Spark Overview

(아파치 스파크를 써야 하는 이유)

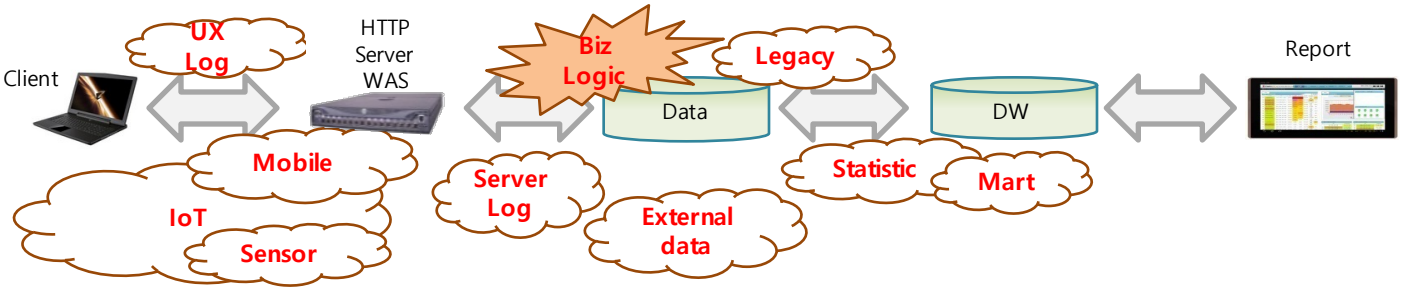
SK 주식회사 C&C
이상훈

Spark Overview

- 빅데이터 플랫폼
- Spark 란?
- Spark Streaming
- 고급 분석

빅데이터 플랫폼

빅데이터 플랫폼의 필요성

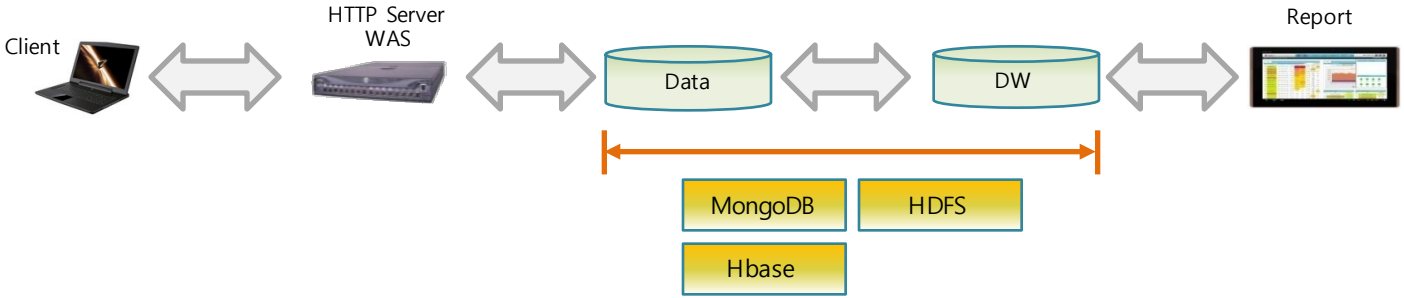


- 3-Tier 의 웹 기반 서비스가 가장 보편화 되어 있음 (웹로그, 사용자 클릭 정보)
- 고객 정보를 통한 통계, 분석을 위한 DW, BI, OLAP 등 도입
- 비즈니스 로직이 DBMS 영역으로 이관되는 경우 (Open API가 대표적)

- Mobile(스마트 폰) 시장 활황으로 고객 데이터가 급격히 증가
- 외부 데이터와 연동을 통한 고급 / 연계 분석 시도
- 센서, 사물인터넷 등의 데이터 증가 지속

-> 데이터베이스의 확장 또는 새로운 개념의 DB 필요

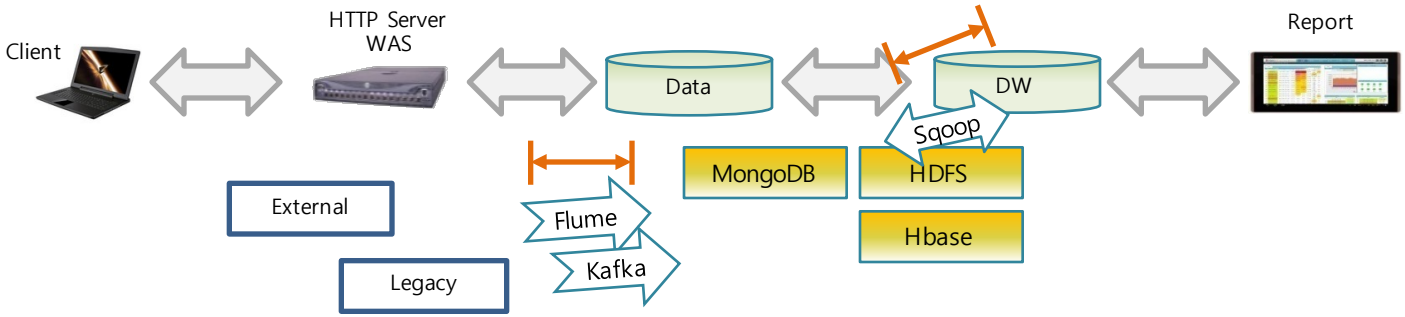
저장 플랫폼



- **분산파일시스템** : 데이터를 분산 환경에 분리하여 저장/처리/요청할 수 있도록 구성된 파일 시스템
- **NoSQL** : 구글의 BigTable 에 기반한 Key/Value DB, Document DB 등
- **사용 구분 :**

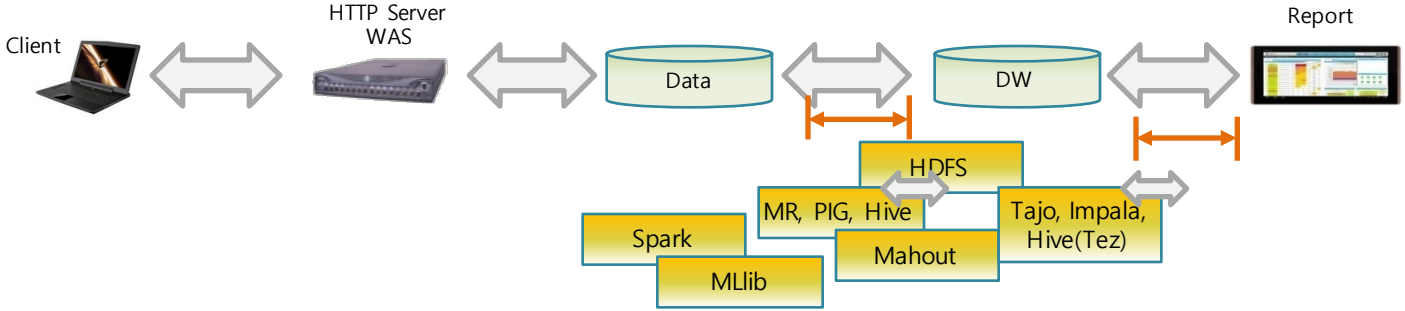
RDBMS	NoSQL	Hadoop HDFS
• 비즈니스 데이타 (계좌, 고객 등) • 엄격한 트랜잭션 처리(ACID) • 다수의 사용자에 대해 정합성과 안정성 보장 • 100% SQL Compliance • 고비용	• SNS, 블로그 등의 텍스트 • Partial Consistency -> Delay 허용 • 유연성과 효율성 • 특화된 용도에 맞게 사용 • RDBMS와는 보완 관계 • 선택의 폭이 넓어짐	• 웹/센서 로그 등의 low density data • Parallel Batch Processing • 트랜잭션 지원 안함 • 데이터 전 처리 및 집계에 적합 • 저비용

수집/연동 플랫폼



- **수집 / 연동 플랫폼** : **다양한** 다수의 서버로 부터 데이터를 수집하여 **다양한** 저장 플랫폼에 저장
- Flume : 설정 및 구성이 비교적 간단하여 대표적인 로그 수집 시스템으로 이용, Fail-over, 유연성 등 대규모 로그 처리에 적합한 기능을 가짐
- Sqoop : JDBC 기반으로 다양한 DBMS 벤더와 공동작업, 데이터 처리하는 MR프로그램 지원, Hive와 통합하여 SQL 기반 환경으로 편리하게 이용가능, 안정적인 성능 보장
- Kafka : 대용량의 실시간 로그처리에 특화된 설계를 통하여 기존 메시징 시스템보다 우수한 TPS
- 기타 데이터 연동 방법 : FTP, Fuse, webHDFS, Chukwa, HIHO 등

배치 처리/분석 플랫폼



- **배치 처리/분석 플랫폼** : 초기의 하둡 기반 플랫폼은 배치 처리에 강점을 가지고 있었으나 실시간 처리, 고급 분석 등의 한계로 인해 관련 에코 시스템들이 포함되고 지속적인 기능 개선이 일어남
- PIG, Hive : Hadoop 초기에 Mapreduce의 숙련 시간을 줄여 비교적 간단한 기능을 수행할 수 있도록 스크립트 레벨의 언어를 제공, 초기 활성화에 기여함
- Mahout : MR을 이용해 클러스터링, 분류, 분석 작업 등의 병렬처리 가능한 기계학습 라이브러리
- MLlib : Spark를 기반으로 빌드된 기계 학습 라이브러리
- **SQL on Hadoop(**Tajo, Impala,presto 등**)** : Hive에 단점을 보완하기 위한 시도, hdfs에 저장된 파일을 MR이 아닌 별도의 컴퓨팅 플랫폼을 이용하여 질의 실행

```
import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount {

  public static class TokenizerMapper
       extends Mapper<Object, Text, Text, IntWritable>{

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(Object key, Text value, Context context
                    ) throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
      }
    }
  }

  public static class IntSumReducer
       extends Reducer<Text,IntWritable,Text,IntWritable> {
    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values,
                       Context context
                       ) throws IOException, InterruptedException {
      int sum = 0;
      for (IntWritable val : values) {
        sum += val.get();
      }
      result.set(sum);
      context.write(key, result);
    }
  }

  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

Hive 등장 배경

기존 시스템을 전부.. MapReduce code로 전환한다면 필요한 시간은?

어떻게 설계해야 성능이 나오지?

잦은 수정이 필요하다면?

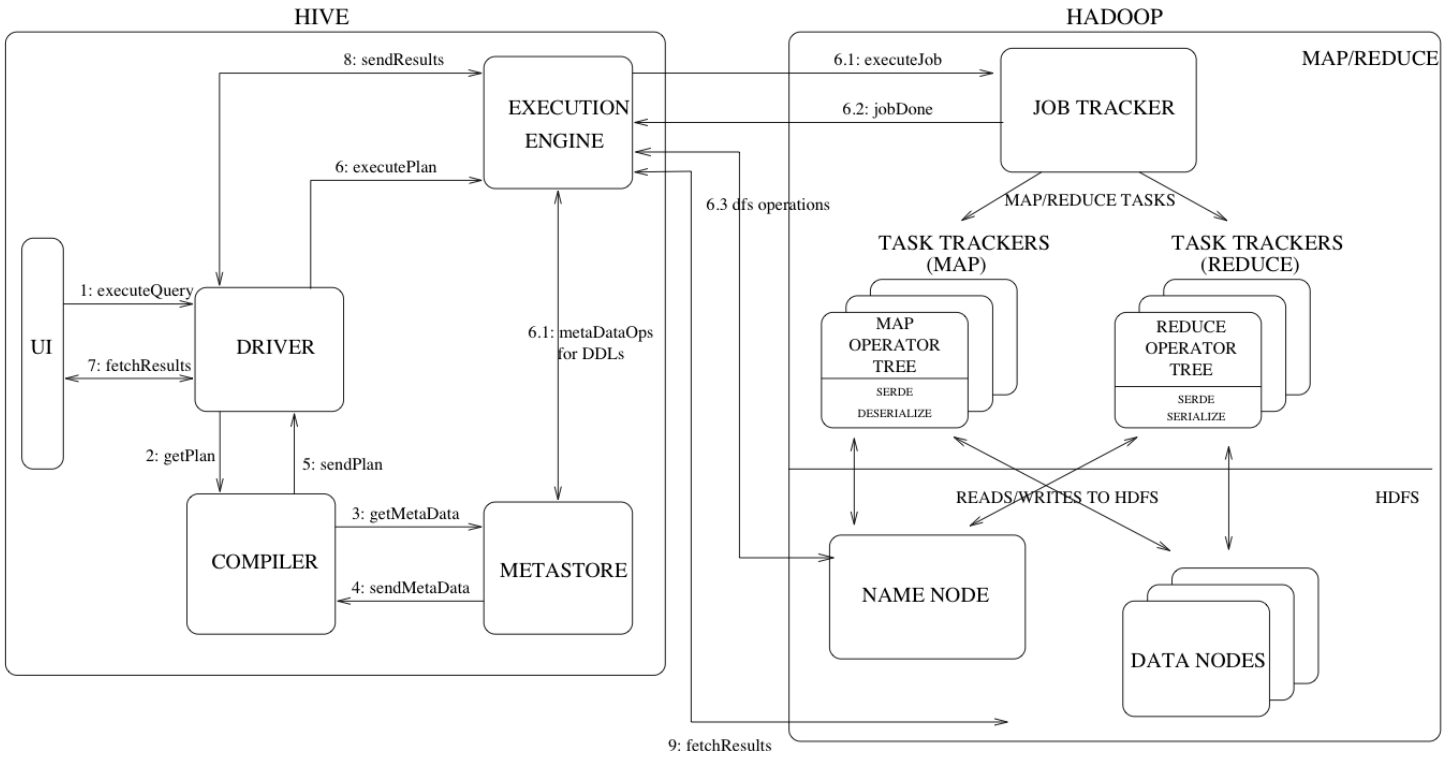
한 번 만 데이터 뽑으면 되는데...

Big Data기반 Platform에서 복잡한 MapReduce 프로그램을 직접 개발하지 않고.. 기존 SQL기반으로 쉽게 개발 가능하다면..?!

Hive 란?

- HiveQL(SQL과 이슈)를 이용하여 MapReduce를 수행하도록 도와주는 오픈소스
- Hadoop에 있는 데이터에 쉽게 접근할 수 있으며 데이터 심화 분석을 위한 사전 분석 작업이나 리포팅 작업으로 많이 사용됨
- 각종 함수 뿐만 아니라 복잡한 분석을 위한 UDF 지원

Hive 아키텍쳐


HIVE
HADOOP
MAP/REDUCE
8: sendResults
EXECUTION ENGINE
6.1: executeJob
6.2: jobDone
JOB TRACKER
6: executePlan
6.3 dfs operations
MAP/REDUCE TASKS
TASK TRACKERS (MAP)
TASK TRACKERS (REDUCE)
1: executeQuery
UI
DRIVER
7: fetchResults
6.1: metaDataOps for DDLs
MAP OPERATOR TREE
SERDE DESERIALIZE
REDUCE OPERATOR TREE
SERDE SERIALIZE
2: getPlan
5: sendPlan
READS/WRITES TO HDFS
HDFS
3: getMetaData
COMPILER
METASTORE
4: sendMetaData
NAME NODE
DATA NODES
9: fetchResults

RDBMS와 Hive의 차이점

- 쿼리 응답 속도가 (작은 데이터 기준) 느림

- 레코드 단위 Insert, delete or update
  - 지원하지 않음
  - 게다가 Transaction도 지원하지 않음
  - 그래서 case문 등을 이용해서 복잡하게 구현해야 함(속도도 더 느림)

- 통계정보도 바로 확인할 수 없음

- 인풋 데이터의 오류를 바로 확인 할 수 없음

Schema On WRITE (RDBMS)

- **Create schema**
  - **CREATE TABLE customer(id string, name string, ...);**
- **Add data**
  - **BULK INSERT custormer FROM "c:₩data₩customer" WITH filedterminator='",";**
- **Query**
  - **SELECT id, name FROM custormer;**

Schema On WRITE (RDBMS)

- Create schema
  - CREATE TABLE customer(id string, name string, ...);
- Add data
  - BULK INSERT custormer FROM "c:₩data₩customer" WITH filedterminator='",";
- Query
  - SELECT id, name FROM custormer;

- SQL에서는 테이블 스키마를 선언하기 전까지는 데이터를 넣을 수 없음
- 테이블 스키마가 변경되게 되면 테이블을 drop하고 데이터를 reload시켜야 함
  - 작은 데이터에서는 문제없음
  - 그러나, 수 백 TB라면? 그리고 foreing key가 변경되었다면?

Schema On READ (Hive)

- **Create schema**
  - **CREATE (EXTERNAL) TABLE customer(id string, name string, ...)**

- **LOAD THE DATA**
  - **hdfs dfs -copyfromlocal /data/ /user/hadoop/customer**

- **Query**
  - **SELECT id, name FROM custormer**

Schema On READ (Hive)

- **Create schema**
  - **CREATE (EXTERNAL) TABLE customer(id string, name string, ...)**

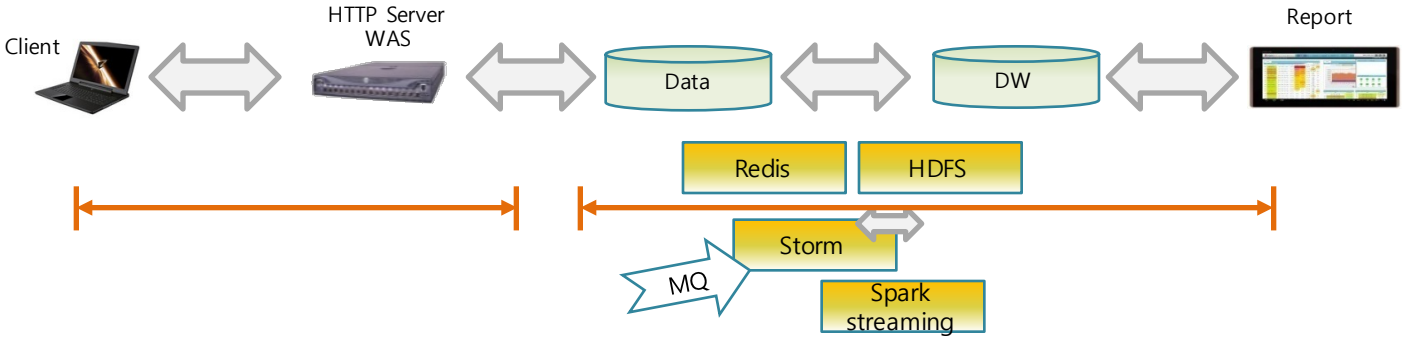
- **LOAD THE DATA**
  - **hdfs dfs -copyfromlocal /data/ /user/hadoop/customer**

- **Query**
  - **SELECT id, name FROM custormer**

- **SQL에서는 테이블 스키마를 선언하기 전에도 데이터를 hdfs에 넣을 수 있음**

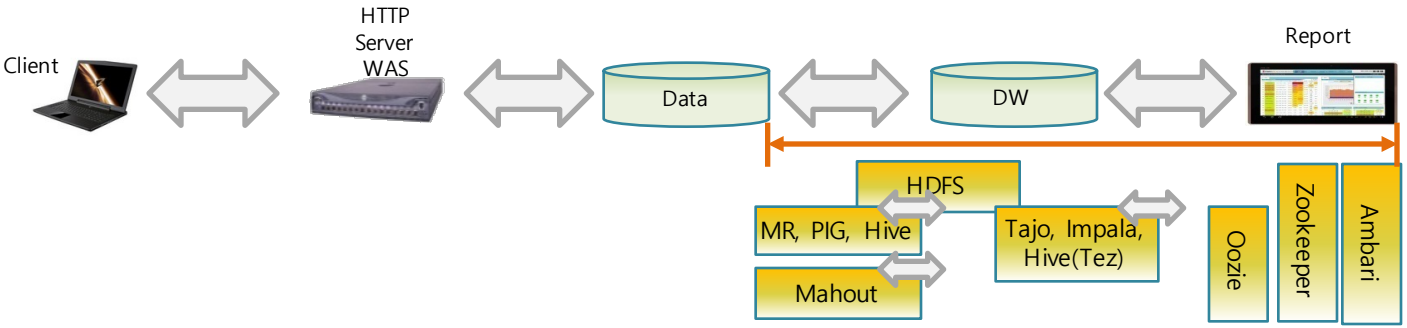
– **=> RDBMS 대비 부족한 점이 있어도 Hive를 써야하는 이유 : 빅데이터는 데이터 사이즈가 클 뿐만 아니라 비정형 데이터도 많기 때문에 데이터 타입이나 컬럼들이 분석함에 따라 자주 바뀜.**

실시간 처리/분석 플랫폼



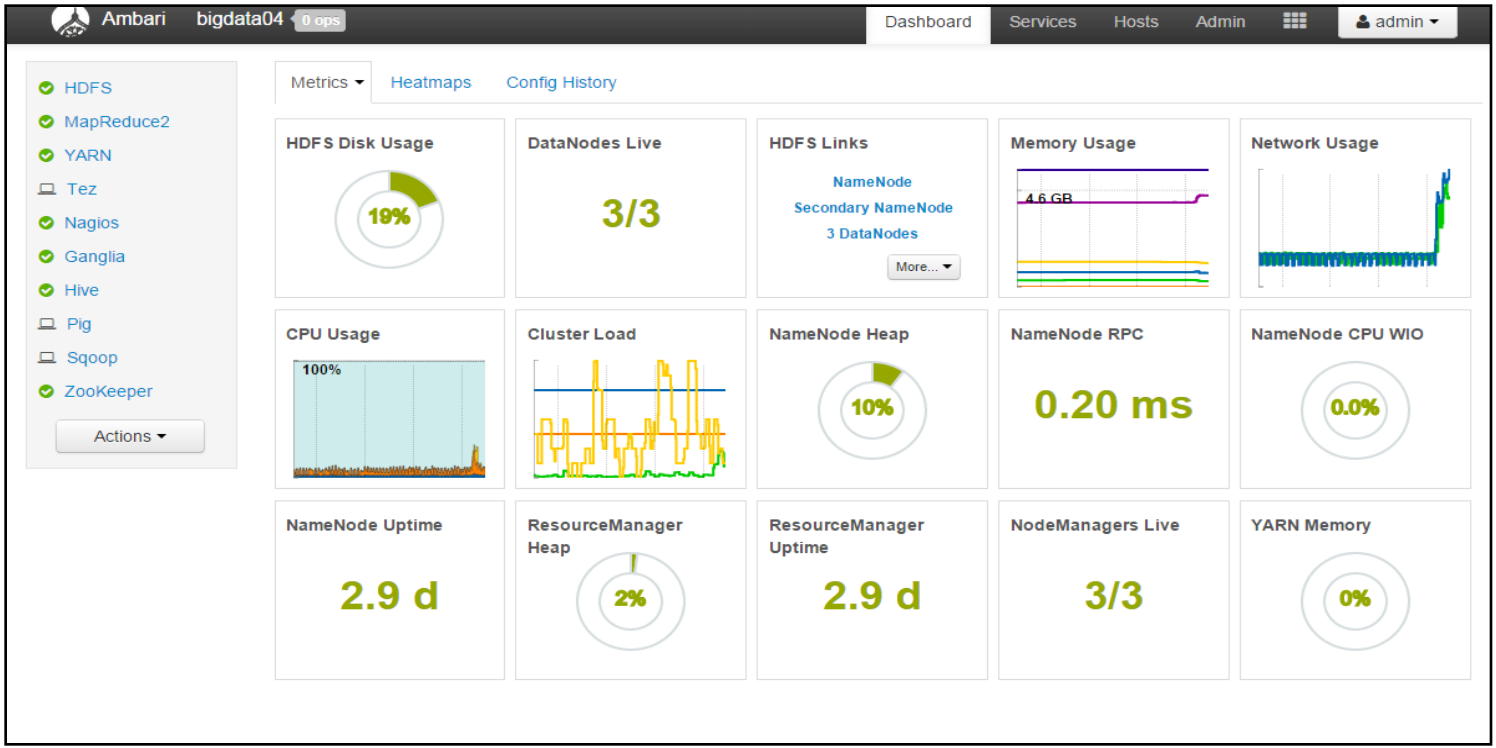
- **실시간 처리/분석 플랫폼** : 데이터 종류에 따라 다양한 형태의 에코시스템을 최적화 조합하여 데이터 수집, 처리, 전송이 모두 실시간으로 이루어지도록 구성
- Message Queue : 실시간으로 수집된 데이터를 Streaming 전송, 실시간 처리를 위한 첫 단추임. Kafka, Storm 등이 MQ 기능을 포함하고 있음.
- Storm : 로컬 및 분산 모드 지원, Hadoop 프로세스를 메모리 상에서 처리한다고 생각하면 간단함
- Spark Streaming : 실시간 데이터를 대규모, 고성능, 장애 허용 가능하게 스트리밍을 처리하는 핵심 Spark API의 확장판
- Redis : **In-Memory Key-Value DB**라 빠른 속도가 강점, 실시간 처리에 적합

관리/운영 플랫폼

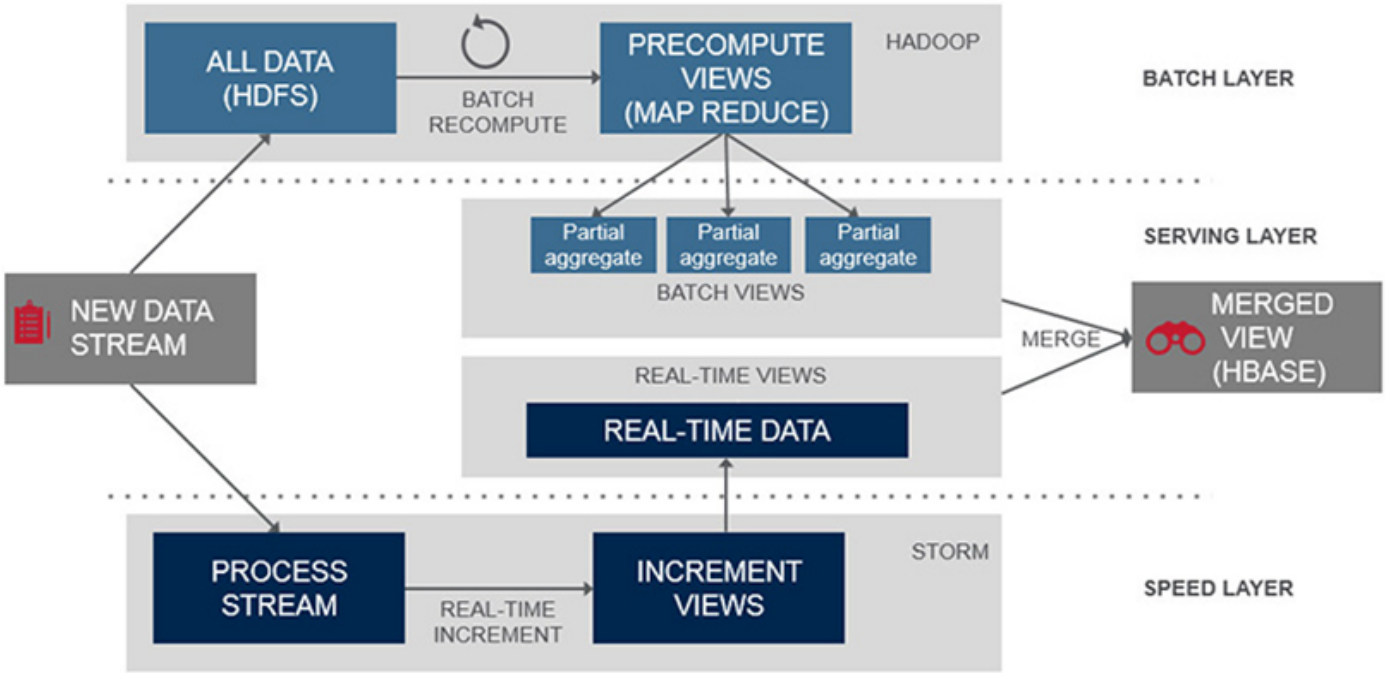


- **관리 운영 플랫폼** : 하둡 에코시스템이 갈수록 복잡해짐에 따라 프로세스 관리 및 클러스터 관리의 필요성이 대두되면서 관련된 오픈 소스 프로젝트들이 등장
- Ambari : 마법사 기반 설치 지원 , 하둡 서비스와 구성 요소의 세부 구성, metrics 수집 및 시스템 경고에 대한 Nagios, Ganglia 포함, 상세 Job 진단 및 문제 해결 도구, 클러스터 히트 맵
- Oozie : MR, Pig, Hive 등을 구현한 프로세스들의 Workflow를 디자인하고 실행하게 해 줌. XML 형식으로 프로세스를 디자인 하므로 복잡한 프로세스 적용에 어려움 - 디자인 UI 가 필요함
- Zookeeper : 분산 환경 서버들간에 상호 조정이 필요한 다양한 서비스를 제공. 분산 동시 처리, 서버들간의 동기화, fail-over로 무중단 서비스, 서버들간 환경 설정 관리 기능 등 제공

Apache Ambari (HDP)



Lambda Architecture



너무 복잡한 기술들

- Lambda Architecture
  - 너무 많은 오픈소스
  - 관리하기 어려움
  - 더 빠른 속도가 필요

- Etc
  - Window Function
  - Machine Learning
  - Analytics

Spark 란?

Spark란?

- **대용량 Data Processing**을 위한 빠르고 General 한 엔진
- Hadoop MapReduce와 비슷한 개념의 새로운 **Computing Framework**
- Written in **Scala, Java, Python** (Mostly in Scala)
- Apache License 2.0
- Developers: U.C Berkeley, AMPLab, ASF
- **In-memory Cluster Computing** 기능을 제공
- Apache에서 가장 활발한 3개 프로젝트 중 하나
- Spark 1.6.2 Version Released recently

설계 목표

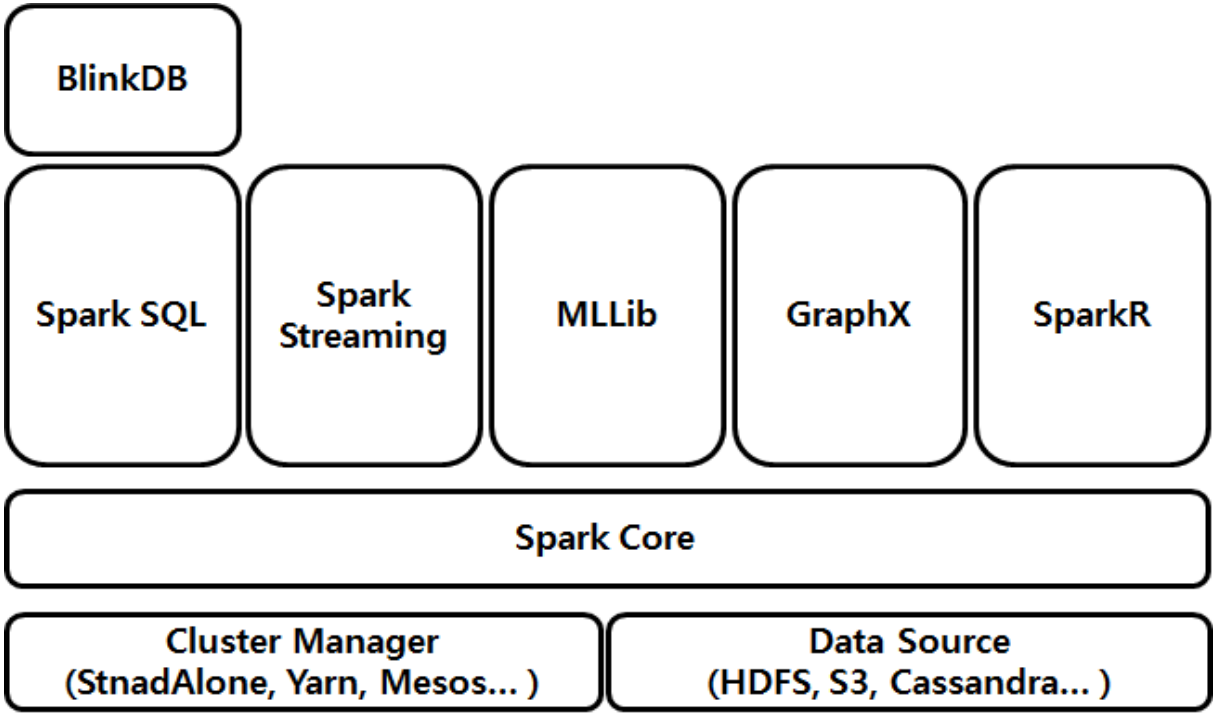
- Low latency (interactive) queries on historical data
  - 과거 데이터들을 빠르게 처리해 대화형 질의가 가능하도록 함.
  - Low Latency가 가능해야 데이터 탐색 - 분석 – 탐색 – 분석의 반복 과정을 통해 제대로 된 데이터 분석을 할 수 있음

- Low latency queries on live data(streaming)
  - 실시간으로 들어오는 데이터를 분석할 수 있어야 함.
  - 스파크는 실시간 스트리밍 처리·분석도 가능하도록 설계하였음.

설계 목표

- Sophisticated data processing
  - 복잡한 분석도 가능해야 함.
  - Anomaly detection, Trend analysis 등 복잡한 분석도 가능해야 좀 더 나은 의사결정을 할 수 있다고 생각했음.
  - 머하웃(Mahout)이나 R과 같은 프로젝트의 목표와 비슷하나 반복처리, 병렬처리에 훨씬 강력함.

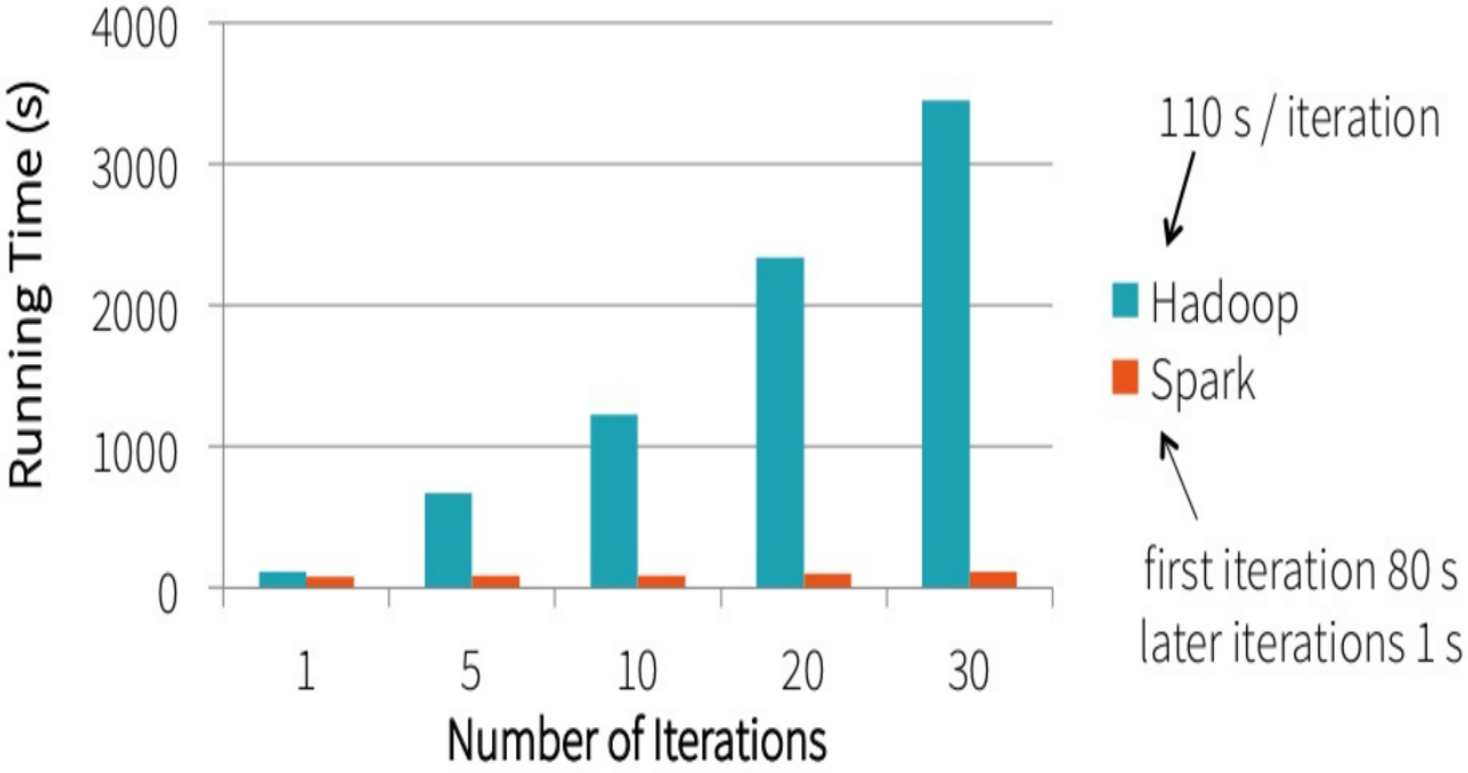
Unified Platform



Fast

	Hadoop World Record	Spark 100 TB *	Spark 1 PB
Data Size	102.5 TB	100 TB	1000 TB
Elapsed Time	72 mins	23 mins	234 mins
# Nodes	2100	206	190
# Cores	50400	6592	6080
# Reducers	10,000	29,000	250,000
Rate	1.42 TB/min	4.27 TB/min	4.27 TB/min
Rate/node	0.67 GB/min	20.7 GB/min	22.5 GB/min
Sort Benchmark Daytona Rules	Yes	Yes	No
Environment	dedicated data center	EC2 (i2.8xlarge)	EC2 (i2.8xlarge)

Fast


Running Time (s)
4000
3000
2000
1000
0
1
5
10
20
30
Number of Iterations
110 s / iteration
Hadoop
Spark
first iteration 80 s
later iterations 1 s

Simple

```
public class WordCount {

  public static class Map extends MapReduceBase implements Mapper<LongWritable, Text, Text, IntWritable> {
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(LongWritable key, Text value, OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException {
      String line = value.toString();
      StringTokenizer tokenizer = new StringTokenizer(line);
      while (tokenizer.hasMoreTokens()) {
        word.set(tokenizer.nextToken());
        output.collect(word, one);
      }
    }
  }

  public static class Reduce extends MapReduceBase implements Reducer<Text, IntWritable, Text, IntWritable> {
    public void reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text, IntWritable> output, Reporter reporter) throws
IOException {
      int sum = 0;
      while (values.ha
        sum += values
      }
      output.collect(
    }
  }

  public static void r
    JobConf conf = ne
    conf.setJobName("

    conf.setOutputKeyClass(Text.class);
    conf.setOutputValueClass(IntWritable.class);

    conf.setMapperClass(Map.class);
    conf.setCombinerClass(Reduce.class);
    conf.setReducerClass(Reduce.class);

    conf.setInputFormat(TextInputFormat.class);
    conf.setOutputFormat(TextOutputFormat.class);

    FileInputFormat.setInputPaths(conf, new Path(args[0]));
    FileOutputFormat.setOutputPath(conf, new Path(args[1]));

    JobClient.runJob(conf);
  }
}
```

```
val file = spark.textFile("hdfs://...")
val counts = file.flatMap(line => line.split(" "))
  .map(word => (word, 1))
  .reduceByKey(_ + _)
counts.saveAsTextFile("hdfs://...")
```

Word count in Spark(Scala)

지원 언어

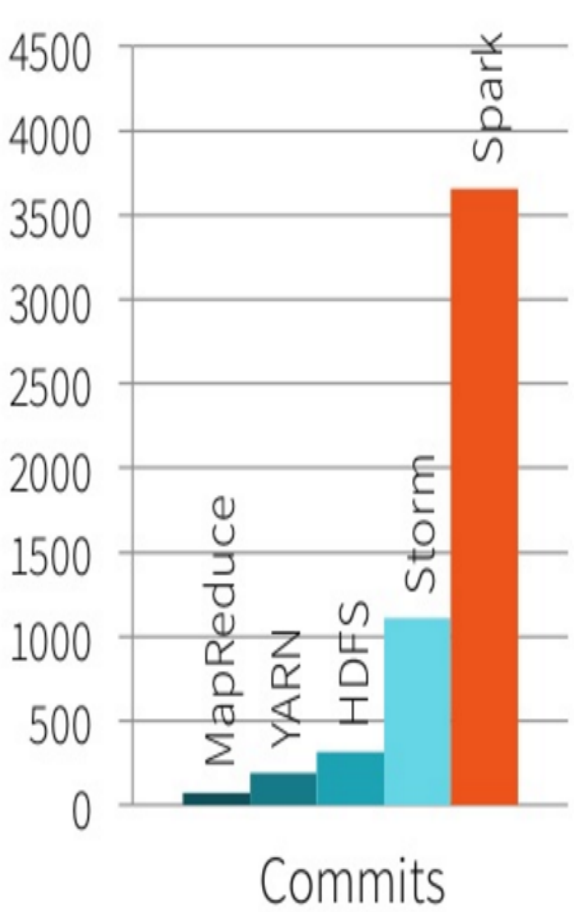
- 스파크는 상당부분이 스칼라(Scala)라는 객체지향 성격과 함수형 성격을 모두 가지는 언어로 프로그래밍 되어 있음.
- 스칼라만 지원하는 것은 아니라 기본적으로 스칼라와 더불어 자바, 파이썬을 지원함.
- 자바는 가장 범용적인 언어 중 하나이고 맵리듀스와 같이 많은 오픈소스들이 자바를 기반으로 프로그램을 만들 수 있도록 하고 있음.
- 또한 파이썬은 최근 간결성과 다양한 기능으로 사용자가 많아지고 있음

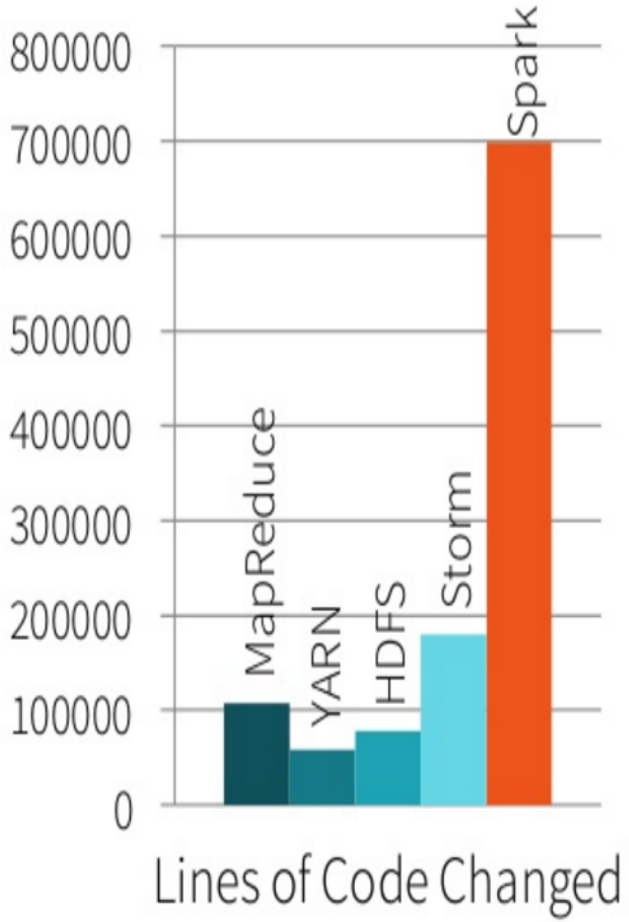
지원 언어

- 스파크에서 3가지 언어를 대부분 지원하지만, 모든 기능을 3가지 언어에 대해 동일하게 지원하지는 않음.
- 버전마다 다르지만 Spark SQL과의 연계, 스트리밍, MLlib의 각종 Matrix는 스칼라를 우선 지원
- 또한 셸 환경은 스칼라와 파이썬만 지원한다.
- 가급적이면 스칼라를 권장하고 자바나 파이썬을 사용할 경우, 사용하려는 기능을 제공하는지 미리 확인해야 해야함

빅데이터 에코시스템과 호환

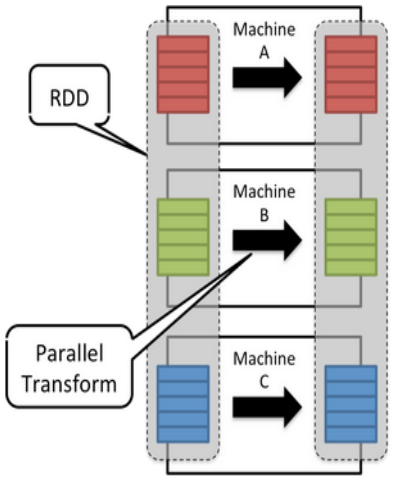
- Spark의 뛰어난 전략
- 하둡
  - 하둡 2.0 완벽한 호환
  - HDFS 및 하둡 에코시스템(Hbase, Casandra, Hive 등)과의 호환
  - Yarn과의 호환
- ETC
  - Amazon EC2
  - R
  - RDBMS
  - Tableau


4500
4000
3500
3000
2500
2000
1500
1000
500
0
MapReduce
YARN
HDFS
Storm
Spark
Commits


800000
700000
600000
500000
400000
300000
200000
100000
0
MapReduce
YARN
HDFS
Storm
Spark
Lines of Code Changed

How Fast?

- RDDs (Resilient Distributed Datasets)
  - 클러스터 전체에서 공유되는 데이터 형태로 대부분 메모리에 올라가 있음
  - 병렬로 처리될 수 있는 Immutable (read-only) , partitioned 된 elements 의 집합
    - 데이터를 수정할 수 있게되면 데이터 유실시 복구가
  - 대신 새로운 메모리를 확보하여 새로운 값 할당.
- Update 무시
- Cache

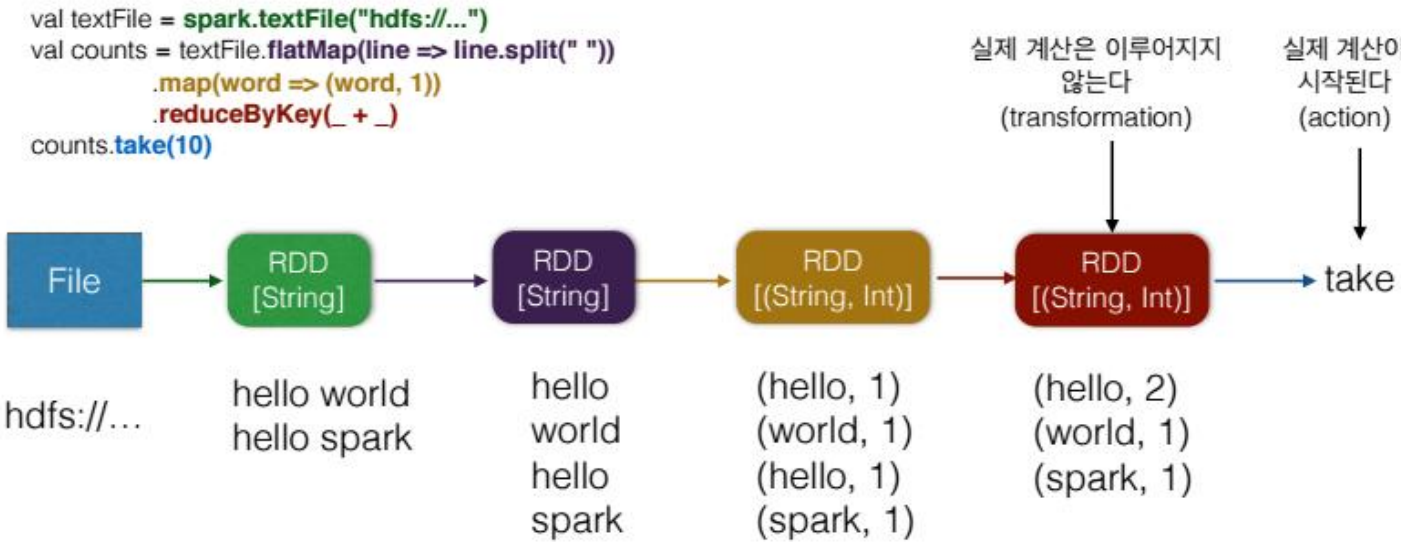


Fault Tolerance?

- RDDs (Resilient Distributed Datasets)
  - Fault Tolerance – Lineage를 이용한 데이터 복구
  - Need not exist in physical storage – RDDs는 메모리에 분산 임시저장하기 때문에 데이터 처리시 디스크를 사용하지 않음. 그러나, 데이터 복구시 매우 안정적인 저장공간으로부터 (ex> HDFS) 데이터를 복원하기 시작함.
  - Laziness : 모든 작업은 여러 작업을 설정해두고 마지막 Operation 함수 수행시 계산

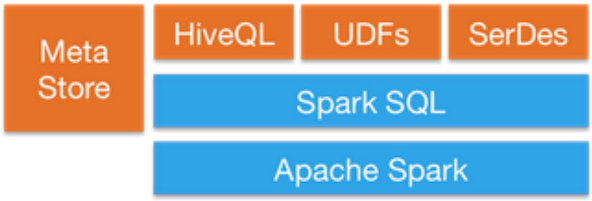
```
messages = textFile(...).filter(_.startswith("ERROR"))
                        .map(_.split('₩t')(2)
```

Spark 데이터 흐름

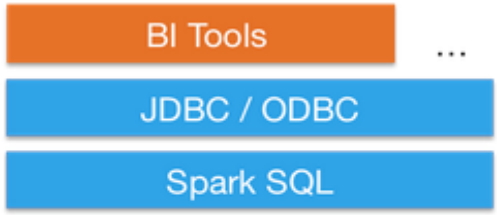

val textFile = spark.textFile("hdfs://...")
val counts = textFile.flatMap(line => line.split(" "))
.map(word => (word, 1))
.reduceByKey(_ + _)
counts.take(10)
실제 계산은 이루어지지 않는다
(transformation)
실제 계산이 시작된다
(action)
File
RDD [String]
RDD [String]
RDD [(String, Int)]
RDD [(String, Int)]
take
hdfs://…
hello world
hello spark
hello
world
hello
spark
(hello, 1)
(world, 1)
(hello, 1)
(spark, 1)
(hello, 2)
(world, 1)
(spark, 1)

Spark SQL Introduction

- 과거의 Shark (SQL on Spark) 는 개발 중단하고 SparkSQL로 프로젝트가 생성되었음
- Spark 프로그램과 SQL 쿼리를 혼합하여 Seamless 하게 사용 가능
- Hive 테이블, Parquet 파일, JSON 파일과 같은 여러 소스에서 Data Access 가능
- 기존의 Hive frontend 와 Metastore를 재사용 하여 기존의 Hive 데이터, 쿼리, UDFs 을 그대로 사용 가능
- JDBC 혹은 ODBC를 통해 서버모드를 포함하여 기존 BI Tool과의 연동도 가능
- DataFrame API(1.4), DataSet API(1.6)

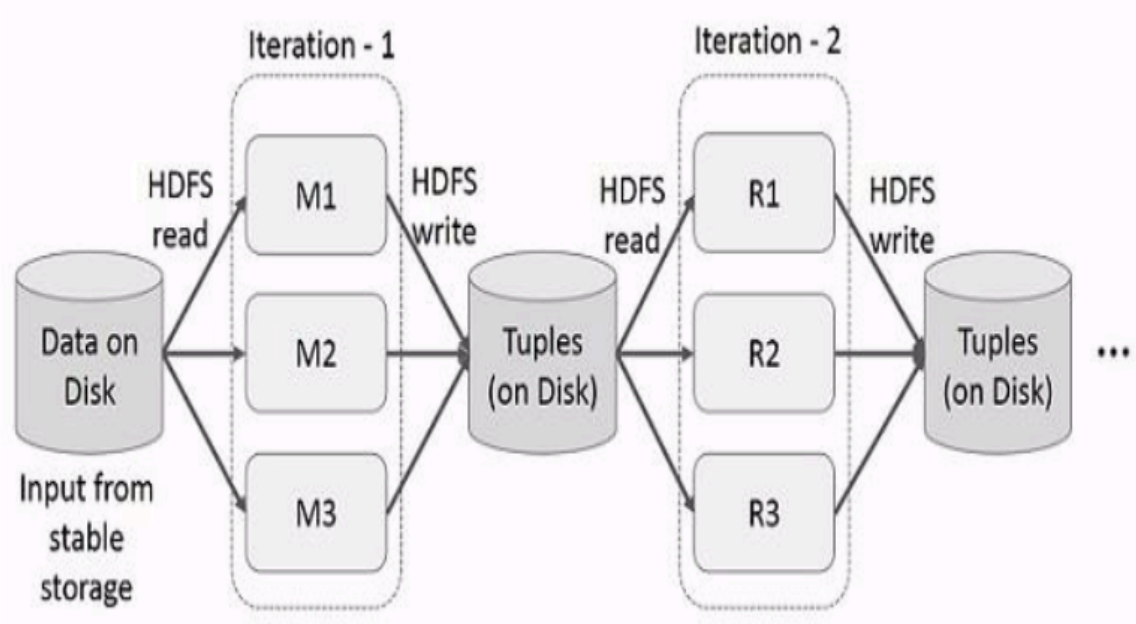


Spark SQL can use existing Hive metastores, SerDes, and UDFs.

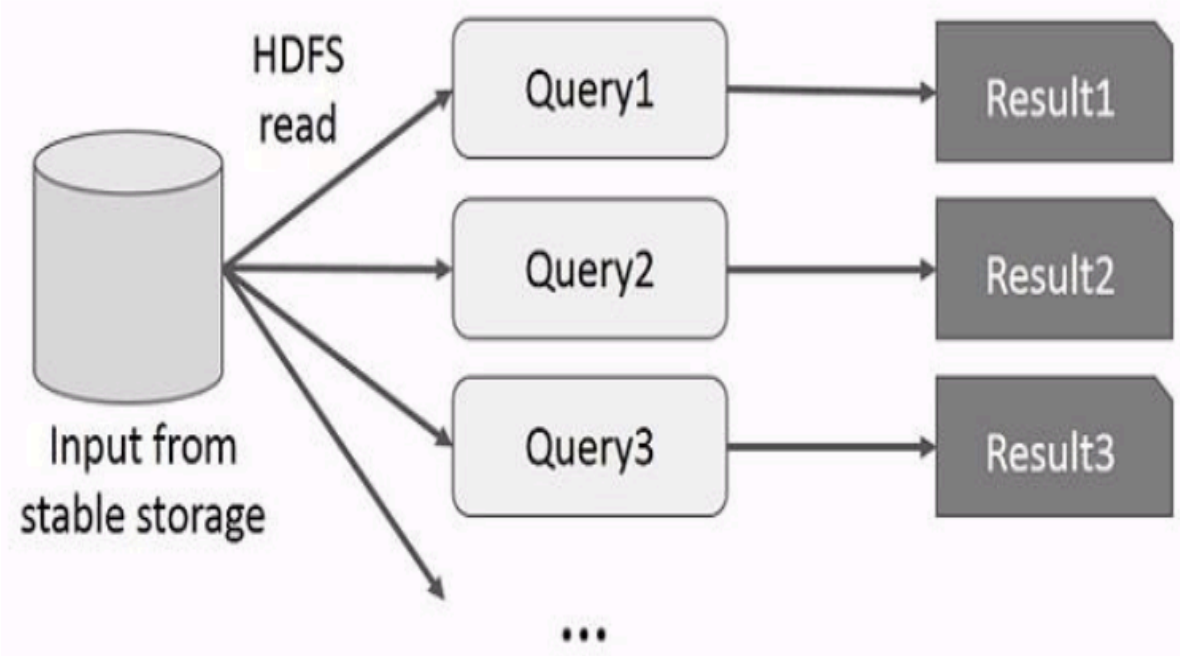


Use your existing BI tools to query big data.

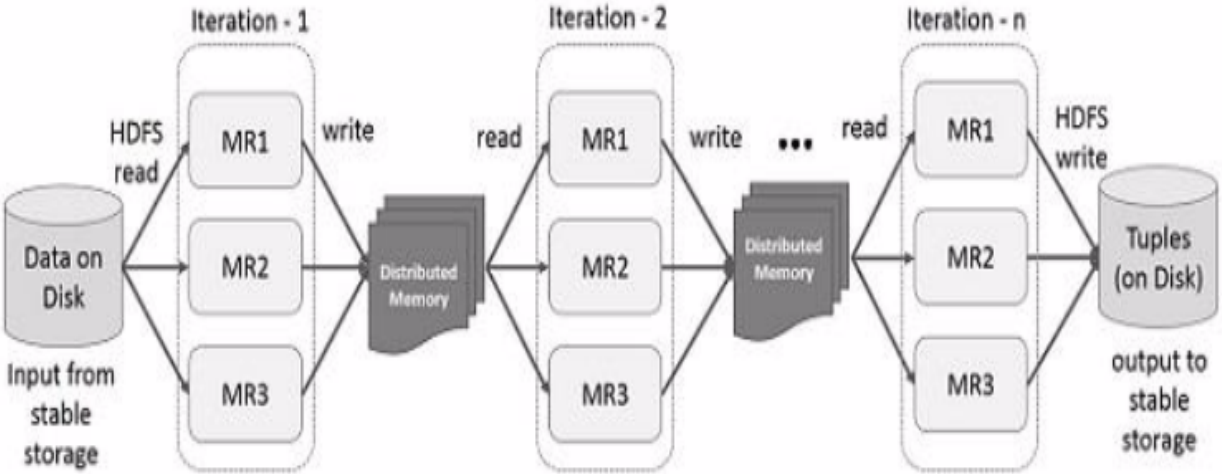
Iterative operations on MapReduce



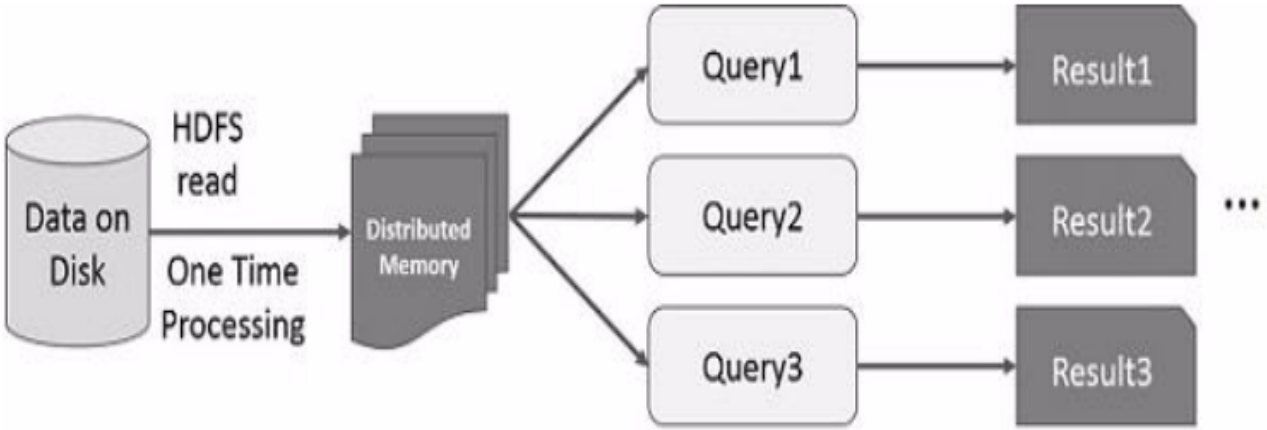
Interactive operations on MapReduce



Iterative operations on Spark RDD



Iterative operations on Spark RDD



스파크 Streaming

Spark Streaming

Flume

- 다양한 소스에서 발생한 대량의 로그 데이터를 중앙 데이터 스토어.
- 효과적으로 수집 집계(aggregating)하거나 이동시킬 수 있는 신뢰할수있는 분산 시스템.
- 스트림 지향의 데이터 플로우를 기반으로 하며 지정된 모든 서버로 부터 로그를 수집.
- 하둡 HDFS와 같은 중앙 저장소에 적재하여 분석하는 시스템을 구축해야 할 때 적합.
- 데이터 소스를 커스터마이징 할 수 있기 때문에 로그 데이터 수집에 제한되지 않음.
- 소셜미디어 데이터, 이메일 메세지등 다량의 이벤트 데이터를 전송하는데에 사용할 수 있음.

Kafka

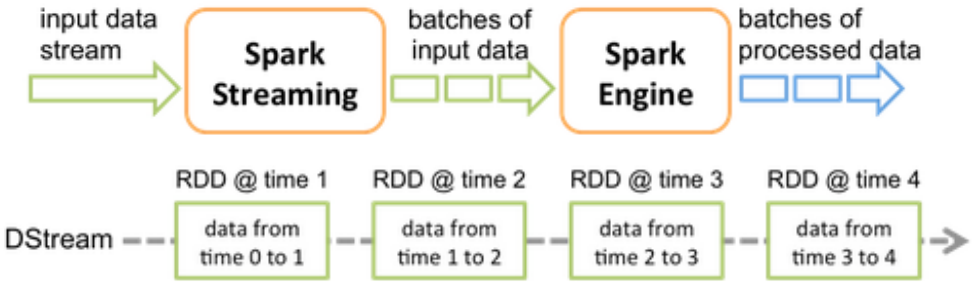
- LinkedIn에서 개발된 대용량 실시간처리를 위한 고성능 분산 메시징 시스템
- "큰 기업이 갖고 있을 모든 실시간 데이터 피드들을 처리하는 통합 플랫폼"
- 실시간 로그 집계와 같은 높은 볼륨의 이벤트 피드들을 위한 높은 처리량을 갖아야 함
- 오프라인 시스템으로부터 주기적인 데이터 로딩을 지원하기 위해, 많은 데이터 백로그들을 처리할 수 있어야 함
- 구식 메시징 use-case들을 처리하기 위해서, low-latency 전송을 처리할 수 있어야 함
- 새로운 피드나 유래된 피드들을 생성하기 위해 분할, 분산, 실시간 처리을 지원함
- 다른 시스템으로 스트림을 전송할 때에, 장비 장애의 fault-tolerance 보장

Spark Streaming Overview

- Scalable, High-throughput, Fault-tolerant stream processing을 가능하게 함.
- Kafka, Flume, Twitter, TCP sockets 등 여러 가지 소스를 사용할 수 있음.
- Map, Reduce, Join, Window 같은 High Level 기능들을 사용하여 Processing 할 수 있음.
- Process된 Data는 File system, Database 등에 저장될 수 있음.

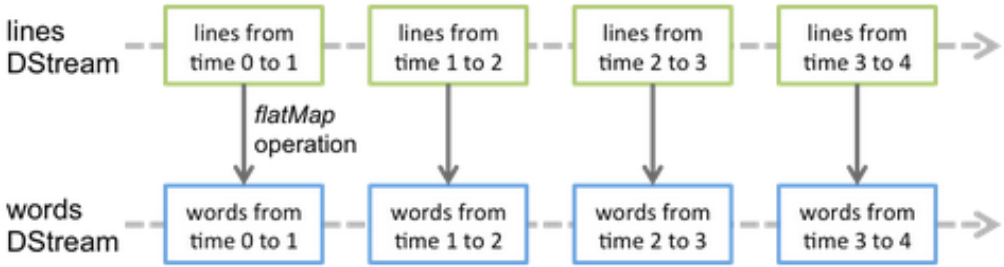
How does it work?

- 실시간으로 들어오는 data stream은 batch 단위로 나뉘어 지고 나뉘어진 batch 단위의 data는 Spark 엔진에 의해서 processing 된 뒤에 최종 final stream이 생성됨.
- Spark streaming은 Discretized stream 혹은 Dstream이라고 하는 High-level abstraction을 제공한다.
- DStream은 여러 input 소스에서부터 생성될 수 있음.
- DStream은 연속적인 RDD라고 볼 수 있음.
- DStream 내 RDD는 일정한 인터벌 시간 내 존재하는 Data가 들어있음.



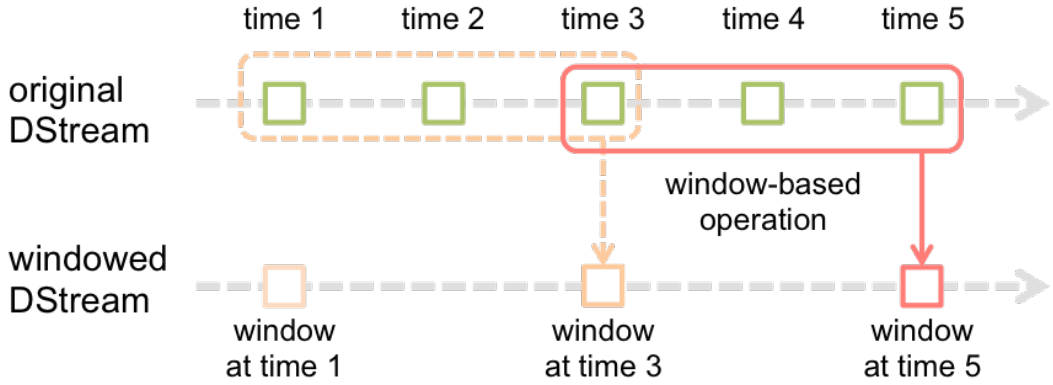
Spark Streaming 예제

- Line을 split을 통해 words로 바꿈.
  - val words = lines.flatMap(_.split(“ “))
- Wordcount를 DStream 의 API 인 map과 reduce를 통해서 수행.
  - val paris = words.map(word => (word, 1))
  - val wrodCounts = pairs.reduceByKey(_ + _)

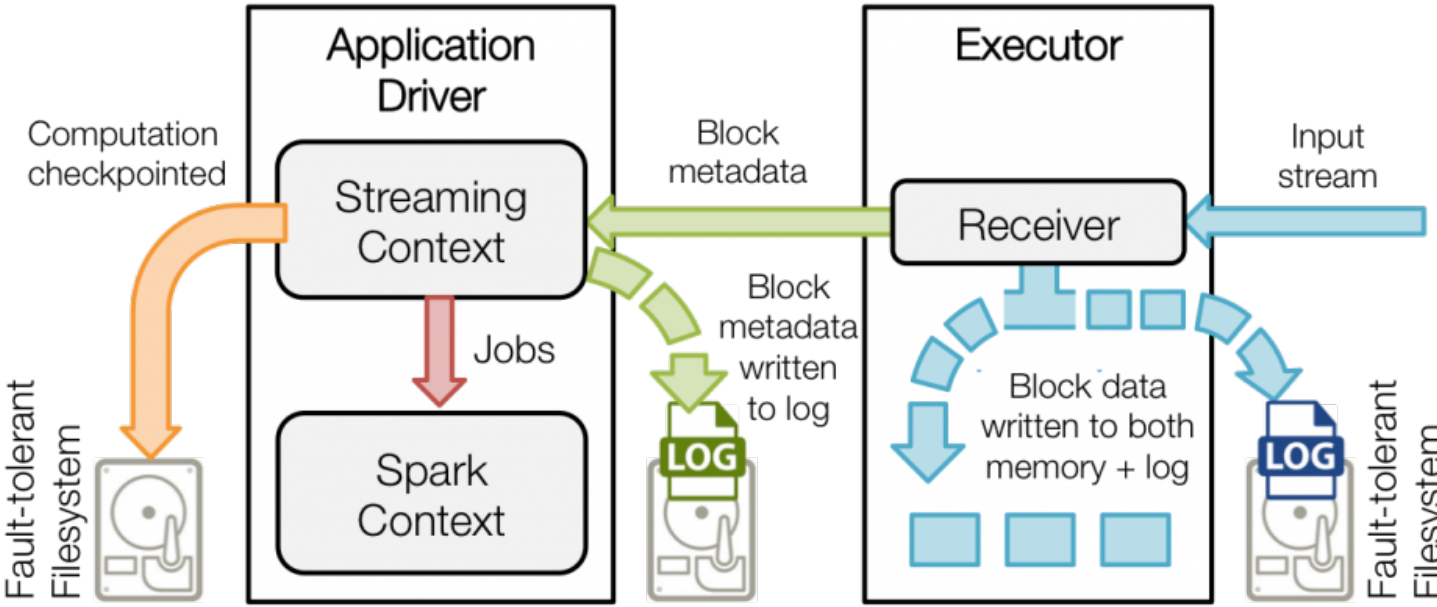


Window Operations

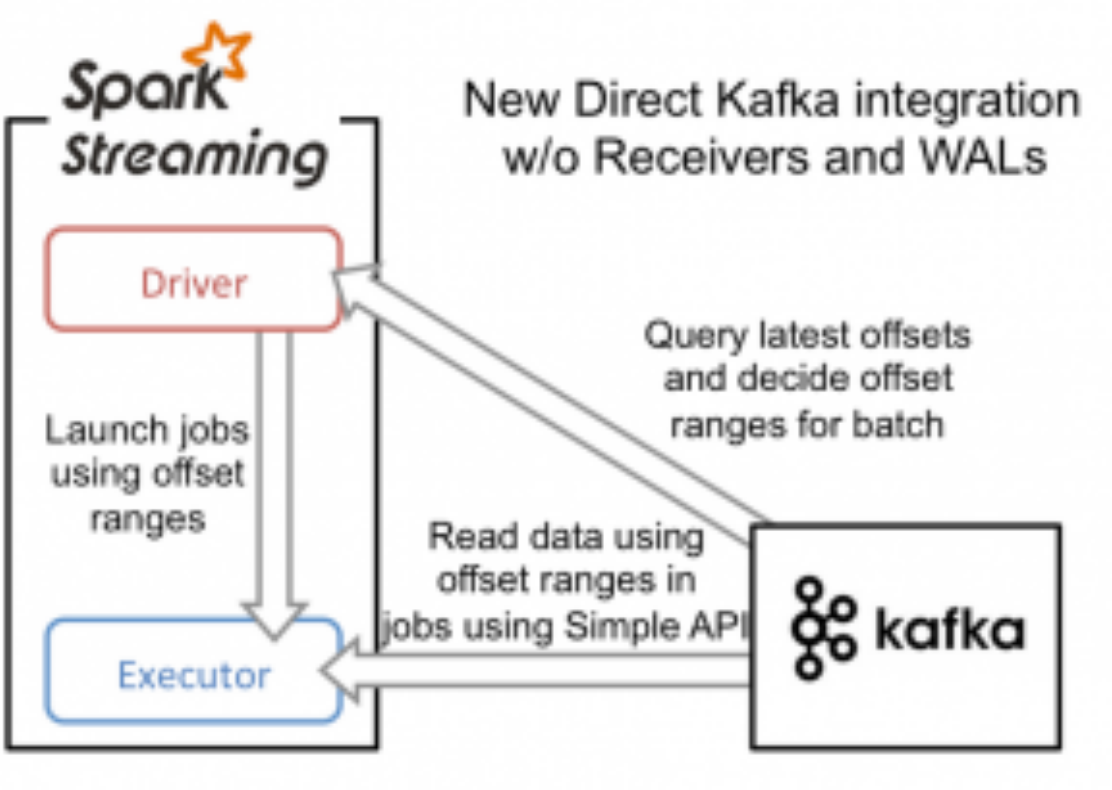
- Windowed computation 기능을 제공하는데 이것은 sliding window 내의 데이터를 transform 하기 위해서 임.
- Window-based operation을 수행하기 위해서는 2개의 파라미터가 필요.
  - Window length – window 사이즈
  - Slide interval – window-based operation이 수행되는 인터벌



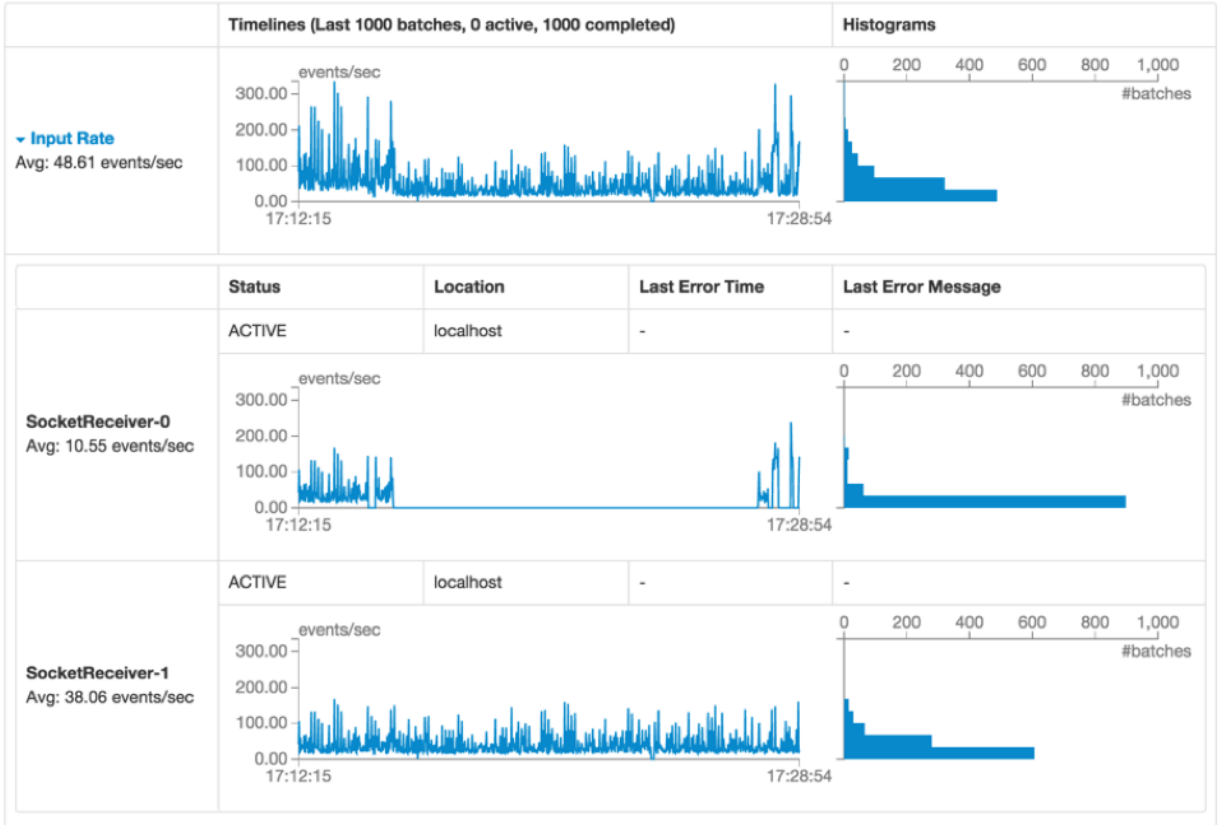
Fault-tolerance and Zero Data Loss



Improvements to Kafka integration



Visualizations
for Understanding Spark Streaming Applications

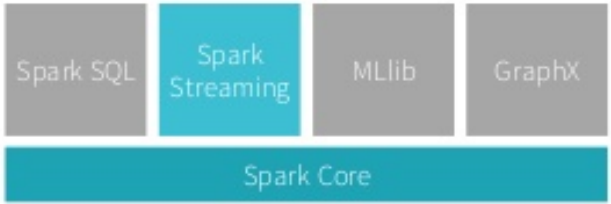


Combine batch

Join data streams with static data sets

```
// Create data set from Hadoop file
val dataset = sparkContext.hadoopFile("file")

// Join each batch in stream with the dataset
kafkaStream.transform { batchRDD =>
    batchRDD.join(dataset)
            .filter( ... )
}
```



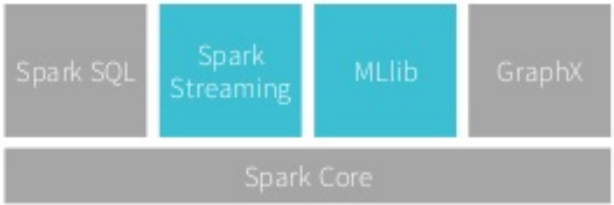
databricks

Combine machine learning

Learn models offline, apply them online

```
// Learn model offline
val model = KMeans.train(dataset, ...)

// Apply model online on stream
kafkaStream.map { event =>
  model.predict(event.feature)
}
```



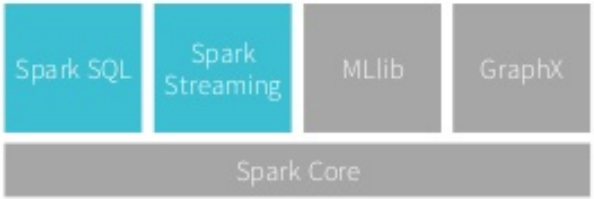
databricks

Combine SQL

Interactively query streaming data with SQL

```
// Register each batch in stream as table
kafkaStream.map { batchRDD =>
   batchRDD.registerTempTable("latestEvents")
}

// Interactively query table
sqlContext.sql("select * from latestEvents")
```



databricks

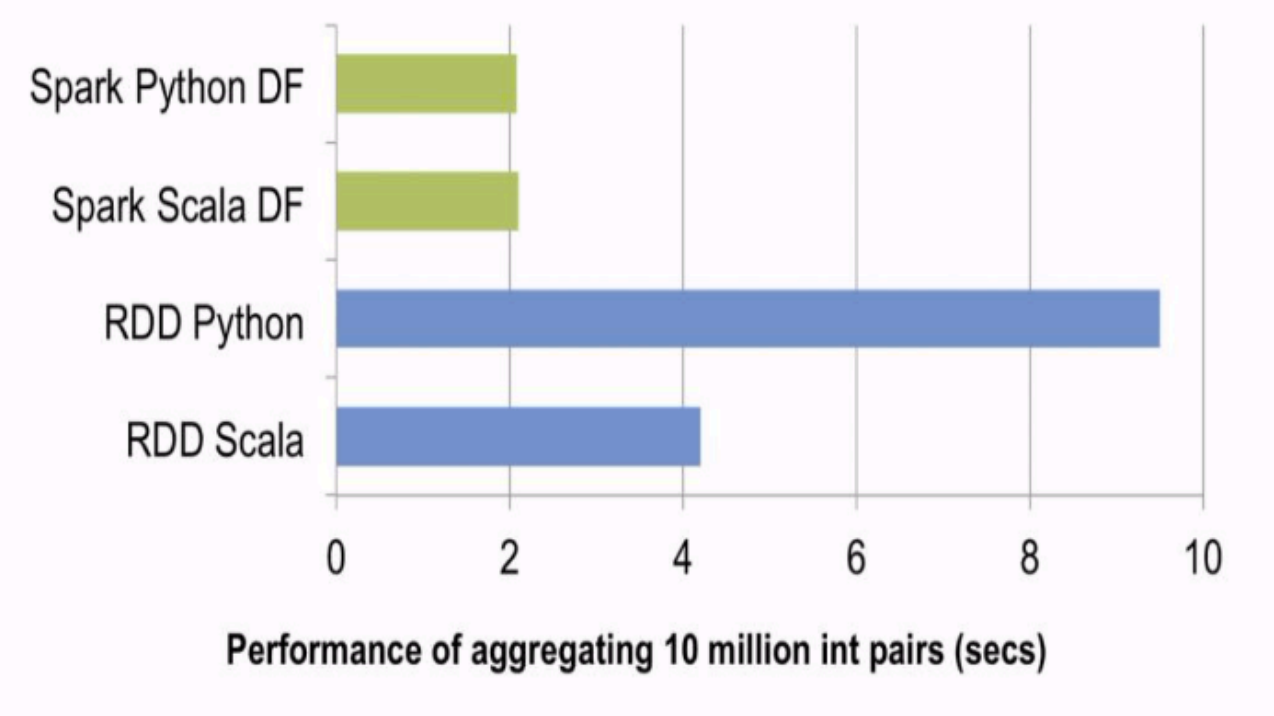
고급 분석

Tungsten execution engine

- Spark 성능 bottleneck은?
  - I/O나 network bandwidth?
  - 실제로 CPU 와 memory에서 더욱 bottleneck 발생!
- 하드웨어의 CPU, Memory 등에서도 최대한 뽑을 수 있는 새로운 아키텍쳐가 필요
- Project Tungsten
  - Memory Management and Binary Processing
  - Cache-aware computation
  - Code generation: using code generation to exploit modern compilers and CPUs
  - DataFrame(1.4), Dataset(1.6)

DataFrame

- 기술적인 개선으로 보이지만 분석을 위한 개선!



Spark 2.0

- Performance optimizations
- Custom encoders
- Python Support.
- Unification of DataFrames with Datasets
  - Static DataFrames -> Countinous DataFrames

MLlib, SparkML

- 보편적인 Machine Learning 알고리즘과 유틸리티를 Spark로 구현한 프로젝트
  - Goal is to make practical machine learning scalable and easy.

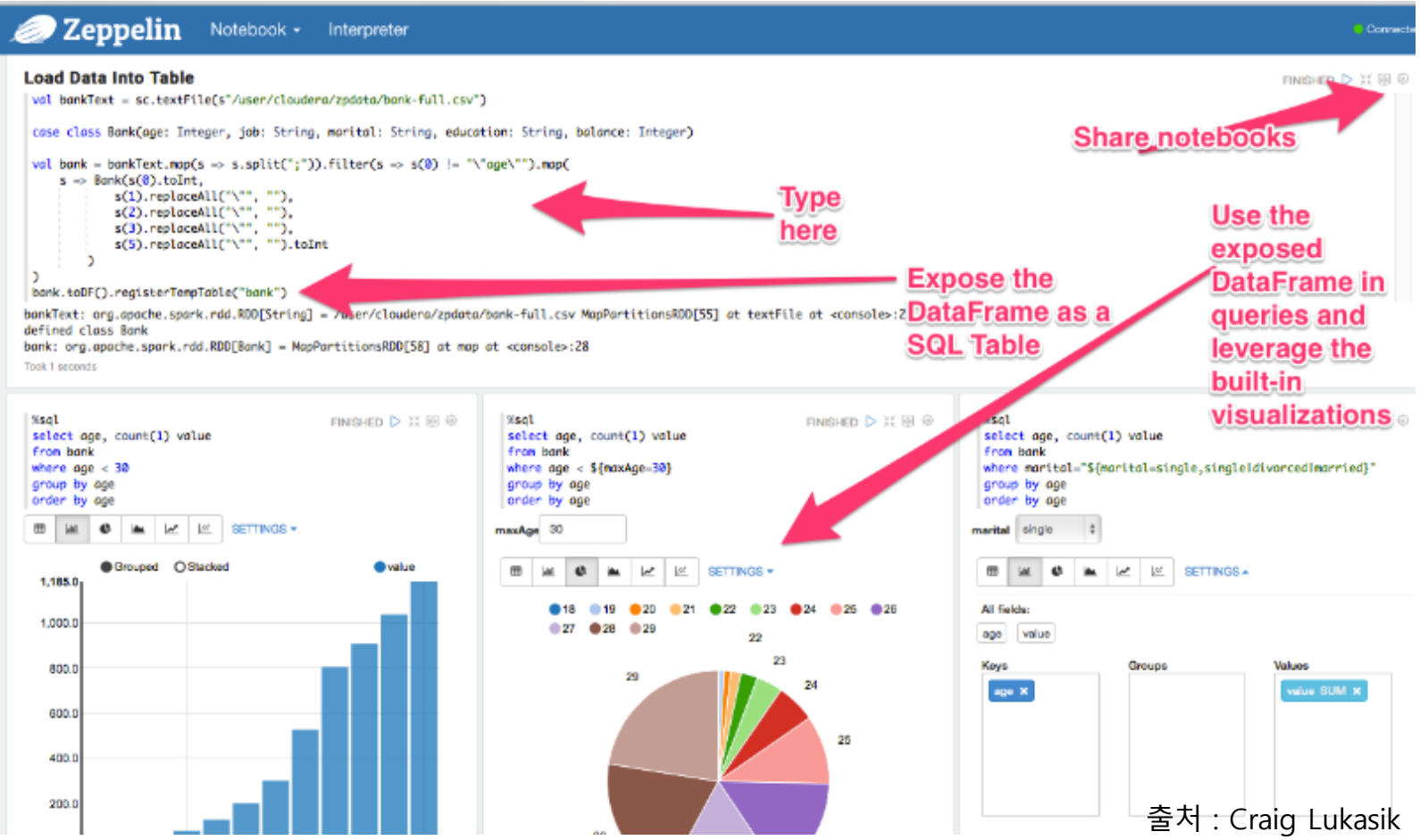
- 2가지 패키지
  - spark.mllib contains the original API built on top of RDDs.
  - spark.ml : provides higher level API built on top of DataFrames for constructing ML pipelines.

- 제공 내용
  - Classification and regression
  - Collaborative filtering
  - Clustering
  - Dimensionality reduction
  - Optimization

Zeppelin

Spark + 

Zeppelin



출처 : Craig Lukasik

R 분산처리 방법

- 데이터베이스 연결
- RHadoop
- SparkR
  - Spark 1.4 버전부터 정식으로 포함된 패키지

R의 한계 및 필요 기능

- 기본적으로 R은 단일 쓰레드를 사용하여 싱글코어, 싱글 머신에서 작동함
- 하드웨어에 따라 처리할 수 있는 데이터 크기가 한정되어 있음(주로 Ram 크기)
- DW 등 큰 데이터에 바로 접근해야 할 경우가 있음

R 분산처리 방법

- 유료
  - Revolution R Enterprise + AzureR
    - 가장 R 표준에 가까움
    - Azure의 클라우드 컴퓨팅을 활용할 수 있음
  - Oracle R Enterprise
    - R의 명령어를 그대로 사용하면서 오라클의 데이터에 접속할 수 있음
    - R언어의 함수는 오라클 내부에서 병렬실행이 되도록 질의로 변환됨
  - IBM Netezza, SAP HANA 등..

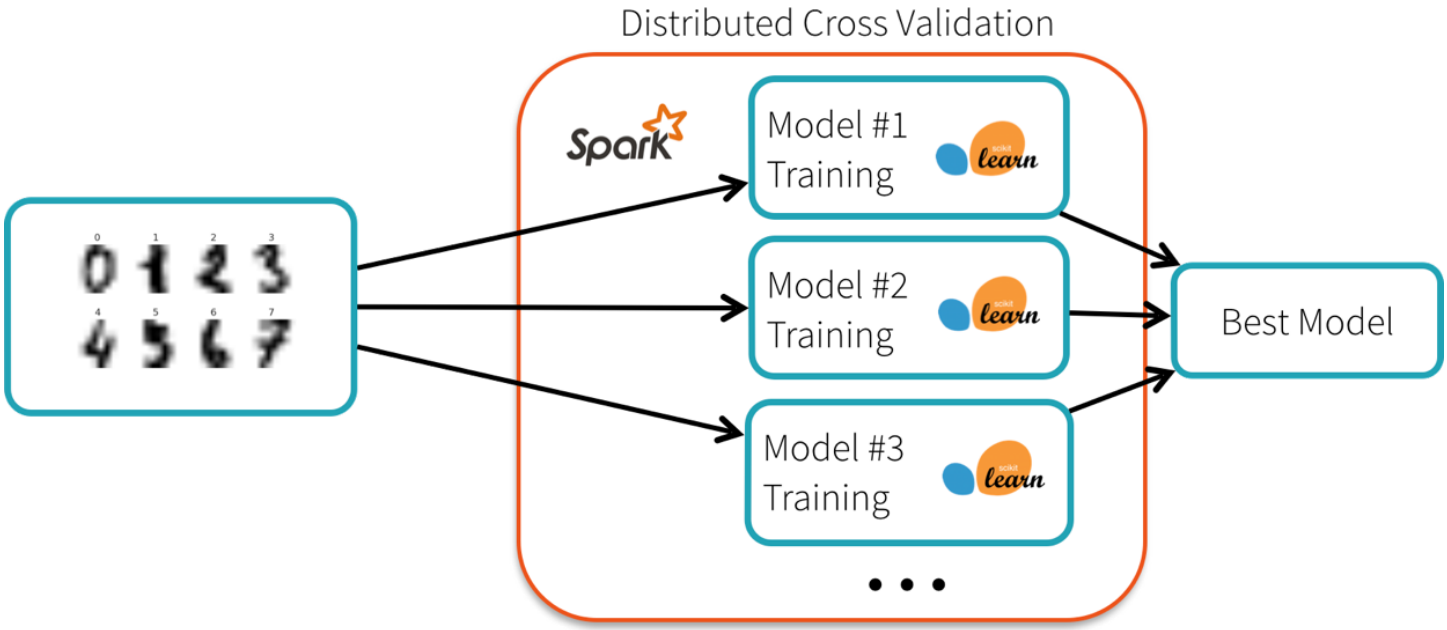
R 분산처리 방법

- Parallel 패키지
  - 내장된 멀티코어 패키지
  - 멀티 쓰레드 지원과 메모리를 해결할 수 있음
- Snow 패키지
  - 내장된 분산처리 패키지
  - 설정이 복잡하다는 단점이 있음

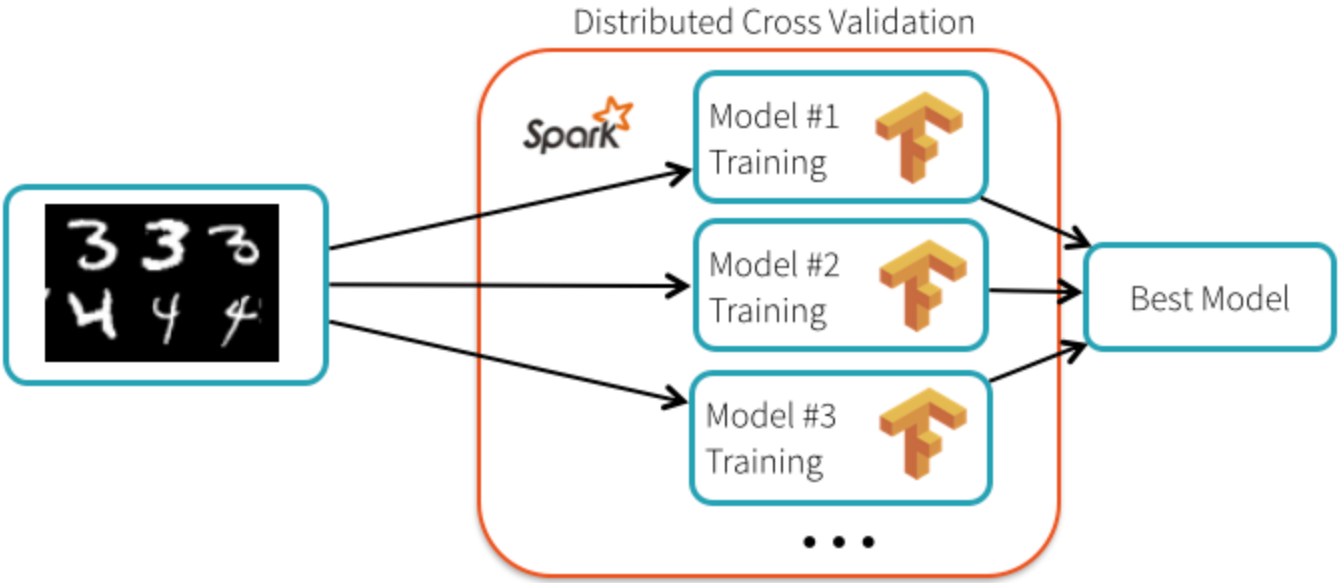
R 분산처리 방법

- 데이터베이스 연결
- RHadoop
- **SparkR**
  - Spark 1.4 버전부터 정식으로 포함된 패키지

Auto scaling scikit-learn with Spark



Deep Learning using Spark



학습 방법

- Databricks Blog
  - Spark를 만든 사람들이 창업한 회사(Databricks)
  - Spark 글들의 좋은 내용들이 다수 올라옴
  - "발표자료 그림의 상당수 출처는 Databricks Blog"
- Spark Summit
  - 대부분의 발표자료와 동영상 공유
- 책
  - 이미 과거 버전이라 자세한 기술보다는 기본 익히기는 좋음
  - Learning Spark, Advanced Analytics with Spark 등

Q&A