# 데이터 불러오기/저장하기

아꿈사 박주희

# 데이터

- 하둡 맵리듀스에 지원하는 inputFormat, outputFormat을 통해 데이터를 접근이 가능하므로 스파크는 넓은 범위의 입출력을 지원한다.

- 스파크가 제공하는 상위 수준 입출력 API의 종류
  - 파일 포맷과 파일 시스템
    - file format: text,Json,SeqFile,Protocol Buffer, etc
    - file system: NFS, HDFS, S3, etc

- - 스파크 SQL을 사용한 구조화된 데이터
    - Json, Apache Hive 같은 구조화된 데이터 지원

- - 데이터베이스와 키/값 저장소
    - 카산드라,HBase, Elastic Search, JDBC support DB

# 파일포맷

- 파일포맷의 범위는 따로 구조가 없는 텍스트 파일 부터, 구조화된 시퀀스 파일까지 지원한다.

표 5-1 일반적으로 지원되는 파일 포맷

| 포맷 이름 | 구조화 여부 | 비고 |
|---|---|---|
| 텍스트 파일 | 아니오 | 고전적인 일반 텍스트 파일. 한 라인을 한 레코드로 간주한다. |
| JSON | 일부 | 범용적인 텍스트 기반 포맷이며 반구조화. 대부분의 라이브러리는 라인을 레코드 하나로 처리한다. |
| CSV | 예 | 매우 흔히 쓰이는 텍스트 기반 포맷. 종종 스프레드시트 애플리케이션에도 쓰인다. |
| 시퀀스 파일 | 예 | 키/값 데이터를 위한 일반적 하둡 파일 포맷. |
| 프로토콜 버퍼 | 예 | 빠르고 효율적으로 공간을 쓰는, 다중 언어를 지원하는 포맷. |
| 오브젝트 파일 | 예 | 공유된 코드끼리 스파크 작업의 데이터를 저장하는 데에 유용하다. 하지만 이는 자바의 객체 직렬화에 의존한 방법이므로 클래스를 변경하면 깨질 위험이 있다. |

# 텍스트 파일

- 텍스트 파일 불러오기
  - 단일 파일의 경우

```
input = sc.textFile("file:///home/holden/repos/spark/README.md")
```

  - 파일이 여러개 일 경우

```
input = sc.wholeTextFile("file:///home/holden/repos/spark/")
```

- 텍스트 파일 저장하기

```
result.saveAsTextFile(outputFile)
```

# JSON

- JSON 데이터를 불러오는 일반적인 방법 :
  1. 텍스트 로딩 후 파서 이용,
  2. JSON 직렬화 라이브러리 이용,
  3. 하둡 포맷을 통한 이용.

- JSON 불러오기

```
import json
data = input.map(lambda x: json.loads(x))
```

JSON 저장하기

```
(data.filter(lamda x: x['lovePandas']).map(lambda x:json.dumps(x))
.saveAsTextFile(outputFile))
```

# 쉼표 구분 데이터와 탭 데이터

- CSV( comma-separated value) or TSV (Tab-separated value) 불러오기

```
import csv
import StringIO
def loadRecord(line):
  #CSV 데이터 한 라인 파싱
  input = StringIO.StringIO(line)
  #주어진 파일의 모든 레코드 읽기 #input = StringIO.StringIO(fileNameContents[1])
  reader = csv.DictReader(input, filenames = ["name", "favouriteAnimal"])
  return reader.next();
input = sc.textFile(inputFile).map(loadRecord)
```

- CSV 저장하기

```
def writeRecords(records)
output = StringIO.StringIO()
writer = csv.DictWriter(output, filednames=["name","favoriteAnimal"])
for record in records:
  writer.writerow(record)
return [output.getValue()]

pandaLovers.mapPartitions(writeRecords).saveAsTextFile(outputFile)
```

# 시퀀스 파일

- 시퀀스 파일은 키/값 쌍의 비중첩 파일로 구성된 하둡의 인기있는 파일 포맷
  - 하둡의 writable 인터페이스를 구현한 데이터로 구성.

표 5-2 기본 타입과 맞는 하둡 Writable 타입들

| 스칼라 타입 | 자바 타입 | 하둡 Writable |
|---|---|---|
| Int | Integer | IntWritable 또는 VIntWritable[2] |
| Long | Long | LongWritable 또는 VLongWritable[2] |
| Float | Float | FloatWritable |
| Double | Double | DoubleWritable |
| Boolean | Boolean | BooleanWritable |
| Array[Byte] | byte [] | BytesWritable |
| String | String | Text |
| Array[T] | T [] | ArrayWritable〈TW〉[3] |
| List[T] | List〈T〉 | ArrayWritable〈TW〉[3] |
| Map[A, B] | Map〈A, B〉 | MapWritable〈AW, BW〉[3] |

# 시퀀스 파일

- 시퀀스 파일 불러오기

```
val data = sc.sequenceFile(inFile, "org.apache.hadoop.io.Text",
"org.apache.hadoop.io.IntWritetable")
```

- 시퀀스 파일 저장하기

```
val data = sc.parallelize(List("Panda",3),("Kay",6))
data.saveASSequenceFile(outputFile)
```

# 오브젝트 파일

- 오브젝트 파일은 시퀀스 파일에 대해 단순한 포장을 더해 값만을 가진 RDD를 다소 편법적인 방법으로 저장할 수 있게 한다.
  시퀀스 파일과는 달리 오브젝트 파일로 값을 쓰는 것은 자바의 직렬화를 사용한다.

- 대부분 스파크 내에서 스파크 작업들끼리 통신하는 용도로 사용되며, 주된 사용 이유는 객체들을 저장하는데 특별한 작업이 없기 때문이다.

# 하둡 입출력 포맷

- 스파크 지원 포맷뿐만 아니라 하둡의 다른 포맷들도 지원한다.

- 다른 하둡 입력 포맷으로 불러오기
  - 신규 하둡 API를 사용시 아래와 같이 셋팅.
    newAPIHadoopFile(Path, 키를 위한 클래스, 값을 위한 클래스, 하둡설정 속성 conf)

- 하둡 출력 포맷으로 저장하기
  자바에서는 페어RDD에서 저장을 지원하는 동일한 편의함수를 지원하지 않음.
  예제에서는 구버전 하둡 포맷 API를 어떻게 사용하여 저장하는지 보여준다.

# 하둡 입출력 포맷

- 파일 시스템 외의 데이터 소스
  HBASE, MongoDB와 같은 키/값 저장소들이 하둡 입력 포맷을 제공 시 HadoopDataSet/saveASHadoopDataset이나 newAPIHadoopDataSet/saveASNewAPIHadoopDataset를 이용해 접근이 가능.

HadoopDataSet류의 함수들은 configuration객체를 통해 데이터 소스에 접근하기 위한 속성들을 셋팅

- HBASE의 경우

```
 val conf = HBaseconfiguration.create()
conf.set(TableInputFormat.INPUT_TABLE, "tableName")

val rdd = sc.newAPIHadoopRDD(conf, classOf[TableInputFormat],
                                    classOf[ImmutableBytesWritable], classOf[Result])
```

# 파일 압축

- 디스크 공간과 네트워크 부하를 줄이기 위해 출력 포맷 중 데이터를 압축하기 위한 압축 코덱을 지정할 수 있으며, 스파크에서는 데이터 읽어들일 시 이러한 압축 타입을 추측 및 다룰 수 있다.

| 포맷 | 분할 가능 | 압축 속도 | 텍스트에 대한 성능 | 하둡 압축 코덱 | 순수 자바 | 기본 옵션 | 비고 |
|---|---|---|---|---|---|---|---|
| zlib | 아니오 | 느림 | 보통 | org.apache.hadoop.io.compress.DefaultCodec | 예 | 예 | 하둡 기본 압축 포맷 |
| Snappy | 아니오 | 매우 빠름 | 낮음 | org.apache.hadoop.io.compress.SnappyCodec | 아니오 | 예 | 스내피의 순수 자바 버전은 존재하나 아직 스파크/하둡에 쓸 수 없다 |
| gzip | 아니오 | 빠름 | 우수 | org.apache.hadoop.io.compress.GzipCodec | 예 | 예 | |
| lzo | 예[6] | 매우 빠름 | 보통 | com.hadoop.compression.lzo.LzoCodec | 예 | 예 | LZO는 모든 작업 노드에 설치 |
| bzip2 | 예 | 느림 | 매우 우수 | org.apache.hadoop.io.compress.Bzip2Codec | 예 | 예 | 분할가능 버전에는 순수 자바를 써야 함 |

# 파일 시스템

- 로컬/일반 FS
  - 스파크는 로컬 파일 시스템에서 파일 불러오기를 지원하지만, 파일들이 클러스터의 모든 노드에서 동일 경로에 있기를 요구한다.

- - 네트워크 파일 시스템을 이용할 경우 각 노드마다 동일한 경로에 데이터가 마운트될 경우 처리가 가능하다.

- 아마존 S3
  - AWS_ACCESS_KEY_ID와 AWS_SECRET_ACESS_KEY 환경 변수에 인증정보를 셋팅후 사용한다.

- HDFS
  -스파크와 HDFS는 동일 머신에 설치 할 수 있으며 이 경우 스파크가 네트워크 오버헤드를 피해 데이터 국지성이 높아지는 장점이 있다. 적용 시 hdfs://master:port/path 식으로 입출력에 지정하면 된다.

-

# 스파크 SQL로 구조화 데이터 다루기

- 스파크 SQL를 통해 SQL 쿼리문을 날려서 데이터 소스에서 실행시키고, 해당 레코드당 하나씩의 ROW객체로 이루어진 RDD를 가져오는 것을 의미.
  - 자바, 스칼라 : 각 Row 객체에는 get(), 기본타입별 getInt(), getString()..등 메소드보유
  - 파이썬: row[column_number], row.column_name으로 가능.

- HIVE
  - 일반 텍스트에서부터 칼럼지향 포맷까지 다양한 형식으로 HDFS 및 다른 저장시스템에 저장 가능. 스파크는 HIVE가 지원하는 어떤 테이블이든 읽을 수 있다.
  : hive-site.xml을 스파크의 conf 디렉토리에 복사하면 스파크SQL로 연결이 가능.

- JSON
  - JSON 데이터가 일관된 스키마를 가지고 있을 경우, 스파크 SQL은 스키마를 예상하여 필요한 데이터를 가져올수 있다.

```
tweets = hivectx.jsonFile("tweets.json")
tweets.registerTempTable("tweets")
result = hivectx.sql("SELECT user.name, text FROM tweets")
```

# 데이터베이스 (자바 데이터베이스 연결)

JDBC 연결이 가능한 어떠한 관계형DB의 데이터든 불러오기 가능.
단 해당 데이터베이스가 스파크에서 병렬로 읽어들일수 있는 부하를 견딜수 있어야 한다.

- 연결방식: org.apache.spark.rdd.JdbcRDD를 생성한 후 Sparkcontext와 다른인자들을 전달하여 연결

```
JdbcRDD( Sparkcontext,
    getconnection: () => connection,
    sql: String,
    lowerBound: Long,  => 카운트 쿼리를 미리 실행하여 upperBound와 lowerBound를 설정한다.
    upperBound: Long,
    numPartitions: Int,
    mapRow: (ResultSet) => T = JdbcRDD.resultSetToObjectArray
)
```

# 데이터베이스 (기타 데이터 소스)

- 카산드라

  - 스파크 카산드라 커넥터는 현재 스파크에 포함되어 있지 않으므로 빌드파일에 추가 의존성을 설정해야 하며 현재 스칼라와 자바만을 지원하고 있다.

- HBase

  - org.apache.hadoop.hbase.mapreduce.TableInputFormat을 구현한 하둡 입력 포맷을 써서 HBASE에 접근할 수 있다.

- 일래스틱 서치

  - 일래스틱서치-하둡을 써서 일랙스틱서치로부터 데이터를 읽어 오거나 쓸 수 있다.

  : 해당 커넥터는 사용자가 제공한 경로를 무시하고 Sparkcontext에 세팅된 설정을 사용함.
  또한 ouputFormat은 스파크의 wrapper 타입을 잘 사용하지 않으므로

- saveASHadoopDataSet을 통해 수동으로 설정해야 한다.