쌩초보도 이해할 수 있는 실전 Big Data 처리 기술

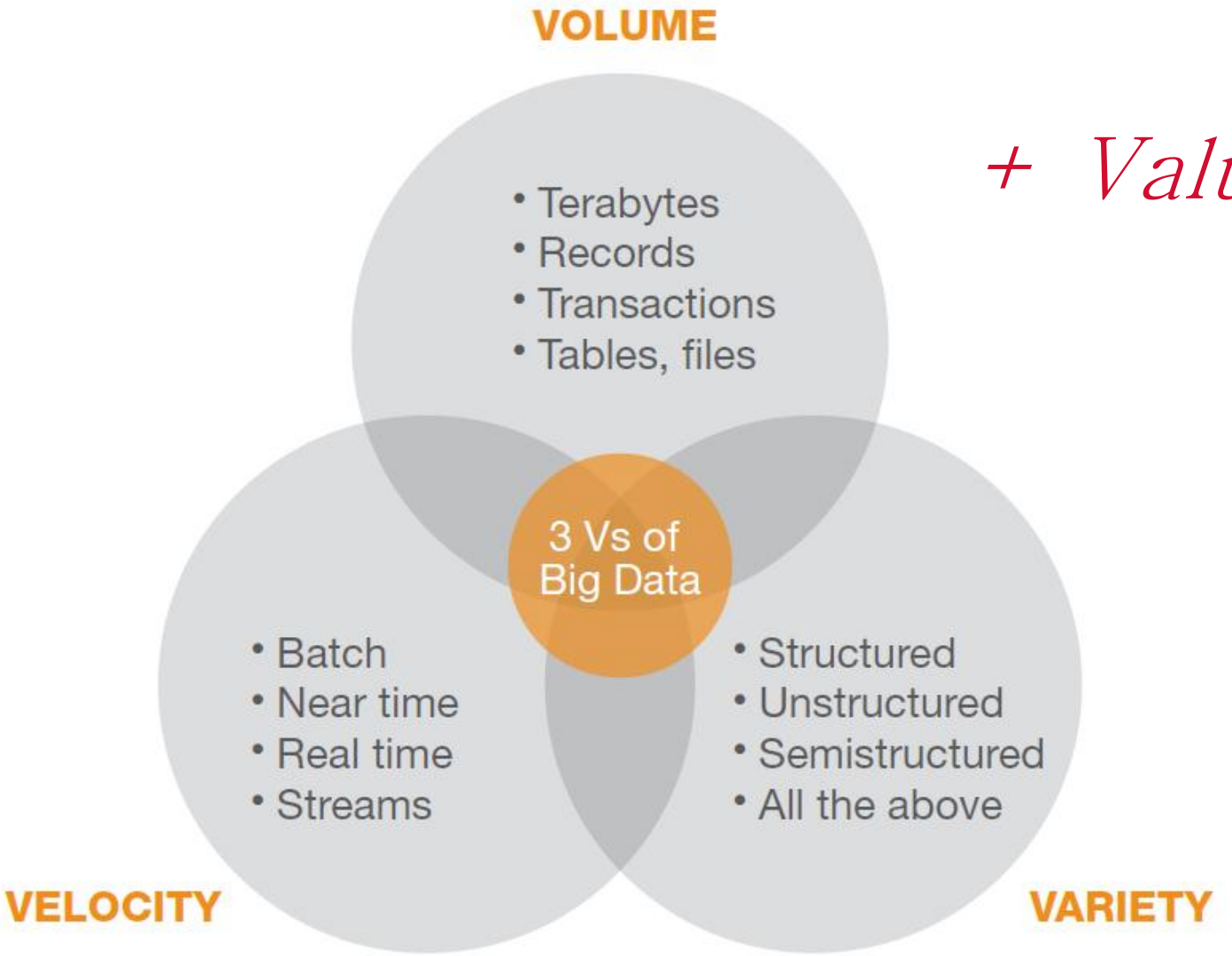
(주)클라우다인 대표 김병곤

ceo@cloudine.co.kr

소개

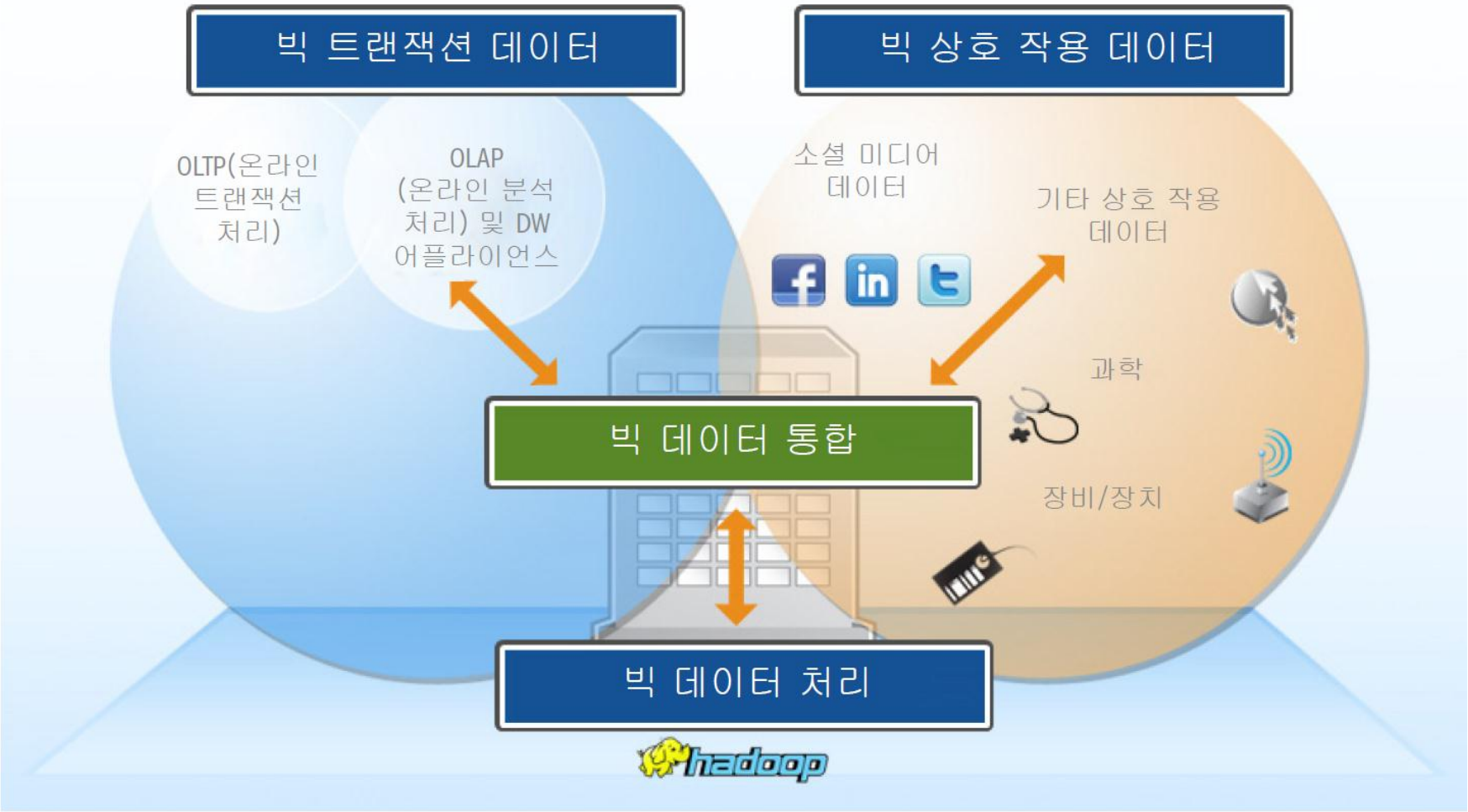
- (주)클라우다인 대표이사
- 한국자바개발자협의회(JCO) 회장
- JBoss User Group 대표
- 한국스마트개발자협회 부회장
- 지경부/NIPA 소프트웨어 마에스트로 멘토
- IT전문가협회 정회원
- 대용량 분산 컴퓨팅 Technical Architect
- 오프라인 Hadoop 교육 및 온라인 Java EE 교육
- 오픈 소스 Open Flamingo 설립(http://www.openflamingo.org)
- Java Application Performance Tuning 전문가
- 다수 책 집필 및 번역
  - JBoss Application Server5, Enterprise JavaBeans 2/3

Big Data의 4가지 속성



+ *Value*

Big Data = Big Opportunity



Gartner, "CEO Advisory: 'Big Data' Equals Big Opportunity," 2011년 3월 31일

Big Data 활용 분야

적용분야	활 용
공공	- **U-City, USN 데이터의** 수집, 분석, 활용 - 환경, 방재, 국방, 기상 등 대용량 데이터 분석 기반의 시스템
금융/통신	- SNS 및 관련 서비스 - **N-Screen Service** - Card사의 결제정보, 로그정보 기반 **개인화 마케팅**
제조 및 일반기업	- Smart TV/Mobile AppStore 등 **제품 기반 B2C 서비스** - 제조 장비 운전 데이터 수집, **분석(SPC), 제어, 모니터링 시스템** - **대용량 EAI, B2Bi 구축** - **BI 2.0**, CRM, ERM, ERP 등의 **의사결정 지원도구 시스템** - 통계 데이터 기반 각종 시뮬레이션 및 예측 시스템
기타	- 인터넷 쇼핑몰의 사용자 패턴 정보 분석 및 타겟 마케팅 - 온라인 게임, 연말정산 등의 일시적 G2C 서비스 등등

Big Data의 세계

- 우리는 Big Data속에서 살고 있다!
  - 뉴욕증권거래소 : 1일에 1테라 바이트의 거래 데이터생성
  - Facebook : 100억장의 사진, 수 페타바이트의 스토리지
  - 통신사 : 시간당 10G 이상의 통화 데이터, 1일240G 생성, 월 생성 데이터의 크기 200T 이상

우리는 엄청나게 큰 로그 데이터를 어떻게 처리해야 할까?

Hadoop의 패러다임의 전환

로직이 데이터에 접근하지 말고

데이터가 있는 곳에 로직을 옮겨라!

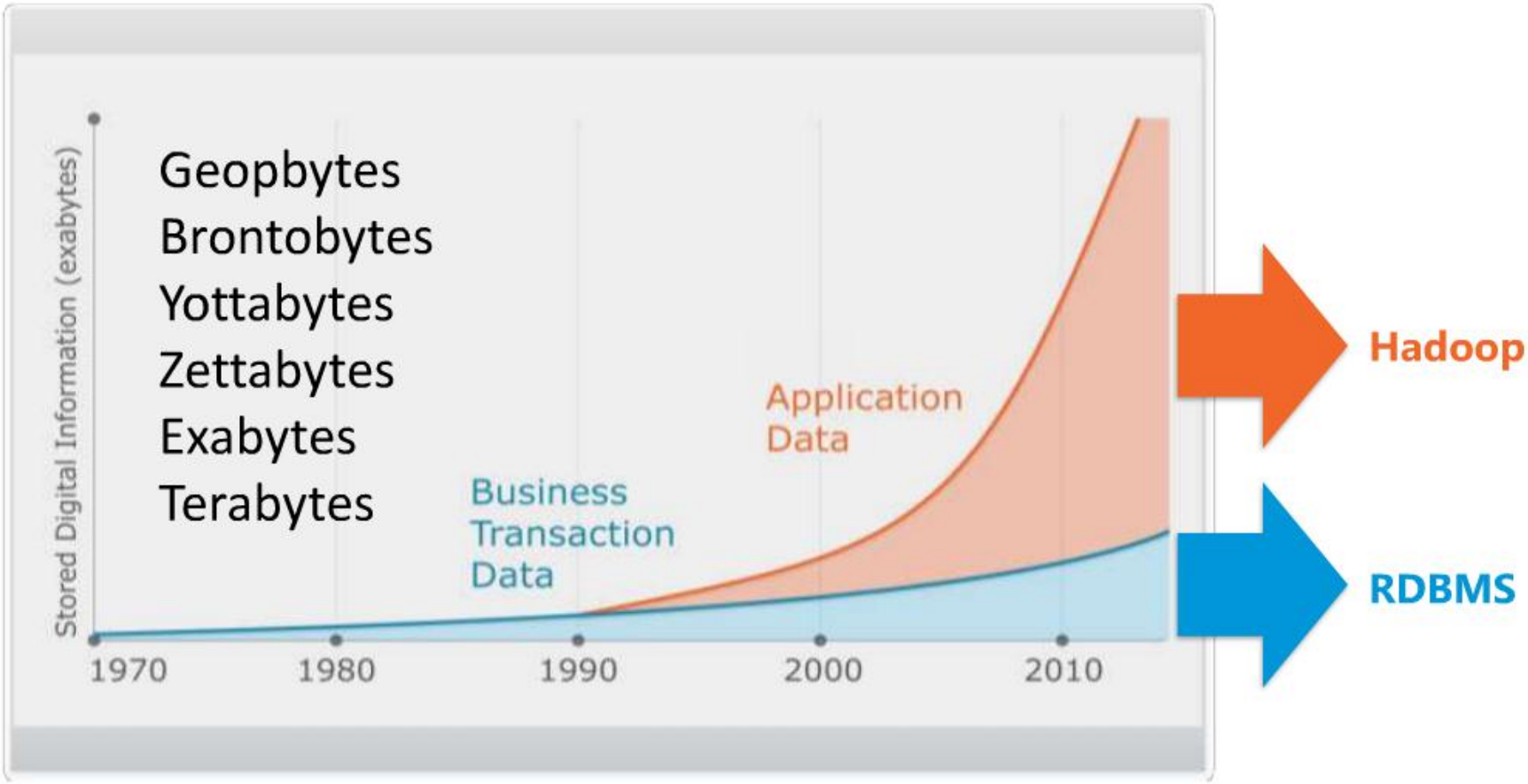
왜 대용량에 Apache Hadoop이 적합한가?

- 애플리케이션/트랜잭션 로그 정보는 매우 크다.
  - 대용량 파일을 저장할 수 있는 분산 파일 시스템을 제공한다.
- I/O 집중적이면서 CPU도 많이 사용한다.
  - 멀티 노드로 부하를 분산시켜 처리한다.
- 데이터베이스는 하드웨어 추가 시 성능 향상이 linear하지 않다.
  - 장비를 증가시킬 수록 성능이 linear에 가깝게 향상된다.
- 데이터베이스는 소프트웨어와 하드웨어가 비싸다.
  - Apache Hadoop은 무료이다.
  - Intel Core 머신과 리눅스는 싸다.

Hadoop의 다양한 응용 분야

- ETL(Extract, Transform, Load)
- Data Warehouse
- Storage for Log Aggregator
- Distributed Data Storage (예; CDN)
- Spam Filtering
- Biometric
- Online Content Optimization
- Parallel Image, Movie Clip Processing
- Machine Learning
- Science
- Search Engine

데이터 처리에 있어서 Hadoop, RDMBS의 위치



데이터베이스와 Hadoop 비교

	Traditional RDBMS	MapReduce
Data size	Gigabytes	Petabytes
Access	Interactive and batch	Batch
Updates	Read and write many times	Write once, read many times
Structure	Static schema	Dynamic schema
Integrity	High	Low
Scaling	Nonlinear	Linear

국내/해외 Hadoop Business 상황

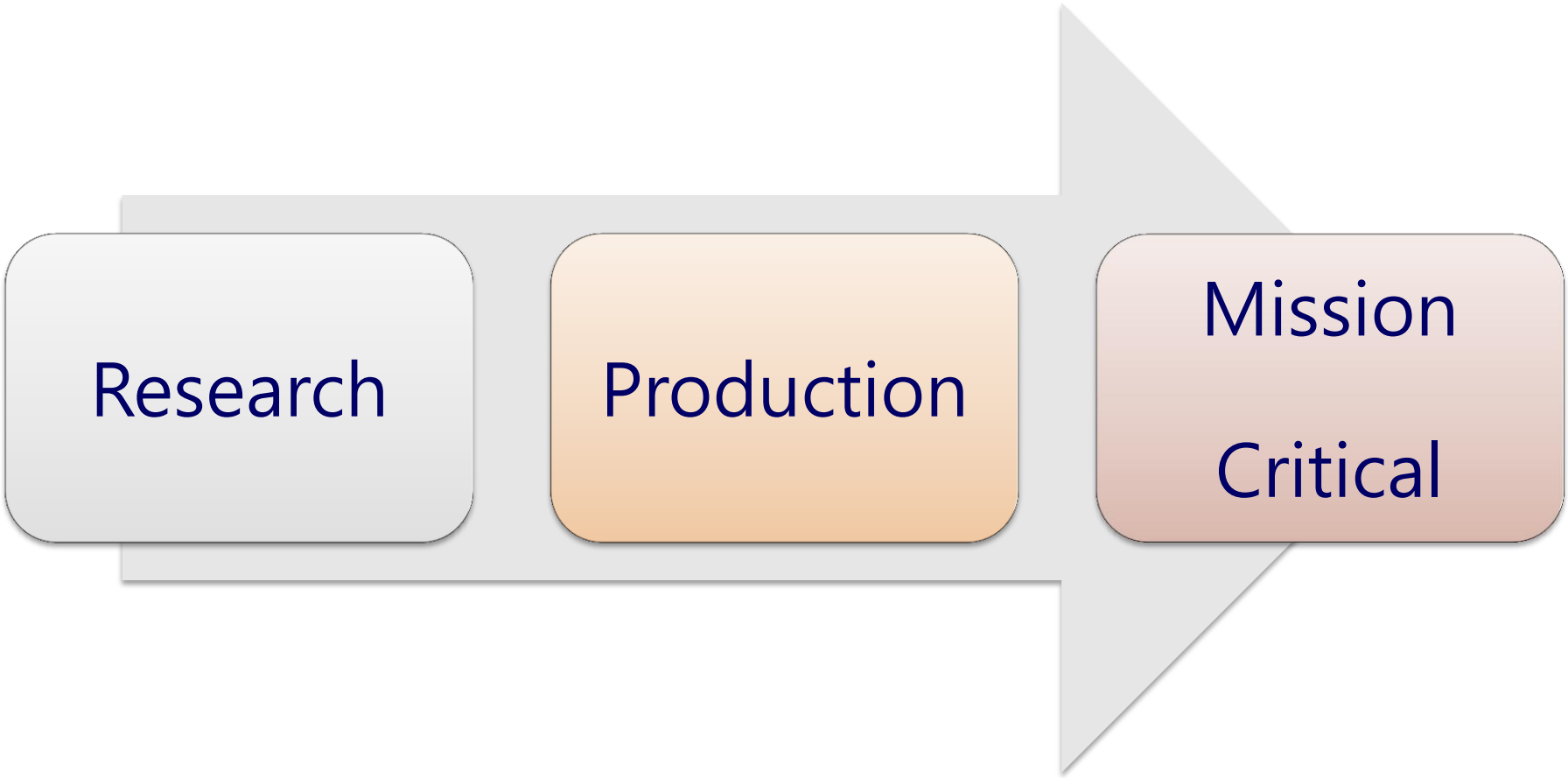
해외

Ready for Business

국내

Research, Ready for Business (일부)

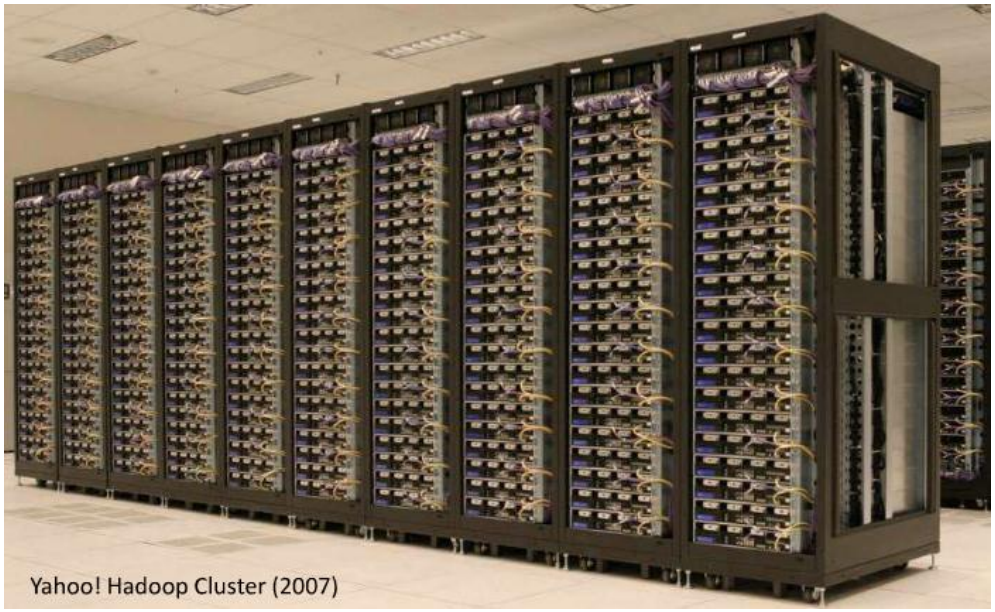
일반적인 Hadoop 적용 단계



복잡하면서 크리티컬한 비즈니스 문제

- Modeling true risk
- Customer churn analysis
- Recommendation engine
- Ad targeting
- PoS transaction analysis
- Analyzing network data to predict failure
- Threat analysis
- Trade surveillance
- Search quality
- Data “sandbox”

Hadoop Cluster



Yahoo! Hadoop Cluster (2007)

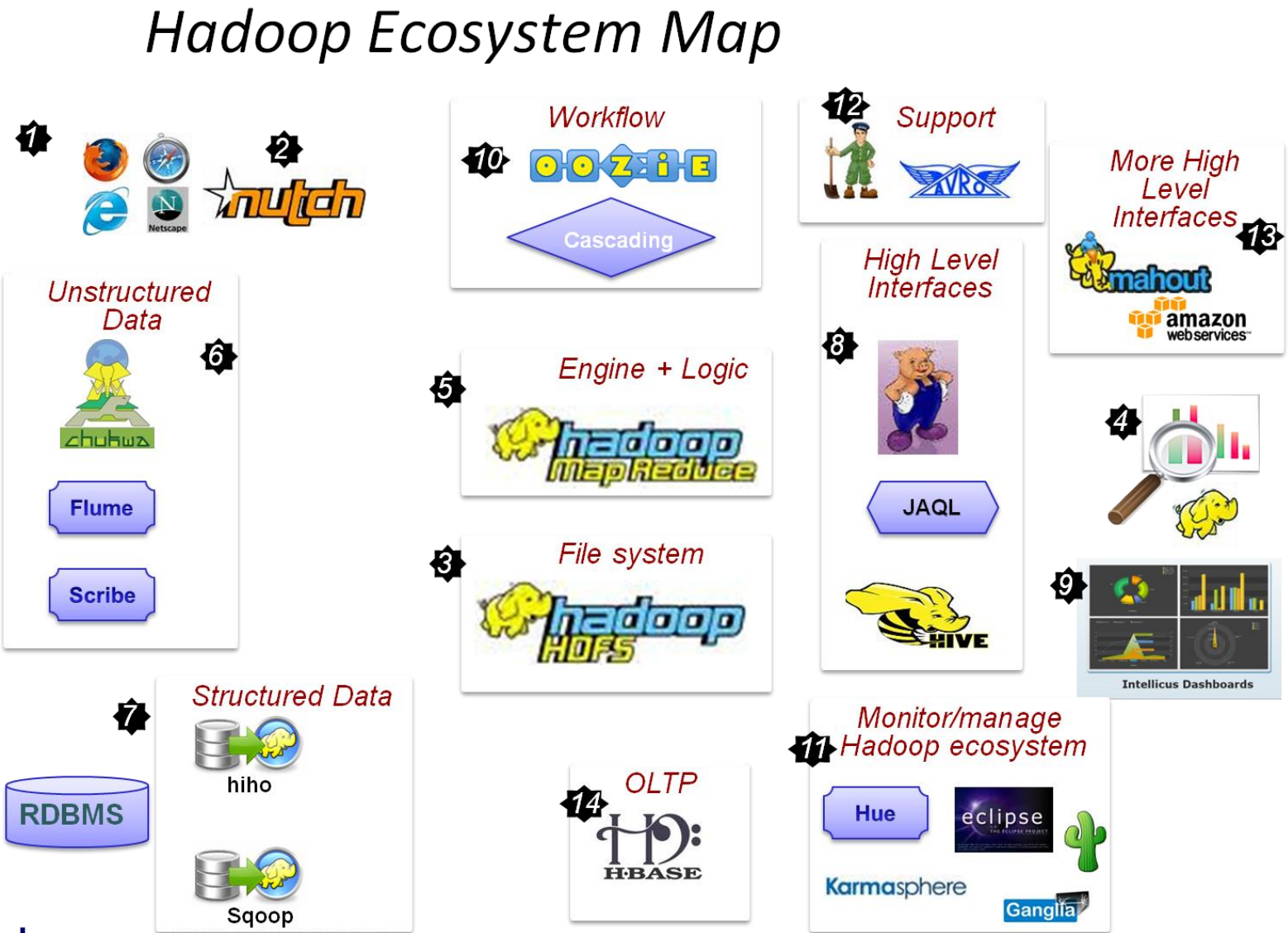
amazon web services™

ucloud 유클라우드

Hadoop Cluster를 구성하는 노드의 시스템 스펙

- 2 CPU(4 Core Per CPU) Xeons 2.5GHz 이상
- 4x1TB SATA
- 16G RAM
- 1G 이더넷
- 10G 스위치
- 랙당 20대의 노드
- Ubuntu Linux Server or CentOS 64bit
- Sun Java SDK 1.6
- Apache Hadoop 0.20.2

Hadoop Echosystem



Hadoop 배포판


hadoop


facebook

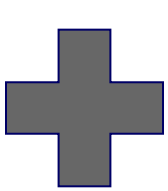

cloudera


IBM


YAHOO!


MAPR
TECHNOLOGIES
EASY. DEPENDABLE. FAST.

R & Hadoop



Statistical Computing

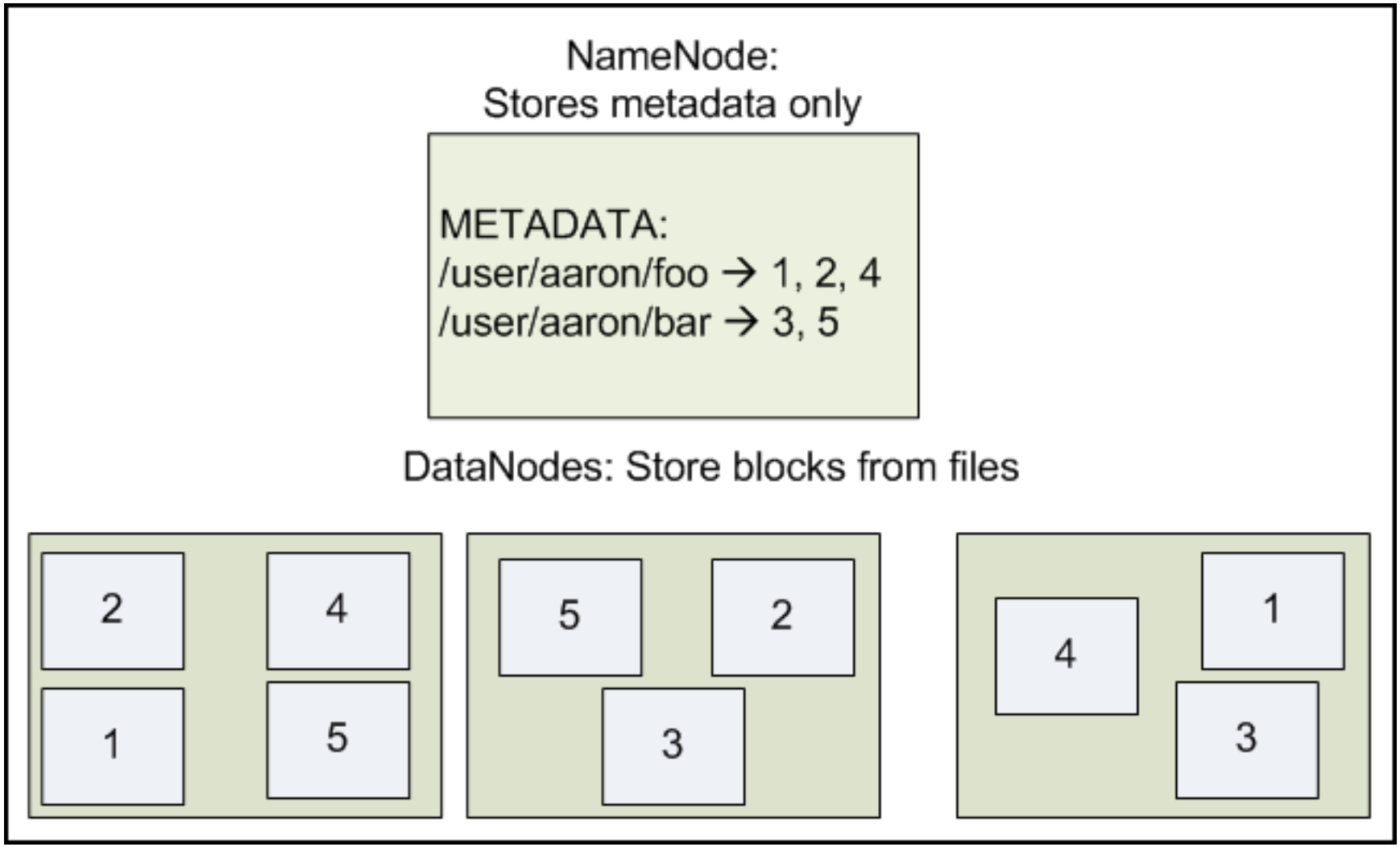
MapReduce

Apache Hadoop 기초

- **File System** : HDFS(Hadoop Distributed File System)
  - 파일을 64M 단위로 나누어 장비에 나누어서 저장하는 방식
  - 사용자는 하나의 파일로 보이나 실제로는 나누어져 있음
  - 2003년 Google이 논문으로 Google File System을 발표

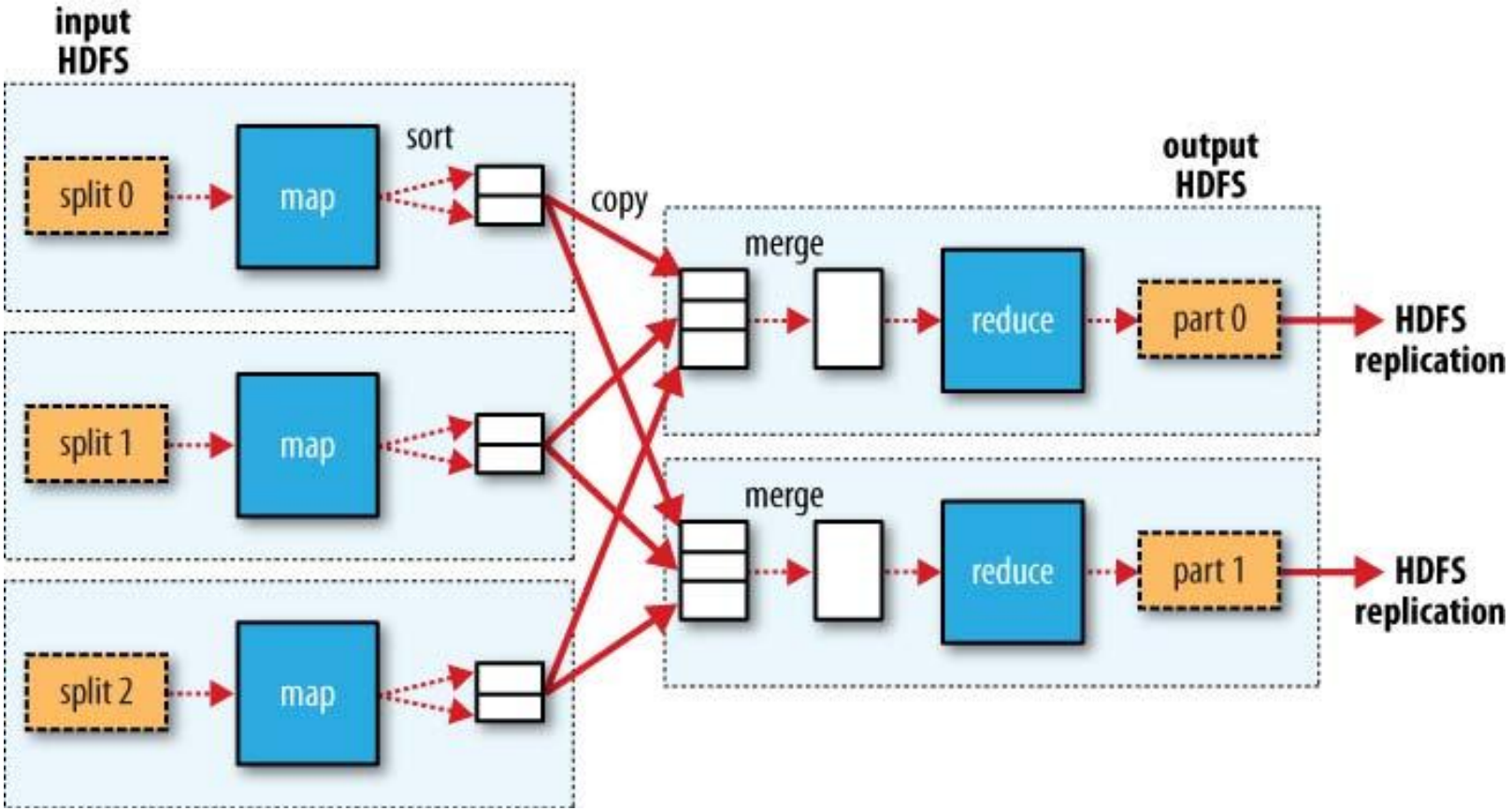
- **프로그래밍 모델**(MapReduce) (2004년 Google이 논문 발표)
  - HDFS의 파일을 이용하여 처리하는 방법을 제공
  - Parallelization, Distribution, Fault-Tolerance …

파일 시스템 : HDFS


NameNode:
Stores metadata only
METADATA:
/user/aaron/foo → 1, 2, 4
/user/aaron/bar → 3, 5
DataNodes: Store blocks from files
2
4
1
5
5
2
3
4
1
3

프로그래밍 모델 : MapReduce

- HDFS의 파일을 처리하기 위한 프로그래밍 모델

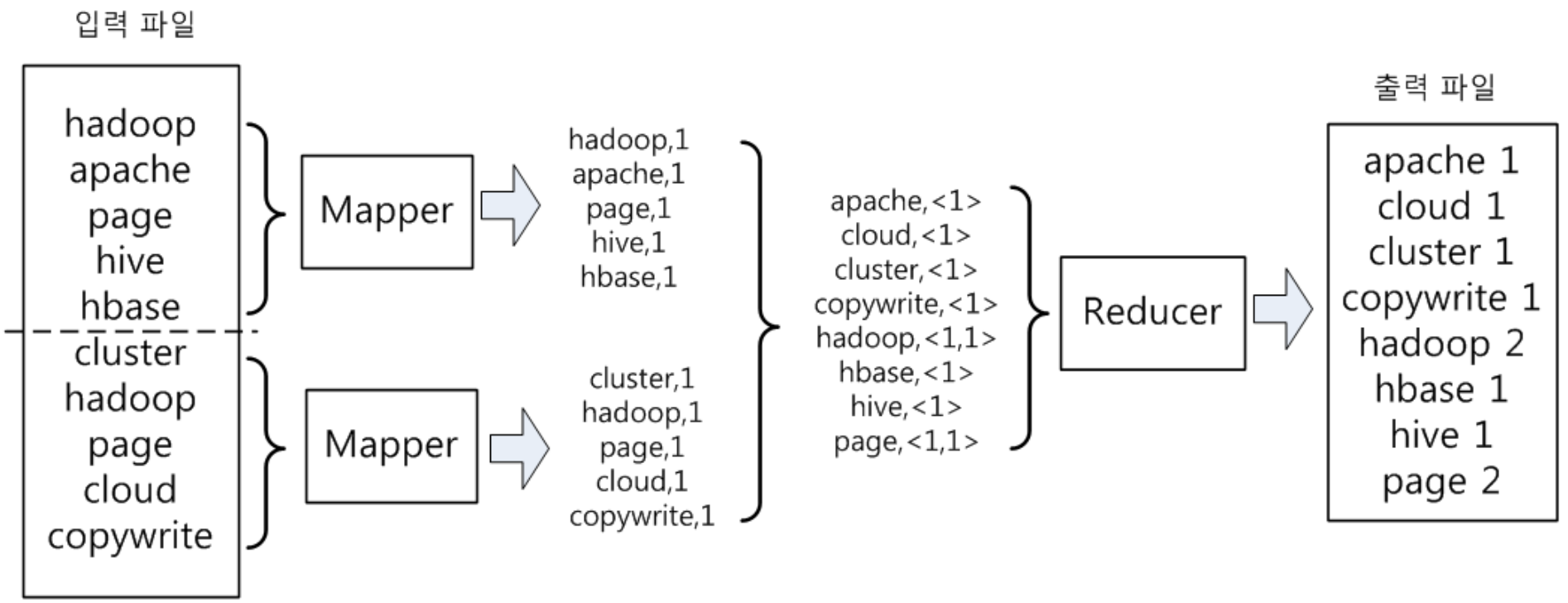


WordCount

- Hadoop의 MapReduce Framework 동작을 이해하는 핵심 예제
- 각각의 ROW에 하나의 Word가 있을 때 Word의 개수를 알아내는 예제

입력 파일(Mapper의 Input)	출력 파일(Reduce Output)
hadoop apache page hive hbase cluster hadoop page cloud copywrite	apache 1 cloud 1 cluster 1 copywrite 1 hadoop 2 hbase 1 hive 1 page 2

WordCount


입력 파일
hadoop
apache
page
hive
hbase
cluster
hadoop
page
cloud
copywrite
Mapper
Mapper
hadoop,1
apache,1
page,1
hive,1
hbase,1
cluster,1
hadoop,1
page,1
cloud,1
copywrite,1
apache,<1>
cloud,<1>
cluster,<1>
copywrite,<1>
hadoop,<1,1>
hbase,<1>
hive,<1>
page,<1,1>
Reducer
출력 파일
apache 1
cloud 1
cluster 1
copywrite 1
hadoop 2
hbase 1
hive 1
page 2

WordCount의 Mapper

- 파일의 1ROW를 읽어서 Word, 1을 출력하는 Mapper

```
public class WordCountMapper
                extends Mapper<Object, Text, Text, IntWritable> {

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

  public void map(Object key, Text word, Context context)
                      throws IOException, InterruptedException {
    context.write(word, one);
  }

}
```

WordCount의 Reducer

- Mapper의 출력 중에서 같은 Key로 묶어서 처리하는 Reducer

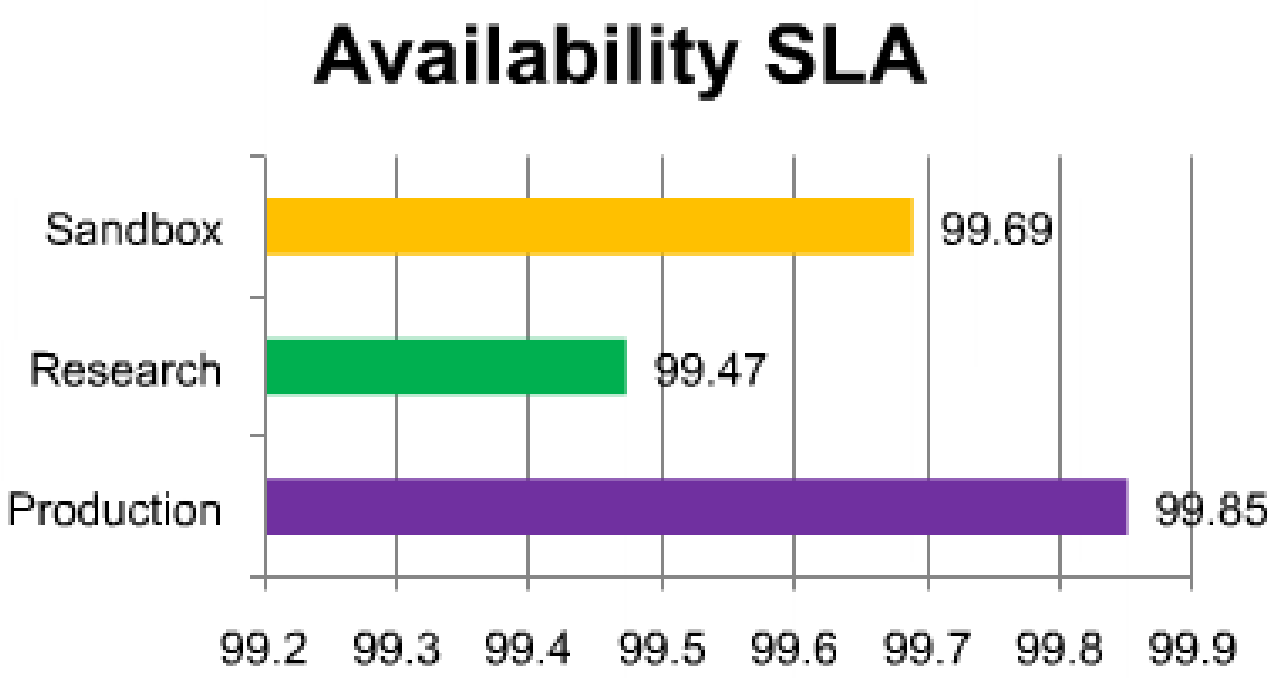
```
public class WordCountReducer
                extends Reducer<Text, IntWritable, Text, IntWritable> {

  private IntWritable count = new IntWritable(); // for Performance

  public void reduce(Text word, Iterable<IntWritable> values,
                  Context context) throws IOException, InterruptedException {
    int sum = 0;
    for (IntWritable val : values) {
      sum += val.get();
    }
    count.set(sum);
    context.write(word, count); // Result: Word Count (ex; hadoop 3)
  }
}
```

Database와 Hadoop의 추천 알고리즘 성능 비교

구분	Oracle 기반 머신	Hadoop 기반 머신
CPU	100%	70%
Core	80 Core	Intel 8 Core * 20 = 160 Core
처리 시간	1시간	34분
기간	1개월	1개월
상품수	120,000,000	
사용자수(T)	1,300,000	
장비 비용	6억 이상 고가 High End Server	300만원 * 20 = 6,000만원
라이선스 비용	예) Core 당 700만원 * 80 = 56,000만원	0

Hadoop의 Availability

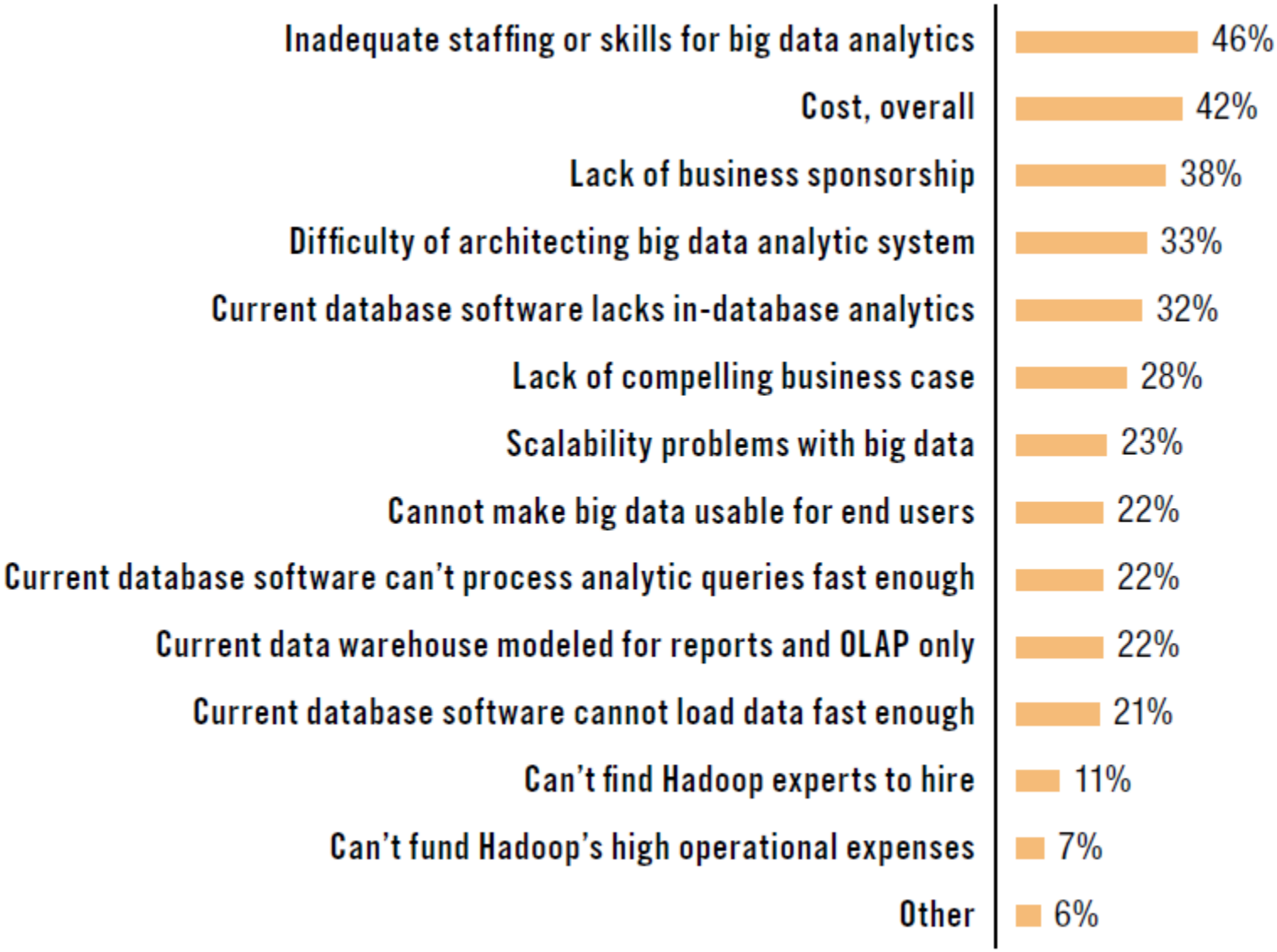
- Hadoop은 기반 인프라 성격을 가지고 있어서 Availability가 매우 중요
- 현재 Hadoop은 Enterprise 수준의 SLA를 제공하지는 않음



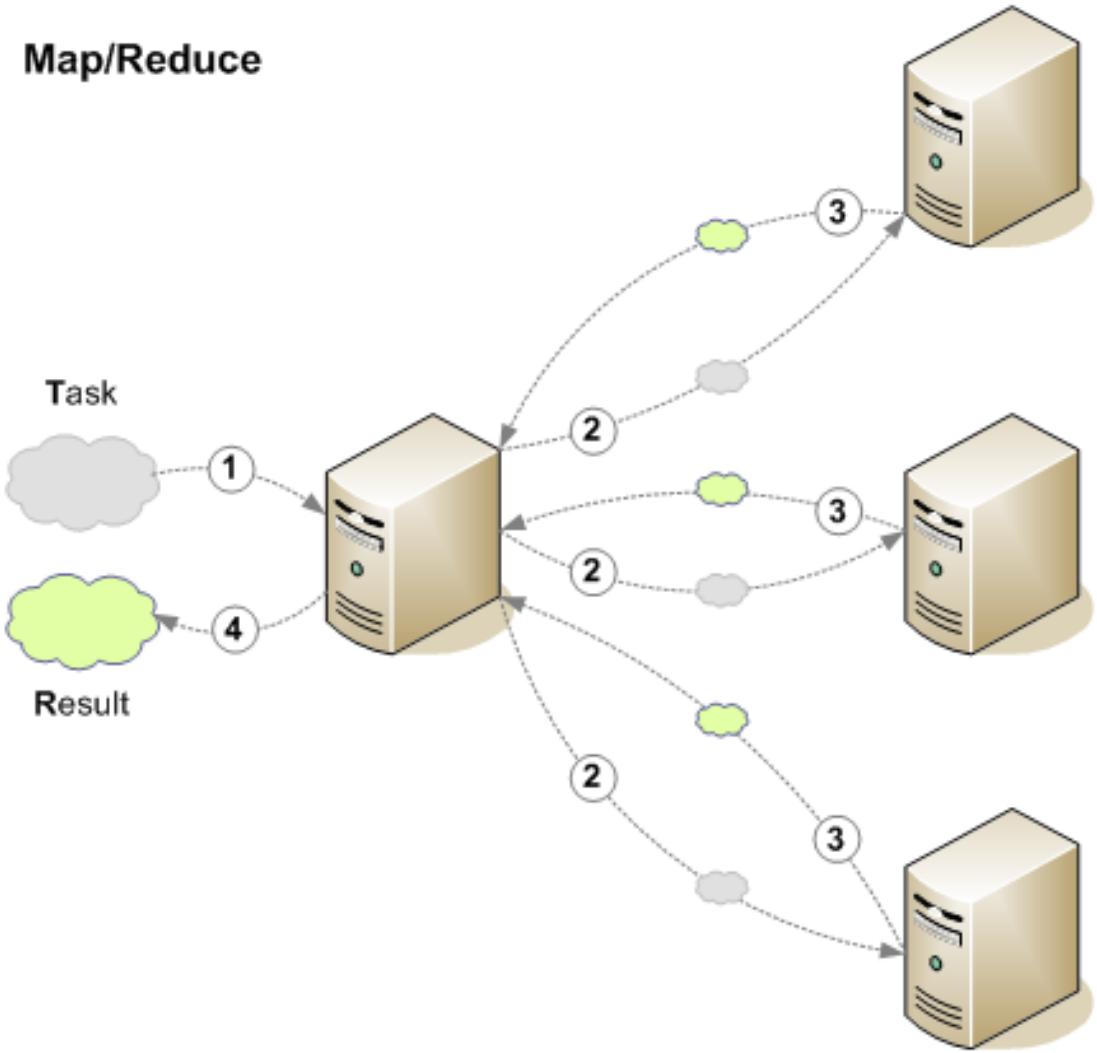
Hadoop 도입 시 고려할 사항

- 기존 오픈 소스와는 다른 유형의 오픈 소스
  - 일반적인 오픈 소스는 Framework, Library 정도 수준이지만 Hadoop은 인프라 성격을 가진 오픈 소스
- 오픈 소스 친화적인 엔지니어 확보 중요
  - 소스코드까지 고치려는 시도를 할 수 있는 엔지니어
- Linux 친화적이면서 System Engineer 성향을 가진 엔지니어 필요
- 적용하기 까지 많은 테스트와 최적화 필요하므로 인내 필요
  - 데이터를 다루는 작업은 기존까지 개발자의 몫이 아니었고
  - 지루한 작업에 매우 많은 염증을 느낄 수 있음

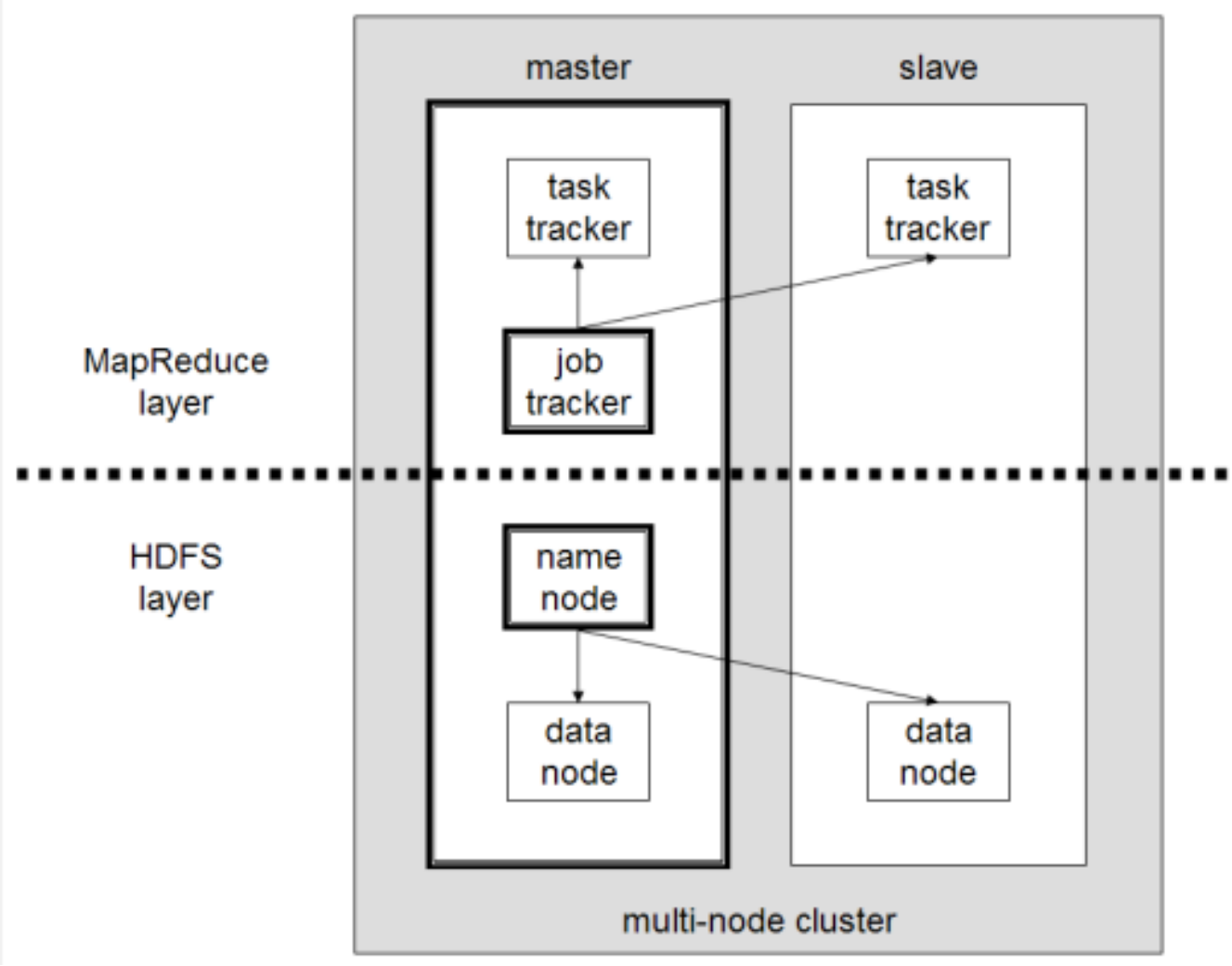
Big Data Analytics 도입 장벽



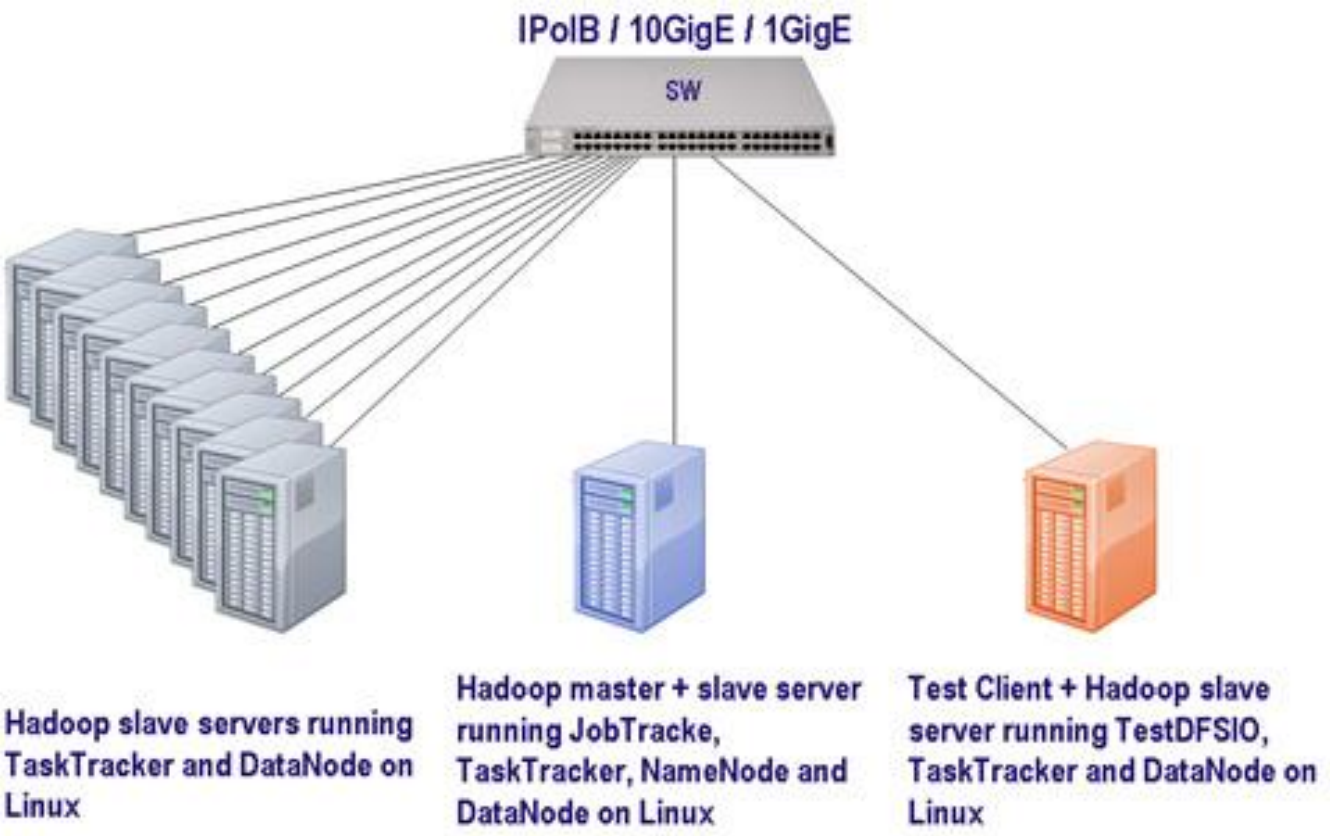
MapReduce



Namenode & Datanode 관계



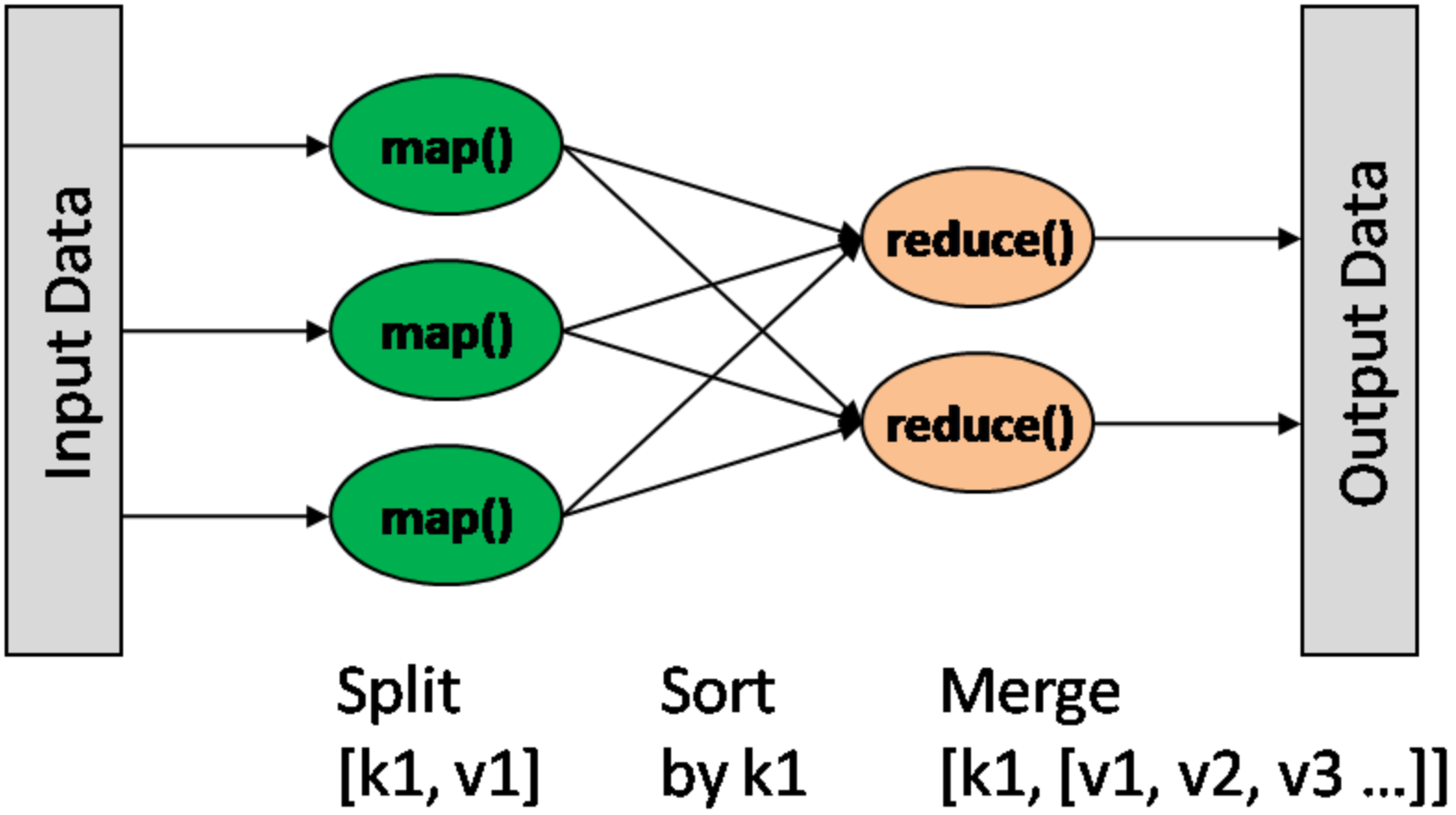
Hadoop Cluster



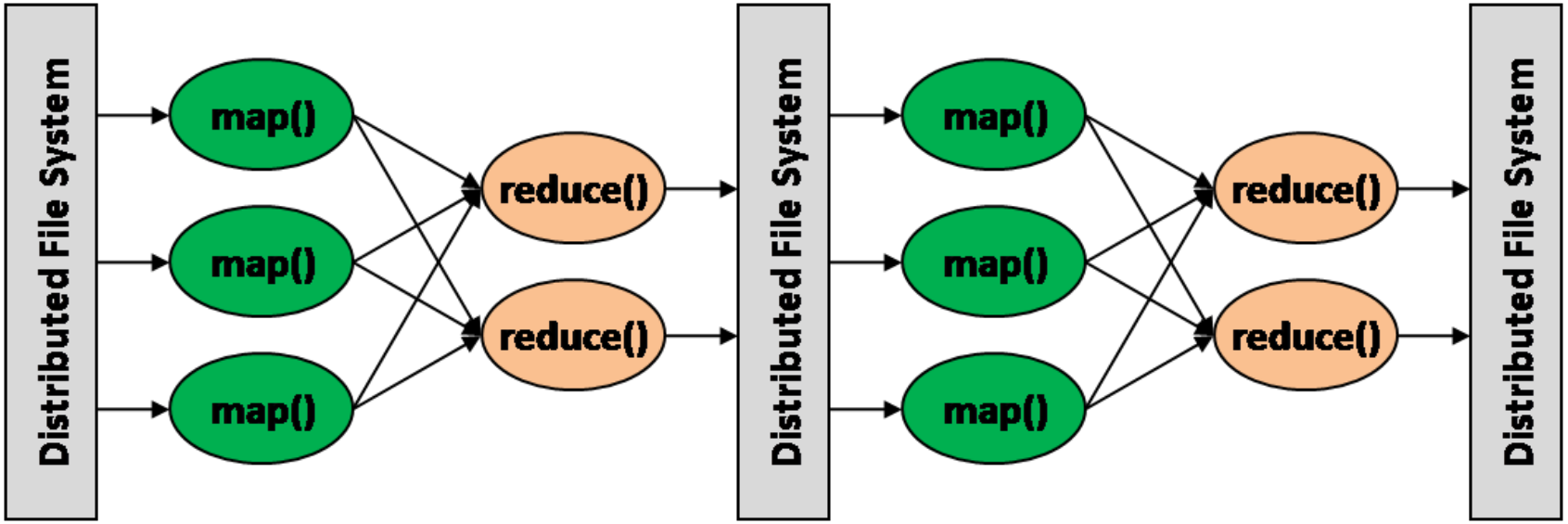
Hadoop Daemon

```
// Namenode
hadoop@namenode:/usr/local/java/hadoop $> jps
16017 Jps
14799 NameNode
15596 JobTracker
14977 SecondaryNameNode

// Datanode
hadoop@datanode1:/usr/local/java/hadoop $> jps
15183 DataNode
15897 TaskTracker
16284 Jps
```

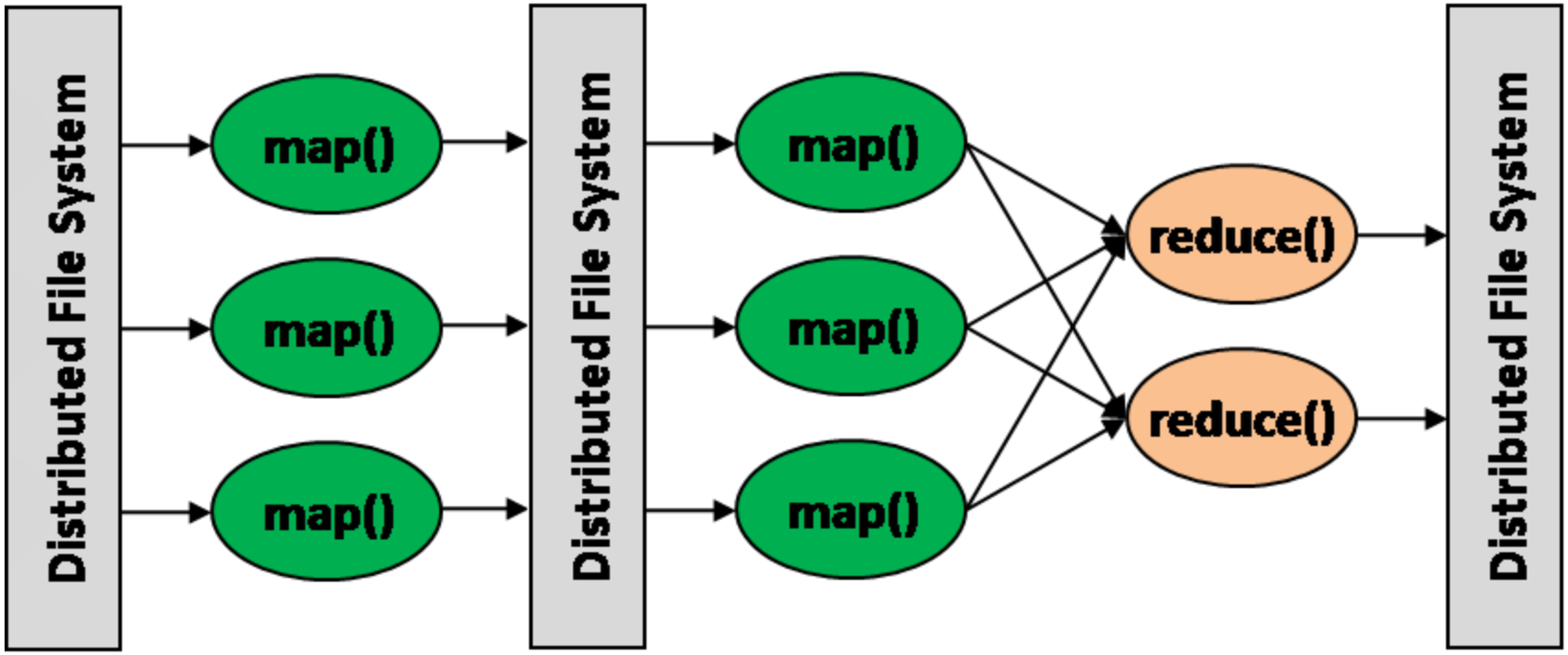
Hadoop MapReduce Job (1)



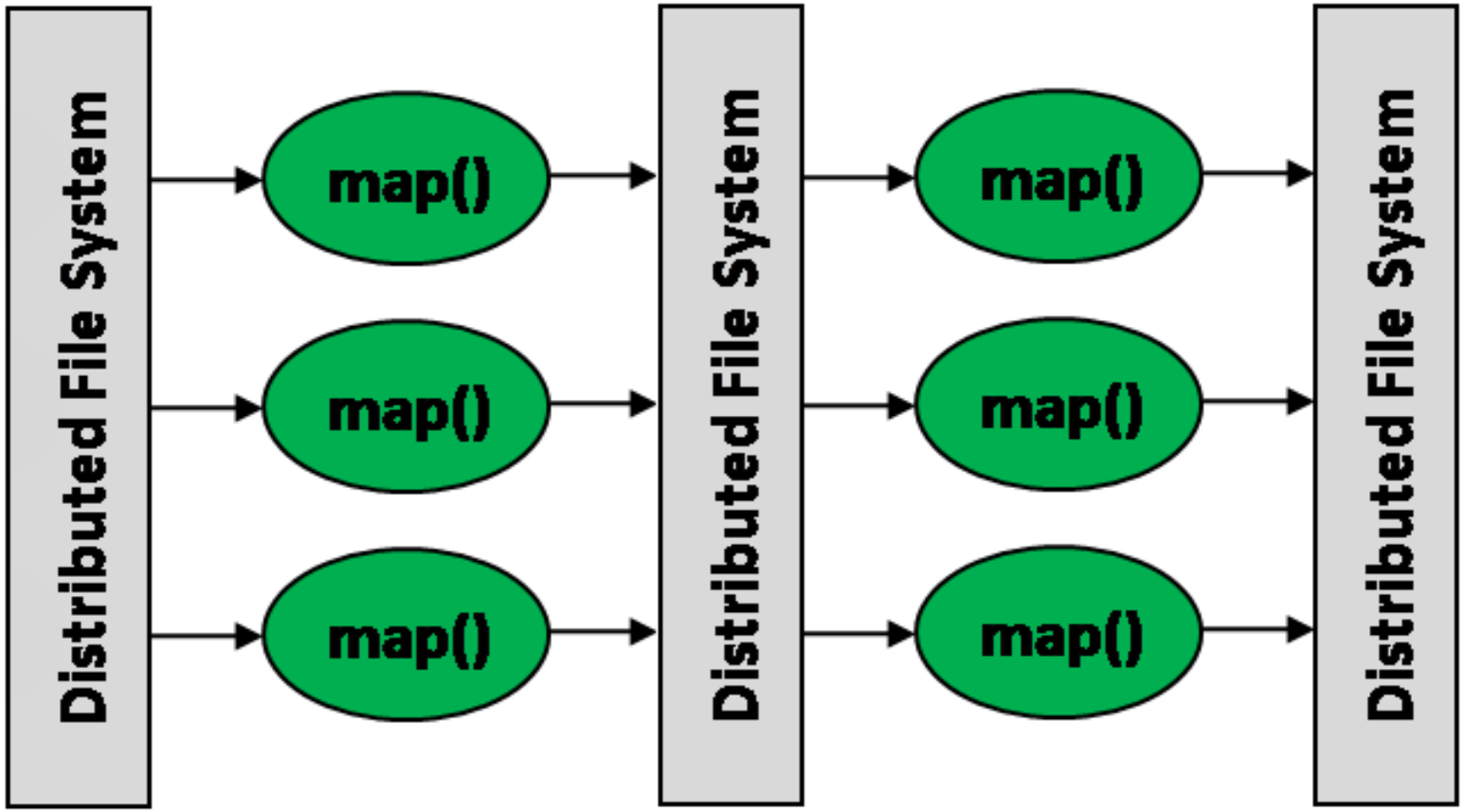
Hadoop MapReduce Job (2)



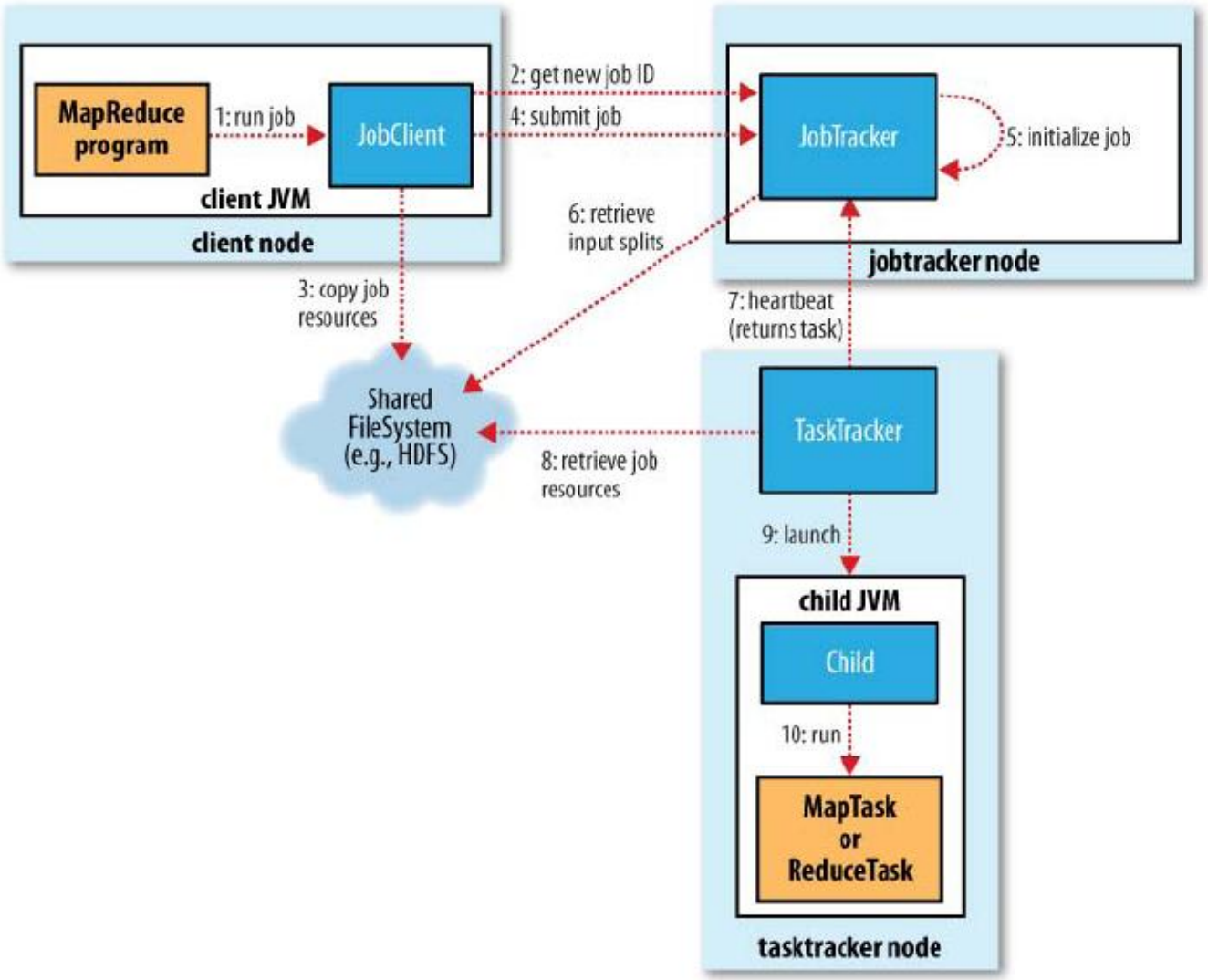
Hadoop MapReduce Job (3)



Hadoop MapReduce Job (4)



MapReduce Job의 동작 원리

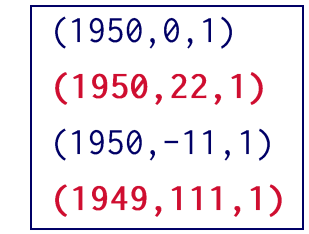


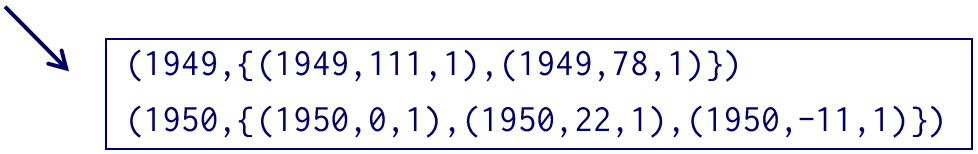
Hadoop & Pig

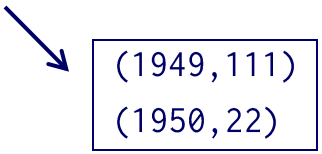
- 대용량 데이터를 고차원적인 방법으로 접근하는 스크립트 언어
  - 스크립트 언어 = Pig Latin
- 사용자가 작성한 스크립트 언어는 MapReduce로 동작
- Pig Latin → MapReduce 과정이 성능에 관건
- 다양한 파일들을 한번에 처리하고자 하는 경우 매우 유용
  - MapReduce의 경우 모두 코드를 작성해야 함
- 다양한 데이터 유형을 제공
  - Bag, Tuple, …

Pig Latin 예제

```
-- max_temp.pig: Finds the maximum temperature by year

records = LOAD 'input/ncdc/micro-tab/sample.txt'
          AS (year:chararray, temperature:int, quality:int);

filtered_records = FILTER records BY temperature != 9999 AND
                   (quality == 0 OR quality == 1 OR
                    quality == 4 OR quality == 5 OR quality == 9);

grouped_records = GROUP filtered_records BY year;

max_temp = FOREACH grouped_records GENERATE group,
                                            MAX(filtered_records.temperature);

DUMP max_temp;
```





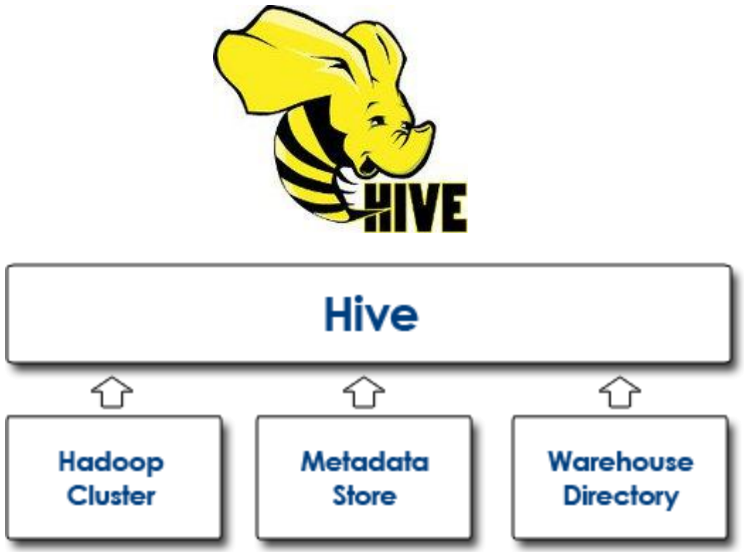


Pig는 만능인가?

- Yahoo!는 80% 정도를 Pig로 사용
  - Why? Yahoo!가 만들었으니까?
- Pig가 편하지만 성능 문제 발생할 수 있음
  - 거지같이 작성한 SQL은 느리다!!
- Pig와 MapReduce를 적절히 혼합해서 사용할 필요가 있다.
  - 그러므로 Workflow Engine이 필요할 수 있다.
    - Oozie?
    - Open Flamingo?
- 특정 작업에 Pig? MapReduce?를 결정하는 것이 중요. 경험!!!

Hive

- Data Warehouse Infrastructure
  - Data Summarization
  - Ad hoc Query on Hadoop
    - MapReduce for Execution
    - HDFS for Storage
- MetaStore
  - Table/Partition
  - Thrift API
  - Metadata stored in any SQL backend
- **Hive Query Language**
  - Basic SQL : Select, From, Join, Group BY
  - Equi-Join, Multi-Table Insert, Multi-Group-By
  - Batch Query
  - https://cwiki.apache.org/Hive/languagemanual.html



DDL Operation

- SQL 기반 DDL Operation

```
hive> CREATE TABLE pokes (foo INT, bar STRING);
hive> CREATE TABLE invites (foo INT, bar STRING)
      PARTITIONED BY (ds STRING);

hive> SHOW TABLES;
hive> SHOW TABLES '.*s';

hive> DESCRIBE invites;

hive> DROP TABLE pokes;

hive> ALTER TABLE pokes ADD COLUMNS (new_col INT);
hive> ALTER TABLE invites ADD COLUMNS (new_col2 INT COMMENT 'a comment');
hive> ALTER TABLE events RENAME TO 3koobecaf;

hive> CREATE TABLE rating (userid STRING, movieid STRING, rating INT) ROW
      FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE;
```

https://cwiki.apache.org/Hive/languagemanual-ddl.html

DML Operation

- 로컬 파일 로딩

```
hive> LOAD DATA LOCAL INPATH './examples/files/kv1.txt' OVERWRITE
      INTO TABLE pokes;
```

- HDFS 로딩

```
hive> LOAD DATA INPATH '/user/myname/kv2.txt' OVERWRITE INTO TABLE
      invites PARTITION (ds='2008-08-15');
```

- Partitioning

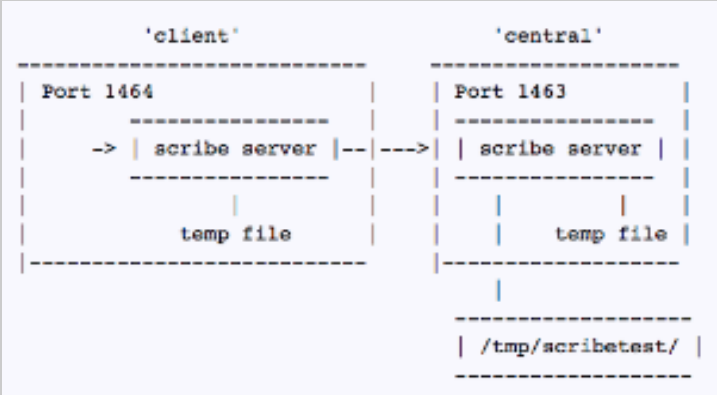
```
hive> LOAD DATA LOCAL INPATH './examples/files/kv2.txt' OVERWRITE INTO
TABLE invites PARTITION (ds='2008-08-15');
hive> LOAD DATA LOCAL INPATH './examples/files/kv3.txt' OVERWRITE INTO
TABLE invites PARTITION (ds='2008-08-08');
```

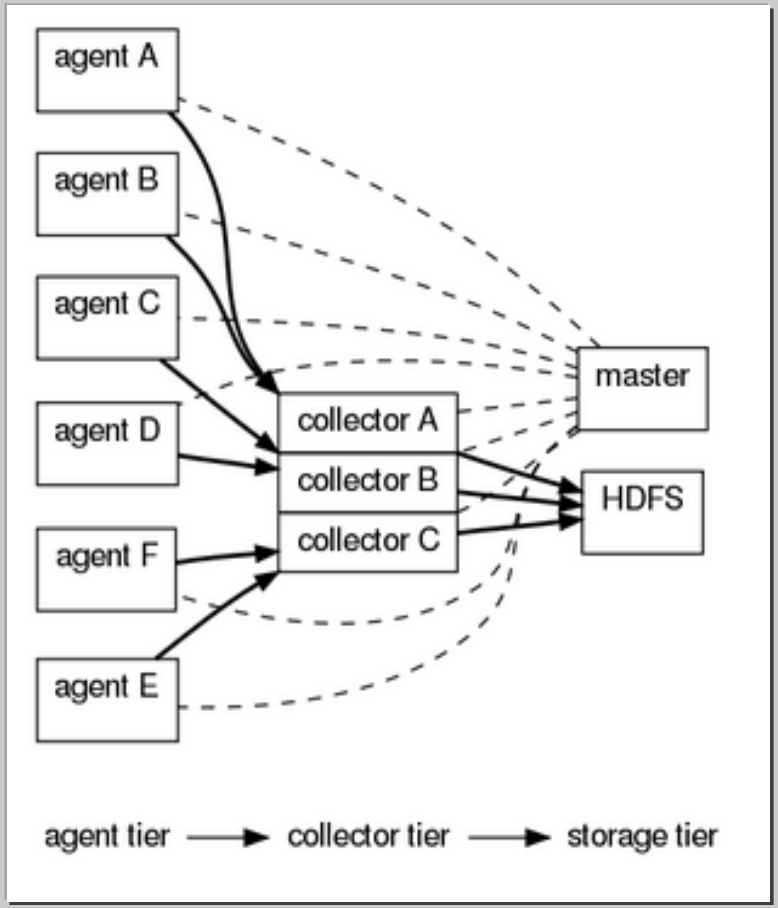
SQL Operation :: Select & Filter

```
hive> SELECT a.foo FROM invites a WHERE a.ds='2008-08-15';

hive> INSERT OVERWRITE DIRECTORY '/tmp/hdfs_out' SELECT a.* FROM invites a
      WHERE a.ds='2008-08-15';

hive> INSERT OVERWRITE LOCAL DIRECTORY '/tmp/local_out' SELECT a.* FROM pokes a;

hive> INSERT OVERWRITE TABLE events SELECT a.* FROM profiles a;
hive> INSERT OVERWRITE TABLE events SELECT a.* FROM profiles a WHERE a.key < 100;
hive> INSERT OVERWRITE LOCAL DIRECTORY '/tmp/reg_3' SELECT a.* FROM events a;
hive> INSERT OVERWRITE DIRECTORY '/tmp/reg_4' select a.invites, a.pokes
      FROM profiles a;
hive> INSERT OVERWRITE DIRECTORY '/tmp/reg_5' SELECT COUNT(*) FROM invites a
      WHERE a.ds='2008-08-15';
hive> INSERT OVERWRITE DIRECTORY '/tmp/reg_5' SELECT a.foo, a.bar FROM invites a;
hive> INSERT OVERWRITE LOCAL DIRECTORY '/tmp/sum' SELECT SUM(a.pc) FROM pc1 a;
```

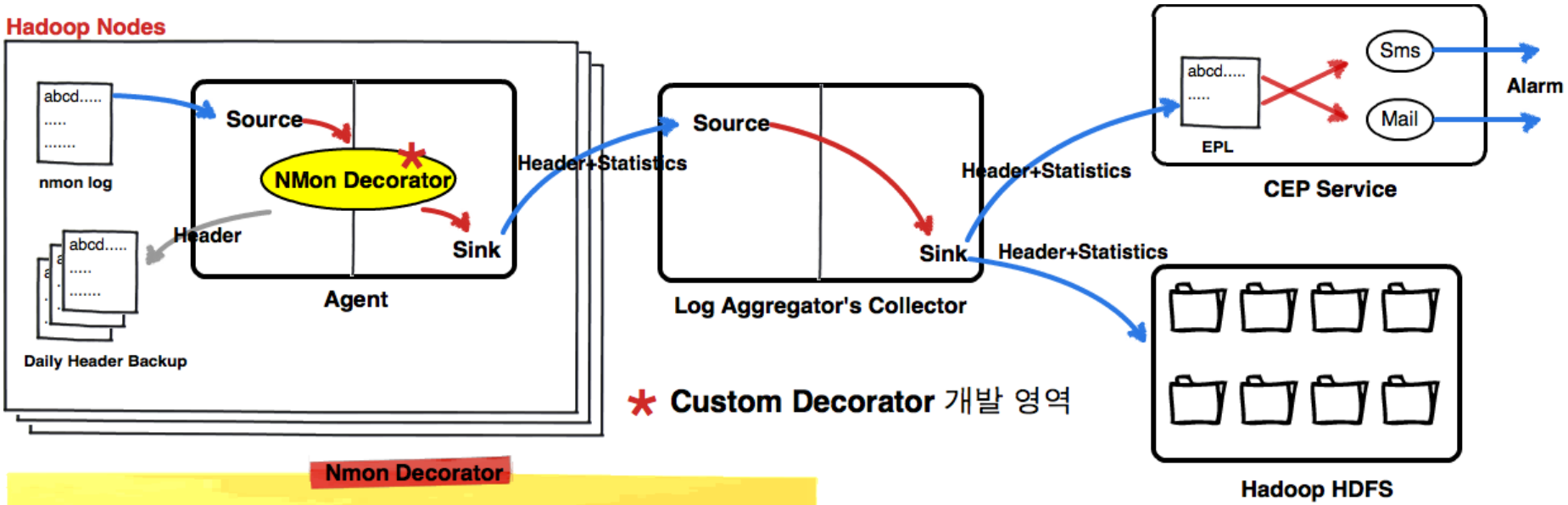
Log Aggregator

Scribe	Flume





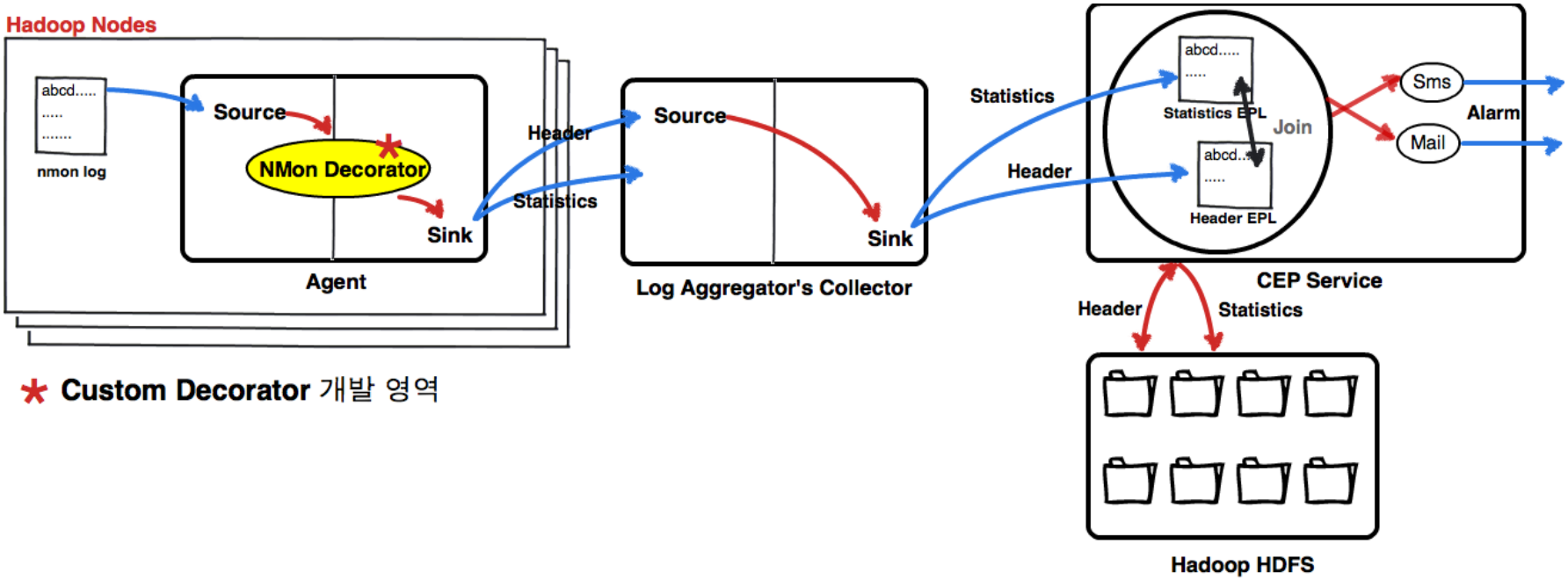
Flume 기반 nmon log 수집 및 실시간 처리 (1)



Nmon Decorator

비정형화 데이터를 정형화 데이터 형식으로 변환하고, Header 데이터와 임계치 데이터를 하나의 이벤트로 합쳐서 CSV형태로 Sink에 보낸다. 또 Header 정보를 Memory로 관리하는 동시에 최초에 로컬 파일로 기록하고 Agent가 비정상 종료 혹은 재시작 시 로컬 파일에서 해당 일에 대한 Header 정보를 검색하고 존재하면 다시 메모리에 적재한다.

Flume 기반 nmon log 수집 및 실시간 처리 (2)



Oozie Workflow Engine

특징

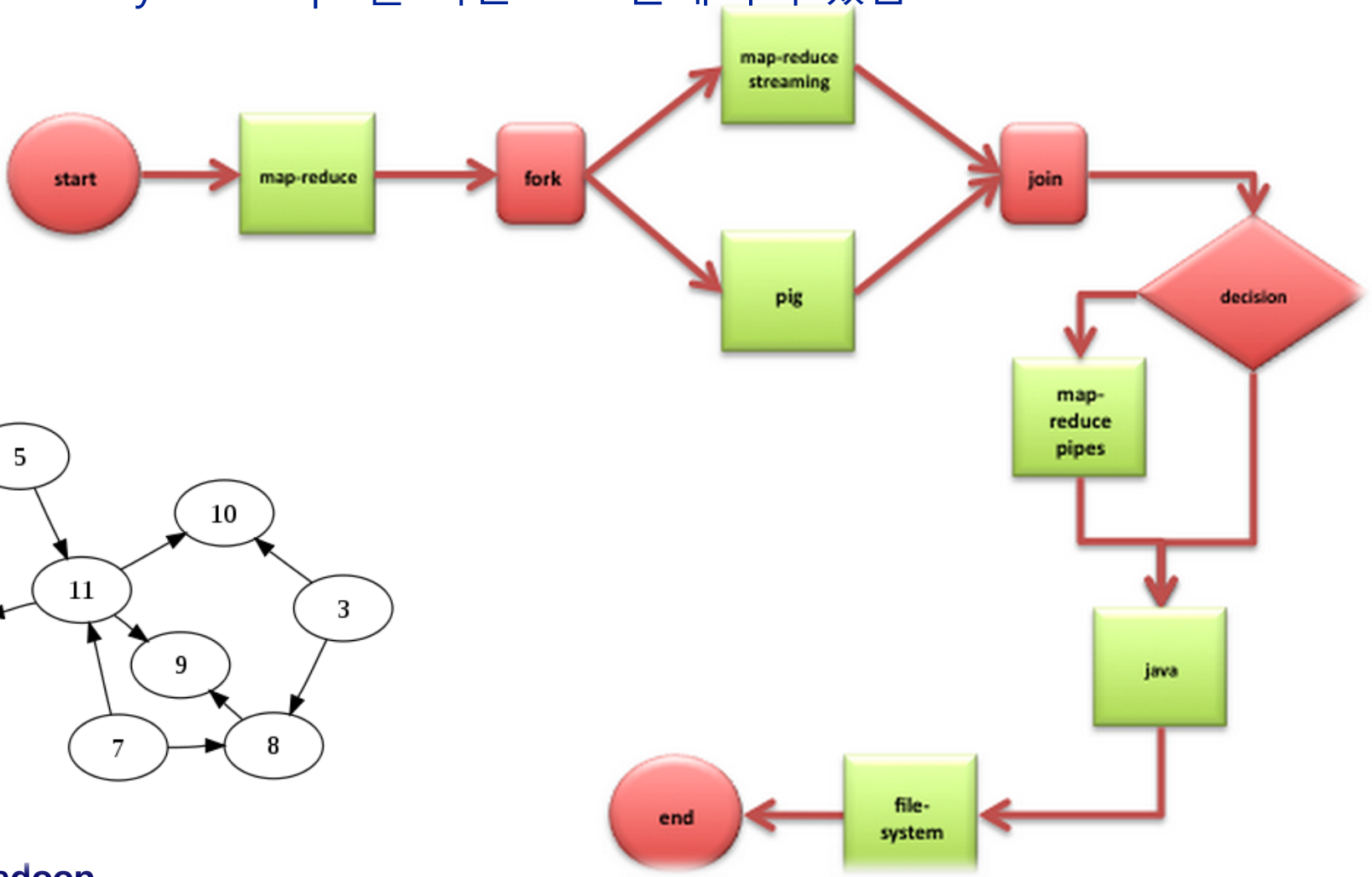
- OpenSource Workflow /Coordination Service for Hadoop Job
- Apache Incubator Project
- Complex Workflow Action
  - ✓ Directed Acyclical Graph (DAG)
  - ✓ Dependency Management
- Reduce Time-To-Market
- Native Hadoop Integration
  - ✓ Support All types of Hadoop Jobs
  - ✓ Integrated with Hadoop Stack
- Frequency Execution
  - ✓ Date/Time Triggered
- Designed for Yahoo!
  - ✓ Yahoo!'s Complex Workflow & Data Pipeline
  - ✓ Global Scale
  - ✓ Yahoo! Distribution of Hadoop with Security

기능

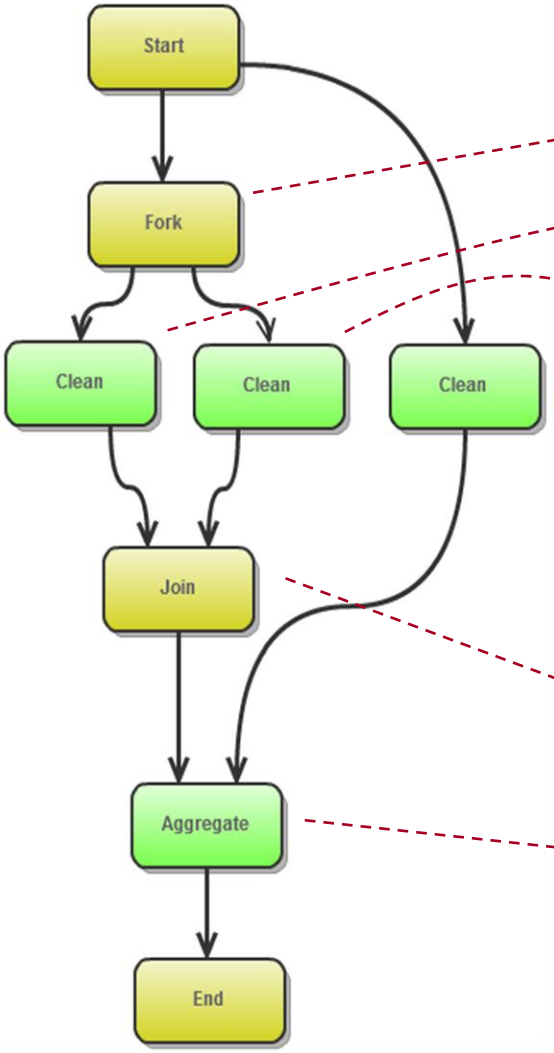
- Job Scheduling (Coordiator)
- Workflow XML
- Workflow Dependency Management
- Parameterization of Action Nodes
  - Expression Language in Workflow XML
- Synchronous Dataset
  - Input/Output
  - Clocked Dataset
- WebService API
- GUI Monitoring (Using ExtJS 2)
- Native Support
  - MapReduce
  - Pig
  - HDFS
  - SSH
  - Java Standalone
  - …

Oozie Workflow Engine - 그래프 이론

그래프 이론 중 하나로 방향성을 가지는 비순환 그래프를 의미하는 Directed Acyclic Graph를 기반으로 설계되어 있음.

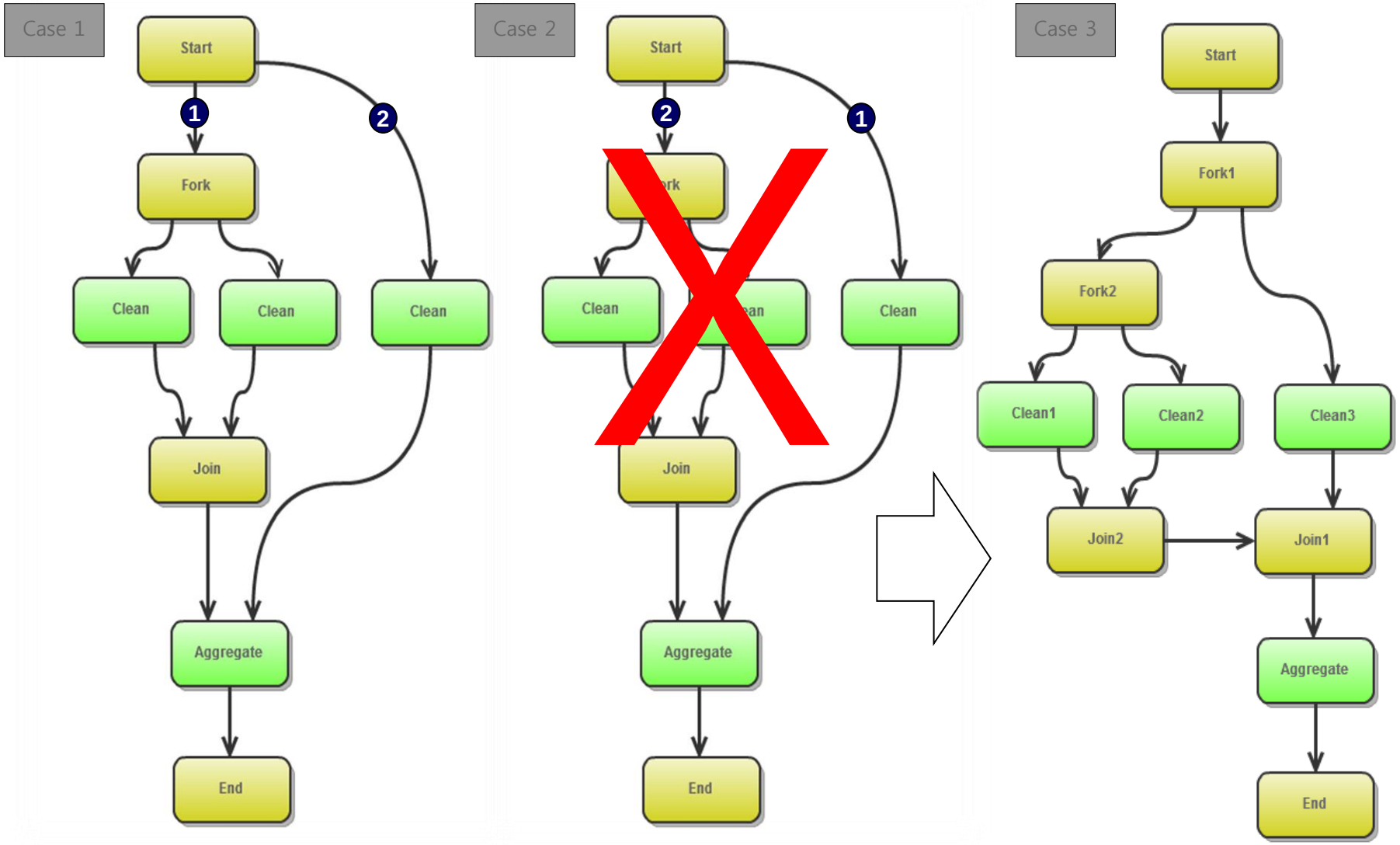


Oozie Workflow Engine – Workflow XML




```
<workflow-app xmlns="uri:oozie:workflow:0.2" name="java-main-wf">
    <start to="cleanup-node"/>
    <action name="cleanup-node"/>

    <fork name="fork-node">
        <path start="clean1-node"/>
        <path start="clean2-node"/>
    </fork>

    <action name="clean1-node">
        <java>
            <main-class>org.openflamingo.mapreduce.etl.clean.CleanDriver</main-class>
        </java>
        <ok to="join-node"/>
        <error to="fail"/>
    </action>
    <action name="clean2-node">
        <java>
            <main-class>org.openflamingo.mapreduce.etl.clean.CleanDriver</main-class>
        </java>
        <ok to="join-node"/>
        <error to="fail"/>
    </action>
    <action name="clean3-node">
        <java>
            <main-class>org.openflamingo.mapreduce.etl.clean.CleanDriver</main-class>
        </java>
        <ok to="join-node"/>
        <error to="fail"/>
    </action>

    <join name="join-node" to="aggregate-node"/>

    <action name="aggregate-node">
        <java>
            <main-class>org.openflamingo.mapreduce.etl.aggregate.AggregateDriver</main-class>
        </java>
        <ok to="end"/>
        <error to="fail"/>
    </action>
    <kill name="fail“/>
    <end name="end"/>
</workflow-app>
```

Oozie Workflow Engine – Workflow XML



Oozie Workflow Engine – Job Scheduling

```
<coordinator-app name="MY_APP" frequency="5" start="2009-02-01T00:00Z" end="2009-02-02T00:00Z" timezone="UTC"
                 xmlns="uri:oozie:coordinator:0.1">
   <datasets>
      <dataset name="input1" frequency="15" initial-instance="2009-01-01T00:00Z" timezone="UTC">
         <uri-template>hdfs://localhost:9000/tmp/revenue_feed/${YEAR}/${MONTH}/${DAY}/${HOUR}/${MINUTE}</uri-template>
      </dataset>
   </datasets>

   <input-events>
      <data-in name="coordInput1" dataset="input1">
          <start-instance>${coord:current(-3)}</start-instance>
          <end-instance>${coord:current(0)}</end-instance>
      </data-in>
   </input-events>

   <action>
      <workflow>
         <app-path>hdfs://localhost:9000/tmp/workflows</app-path>
         <configuration>
           <property>
              <name>input_files</name>
              <value>${coord:dataIn('coordInput1')}</value>
           </property>
         </configuration>
      </workflow>
   </action>
</coordinator-app>
```

구분	설정 방법
YEAR	YYYY
MONT	DD
DAY	DD
HOUR	HH
MIN	mm

Dataset을 정의하고 생성 주기 및 디렉토리 규칙을 정의

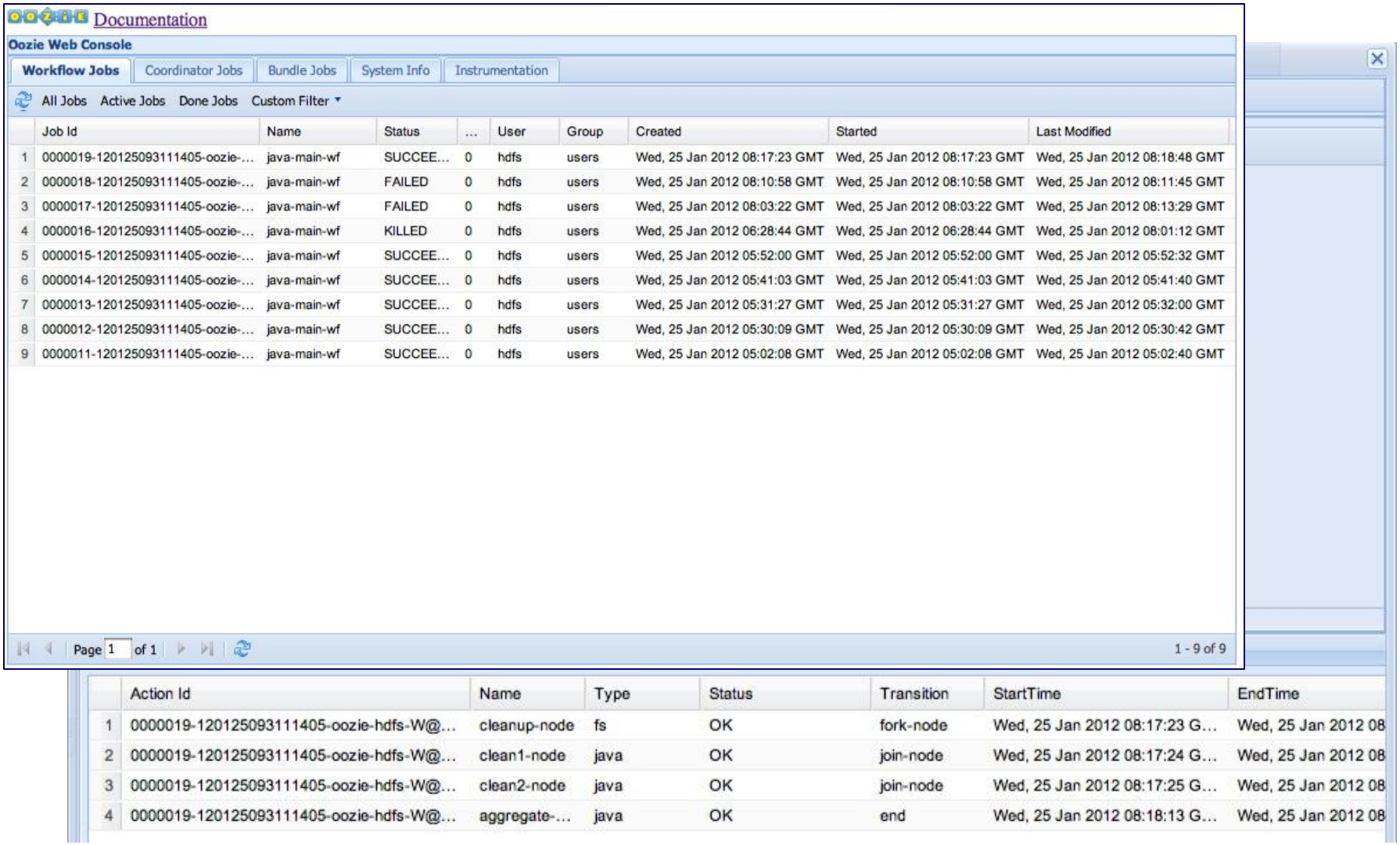
입력으로 사용할 파일의 범위 지정
(오늘부터 4일정까지 파일을 지정)

```
/tmp/revenue_feed/2009/01/01/00/
/tmp/revenue_feed/2009/01/01/01/
/tmp/revenue_feed/2009/01/01/02/
...
```

Frequency 지정 방법

구분	설정 방법
매5분	frequency="5" frequency="${coord:minutes(5)}"
매시간	frequency="60" frequency="${coord:hours(1)}"
매일	frequency="1440" frequency="${coord:days(1)}"
매주	frequency="${coord:days(7)}"
매월	frequency="${coord:months(1)}"

Oozie Workflow Engine - Console



Amazon 상품 추천

What Other Customers Are Looking At Right Now

Patriot Xporter XT Boost 8 GB USB 2.0...
~~$43.99~~ $24.99

Superwinch 1585200 LP8500 Series...
~~$652.74~~ $455.99

Kindle DX Wireless Reading Device
$489.00

Autel MaxiScan MS300 CAN OBD-II Scan...
~~$39.99~~ $27.50

More Items to Consider

You viewed | Customers who viewed this also viewed

Bose® SoundDock® Series II digital...
$299.95

Bose® SoundDock® Portable digital...
$399.95

Bose® SoundDock® Series II digital...
$299.95

Bose SoundDock digital music system...
~~$249.99~~ $224.88

› View or edit your browsing history

추천 로직(시스템)을 구현하는 방법

- 협업 필터링(Collaborating Filtering; CF)
  - 사용자로부터 얻은 기호 정보(예; Rating)를 토대로 사용자들의 관심사를 예측
  - 사용자 기반 필터링 : 취향이 비슷한 사용자
  - 아이템 기반 필터링 : 유사한 제품
- **연관 규칙(Association Rule)**
  - **상품간의 연관성을 분석하여 다른 사용자에게 구매하지 않은 상품을 추천**
- 클러스터링(Clustering)
  - 비슷한 대상들끼리 묶어서 군집을 형성
  - 군집을 형성한 대상은 유사하다고 판단
- 분류(Classification)
- … 등등

추천 시스템을 이용한 웹 사이트

- Melon (Music)
- IPTV
- Portal
- Amazon (Product)
- Last.fm (Music)
- Digg (News)
- Delicious (Web)
- Netflix (Movie)

Association Rule

- 로그 정보(예; 구매 이력)를 토대로 항목간 관계(연관 규칙)를 알아내는 데이터 마이닝 기법
- 가장 구현하기 쉽기 때문에 현업에서 많이 활용
- 월마트의 분석 사례
  - "기저귀를 사는 젊은 남성은 맥주도 산다"
    → 기저귀 근처에 맥주를 위치
  - "허리케인이 상륙하기 전에 딸기과자와 맥주가 많이 팔린다"
    → 허리케인이 상륙하기 전에는 딸기과자와 맥주를 같이 판매
- 상품간 연관 규칙을 알아내어 구매하지 않은 고객에게 구매를 유도

Association Rule

- 트랜잭션(거래수) = T
- 지지도(Support) = $P(A \cap B) = N(A \cap B) / N(T)$
  - 많이 구매할수록 지지도는 상승
  - A와 B가 동시에 구매할 빈도수
- 신뢰도(Confidence) = $P(B|A) = P(A \cap B) / P(A)$
  - A를 포함하는 장바구니 중에서 B를 포함하는 빈도수
- 향상도(Lift) = $P(B|A) / P(B) = P(A \cap B) / P(A)P(B)$
  - A를 구매한 경우 그 트랜잭션이 B를 포함하는 경우와 B가 임의로 구매하는 경우의 비율
  - Lift > 1 : 높을 수록 연관이 깊다
- S,C는 얼마나 규칙이 유용한가 여부, L은 매출 향상의 기여도

Association Rule

T	품목
1	우유, 빵, 버터
2	우유, 버터, 콜라
3	빵, 버터, 콜라
4	우유, 콜라, 라면
5	빵, 버터, 라면

품목	개수	지지도
우유	3	3/5=0.6
빵	3	3/5=0.6
버터	4	4/5=0.8
콜라	3	3/5=0.6
라면	2	2/5=0.4

품목(A→B)	개	지지도(S)	신뢰도(C)	향상도(L)
빵→버터	**3**	**3/5=0.6**	**0.6/0.6=1**	**0.6/0.6*0.8=1.25**
버터→빵	**3**	**3/5=0.6**	**0.75**	**1.25**
콜라→우유	**2**	**2/5=0.4**	**0.66**	**1.11**
우유→콜라	**2**	**2/5=0.4**	**0.66**	**1.11**
우유→버터	2	2/5=0.4	0.66	0.83
콜라→버터	2	2/5=0.4	0.66	0.83
버터→우유	2	2/5=0.4	0.5	0.83
버터→콜라	2	2/5=0.4	0.5	0.83
라면→우유	1	1/5=0.2	0.5	0.83
라면→빵	1	1/5=0.2	0.5	0.83
라면→콜라	1	1/5=0.2	0.5	0.83
콜라→라면	1	1/5=0.2	0.33	0.83
우유→라면	1	1/5=0.2	0.33	0.83

신뢰도(Confidence) = P(B|A) = P(A∩B) / P(A)

향상도(Lift) = P(B|A) / P(B) = P(A∩B)/P(A)P(B)

Association Rule

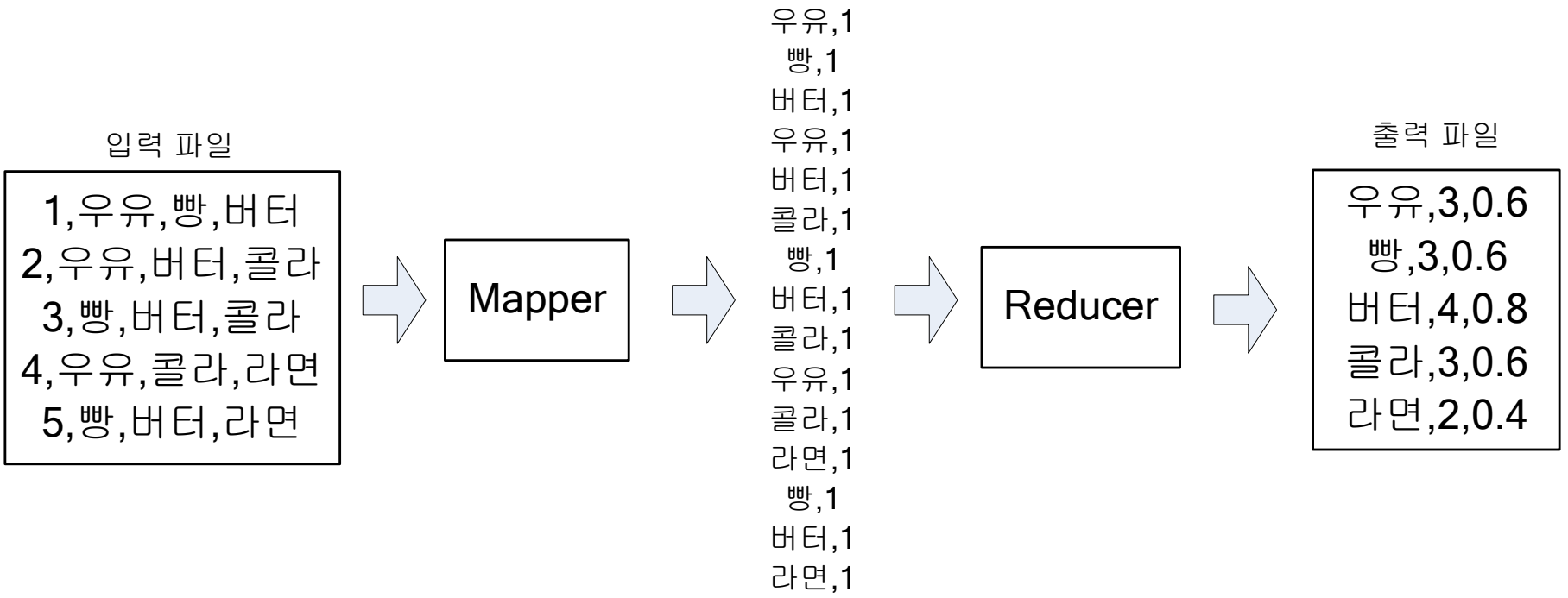
입력 파일	출력 파일
1,우유,빵,버터 2,우유,버터,콜라 3,빵,버터,콜라 4,우유,콜라,라면 5,빵,버터,라면	빵,버터,0.6,1,1.25 버터,빵,0.6,0.75,1.25 콜라,우유,0.4,0.66,1.1111112 우유,콜라,0.4,0.66,1.1111112 우유,버터,0.4,0.66,0.8333333 콜라,버터,0.4,0.66,0.8333333 버터,우유,0.4,0.5,0.8333333 버터,콜라,0.4,0.5,0.8333333 라면,우유,0.4,0.5,0.8333333 라면,빵,0.2,0.5,0.8333333 라면,콜라,0.2,0.5,0.8333333 콜라,라면,0.2,0.33,0.8333333 우유,라면,0.2,0.33,0.8333333 빵,라면,0.2,0.33,0.8333333 라면,버터,0.2,0.5,0.625 버터,라면,0.2,0.25,0.625

Hadoop으로 Association Rule 구현하기

- AR 구현 시 알아내야 하는 값
  - 트랜잭션
  - 개별 아이템의 개수(지지도 계산을 위해서 필요)
  - 아이템의 조합 그리고 그 조합의 개수(지지도 계산)
- 성능 이슈
  - 조합의 개수가 늘어날 수록 급격하게 성능 저하 발생
  - 이런 이유로 2개의 아이템 조합을 현업에서 주로 사용
- MapReduce를 두 번 실행해서 처리
  - 첫 번째 단계 : 개별 아이템의 지지도 계산
  - 두 번째 단계 : 조합을 생성하고 조합에 대한 S,C,L을 계산

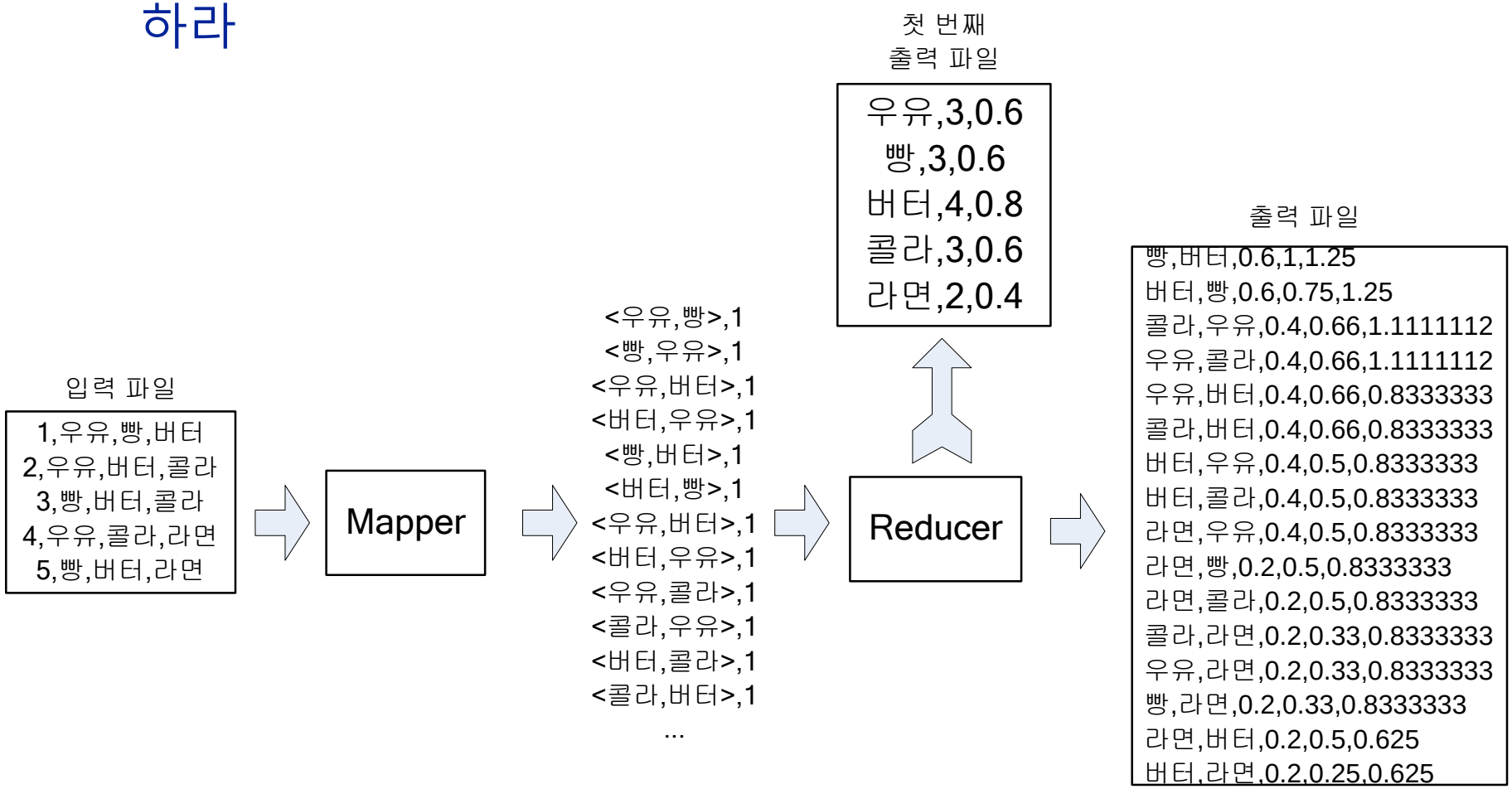
첫 번째 단계 : 아이템의 지지도를 계산하라

- AR을 제대로 처리하려면 무조건 개별 아이템의 지지도를 알아내야한다!



두 번째 단계 : 아이템 조합을 생성해라

- 아이템 조합을 생성해서 첫 번째 단계의 지지도로 S,C,L을 계산하라



사용자별 상품 추천

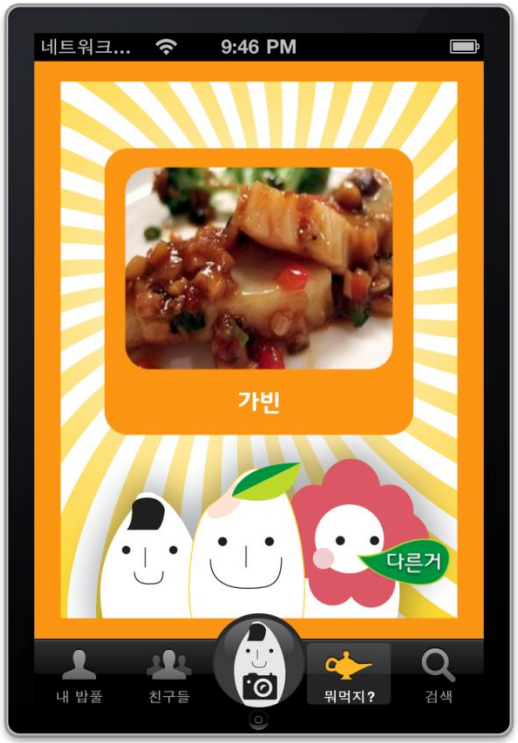
T	품목
1	**우유**, 빵, 버터
2	우유, 버터, 콜라
3	빵, 버터, **콜라**
4	우유, 콜라, 라면
5	빵, 버터, 라면

품목(A→B)	개	지지도(S)	신뢰도(C)	향상도(L)
빵→버터	**3**	**3/5=0.6**	**0.6/0.6=1**	**0.6/0.6*0.8=1.25**
버터→빵	**3**	**3/5=0.6**	**0.75**	**1.25**
콜라→우유	**2**	**2/5=0.4**	**0.66**	**1.11**
우유→콜라	**2**	**2/5=0.4**	**0.66**	**1.11**
우유→버터	2	2/5=0.4	0.66	0.83
콜라→버터	2	2/5=0.4	0.66	0.83
버터→우유	2	2/5=0.4	0.5	0.83
버터→콜라	2	2/5=0.4	0.5	0.83
라면→우유	1	1/5=0.2	0.5	0.83
라면→빵	1	1/5=0.2	0.5	0.83
라면→콜라	1	1/5=0.2	0.5	0.83
콜라→라면	1	1/5=0.2	0.33	0.83
우유→라면	1	1/5=0.2	0.33	0.83

사용자 1 : 콜라를 추천
사용자 3 : 우유를 추천

애플리케이션 상태에 따른 추천 방법의 변화

마이닝 알고리즘을 혼합하여 최종 맛집을 추천하기 위한 데이터 세트 생성



현위치

User Id

사용자의 개인 선호도 정보가 존재하는 근처 맛집이 있는가?

Yes

(AR+CF)

근처 맛집 중 사용자의

예상 선호도가 높은 맛집을 검색

No(Cold Start)

친구의 맛집 리스트

중에서 검색

추천 맛집이 없는 경우

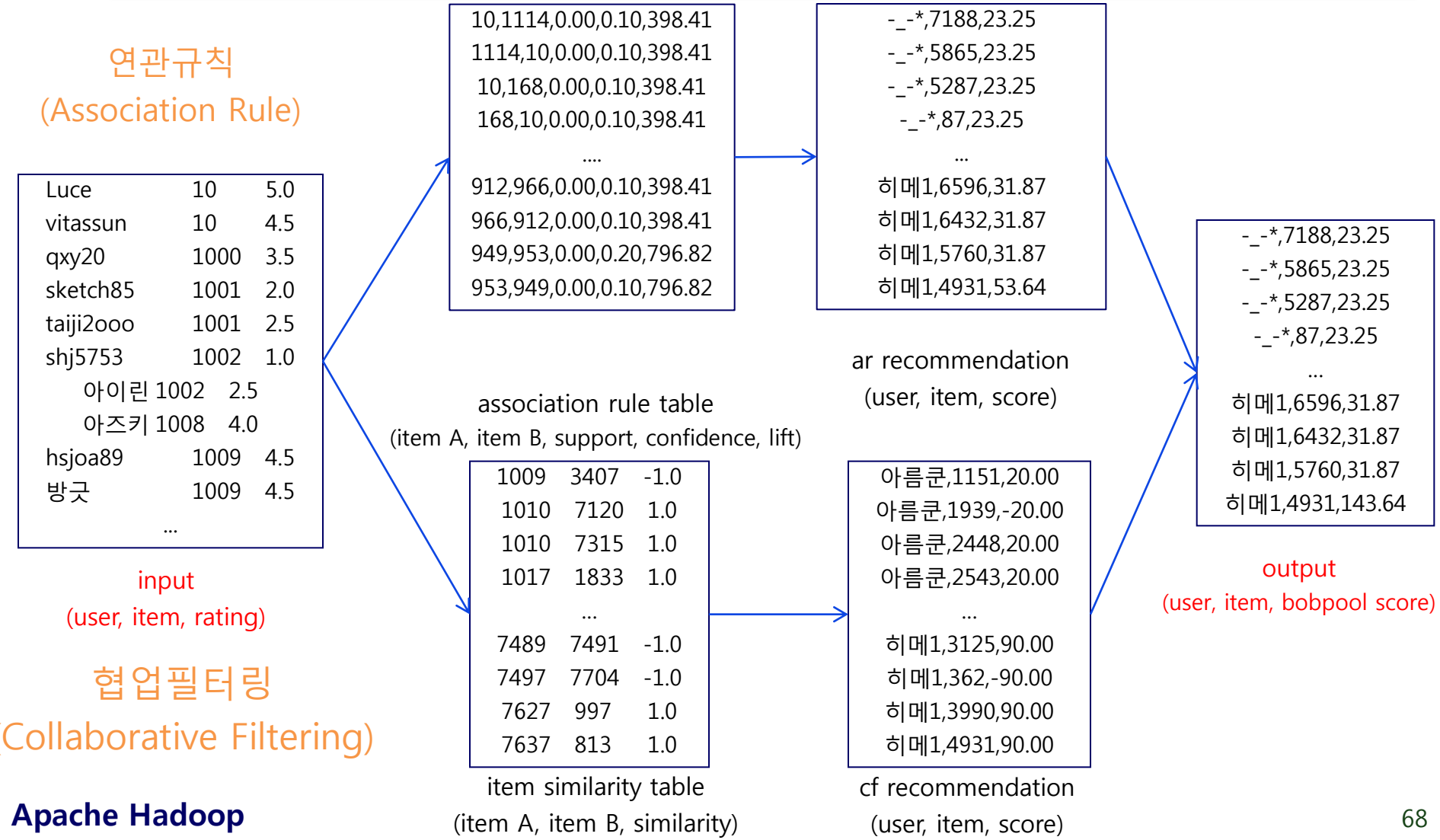
자신의 맛집 리스트 중에서 추천

추천 맛집이 없는 경우

근처 맛집중 점수가 높은 맛집을 추천

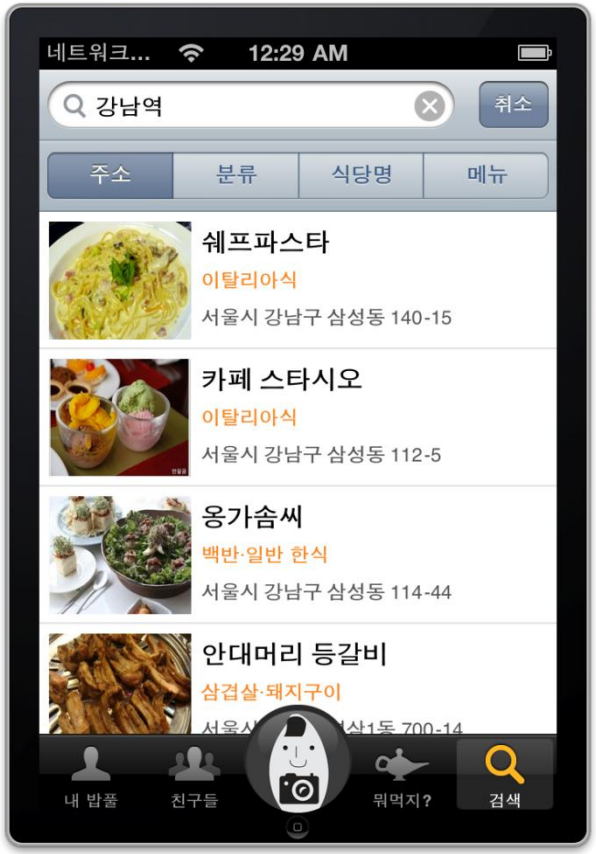
하나 이상의 추천 알고리즘의 결합

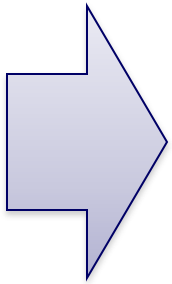
최종 추천 데이터는 매우 복잡한 데이터 가공이 필요하며 시간이 오래 소요됨

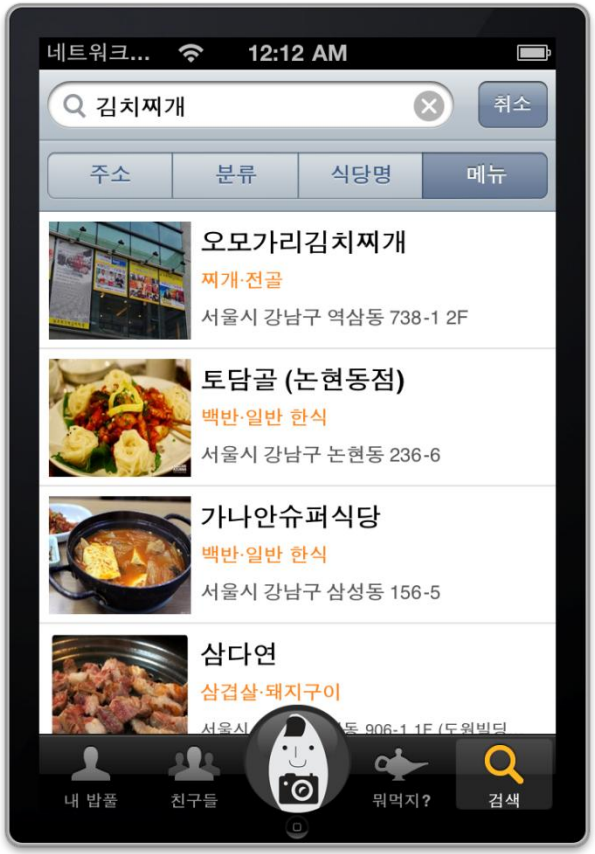


추천 결과 데이터를 이용한 다양한 정보 제공

추천 데이터를 생성하여 검색 엔진의 근간 데이터를 생성







단일 조건으로 '강남역' 검색

복합조건으로 '강남역 김치찌개' 검색

Big Data 활용시 어려운점

- 오픈소스 제품을 신뢰하지 마라
- 데이터 처리는 지금까지 상용 솔루션으로 처리해왔다.
  - 오픈소스로 상용처럼 해달라는 요건
- 경험 부족으로 발생하는 일정 이슈
  - 전부 개발을 해야하나 솔루션 도입 관점에서 생각
- 데이터 프로세싱의 개념을 이해 못하는 이슈
  - Big Data는 멀티 코어 및 장비에서 분산 처리되는 개념
- 경험자는 그냥 된다는 인식의 이슈
  - 모든 Biz는 항상 새롭게 시작해야 한다.