
from pyspark.context import SparkContext
from pyspark.conf import SparkConf

sc = SparkContext.getOrCreate(SparkConf())
text_file = sc.textFile("hdfs://127.0.0.1:9000/user/ubuntu/README.md")
counts = text_file.flatMap(lambda line: line.split(" ")).map(lambda word: (word,
1)).reduceByKey(lambda a, b: a + b)
counts.saveAsTextFile("hdfs://127.0.0.1:9000/user/ubuntu/output_python")
```

WordCount(pyspark)

```
from pyspark.context import SparkContext
from pyspark.conf import SparkConf

sc = SparkContext.getOrCreate(SparkConf())
text_file = sc.textFile("../README.md")
counts = text_file.flatMap(lambda line: line.split(" ")).map(lambda word: (word,
1)).reduceByKey(lambda a, b: a + b)
counts.saveAsTextFile("./output_python")
```

```
!pwd
```

스파크 예제(WordCount - 자바)

```
JavaRDD<String> textFile = sc.textFile("hdfs://...");

JavaPairRDD<String, Integer> counts = textFile
    .flatMap(s -> Arrays.asList(s.split(" ")).iterator())
    .mapToPair(word -> new Tuple2<>(word, 1))
    .reduceByKey((a, b) -> a + b);
counts.saveAsTextFile("hdfs://...");
```

SparkSQL

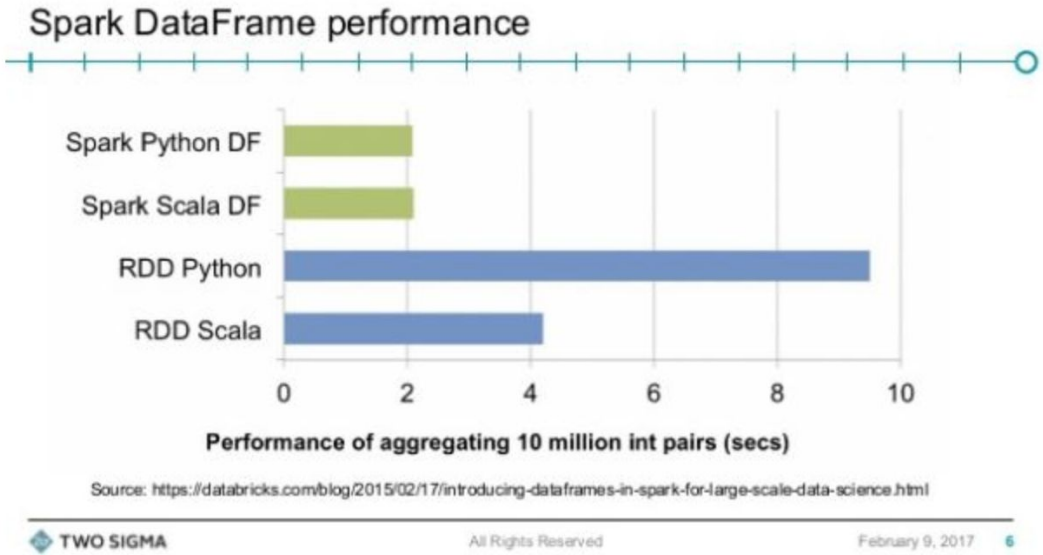
데이터유형

- 정형데이터(Structured Data)
  - RDBMS,XML
  - 스키마(schema)
    - varchar(20) -> 글자수 / null
  - **SparkSession + 데이터셋/데이터프레임**
- 반정형(Semi-structured Data)
  - Schema-less(자유도를 허용)
    - 취미가 없으면 컬럼(key)삭제, 문자열, 배열, 사진/동영상
  - CSV, **JSON**, parquet
  - **SparkSession + 데이터셋/데이터프레임**
- 비정형(Un-structured Data)
  - HTML, 멀티미디어
  - **SparkContext / RDD사용**

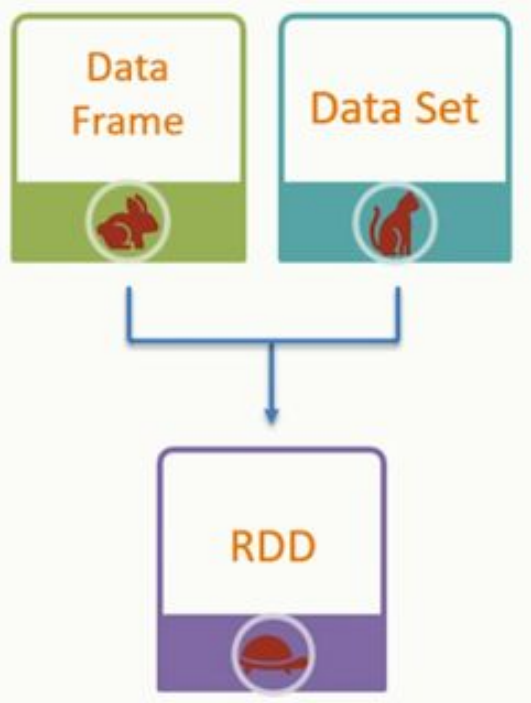
RDD / DataFrame / Dataset의 관계

- 스파크에서 제공하는 데이터추상화
- **RDD(1.0)**
  - 기본 자료구조
  - 내부적으로는 모두 RDD로 변환되어 계산됨
- **DataFrame(1.3)**
  - 데이터의 스키마뷰 제공(DBMS의 테이블과 유사)
  - RDD보다 성능이 개선
    - 사용자정의 메모리관리(프로젝트 텅스텐)
    - 실행계획 최적화(Catalyst Optimizer)
- **Dataset(1.6)**
  - RDD와 DataFrame의 통합 인터페이스
  - 파이썬에서는 지원 안 됨

Scala vs. Python(Spark 2.x - DF/DT)

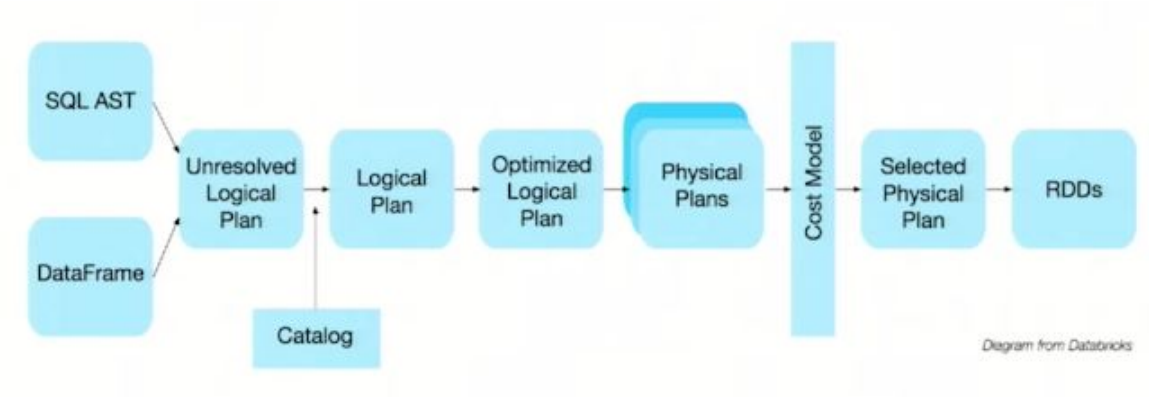


RDD / DataFrame / Dataset의 관계



RDD / DataFrame / Dataset의 관계

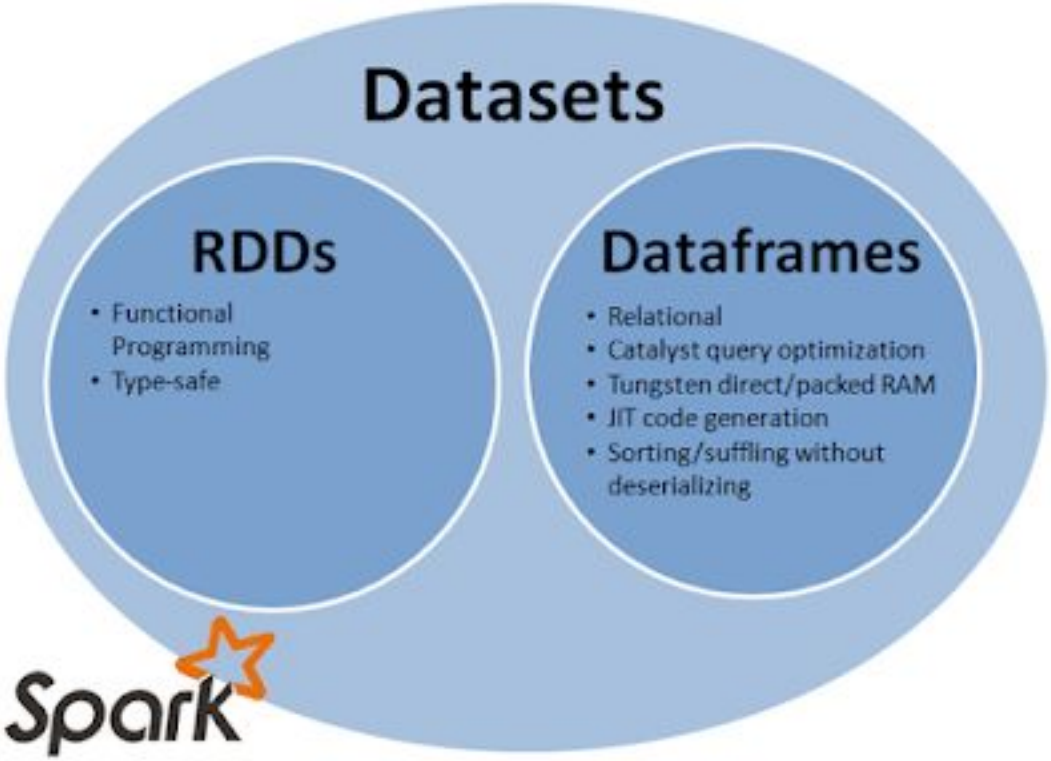
- DataFrame(1.3)
  - 데이터의 스키마뷰 제공(DBMS의 테이블과 유사)
  - RDD보다 성능이 개선
    - 사용자정의 메모리관리(프로젝트 텅스텐)
    - **실행계획 최적화(Catalyst Optimizer)**



Catalyst 최적화기

- 선언적인 API와 인터페이스를 효과적인 컴퓨팅 연산에서 분리한 소프트웨어 계층
- RDBMS의 **CBO(비용)**기반 쿼리최적화 엔진과 유사

RDD / DataFrame / Dataset의 관계



SparkSession

- 스파크 클러스터와 연결, 양방향 통신하는 엔트리포인트(2.0)
- 설정값
  - 마스터 URL, 애플리케이션명, 스파크 홈, JARs

- SparkContext
  - RDD와의 연결
- **SqlContext(1.x) -> SparkSession(2.x 이후)**
  - SparkSQL과 연결
- HiveContext
  - Hive스토어와의 연결

SparkSession

- org.apache.spark.sql.SparkSession(스칼라) / pyspark.sql.sparkSession(파이썬)
- 스칼라/자바
  - Dataset(정형-**Typed**)
- 파이썬과 R
  - DataFrame(반정형-**Untyped**)

- 데이터셋/데이터프레임 -> 내부적으로 RDD유지
- SparkSession은 SparkContext를 캡슐화

SparkSession 생성

- 스칼라 스파크
  - val sparkSession = new SparkSession.builder.master(master_path).appName(“application name”).config(“optional configuration parameters”).getOrCreate()

- spark-shell에서는 자동 생성됨

데이터프레임 생성하기(scala)

- read.json()
  val df =spark.read.json("/a.json")

- RDD에서 변환
  RDD.toDF() cf. DF.toRDD()

- ...

데이터프레임 만들기(파이썬)

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("hello").getOrCreate()
df = spark.read.json("a.json")
df.show()
```

샘플 JSON

[{"id":"123", "name":"Katie", "eyeColor":"brown"},
{"id":"234", "name":"Michael", "age":22},
{"id":"345", "name":"Simone", "age":23, "eyeColor":"blue"}]

- 키:밸류, 배열 [ ] , 오브젝트 { }
  - 중첩(nesting) 가능
- Schema-less

감사합니다