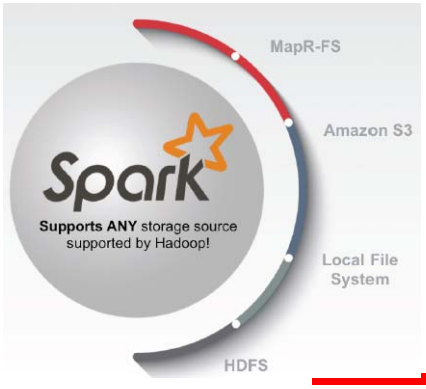


스파크 스트리밍

순천향대학교 컴퓨터공학과

이 상 정

학습 내용

1. 스파크 스트리밍 아키텍처
2. 센서 데이터 응용 – 데이터 조사
3. 센서 데이터 응용 – 스트리밍 처리
4. 센서 데이터 추가 응용
5. 단어 카운트 응용

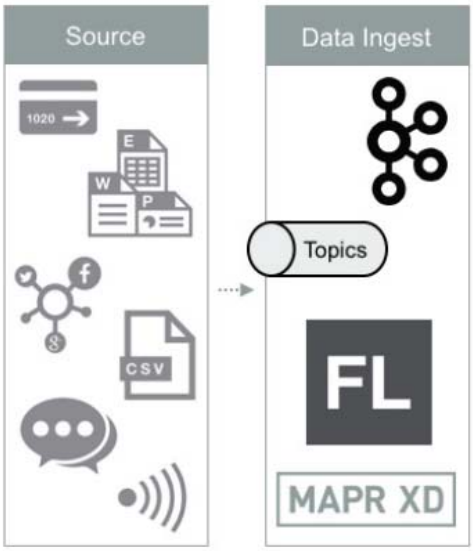
1. 스파크 스트리밍 아키텍처

스트림 처리 아키텍처 – 데이터 소스

- 일반적으로 스트림 처리 아키텍처(streaming processing architecture)는 외부의 소스에서 데이터 스트림을 수집
  - 센서 네트워크, 모바일 응용, SNS, 웹 클라이언트, 서버의 로그, 사물 통신의 사물 ("Things" from the Internet of Things)

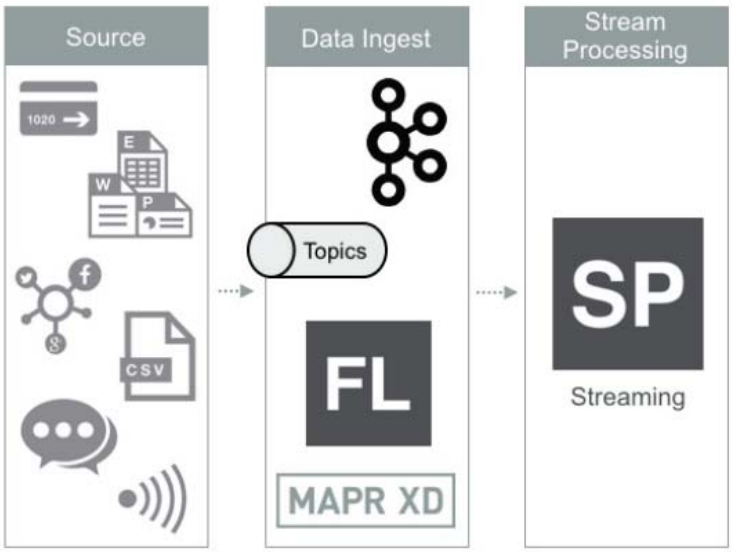
스트림 처리 아키텍처 – 데이터 수집

- 데이터는 Kafka, Flume, Redis, MAPR Streams, 또는 내장된 파일 시스템 등과 같은 메시징 시스템을 경유하여 전달



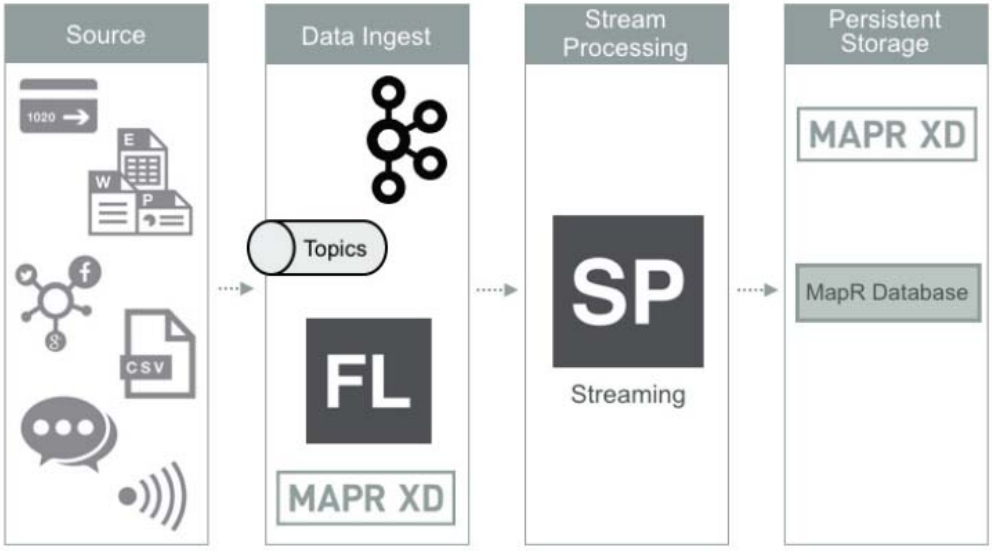
스트림 처리 아키텍처 – 데이터 처리

- 데이터는 스파크 스트리밍 등과 같은 스트림 처리 시스템 상에서 처리



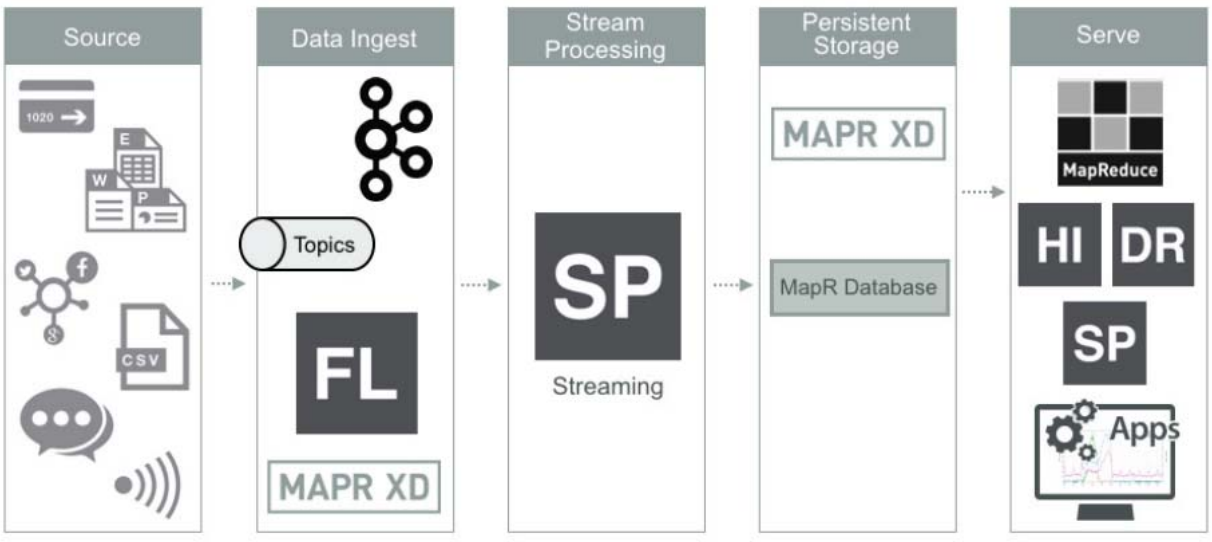
스트림 처리 아키텍처 – 데이터 저장

- 처리된 데이터는 HBase, Cassandra, MAPR-DB 등과 같은 NoSQL 데이터베이스에 저장



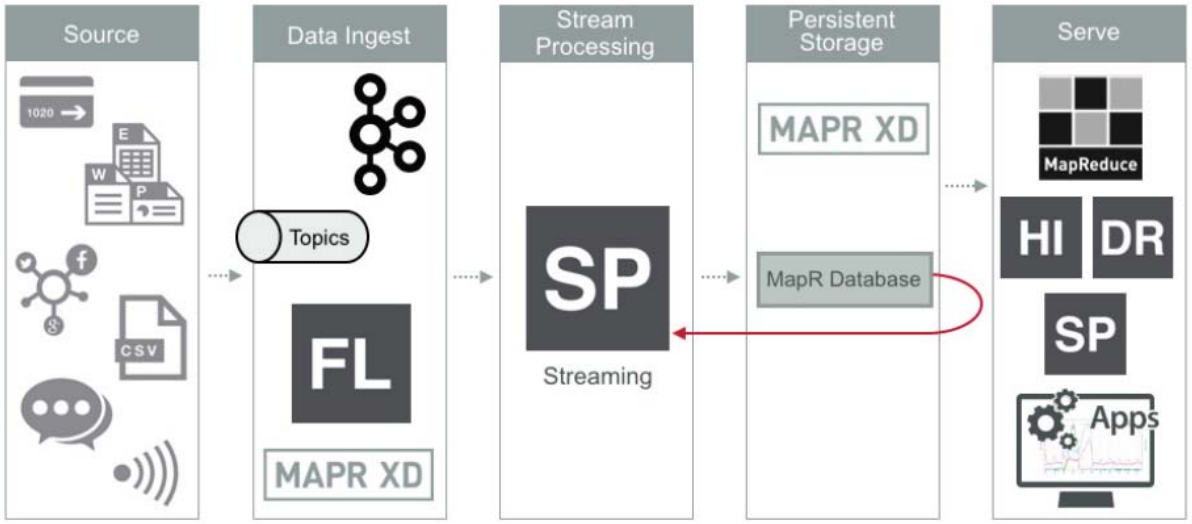
스트림 처리 아키텍처 – 분석

- 대시보드, 분석 툴 등과 같은 종단의 응용들이 처리된 데이터를 사용



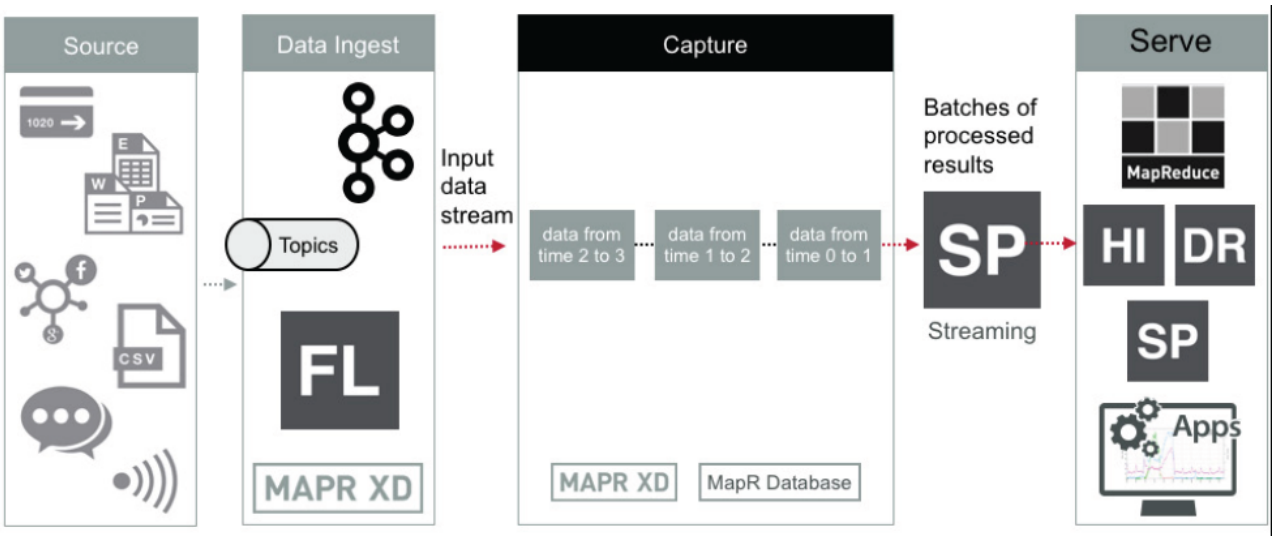
스트림 처리 아키텍처 – 출력 저장

- 출력의 값의 차후 처리를 위해 데이터베이스에 다시 저장 될 수도 있음



스파크 스트림 처리 – 배치

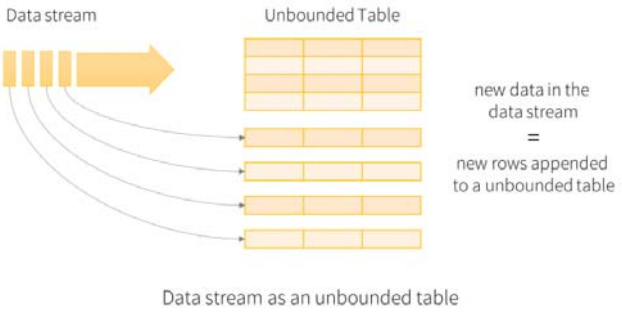
- 연속적인 스트림 데이터 처리를 위해 특정 구간 단위로 분할한 배치(묶음, batch)으로 처리



스파크 스트림 처리 – 구조적 스트리밍

❑ 구조적 스트리밍 (structured streaming)

- 스파크는 라이브 데이터 스트림을 연속적으로 추가되는 테이블로 관리



- 신용카드 트랜잭션 예

Time

Transactions

id	first	last	amt	city	…
11894	Jane	Roberts	1255.76	San Jose	
90083	Rajesh	Gidwani	504.12	Edinburgh	
16884	Hanna	Park	75.99	Hamburg	
66792	Ankur	Dawar	446.90	Madison	

구조적 스트리밍 – 추가

❑ 새로운 트랜잭션 데이터가 들어오면 각 레코드가 테이블의 행으로 추가 (append)

- 테이블의 크기는 제한이 없음

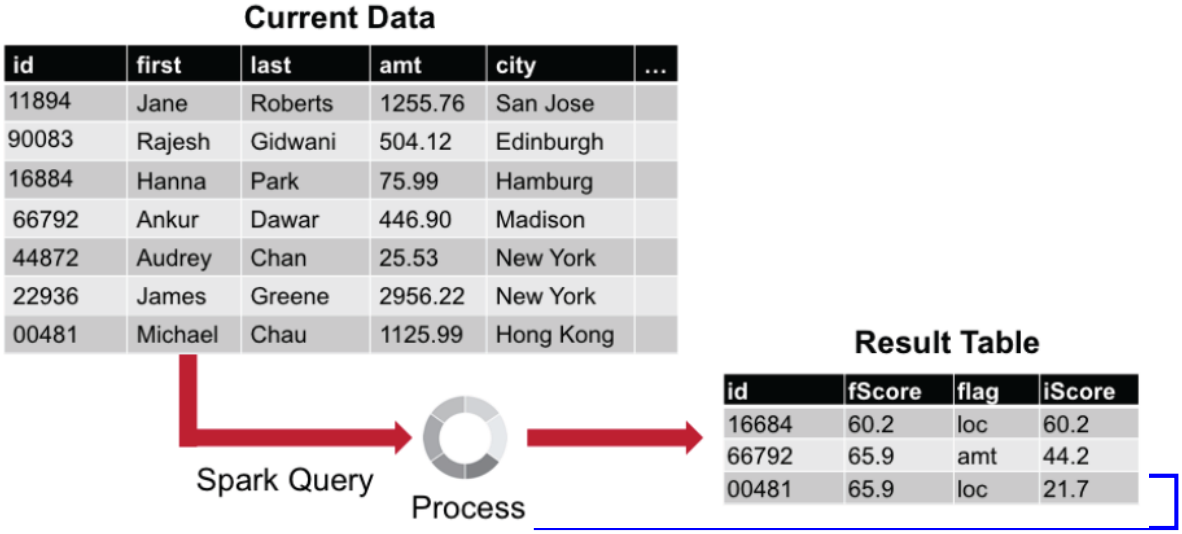
Time

Transactions

id	first	last	amt	city	…
11894	Jane	Roberts	1255.76	San Jose	
90083	Rajesh	Gidwani	504.12	Edinburgh	
16884	Hanna	Park	75.99	Hamburg	
66792	Ankur	Dawar	446.90	Madison	
44872	Audrey	Chan	25.53	New York	
22936	James	Greene	2956.22	New York	
00481	Michael	Chau	1125.99	Hong Kong	

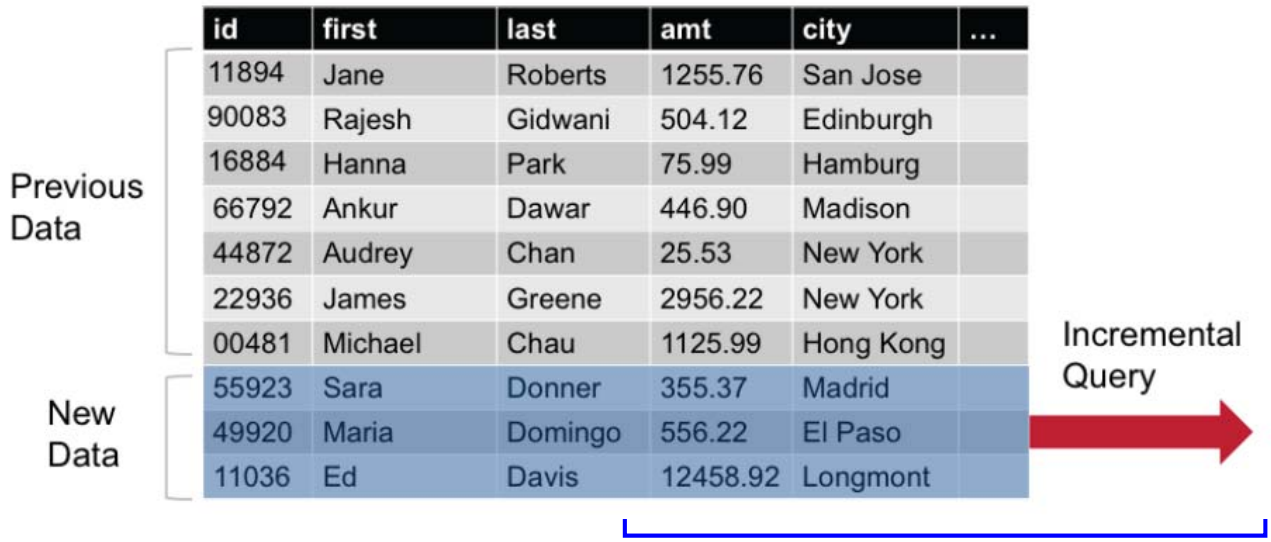
구조적 스트리밍 – 처리 (질의)

- 특정 시점에서의 데이터 처리
  - 신용카드 사기 여부 조사를 위해 특정 시점에서의 데이터 분석
  - 고객의 현재 레코드와 이전 기록을 비교하여 분석
  - 신용카드 사기로 의심되는 결과를 출력 (Result Table)



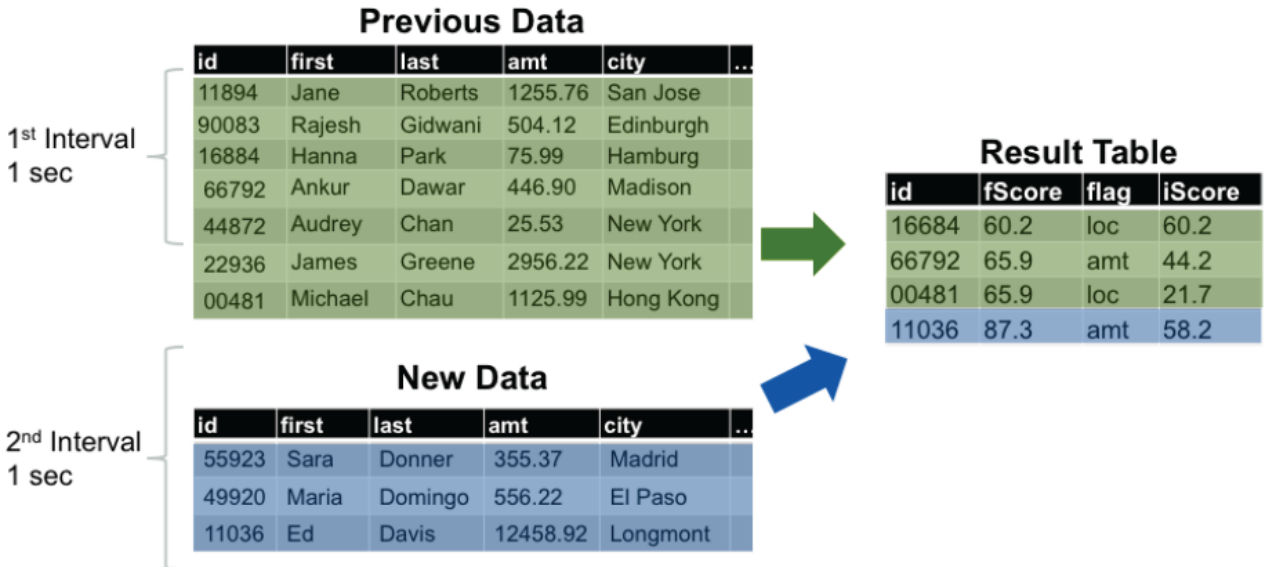
구조적 스트리밍 – 증분 질의

- 특정 시점 질의 후에도 데이터는 연속적으로 계속 추가
- 새로운 질의 요청 시에는 이전 질의 이후에 추가된 데이터에 대해서만 증분 질의(incremental query) 처리



구조적 스트리밍 – 인터벌 처리

- 각 인터벌(예를 들어 1초) 동안 새로운 행들이 입력 테이블에 추가되고 처리되고 출력 테이블을 갱신



구조적 스트리밍 – 저장

- 출력 테이블이 3가지 모드로 저장
  - 완료 (Complete)
  - 추가 (Append) – 디폴트 모드
  - 갱신 (Update)

Save output as:
- Complete
- Append
- Update

구조적 스트리밍 – 완료 저장 모드

❑ 완료 저장 모드는 전체 갱신된 테이블이 외부 저장소에 저장

The entire Result Table is written to external storage each interval

(No fraud detected at Time Interval 1)

Time Interval 2

Input Table

id	amount	city
90083	504.12	Edinburgh
16884	75.99	Hamburg
66792	446.90	Madison

Result Table

id	fScore	flag	iScore
16684	60.2	loc	60.2
66792	446.90	amt	44.2

Output Saved

id	fScore	flag	iScore
16684	60.2	loc	60.2
66792	446.90	amt	55.2

Time Interval 3

Input Table

id	amount	city
90083	504.12	Edinburgh
16884	75.99	Hamburg
66792	446.90	Madison
00481	1125.99	Hong Kong

Result Table

id	fScore	flag	iScore
16684	70.1	loc	60.2
66792	65.9	amt	44.2
00481	65.9	loc	21.7

Output Saved

id	fScore	flag	iScore
16684	70.1	loc	60.2
66792	65.9	amt	44.2
00481	65.9	loc	21.7

new row

updated row

구조적 스트리밍 – 추가 저장 모드

❑ 추가 저장 모드는 새로이 추가된 출력 테이블 행들만 저장

- 이전 인터벌의 결과는 이미 저장 되었으므로 저장할 필요가 없는 경우 사용
- 갱신된 행은 저장하지 않음
  - 아래 예의 ID 16684

Only new rows in the result table are written each interval

(No fraud detected at Time Interval 1)

Time Interval 2

Input Table

id	amount	city
90083	504.12	Edinburgh
16884	75.99	Hamburg
66792	446.90	Madison

Result Table

id	fScore	flag	iScore
16684	60.2	loc	60.2
66792	446.90	amt	44.2

Output Saved

id	fScore	flag	iScore
16684	60.2	loc	60.2
66792	446.90	amt	55.2

Time Interval 3

Input Table

id	amount	city
90083	504.12	Edinburgh
16884	75.99	Hamburg
66792	446.90	Madison
00481	1125.99	Hong Kong

Result Table

id	fScore	flag	iScore
16684	70.1	loc	60.2
66792	65.9	amt	44.2
00481	65.9	loc	21.7

Output Saved

id	fScore	flag	iScore
00481	65.9	loc	21.7

new row

updated row

구조적 스트리밍 – 갱신 저장 모드

- 갱신 저장 모드는 갱신된 출력 테이블의 행들만 저장
  - 갱신 또는 추가된 행들이 저장

Updated rows in the result table are written each interval (new rows are considered updated)

new row

updated row

(No fraud detected at Time Interval 1)

Input Table

Time Interval 2

id	amount	city
90083	504.12	Edinburgh
16884	75.99	Hamburg
66792	446.90	Madison

Time Interval 3

id	amount	city
90083	504.12	Edinburgh
16884	75.99	Hamburg
66792	446.90	Madison
00481	1125.99	Hong Kong

Result Table

id	fScore	flag	iScore
16684	60.2	loc	60.2
66792	446.90	amt	44.2

id	fScore	flag	iScore
16684	70.1	loc	60.2
66792	446.90	amt	44.2
00481	65.9	loc	21.7

Output Saved

id	fScore	flag	iScore
16684	60.2	loc	60.2
66792	446.90	amt	55.2

id	fScore	flag	iScore
16684	70.1	loc	60.2
00481	65.9	loc	21.7

스트리밍 데이터프레임/데이터세트

- 스트리밍 데이터 상에서 데이터프레임/데이터세트를 생성
  - readStream() 메서드

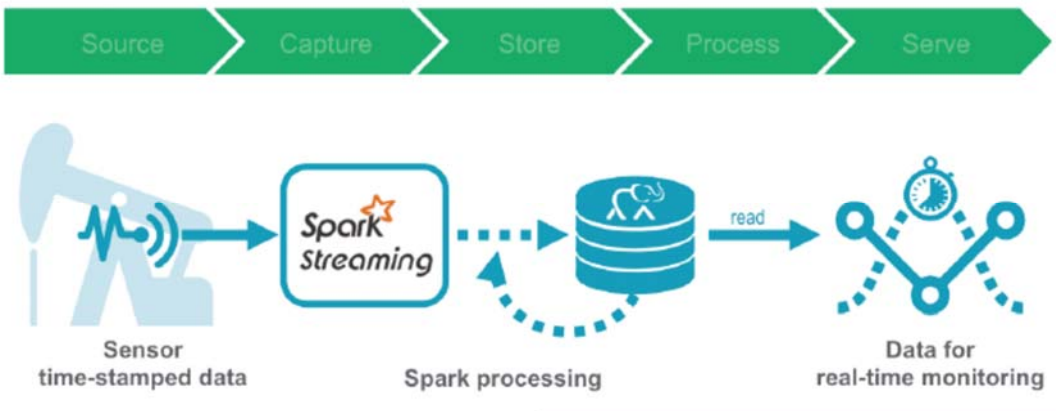
```
val spark =
SparkSession.builder.appName("SensorData").getOrCreate()

val sensorCsvDF = spark.readStream ("sep", ",")
.schema(userSchema) // Specify schema of the csv files
.csv("/path/to/directory") // Equivalent to format("csv")
```

2. 센서 데이터 응용 – 데이터 조사

스파크 스트리밍

센서 데이터 모니터링 예 – 타임 시리즈 데이터

- 석유 시추 시설의 센서 데이터를 모니터링하는 예
  - 석유 시추 설비의 오일 펌프 센서들이 스트리밍 데이터를 생성
  - 스파크가 처리하여 HBase에 저장
    - 일별 스파크 처리 내용은 요약 통계로 집계(aggregate)
    - 데이터를 필터링하고 알람도 저장
    - 다양한 분석 및 리포팅 툴이 저장된 내용 사용



센서 데이터 모니터링 예 – 데이터 조사

- 오일 펌프 센서 데이터는 CSV 파일 형태로 제공
  - 실제 응용에서는 실시간으로 생성된 센서 데이터를 메시징 시스템을 경유하여 스파크에 입력
  - sensordata.csv
    - resid (시설 ID), date, time, hz (주기 ?), disp (displacement, 배수량 ?), flow (유량), sedPPM (sediment, 침전물), psi (오일 압력), chlPPM (chlorine, 염소)
- 스트리밍 처리 전에 센서 데이터 조사
  - 센서 데이터 스키마 정의하고 적재하여 데이터 프레임 생성
  - 적재된 데이터 확인
  - 데이터 총 개수 확인
  - 오일의 압력이 0.5 psi(pound per square inch) 이하인 레코드 조사
  - 오일 압력의 일별 통계 조사
  - SQL을 사용하여 센서 데이터의 일별 통계 값 계산

센서 데이터 다운로드 (1)

- 센서 데이터 예제 디렉토리: ~/spark/oil

- 디렉토리 생성 및 다운로드
  - oil 디렉토리 생성
    ```
    $ mkdir ~/spark/oil
    $ cd ~/spark/oil
    ```
  - 강의 홈페이지에서 다운로드
    ```
    $ wget http://cs.sch.ac.kr/lecture/BigData/download/sensordata.csv
    ```
    - 또는, MapR 사이트에서 예제 소스 및 입력 데이터 다운로드
      ```
      $ wget http://course-files.mapr.com/DEV3600-R2/DEV362Files.zip
      $ unzip DEV362Files.zip
      ```

센서 데이터 다운로드 (2)

```
bigdata@master:~$
bigdata@master:~$ mkdir ~/spark/oil
bigdata@master:~$ cd ~/spark/oil
bigdata@master:~/spark/oil$
bigdata@master:~/spark/oil$ wget http://cs.sch.ac.kr/lecture/BigData/download/sensordata.csv
--2020-06-16 02:45:24--  http://cs.sch.ac.kr/lecture/BigData/download/sensordata.csv
Resolving cs.sch.ac.kr (cs.sch.ac.kr)... 220.69.209.31
Connecting to cs.sch.ac.kr (cs.sch.ac.kr)|220.69.209.31|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 2386833 (2.3M) [text/csv]
Saving to: 'sensordata.csv'

sensordata.csv              100%[=========================================>]   2.28M  7.80MB/s

2020-06-16 02:45:24 (7.80 MB/s) - 'sensordata.csv' saved [2386833/2386833]

bigdata@master:~/spark/oil$ more sensordata.csv
COHUTTA,3/10/14,1:01,10.27,1.73,881,1.56,85,1.94
COHUTTA,3/10/14,1:02,9.67,1.731,882,0.52,87,1.79
COHUTTA,3/10/14,1:03,10.47,1.732,882,1.7,92,0.66
COHUTTA,3/10/14,1:05,9.56,1.734,883,1.35,99,0.68
COHUTTA,3/10/14,1:06,9.74,1.736,884,1.27,92,0.73
COHUTTA,3/10/14,1:08,10.44,1.737,885,1.34,93,1.54
COHUTTA,3/10/14,1:09,9.83,1.738,885,0.06,76,1.44
COHUTTA,3/10/14,1:11,10.49,1.739,886,1.51,81,1.83
COHUTTA,3/10/14,1:12,9.79,1.739,886,1.74,82,1.91
COHUTTA,3/10/14,1:13,10.02,1.739,886,1.24,86,1.79
COHUTTA,3/10/14,1:15,9.54,1.74,886,0.66,98,0.79
COHUTTA,3/10/14,1:16,10.27,1.741,887,1.98,85,1.72
COHUTTA,3/10/14,1:18,10.29,1.741,887,0.19,94,1.23
COHUTTA,3/10/14,1:19,9.8,1.742,887,0.41,93,0.87
COHUTTA,3/10/14,1:21,10.36,1.742,887,1.09,87,1.23
COHUTTA,3/10/14,1:22,10.12,1.743,888,0.91,77,1.06
```


센서 데이터 하둡 적재

- 입력 데이터 로컬 파일 sensordata.csv를 하둡 파일 시스템의 파일로 복사
  - 로컬 파일: ~/spark/oil/sensordata.csv
  - 하둡 파일: /sparkdata/oil/sensordata.csv

```
$ hadoop fs -mkdir /sparkdata/oil
$ hadoop fs -put sensordata.csv /sparkdata/oil
```

```
bigdata@master:~/spark/oil$
bigdata@master:~/spark/oil$ hadoop fs -mkdir /sparkdata/oil
bigdata@master:~/spark/oil$ hadoop fs -put sensordata.csv /sparkdata/oil
bigdata@master:~/spark/oil$
bigdata@master:~/spark/oil$ hadoop fs -ls /sparkdata/oil
Found 1 items
-rw-r--r--   3 bigdata supergroup    2386833 2020-06-16 02:49 /sparkdata/oil/sensordata.csv
bigdata@master:~/spark/oil$
```

센서 데이터 조사 – 데이터프레임 생성

```
import spark.implicits._
import org.apache.spark.sql.types._

// 센서 데이터 스키마 정의
val userSchema = new StructType().add("resid", "string").add("date",
"string").add("time", "string").add("hz", "double").add("disp",
"double").add("flow", "double").add("sedPPM", "double").add("psi",
"double").add("chlPPM", "double")

// 데이터 적재하고 데이터 프레임 생성
val sensorDF = spark.read.format("csv").option("header",
"false").schema(userSchema).load("/sparkdata/oil/sensordata.csv")
sensorDF.printSchema()                    // 스키마 확인
```


센서 데이터 조사 – 데이터프레임 생성 실행

FINISHED

```
import spark.implicits._
import org.apache.spark.sql.types._

// 센서 데이터 스키마 정의
val userSchema = new StructType().add("resid", "string").add("date", "string").add("time", "string").add("hz", "double"
    ).add("disp", "double").add("flow", "double").add("sedPPM", "double").add("psi", "double").add("chlPPM", "double")

// 데이터 적재하고 데이터 프레임 생성
val sensorDF = spark.read.format("csv").option("header", "false").schema(userSchema).load("/sparkdata/oil/sensordata.csv")
sensorDF.printSchema()      // 스키마 확인

import spark.implicits._
import org.apache.spark.sql.types._
userSchema: org.apache.spark.sql.types.StructType = StructType(StructField(resid,StringType,true), StructField(date,StringTyp
e,true), StructField(time,StringType,true), StructField(hz,DoubleType,true), StructField(disp,DoubleType,true), StructField(f
low,DoubleType,true), StructField(sedPPM,DoubleType,true), StructField(psi,DoubleType,true), StructField(chlPPM,DoubleType,tr
ue))
sensorDF: org.apache.spark.sql.DataFrame = [resid: string, date: string ... 7 more fields]
root
 |-- resid: string (nullable = true)
 |-- date: string (nullable = true)
 |-- time: string (nullable = true)
 |-- hz: double (nullable = true)
 |-- disp: double (nullable = true)
 |-- flow: double (nullable = true)
 |-- sedPPM: double (nullable = true)
 |-- psi: double (nullable = true)
 |-- chlPPM: double (nullable = true)
```

센서 데이터 조사 – 데이터 확인

```
// 적재된 데이터 확인
sensorDF.show(5)
// 데이터 총 갯수 확인
sensorDF.count()
```

```
// 적재된 데이터 확인
sensorDF.show(5)
// 데이터 총 갯수 확인
sensorDF.count()
```

```
+-------+-------+----+-----+-----+-----+------+----+------+
|  resid|   date|time|   hz| disp| flow|sedPPM| psi|chlPPM|
+-------+-------+----+-----+-----+-----+------+----+------+
|COHUTTA|3/10/14|1:01|10.27| 1.73|881.0|  1.56|85.0|  1.94|
|COHUTTA|3/10/14|1:02| 9.67|1.731|882.0|  0.52|87.0|  1.79|
|COHUTTA|3/10/14|1:03|10.47|1.732|882.0|   1.7|92.0|  0.66|
|COHUTTA|3/10/14|1:05| 9.56|1.734|883.0|  1.35|99.0|  0.68|
|COHUTTA|3/10/14|1:06| 9.74|1.736|884.0|  1.27|92.0|  0.73|
+-------+-------+----+-----+-----+-----+------+----+------+
only showing top 5 rows
```

```
res84: Long = 47900
```


센서 데이터 조사 – 오일 압력

```
// 오일 압력이 0.5 psi 이하인 데이터 프레임 생성
val oilDF = sensorDF.filter(col("psi")<0.5)
// val oilDF = sensorDF.select("*").where("psi<0.5")
oilDF.take(3).foreach(println)          // 처음 3개 레코드 확인

// 각 자원에 대한 오일 압력의 일별 통계 조사
sensorDF.groupBy("resid", "date").agg(avg(col("psi"))).show(3)
```

```
// 오일 압력이 0.5 psi 이하인 데이터 프레임 생성
val oilDF = sensorDF.filter(col("psi")<0.5)
//val oilDF = sensorDF.select("*").where("psi<0.5")
oilDF.take(3).foreach(println)     // 처음 3개 레코드 확인

// 각 자원에 대한 오일 압력의 일별 통계 조사
sensorDF.groupBy("resid", "date").agg(avg(col("psi"))).show(3)
```

```
oilDF: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [resid: string, date: string ... 7 more fields]
[NANTAHALLA,3/13/14,2:05,0.0,0.0,0.0,1.73,0.0,1.51]
[NANTAHALLA,3/13/14,2:07,0.0,0.0,0.0,1.21,0.0,1.51]
[NANTAHALLA,3/13/14,2:08,0.0,0.0,0.0,1.29,0.0,1.15]
+----------+-------+-----------------+
|     resid|   date|         avg(psi)|
+----------+-------+-----------------+
|      CHER|3/10/14|87.44885177453027|
|     CARGO|3/10/14|87.39352818371607|
|THERMALITO|3/10/14| 87.1169102296451|
+----------+-------+-----------------+
only showing top 3 rows
```

센서 데이터 조사 – SQL 일별 통계

```
// sensorDF의 뷰 등록
sensorDF.createOrReplaceTempView("sensor")

// 각 자원별로 센서 값의 일별 통계 (최대, 최소, 평균)
val sensorStatDF = spark.sql("SELECT resid, date,MAX(hz) as maxhz, min(hz) as
    minhz, avg(hz) as avghz, MAX(disp) as maxdisp, min(disp) as mindisp,
    avg(disp) as avgdisp, MAX(flow) as maxflo, min(flow) as minflo, avg(flow)
    as avgflo,MAX(sedPPM) as maxsedPPM, min(sedPPM) as minsedPPM,
    avg(sedPPM) as avgsedPPM, MAX(psi) as maxpsi, min(psi) as minpsi,
    avg(psi) as avgpsi,MAX(chlPPM) as maxchlPPM, min(chlPPM) as
    minchlPPM, avg(chlPPM) as avgchlPPM FROM sensor GROUP BY
    resid,date")

// 일별 통계 출력
sensorStatDF.show(3)
```


센서 데이터 조사 – SQL 일별 통계 실행

SPARK JOBS FINISHED

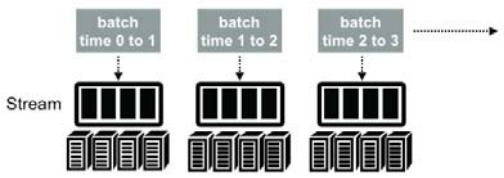
```
// sensorDF의 뷰 등록
sensorDF.createOrReplaceTempView("sensor")

// 각 자원별로 센서 값의 일별 통계 (최대, 최소, 평균)
val sensorStatDF = spark.sql("SELECT resid, date,MAX(hz) as maxhz, min(hz) as minhz, avg(hz) as avghz, MAX(disp) as maxdisp
    , min(disp) as mindisp, avg(disp) as avgdisp, MAX(flow) as maxflo, min(flow) as minflo, avg(flow) as avgflo,MAX(sedPPM)
    as maxsedPPM, min(sedPPM) as minsedPPM, avg(sedPPM) as avgsedPPM, MAX(psi) as maxpsi, min(psi) as minpsi, avg(psi) as
    avgpsi,MAX(chlPPM) as maxchlPPM, min(chlPPM) as minchlPPM, avg(chlPPM) as avgchlPPM FROM sensor GROUP BY resid,date")

// 일별 통계 출력
sensorStatDF.show(3)
```

```
sensorStatDF: org.apache.spark.sql.DataFrame = [resid: string, date: string ... 18 more fields]
+----------+-------+-----+-----+------------------+-------+-------+------------------+------+------+------------------+------
---+---------+------------------+------+------+----------------+---------+---------+------------------+
|     resid|   date|maxhz|minhz|             avghz|maxdisp|mindisp|           avgdisp|maxflo|minflo|            avgflo|maxsed
PPM|minsedPPM|         avgsedPPM|maxpsi|minpsi|           avgpsi|maxchlPPM|minchlPPM|         avgchlPPM|
+----------+-------+-----+-----+------------------+-------+-------+------------------+------+------+------------------+------
---+---------+------------------+------+------+----------------+---------+---------+------------------+
|      CHER|3/10/14| 10.5|  9.5| 9.998726513569954|  3.172|  1.653|2.4233079331941516|1492.0| 777.0|1139.4488517745303|
2.0|      0.0|1.0085490605427974| 100.0|  75.0|87.44885177453027|      2.0|      0.5|1.2520772442588726|
|     CARGO|3/10/14| 10.5|  9.5| 9.998507306889353|  3.752|  1.903|2.8525417536534423|1533.0| 778.0|1165.5302713987473|
2.0|      0.0|1.0162630480167023| 100.0|  75.0|87.39352818371607|      2.0|      0.5|1.2492693110647182|
|THERMALITO|3/10/14| 10.5|  9.5|10.000782881002088|  3.407|   1.75|  2.63799478079332|1513.0| 777.0| 1171.419624217119|
2.0|      0.0|0.9875887265135691| 100.0|  75.0| 87.1169102296451|      2.0|      0.5|1.2625782881002088|
+----------+-------+-----+-----+------------------+-------+-------+------------------+------+------+------------------+------
---+---------+------------------+------+------+----------------+---------+---------+------------------+
only showing top 3 rows
```

3. 센서 데이터 응용 – 스트리밍 처리

구조적 스트리밍 처리 절차

- 구조적 스트리밍 처리 절차
  - SparkSession 객체 생성 및 초기화
  - 입력 스트리밍 데이터프레임 생성
    - readStream() 메서드 사용
    - 다양한 소스(파일, 소켓, Kafka, Flume, ..) 및 옵션 지정
  - 스트링 데이터프레임 연산 적용
    - 입력 데이터프레임에 변환 및 액션 적용
    - 새로운 스트리밍 데이터프레임 생성
  - 질의(query)를 설정 및 처리 시작
    - writeStream() 메서드 사용
    - 출력 형식 지정
    - 저장모드는 outputMode()로 지정: complete, append, update
  - 스트리밍 데이터 처리 종료 시까지 대기
    - awaitTermination() 메서드 사용



스트리밍 데이터프레임 생성

- SparkSession 객체 생성 후 데이터프레임 생성
  - readStream() 메서드 사용
    - 파일이 위치한 디렉토리 기술

```
import spark.implicits._
import org.apache.spark.sql.types._

// 센서 데이터 스키마 정의
val userSchema = new StructType().add("resid", "string").add("date","string")
.add("time", "string").add("hz", "double").add("disp", "double").add("flow",
"double").add("sedPPM", "double").add("psi", "double").add("chlPPM", "double")

// 입력 스트림 데이터프레임 생성
val sensorCsvDF = spark.readStream.option("sep" , ",").schema(userSchema)
.csv("/sparkdata/oil/")
```


스트리밍 데이터프레임 생성 실행

```
import spark.implicits._
import org.apache.spark.sql.types._

// 센서 데이터 스키마 정의
val userSchema = new StructType().add("resid", "string").add("date","string") .add("time", "string").add("hz", "double").add("disp", "double").add("flow",
    "double").add("sedPPM", "double").add("psi", "double").add("chlPPM", "double")

// 입력 스트림 데이터프레임 생성
val sensorCsvDF = spark.readStream.option("sep" , ",").schema(userSchema) .csv("/sparkdata/oil/")
```

FINISHED

```
import spark.implicits._
import org.apache.spark.sql.types._
userSchema: org.apache.spark.sql.types.StructType = StructType(StructField(resid,StringType,true), StructField(date,StringType,true), StructField(time,Strin
gType,true), StructField(hz,DoubleType,true), StructField(disp,DoubleType,true), StructField(flow,DoubleType,true), StructField(sedPPM,DoubleType,true), Str
uctField(psi,DoubleType,true), StructField(chlPPM,DoubleType,true))
sensorCsvDF: org.apache.spark.sql.DataFrame = [resid: string, date: string ... 7 more fields]
```

Took 3 sec. Last updated by admin at June 16 2020, 12:07:48 PM.

스트리밍 데이터프레임 연산 및 질의

- 스트링 데이터프레임 연산 적용 질의(query)를 설정 및 처리 시작
  - 입력 데이터프레임에 변환 및 액션 적용
  - writeStream() 메서드 사용
  - awaitTermination() 메서드 사용

```
// 스트리밍 데이터프레임 연산
val alertOilDF = sensorCsvDF.select("*").where("psi < 0.5")
println("low pressure alert ")

// 질의 설정 및 처리 시작
val query = alertOilDF.writeStream.format("console").start()

// 처리 종료시까지 대기
query.awaitTermination()
```


스트리밍 처리 실행 (1)

```
// 스트리밍 데이터프레임 연산
val alertOilDF = sensorCsvDF.select("*").where("psi < 0.5")
println("low pressure alert ")

// 질의 설정 및 처리 시작
val query = alertOilDF.writeStream.format("console").start()

// 처리 종료시까지 대기
query.awaitTermination()
```

RUNNING 0

```
alertOilDF: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [resid: string, date: string ... 7 more fields]
low pressure alert
query: org.apache.spark.sql.streaming.StreamingQuery = org.apache.spark.sql.execution.streaming.StreamingQueryWrapper@4d60a93d
-------------------------------------------
Batch: 0
-------------------------------------------
+----------+-------+----+---+----+----+------+---+------+
|     resid|   date|time| hz|disp|flow|sedPPM|psi|chlPPM|
+----------+-------+----+---+----+----+------+---+------+
|NANTAHALLA|3/13/14|2:05|0.0| 0.0| 0.0|  1.73|0.0|  1.51|
|NANTAHALLA|3/13/14|2:07|0.0| 0.0| 0.0|  1.21|0.0|  1.51|
|NANTAHALLA|3/13/14|2:08|0.0| 0.0| 0.0|  1.29|0.0|  1.15|
|NANTAHALLA|3/13/14|2:10|0.0| 0.0| 0.0|  1.93|0.0|  1.25|
|NANTAHALLA|3/13/14|2:11|0.0| 0.0| 0.0|  1.75|0.0|  1.29|
|NANTAHALLA|3/13/14|2:13|0.0| 0.0| 0.0|  0.27|0.0|  1.24|
|NANTAHALLA|3/13/14|2:14|0.0| 0.0| 0.0|  0.69|0.0|   1.8|
|NANTAHALLA|3/13/14|2:15|0.0| 0.0| 0.0|  1.91|0.0|  1.41|
|NANTAHALLA|3/13/14|2:17|0.0| 0.0| 0.0|  1.12|0.0|  0.74|
```

스트리밍 처리 실행 (2)

- 첫 배치 실행 후 다음 파일이 디렉토리에 복사될 때까지 기다림
  - 로컬 파일 데이터를 다른 이름으로 적재하면, 새로운 데이터로 인식하여 두 번째 배치로 처리
  - 동일한 데이터를 temp.csv로 복사하고, HDFS에 적재

```
bigdata@master:~/spark/oil$
bigdata@master:~/spark/oil$ cp sensordata.csv temp.csv
bigdata@master:~/spark/oil$ hadoop fs -put temp.csv /sparkdata/oil
bigdata@master:~/spark/oil$ hadoop fs -ls /sparkdata/oil
Found 2 items
-rw-r--r--   3 bigdata supergroup    2386833 2020-06-16 02:49 /sparkdata/oil/sensordata.csv
-rw-r--r--   3 bigdata supergroup    2386833 2020-06-16 03:10 /sparkdata/oil/temp.csv
bigdata@master:~/spark/oil$
```


스트리밍 처리 실행 (3)

- temp.csv가 적재되면 두 번째 배치 실행 시작

```
// 질의 설정 및 처리 시작
val query = alertOilDF.writeStream.format("console").start()

// 처리 종료시까지 대기
query.awaitTermination()
```

```
|NANTAHALLA|3/13/14|2:31|0.0| 0.0| 0.0|  1.14|0.0|  0.99|
|NANTAHALLA|3/13/14|2:33|0.0| 0.0| 0.0|  1.46|0.0|  0.69|
+----------+-------+----+---+----+----+------+---+------+
only showing top 20 rows

-------------------------------------------
Batch: 1
-------------------------------------------
+----------+-------+----+---+----+----+------+---+------+
|     resid|   date|time| hz|disp|flow|sedPPM|psi|chlPPM|
+----------+-------+----+---+----+----+------+---+------+
|NANTAHALLA|3/13/14|2:05|0.0| 0.0| 0.0|  1.73|0.0|  1.51|
|NANTAHALLA|3/13/14|2:07|0.0| 0.0| 0.0|  1.21|0.0|  1.51|
|NANTAHALLA|3/13/14|2:08|0.0| 0.0| 0.0|  1.29|0.0|  1.15|
|NANTAHALLA|3/13/14|2:10|0.0| 0.0| 0.0|  1.93|0.0|  1.25|
|NANTAHALLA|3/13/14|2:11|0.0| 0.0| 0.0|  1.75|0.0|  1.29|
|NANTAHALLA|3/13/14|2:13|0.0| 0.0| 0.0|  0.27|0.0|  1.24|
```

스트리밍 처리 실행 (3)

- 제플린에서 스트리밍 처리 종료 방법
  - 제플린에서 스트리밍 데이터를 계속 대기하여 종료할 수 없는 경우에 제플린을 마스터 서버에서 재시작

```
bigdata@master:~$ $ZEPPELIN_HOME/bin/zeppelin-daemon.sh restart
Zeppelin stop                                              [  OK  ]
Zeppelin start                                             [  OK  ]
bigdata@master:~$
bigdata@master:~$ ls
```


스트리밍 처리 응용

- 제플린이 아닌 독립 프로그램 응용 작성
  - 오일의 압력이 0.5 psi(pound per square inch)인 레코드 조사하여 콘솔에 출력
  - 소스 프로그램: ~/spark/oil/src/main/scala/SensorApp.scala

```
bigdata@slave1:~/spark/oil$
bigdata@slave1:~/spark/oil$ mkdir src
bigdata@slave1:~/spark/oil$ mkdir src/main
bigdata@slave1:~/spark/oil$ mkdir src/main/scala
bigdata@slave1:~/spark/oil$
bigdata@slave1:~/spark/oil$ find .
.
./sensordata.csv
./src
./src/main
./src/main/scala
bigdata@slave1:~/spark/oil$
```

스트리밍 처리 응용 – SBT 프로젝트 스크립트

- SBT 프로젝트 스크립트 작성
  - 프로젝트 스크립트, ~/spark/oil/sensor.sbt
  - 라이브러리에 spark-streaming 의존성 추가

```
name := "SensorStream Project"
version := "1.0"
scalaVersion := "2.12.11"
libraryDependencies += "org.apache.spark" %% "spark-core" % "2.4.5"
libraryDependencies += "org.apache.spark" %% "spark-sql" % "2.4.5"
libraryDependencies += "org.apache.spark" %% "spark-streaming" % "2.4.5"
```


SensorApp.scala

```
/* ~/spark/oil/src/main/scala/SensorApp.scala */
// 클래스 임포트
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.types._

// SensorApp 클래스 정의
object SensorApp {
   def main(args: Array[String]) {
            // SparkSession 객체 생성
            val spark = SparkSession.builder.appName("SensorApp").getOrCreate()
            // spark.implicits 임포트
            import spark.implicits._

            // 콘솔 출력 메시지의 수준 조정
            val sc = spark.sparkContext
            sc.setLogLevel("WARN")
```

```
        // 센서 데이터 스키마 정의
        val userSchema = new StructType().add("resid", "string").add("date", "string")
                        .add("time", "string").add("hz", "double").add("disp", "double")
                        .add("flow", "double").add("sedPPM", "double")
                        .add("psi", "double").add("chlPPM", "double")

        // 입력 스트림 데이터프레임 생성
        val sensorCsvDF = spark.readStream.option("sep" , ",").schema(userSchema)
                                .csv("/sparkdata/oil/")

        // 스트리밍 데이터프레임 연산
        val alertOilDF = sensorCsvDF.select("*").where("psi < 0.5")
        println("low pressure alert ")

        // 질의 설정 및 처리 시작
        val query = alertOilDF.writeStream.format("console").start()

        // 처리 종료시까지 대기
        query.awaitTermination()
    }
}
```


SensorApp 응용의 빌드

- 응용 빌드
  - $ sbt package
    - target/scala-2.12/sensor-project_2.12-1.0.jar 파일 생성

```
bigdata@master:~/spark/oil$
bigdata@master:~/spark/oil$ sbt package
[info] Loading project definition from /home/bigdata/spark/oil/project
[info] Loading settings for project oil from sensor.sbt ...
[info] Set current project to SensorStream Project (in build file:/home/bigdata/spark/oil/)
[info] Updating
https://repo1.maven.org/maven2/org/apache/spark/spark-streaming_2.12/2.4.5/spark-streaming_2.12-2.4.5.pom
  100.0% [##########] 9.7 KiB (8.7 KiB / s)
[info] Resolved  dependencies
[info] Fetching artifacts of
https://repo1.maven.org/maven2/org/apache/spark/spark-streaming_2.12/2.4.5/spark-streaming_2.12-2.4.5.jar
  100.0% [##########] 1.1 MiB (1.0 MiB / s)
[info] Fetched artifacts of
[warn] There may be incompatibilities among your library dependencies; run 'evicted' to see detailed eviction warnings.
[info] Compiling 1 Scala source to /home/bigdata/spark/oil/target/scala-2.12/classes ...
[success] Total time: 18 s, completed Jun 16, 2020 3:32:15 AM
bigdata@master:~/spark/oil$
```

```
bigdata@master:~/spark/oil$
bigdata@master:~/spark/oil$ ls target/scala-2.12
classes  sensorstream-project_2.12-1.0.jar  update
bigdata@master:~/spark/oil$
```

SensorApp 응용 실행

- spark-submit 명령을 사용하여 실행
  - hdfs의 /sparkdata/oil 에 새로운 파일 적재 할 때마다 배치 실행

$ $SPARK_HOME/bin/spark-submit --class SensorApp --master yarn target/scala-2.12/sensorstream-project_2.12-1.0.jar

```
bigdata@master:~/spark/oil$ $SPARK_HOME/bin/spark-submit  --class SensorApp  --master yarn  target/scala-2.12/sensorst
ream-project_2.12-1.0.jar
20/06/16 03:37:54 INFO spark.SparkContext: Running Spark version 2.4.5
20/06/16 03:37:54 INFO spark.SparkContext: Submitted application: SensorApp
20/06/16 03:37:54 INFO spark.SecurityManager: Changing view acls to: bigdata
20/06/16 03:37:54 INFO spark.SecurityManager: Changing modify acls to: bigdata

20/06/16 03:38:26 INFO cluster.YarnClientSchedulerBackend: SchedulerBackend is ready
iting maxRegisteredResourcesWaitingTime: 30000(ms)
low pressure alert
-------------------------------------------
Batch: 0
-------------------------------------------
+----------+-------+----+---+----+----+------+---+------+
|     resid|   date|time| hz|disp|flow|sedPPM|psi|chlPPM|
+----------+-------+----+---+----+----+------+---+------+
|NANTAHALLA|3/13/14|2:05|0.0| 0.0| 0.0|  1.73|0.0|  1.51|
|NANTAHALLA|3/13/14|2:07|0.0| 0.0| 0.0|  1.21|0.0|  1.51|
|NANTAHALLA|3/13/14|2:08|0.0| 0.0| 0.0|  1.29|0.0|  1.15|
|NANTAHALLA|3/13/14|2:10|0.0| 0.0| 0.0|  1.93|0.0|  1.25|
|NANTAHALLA|3/13/14|2:11|0.0| 0.0| 0.0|  1.75|0.0|  1.29|
|NANTAHALLA|3/13/14|2:13|0.0| 0.0| 0.0|  0.27|0.0|  1.24|
|NANTAHALLA|3/13/14|2:14|0.0| 0.0| 0.0|  0.69|0.0|   1.8|
|NANTAHALLA|3/13/14|2:15|0.0| 0.0| 0.0|  1.91|0.0|  1.41|
```


4. 센서 데이터 추가 응용

석유 시추 시설 오일 펌프 조사

- 앞의 오일 펌프 센서 입력 데이터 스트림에 대해 데이터를 추가하여 조사
  - 펌프 정보 데이터 추가 입력, sensorvendor.csv
    - 자원 ID, 펌프 타입, 구매 일자, 서비스 일자, 제조업자, 경도, 위도
    - 다운로드
      $ wget http://cs.sch.ac.kr/lecture/BigData/download/sensorvendor.csv
  - 유지보수 정보 추가 입력, sensormaint.csv
    - 자원 ID, 이벤트 날자, 서비스 요원, 설명
    - 다운로드
      $ wget http://cs.sch.ac.kr/lecture/BigData/download/sensormaint.csv

- 낮은 압력의 알람 경고가 발생한 센서들의 펌프 제조 업자 및 유지보수 관리 정보는?

추가 데이터 파일 하둡 적재

- 로컬 파일 sensorvendor.csv, sensormaint.csv를 하둡 파일 시스템에 적재
  - /sparkdata/oil/info 디렉토리에 적재

$ hadoop fs -mkdir /sparkdata/oil/info
$ hadoop fs -put sensorvendor.csv /sparkdata/oil/info
$ hadoop fs -put sensormaint.csv /sparkdata/oil/info

```
bigdata@master:~/spark/oil$
bigdata@master:~/spark/oil$ hadoop fs -mkdir /sparkdata/oil/info
bigdata@master:~/spark/oil$ hadoop fs -put sensorvendor.csv /sparkdata/oil/info
bigdata@master:~/spark/oil$ hadoop fs -put sensormaint.csv /sparkdata/oil/info
bigdata@master:~/spark/oil$ hadoop fs -ls /sparkdata/oil/info
Found 2 items
-rw-r--r--   3 bigdata supergroup       4060 2020-06-16 04:10 /sparkdata/oil/info/sensormaint.csv
-rw-r--r--   3 bigdata supergroup        622 2020-06-16 04:09 /sparkdata/oil/info/sensorvendor.csv
bigdata@master:~/spark/oil$
```

석유 시추 시설 오일 펌프 조사 – 코드

```
import spark.implicits._
import org.apache.spark.sql.types._

// 센서 데이터 스키마 정의
val sensorSchema = new StructType().add("resid", "string").add("date", "string").add("time", "string").add("hz", "double").add("disp", "double").add("flow", "double").add("sedPPM", "double").add("psi", "double").add("chlPPM", "double")
// 펌프 정보 스키마 정의
val pumpSchema = new StructType().add("resid", "string").add("pumpType", "string").add("purchaseDate", "string").add("serviceDate", "string").add("vendor", "string").add("longitude", "float").add("lattitude", "float")
// 유지보수 정보 스키마 정의
val maintSchema = new StructType().add("resid", "string").add("eventDate", "string").add("technician", "string").add("description", "string")

// 입력 센서 스트림 데이터프레임 생성
val sensorDF = spark.readStream.option("sep", ",").schema(sensorSchema).csv("/sparkdata/oil/")
```



```
// 펌프 정보 데이터 스트림 생성
val pumpDF = spark.read.format("csv").schema(pumpSchema).load("/sparkdata/oil/info/sensorvendor.csv").toDF("resid", "pumpType", "purchaseDate", "serviceDate", "vendor", "longitude", "lattitude")
// 유지보수 정보 데이터 스트림 생성
val maintDF = spark.read.format("csv").schema(maintSchema).load("/sparkdata/oil/info/sensormaint.csv").toDF("resid", "eventDate", "technician", "description")
// 데이터프레임 뷰 등록
pumpDF.createOrReplaceTempView("pump")
maintDF.createOrReplaceTempView("maint")

// 스트리밍 데이터프레임 연산
val alertDF = sensorDF.select("*").where("psi < 0.5")
alertDF.createOrReplaceTempView("alert")        // 뷰 등록

// 낮은 압력의 알림 경고가 발생한 펌프 제조업자, 유지관리 정보 조사 SQL
val alertpumpmaintViewDF = spark.sql("select s.resid, s.date, s.psi, p.pumpType, p.purchaseDate, p.serviceDate, p.vendor, m.eventDate, m.technician, m.description from alert s join pump p on s.resid = p.resid join maint m on p.resid=m.resid")
println("alert pump maintenance data")
// 질의 설정 및 처리 시작
val query = alertpumpmaintViewDF.writeStream.format("console").start()
// 처리 종료시까지 대기
query.awaitTermination()
```

석유 시추 시설 오일 펌프 조사 – 실행 (1)

```
import spark.implicits._
import org.apache.spark.sql.types._

// 센서 데이터 스키마 정의
val sensorSchema = new StructType().add("resid", "string").add("date", "string").add("time", "string").add("hz", "double").add("disp", "double").add
    ("flow", "double").add("sedPPM", "double").add("psi", "double").add("chlPPM", "double")
// 펌프 정보 스키마 정의
val pumpSchema = new StructType().add("resid", "string").add("pumpType", "string").add("purchaseDate", "string").add("serviceDate", "string").add("vendor"
    , "string").add("longitude", "float").add("lattitude", "float")
// 유지보수 정보 스키마 정의
val maintSchema = new StructType().add("resid", "string").add("eventDate", "string").add("technician", "string").add("description", "string")

// 입력 센서 스트림 데이터프레임 생성
val sensorDF = spark.readStream.option("sep", ",").schema(sensorSchema).csv("/sparkdata/oil/")
// 펌프 정보 데이터 스트림 생성
val pumpDF = spark.read.format("csv").schema(pumpSchema).load("/sparkdata/oil/info/sensorvendor.csv").toDF("resid", "pumpType", "purchaseDate",
    "serviceDate", "vendor", "longitude", "lattitude")
// 유지보수 정보 데이터 스트림 생성
val maintDF = spark.read.format("csv").schema(maintSchema).load("/sparkdata/oil/info/sensormaint.csv").toDF("resid", "eventDate", "technician",
    "description")

// 데이터프레임 뷰 등록
pumpDF.createOrReplaceTempView("pump")
maintDF.createOrReplaceTempView("maint")
```

FINISHED

```
import spark.implicits._
import org.apache.spark.sql.types._
sensorSchema: org.apache.spark.sql.types.StructType = StructType(StructField(resid,StringType,true), StructField(date,StringType,true), StructField(time,Str
ingType,true), StructField(hz,DoubleType,true), StructField(disp,DoubleType,true), StructField(flow,DoubleType,true), StructField(sedPPM,DoubleType,true), S
tructField(psi,DoubleType,true), StructField(chlPPM,DoubleType,true))
pumpSchema: org.apache.spark.sql.types.StructType = StructType(StructField(resid,StringType,true), StructField(pumpType,StringType,true), StructField(purcha
seDate,StringType,true), StructField(serviceDate,StringType,true), StructField(vendor,StringType,true), StructField(longitude,FloatType,true), StructField(l
attitude,FloatType,true))
maintSchema: org...
```

Took 37 sec. Last updated by admin at June 16 2020, 1:17:12 PM.

석유 시추 시설 오일 펌프 조사 – 실행 (2)

```
// 스트리밍 데이터프레임 연산
val alertDF = sensorDF.select("*").where("psi < 0.5")
alertDF.createOrReplaceTempView("alert")          // 뷰 등록

// 낮은 압력의 알림 경고가 발생한 펌프 제조업자, 유지관리 정보 조사 SQL
val alertpumpmaintViewDF = spark.sql("select s.resid, s.date, s.psi, p.pumpType, p.purchaseDate, p.serviceDate, p.vendor, m.eventDate, m.technician, m
    .description from alert s join pump p on s.resid = p.resid join maint m on p.resid=m.resid")
println("alert pump maintenance data")
// 질의 설정 및 처리 시작
val query = alertpumpmaintViewDF.writeStream.format("console").start()
// 처리 종료시까지 대기
query.awaitTermination()
```

RUNNING 0%

```
alert pump maintenance data
-------------------------------------------
Batch: 0
-------------------------------------------
+----------+-------+---+---------+------------+-----------+--------+---------+----------+--------------------+
|     resid|   date|psi| pumpType|purchaseDate|serviceDate|  vendor|eventDate|technician|         description|
+----------+-------+---+---------+------------+-----------+--------+---------+----------+--------------------+
|NANTAHALLA|3/13/14|0.0|HYDROPUMP|    11/27/10|    3/15/11|HYDROCAM|  3/13/14|E. Simmons|    Shutdown Failure|
|NANTAHALLA|3/13/14|0.0|HYDROPUMP|    11/27/10|    3/15/11|HYDROCAM|   3/3/14|E. Simmons|Adjust bearing al...|
|NANTAHALLA|3/13/14|0.0|HYDROPUMP|    11/27/10|    3/15/11|HYDROCAM|  2/26/14|  J.Thomas|          Inspection|
|NANTAHALLA|3/13/14|0.0|HYDROPUMP|    11/27/10|    3/15/11|HYDROCAM|  6/14/13|  J.Thomas|      Tighten Mounts|
|NANTAHALLA|3/13/14|0.0|HYDROPUMP|    11/27/10|    3/15/11|HYDROCAM|  1/12/13|  J.Thomas|          Inspection|
|NANTAHALLA|3/13/14|0.0|HYDROPUMP|    11/27/10|    3/15/11|HYDROCAM|  2/19/12|  J.Thomas|          Inspection|
|NANTAHALLA|3/13/14|0.0|HYDROPUMP|    11/27/10|    3/15/11|HYDROCAM|  3/15/11|  J.Thomas|             Install|
|NANTAHALLA|3/13/14|0.0|HYDROPUMP|    11/27/10|    3/15/11|HYDROCAM|  3/13/14|E. Simmons|    Shutdown Failure|
|NANTAHALLA|3/13/14|0.0|HYDROPUMP|    11/27/10|    3/15/11|HYDROCAM|   3/3/14|E. Simmons|Adjust bearing al...|
|NANTAHALLA|3/13/14|0.0|HYDROPUMP|    11/27/10|    3/15/11|HYDROCAM|  2/26/14|  J.Thomas|          Inspection|
|NANTAHALLA|3/13/14|0.0|HYDROPUMP|    11/27/10|    3/15/11|HYDROCAM|  6/14/13|  J.Thomas|      Tighten Mounts|
```

Started a minute ago.

5. 단어 카운트 응용

단어 카운트 응용 소개

- 파일이 아닌 실시간 스트림으로 입력을 수신하는 예로 단어 카운트 응용 소개
- TCP 소켓으로부터 스트리밍 데이터를 입력 받아 단어의 총 개수를 카운트하여 출력하는 예
  - 드라이버의 9999 포트로 부터 텍스트 스트리밍 데이터 수신
    - 드라이버에서 리눅스의 nc(net cat) 유틸리티 명령을 사용하여 텍스트 전송
      - cat과 유사하지만 파일이 아닌 네트워크 소켓으로부터 읽고 쓰는 명령
    - nc -lk 9999
      - Listen mode, keepalive 옵션
  - 새로운 텍스트 스트리밍 데이터를 받을 때 마다 단어 카운트를 갱신
    - complete 모드

단어 카운트 응용 – 데이터프레임 생성

1. 스트리밍 텍스트 데이터로 구성되는 무제한의 테이블을 갖는 데이터프레임을 생성
   - 스트리밍 텍스트 데이터의 각 라인이 테이블의 행으로 추가
   - 옵션으로 sockt, IP 주소 지정
   - 테이블의 열은 value이고 타입은 문자열

```
// 스트리밍 데이터프레임 생성
val lines = spark.readStream.format("socket").option("host",
"localhost").option("port",9999).load()
lines.printSchema()     // 스키마 출력
```

```
// 스트리밍 데이터프레임 생성
val lines = spark.readStream.format("socket").option("host", "localhost").option("port", 9999).load()
lines.printSchema()

lines: org.apache.spark.sql.DataFrame = [value: string]
root
 |-- value: string (nullable = true)
```


단어 카운트 응용 – 단어 분리 및 카운트

2. 텍스트 라인에서 단어를 분리하여 카운트
   - 각 라인에서 단어를 분리
     - flatMap 연산 수행을 위해 데이터프레임을 데이터세트로 변환
       - as[String]
   - value로 그룹핑되는 wordCounts 데이터프레임 생성

```
// 단어를 분리
val words = lines.as[String].flatMap(_.split(" "))

// 분리된 단어들 카운트
val wordCounts = words.groupBy("value").count()
```

```
// 단어를 분리하여 카운트
val words = lines.as[String].flatMap(_.split(" "))
// 분리된 단어들 카운트
val wordCounts = words.groupBy("value").count()

words: org.apache.spark.sql.Dataset[String] = [value: string]
wordCounts: org.apache.spark.sql.DataFrame = [value: string, count: bigint]
```

단어 카운트 응용 – 질의 설정

3. 질의를 설정하고 스트리밍 계산을 수행을 시작
   - 단어의 총 누적 개수를 질의, outputMode("complete")
   - 스트리밍 데이터 갱신 시 마다 콘솔에 출력
     - writestream, format("console")
   - 질의가 실행 중에 종료를 방지, awaitTermination()

```
// 질의 설정 후 스트리밍 처리 시작
val query = wordCounts.writeStream.outputMode("complete").format("console")
.start()

// 처리 종료시까지 대기
query.awaitTermination()
```


단어 카운트 응용 – 실행 예 (1)

- 드라이버(마스터)에서 net cat 서버 실행

```
bigdata@master:~$ nc -lk 9999
Hi, This is a message from the socket
```

- 스트리밍 처리 제플린 실행

```
// 질의 설정 후 스트리밍 처리 시작
val query = wordCounts.writeStream.outputMode("complete").format("console").start()

// 처리 종료시까지 대기
query.awaitTermination()
```

```
query: org.apache.spark.sql.streaming.StreamingQuery = org.apache.spark.sql.execution
d97ade
-------------------------------------------
Batch: 0
-------------------------------------------
+-------+-----+
|  value|count|
+-------+-----+
|     is|    1|
|    Hi,|    1|
|    the|    1|
|   from|    1|
|   This|    1|
|      a|    1|
|message|    1|
| socket|    1|
+-------+-----+
```

단어 카운트 응용 – 실행 예 (2)

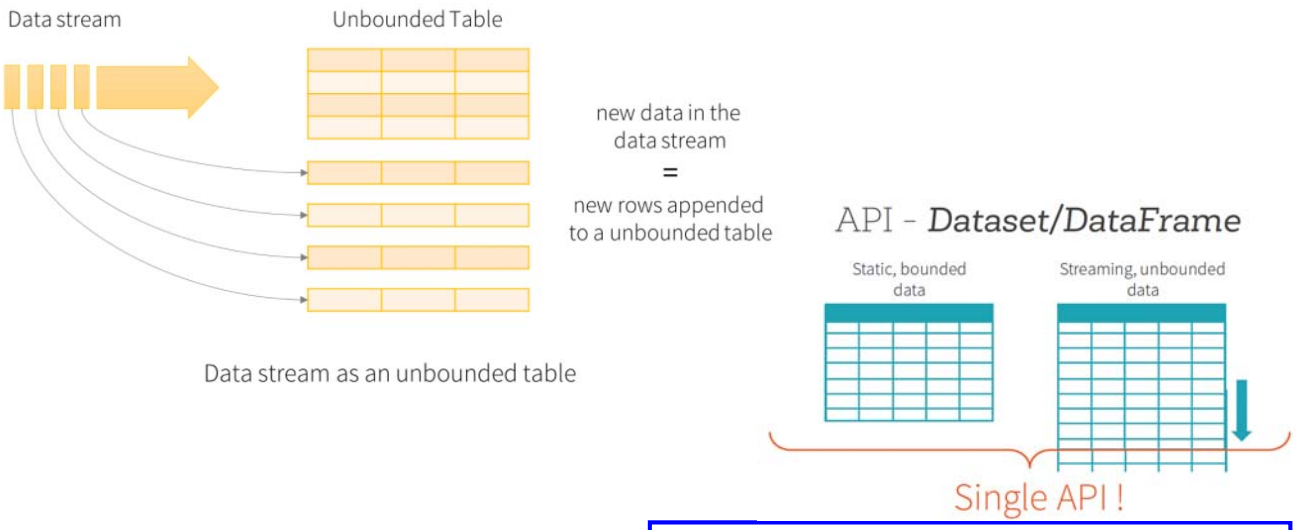
```
bigdata@master:~$ nc -lk 9999
Hi, This is a message from the socket
This message comes from the port 9999 of the socket
```

```
// 질의 설정 후 스트리밍 처리 시작
val query = wordCounts.writeStream.outputMode("complete").format("console").start()

// 처리 종료시까지 대기
query.awaitTermination()
```

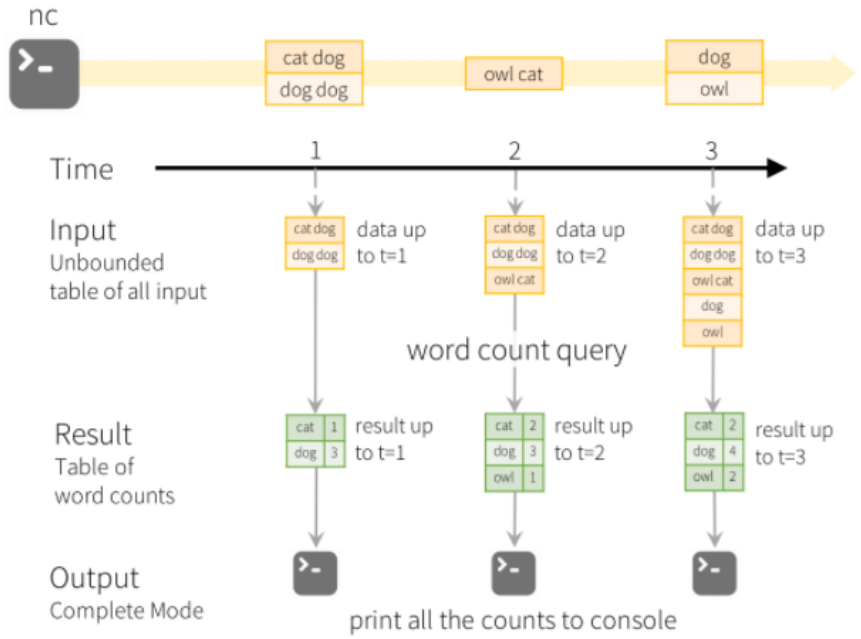
```
-------------------------------------------
Batch: 1
-------------------------------------------
+-------+-----+
|  value|count|
+-------+-----+
|   port|    1|
|  comes|    1|
|     is|    1|
|    Hi,|    1|
|    the|    3|
|   from|    2|
|   9999|    1|
|     of|    1|
|   This|    2|
|      a|    1|
|message|    2|
| socket|    2|
+-------+-----+
```


구조적 스트리밍 기본 개념

- 스트리밍 데이터 도착할 때 마다 테이블에 추가되는 개념적인 모델
  - 배치 처리와 유사한 개념으로 같은 API 사용이 가능



단어 카운트 – 구조화 스트리밍 처리



과제

- 강의 시간의 실습 내용을 정리하여 제출
- 텀 프로젝트 과제
  - 텀 프로젝트 데이터를 사용하여 앞에서 배운 스파크 응용 모니터링을 적용하고 실행

참고 자료

- Advanced Apache Spark
  - https://learn.mapr.com/series/sparkv2/dev-362-advanced-apache-spark-spark-v21
    - Lesson 6: Create an Apache Spark Streaming Application
- Spark Programming Guide
  - https://spark.apache.org/docs/latest/streaming-programming-guide.html
- Real-Time Streaming Data Pipelines with Apache APIs: Kafka, Spark Streaming, and HBase
  - https://mapr.com/blog/real-time-streaming-data-pipelines-apache-apis-kafka-spark-streaming-and-hbase/