Spark Streaming

데이터 처리

- 배치 처리
  - 고정된 과거의 데이터 처리
  - 처리하는 데이터의 크기가 크고 작업 수행시간이 상대적으로 긴 처리
  - 작업 도중 처리에 실패하더라도 재작업을 통해 동일한 최종 결과 재산출 가능

- 실시간 처리
  - 데이터 생성 시점과 처리되는 시점 사이의 간격이 짧은 처리
  - 생성된 후 가능한 빠르게 처리해야 유의미한 결과를 얻을 수 있는 경우

스파크 스트리밍

- 실시간 처리가 필요한 데이터를 다루기 위한 스파크의 서브 모듈
- 실시간으로 변하는 데이터를 (배치 처리보다) 짧은 주기에 맞춰 빠르고 안정적으로 처리하는 데 필요한 기능 제공
- 일정한 주기마다 연속적으로 발생한 새로운 데이터를 읽어서 처리
  - 이전에 발생한 데이터와 새로 발생한 데이터를 결합해서 필요한 처리 수행
  - 이 작업을 애플리케이션이 종료될 때까지 무한 반복
- 배치 처리에 비해 처리 주기 사이의 시간 간격에 대한 정보가 추가로 필요

주요 객체

- StreamingContext 객체
  - RDD와 데이터셋을 사용하기 위해 SparkContext와 SparkSession을 사용한 것처럼 스파크 스트리밍 작업을 수행하기 위해 StreamingContext 객체 필요

```
from pyspark.streaming import StreamingContext
from pyspark.streaming.kafka import KafkaUtils

ssc = new StreamingContext(sc, 5)
```

- DStream(Discretized Streams)
  - 고정되지 않고 끊임없이 생성되는 연속된 데이터를 나타내기 위한 추상 모델
  - 일정 시간마다 데이터를 모아서 RDD 생성 → 이 RDD로 만들어진 시퀀스

데이터 소스의 종류

- 기본 데이터 소스
  - 스파크 단독으로 처리할 수 있는 데이터 소스
  - 소켓, 파일, RDD 큐 등

- 어드밴스드 데이터 소스
  - 외부 라이브러리의 도움이 필요한 데이터 소스
  - 카프카, 플럼, 키니시스, 트위터 등

주요 연산

연산	설명
print()	DStream에 포함된 각 RDD의 내용을 콘솔에 출력 기본 10개 → 개수 지정 가능
map(func)	DStream의 RDD에 포함된 각 원소에 func 함수를 적용한 결괏값으로 구성된 새로운 DStream 반환
flatMap(func)	RDD의 flatMap()과 같으며 DStream의 RDD에 포함된 각 원소에 func 함수를 적용 하나의 입력이 0 ~ N개의 출력으로 변환
reduce(func), reduceByKey(func)	DStream에 포함된 RDD 값들을 집계해서 하나의 값으로 변환 RDD가 key ~ value 형식의 튜플인 경우 reduceByKey를 사용해서 key 별로 집계 수행 가능 반환 값은 새로운 DStream
filter	DStream의 모든 요소에 함수를 적용하고 그 결과가 true인 요소만 포함한 새로운 DStream 반환
union	두 개의 DStream 요소를 모두 포함하는 새로운 DStream 생성
join, leftOuterJoin, rightOuterJoin, fullOuterJoin	키와 값 쌍으로 구성된 두 개의 DStream을 키를 이용해 결합(Join)

주요 연산

연산	설명
transform(func)	DStream 내부의 RDD에 func를 적용하고 그 결과로 새로운 DStream 반환
updateStateByKey	새로 생성된 데이터와 이전 배치의 최종 상태 값을 함께 전달해서 현재까지의 누적 연산 수행
window, reduceByWindow, reduceByKeyAndWindow	윈도우 연산. 현재 배치 주기의 데이터와 집계되지 않은 이전 배치 주기의 입력 데이터와 함께 처리 가능
saveAsTextFiles. saveAsObjectFiles, saveAsHadoopFiles	DStream의 데이터를 텍스트, 객체 등의 형식으로 파일에 저장
foreachRDD	DStream에 포함된 각 RDD 별로 연산 수행
CheckPoint	잡이 실행되는 동안 생성된 중간 결과물을 안정성이 높은 저장소에 저장 → 장애 등의 복구에 사용
cache, persist	반복적으로 사용되는 데이터를 저장해두고 재사용

Structured Streaming

- 지속적으로 생성되는 데이터를 무한히 증가하는 하나의 커다란 데이터셋으로 추상화

- 일반적인 고정된 형태의 데이터 처리(배치 처리)와 유사한 방법을 사용해서 스트리밍 데이터 처리

- 데이터가 생성될 때마다 필요한 처리를 수행하고 이를 기존 처리 결과에 합치는 처리 방법
  - 기존 처리와 생성된 데이터 간의 병합 작업은 스파크에서 처리

주요 객체

- DataStreamReader
  - DataSet을 생성하는 도구
  - 스파크 세션의 readStream 메서드를 사용해서 생성
  - 파일 소스, 소켓 소스, 카프카 소스 등 지원

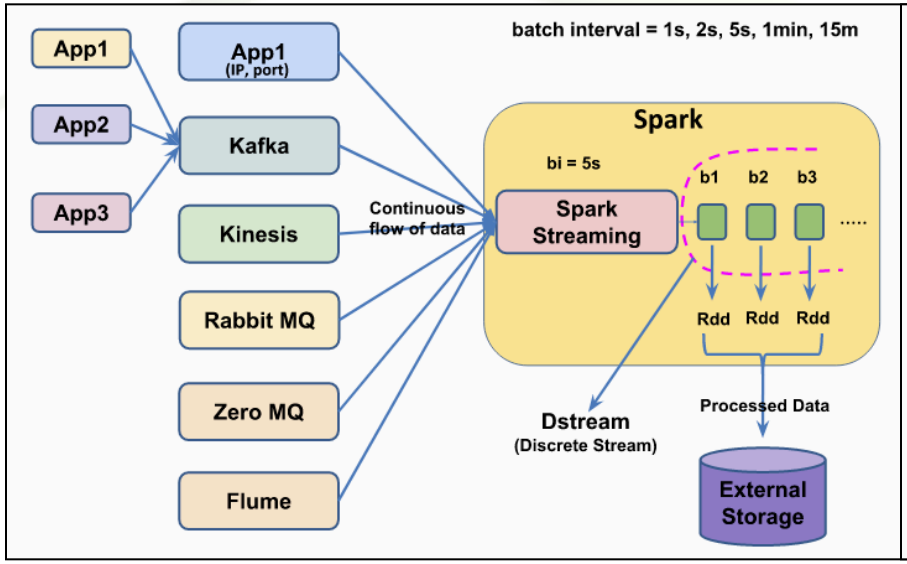
```
df = spark.readStream \
          .format('kafka') \
          .option('kafka.bootstrap.servers', 'server-pd:9092′) \
          .option('subscribe', 'Demo-Topic') \
          .load()
```

- DataStreamWriter
  - 데이터 저장을 지원하는 도구
  - 데이터를 저장할 대상, 저장 모드, 쿼리명, 트리거 주기 등을 설정

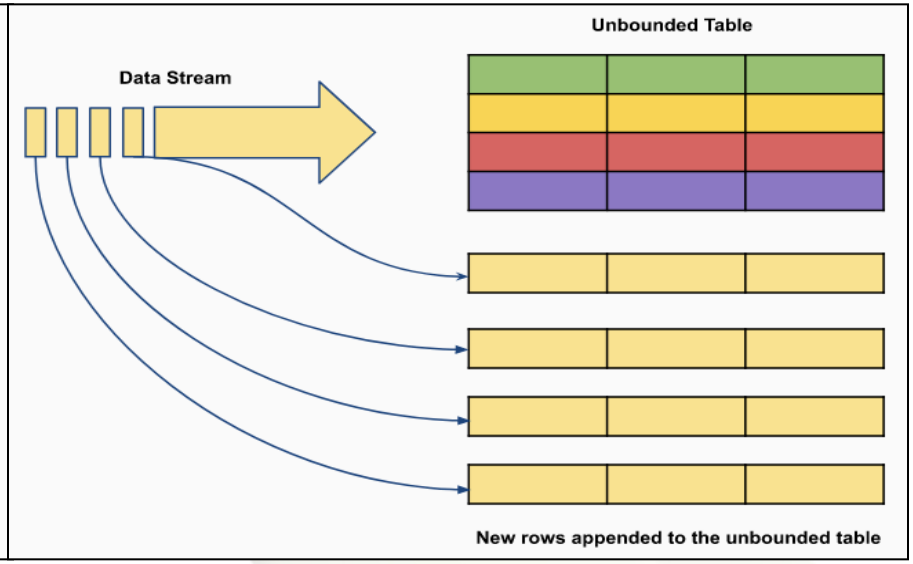
```
query = word_count.writeStream \
                  .outputMode('complete’) \
                  .format('console') \
                  .start()
```

Dstream vs Structured Streaming

DStream



Structured Streaming



DStream	Structured Streaming
작은 batch 단위 처리	batch 단위 처리를 추상화하고 크기가 확정되지 않은 데이터프레임 제공
RDD 기반 API 사용	DataFrame & DataSet 기반 API 사용
데이터 발생 순서와 처리 순서 불일치 가능 데이터 처리 정확도와 신뢰도가 상대적으로 낮음	데이터 발생 순서와 처리 순서의 일치 보장 데이터 처리 정확도와 신뢰도가 상대적으로 높은
입력 소스와 출력 타겟이 다양	제한된 입력 소스와 출력 타겟

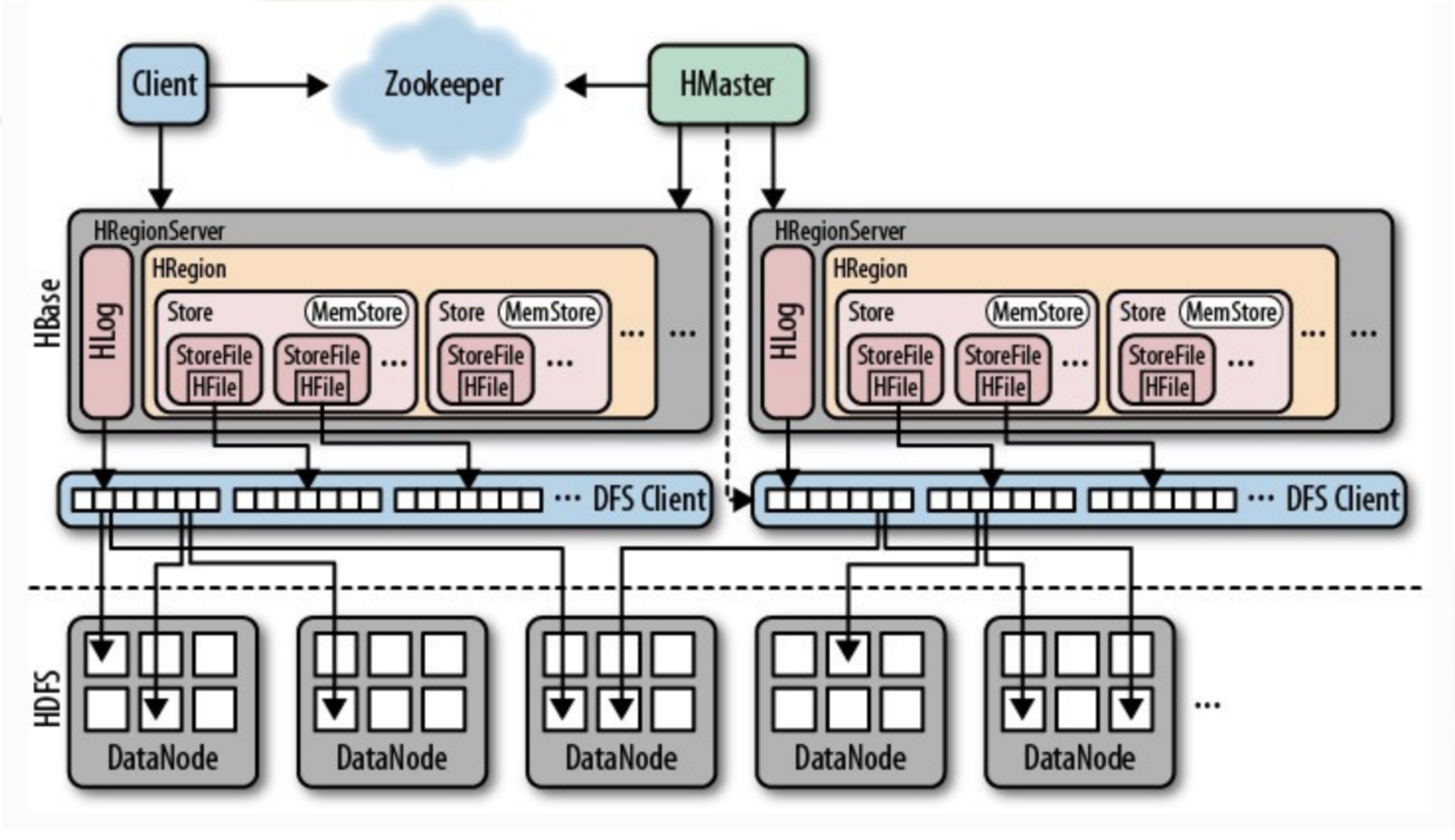
hbase

- Hadoop 기반 컬럼 지향 NoSQL 데이터베이스
  - 데이터베이스를 키/값 형식으로 단순하게 구조화
  - 고성능의 읽기/쓰기 지원 (특히 쓰기 중심의 최적화)

- 대규모 데이터셋에서 실시간으로 읽고 쓰는 랜덤 액세스가 필요할 때 사용할 수 있는 Hadoop 애플리케이션

- 노드만 추가하면 선형으로 확장 가능한 구조 → 클러스터에서 대용량의 산재된 테이블 관리 가능

Hbase Architecture



주요 구성 요소

구성요소	설명
HTable	• 컬럼 기반 데이터 구조를 정의한 테이블 • 공통점이 있는 컬럼들의 그룹을 묶은 컬럼패밀리와 테이블의 로우를 식별하기 위한 로우키로 구성
HMaster	• HRegion Server 관리 • HRegion들이 속한 Server의 메타 정보 관리
HRegion	• HTable의 크기에 따라 자동으로 수평 분할 발생 → 이 때 분할된 블럭을 HRegion 단위로 지정
HRegionServer	• 분산 노드별로 HRegionServer 구성 • 하나의 HRegionServer에 포함된 다수의 HRegion 관리
Store	• 컬럼 패밀리 저장 관리 • MemStore와 HFile로 구성
MemStore	• Store 내의 데이터를 인메모리에 저장 관리하는 데이터 캐시 영역
HFile	• Store 내의 데이터를 스토리지에 저장 및 관리하는 영구 저장 영역

특징

- 인덱스 없음 →
  - 로우는 순차적으로 저장되며 각 로우별 컬럼도 순차적으로 저장
  - 인덱스 확장 문제가 발생하지 않고 입력 성능은 테이블 크기와 무관

- 자동 분할
  - 테이블의 크기가 증가하면 자동으로 여러 리전으로 분할되고 가용한 모든 노드로 분산

- 새로운 노드를 추가하여 자동으로 선형 확장
  - 노드를 추가하고 이를 기존 클러스터에 붙이고 리전 서버 실행 → 리전은 자동으로 균형을 맞추고 부하는 균등하게 분산

- 범용 하드웨어
  - 상대적으로 저렴한 하드웨어 사용 가능

특징

- 내고장성
  - 수많은 노드로 구성되기 때문에 개별 노드의 문제는 상대적으로 큰 문제가 되지 않음 → 개별 노드의 장애가 전체 클러스트에 영향을 주지 않는 구조

- 배치 처리
  - 맵리듀스와의 통합으로 데이터의 지역성을 준수하고 완전한 병렬성과 분산 작업 지원

HBase 동작

- 쓰기
  - 클라이언트는 데이터를 저장하기 전 zookeeper를 통해 HTable의 기본 정보와 해당 HRegion의 위치 정보 파악
  - HRegion 서버에 접속해서 HRegion의 메모리 영역인 MemStore에 데이터 저장
  - MemStore의 임계값을 넘으면 MemStore의 데이터는 HFile로 플러시
  - HFile의 임계값을 넘으면 HDFS로 데이터 플러시

- 읽기
  - zookeeper를 통해 로우키에 해당하는 데이터 위치 정보 파악
  - HRegion 서버에 포함된 HRegion의 메모리 영역인 MemStore에서 데이터 검색
  - 데이터가 MemStore에 없으면 HFile에서 데이터 검색
  - HFile에 데이터가 없으면 HDFS에서 데이터 검색