빅데이터 마스터과정
(DAM)

하석재
CEO, 2HCUBE
sjha72@gmail.com

SparkSQL

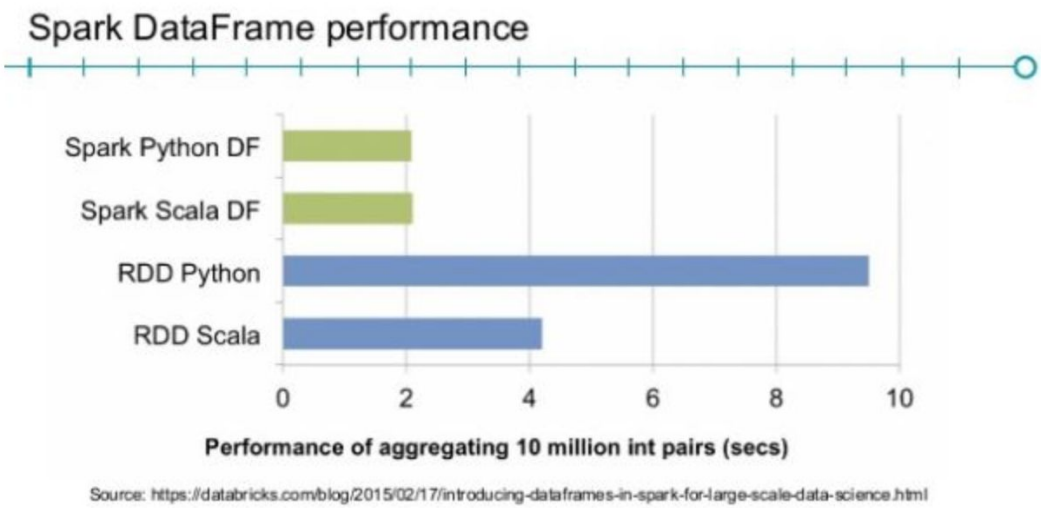
데이터유형

- 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

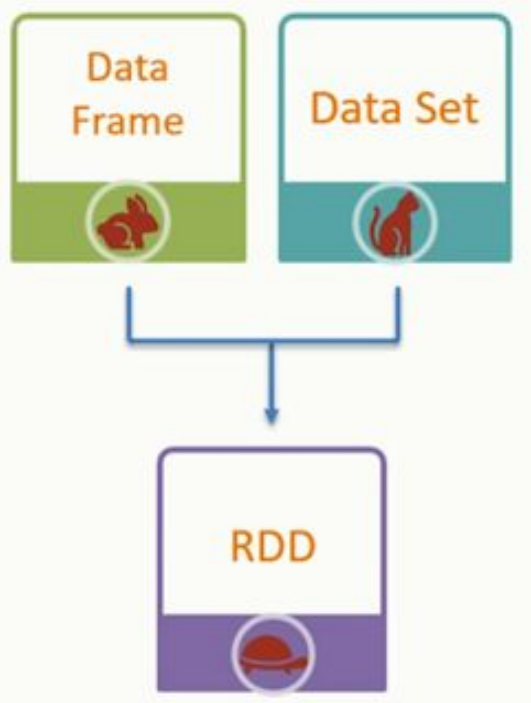
- 스파크에서 제공하는 데이터추상화
- **RDD(1.0)**
  - 기본 자료구조
  - 내부적으로는 모두 RDD로 변환되어 계산됨
- **DataFrame(1.3)**
  - 데이터의 스키마뷰 제공(DBMS의 테이블과 유사)
  - RDD보다 성능이 개선
    - 사용자정의 메모리관리(프로젝트 텅스텐)
    - 실행계획 최적화(Catalyst Optimizer)
- **Dataset(1.6)**
  - RDD와 DataFrame의 통합 인터페이스
  - 파이썬에서는 지원 안 됨

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



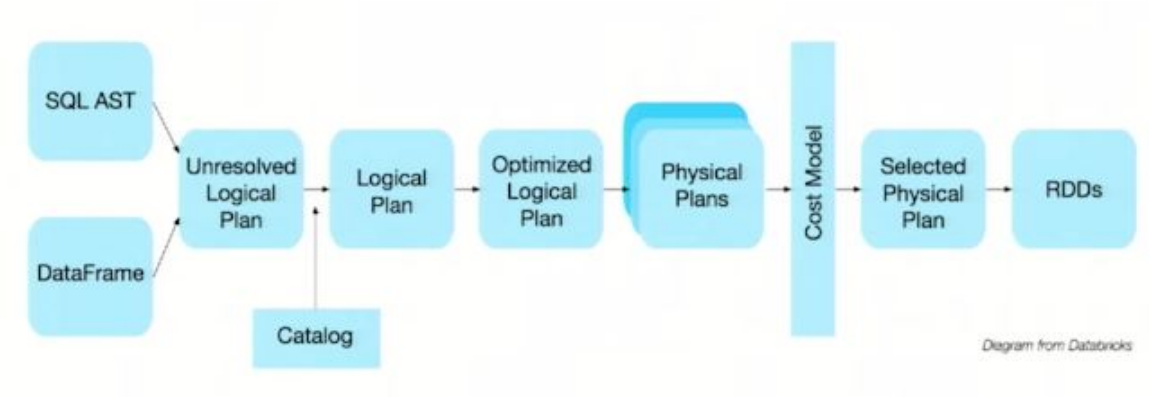
Source: https://databricks.com/blog/2015/02/17/introducing-dataframes-in-spark-for-large-scale-data-science.html

RDD / DataFrame / Dataset의 관계



RDD / DataFrame / Dataset의 관계

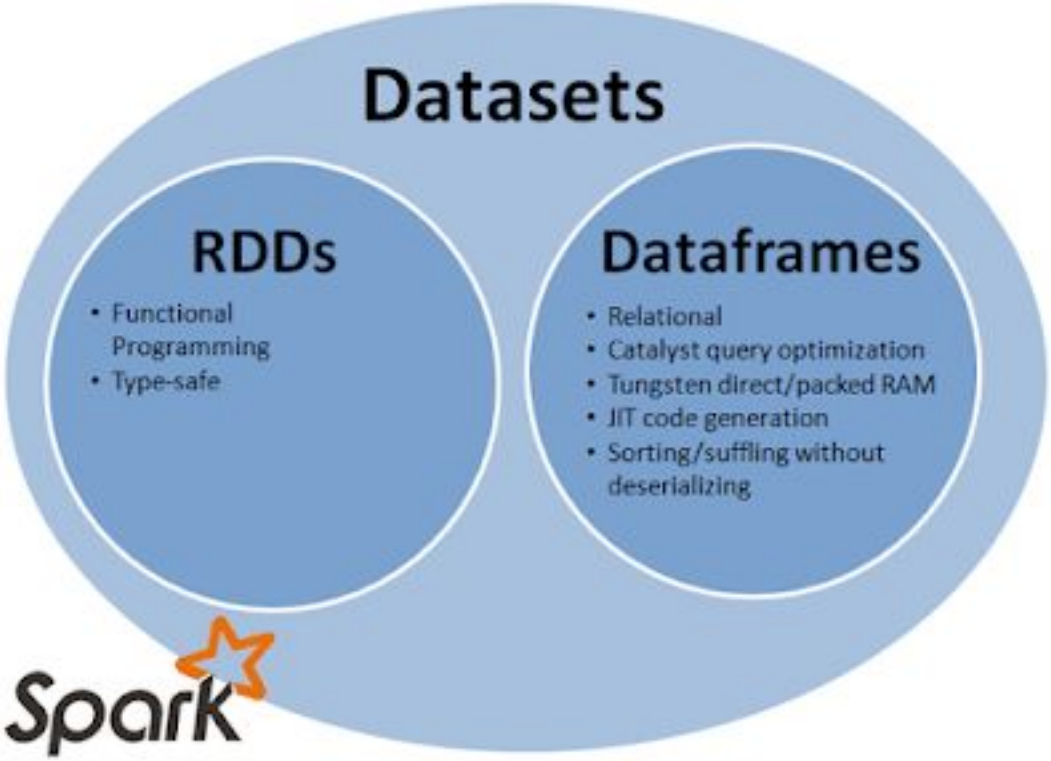
- DataFrame(1.3)
  - 데이터의 스키마뷰 제공(DBMS의 테이블과 유사)
  - RDD보다 성능이 개선
    - 사용자정의 메모리관리(프로젝트 텅스텐)
    - **실행계획 최적화(Catalyst Optimizer)**



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

JSON 포맷의 개선

- 스파크가 지원하는 파일저장포맷 CSV, JSON, XML, **Parquet, ORC**, ...

- 텍스트는 단순하지만 파일사이즈가 큼
- 192.168.99.100(IPv4:4 Byte) vs. 14 Byte
  - ASCII/Unicode/UTF-8(알파벳1바이트, 한글3바이트)
- JSON vs. BSON(Google)

Parquet

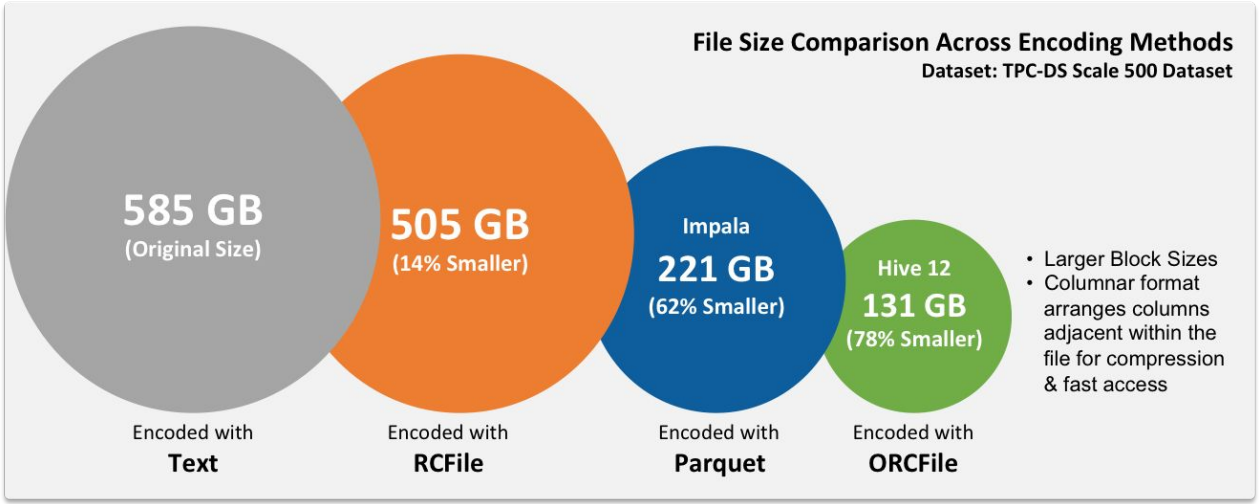
- Dictionary encoding
  - String들을 압축할 때 dictionary를 만들어서 압축하는 방식

- Column pruning
  - 필요한 컬럼만을 읽어 들이는 기법

- Predicate pushdown, row group skipping
  - predicate, 즉 필터를 데이터를 읽어 들인 후 적용하는 것이 아니라 저장소 레벨에서 적용하는 기법

Parquet / ORC

- json vs gzip vs parquet vs. orc
- 용량 gzip > orc > parquet > raw_data(json)
- 쓰기 orc > parquet > raw_data(json) > gzip
- 읽기 parquet > orc > raw_data(json) > gzip



샘플 JSON 데이터셋

- JSON 샘플데이터(100GB)
  https://towardsdatascience.com/interactively-analyse-100gb-of-json-data-with-spark-e018f9436e76

  $ wget --continue http://openlibrary.org/data/ol_cdump_latest.txt.gz (100G)
  $ wget https://s3-eu-west-1.amazonaws.com/csparkdata/ol_cdump.json (133MB)
  $ hdfs dfs -put $HOME/ol_cdump.json /input

gzip / bzip2

- 원본JSON 133MB / gzip 26MB / bzip2 17MB / parquet 30MB / ORC 28M

  $ gzip xxx.json (133M->26M)
  $ gzip -d xxxx.gz
  $ bzip2 xxx.json
  $ bzip2 -d xxx.bz2 (133M->17M)

Parquet/ORC 실습

- Parquet 포맷
  val df = spark.read.json("ol_cdump.json")
  df**.write.parquet**("ol_cdump")
  val df2 = spark**.read.parquet**("ol_cdump")

- ORC 포맷
  val df = spark.read.json("ol_cdump.json")
  df**.write.format("orc").save**("ol_cdump_orc")
  val df3= spark**.read.format("orc").load**("ol_cdump_orc")

데이터프레임(DF) 정의방법

1. read.json() -> DF
2. RDD.toDF() -> DF
    - cf. DF.toRDD()
3. ...

데이터프레임 만들기(파이썬)

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

DataFrame 실습

```
$ spark-shell
import org.apache.spark.sql.SparkSession
val spark = SparkSession.builder().appName("Spark SQL basic
example").config("spark.some.config.option", "some-value").getOrCreate()
val df = spark.read.json("ol_cdump.json")
df.printSchema()
df.createOrReplaceTempView("cdump")
```

DataFrame 실습

```
spark.sql("SELECT count(*) FROM cdump").show()
spark.sql("SELECT * from cdump limit 10").show()
spark.sql("SELECT count(distinct authors) from cdump").show()
spark.sql("SELECT authors,genres,title,type,isbn_10,number_of_pages,languages from cdump  order by title desc limit 10").show()
spark.sql("SELECT authors,genres,title,type,isbn_10,number_of_pages,languages from cdump where genres is not null order by number_of_pages desc limit 10").show()
spark.sql("SELECT authors,genres,title,type,isbn_10,number_of_pages,languages from cdump where title like '%travel%' limit 20").show()
```

DataFrame 실습(조인)

- val person = sc.parallelize(Array((1, "samuel"), (2, "jackson"), (3, "redis"))).toDF("number", "name")
- val address = sc.parallelize(Array(("samuel", "seoul"), ("jackson", "new york"), ("juno", "iceland"))).toDF("name", "address")

- person.createOrReplaceTempView("person")
- address.createOrReplaceTempView("address")

- spark.sql("select * from person join address on person.name = address.name").show()

https://knight76.tistory.com/entry/spark-join-예제

JSON을 이용한 조인

- XML->JSON(중첩O)
- JSON은 외래키개념이 존재하지 않음
  - JSON은 오브젝트 embedding(내포) - 중첩가능

  cf. { }-오브젝트(struct) , [ ]-배열(array) , "key_name": value

- JSON-> DF(Table/View로 변환) -> JOIN
  - JSON이 테이블로 변환될 때의 문제
  - JSON -> 외래키와 조인 개념이 없슴

    -> 오브젝트를 바로 임베딩(embedding)

LATERAL VIEW(deprecated)

{"name":"Yin", "address":{"city":"Columbus","state":"Ohio"}}
{"name":"Michael", "address":{"city":null, "state":"California"}}

```
SELECT
  v1.name, v2.city, v2.state
FROM people
  LATERAL VIEW json_tuple(people.jsonObject, 'name', 'address') v1
    as name, address
  LATERAL VIEW json_tuple(v1.address, 'city', 'state') v2
    as city, state;
```

- https://databricks.com/blog/2015/02/02/an-introduction-to-json-support-in-spark-sql.html

DataFrame 실습(explode())

- https://moons08.github.io/programming/spark_melting/

```
val data = sc.parallelize(Seq(
    """{"userId": 1, "someString": "example1",
        "Date": [20190101, 20190102, 20190103], "val": [1, 2, 9]}""",
    """{"userId": 2, "someString": "example2",
        "Date": [20190101, 20190103, 20190105], "val": [9, null, 6]}"""
))

val df = spark.read.json(data)
```

DataFrame 실습(explode())

- https://moons08.github.io/programming/spark_melting/

```
df.printSchema
/*
root
 |-- Date: array (nullable = true)
 |    |-- element: long (containsNull = true)
 |-- someString: string (nullable = true)
 |-- userId: long (nullable = true)
 |-- val: array (nullable = true)
 |    |-- element: long (containsNull = true)
*/
```

DataFrame 실습(explode())

- https://moons08.github.io/programming/spark_melting/

define zip udf:

```
import org.apache.spark.sql.functions.{udf, explode}

val zip = udf((xs: Seq[Long], ys: Seq[Long]) => xs.zip(ys))

df.withColumn("result", explode(zip('Date, 'val))).
  select('userId, 'someString,
    $"result._1".alias("date"), $"result._2".alias("value")).
    show
```

DataFrame 실습(explode())

- https://moons08.github.io/programming/spark_melting/

```
+------+----------+--------+-----+
|userId|someString|    date|value|
+------+----------+--------+-----+
|     1|  example1|20190101|    1|
|     1|  example1|20190102|    2|
|     1|  example1|20190103|    9|
|     2|  example2|20190101|    9|
|     2|  example2|20190103|    0|
|     2|  example2|20190105|    6|
+------+----------+--------+-----+
```

DataFrame 실습(explode())

- https://moons08.github.io/programming/spark_melting/

스파크 2.4이상에서

```
df.withColumn("result", explode(arrays_zip($"date", $"val"))).
  select($"userId", $"someString", $"result.date", $"result.val").
    show
```

DataSet

- 정형데이터
  - 스키마 정의가 필요함(필수)

  case class People(name: String, age: Long)
  val peopleDF = spark.read.json("people.json")
  val peopleDS = peopleDF.**as**[People]

  scala> peopleDS.show()

- **DataFrame / RDD의 통합**
  peopleDS.filter("age is not null").map(p => p.name + ", " + p.age).show()

감사합니다