HBase 데이터베이스

순천향대학교 컴퓨터공학과

이 상 정

학습 내용

1. HBase 소개
   - HBase 개요, 데이터 모델, 아키텍처
2. HBase 설치
3. HBase 셸
4. 스파크 스트리밍의 HBase 저장
5. 스파크 HBase 입출력

1. HBase 소개

HBase 개요 (1)

- 열-지향형(Column-Oriented) 데이터 저장
  - 하둡 데이터베이스 라고도 함
  - 구글의 BigTable 기반으로 설계
    - http://labs.google.com/papers/bigtable.html
- 대규모 테이블을 지원하도록 설계된 분산 데이터베이스
  - 수십억 개의 행(rows)과 수백만개의 열(column)
  - 하드웨어의 클러스터 상에서 실행
  - 자동 장애 극복 (automatic fail-over)
  - 수평적 확대 가능 (horizontally scalabable)
    - 자동 샤딩(automatic sharding),
      - 샤딩(sharding): 데이터베이스 분할

HBase 개요 (2)

- NoSQL 데이터베이스 형태
  - SQL 기반 접근 안됨
  - 관계형 데이터베이스 모델 적용 안됨
  - HDFS와 달리 랜덤 실시간(random real-time) CRUD 연산 지원, *CRUD (Create, Retrieve, Update, Delete)*
- 자바로 작성된 오픈 소스
  - 간단한 자바 API
  - Map/Reduce 프레임워크과 통합

데이터 모델

- 데이터는 테이블에 저장
- 테이블은 행(row)들로 구성
  - 행은 고유의 키(unique key)로 참조
    - 키는 바이트의 배열
    - 어떤 자료형도 키가 될 수 있음
      - 문자열, long, 자체의 직렬화 데이터 구조
- 행은 열(column)로 구성되며, 열은 컬럼 패밀리(column family)로 그룹화
- 데이터는 셀(cell)에저장
  - 행 x 컬럼 패밀리 x 열 로 구분
  - 셀의 내용도 바이트의 배열

컬럼 패밀리 (Column Family) (1)

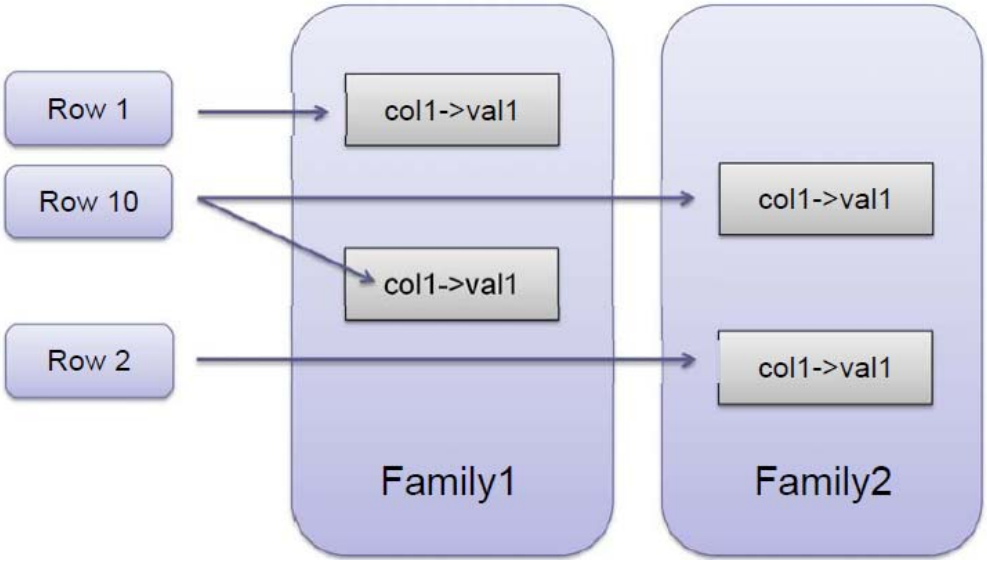
- 열들은 컬럼 패밀리로 그룹화
  - family:column 으로 표시
  - 예, user:first_name
  - 데이터를 조직화하는 방법 제공
  - 컬럼 패밀리에 다양한 특성 적용
    - 압축
    - 메모리 내장 옵션 (in-memory option)
    - HFile/StoreFile 파일에 함께 저장
- 컬럼 패밀리 정의는 정적(static)
  - 테이블과 함께 생성되며, 추가 및 변경이 거의 없음
  - 컬럼 패밀리 수는 제한

컬럼 패밀리 (2)

- 컬럼 패밀리 이름은 인쇄 가능한 문자
  - 키와 셀 값들과 같은 바이트가 아님
- family:column은 셀 값에 대한 태그와 같음
- 반면 열(column)은 정적이 아님
  - 실행 시에 새로운 열 생성
  - 한 패밀리에 대해 수백 만개까지 확장 가능

컬럼 패밀리 (3)

- 행은 Families:Columns 안에 저장된 셀들로 구성



타임스탬프 (Timestamps)

- 셀 값들은 여러 버전을 가짐
  - 각 셀의 여러 개의 버전들이 저장
    - 디폴트로 3
  - 데이터를 구분하는 또 다른 차원
  - 타임스탬프는 영역 서버(region server)가 명식적으로 타임스탬프를 찍거나 또는 클라이언트가 제공
    - 버전들은 시간이 감소하는 내림차순으로 저장
    - 가장 최근의 것들을 먼저 읽음
      - 현재의 값을 읽도록 최적화됨
- 제공되는 버전들의 갯수는 지정 가능

행의 키 (Row Key)

- 행들은 키 값의 사전식(lexicographically)으로 정렬
- 관계형 데이터베이스의 주 키(primary key)와 비슷
  - 고유한 값
  - 최소의 보조 인덱스(seondary index) 지원

셀 (cell)

- Value = Table+RowKey+Family+Column+Timestamp
- 예

Row Key	Time stamp	Name Family		Address Family	
		first_name	last_name	number	address
row1	t1	**Bob**	**Smith**		
	t5			10	First Lane
	t10			30	Other Lane
	t15			**7**	**Last Street**
row2	t20	**Mary**	Tompson		
	t22			**77**	**One Street**
	t30		**Thompson**		

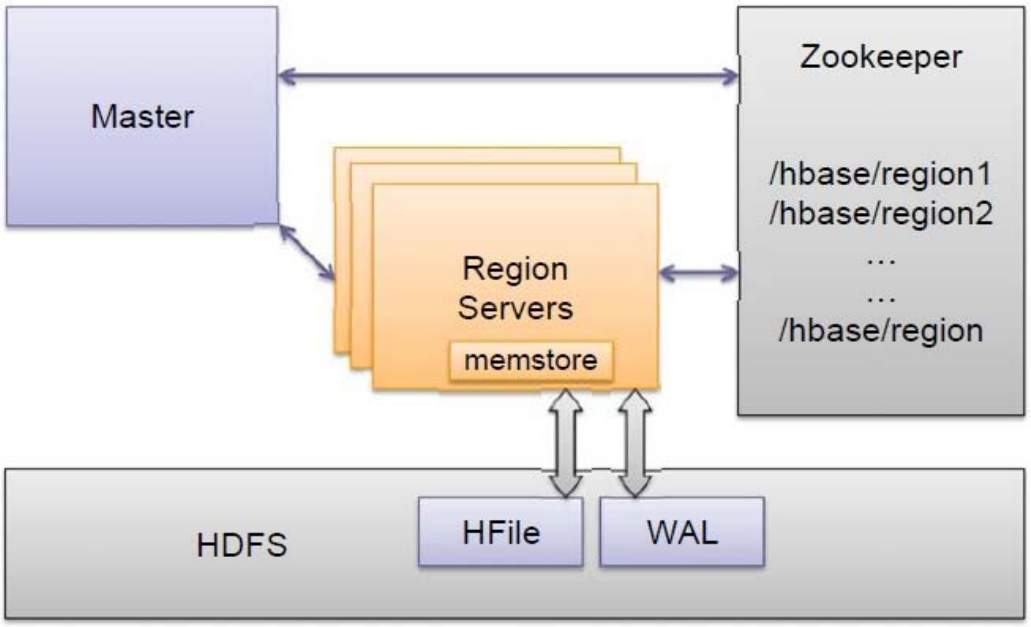
셀의 참조

- 가장 최근의 값 (디폴트)
- 특정 타임스탬프
- 타임스탬프 범위 내의 여러 값들

HBase 아키텍처

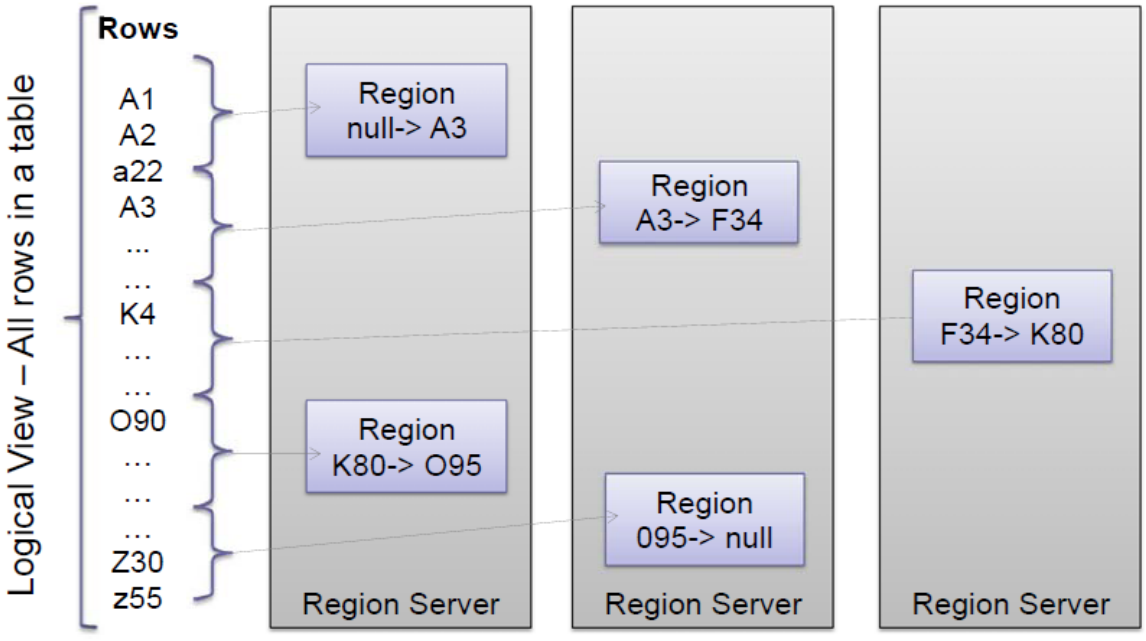
- 테이블은 영역(region)들로 구성
- 영역 (Region)
  - 함께 저장되는 행들의 범위
  - 너무 커지면 동적으로 분할되고, 너무 적으면 동적으로 통합
- 영역 서버 (Region Server)
  - 하나 이상의 영역들을 저장
  - 하나의 영역은 오직 한 개의 영역 서버에 의해 관리됨
- 마스터 서버 (Master Server)
  - HBase 클러스터(즉, 영역 서버들)를 관리하는 데몬
- HBase 의 데이터는 HDFS에 저장
  - HDFS의 높은 가용성(availability)과 고장감내(fault-tolerance) 특성에 의존
- 분산 조정(distributed coordination)을 위해 Zookeeper 사용

HBase 컴포넌트



참조: WAL (Write-Ahead Log)

행들의 영역 서버 간 분산



HBase 영역 (Regions) (1)

- 영역은 키들의 한 범위
  - 시작 키 -> 끝 키 (예, k3cod -> odiekd)
  - 시작 키는 영역에 포함되고, 끝 키는 영역에 포함되지 않음
- 데이터의 추가
  - 처음에는 한 개 영역만 존재
  - 데이터가 추가되어 미리 설정된 최대값의 개수를 넘으면 영역은 분리
    - 디폴트는 256 MB
  - 중간 키에서 2개 영역으로 분리
- 서버 당 영역의 수는 하드웨어 사양에 따라 다양
  - 영역 서버 당 10 ~ 1000개 영역
  - 영역 당 1 GB ~ 2GB 만큼 관리

HBase 영역 (2)

- 데이터를 영역으로 분리 이점
  - 한 영역이 실패한 경우 빠른 회복(fast recovery)이 가능
  - 서버가 과부하된 경우 부하 균형(load balancing)이 가는
    - 서버들 간에 이동됨
- 영역 분리는 빠르게 수행
  - 비동기 프로세스가 영역 분리를 수행하는 동안 원래 파일들에서 읽음
- 이 모든 것들이 사용자의 개입 없이 자동으로 수행

데이터 저장 (Data Storage) (1)

- 데이터는 HDFS의 HFiles/StoreFiles 라는 파일들에 저장
- HFile은 기본적으로 키-값 맵 (key-value map)
  - 키들은 사전식으로 정렬
- 데이터가 추가될 때 WAL(Write Ahead Log)라는 로그에 저장되고, 또한 메모리에도 저장(memstore)

- 플러시 (Flush)
  - 메모리의 데이터 크기가 최대 설정 값을 넘어서면 HFile로 내보내짐 (플러시)
  - 데이터가 HFile로 저장되면 WAL에서 제거
  - 영역 서버는 플러시 동안 읽기-쓰기, WAL/memstore 쓰기를 계속 진행

데이터 저장 (2)

- HDFS는 기존 파일의 갱신(update)을 지원하지 않아서 HFile은 불변(immutalble)
  - HFile의 키-값 들을 제거할 수 없음
  - 시간이 지남에 따라 점점 많은 HFiles 들이 생성

- 한 레코드가 삭제되었음을 표시하는 삭제 마커(delete marker)가 저장
  - 이 마커들은 데이터를 필터링하는데 사용
    - 제거된 레코드들을 숨김
  - 실행 시에 HFile과 WAL의 내용 사이 통합

데이터 저장 (3)

- HFile들의 수를 조정하고 클러스터 간의 부하 균형을 유지하기 위해 HBase는 주기적으로 데이터 압축 (data compaction)을 수행
  - 작은 압축 (minor compaction)
    - 작은 HFile들을 좀 더 큰 HFile들로 통합 (n-way merge)
    - 데이터들이 이미 파일 내에서 정렬되었기 때문에 빠르게 수행
    - 삭제 마커에는 적용되지 않음
  - 큰 압축 (major compaction)
    - 각 영역에서 한 컬럼-패밀리 내에 있는 모든 파일들을 하나의 파일로 통합
    - 모든 엔트리들을 스캔하고, 필요하면 삭제도 적용

HBase 마스터 (HBase Master)

- 영역(region)들과 이들의 위치(location)를 관리
  - 영역들을 영역 서버에 할당
  - 작업부하들을 수용하기 위해 재-균형화(re-balanced)
  - 영역 서버가 가용하지 않은 경우 복구
  - 분산 조정 서비스로 Zookeeper 사용
- 실제 데이터를 저장하거나 읽지는 않음
  - 클라이언트는 직접 영역 서버들과 통신
  - 일반적으로 부하는 적음
- 스키마 관리 및 변경에 책임
  - 테이블과 컬럼 패밀리의 추가 및 삭제

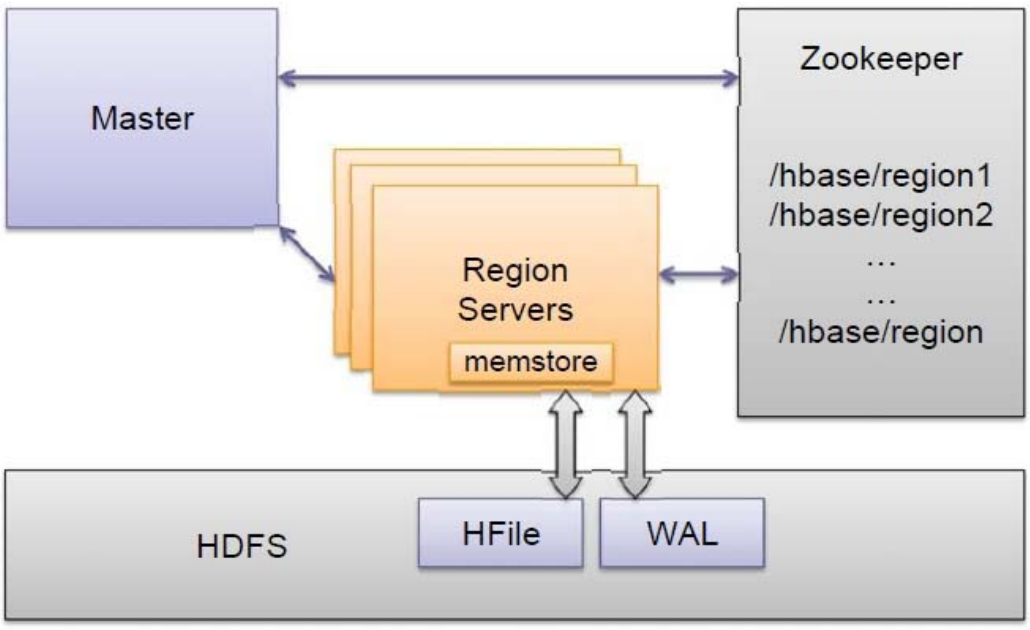
HBase와 Zookeeper

- HBase는 영역 할당을 위해 광범위하게 Zookeeper를 사용

"Zookeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services" - zookeeper.apache.org

- HBase가 Zookeeper 데몬을 관리할 수도 있고, 사용자가 분리하여 설치/관리 할 수도 있음
- http://zookeeper.apache.org 참조

HBase 컴포넌트



참조: WAL (Write-Ahead Log)

2. HBase 설치

HBase 설치

- 마스터 노드에서 설치한 후에 슬레이브 노드에 복사
  - 2개의 노드 클러스터 설치: master(192.168.0.200), slave1(192.168.0.201)
  - 설치 디렉토리: ~/hbase-1.2.6
- HBase 1.2.6 다운로드
  - hadoop – 2.7.3 호환 버전
  - http://www.apache.org/dyn/closer.cgi/hbase/
  - 미러 사이트 선택하여 다운로드
    - http://apache.mirror.cdnetworks.com/hbase/1.2.6/hbase-1.2.6-bin.tar.gz

```
$ wget http://apache.mirror.cdnetworks.com/hbase/1.2.6/hbase-1.2.6-bin.tar.gz
```

- 압축 해제

```
$ tar -xvzf hbase-1.2.6-bin.tar.gz
```

HBase 설정 – 환경 변수

- HBase 환경 변수 설정
  - ~/.bashrc 에 추가

```
# HBase Path
export HBASE_HOME=$HOME/hbase-1.2.6
export PATH=$PATH:$HBASE_HOME/bin
```

- ssh 를 통한 배포 및 수정

```
$ scp .bashrc slave1:~/

$ ssh slave1 'source .bashrc'
```


HBase 설정 – hbase-site.xml (1)

- ~/hbase-1.2.6/conf /hbase-site.xml

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
        <property>
                <name>hbase.rootdir</name>
                <value>hdfs://master:9000/hbase</value>
        </property>
        <property>
                <name>hbase.master</name>
                <value>master</value>
                <final>true</final>
        </property>
        <property>
                <name>hbase.zookeeper.quorum</name>
                <value>192.168.0.200,192.168.0.201</value>
        </property>
```

HBase 설정 – hbase-site.xml (2)

```
        <property>
                <name>hbase.cluster.distributed</name>
                <value>true</value>
        </property>
        <property>
                <name>hbase.zookeeper.property.dataDir</name>
                <value>/home/hadoop/hbase-1.2.6/zk_data</value>
        </property>
        <property>
                <name>hbase.zookeeper.property.clientPort</name>
                <value>2181</value>
        </property>
</configuration>
```


HBase 설정 – hbase-env.sh, regionservers

- ~/hbase-1.2.6/conf/hbase-env.sh

```
export JAVA_HOME=/usr/lib/jvm/java-8-oracle
export HBASE_PID_DIR=/home/hadoop/hbase-1.2.6
export HBASE_MANAGES_ZK=true
export HBASE_OPTS="-XX:+UseConcMarkSweepGC"
export HBASE_MASTER_OPTS="$HBASE_MASTER_OPTS"
export HBASE_REGIONSERVER_OPTS="$HBASE_REGIONSERVER_OPTS"
```

- ~/hbase-1.2.6/conf/regionservers

```
master
slave1
```

- ssh 를 통한 배포 및 수정

```
$ scp -r /home/hadoop/hbase-1.2.6 slave1:~/
```

HBase 구동

- 마스터 노드에서 실행 중인 스파크/하둡 중지 후 하둡/스파크/HBase 실행

```
$ ~/spark-2.1.0-bin-hadoop2.7/stop-master.sh
$ ~/hadopp-2.7.3/sbin/stop-all.sh

$ ~/hadopp-2.7.3/sbin/start-all.sh
$ ~/spark-2.1.0-bin-hadoop2.7/start-master.sh
$ ~/hbase-1.2.6/bin/start-hbase.sh
```


HBase 구동 확인 – jps

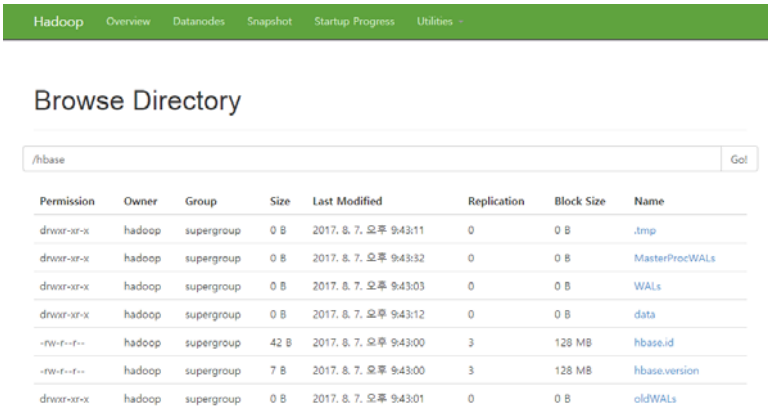
```
[redacted]@MASTER:~$ sudo jps
[sudo] password for [redacted]:
11953 Jps
819 ResourceManager
1559 HMaster
535 NameNode
1432 HQuorumPeer
2123 Master
1725 HRegionServer
3486 SparkSubmit
3390 ZeppelinServer
```

HBase 디렉토리

- /hbase 디렉토리

```
   _  _@MASTER:~$ hadoop fs -ls /hbase
Found 7 items
drwxr-xr-x   - hadoop supergroup          0 2017-08-08 12:44 /hbase/.tmp
drwxr-xr-x   - hadoop supergroup          0 2017-08-17 09:44 /hbase/MasterProcWALs
drwxr-xr-x   - hadoop supergroup          0 2017-08-08 12:44 /hbase/WALs
drwxr-xr-x   - hadoop supergroup          0 2017-08-08 12:44 /hbase/data
-rw-r--r--   3 hadoop supergroup         42 2017-08-08 12:44 /hbase/hbase.id
-rw-r--r--   3 hadoop supergroup          7 2017-08-08 12:44 /hbase/hbase.version
drwxr-xr-x   - hadoop supergroup          0 2017-08-17 09:55 /hbase/oldWALs
```

- 웹(http://192.168.0.200:50160)으로 표시
  - 웹 관리 UI는 60010 포트



3. HBase 셸

셸 시작

- JRuby IRB (Interactive Ruby Shell)
  - HBase 명령이 추가
  - IRB에 수행되는 것은 HBase 셸에서도 수행
    - http://en.wikipedia.org/wiki/Interactive_Ruby_Shell
- 셸 시작

  $ $HBASE_HOME/bin/hbase shell

```
@MASTER:~$ $HBASE_HOME/bin/hbase shell
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/hadoop/hbase-1.2.6/lib/slf4j-log4j12-1.7.
5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/hadoop/hadoop-2.7.3/share/hadoop/common/l
ib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 1.2.6, rUnknown, Mon May 29 02:25:32 CDT 2017

hbase(main):001:0>
```


셸 명령 – help

- 명령 리스트를 보기 위해선 help 명령 사용

  hbase> help “command” (quotes are required)

  -> help “get”

```
hbase(main):027:0> help "get"
Get row or cell contents; pass table name, row, and optionally
a dictionary of column(s), timestamp, timerange and versions. Examples:

  hbase> get 'ns1:t1', 'r1'
  hbase> get 't1', 'r1'
  hbase> get 't1', 'r1', {TIMERANGE => [ts1, ts2]}
  hbase> get 't1', 'r1', {COLUMN => 'c1'}
  hbase> get 't1', 'r1', {COLUMN => ['c1', 'c2', 'c3']}
  hbase> get 't1', 'r1', {COLUMN => 'c1', TIMESTAMP => ts1}
  hbase> get 't1', 'r1', {COLUMN => 'c1', TIMERANGE => [ts1, ts2], VERSIONS => 4}
  hbase> get 't1', 'r1', {COLUMN => 'c1', TIMESTAMP => ts1, VERSIONS => 4}
  hbase> get 't1', 'r1', {FILTER => "ValueFilter(=, 'binary:abc')"}
  hbase> get 't1', 'r1', 'c1'
  hbase> get 't1', 'r1', 'c1', 'c2'
  hbase> get 't1', 'r1', ['c1', 'c2']
  hbase> get 't1', 'r1', {COLUMN => 'c1', ATTRIBUTES => {'mykey'=>'myvalue'}}
  hbase> get 't1', 'r1', {COLUMN => 'c1', AUTHORIZATIONS => ['PRIVATE','SECRET']}
  hbase> get 't1', 'r1', {CONSISTENCY => 'TIMELINE'}
```

인용 부호 (quote) 사용

- 모든 이름에 인용부호(따옴표) 사용
  - 테이블, 열 이름
  - 텍스트는 단일 따옴표
    - hbase> get 't1', 'myRowId'
  - 이진값에는 이중 따옴표
    - 이진 값에는 16진수 표현 사용
    - hbase> get 't1', "key₩x03₩x3f₩xcd"
- 파라미터 지정에는 루비(ruby) 해시 사용
  - {'key1' => 'value1', 'key2' => 'value2', …}
  - 예
    hbase> get 'UserTable', 'userId1', {COLUMN => 'address:str'}

HBase 셸 명령

- HBase 셸은 다양한 명령 지원
  - 일반
    - status, version
  - 데이터 정의 언어 (Data Definition Language, DDL)
    - alter, create, describe, disable, drop, enable, exists, is_disabled, is_enabled, list
  - 데이터 관리 언어 (Data Manipulation Language, DML)
    - count, delete, deleteall, get, get_counter, incr, put, scan, truncate
  - 클러스터 관리 (cluster administration)
    - balancer, close_region, compact, flush, major_compact, move, split,unassign, zk_dump, add_peer, disable_peer, enable_peer,remove_peer, start_replication, stop_replication
- 각 명령 상세 내용
  - hbase> help "<command>"

상태 검사

- status 명령으로 클러스터의 상태를 디스플레이
  - hbase> status
  - hbase> status "detailed"

```
hbase(main):034:0> status
1 active master, 0 backup masters, 2 servers, 0 dead, 1.0000 average load

hbase(main):035:0> status "detailed"
version 1.2.6
0 regionsInTransition
active master:  MASTER:16000 1502163841979
0 backup masters
master coprocessors: []
2 live servers
    MASTER:16020 1502163842002
        requestsPerSecond=0.0, numberOfOnlineRegions=1, usedHeapMB=23, maxHeapMB=
1932, numberOfStores=1, numberOfStorefiles=1, storefileUncompressedSizeMB=0, stor
efileSizeMB=0, memstoreSizeMB=0, storefileIndexSizeMB=0, readRequestsCount=2567,
writeRequestsCount=2, rootIndexSizeKB=0, totalStaticIndexSizeKB=0, totalStaticBlo
omSizeKB=0, totalCompactingKVs=0, currentCompactedKVs=0, compactionProgressPct=Na
N, coprocessors=[MultiRowMutationEndpoint]
        "hbase:meta,,1"
            numberOfStores=1, numberOfStorefiles=1, storefileUncompressedSizeMB=0
```


데이터 정의 및 조작 명령 (DDL and DML)

- 예를 통하여 소개

1. 테이블 생성
   - 컬럼 패밀리 정의
2. 테이블에 데이터 레코드 입력
   - 다수 레코드들 입력
3. 데이터 접근
   - count, get, scan
4. 데이터 편집
5. 레코드 삭제
6. 테이블 삭제

1. 테이블 생성

- 다음 스키마의 Blog 테이블 생성
  - 2개 컬럼 패밀리
    - info 패밀리는 3개의 열: title, author, date
    - content 패밀리는 1개의 열: post

Blog		
Family:	info:	Columns: title, author, date
	content:	Columns: post

1. 테이블 생성 예

- 테이블과 컬럼 패밀리 생성에 여러 옵션 사용 가능
  - hbase> create 't1', {NAME => 'f1', VERSIONS => 5}
  - hbase> create 't1', {NAME => 'f1'}, {NAME => 'f2'}, {NAME => 'f3'}
  - hbase> create 't1', 'f1', 'f2', 'f3'

```
hbase(main):036:0> create 'Blog', {NAME=>'info'}, {NAME=>'content'}
0 row(s) in 4.6990 seconds

=> Hbase::Table - Blog
hbase(main):037:0>
```

2. 테이블에 데이터 레코드 입력

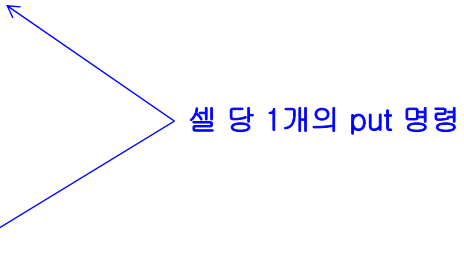
- 여러 개 레코드의 데이터 입력

Row Id	info:title	info:author	info:date	content:post
Matt-001	Elephant	Matt	2009.05.06	Do elephants like monkeys?
Matt-002	Monkey	Matt	2011.02.14	Do monkeys like elephants?
Bob-003	Dog	Bob	1995.10.20	People own dogs!
Michelle-004	Cat	Michelle	1990.07.06	I have a cat!
John-005	Mouse	John	2012.01.15	Mickey mouse.

- put 명령: 셀의 값 입력
  - hbase> put 'table', 'row_id', 'family:column', 'value'

2. 테이블에 데이터 레코드 입력 예 (1)

```
# insert row 1
put 'Blog', 'Matt-001', 'info:title', 'Elephant'
put 'Blog', 'Matt-001', 'info:author', 'Matt'
put 'Blog', 'Matt-001', 'info:date', '2009.05.06'
put 'Blog', 'Matt-001', 'content:post', 'Do elephants like monkeys?'
# insert rows 2-4
...
...
# row 5
put 'Blog', 'John-005', 'info:title', 'Mouse'
put 'Blog', 'John-005', 'info:author', 'John'
put 'Blog', 'John-005', 'info:date', '1990.07.06'
put 'Blog', 'John-005', 'content:post', 'Mickey mouse.'
```



2. 테이블에 데이터 레코드 입력 예 (2)

```
hbase(main):052:0> put 'Blog', 'Matt-001', 'info:title', 'Elephant'
0 row(s) in 0.2410 seconds

hbase(main):053:0> put 'Blog', 'Matt-001', 'info:author', 'Matt'
0 row(s) in 0.0100 seconds

hbase(main):054:0> put 'Blog', 'Matt-001', 'info:date', '2009.05.06'
0 row(s) in 0.0070 seconds

hbase(main):055:0> put 'Blog', 'Matt-001', 'content:post', 'Do elephants like
keys?'
0 row(s) in 0.0080 seconds

hbase(main):056:0> put 'Blog', 'Matt-002', 'info:title', 'Monkey'
0 row(s) in 0.0430 seconds
```

```
hbase(main):072:0> put 'Blog', 'Michelle-004', 'content:post', 'I have a cat!'
0 row(s) in 0.0200 seconds

hbase(main):073:0> put 'Blog', 'John-005', 'info:title', 'Mouse'
0 row(s) in 0.0080 seconds

hbase(main):074:0> put 'Blog', 'John-005', 'info:author', 'John'
0 row(s) in 0.0100 seconds

hbase(main):075:0> put 'Blog', 'John-005', 'info:date', '1990.07.06'
0 row(s) in 0.0110 seconds

hbase(main):076:0> put 'Blog', 'John-005', 'content:post', 'Mickey mouse.'
0 row(s) in 0.0090 seconds
```


3. 데이터 접근 (Data Access) – count

- 데이터 접근
  - count: 레코드의 전체 개수를 표시
  - get: 한 행을 읽음
  - scan: 범위로 지정된 여러 행들을 읽음
- count 명령
  - hbase> count 'table_name'
  - 전체 테이블을 스캔해서 큰 테이블인 경우 속도 느림
  - 매 n개 행들 간격으로 갯 수를 표시, 디폴트는 1000
    - hbase> count 't1', INTERVAL => 10

3. 데이터 접근 – count 예

```
hbase(main):079:0> count 'Blog'
5 row(s) in 0.0110 seconds

=> 5
hbase(main):080:0> count 'Blog', {INTERVAL=>2}
Current count: 2, row: John-005
Current count: 4, row: Matt-002
5 row(s) in 0.0140 seconds

=> 5
hbase(main):081:0> count 'Blog', {INTERVAL=>1}
Current count: 1, row: Bob-003
Current count: 2, row: John-005
Current count: 3, row: Matt-001
Current count: 4, row: Matt-002
Current count: 5, row: Michelle-004
5 row(s) in 0.0130 seconds

=> 5
hbase(main):082:0>
```


3. 데이터 접근 – get

- 단일 행 읽기
  - hbase> get 'table', 'row_id'
    - 전체 행을 리턴
  - 테이블 이름과 row id를 기술
  - 옵션: 타임스탬프, 타임-범위(time-range), 버전
- 특정 열 읽기
  - hbase> get 't1', 'r1', {COLUMN => 'c1'}
  - hbase> get 't1', 'r1', {COLUMN => ['c1', 'c2', 'c3']}
- 특정 타임스탬프, 타임-범위 읽기
  - hbase> get 't1', 'r1', {TIMERANGE => [ts1, ts2]}
  - hbase> get 't1', 'r1', {COLUMN => 'c1', TIMESTAMP => ts1}
- 다수 버전 읽기
  - hbase> get 't1', 'r1', {VERSIONS => 4}

3. 데이터 접근 – get 예

```
hbase(main):082:0> get 'Blog', 'unknownRowId'
COLUMN                CELL
0 row(s) in 0.0100 seconds

hbase(main):083:0> get 'Blog', 'Michelle-004'
COLUMN                CELL
 content:post         timestamp=1502935378057, value=I have a cat!
 info:author          timestamp=1502935300950, value=Michelle
 info:date            timestamp=1502935338079, value=1990.07.06
 info:title           timestamp=1502935279189, value=Cat
4 row(s) in 0.0090 seconds

hbase(main):084:0> get 'Blog', 'Michelle-004',{COLUMN=>['info:author','content:post']}
COLUMN                  CELL
 content:post           timestamp=1502935378057, value=I have a cat!
 info:author            timestamp=1502935300950, value=Michelle
2 row(s) in 0.0170 seconds

hbase(main):085:0> get 'Blog', 'Michelle-004',{COLUMN=>['info:author','content:post'], TIMEST
AMP=>1502935378057}
COLUMN                  CELL
 content:post           timestamp=1502935378057, value=I have a cat!
1 row(s) in 0.0090 seconds
```


3. 데이터 접근 – scan (1)

- 전체 테이블이나 부분을 스캔
- 전체 행 또는 명시적으로 지정된 컬럼 패밀리, 열, 특정 셀을 가져옴
- 전체 테이블 스캔
  - hbase> scan 'table_name'
- 결과 값의 개수 제한
  - hbase> scan 'table_name', {LIMIT=>1}
- 특정 범위 스캔
  - hbase> scan 'Blog', {STARTROW=>'startRow', STOPROW=>'stopRow'}
  - 시작 행은 포함되고, 끝 행은 포함되지 않음
  - 시작 행 또는 끝 행만 지정할 수도 있음

3. 데이터 접근 – scan (2)

- 특정 열로 제한
  - hbase> scan 'table', {COLUMNS=>['col1', 'col2']}
- 특정 시간 범위
  - hbase> scan 'table', {TIMERANGE => [1303, 13036]}
- 필터로 결과 값 제한
  - hbase> scan 'Blog', {FILTER => org.apache.hadoop.hbase.filter.ColumnPaginationFilter.new(1, 0)}
  - 필터는 나중에 소개

3. 데이터 접근 – scan 예

```
hbase(main):087:0> scan 'Blog'
ROW                   COLUMN+CELL
 Bob-003              column=content:post, timestamp=1502935083641, value=People own dogs!
 Bob-003              column=info:author, timestamp=1502934985811, value=Bob
 Bob-003              column=info:date, timestamp=1502935014355, value=1995.10.20
 Bob-003              column=info:title, timestamp=1502934956461, value=Dog
 John-005             column=content:post, timestamp=1502935422868, value=Mickey mouse.
 John-005             column=info:author, timestamp=1502935407197, value=John
 John-005             column=info:date, timestamp=1502935414776, value=1990.07.06
 John-005             column=info:title, timestamp=1502935389827, value=Mouse
 Matt-001             column=content:post, timestamp=1502934644453, value=Do elephants lik
                      e monkeys?
 Matt-001             column=info:author, timestamp=1502934626655, value=Matt
 Matt-001             column=info:date, timestamp=1502934635030, value=2009.05.06
 Matt-001             column=info:title, timestamp=1502934618553, value=Elephant
 Matt-002             column=content:post, timestamp=1502934906009, value=Do monkeys like
                      elephants?
 Matt-002             column=info:author, timestamp=1502934804588, value=Matt
 Matt-002             column=info:date, timestamp=1502934840726, value=2011.02.14
 Matt-002             column=info:title, timestamp=1502934742790, value=Monkey
 Michelle-004         column=content:post, timestamp=1502935378057, value=I have a cat!
 Michelle-004         column=info:author, timestamp=1502935300950, value=Michelle
 Michelle-004         column=info:date, timestamp=1502935338079, value=1990.07.06
 Michelle-004         column=info:title, timestamp=1502935279189, value=Cat
5 row(s) in 0.0370 seconds

hbase(main):088:0> scan 'Blog', {STOPROW=>'John'}
ROW                   COLUMN+CELL
 Bob-003              column=content:post, timestamp=1502935083641, value=People own dogs!
 Bob-003              column=info:author, timestamp=1502934985811, value=Bob
 Bob-003              column=info:date, timestamp=1502935014355, value=1995.10.20
 Bob-003              column=info:title, timestamp=1502934956461, value=Dog
1 row(s) in 0.0180 seconds

hbase(main):089:0> scan 'Blog', {COLUMNS=>'info:title',STARTROW=>'John', STOPROW=>'Michelle'}
ROW                   COLUMN+CELL
 John-005             column=info:title, timestamp=1502935389827, value=Mouse
 Matt-001             column=info:title, timestamp=1502934618553, value=Elephant
 Matt-002             column=info:title, timestamp=1502934742790, value=Monkey
3 row(s) in 0.0140 seconds
```

4. 데이터 편집

- put 명령으로 row id가 존재하지 않는 경우 새로운 값을 삽입
- 행이 이미 존재하면 값을 갱신
  - 여러 개의 값은 버전(VERSION)으로 관리

- 실제로 갱신되는 가?
  - 셀에 새로운 버전을 삽입
  - 디폴트로 가장 최근의 버전만 선택
  - 셀 당 N 버전이 보관
    - 테이블 생성 시에 컬럼 패밀리 당 버전 개수 설정
      hbase> create ' table ' , {NAME => ' family ' , VERSIONS => 3}
    - 이미 존재하는 테이블에 버전 수정
      hbase> alter 'table', {NAME => 'family', VERSIONS => 3}

4. 데이터 편집 예 (1)

```
hbase(main):120:0> describe 'Blog'
Table Blog is ENABLED
Blog
COLUMN FAMILIES DESCRIPTION
{NAME => 'content', BLOOMFILTER => 'ROW', VERSIONS => '1', IN_MEMORY => 'false', KEEP_DELETED_
CELLS => 'FALSE', DATA_BLOCK_ENCODING => 'NONE', TTL => 'FOREVER', COMPRESSION => 'NONE', MIN_
VERSIONS => '0', BLOCKCACHE => 'true', BLOCKSIZE => '65536', REPLICATION_SCOPE => '0'}
{NAME => 'info', BLOOMFILTER => 'ROW', VERSIONS => '1', IN_MEMORY => 'false', KEEP_DELETED_CEL
LS => 'FALSE', DATA_BLOCK_ENCODING => 'NONE', TTL => 'FOREVER', COMPRESSION => 'NONE', MIN_VER
SIONS => '0', BLOCKCACHE => 'true', BLOCKSIZE => '65536', REPLICATION_SCOPE => '0'}
2 row(s) in 0.0130 seconds

hbase(main):121:0> alter 'Blog', {NAME=>'info', VERSIONS=>3}
Updating all regions with the new schema...
1/1 regions updated.
Done.
0 row(s) in 2.8480 seconds

hbase(main):122:0> describe 'Blog'
Table Blog is ENABLED
Blog
COLUMN FAMILIES DESCRIPTION
{NAME => 'content', BLOOMFILTER => 'ROW', VERSIONS => '1', IN_MEMORY => 'false', KEEP_DELETED_
CELLS => 'FALSE', DATA_BLOCK_ENCODING => 'NONE', TTL => 'FOREVER', COMPRESSION => 'NONE', MIN_
VERSIONS => '0', BLOCKCACHE => 'true', BLOCKSIZE => '65536', REPLICATION_SCOPE => '0'}
{NAME => 'info', BLOOMFILTER => 'ROW', VERSIONS => '3', IN_MEMORY => 'false', KEEP_DELETED_CEL
LS => 'FALSE', DATA_BLOCK_ENCODING => 'NONE', TTL => 'FOREVER', COMPRESSION => 'NONE', MIN_VER
SIONS => '0', BLOCKCACHE => 'true', BLOCKSIZE => '65536', REPLICATION_SCOPE => '0'}
2 row(s) in 0.0160 seconds

hbase(main):123:0>
```

4. 데이터 편집 예 (2)

```
hbase(main):124:0> put 'Blog', 'Michelle-004', 'info:date', '1990.07.06'
0 row(s) in 0.0110 seconds

hbase(main):125:0> put 'Blog', 'Michelle-004', 'info:date', '1990.07.07'
0 row(s) in 0.0090 seconds

hbase(main):126:0> put 'Blog', 'Michelle-004', 'info:date', '1990.07.08'
0 row(s) in 0.0140 seconds

hbase(main):127:0> get 'Blog', 'Michelle-004', {COLUMN=>'info:date', VERSIONS=>3}
COLUMN                CELL
 info:date            timestamp=1502938152365, value=1990.07.08
 info:date            timestamp=1502938144747, value=1990.07.07
 info:date            timestamp=1502938136109, value=1990.07.06
3 row(s) in 0.0080 seconds

hbase(main):128:0> get 'Blog', 'Michelle-004',{COLUMN=>'info:date', VERSIONS=>2}
COLUMN                CELL
 info:date            timestamp=1502938152365, value=1990.07.08
 info:date            timestamp=1502938144747, value=1990.07.07
2 row(s) in 0.0270 seconds

hbase(main):129:0> get 'Blog', 'Michelle-004', {COLUMN=>'info:date'}
COLUMN                CELL
 info:date            timestamp=1502938152365, value=1990.07.08
1 row(s) in 0.0120 seconds
```


5. 레코드 삭제

- 지정된 테이블의 행과 열을 삭제
  - hbase> delete 'table', 'rowId', 'column'
  - 해당 셀의 모든 버전 삭제
- 타임스탬프를 지정하면 지정된 타임스탬프 이전의 버전들만 삭제
  - hbase> delete 'table', 'rowId', 'column', timestamp

5. 레코드 삭제 예

```
hbase(main):162:0> get 'Blog', 'Bob-003', 'info:date'
COLUMN                    CELL
 info:date                timestamp=1502938884384, value=1995.10.20
1 row(s) in 0.0100 seconds

hbase(main):163:0> delete 'Blog', 'Bob-003', 'info:date'
0 row(s) in 0.0100 seconds

hbase(main):164:0> get 'Blog', 'Bob-003', 'info:date'
COLUMN                    CELL
0 row(s) in 0.0090 seconds
```

```
hbase(main):167:0> get 'Blog', 'Michelle-004', {COLUMN=>'info:date', VERSIONS=>3}
COLUMN                    CELL
 info:date                timestamp=1502939047363, value=1990.07.08
 info:date                timestamp=1502939039923, value=1990.07.07
 info:date                timestamp=1502939031723, value=1990.07.06
3 row(s) in 0.0110 seconds

hbase(main):168:0> delete 'Blog', 'Michelle-004', 'info:date', 1502939039930
0 row(s) in 0.0110 seconds

hbase(main):169:0> get 'Blog', 'Michelle-004', {COLUMN=>'info:date', VERSIONS=>3}
COLUMN                    CELL
 info:date                timestamp=1502939047363, value=1990.07.08
1 row(s) in 0.0070 seconds
```


6. 테이블 삭제

- 테이블 삭제 전에 테이블 오프라인 설정(disable)
  - 테이블이 오프라인으로 설정되어야만 스키마 기반 연산이 수행
  - hbase> disable 'table_name'
  - hbase> drop 'table_name'
  - 예:

    hbase> disable 'Blog'

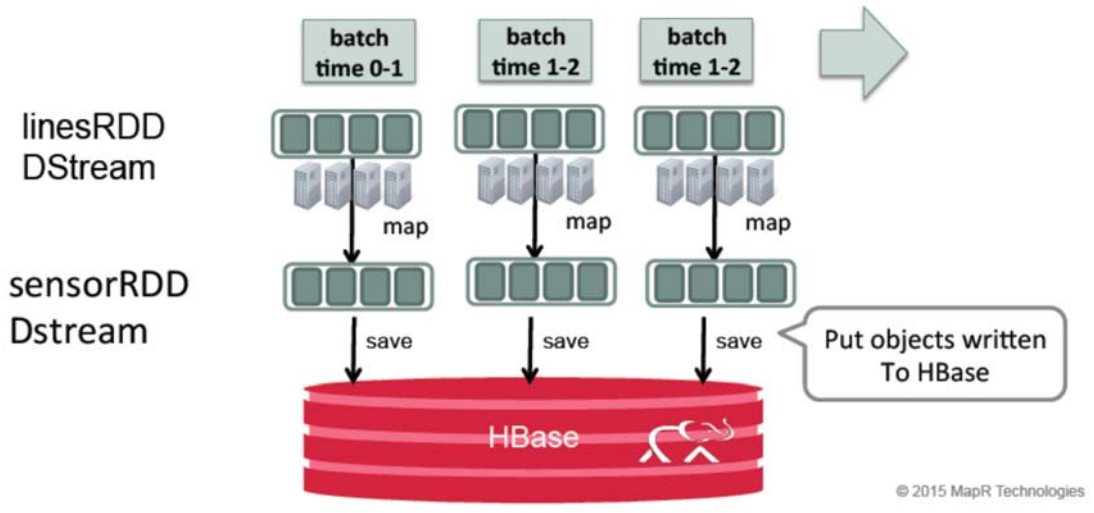
    hbase> drop 'Blog'

- 큰 테이블의 경우 상당한 시간이 걸림

4. 스파크 스트리밍의 HBase 저장

스트리밍 결과 HBase 저장 예

- 석유 시추 시설 모니터링 예에서의 센서 스트리밍의 데이터 프레임을 HBase 테이블에 저장



HBase 테이블 스키마

- 테이블 이름: oil_sensor
- 컬럼 패밀리
  - data: 모든 이벤트
  - alerts: 낮은 압력의 알림 경고
  - stats: 일자별 통계
- 행의 키
  - 오일펌프 이름, 일자, 시간

Row key	CF data			CF alerts		CF stats		
	hz	...	psi	psi	...	hz_avg	...	psi_min
COHUTTA_3/10/14_1:01	10.37		84	0				
COHUTTA_3/10/14						10		0

HBase 출력 설정

- 맵리듀스의 TableOutputFormat 클래스 사용하여 HBase 테이블 출력 작업 설정
  - org.apache.hadoop.hbase.mapred.TableOutputFormat

```
final val tableName = "oil_sensor"  // HBase Table

// set up HBase Table configuration
val conf = HBaseConfiguration.create()
conf.set(TableOutputFormat.OUTPUT_TABLE, tableName)

val jobConfig: JobConf = new JobConf(conf, this.getClass)
jobConfig.set("mapreduce.output.fileoutputformat.outputdir", "/sparkdata/stream/out")
jobConfig.setOutputFormat(classOf[TableOutputFormat])
jobConfig.set(TableOutputFormat.OUTPUT_TABLE, tableName)
```

HBase 테이블 저장

- 센서 값을 HBase의 Put 객체로 변환 후 saveAsHadoop dataset() 메서드를 사용하여 HBase 테이블에 저장

```
//  Convert a row of sensor object data to an HBase put object
def convertToPut(sensor: Sensor): (ImmutableBytesWritable, Put) = {
  val dateTime = sensor.date + " " + sensor.time
  // create a composite row key: sensorid_date time
  val rowkey = sensor.resid + "_" + dateTime
  val put = new Put(Bytes.toBytes(rowkey))
  // add to column family data, column  data values to put object
  put.add(cfDataBytes, colHzBytes, Bytes.toBytes(sensor.hz))
  put.add(cfDataBytes, colDispBytes, Bytes.toBytes(sensor.disp))
  put.add(cfDataBytes, colFloBytes, Bytes.toBytes(sensor.flo))
  put.add(cfDataBytes, colSedBytes, Bytes.toBytes(sensor.sedPPM))
  put.add(cfDataBytes, colPsiBytes, Bytes.toBytes(sensor.psi))
  put.add(cfDataBytes, colChlBytes, Bytes.toBytes(sensor.chlPPM))
  return (new ImmutableBytesWritable(Bytes.toBytes(rowkey)), put)
}
.....
rdd.map(HBaseSensorStream.convertToPut).saveAsHadoopDataset(jobConfig)
```


스트림 데이터의 HBase 테이블 저장 예 – SBT 프로젝트 스크립트

- SBT 프로젝트 스크립트 작성
  - 소스 프로그램: src/main/scala/HBaseSensorStream.scala
  - 프로젝트 스크립트, hbasesensor.sbt
  - 라이브러리에 hadoop, hbase 의존성 추가

```
name := "SensorStream Project"
version := "1.0"
scalaVersion := "2.11.11"
libraryDependencies += "org.apache.spark" % "spark-core_2.10" % "2.1.1"
libraryDependencies += "org.apache.spark" % "spark-streaming_2.10" % "2.1.1"

libraryDependencies += "org.apache.hadoop" % "hadoop-common" % "2.7.3"
libraryDependencies += "org.apache.hbase" % "hbase" % "1.1.2"
libraryDependencies += "org.apache.hbase" % "hbase-server" % "1.1.2"
libraryDependencies += "org.apache.hbase" % "hbase-client" % "1.1.2"
libraryDependencies += "org.apache.hbase" % "hbase-common" % "1.1.2"
```

HBaseSensorStream.scala

```
import org.apache.hadoop.hbase.HBaseConfiguration
import org.apache.hadoop.hbase.client.Put
import org.apache.hadoop.hbase.io.ImmutableBytesWritable
import org.apache.hadoop.hbase.mapred.TableOutputFormat
import org.apache.hadoop.hbase.util.Bytes
import org.apache.hadoop.mapred.JobConf

import org.apache.spark.SparkConf
import org.apache.spark.SparkContext
import org.apache.spark.streaming.Seconds
import org.apache.spark.streaming.StreamingContext

object HBaseSensorStream extends Serializable {
  // HBase Table
  final val tableName = "oil_sensor"
  // Column Family
  final val cfDataBytes = Bytes.toBytes("data")
  final val cfAlertBytes = Bytes.toBytes("alerts")
  // Columns
  final val colHzBytes = Bytes.toBytes("hz")
  final val colDispBytes = Bytes.toBytes("disp")
  final val colFloBytes = Bytes.toBytes("flo")
  final val colSedBytes = Bytes.toBytes("sedPPM")
```

```
  final val colPsiBytes = Bytes.toBytes("psi")
  final val colChlBytes = Bytes.toBytes("chlPPM")

  // schema for sensor data
  case class Sensor(resid: String, date: String, time: String, hz: Double, disp: Double, flo: Double,
                    sedPPM: Double, psi: Double, chlPPM: Double) extends Serializable

  object Sensor extends Serializable {
    // function to parse line of sensor data into Sensor class
    def parseSensor(str: String): Sensor = {
      val p = str.split(",")
      Sensor(p(0), p(1), p(2), p(3).toDouble, p(4).toDouble, p(5).toDouble, p(6).toDouble,
                                                          p(7).toDouble, p(8).toDouble)
    }
    //  Convert a row of sensor object data to an HBase put object
    def convertToPut(sensor: Sensor): (ImmutableBytesWritable, Put) = {
      val dateTime = sensor.date + " " + sensor.time
      // create a composite row key: sensorid_date time
      val rowkey = sensor.resid + "_" + dateTime
      val put = new Put(Bytes.toBytes(rowkey))
      // add to column family data, column  data values to put object
      put.add(cfDataBytes, colHzBytes, Bytes.toBytes(sensor.hz))
      put.add(cfDataBytes, colDispBytes, Bytes.toBytes(sensor.disp))
      put.add(cfDataBytes, colFloBytes, Bytes.toBytes(sensor.flo))
      put.add(cfDataBytes, colSedBytes, Bytes.toBytes(sensor.sedPPM))
      put.add(cfDataBytes, colPsiBytes, Bytes.toBytes(sensor.psi))
```

```
    put.add(cfDataBytes, colChlBytes, Bytes.toBytes(sensor.chlPPM))
    return (new ImmutableBytesWritable(Bytes.toBytes(rowkey)), put)
  }
  // convert psi alert to an HBase put object
  def convertToPutAlert(sensor: Sensor): (ImmutableBytesWritable, Put) = {
    val dateTime = sensor.date + " " + sensor.time
    // create a composite row key: sensorid_date time
    val key = sensor.resid + "_" + dateTime
    val p = new Put(Bytes.toBytes(key))
    // add to column family alert, column psi data value to put object
    p.add(cfAlertBytes, colPsiBytes, Bytes.toBytes(sensor.psi))
    return (new ImmutableBytesWritable(Bytes.toBytes(key)), p)
  }
 }

 def main(args: Array[String]): Unit = {
  // set up HBase Table configuration
  val conf = HBaseConfiguration.create()
  conf.set(TableOutputFormat.OUTPUT_TABLE, tableName)
  val jobConfig: JobConf = new JobConf(conf, this.getClass)
  jobConfig.set("mapreduce.output.fileoutputformat.outputdir", "/sparkdata/stream/out")
  jobConfig.setOutputFormat(classOf[TableOutputFormat])
  jobConfig.set(TableOutputFormat.OUTPUT_TABLE, tableName)
  println("set configuration")
  val sparkConf = new SparkConf().setAppName("HBaseSensorStream").set("spark.files.overwrite",
"true")
```

```
  val sc = new SparkContext(sparkConf)

  // create a StreamingContext, the main entry point for all streaming functionality
  val ssc = new StreamingContext(sc, Seconds(2))

  // parse the lines of data into sensor objects  maprfs:///
  val sensorDStream = ssc.textFileStream("/sparkdata/stream/filestream").map(Sensor.parseSensor)
  sensorDStream.print()

  sensorDStream.foreachRDD { rdd =>
    // filter sensor data for low psi
    val alertRDD = rdd.filter(sensor => sensor.psi < 5.0)
    alertRDD.take(1).foreach(println)
    // convert sensor data to put object and write to HBase table column family data
    rdd.map(Sensor.convertToPut).saveAsHadoopDataset(jobConfig)
    // convert alert data to put object and write to HBase table column family alert
    alertRDD.map(Sensor.convertToPutAlert).saveAsHadoopDataset(jobConfig)
  }
  // Start the computation
  ssc.start()
  println("start streaming")
  // Wait for the computation to terminate
  ssc.awaitTermination()
 }
}
```

HBaseSensorStream 응용의 빌드

- 응용 빌드

  $ sbt package

  - target/scala-2.11/hbasesensorstream-project_2.11-1.0.jar 파일 생성

```
---  :@MASTER:~/spark/stream/hbasesensor$ sbt package
[warn] Executing in batch mode.
[warn]   For better performance, hit [ENTER] to switch to interactive mode, or
[warn]   consider launching sbt without any commands, or explicitly passing 'shell'
[info] Loading project definition from /home/c  _-  'spark/stream/hbasesensor/project
[info] Set current project to HBaseSensorStream Project (in build file:/home/     :/spar
k/stream/hbasesensor/)
[info] Compiling 1 Scala source to /home/c  `  'spark/stream/hbasesensor/target/scala-2.
11/classes...
[warn] there were 7 deprecation warnings; re-run with -deprecation for details
[warn] one warning found
[info] Packaging /home/c  _   'spark/stream/hbasesensor/target/scala-2.11/hbasesensorstre
am-project_2.11-1.0.jar ...
[info] Done packaging.
[success] Total time: 5 s, completed Aug 17, 2017 3:46:05 PM
```


HBase 테이블 생성

- HBase 셸에서 HBase 테이블 생성

  hbase> create 'oil_sensor', 'data', 'alerts', 'stats'

```
hbase(main):014:0> create 'oil_sensor', 'data', 'alerts',  'stats'
0 row(s) in 2.3010 seconds

=> Hbase::Table - oil_sensor
hbase(main):015:0> describe 'oil_sensor'
Table oil_sensor is ENABLED
oil_sensor
COLUMN FAMILIES DESCRIPTION
{NAME => 'alerts', BLOOMFILTER => 'ROW', VERSIONS => '1', IN_MEMORY => 'false', KEEP_DELETED_CEL
LS => 'FALSE', DATA_BLOCK_ENCODING => 'NONE', TTL => 'FOREVER', COMPRESSION => 'NONE', MIN_VERSI
ONS => '0', BLOCKCACHE => 'true', BLOCKSIZE => '65536', REPLICATION_SCOPE => '0'}
{NAME => 'data', BLOOMFILTER => 'ROW', VERSIONS => '1', IN_MEMORY => 'false', KEEP_DELETED_CELLS
 => 'FALSE', DATA_BLOCK_ENCODING => 'NONE', TTL => 'FOREVER', COMPRESSION => 'NONE', MIN_VERSION
S => '0', BLOCKCACHE => 'true', BLOCKSIZE => '65536', REPLICATION_SCOPE => '0'}
{NAME => 'stats', BLOOMFILTER => 'ROW', VERSIONS => '1', IN_MEMORY => 'false', KEEP_DELETED_CELL
S => 'FALSE', DATA_BLOCK_ENCODING => 'NONE', TTL => 'FOREVER', COMPRESSION => 'NONE', MIN_VERSIO
NS => '0', BLOCKCACHE => 'true', BLOCKSIZE => '65536', REPLICATION_SCOPE => '0'}
3 row(s) in 0.0140 seconds
```

HBaseSensorStream 응용 실행

- 스파크의 드라이버 및 실행자 노드에 HBase 라이브러리 클래스들의 위치 지정
  - $SPARK_HOME/conf/spark-defaults.conf 에 다음 추가

    spark.driver.extraClassPath /home/hadoop/hbase-1.2.6/lib/*

    spark.executor.extraClassPath /home/hadoop/hbase-1.2.6/lib/*

- spark-submit 명령을 사용하여 실행
  - 실행 전에 sensordata.csv 제거하고, 실행 후에 다시 복사

```
$SPARK_HOME/bin/spark-submit --class HBaseSensorStream
  --master yarn target/scala-2.11/hbasesensorstream-project
  _2.11-1.0.jar
```



```
@MASTER:~/spark/stream/hbasesensor$ $SPARK_HOME/bin/spark-submit --class HBaseSensorStream
 --master yarn  target/scala-2.11/hbasesensorstream-project_2.11-1.0.jar
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/hadoop/hbase-1.2.6/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/
impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/hadoop/spark-2.1.0-bin-hadoop2.7/jars/slf4j-log4j12-1.7.16
.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
set configuration
start streaming
-------------------------------------------
Time: 1503207688000 ms
-------------------------------------------

-------------------------------------------
Time: 1503207698000 ms
-------------------------------------------

-------------------------------------------
Time: 1503207700000 ms
-------------------------------------------
Sensor(COHUTTA,3/10/14,1:01,10.27,1.73,881.0,1.56,85.0,1.94)
Sensor(COHUTTA,3/10/14,1:02,9.67,1.731,882.0,0.52,87.0,1.79)
Sensor(COHUTTA,3/10/14,1:03,10.47,1.732,882.0,1.7,92.0,0.66)
Sensor(COHUTTA,3/10/14,1:05,9.56,1.734,883.0,1.35,99.0,0.68)
Sensor(COHUTTA,3/10/14,1:06,9.74,1.736,884.0,1.27,92.0,0.73)
Sensor(COHUTTA,3/10/14,1:08,10.44,1.737,885.0,1.34,93.0,1.54)
Sensor(COHUTTA,3/10/14,1:09,9.83,1.738,885.0,0.06,76.0,1.44)
Sensor(COHUTTA,3/10/14,1:11,10.49,1.739,886.0,1.51,81.0,1.83)
Sensor(COHUTTA,3/10/14,1:12,9.79,1.739,886.0,1.74,82.0,1.91)
Sensor(COHUTTA,3/10/14,1:13,10.02,1.739,886.0,1.24,86.0,1.79)
...
Sensor(NANTAHALLA,3/13/14,2:05,0.0,0.0,0.0,1.73,0.0,1.51)
-------------------------------------------
Time: 1503207702000 ms
-------------------------------------------

@MASTER:~$ hadoop fs -ls /sparkdata/stream/out
Found 1 items
-rw-r--r--   3          supergroup          0 2017-08-20 14:42 /sparkdata/stream/out/_SUCCESS
@MASTER:~$
```

spark-submit 실행 에러 - Netty 버전 에러

- spark-submit 실행 시 다음과 같은 에러의 경우
  - Error sending result StreamResponse, io.netty.handler.codec.EncoderException: java.lang.NoSuchMethodError:
  - Netty 라이브러리 중복 에러
    - $SPARK_HOME/jars: netty-3.8.0.Final.jar, netty-all-4.0.42.Final.jar
    - $HBASE_HOME/lib: netty-all-4.0.23.Final.jar
  - 해결 방안
    - $HBASE_HOME/lib 에서 netty-all-4.0.23.Final.jar 제거하고 $SPARK_HOME/jars의 netty-all-4.0.42.Final.jar 최신 버전으로 복사
- Netty
  - *asynchronous event-driven network application framework* for rapid development of maintainable high performance protocol servers & clients.
  - https://netty.io/

HBase 테이블 확인

- 스파크 셸에서 HBase 테이블 확인

  hbase> scan 'oil_sensor', {COLUMNS=>['data'], LIMIT=> 2}

  hbase> scan 'oil_sensor', {COLUMNS=>['alerts'], LIMIT=> 2}

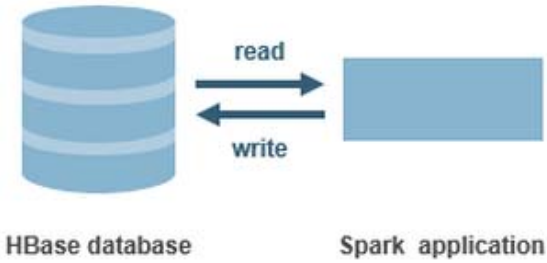
```
hbase(main):007:0> scan 'oil_sensor', {COLUMNS=>['data'], LIMIT=> 2}
ROW                   COLUMN+CELL
 ANDOUILLE_3/10/14 10:01   column=data:chlPPM, timestamp=1503207703047, value=?\xF7\x0A=p\xA3\xD7\x0A
 ANDOUILLE_3/10/14 10:01   column=data:disp, timestamp=1503207703047, value=?\xFB|\xED\x91hr\xB0
 ANDOUILLE_3/10/14 10:01   column=data:flo, timestamp=1503207703047, value=@\x93\xEC\x00\x00\x00\x00\x00
                           00
 ANDOUILLE_3/10/14 10:01   column=data:hz, timestamp=1503207703047, value=@#\xA3\xD7\x0A=p\xA4
 ANDOUILLE_3/10/14 10:01   column=data:psi, timestamp=1503207703047, value=@S\x00\x00\x00\x00\x00\x00
 ANDOUILLE_3/10/14 10:01   column=data:sedPPM, timestamp=1503207703047, value=?\xD3333333
 ANDOUILLE_3/10/14 10:02   column=data:chlPPM, timestamp=1503207703047, value=?\xECz\xE1G\xAE\x14{
 ANDOUILLE_3/10/14 10:02   column=data:disp, timestamp=1503207703047, value=?\xFBt\xBCj~\xF9\xDB
 ANDOUILLE_3/10/14 10:02   column=data:flo, timestamp=1503207703047, value=@\x93\xE4\x00\x00\x00\x00\x00
                           00
 ANDOUILLE_3/10/14 10:02   column=data:hz, timestamp=1503207703047, value=@#\xC2\x8F\x5C(\xF5\xC3
 ANDOUILLE_3/10/14 10:02   column=data:psi, timestamp=1503207703047, value=@T\x00\x00\x00\x00\x00\x00
 ANDOUILLE_3/10/14 10:02   column=data:sedPPM, timestamp=1503207703047, value=?\xB9\x99\x99\x99\x99\x9
                           9\x9A
2 row(s) in 0.0200 seconds

hbase(main):008:0> scan 'oil_sensor', {COLUMNS=>['alerts'], LIMIT=> 2}
ROW                   COLUMN+CELL
 LAGNAPPE_3/14/14 19:39    column=alerts:psi, timestamp=1503207703666, value=\x00\x00\x00\x00\x00\x00\
                           x00\x00
 LAGNAPPE_3/14/14 19:41    column=alerts:psi, timestamp=1503207703666, value=\x00\x00\x00\x00\x00\x00\
                           x00\x00
2 row(s) in 0.0100 seconds
```

5. 스파크 HBase 입출력

HBase 입출력 예

- 앞의 HBase 테이블에 저장된 오일 센서 데이터를 읽고, 일별 통계를 계산한 후 stats 컬럼 패밀리에 저장
  - 센서 데이터를 RDD로 읽어서 데이터프레임 변환
  - 테이블 스키마 출력, 데이터 5개 출력, 일별 오일 압력(psi) 평균 출력
  - 스파크 SQL 사용하여 일별 통계 계산
  - 일별 통계 데이터프레임 생성, 출력
  - 데이터프레임을 RDD로 변환 후 HBase에 저장



HBase 입력

- 맵리듀스의 TableInputFormat 클래스 사용하여 HBase 테이블 입력 (scan) 작업 설정
  - org.apache.hadoop.hbase.mapreduce.TableInputFormat

```
val conf = HBaseConfiguration.create()
conf.set(TableInputFormat.INPUT_TABLE, tableName)
// scan data column family
conf.set(TableInputFormat.SCAN_COLUMNS, "data")
 // Load an RDD of result(ImmutableBytesWritable, Result) tuples from the table
val hBaseRDD = sc.newAPIHadoopRDD(conf, classOf[TableInputFormat],
                        classOf[org.apache.hadoop.hbase.io.ImmutableBytesWritable],
                        classOf[org.apache.hadoop.hbase.client.Result])
 hBaseRDD.count()

// transform (ImmutableBytesWritable, Result) tuples into an RDD of Results
val resultRDD = hBaseRDD.map(tuple => tuple._2)
resultRDD.count()

// transform RDD of Results into an RDD of SensorRow objects
val sensorRDD = resultRDD.map(SensorRow.parseSensorRow)
```


HBase 입력 데이터의 RDD 변환

- 케이스 클래스 사용하여 RDD 객체 변환

```
case class SensorRow(rowkey: String, hz: Double, disp: Double, flo: Double,
                                        sedPPM: Double, psi: Double, chlPPM: Double)

 object SensorRow extends Serializable{
   def parseSensorRow(result: Result): SensorRow = {
     val rowkey = Bytes.toString(result.getRow())
     // remove time from rowKey, stats row key is for day
     val p0 = rowkey.split(" ")(0)

     val p1 = Bytes.toDouble(result.getValue(cfDataBytes, Bytes.toBytes("hz")))
     val p2 = Bytes.toDouble(result.getValue(cfDataBytes, Bytes.toBytes("disp")))
     val p3 = Bytes.toDouble(result.getValue(cfDataBytes, Bytes.toBytes("flo")))
     val p4 = Bytes.toDouble(result.getValue(cfDataBytes, Bytes.toBytes("sedPPM")))
     val p5 = Bytes.toDouble(result.getValue(cfDataBytes, Bytes.toBytes("psi")))
     val p6 = Bytes.toDouble(result.getValue(cfDataBytes, Bytes.toBytes("chlPPM")))

     SensorRow(p0, p1, p2, p3, p4, p5, p6)
   }
 }
```

HBase 입출력 예 예 - SBT 프로젝트 스크립트

- SBT 프로젝트 스크립트 작성
  - 소스 프로그램: src/main/scala/HBaseSensorStream.scala
  - 프로젝트 스크립트, hbaserwstats.sbt

```
name := " HBaseRWStats Project"
version := "1.0"
scalaVersion := "2.11.11"
libraryDependencies += "org.apache.spark" % "spark-core_2.10" % "2.1.1"
libraryDependencies += "org.apache.spark" % "spark-sql_2.10" % "2.1.0"

libraryDependencies += "org.apache.hadoop" % "hadoop-common" % "2.7.3"
libraryDependencies += "org.apache.hbase" % "hbase" % "1.1.2"
libraryDependencies += "org.apache.hbase" % "hbase-server" % "1.1.2"
libraryDependencies += "org.apache.hbase" % "hbase-client" % "1.1.2"
libraryDependencies += "org.apache.hbase" % "hbase-common" % "1.1.2"
```


HBaseRWStats.scala

```
import org.apache.spark._
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.functions.avg
import org.apache.spark.sql.Row

import org.apache.hadoop.hbase.HBaseConfiguration
import org.apache.hadoop.hbase.client.Put
import org.apache.hadoop.hbase.client.Result
import org.apache.hadoop.hbase.io.ImmutableBytesWritable
import org.apache.hadoop.hbase.mapred.TableOutputFormat
import org.apache.hadoop.hbase.mapreduce.TableInputFormat
import org.apache.hadoop.hbase.util.Bytes
import org.apache.hadoop.mapred.JobConf

object HBaseRWStats {
  // HBase Table
  final val tableName = "oil_sensor"
  // Column Family
  final val cfDataBytes = Bytes.toBytes("data")
  final val cfStatsBytes = Bytes.toBytes("stats")
```

```
case class SensorRow(rowkey: String, hz: Double, disp: Double, flo: Double, sedPPM: Double,
                                                                  psi: Double, chlPPM: Double)

object SensorRow extends Serializable{
  def parseSensorRow(result: Result): SensorRow = {
    val rowkey = Bytes.toString(result.getRow())
    // remove time from rowKey, stats row key is for day
    val p0 = rowkey.split(" ")(0)
    val p1 = Bytes.toDouble(result.getValue(cfDataBytes, Bytes.toBytes("hz")))
    val p2 = Bytes.toDouble(result.getValue(cfDataBytes, Bytes.toBytes("disp")))
    val p3 = Bytes.toDouble(result.getValue(cfDataBytes, Bytes.toBytes("flo")))
    val p4 = Bytes.toDouble(result.getValue(cfDataBytes, Bytes.toBytes("sedPPM")))
    val p5 = Bytes.toDouble(result.getValue(cfDataBytes, Bytes.toBytes("psi")))
    val p6 = Bytes.toDouble(result.getValue(cfDataBytes, Bytes.toBytes("chlPPM")))
    SensorRow(p0, p1, p2, p3, p4, p5, p6)
  }
}

case class SensorStatsRow(rowkey: String, maxhz: Double, minhz: Double, avghz: Double,
  maxdisp: Double, mindisp: Double, avgdisp: Double, maxflo: Double, minflo: Double,
  avgflo: Double,  maxsedPPM: Double, minsedPPM: Double, avgsedPPM: Double,
  maxpsi: Double, minpsi: Double, avgpsi: Double, maxchlPPM: Double, minchlPPM: Double,
  avgchlPPM: Double)

object SensorStatsRow {
    def convertToPutStats(row: SensorStatsRow): (ImmutableBytesWritable, Put) = {
    val p = new Put(Bytes.toBytes(row.rowkey))
```



```
    // add columns with data values to put
    p.add(cfStatsBytes, Bytes.toBytes("hzmax"), Bytes.toBytes(row.maxhz))
    p.add(cfStatsBytes, Bytes.toBytes("hzmin"), Bytes.toBytes(row.minhz))
    p.add(cfStatsBytes, Bytes.toBytes("hzavg"), Bytes.toBytes(row.avghz))
    p.add(cfStatsBytes, Bytes.toBytes("dispmax"), Bytes.toBytes(row.maxdisp))
    p.add(cfStatsBytes, Bytes.toBytes("dispmin"), Bytes.toBytes(row.mindisp))
    p.add(cfStatsBytes, Bytes.toBytes("dispavg"), Bytes.toBytes(row.avgdisp))
    p.add(cfStatsBytes, Bytes.toBytes("flomax"), Bytes.toBytes(row.maxflo))
    p.add(cfStatsBytes, Bytes.toBytes("flomin"), Bytes.toBytes(row.minflo))
    p.add(cfStatsBytes, Bytes.toBytes("floavg"), Bytes.toBytes(row.avgflo))
    p.add(cfStatsBytes, Bytes.toBytes("sedPPMmax"), Bytes.toBytes(row.maxsedPPM))
    p.add(cfStatsBytes, Bytes.toBytes("sedPPMmin"), Bytes.toBytes(row.minsedPPM))
    p.add(cfStatsBytes, Bytes.toBytes("sedPPMavg"), Bytes.toBytes(row.avgsedPPM))
    p.add(cfStatsBytes, Bytes.toBytes("psimax"), Bytes.toBytes(row.maxpsi))
    p.add(cfStatsBytes, Bytes.toBytes("psimin"), Bytes.toBytes(row.minpsi))
    p.add(cfStatsBytes, Bytes.toBytes("psiavg"), Bytes.toBytes(row.avgpsi))
    p.add(cfStatsBytes, Bytes.toBytes("chlPPMmax"), Bytes.toBytes(row.maxchlPPM))
    p.add(cfStatsBytes, Bytes.toBytes("chlPPMmin"), Bytes.toBytes(row.minchlPPM))
    p.add(cfStatsBytes, Bytes.toBytes("chlPPMavg"), Bytes.toBytes(row.avgchlPPM))
    (new ImmutableBytesWritable, p)
  }
}

def main(args: Array[String]) {
  val spark = SparkSession.builder.appName("HBaseRWStats").getOrCreate()
```

```
// for implicit conversions from Spark RDD to Dataframe
val sc = spark.sparkContext
import spark.implicits._

val conf = HBaseConfiguration.create()
conf.set(TableInputFormat.INPUT_TABLE, tableName)
// scan data column family
conf.set(TableInputFormat.SCAN_COLUMNS, "data")

// Load an RDD of rowkey, result(ImmutableBytesWritable, Result) tuples from the table
val hBaseRDD = sc.newAPIHadoopRDD(conf, classOf[TableInputFormat],
                        classOf[org.apache.hadoop.hbase.io.ImmutableBytesWritable],
                        classOf[org.apache.hadoop.hbase.client.Result])
 hBaseRDD.count()

// transform (ImmutableBytesWritable, Result) tuples into an RDD of Results
val resultRDD = hBaseRDD.map(tuple => tuple._2)
resultRDD.count()
// transform RDD of Results into an RDD of SensorRow objects
val sensorRDD = resultRDD.map(SensorRow.parseSensorRow)
// change  RDD of SensorRow  objects to a DataFrame
val sensorDF = sensorRDD.toDF()
// Return the schema of this DataFrame
sensorDF.printSchema()
// Display the top 20 rows of DataFrame
sensorDF.show()
```

```
// group by the rowkey (sensorid_date) get average psi
sensorDF.groupBy("rowkey").agg(avg(sensorDF("psi"))).take(5).foreach(println)
// register the DataFrame as a temp table
sensorDF.createOrReplaceTempView("SensorRow")

// group by the rowkey (sensorid_date) get average, max , min for all columns
val sensorStatDF = spark.sql("SELECT rowkey,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(flo) as maxflo,
      min(flo) as minflo, avg(flo) 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 SensorRow GROUP BY rowkey")
sensorStatDF.printSchema()
sensorStatDF.take(5).foreach(println)

// map the query result row to the SensorStatsRow object
val sensorStatsRowRDD = sensorStatDF.map {
  case Row(rowkey: String, maxhz: Double, minhz: Double, avghz: Double, maxdisp: Double,
    mindisp: Double, avgdisp: Double, maxflo: Double, minflo: Double, avgflo: Double,
    maxsedPPM: Double, minsedPPM: Double, avgsedPPM: Double, maxpsi: Double, minpsi: Double,
    avgpsi: Double, maxchlPPM: Double, minchlPPM: Double, avgchlPPM: Double)
  =>
   SensorStatsRow(rowkey: String, maxhz: Double, minhz: Double, avghz: Double, maxdisp: Double,
     mindisp: Double, avgdisp: Double, maxflo: Double, minflo: Double, avgflo: Double,
     maxsedPPM: Double, minsedPPM: Double, avgsedPPM: Double, maxpsi: Double, minpsi: Double,
     avgpsi: Double, maxchlPPM: Double, minchlPPM: Double, avgchlPPM: Double)
}
```

```
  sensorStatsRowRDD.take(5).foreach(println)

  // set JobConfiguration variables for writing to HBase
  val jobConfig: JobConf = new JobConf(conf, this.getClass)
  jobConfig.set("mapreduce.output.fileoutputformat.outputdir", "/sparkdata/stream/out1")

  // set the HBase output table
  jobConfig.setOutputFormat(classOf[TableOutputFormat])
  jobConfig.set(TableOutputFormat.OUTPUT_TABLE, tableName)

  // convert into rdd and the SensorStatsRow objects into HBase put objects and write to HBase
  sensorStatsRowRDD.rdd.map(SensorStatsRow.convertToPutStats).saveAsHadoopDataset(jobConfig)
 }

}
```


HBaseRWStats 응용의 빌드

- 응용 빌드

  $ sbt package

  - target/scala-2.11/hbaserwstats-project_2.11-1.0.jar 파일 생성

```
MASTER:~/spark/stream/hbaserwstats$ sbt package
[warn] Executing in batch mode.
[warn]   For better performance, hit [ENTER] to switch to interactive mode, or
[warn]   consider launching sbt without any commands, or explicitly passing 'shell'
[info] Loading project definition from /home/cssjlee/spark/stream/hbaserwstats/project
[info] Set current project to HBaseRWStats Project (in build file:/home/cssjlee/spark/stream/hbaserwstats/)
[info] Compiling 1 Scala source to /home/cssjlee/spark/stream/hbaserwstats/target/scala-2.11/classes...
[warn] there were 18 deprecation warnings; re-run with -deprecation for details
[warn] one warning found
[info] Packaging /home/cssjlee/spark/stream/hbaserwstats/target/scala-2.11/hbaserwstats-project_2.11-1.0.jar ...
[info] Done packaging.
[success] Total time: 6 s, completed Aug 21, 2017 12:08:27 PM
```

HBaseRWStats 응용 실행

- spark-submit 명령을 사용하여 실행

  $SPARK_HOME/bin/spark-submit --class HBaseRWStats --master yarn target/scala-2.11/hbaserwstats-project_2.11-1.0.jar

```
c---_ z@MASTER:~/spark/stream/hbaserwstats$ $SPARK_HOME/bin/spark-submit --class HBaseRWStats --ma
ster yarn  target/scala-2.11/hbaserwstats-project_2.11-1.0.jar
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/hadoop/hbase-1.2.6/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/
impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/hadoop/spark-2.1.0-bin-hadoop2.7/jars/slf4j-log4j12-1.7.16
.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
root
 |-- rowkey: string (nullable = true)
 |-- hz: double (nullable = true)
 |-- disp: double (nullable = true)
 |-- flo: double (nullable = true)
 |-- sedPPM: double (nullable = true)
 |-- psi: double (nullable = true)
 |-- chlPPM: double (nullable = true)
```



```
+-----------------+-----+-----+------+------+-----+------+
|           rowkey|   hz| disp|   flo|sedPPM|  psi|chlPPM|
+-----------------+-----+-----+------+------+-----+------+
|ANDOUILLE_3/10/14| 9.82|1.718|1275.0|   0.3| 76.0|  1.44|
|ANDOUILLE_3/10/14| 9.88|1.716|1273.0|   0.1| 80.0|  0.89|
|ANDOUILLE_3/10/14| 9.61|1.715|1272.0|  0.59| 93.0|   2.0|
|ANDOUILLE_3/10/14| 10.3|1.706|1266.0|  1.44| 92.0|  1.88|
|ANDOUILLE_3/10/14| 9.63|1.704|1264.0|  1.81| 92.0|   0.9|
|ANDOUILLE_3/10/14| 9.77|1.702|1262.0|  1.64| 91.0|  0.59|
|ANDOUILLE_3/10/14| 9.99|1.697|1259.0|  1.78| 87.0|  0.74|
|ANDOUILLE_3/10/14|10.27|1.695|1258.0|  0.81| 86.0|  0.75|
|ANDOUILLE_3/10/14| 9.92|1.692|1255.0|  1.95| 79.0|  1.34|
|ANDOUILLE_3/10/14|10.19|1.687|1251.0|  0.91| 78.0|  0.56|
|ANDOUILLE_3/10/14|  9.7|1.683|1249.0|  0.79|100.0|  1.58|
|ANDOUILLE_3/10/14|10.36| 1.68|1246.0|  1.59| 83.0|   0.7|
|ANDOUILLE_3/10/14| 9.87|1.674|1242.0|  1.53| 83.0|   1.5|
|ANDOUILLE_3/10/14| 9.92|1.669|1238.0|  0.79| 92.0|  0.56|
|ANDOUILLE_3/10/14| 9.92|1.666|1236.0|  1.83| 76.0|   1.2|
|ANDOUILLE_3/10/14| 9.93|1.663|1234.0|  0.23| 77.0|  0.71|
|ANDOUILLE_3/10/14|10.28|1.653|1226.0|  1.34| 92.0|  0.77|
|ANDOUILLE_3/10/14| 9.74|1.645|1221.0|  1.23| 92.0|   1.9|
|ANDOUILLE_3/10/14|10.44|1.644|1220.0|  1.35| 99.0|  1.01|
|ANDOUILLE_3/10/14|10.21|1.635|1213.0|  0.39| 87.0|  1.83|
+-----------------+-----+-----+------+------+-----+------+
only showing top 20 rows

[MOJO_3/10/14,87.20876826722338]
[CARGO_3/11/14,87.2901878914405]
[MOJO_3/12/14,87.55219206680584]
[NANTAHALLA_3/11/14,79.61169102296451]
[CARGO_3/12/14,87.76304801670146]
root
 |-- rowkey: string (nullable = true)
 |-- maxhz: double (nullable = true)
 |-- minhz: double (nullable = true)
 |-- avghz: double (nullable = true)
 |-- maxdisp: double (nullable = true)
 |-- mindisp: double (nullable = true)
 |-- avgdisp: double (nullable = true)
 |-- maxflo: double (nullable = true)
 |-- minflo: double (nullable = true)
 |-- avgflo: double (nullable = true)
 |-- maxsedPPM: double (nullable = true)
 |-- minsedPPM: double (nullable = true)
 |-- avgsedPPM: double (nullable = true)
 |-- maxpsi: double (nullable = true)
 |-- minpsi: double (nullable = true)
 |-- avgpsi: double (nullable = true)
 |-- maxchlPPM: double (nullable = true)
 |-- minchlPPM: double (nullable = true)
 |-- avgchlPPM: double (nullable = true)

[MOJO_3/10/14,10.5,9.5,9.999457202505226,3.345,1.8
08352,2.0,0.0,0.9798121085594999,100.0,75.0,87.208
[CARGO_3/11/14,10.5,9.5,10.010824634655517,3.864,1
699374,2.0,0.0,0.9811482254697279,100.0,75.0,87.29
[MOJO_3/12/14,10.5,9.5,10.006482254697284,3.589,2.
887266,2.0,0.0,0.9952087682672222,100.0,75.0,87.55
[NANTAHALLA_3/11/14,10.5,9.5,9.993350730688954,4.0
12317327766,2.0,0.0,1.002672233820459,100.0,66.0,7
[CARGO_3/12/14,10.5,9.5,10.01497912317328,3.905,1.
41545,1.99,0.0,1.0017954070981216,100.0,75.0,87.76
17/08/21 12:21:00 WARN Utils: Truncated the string
. This behavior can be adjusted by setting 'spark.
SensorStatsRow(MOJO_3/10/14,10.5,9.5,9.99945720250
,1385.8131524008352,2.0,0.0,0.9798121085594999,100
9168)
SensorStatsRow(CARGO_3/11/14,10.5,9.5,10.010824634
0,1204.7265135699374,2.0,0.0,0.9811482254697279,10
4743)
SensorStatsRow(MOJO_3/12/14,10.5,9.5,10.006482254
0,1528.2724425887266,2.0,0.0,0.9952087682672222,10
11684)
SensorStatsRow(NANTAHALLA_3/11/14,10.5,9.5,9.99335
,817.0,1188.2912317327766,2.0,0.0,1.00267223382045
2797494774)
SensorStatsRow(CARGO_3/12/14,10.5,9.5,10.014979123
0,1230.579331941545,1.99,0.0,1.0017954070981216,10
0771)
```

HBase 테이블 확인

- 스파크 셸에서 HBase 테이블 확인

  hbase> scan 'oil_sensor', {COLUMNS=>['stats'], LIMIT=> 2}

```
hbase(main):003:0> scan 'oil_sensor', {COLUMNS=>['stats'], LIMIT=> 2}
ROW                           COLUMN+CELL
 ANDOUILLE_3/10/14            column=stats:chlPPMavg, timestamp=1503285665080, value=?\xF3\x9EX\x8C\x11[\xE1
 ANDOUILLE_3/10/14            column=stats:chlPPMmax, timestamp=1503285665080, value=@\x00\x00\x00\x00\x00\x00\x
                              00
 ANDOUILLE_3/10/14            column=stats:chlPPMmin, timestamp=1503285665080, value=?\xE0\x00\x00\x00\x00\x00\x
                              00
 ANDOUILLE_3/10/14            column=stats:dispavg, timestamp=1503285665080, value=?\xF9\x83KY3\x88\x8D
 ANDOUILLE_3/10/14            column=stats:dispmax, timestamp=1503285665080, value=@\x00\xE71\x8BC\x95\x81
 ANDOUILLE_3/10/14            column=stats:dispmin, timestamp=1503285665080, value=?\xF0\xC4\x9B\xA5\xE3S\xF8
 ANDOUILLE_3/10/14            column=stats:floavg, timestamp=1503285665080, value=@\x92{\xF2\x1A\xB8\xEB(
 ANDOUILLE_3/10/14            column=stats:flomax, timestamp=1503285665080, value=@\x98|\x00\x00\x00\x00\x00
 ANDOUILLE_3/10/14            column=stats:flomin, timestamp=1503285665080, value=@\x88H\x00\x00\x00\x00\x00
 ANDOUILLE_3/10/14            column=stats:hzavg, timestamp=1503285665080, value=@$\x07\xC3\xF8\x8F|\xFE
 ANDOUILLE_3/10/14            column=stats:hzmax, timestamp=1503285665080, value=@%\x00\x00\x00\x00\x00\x00
 ANDOUILLE_3/10/14            column=stats:hzmin, timestamp=1503285665080, value=@#\x00\x00\x00\x00\x00\x00
 ANDOUILLE_3/10/14            column=stats:psiavg, timestamp=1503285665080, value=@U\xF0o*8\xA6\xBF
 ANDOUILLE_3/10/14            column=stats:psimax, timestamp=1503285665080, value=@Y\x00\x00\x00\x00\x00\x00
 ANDOUILLE_3/10/14            column=stats:psimin, timestamp=1503285665080, value=@R\xC0\x00\x00\x00\x00\x00
 ANDOUILLE_3/10/14            column=stats:sedPPMavg, timestamp=1503285665080, value=?\xF08.X\xE3\xA1\xB0
 ANDOUILLE_3/10/14            column=stats:sedPPMmax, timestamp=1503285665080, value=@\x00\x00\x00\x00\x00\x00\x
                              00
 ANDOUILLE_3/10/14            column=stats:sedPPMmin, timestamp=1503285665080, value=\x00\x00\x00\x00\x00\x00\x0
                              0\x00
```


과제

- 자신만의 데이터를 사용하여 앞에서 배운 스파크를 적용하고 실행
  - 텀 프로젝트와 관련하여 연관성이 있는 데이터

참고 자료

- HBase
  - http://hbase.apache.org
- Coreservelets.com Hadoop Tutorial
  - HBase Part 1 – Overview
    - http://www.coreservlets.com/hadoop-tutorial/#HBase-1
  - HBase Part 2 – Installation and Shell
    - http://www.coreservlets.com/hadoop-tutorial/#HBase-2
  - HBase Part 3 – Java Client API
    - http://www.coreservlets.com/hadoop-tutorial/#HBase-3
- Overview of Spark Streaming
  - https://mapr.com/blog/spark-streaming-hbase/