



정보화 사회 실천 연합  
PCIS  
Practice Coalition for Information Society

# 빅데이터와 분석 알고리즘

2013. 08. 27

정보화사회실천연합

qna.pcis@daum.net

## 01 Big Data

- 1.1 Big Data
- 1.2 Big Data Technology
- 1.3 Big Data 현황
- 1.4 Apache Project

## 02 Data

- 2.1 Data의 특성
- 2.2 통계 분석기법
- 2.3 마이닝 분석기법

## 03 Algorithm

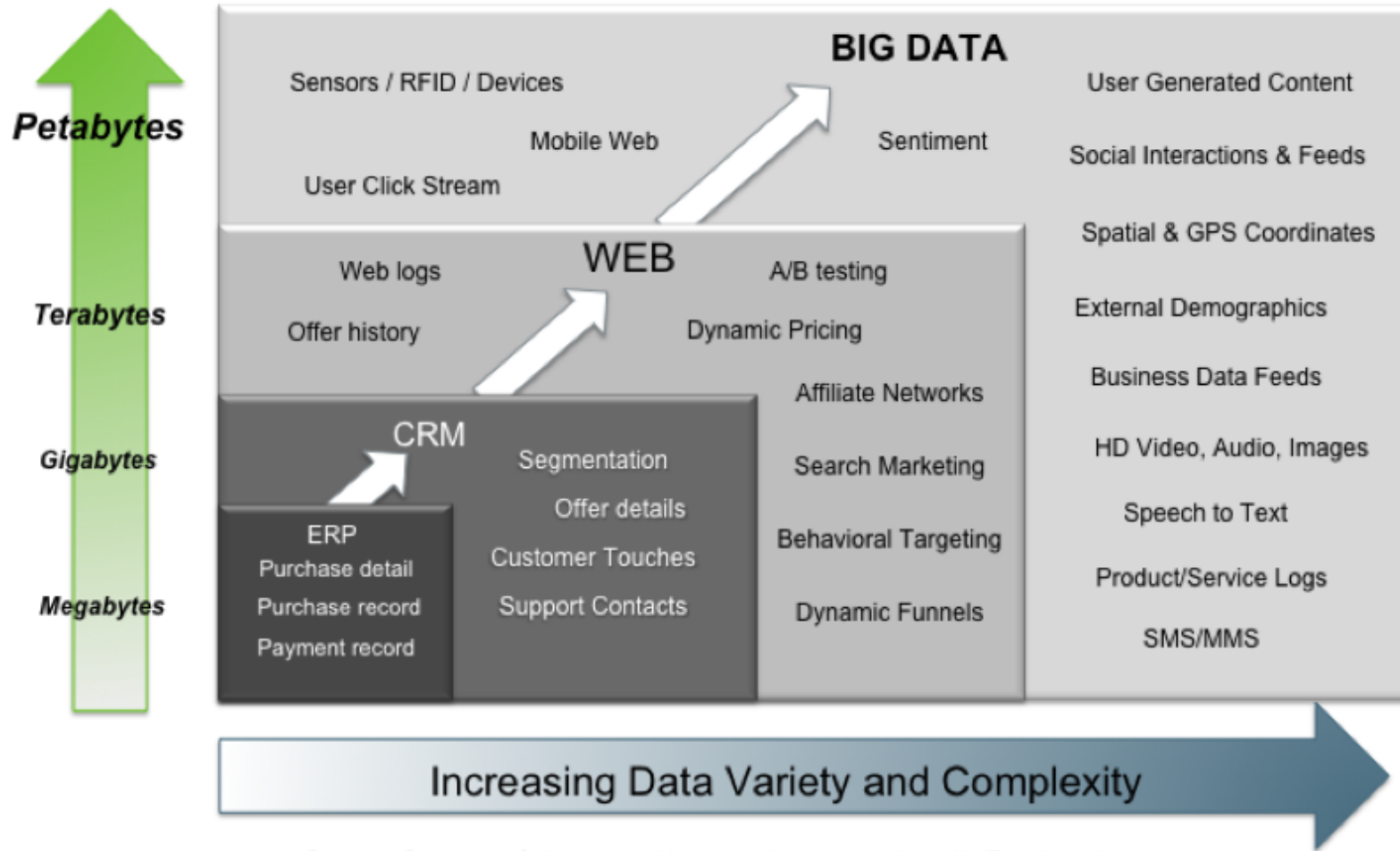
- 3.1 알고리즘의 특성
- 3.2 알고리즘의 분산처리
- 3.3 분석자료의 범위 선정

## 04 Data mining

- 4.1 Data Mining
- 4.2 Classification rules
- 4.3 Clustering Rules
- 4.4 Association Rules
- 4.5 Link Analysis

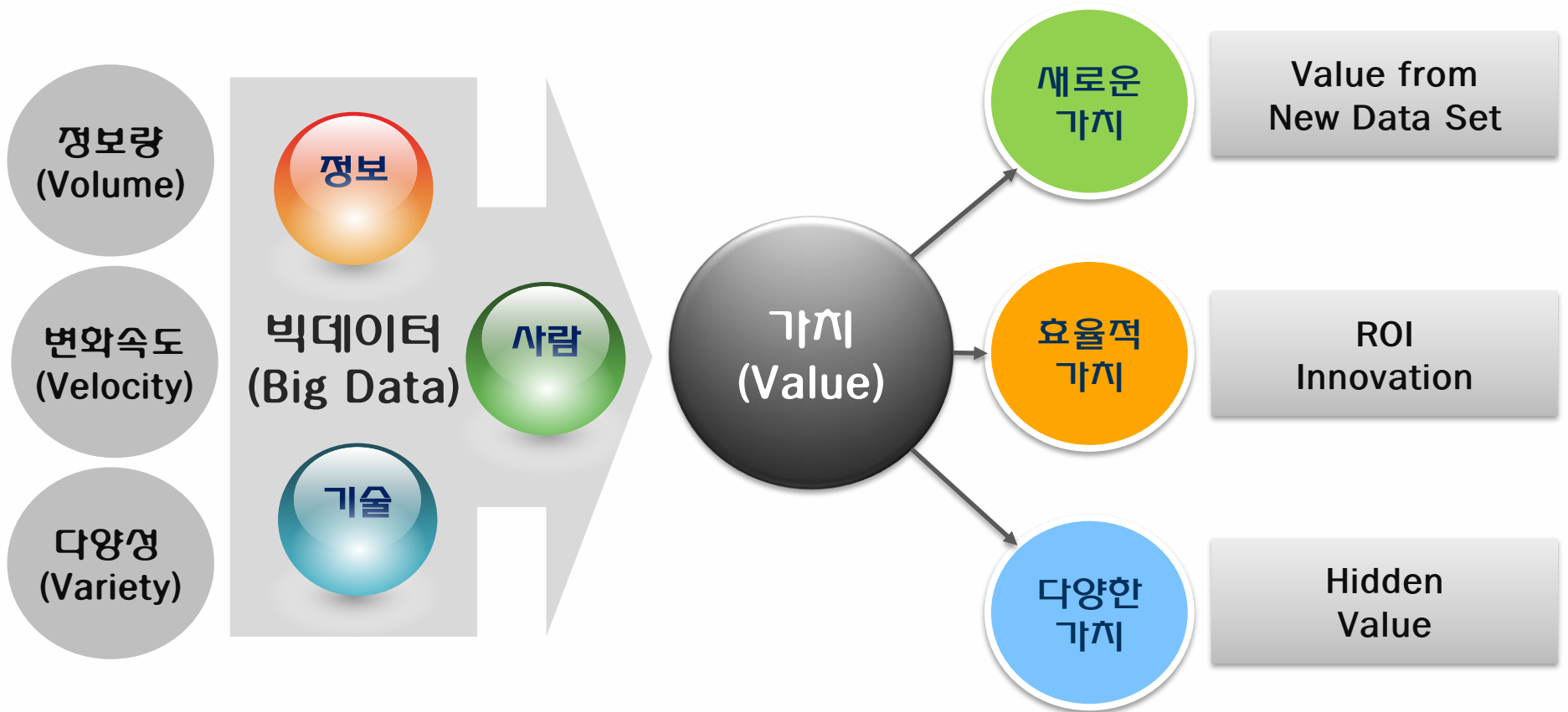
## Big Data

Big Data = Transactions + Interactions + Observations

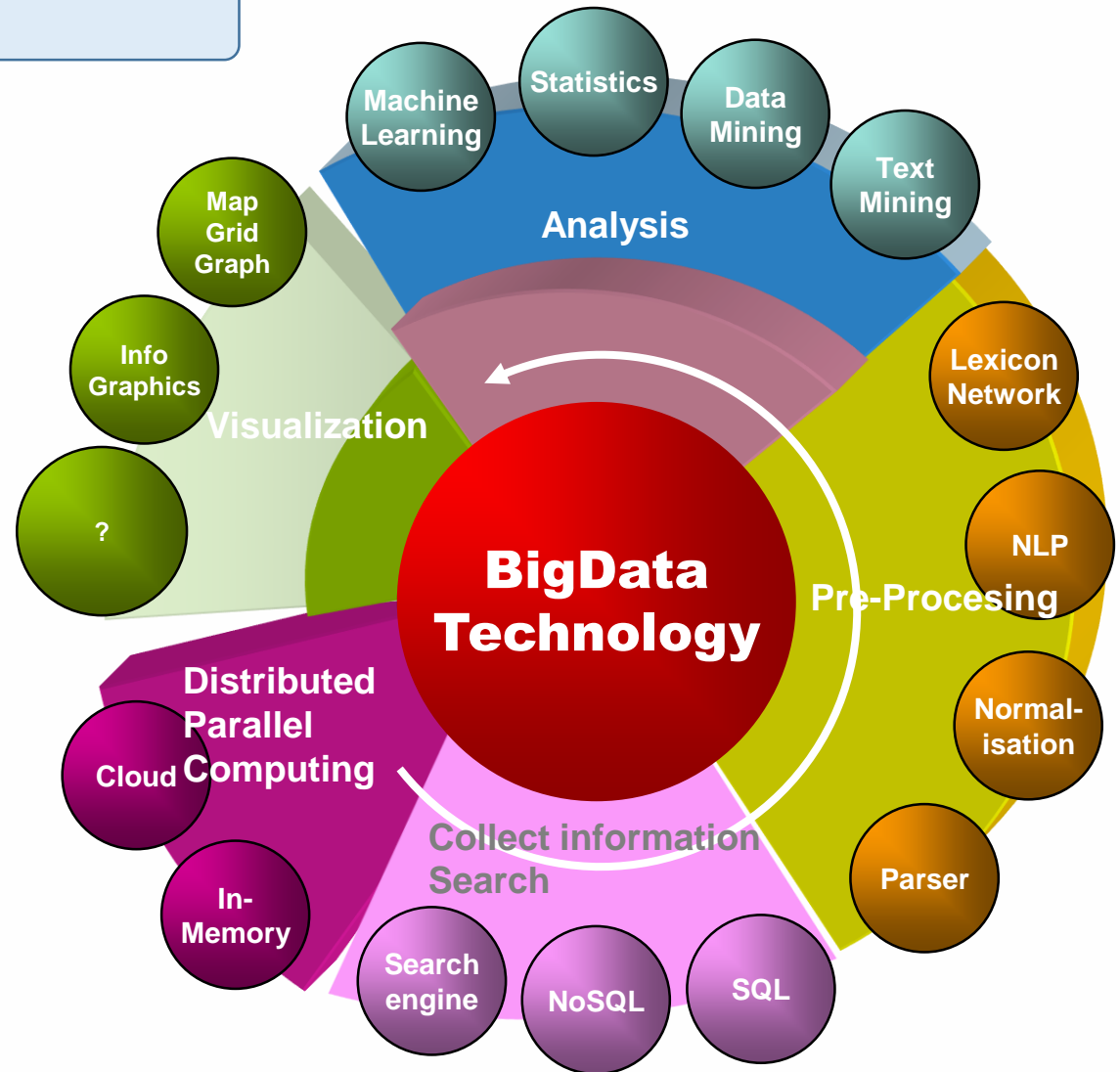


Source: Contents of above graphic created in partnership with Teradata, Inc.

## Big Data



## Big Data Technology



## Big Data Process Flow



## Big Data를 위한 역할과 요구기술

도메인 전문가

- 추천 로직 기획, 광고 플랫폼
- Financial & Stock Market
- Health Care
- BioInfomatics
- Power Management

데이터 분석가

- 데이터 수집
- 마이닝 알고리즘 & ML 구현
- 데이터 처리 엔진 구현
- 데이터 저장소 최적화
- 분산 알고리즘 구현

S/W 엔지니어

- 통계 & 데이터 탐색
- 데이터 마이닝 & 기계학습
- 데이터 분석
- 리포팅
- 데이터 시각화

System 엔지니어

- 운영 체계 최적화
- 컴퓨팅 H/W, N/W 최적화

- Visualization
- Infograph
- IR & RecSys

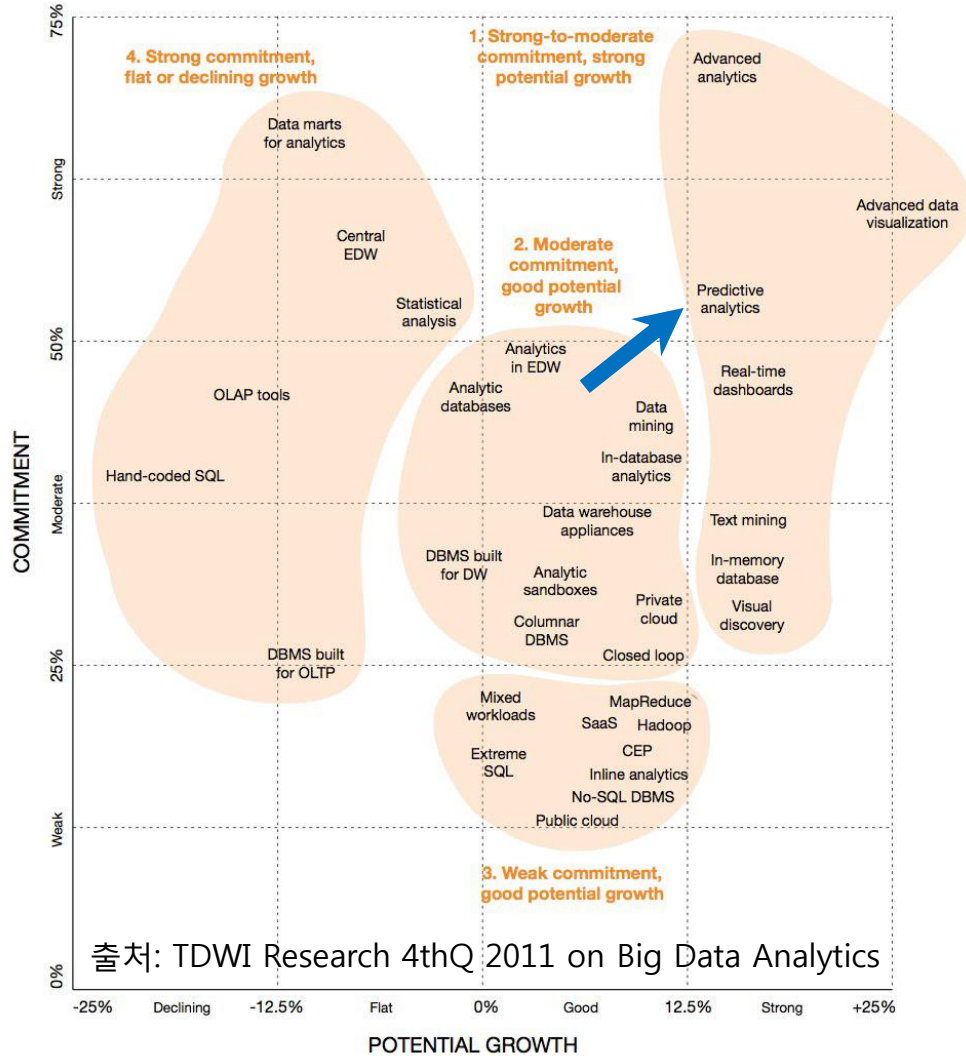
- OLAP Tools
- SAS, SPSS, R
- SQL
- RDBMS
- ETL
- Script Language
- Pig, Hive
- MapReduce

- Log Aggregator
- NoSQL
- Hadoop
- Linux
- X86
- Network



# 1.2 Big Data의 발전 방향

## Big Data의 발전 방향

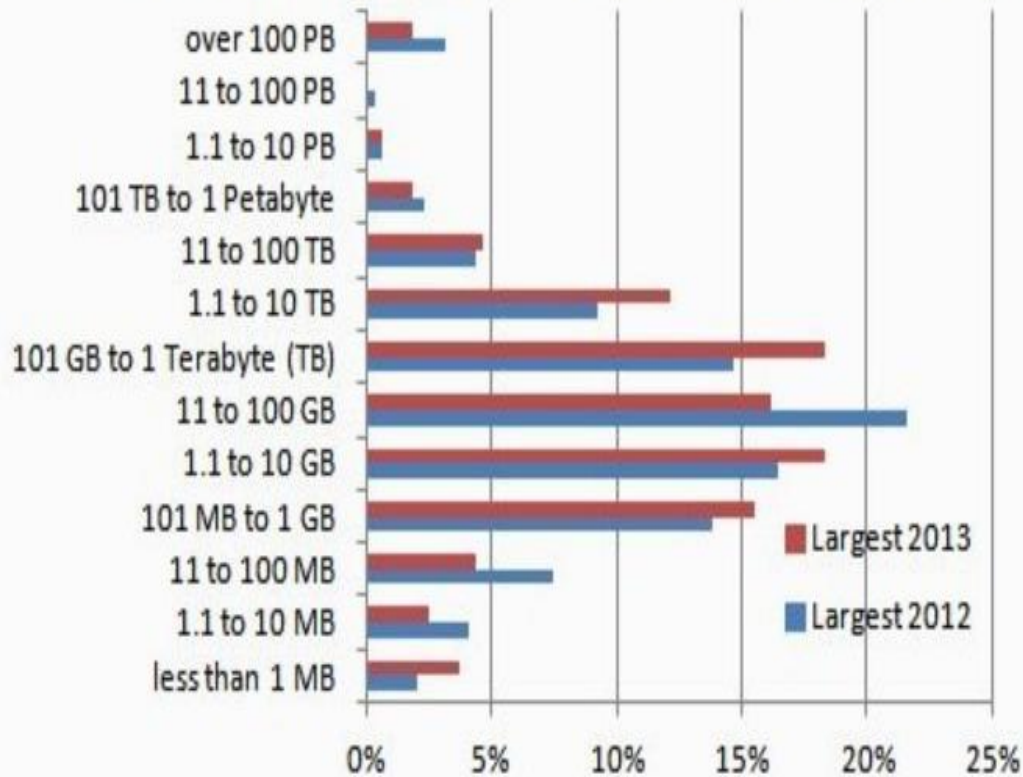


Real-time Analytics  
 Advanced & Predictive Analytics  
 Advanced Data Visualization



## 분석 정보 규모

### 2013 Largest Database Analyze/Data Mined









### 지역별 분석 정보 규모

Region (voters)	Largest Dataset Analyzed (median)	% analyzed TB+ data
US/Canada (156)	30-50 GB	28%
Europe (92)	5-10 GB	12%
Asia (47)	1-5 GB	11%
Latin America (12)	11-100 GB	17%
AU/New Zealand (7)	5-10 GB	14%

출처 : [KDnuggets Home](#) » [Polls](#) » Algorithms for Data Mining (Nov 2011)

## Big Data의 활용

Did you use analytics in the cloud, Hadoop, EC2, etc in 2011?	
Yes	 14%
No	 86%

Employment type:	Percent all	Avg Num Algorithms
Industry analyst/consultant (172)	 55.3%	6.3
Academic researcher (85)	 27.3%	5.1
Student (37)	 11.9%	4.3
Government/Other (17)	 5.5%	5.0

Regional breakdown is

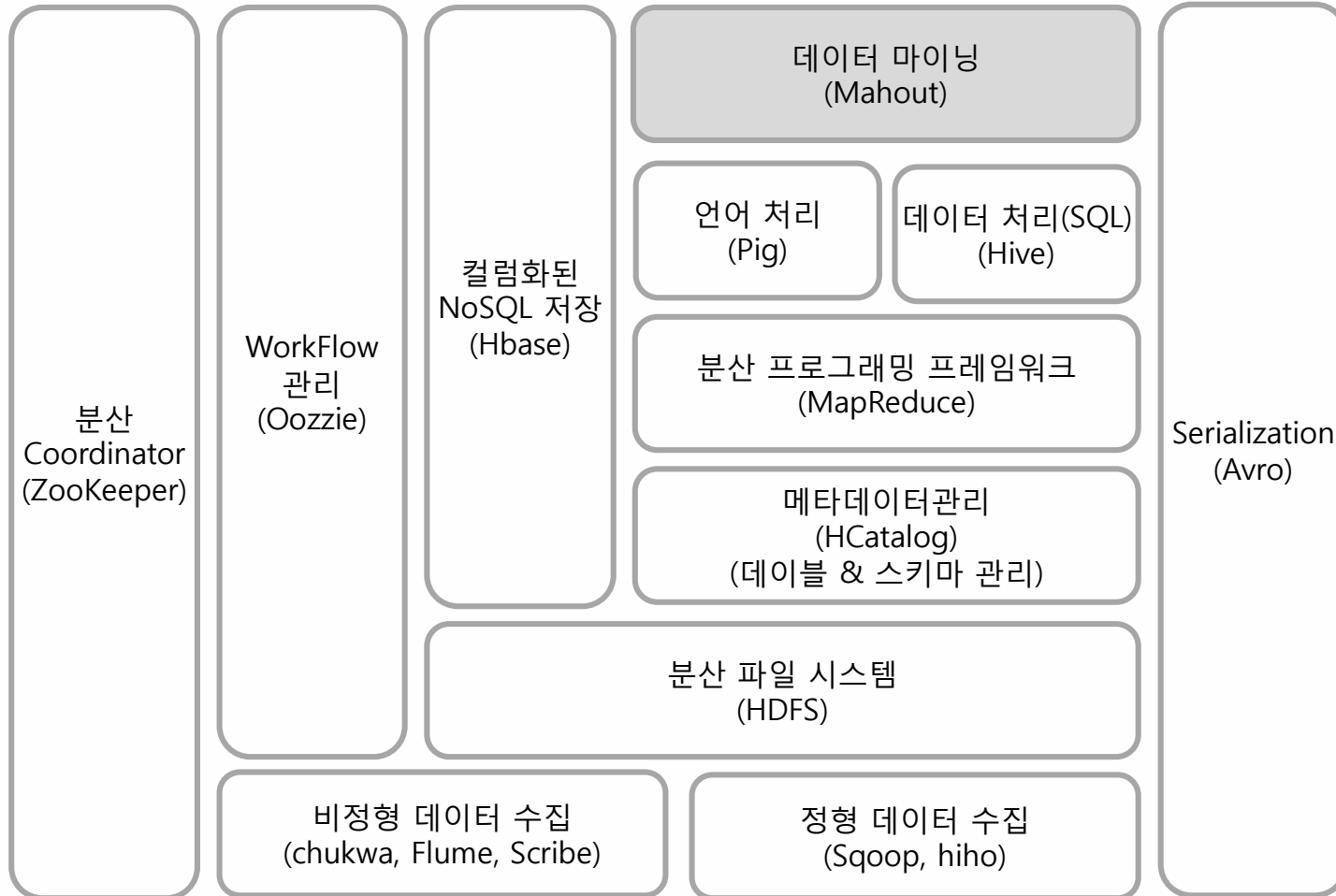
1. US/Canada, 40.2%
2. Europe, 37.6%
3. Asia, 10.3%
4. Latin America, 5.8%
5. Africa/Middle East, 3.2%
6. Australia/NZ 2.9%

## Algorithm

Which methods/algorithms did you use for data analysis in 2011? [311 voters]	
Decision Trees/Rules (186)	59.8 %
Regression (180)	57.9 %
Clustering (163)	52.4 %
Statistics (descriptive) (149)	47.9 %
Visualization (119)	38.3 %
Time series/Sequence analysis (92)	29.6 %
Support Vector (SVM) (89)	28.6 %
Association rules (89)	28.6 %
Ensemble methods (88)	28.3 %
Text Mining (86)	27.7 %
Neural Nets (84)	27.0 %
Boosting (73)	23.5 %
Bayesian (68)	21.9 %
Bagging (63)	20.3 %
Factor Analysis (58)	18.7 %
Anomaly/Deviation detection (51)	16.4 %
Social Network Analysis (44)	14.2 %
Survival Analysis (29)	9.32 %
Genetic algorithms (29)	9.32 %
Uplift modeling (15)	4.82 %

Algorithm	Academic/Student	Industry / Gov
Uplift modeling		INF
Survival Analysis		2.47
Regression		2.00
Visualization		1.55
Statistics		1.54
Boosting		1.50
Time series/Sequence analysis		1.48
Bagging		1.39
Factor Analysis		1.32
Anomaly/Deviation detection		1.29
Text Mining		1.27
Decision Trees		1.20
Neural Nets		1.16
Clustering		1.14
Ensemble methods		1.08
Social Network Analysis	0.93	
Bayesian	0.92	
Association rules	0.83	
Support Vector -SVM	0.66	
Genetic algorithms	0.60	

### Apache Project



### Apache Frameworks and more...

#### ▪ Data storage (HDFS)

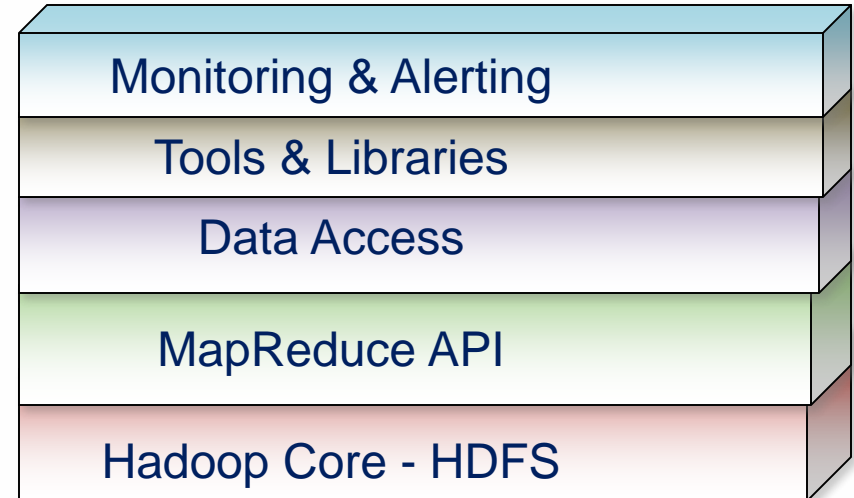
- Runs on commodity hardware (usually Linux)
- Horizontally scalable

#### ▪ Processing (MapReduce)

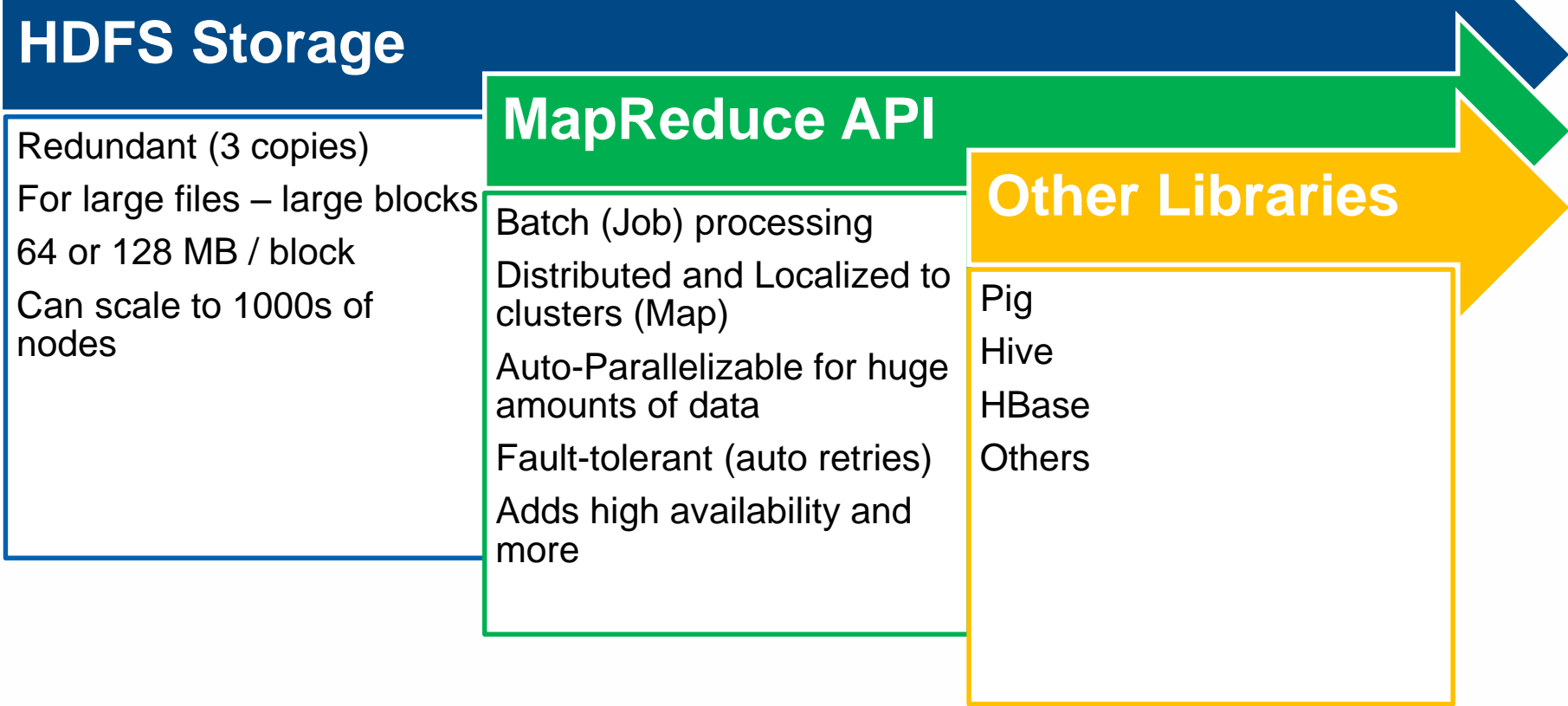
- Parallelized (scalable) processing
- Fault Tolerant

#### ▪ Other Tools / Frameworks

- Data Access
  - HBase, Hive, Pig, Mahout
- Tools
  - Hue, Sqoop
- Monitoring
  - Greenplum, Cloudera



Hadoop distribution?



### Cluster HDFS (Physical) Storage

#### One Name Node

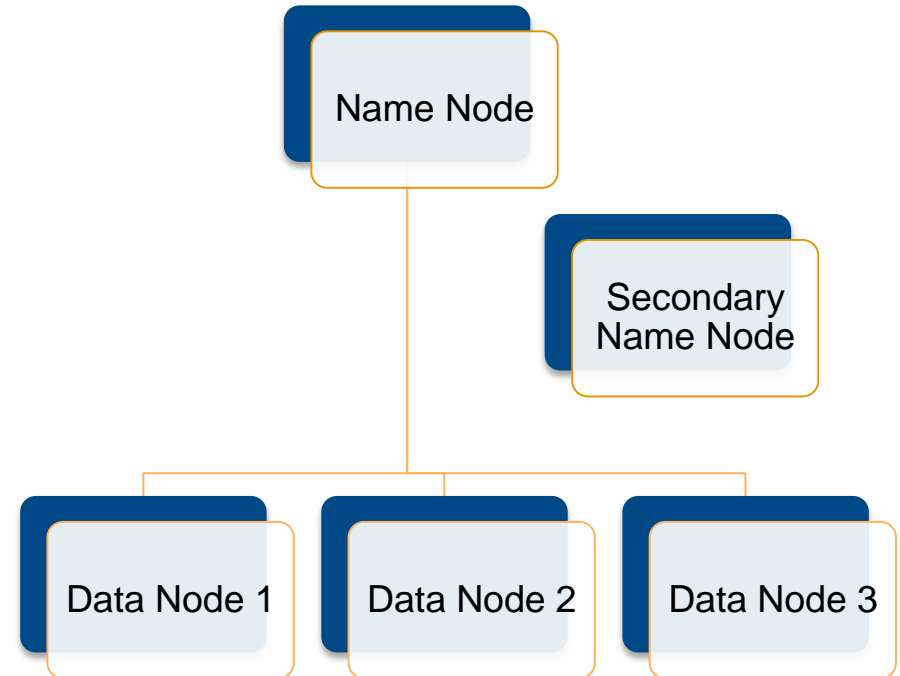
- Contains web site to view cluster information
- V2 Hadoop uses multiple Name Nodes for HA

#### Many Data Nodes

- 3 copies of each node by default

#### Work with data in HDFS

- Using common Linux shell commands
- Block size is 64 or 128 MB



## MapReduce Job – Logical View

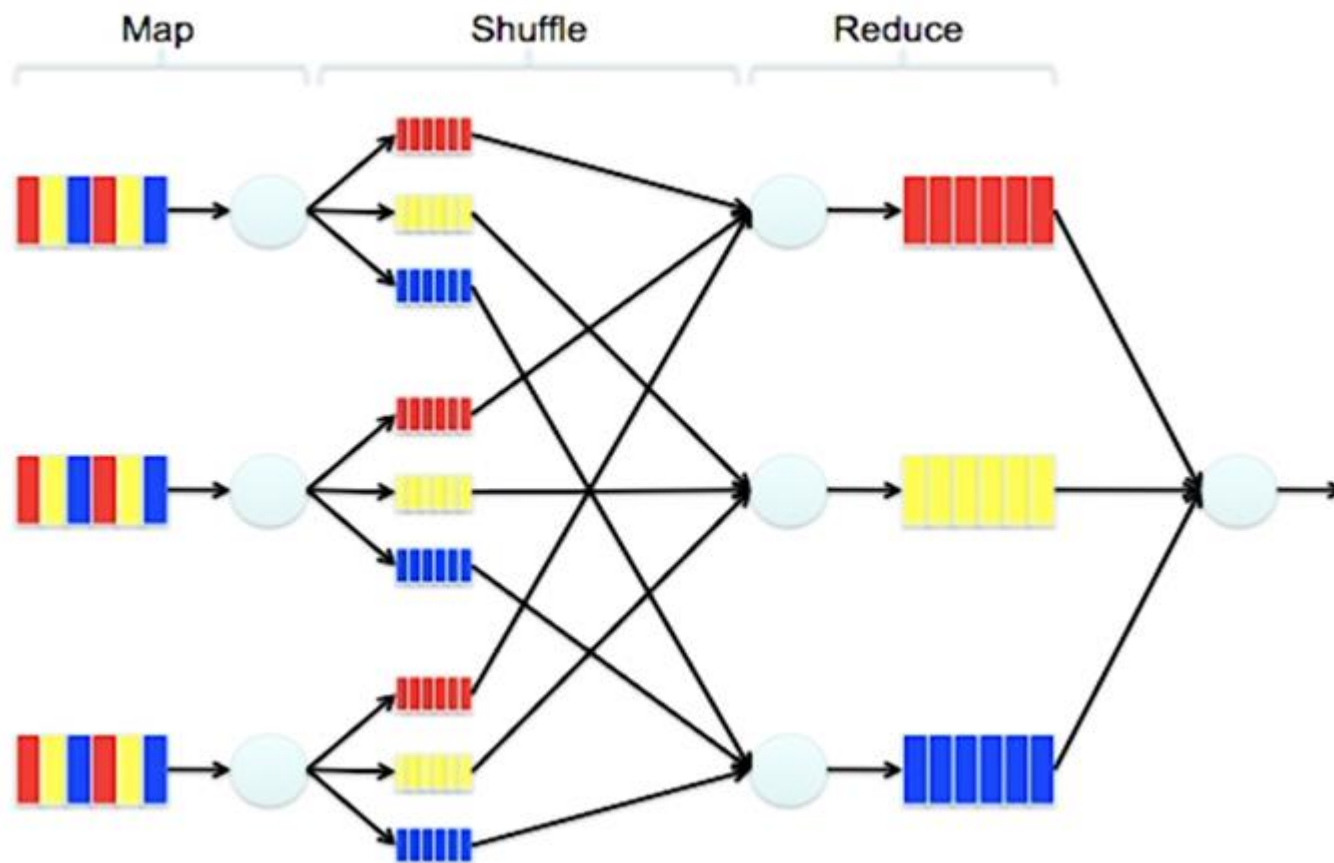
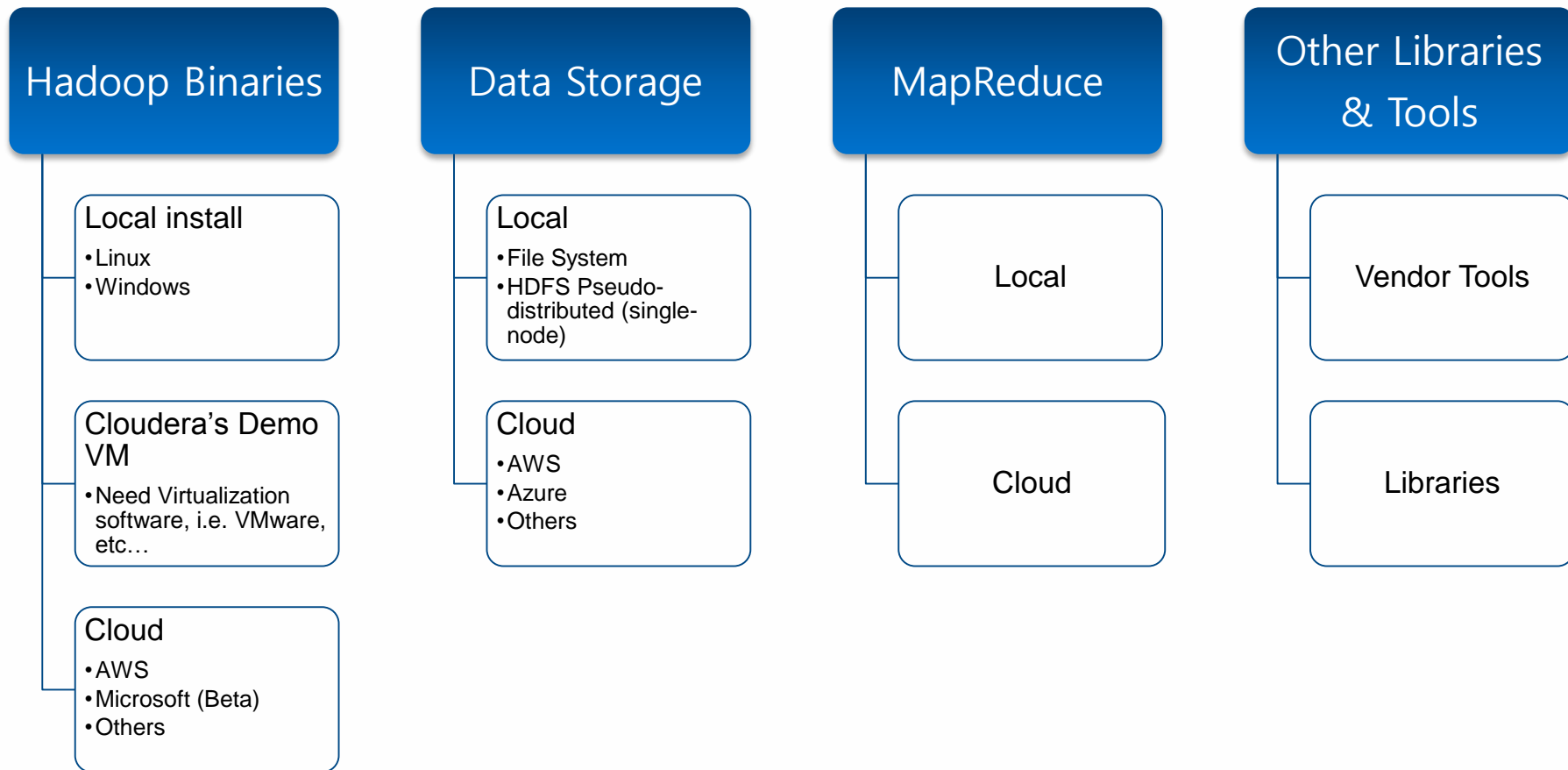


Image from - <http://mm-tom.s3.amazonaws.com/blog/MapReduce.png>



## Setting up Hadoop Development



### Common Data Sources



#### **Text Files – i.e. log files**

- Semi-structured
- Unstructured



#### **Statistical information – piles of numbers, often scientific sources**



#### **Geospatial information – i.e. cell phone activity**

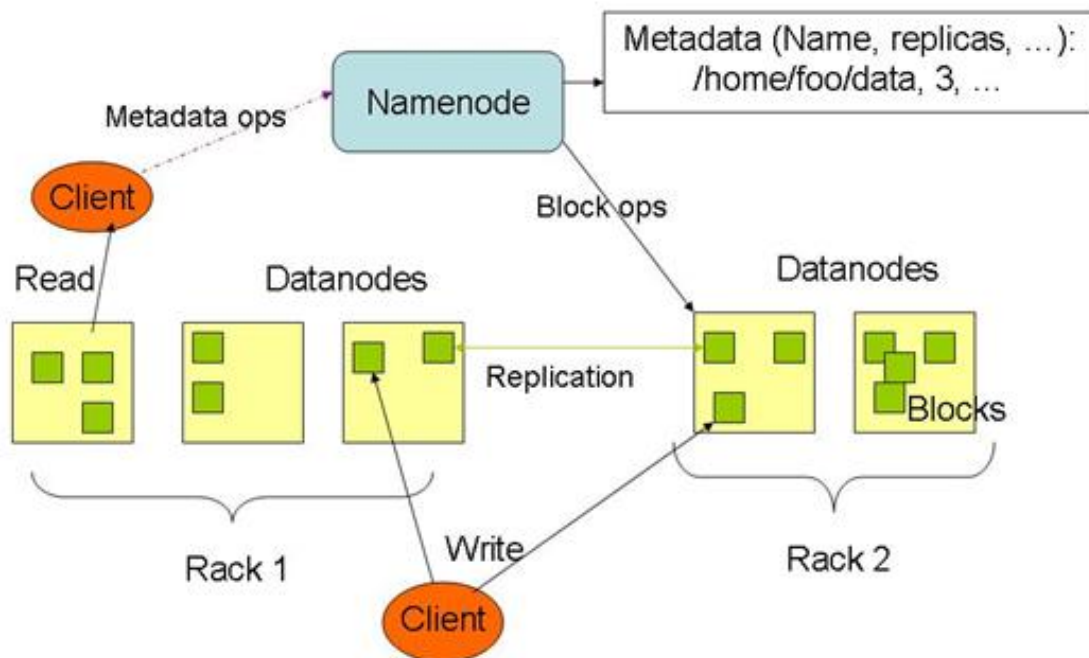


#### **Clickstream – advertising, website traversals**

## Hadoop Distributed File System

Hadoop Distributed File System (HDFS™) is the primary storage system used by Hadoop applications. HDFS creates multiple replicas of data blocks and distributes them on compute nodes throughout a cluster to enable reliable, extremely rapid computations.

### HDFS Architecture



**Single Namespace for entire cluster**

**Data Coherency**

- Write-once-read-many access model
- Client can only append to existing files

**Files are broken up into blocks**

- Typically 128 MB block size
- Each block replicated on multiple DataNodes

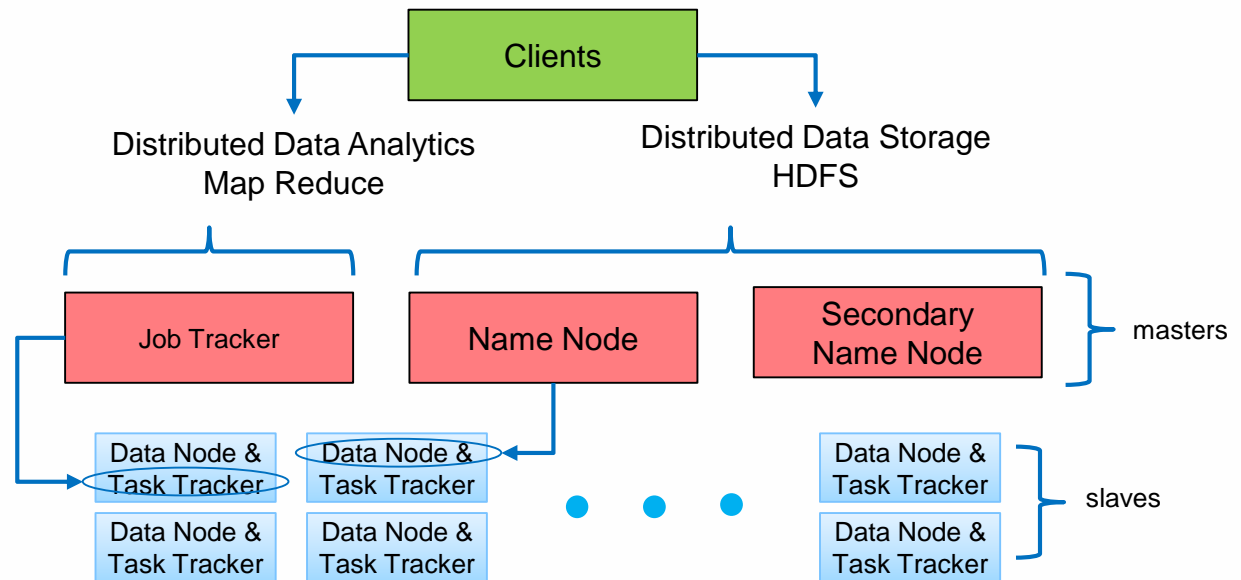
**Intelligent Client**

- Client can find location of blocks
- Client accesses data directly from DataNode

## Building Blocks of Hadoop

A fully configured cluster, “running Hadoop” means running a set of daemons, or resident programs, on the different servers in your network. These daemons have specific roles; some exist only on one server, some exist across multiple servers.

### Hadoop Server Roles

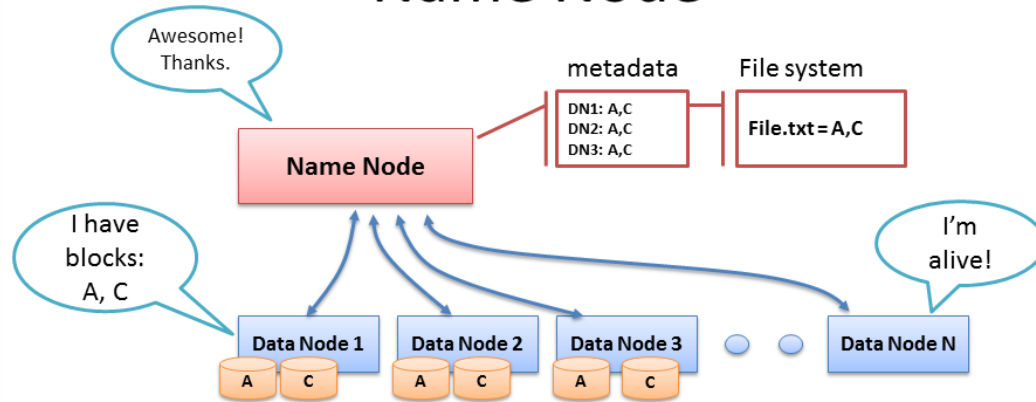


The daemons include

- NameNode
- Secondary NameNode
- DataNode
- JobTracker
- TaskTracker

### NameNode

### Name Node



- Data Node sends Heartbeats
- Every 10<sup>th</sup> heartbeat is a Block report
- Name Node builds metadata from Block reports
- TCP – every 3 seconds
- If Name Node is down, HDFS is down

BRAD HEDLUND .com

The most vital of the Hadoop daemons—the NameNode .Hadoop employs a master/slave architecture for both distributed storage and distributed computation.

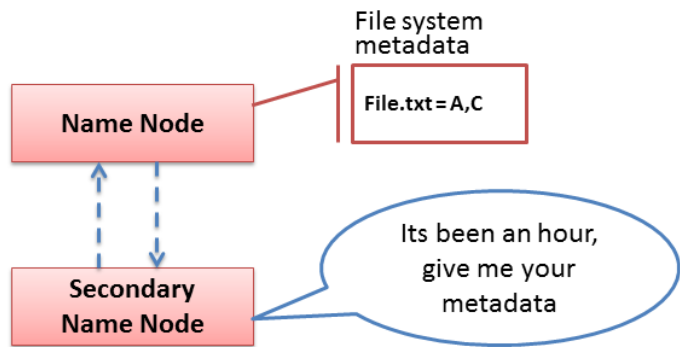
The distributed storage system is called the *Hadoop File System*, or HDFS. The NameNode is the master of HDFS that directs the slave DataNode daemons to perform the low-level I/O tasks.

The NameNode is the bookkeeper of HDFS; it keeps track of how your files are broken down into file blocks, which nodes store those blocks, and the overall health of the distributed filesystem.

The function of the NameNode is memory and I/O intensive. As such, the server hosting the NameNode typically doesn't store any user data or perform any computations for a MapReduce program to lower the workload on the machine

## Secondary NameNode

### Secondary Name Node

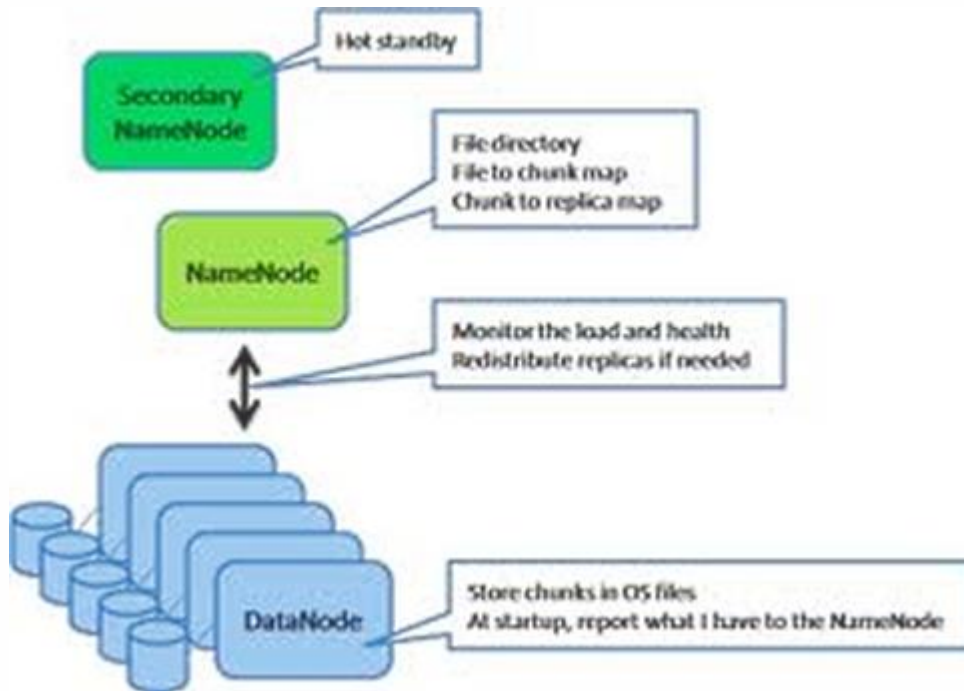


- Not a hot standby for the Name Node
- Connects to Name Node every hour\*
- Housekeeping, backup of Name Node metadata
- Saved metadata can rebuild a failed Name Node

BRAD HEDLUND .com

- The Secondary NameNode (SNN) is an assistant daemon for monitoring the state of the cluster HDFS. Like the NameNode, each cluster has one SNN, and it typically resides on its own machine as well. No other DataNode or TaskTracker daemons run on the same server.
- The SNN differs from the NameNode in that this process doesn't receive or record any real-time changes to HDFS. Instead, it communicates with the NameNode to take snapshots of the HDFS metadata at intervals defined by the cluster configuration.
- As mentioned earlier, the NameNode is a single point of failure for a Hadoop cluster, and the SNN snapshots help minimize the downtime and loss of data

## DataNode

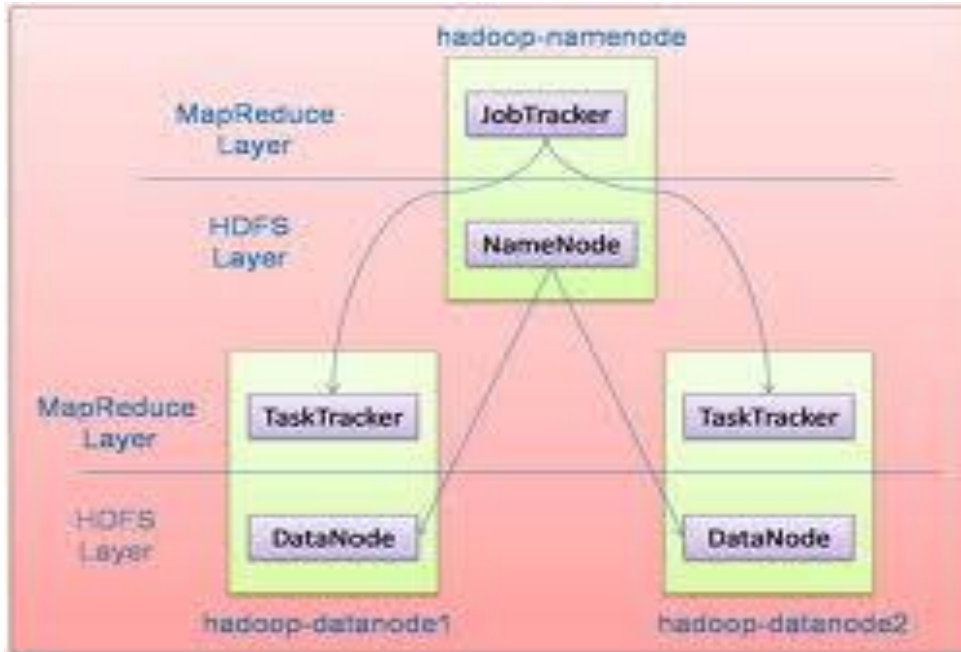


### ➤ *DataNode*

Each slave machine in your cluster will host a DataNode daemon to perform the grunt work of the distributed filesystem—reading and writing HDFS blocks to actual files on the local filesystem. When you want to read or write a HDFS file, the file is broken into blocks and the NameNode will tell your client which DataNode each block resides in.

Your client communicates directly with the DataNode daemons to process the local files corresponding to the blocks. Furthermore, a DataNode may communicate with other DataNodes to replicate its data blocks for redundancy.

### Trackers



#### ➤ **JobTracker**

The JobTracker daemon is the liaison between your application and Hadoop. Once you submit your code to your cluster, the JobTracker determines the execution plan by determining which files to process, assigns nodes to different tasks, and monitors all tasks as they're running.

Should a task fail, the JobTracker will automatically re-launch the task, possibly on a different node, up to a predefined limit of retries. There is only one JobTracker daemon per Hadoop cluster. It's typically run on a server as a master node of the cluster.

#### ➤ **TaskTracker**

As with the storage daemons, the computing daemons also follow a master/slave architecture: the JobTracker is the master overseeing the overall execution of a MapReduce job and the TaskTrackers manage the execution of individual tasks on each slave node.

Each TaskTracker is responsible for executing the individual tasks that the JobTracker assigns. Although there is a single TaskTracker per slave node, each TaskTracker can spawn multiple JVMs to handle many map or reduce tasks in parallel. One responsibility of the TaskTracker is to constantly communicate with the JobTracker. If the JobTracker fails to receive a heartbeat from a TaskTracker within a specified amount of time, it will assume the TaskTracker has crashed and will resubmit the corresponding tasks to other nodes in the cluster.



## MapReduce Thinking

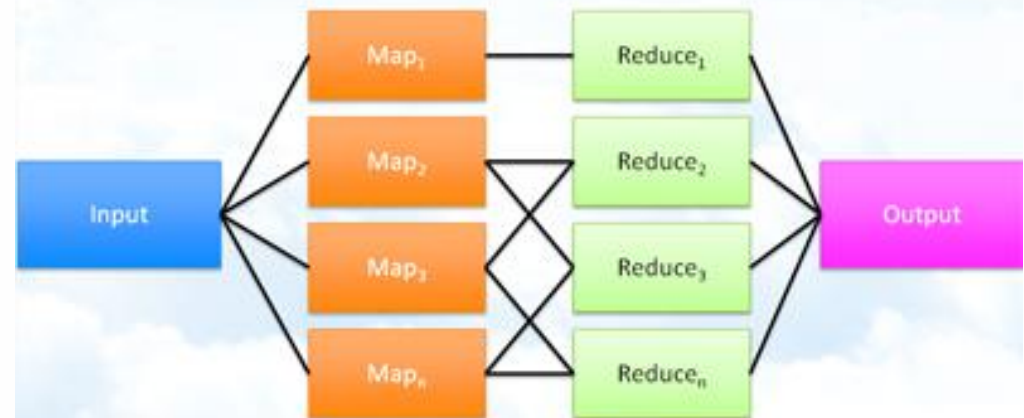
- MapReduce programs are designed to compute large volumes of data in a parallel fashion. This requires dividing the workload across a large number of machines.
- MapReduce programs transform lists of input data elements into lists of output data elements. A MapReduce program will do this twice, using two different list processing idioms: *map*, and *reduce*.

A MapReduce program processes data by manipulating (key/value) pairs in the general form

map:  $(K1, V1) \rightarrow list(K2, V2)$

reduce:  $(K2, list(V2)) \rightarrow list(K3, V3)$

### MapReduce Divides the Work



### Input

- **Input files** : This is where the data for a MapReduce task is initially stored. While this does not need to be the case, the input files typically reside in HDFS. The format of these files is arbitrary; while line-based log files can be used, we could also use a binary format, multi-line input records, or something else entirely. It is typical for these input files to be very large -- tens of gigabytes or more.
- **InputFormat** : How these input files are split up and read is defined by the InputFormat. An InputFormat is a class that provides the following functionality:
  - Selects the files or other objects that should be used for input
  - Defines the *InputSplits* that break a file into tasks
  - Provides a factory for *RecordReader* objects that read the file

Several InputFormats are provided with Hadoop. An abstract type is called *FileInputFormat*; all InputFormats that operate on files inherit functionality and properties from this class. When starting a Hadoop job, FileInputFormat is provided with a path containing files to read. The FileInputFormat will read all files in this directory. It then divides these files into one or more InputSplits each. You can choose which InputFormat to apply to your input files for a job by calling the `setInputFormat()` method of the *JobConf* object that defines the job. A table of standard InputFormats is given below.

InputFormat	Description	Key	Value
TextInputFormat	Default format; reads lines of text files	The byte offset of the line	The line contents
KeyValueInputFormat	Parses lines into key, val pairs	Everything up to the first tab character	The remainder of the line
SequenceFileInputFormat	A Hadoop-specific high-performance binary format	user-defined	user-defined

### Input Contd....

#### Input Splits:

An `InputSplit` describes a unit of work that comprises a single *map task* in a MapReduce program. A MapReduce program applied to a data set, collectively referred to as a *Job*, is made up of several (possibly several hundred) tasks. Map tasks may involve reading a whole file; they often involve reading only part of a file. By default, the `FileInputFormat` and its descendants break a file up into 64 MB chunks (the same size as blocks in HDFS). You can control this value by setting the `mapred.min.split.size` parameter in `hadoop-site.xml`, or by overriding the parameter in the `JobConf` object used to submit a particular MapReduce job.

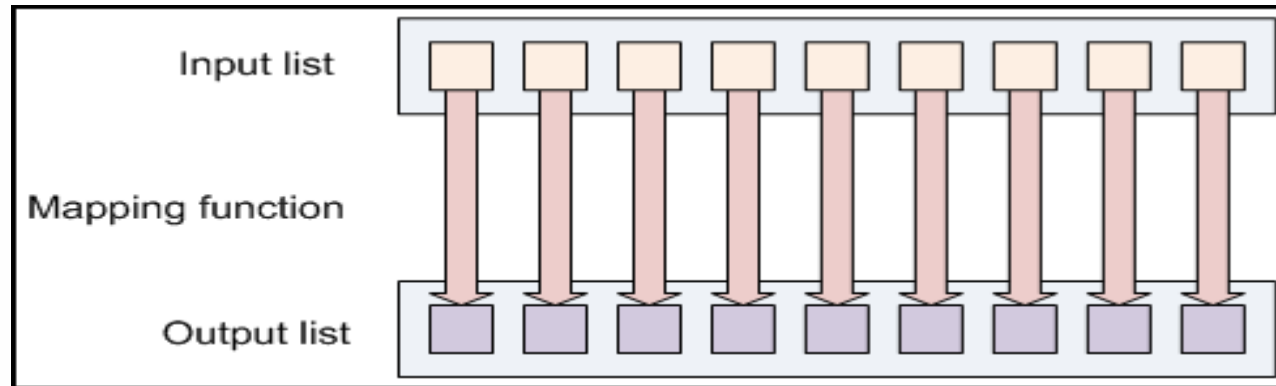
By processing a file in chunks, we allow several map tasks to operate on a single file in parallel. If the file is very large, this can improve performance significantly through parallelism. Even more importantly, since the various blocks that make up the file may be spread across several different nodes in the cluster, it allows tasks to be scheduled on each of these different nodes; the individual blocks are thus all processed locally, instead of needing to be transferred from one node to another. Of course, while log files can be processed in this piece-wise fashion, some file formats are not amenable to chunked processing. By writing a custom `InputFormat`, you can control how the file is broken up (or is not broken up) into splits.

The `InputFormat` defines the list of tasks that make up the mapping phase; each task corresponds to a single input split. The tasks are then assigned to the nodes in the system based on where the input file chunks are physically resident. An individual node may have several dozen tasks assigned to it. The node will begin working on the tasks, attempting to perform as many in parallel as it can. The on-node parallelism is controlled by the `mapred.tasktracker.map.tasks.maximum` parameter.

#### RecordReader:

The `InputSplit` has defined a slice of work, but does not describe how to access it. The `RecordReader` class actually loads the data from its source and converts it into (key, value) pairs suitable for reading by the Mapper. The `RecordReader` instance is defined by the `InputFormat`. The default `InputFormat`, `TextInputFormat`, provides a `LineRecordReader`, which treats each line of the input file as a new value. The key associated with each line is its byte offset in the file. The `RecordReader` is invoked repeatedly on the input until the entire `InputSplit` has been consumed. Each invocation of the `RecordReader` leads to another call to the `map()` method of the Mapper.

### Mapper



The Mapper performs the interesting user-defined work of the first phase of the MapReduce program. Given a key and a value, the `map()` method emits (key, value) pair(s) which are forwarded to the Reducers. A new instance of Mapper is instantiated in a separate Java process for each map task (InputSplit) that makes up part of the total job input. The individual mappers are intentionally not provided with a mechanism to communicate with one another in any way. This allows the reliability of each map task to be governed solely by the reliability of the local machine. The `map()` method receives two parameters in addition to the key and the value:

The **Context** object has a method named `write()` which will forward a (key, value) pair to the reduce phase of the job.

The Mapper interface is responsible for the data processing step. Its single method is to process an individual (key/value) pair:

```
public void map(K1 key, V1 value, Context context) throws IOException
```

### In Between Phases

- **Partition & Shuffle:**

After the first map tasks have completed, the nodes may still be performing several more map tasks each. But they also begin exchanging the intermediate outputs from the map tasks to where they are required by the reducers. This process of moving map outputs to the reducers is known as *shuffling*.

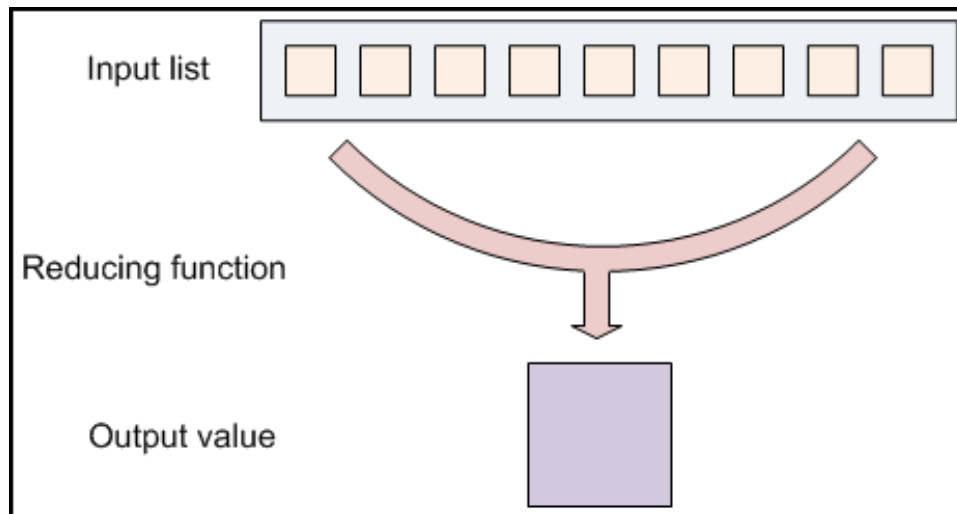
A different subset of the intermediate key space is assigned to each reduce node; these subsets (known as "partitions") are the inputs to the reduce tasks. Each map task may emit (key, value) pairs to any partition; all values for the same key are always reduced together regardless of which mapper is its origin.

Therefore, the map nodes must all agree on where to send the different pieces of the intermediate data. The *Partitioner* class determines which partition a given (key, value) pair will go to. The default partitioner computes a hash value for the key and assigns the partition based on this result.

- **Sort:**

Each reduce task is responsible for reducing the values associated with several intermediate keys. The set of intermediate keys on a single node is automatically sorted by Hadoop before they are presented to the Reducer.

## Reducer

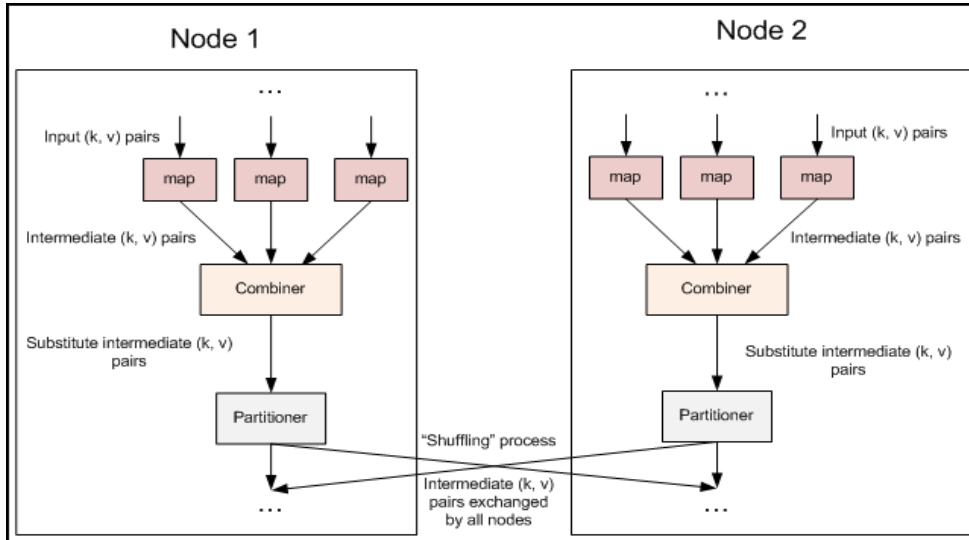


A Reducer instance is created for each reduce task. This is an instance of user-provided code that performs the second important phase of job-specific work. For each key in the partition assigned to a Reducer, the Reducer's `reduce()` method is called once. This receives a key as well as an iterator over all the values associated with the key. The values associated with a key are returned by the iterator in an undefined order.

The Reducer also receives the Context object; that is used to write the output in the same manner as in the `map()` method.

```
void reduce(K2 key, Iterable <V2> values, Context context) throws IOException
```

## Combiner



**Combiner:** The pipeline showed earlier omits a processing step which can be used for optimizing bandwidth usage by your MapReduce job. Called the *Combiner*, this pass runs after the Mapper and before the Reducer. Usage of the Combiner is optional. If this pass is suitable for your job, instances of the Combiner class are run on every node that has run map tasks. The Combiner will receive as input all data emitted by the Mapper instances on a given node. The output from the Combiner is then sent to the Reducers, instead of the output from the Mappers. The Combiner is a **"mini-reduce" process which operates only on data generated by one machine.**

### Example

Word count is a prime example for where a Combiner is useful. The Word Count program emits a (*word*, 1) pair for every instance of every word it sees. So if the same document contains the word "cat" 3 times, the pair ("cat", 1) is emitted three times; all of these are then sent to the Reducer. By using a Combiner, these can be condensed into a single ("cat", 3) pair to be sent to the Reducer.

Now each node only sends a single value to the reducer for each word -- drastically reducing the total bandwidth required for the shuffle process, and speeding up the job. The best part of all is that we do not need to write any additional code to take advantage of this! If a reduce function is both *commutative* and *associative*, then it can be used as a Combiner as well. You can enable combining in the word count program by adding the following line to the driver:

```
conf.setCombinerClass(Reducer.class);
```

The Combiner should be an instance of the *Reducer* interface. If your Reducer itself cannot be used directly as a Combiner because of commutativity or associativity, you might still be able to write a third class to use as a Combiner for your job

### Output

**OutputFormat**: The (key, value) pairs provided to this OutputCollector are then written to output files. The way they are written is governed by the *OutputFormat*. The OutputFormat functions much like the InputFormat class described earlier. The instances of OutputFormat provided by Hadoop write to files on the local disk or in HDFS; they all inherit from a common *FileOutputFormat*. Each Reducer writes a separate file in a common output directory. These files will typically be named part-*nnnnn*, where *nnnnn* is the partition id associated with the reduce task. The output directory is set by the *FileOutputFormat.setOutputPath()* method. You can control which particular OutputFormat is used by calling the *setOutputFormat()* method of the *JobConf* object that defines your MapReduce job.

A table of provided OutputFormats is given below.

OutputFormat:	Description
TextOutputFormat	Default; writes lines in "key \t value" form
SequenceFileOutputFormat	Writes binary files suitable for reading into subsequent MapReduce jobs
NullOutputFormat	Disregards its inputs

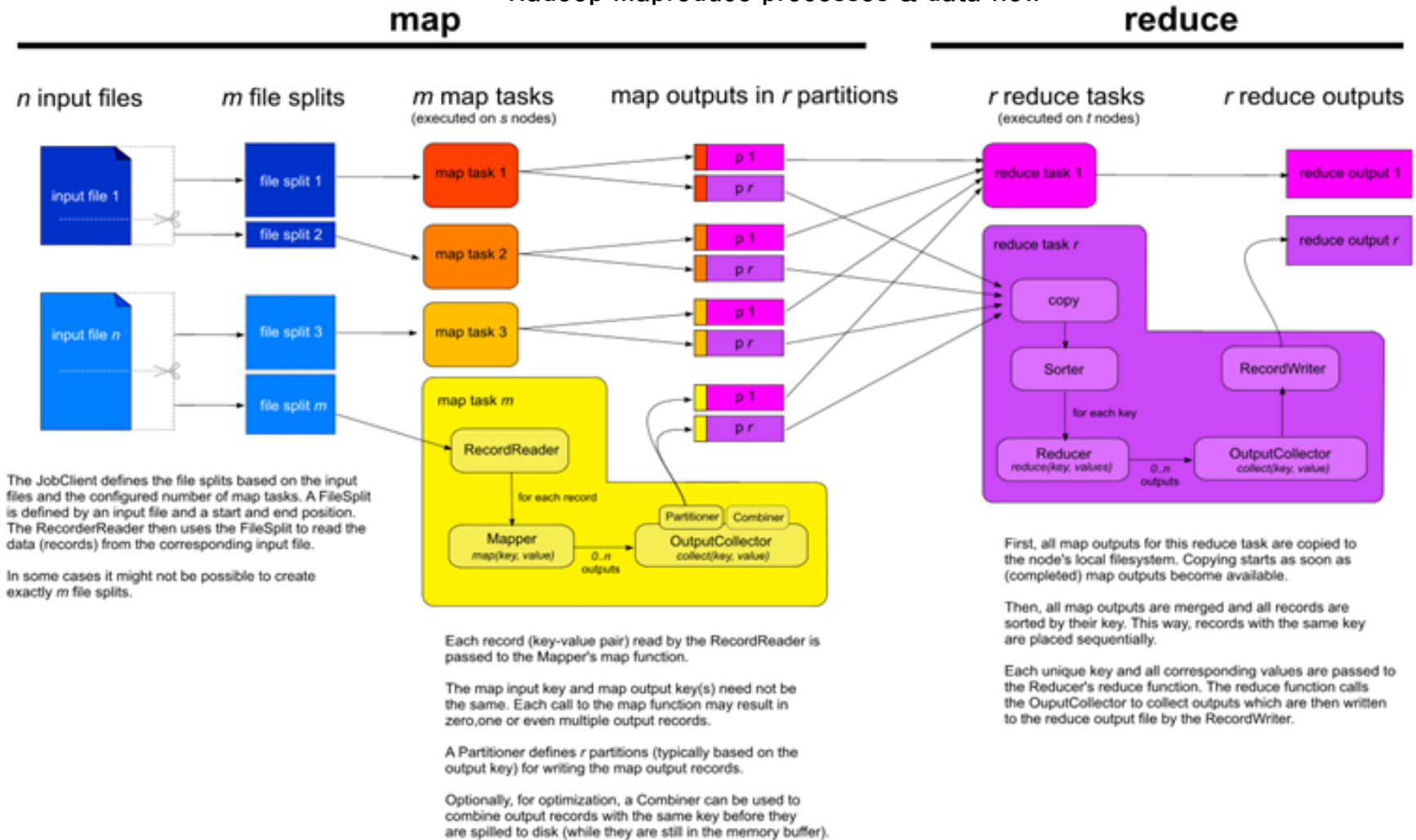
Hadoop provides some OutputFormat instances to write to files. The basic (default) instance is *TextOutputFormat*, which writes (key, value) pairs on individual lines of a text file. This can be easily re-read by a later MapReduce task using the *KeyValueInputFormat* class, and is also human-readable. A better intermediate format for use between MapReduce jobs is the *SequenceFileOutputFormat* which rapidly serializes arbitrary data types to the file; the corresponding *SequenceFileInputFormat* will deserialize the file into the same types and presents the data to the next Mapper in the same manner as it was emitted by the previous Reducer. The *NullOutputFormat* generates no output files and disregards any (key, value) pairs passed to it by the OutputCollector. This is useful if you are explicitly writing your own output files in the *reduce()* method, and do not want additional empty output files generated by the Hadoop framework.

- **RecordWriter**: Much like how the InputFormat actually reads individual records through the RecordReader implementation, the OutputFormat class is a factory for *RecordWriter* objects; these are used to write the individual records to the files as directed by the OutputFormat. The **output files** written by the Reducers are then left in HDFS for your use, either by another MapReduce job, a separate program, for for human inspection.

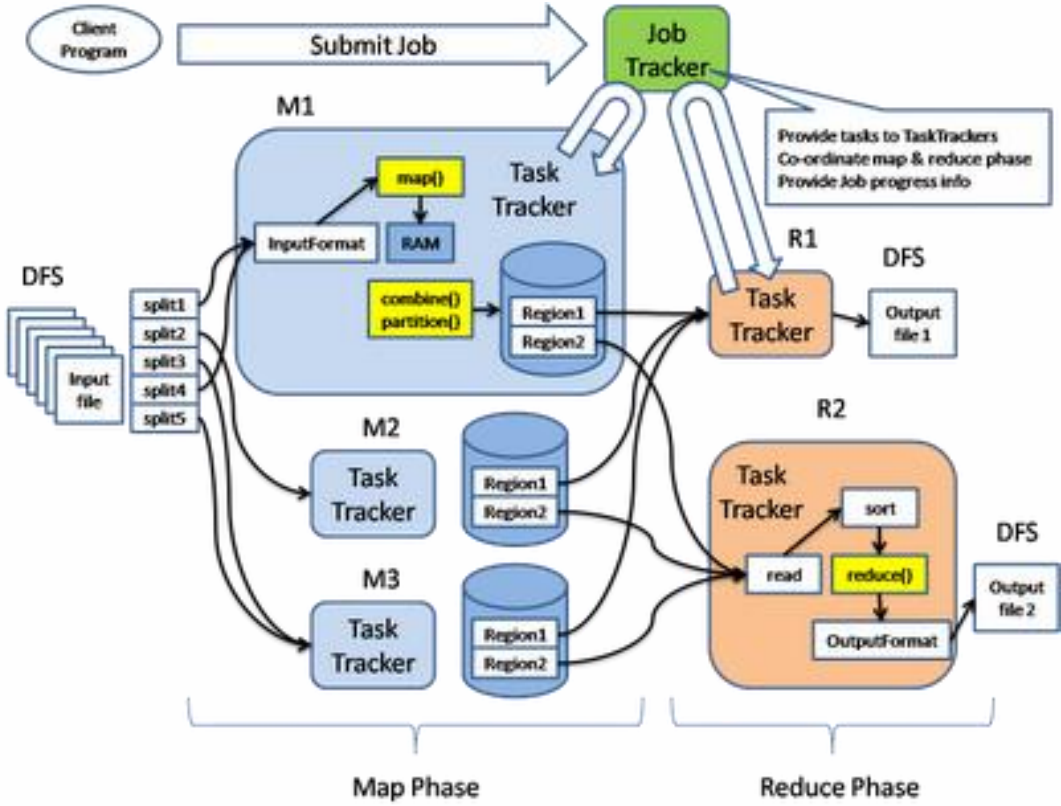


## Hadoop Mapreduce

Hadoop Mapreduce processes & data flow



## Job Execution



Hadoop MapRed is based on a “pull” model where multiple “TaskTrackers” poll the “JobTracker” for tasks (either map task or reduce task).

The job execution starts when the client program uploading three files: “job.xml” (the job config including map, combine, reduce function and input/output data path, etc.), “job.split” (specifies how many splits and range based on dividing files into ~16 – 64 MB size), “job.jar” (the actual Mapper and Reducer implementation classes) to the HDFS location (specified by the “mapred.system.dir” property in the “hadoop-default.conf” file).

Then the client program notifies the JobTracker about the Job submission. The JobTracker returns a Job id to the client program and starts allocating map tasks to the idle TaskTrackers when they poll for tasks. Each TaskTracker has a defined number of “task slots” based on the capacity of the machine. There are heartbeat protocol allows the JobTracker to know how many free slots from each TaskTracker. The JobTracker will determine appropriate jobs for the TaskTrackers based on how busy they are, their network proximity to the data sources (preferring same node, then same rack, then same network switch).

The assigned TaskTrackers will fork a MapTask (separate JVM process) to execute the map phase processing. The MapTask extracts the input data from the splits by using the “RecordReader” and “InputFormat” and it invokes the user provided “map” function which emits a number of key/value pair in the memory buffer.

## Job Execution contd ...

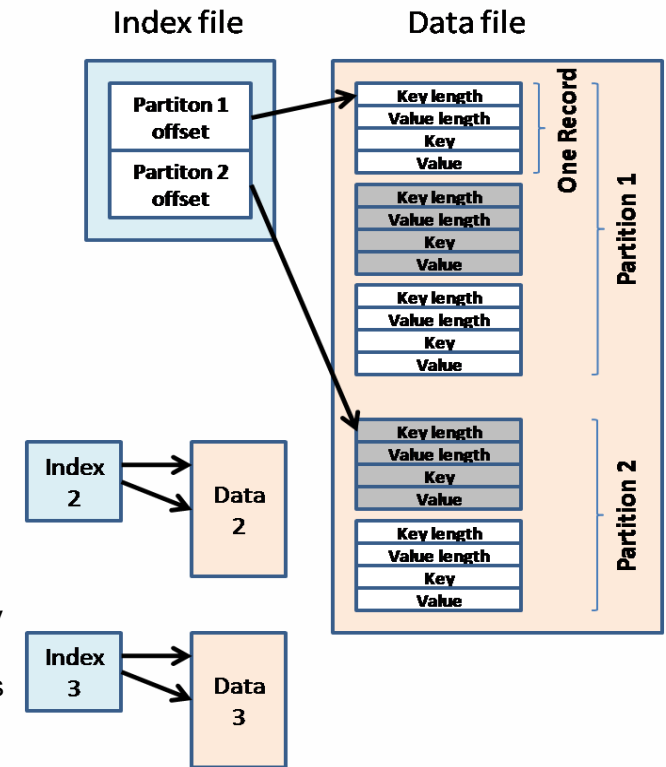
When the buffer is full, the output collector will spill the memory buffer into disk. For optimizing the network bandwidth, an optional “combine” function can be invoked to partially reduce values of each key. Afterwards, the “partition” function is invoked on each key to calculate its reducer node index.

The memory buffer is eventually flushed into 2 files, the first index file contains an offset pointer of each partition. The second data file contains all records sorted by partition and then by key.

When the map task has finished executing all input records, it start the commit process, it first flush the in-memory buffer (even it is not full) to the index + data file pair. Then a merge sort for all index + data file pairs will be performed to create a single index + data file pair.

The index + data file pair will then be splitted into are R local directories, one for each partition. After all the MapTask completes (all splits are done), the TaskTracker will notify the JobTracker which keeps track of the overall progress of job. JobTracker also provide a web interface for viewing the job status.

When the JobTracker notices that some map tasks are completed, it will start allocating reduce tasks to subsequent polling TaskTrackers (there are R TaskTrackers will be allocated for reduce task). These allocated TaskTrackers remotely download the region files (according to the assigned reducer index) from the completed map phase nodes and concatenate (merge sort) them into a single file. Whenever more map tasks are completed afterwards, JobTracker will notify these allocated TaskTrackers to download more region files (merge with previous file). In this manner, downloading region files are interleaved with the map task progress. The reduce phase is not started at this moment yet.



Eventually all the map tasks are completed. The JobTracker then notifies all the allocated TaskTrackers to proceed to the reduce phase. Each allocated TaskTracker will fork a ReduceTask (separate JVM) to read the downloaded file (which is already sorted by key) and invoke the “reduce” function, which collects the key/aggregatedValue into the final output file (one per reducer node). Note that each reduce task (and map task as well) is single-threaded. And this thread will invoke the reduce(key, values) function in assending (or descending) order of the keys assigned to this reduce task. This provides an interesting property that all entries written by the reduce() function is sorted in increasing order. The output of each reducer is written to a temp output file in HDFS. When the reducer finishes processing all keys, the temp output file will be renamed atomically to its final output filename.

## MapReduce Example – WordCount

The overall MapReduce word count process

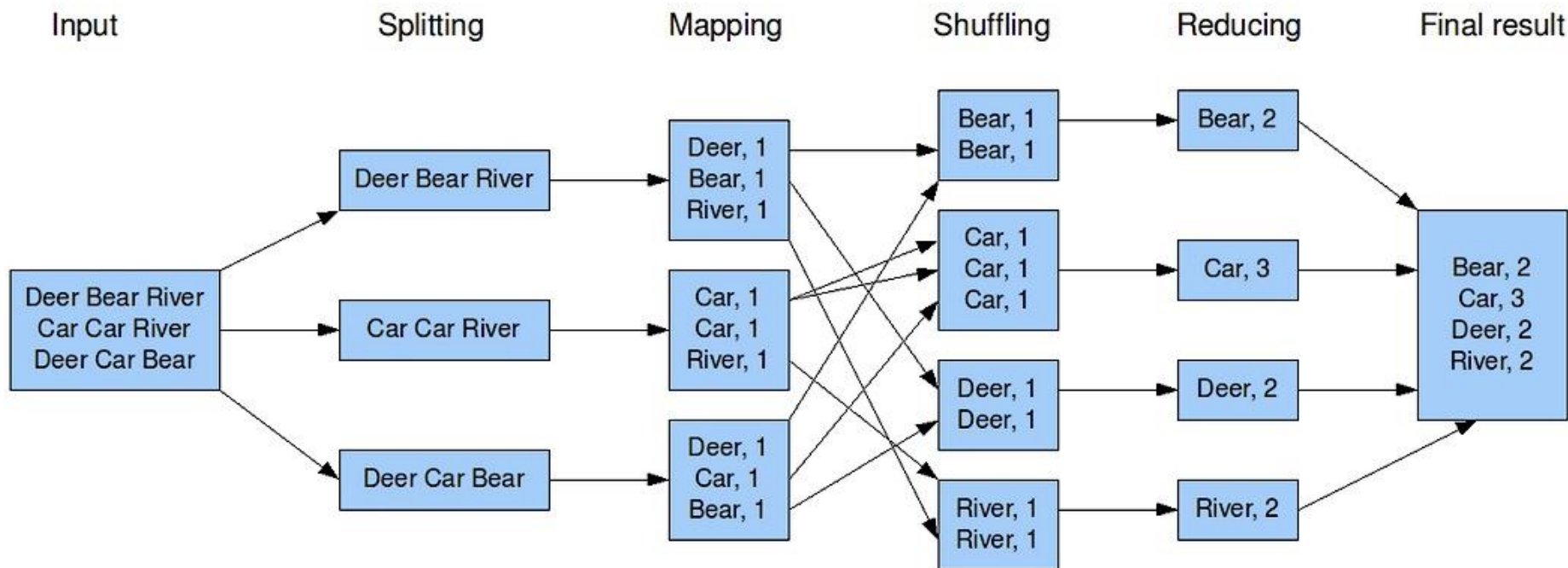


Image from: <http://blog.jteam.nl/wp-content/uploads/2009/08/MapReduceWordCountOverview1.png>

### Classification

- Logistic Regression (SGD)
- Bayesian
- Support Vector Machines (SVM) (open)
- Perceptron and Winnow (open)
- Neural Network (open)
- Random Forests (integrated)
- Restricted Boltzmann Machines (open)
- Online Passive Aggressive (integrated)
- Boosting (awaiting patch commit)
- Hidden Markov Models (HMM)
  - ✓ Training is done in Map-Reduce

### Clustering

- Canopy Clustering (integrated)
- K-Means Clustering (integrated)
- Fuzzy K-Means (integrated)
- Expectation Maximization (EM)
- Mean Shift Clustering (integrated)
- Hierarchical Clustering
- Dirichlet Process Clustering (integrated)
- Latent Dirichlet Allocation (integrated)
- Spectral Clustering (integrated)
- Minhash Clustering (integrated)
- Top Down Clustering (integrated)

## Pattern Mining

- Parallel FP Growth Algorithm
  - ✓ Also known as Frequent Itemset mining
  - ✓ use Map-Reduce

## Regression

- Locally Weighted Linear Regression (open)

## Dimension reduction

- Principal Components Analysis (PCA) (open)
- Independent Component Analysis (open)
- Gaussian Discriminative Analysis (GDA) (open)

### 자료의 형식

정형 데이터

고정된 필드로 정의된 정보  
- 데이터베이스, 스프레드시트 등

반정형 데이터

일정한 구조를 갖고 있는 정보로서 메타정보, 스키마 등을 포함하는 정보  
- XML, HTML 등

비정형 데이터

고정된 필드로 정의 되어 있지 않은 정보  
- 문서파일, 게시 글, 뉴스기사, SNS글, 이미지, 동영상, 음성정보 등

### 자료의 종류

양적 자료(숫자형)  
(Quantitative Data)

숫자로 표현  
1) 이산형 자료(Discrete Data) : 셀 수 있는 자료  
2) 연속형 자료(Continuous Data) : 셀 수 없는 자료

질적 자료(문자형)  
(Qualitative Data)

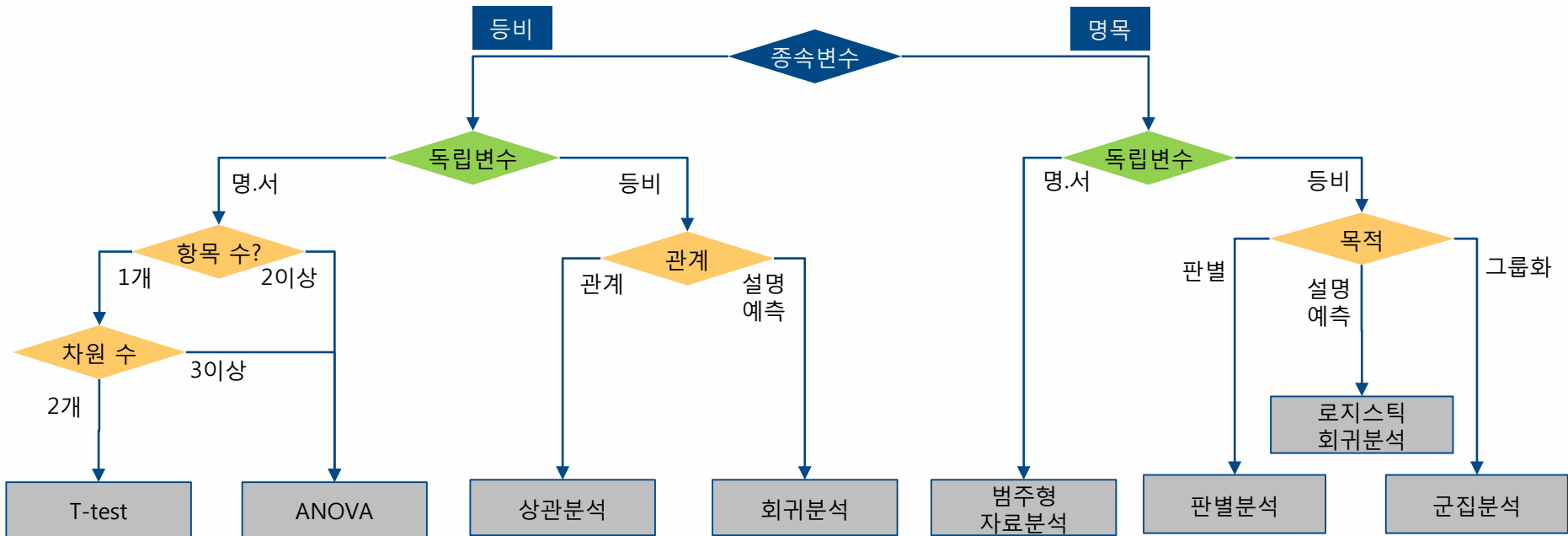
특성이 범주형으로만 구분되고 수치적으로는 측정이 되지 않는 자료



## I 자료의 구분

구분	질적(범주형) 자료		양적자료	
	명목척도	서열(순위)척도	등간척도	비율척도
정의	어떤 대상의 내용이나 특성을 구분하기 위한 기호	순서의 의미가 포함되어 있는 척도, 숫자의 크기로 상대적 비교가 가능	순위와 더불어, 측정치간의 차이에 대해서도 의미가 있는 척도	구간척도의 특성 외에 '절대원점'의 개념이 포함, 일반적으로 통계기법에 적용되는 척도
범주	O	O	O	O
순위	X	O	O	O
등간격	X	X	O	O
절대영점	X	X	X	O
비교 방법	확인, 분류	순위비교	간격비교	크기비교
산술적 계산	=	=><	=>< + -	=>< + - * /
평균의 측정	최빈값	중앙값	산술평균	기하평균 모든 통계
예	성별(남자 = '1', 여자 = '2') 계절, 상품유형, 결혼여부, 선수의등번호, 지역 등	선호도(좋다 = '1', 보통 = '2', 싫다 = '3') 석차, 학력 등	지능지수(IQ), 온도, 사회지표 등	TV시청률, 투표율, 무게, 연령, 생산원가, 신장, 출석률 등

## 변수(척도)별 통계분석 기법



### I 데이터 유형별 분석 기법

구분	연속형 종속변수	이산형 종속변수	종속변수가 없는 경우
연속형 독립변수 (Continuous Independent Variable)	예측(Forecasting) 분류(Classification)	예측(Forecasting) 분류(Classification)	군집화(Clustering)
이산형 독립변수 (Discrete Independent Variable)	예측(Forecasting) 분류(Classification)	예측(Forecasting) 분류(Classification)	군집화(Clustering)
범주형 독립변수 (Categorical Independent Variable)	분류(Classification)	분류(Classification)	연관성(Association) 연속성(Sequencing) 관계성(Link Analysis)

▪ 예측(Forecasting)	대용량 데이터집합내의 패턴을 기반으로 미래를 예측 (예: 수요예측)
▪ 분류(Classification)	일정한 집단에 대한 특정 정의를 통해 분류 및 구분을 추론 (예: 이탈한 고객)
▪ 군집화(Clustering)	구체적인 특성을 공유하는 자료들을 분류. 미리 정의된 특성에 대한 정보를 가지지 않는다는 점에서 분류와 다름 (예 : 유사 행동 집단의 구분)
▪ 연관성(Association)	동시에 발생한 사건간의 상호연관성을 탐색 (예: 장바구니의 상품들의 관계 규명)
▪ 연속성(Sequencing)	연관 규칙에 시간(time)의 개념을 첨가하여 시계열(time series)에 따른 패턴들의 상호연관성을 탐색 (예: 금융상품 사용에 대한 반복 방문)
▪ 관계성(Link Analysis)	대용량의 정보 값들 간의 관계를 규명 (예: SNS에서 관계분석)

## 알고리즘 성능 특성

표기법	형	설명
$O(1)$	상수형	자료의 량이 증가하더라도 같은 시간을 보장한다.
$O(\log n)$	로그형	$n$ 이 증가함에 따라서 $\log n$ 만큼 시간이 증가.
$O(n)$	선형	$n$ 증가 시, 시간도 비례해서 증가. 동일한 처리를 하는 경우.
$O(n \log n)$	선형로그형	$n$ 이 2배로 늘어나면 시간은 2배보다 약간 증가.
$O(n^2)$	평방형	이중루프
$O(n^3)$	입방형	삼중루프
$O(2^n)$	지수형	입력자료에 따라 시간이 급격히 증가
$O(n!)$	팩토리얼형	

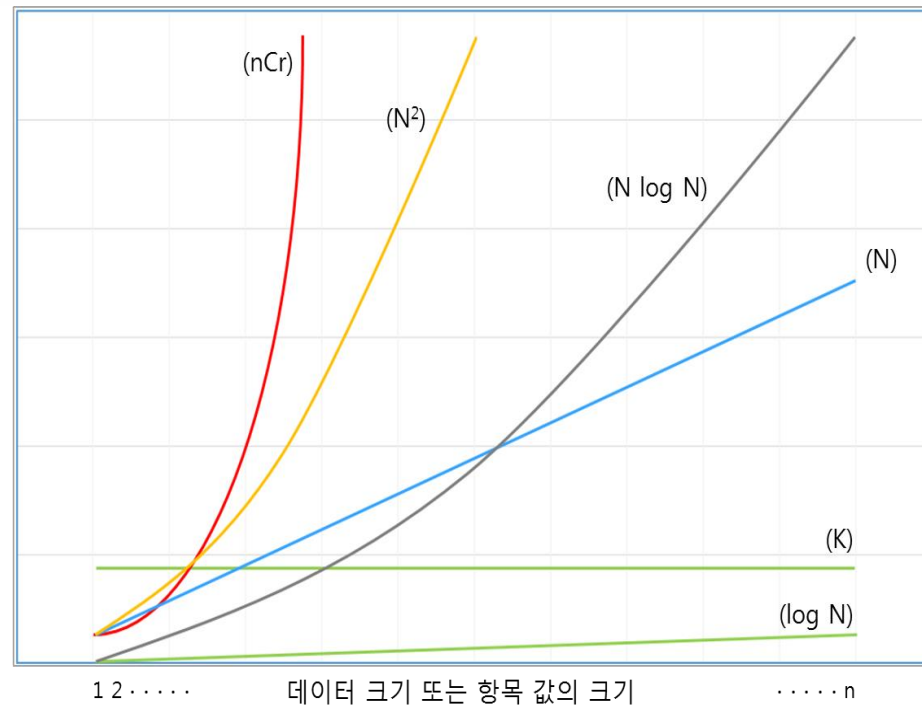
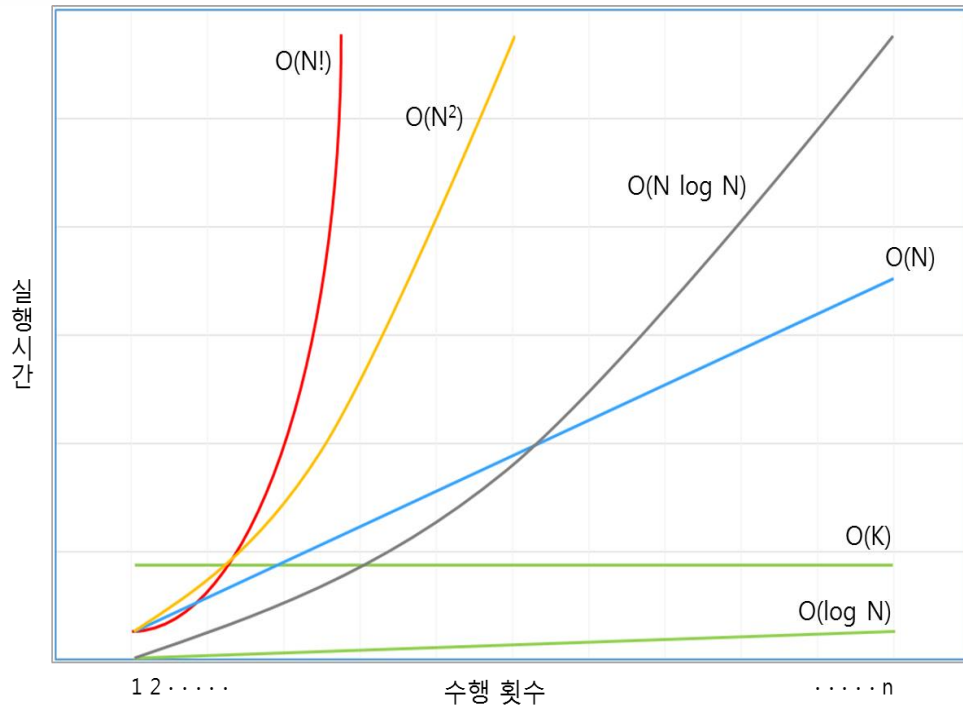
※ big-oh표기법에 의한 알고리즘의 수행 시간 비교

$$O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(n^3) < O(2^n) < O(n!)$$

수학적 정의

- 두 개의 함수  $f(n)$ 과  $g(n)$ 이 주어졌을때 모든  $n \geq n_1$ 에 대하여  $|f(n)| \leq c|g(n)|$ 을 만족하는 상수  $c$ 와  $n_1$ 이 존재하면  $f(n) = O(g(n))$
- 정의를 이용하여 위  $T(n)$ 을 증명하면 다음과 같다.(여기서  $n_1$ 과  $c$  는 여러 경우수가 나올 수 있다)  
 $n_1 = 2, c = 3$  일 때,  $n \geq 2$  에 대하여  $n^2 + n + 1 \leq 3n^2$  을 만족.

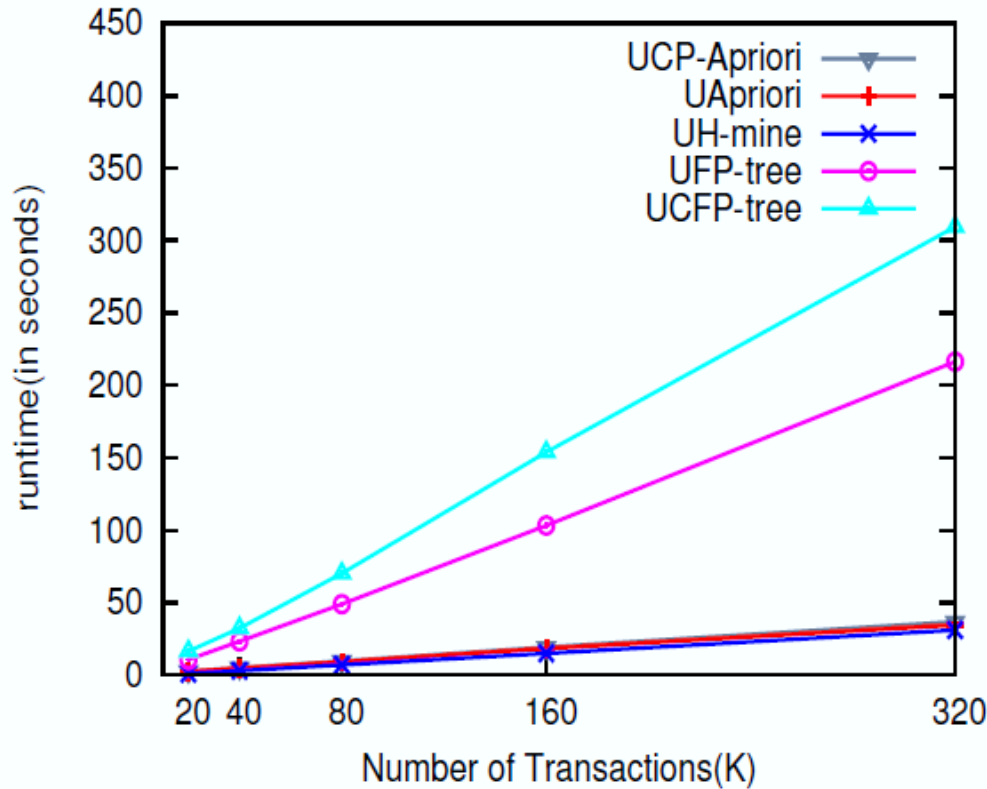
## 실행시간 및 메모리 증가 유형



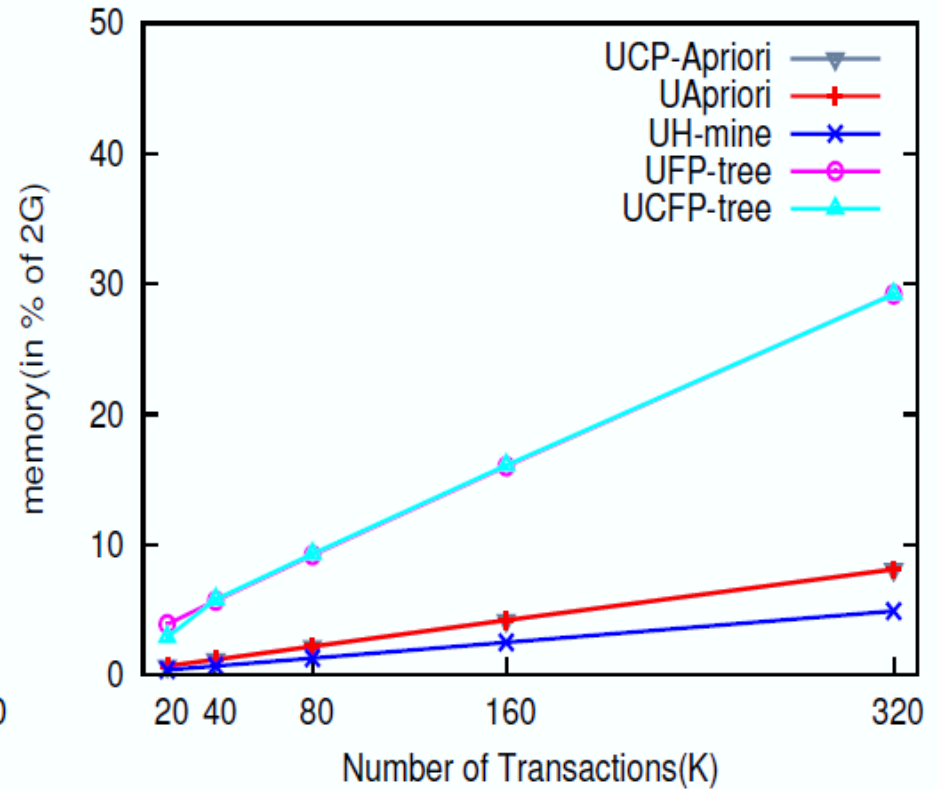
### ▪ Apriori Algorithm (예)

항목수	조합수	조합의 결과값
1,000	2	499,500
1,000	4	41,417,124,750
1,000	6	1,368,173,298,991,500
1,000	8	24,115,080,524,699,400,000
1,000	10	263,409,560,461,970,000,000,000

## 알고리즘 성능 비교 사례



a) Runtime

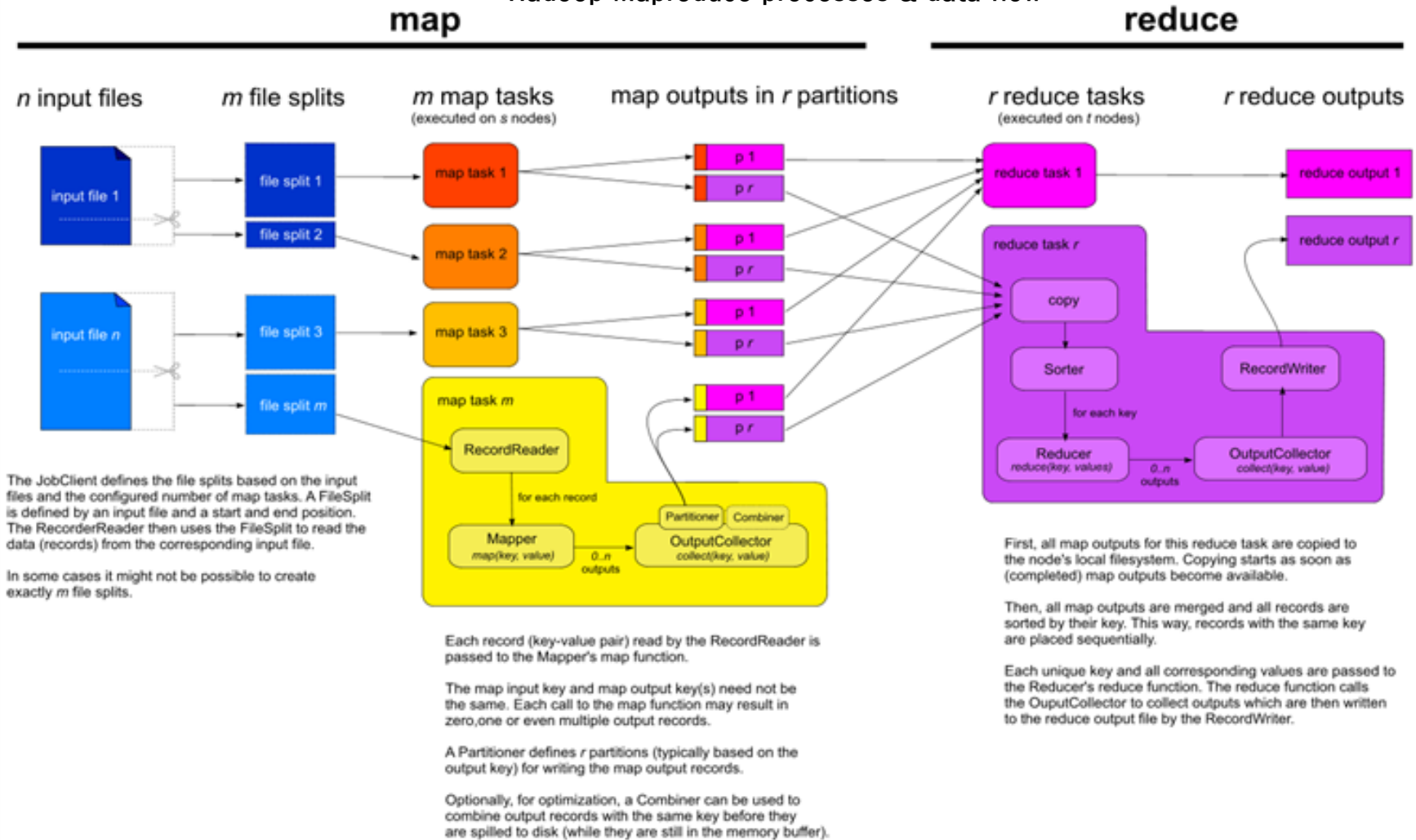


b) Memory

## Figure 6: Scalability Comparison

## Hadoop Mapreduce

Hadoop Mapreduce processes & data flow



## MapReduce Example – WordCount

The overall MapReduce word count process

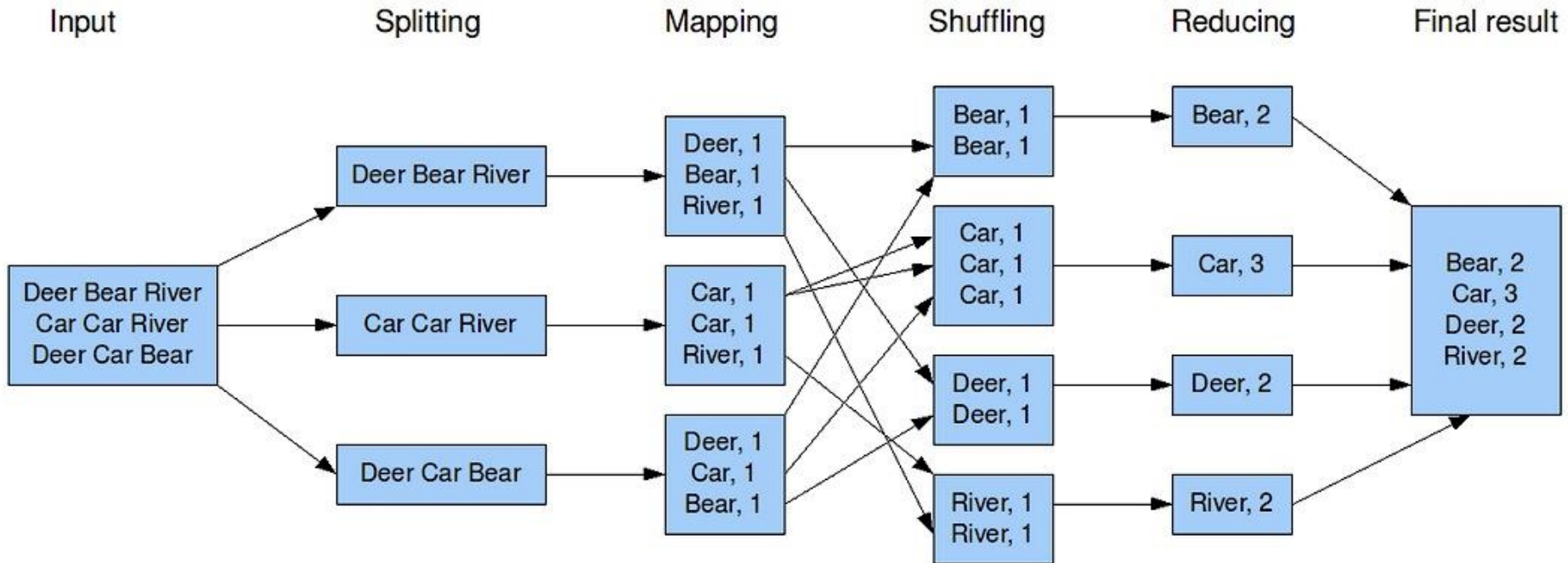


Image from: <http://blog.jteam.nl/wp-content/uploads/2009/08/MapReduceWordCountOverview1.png>



**사례 : Variance 구하기 (단일처리)**

$$\sigma^2 = \frac{\sum(x-\mu)^2}{n} = \frac{\sum(x^2-2\mu x+\mu^2)}{n} = \frac{\sum x^2}{n} - \frac{2\mu \sum x}{n} + \frac{\sum \mu^2}{n} = \frac{\sum x^2}{n} - 2\mu\mu + \frac{n\mu^2}{n} = \frac{\sum x^2}{n} - \mu^2$$

$$\sigma^2 = \frac{\sum x^2}{n} - \mu^2 = \frac{\sum x^2}{n} - \left(\frac{\sum x}{n}\right)^2$$

σ<sup>2</sup>을 결과 값으로 분산처리를 하면 취합 시 σ<sup>2</sup>의 값이 왜곡됨

σ<sup>2</sup>을 구하기 위한 인자 값만 분산처리로 수행

분산처리에서 구한 인자 값을 취합시 아래 식에 대입하여 연산

$$\frac{\sum x^2}{n} - \mu^2$$

	25		25	x*x=	625
	32		32	x*x=	1,024
	27		27	x*x=	729
	45		45	x*x=	2,025
	39		39	x*x=	1,521
	21		21	x*x=	441
	51		51	x*x=	2,601
	46		46	x*x=	2,116
sum=	286.00	n=	8	sumSq=	11,082
분산값=	<b>107.19</b>	u=	35.75		

A		C	분산
sumSq/n	(sum/n)=u	u*u	=A-C
1,385.25	35.75	1,278.06	<b>107.19</b>

## 사례 : Variance 구하기 (분산처리)

분할 자료 A	
	25
	32
	27
	45
sum=	129.00
분산값=	60.69

분할 자료 B	
	39
	21
	51
	46
sum=	157.00
분산값=	129.19

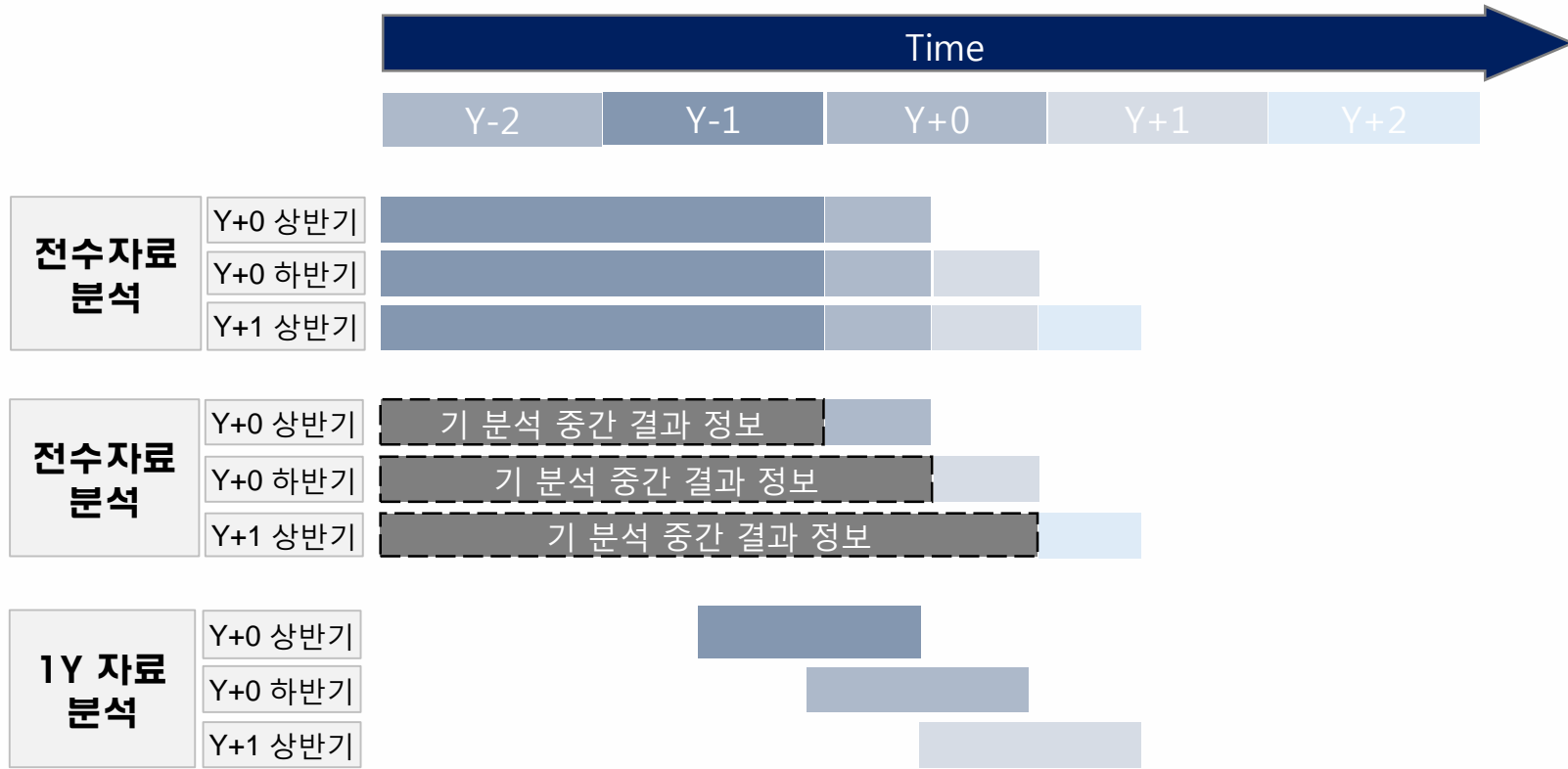
task A "분산값"=	60.69
task B "분산값"=	129.19
sum(task A + task B)=	189.88
<b>취합 값(sum / 2) =</b>	<b>94.94</b>

분할 자료 A					
	25		25	x*x=	625
	32		32	x*x=	1,024
	27		27	x*x=	729
	45		45	x*x=	2,025
sum=	129.00	n=	4	sumSq=	4,403

분할 자료 B					
	39		39	x*x=	1,521
	21		21	x*x=	441
	51		51	x*x=	2,601
	46		46	x*x=	2,116
sum=	157.00	n=	4	sumSq=	6,679

sum=	286.00	n=	8	sumSq=	11,082
A		C	분산		
sumSq/n	(sum/n)=u	u*u	=A-C		
1,385.25	35.75	1,278.06	<b>107.19</b>		

### 분석 시 데이터의 구간



분석 대상 정보량의 증가는 실행시간 및 Memory 용량과 밀접한 관계를 가짐.

## 알고리즘 발전 과정

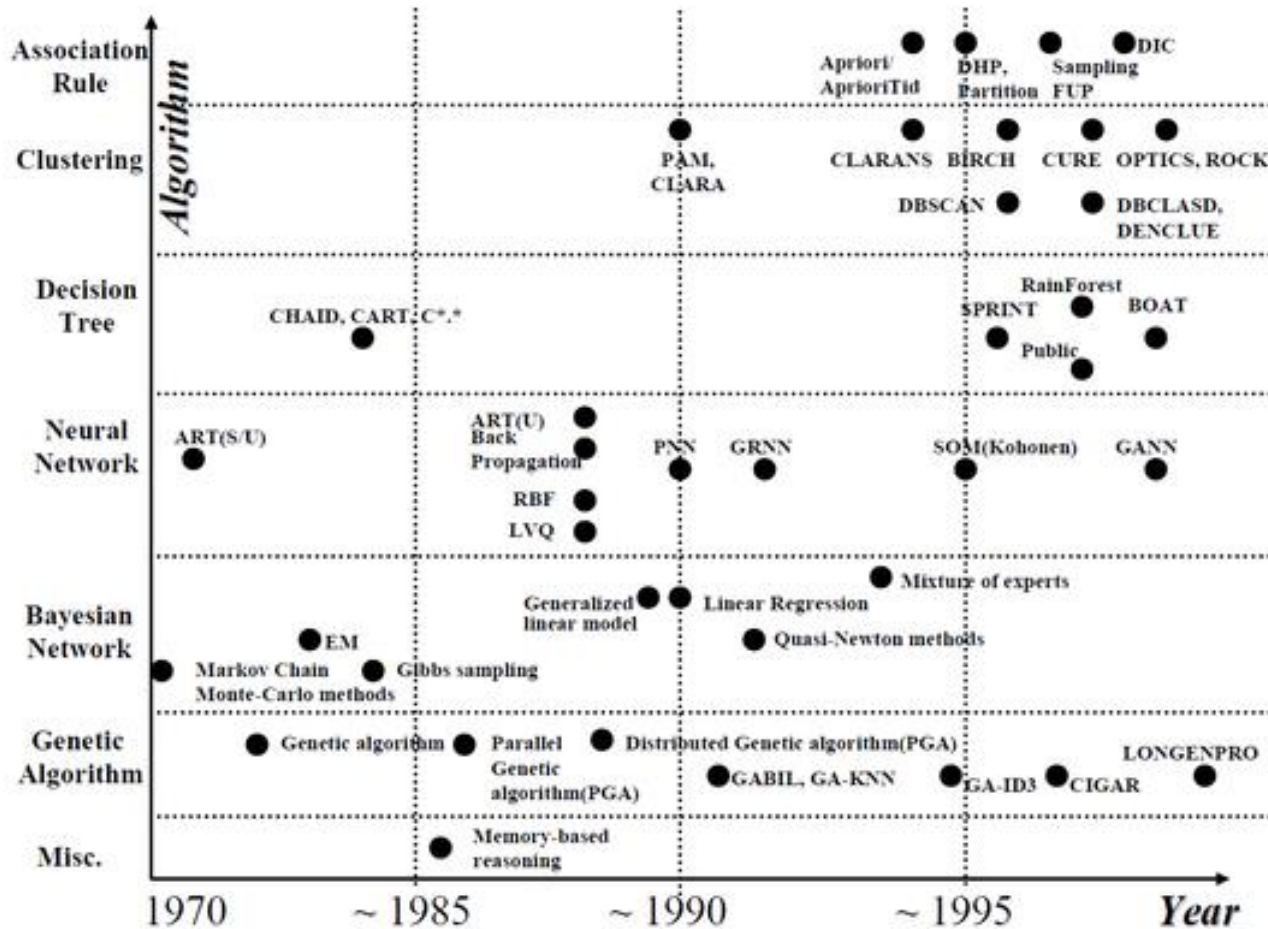


그림 2 선정된 데이터마이닝 알고리즘의 발전 과정과 계보

### Top 10 Data Mining Algorithms

- 1.C4.5
- 2.k-Means
- 3.SVM(Support Vector Machines)
- 4.Apriori
- 5.EM(Expectation Maximization)
- 6.PageRank
- 7.AdaBoost
- 8.kNN
- 9.Naive Bayes
- 10.CART

출처 : IEEE ICDM December 2006

## I 목적에 따른 DataMining 접근 방법

목적	분석 유형		설명	모델 종류
예측 Predictive Modeling	분류규칙 Classification		<ul style="list-style-type: none"> <li>과거의 데이터로부터 정보의 특성을 찾아내어 분류모형을 만들어 <b>새로운 정보의 결과값을 예측</b>하는 기법</li> <li>목표마케팅 및 고객 신용평가모형 등에 활용</li> </ul>	회귀분석, 의사결정나무 신경망분석, 유전자알고리즘
설명 Descriptive Modeling	데이터 군집 Clustering		<ul style="list-style-type: none"> <li><b>데이터의 유사 특성을 분석하여 몇 개의 그룹으로 분할하는 기법</b></li> <li>(분류와 유사하나 분석대상 데이터에 결과값이 없음)</li> <li>판촉활동이나 이벤트 대상선정에 활용</li> </ul>	Clustering
	유사성	연관규칙 Association	<ul style="list-style-type: none"> <li><b>데이터에 존재하는 항목간의 관계를</b> 찾아내는 기법</li> <li>제품이나 서비스의 교차판매(Cross Selling), 매장진열, 사기적발(Fraud Detection) 등에 활용</li> </ul>	패턴분석
		순차규칙 Sequence	<ul style="list-style-type: none"> <li><b>연관규칙에 시간개념이 적용된 기법</b></li> <li>목표마케팅(Target Marketing), 개인화 서비스 등에 활용</li> </ul>	순차패턴분석
	연결 분석 Link Analysis		<ul style="list-style-type: none"> <li><b>데이터의 값들 간의 관계를 파악하는 기법</b></li> <li>사회관계망분석, 감성분석 등에 활용</li> </ul>	Social Network Analysis Relational Content Analysis

### 분류 규칙(classification rules)

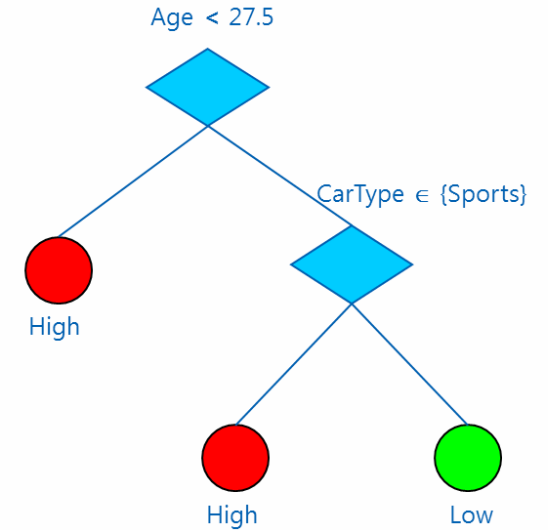
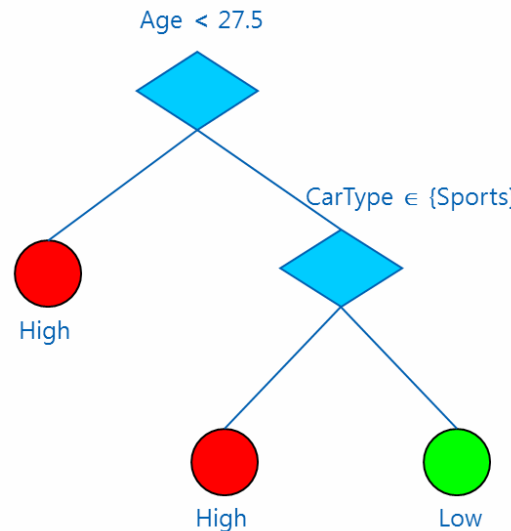
- ◆ Decision trees : 각 변수에 따라 수직적 분류
  - ID3 (Iterative Dichotomiser 3) : 명목형 예측변수 이지분리
  - C4.5 (successor of ID3) : 명목형 예측변수 다지분리
  - CART (Classification And Regression Tree) : 지니지수(Gini index: 범주형 목표변수에 적용) 또는 분산의 감소량(variance reduction: 연속형 목표변수에 적용) 등을 이용하여 분리
  - CHAID (CHI-squared Automatic Interaction Detector)
- ◆ Neural networks : 각 변수에 가중치를 사용, 분류율을 최대(오류율 최소)로 하는 것을 기반
  - multi-layer perceptron
- ◆ Genetic algorithms
- ◆ Linear classifiers
  - Logistic regression : 독립변수로 명목척도(성별, 인종 등) 사용
  - Naive Bayes classifier : 베이즈 정리(Bayes' theorem)를 기반의 단순한 확률 분류 (스팸메일)
  - Support vector machines : 분류율은 최대, 분류를 구분하는 기준(여백)을 최대화
- ◆ Kernel estimation
  - k-nearest neighbor(KNN) : 기계 학습의 방법 중에 가장 간단한 방법 중 하나
- ◆ Bayesian networks

## Decision-tree Classification

C 4.5 알고리즘의 엔트로피지수(Entropy index) 는 다항분포에서 우도 비 검정통계량을 사용하는 것으로, 이 지수가 가장 작은 예측변수와 그 때의 최적분리에 의해 마디를 생성

Numeric Categorical

Tid	Age	Car Type	Class
0	23	Family	High
1	17	Sports	High
2	43	Sports	High
3	68	Family	Low
4	32	Truck	Low
5	20	Family	High </td

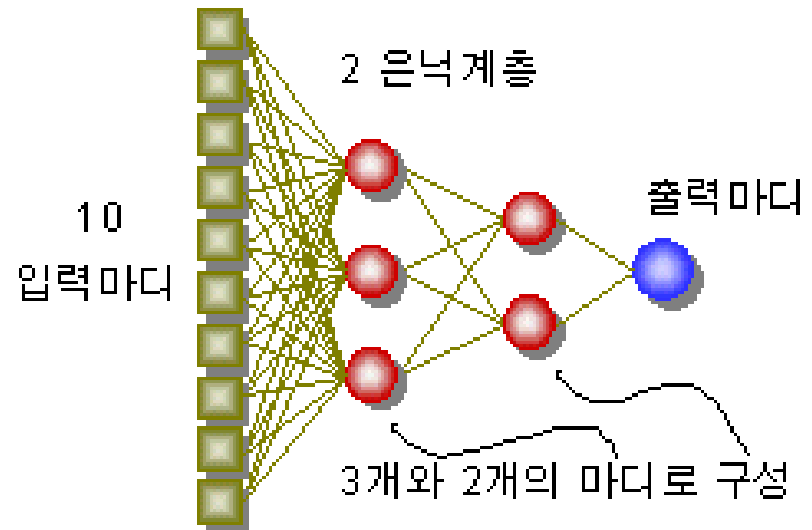
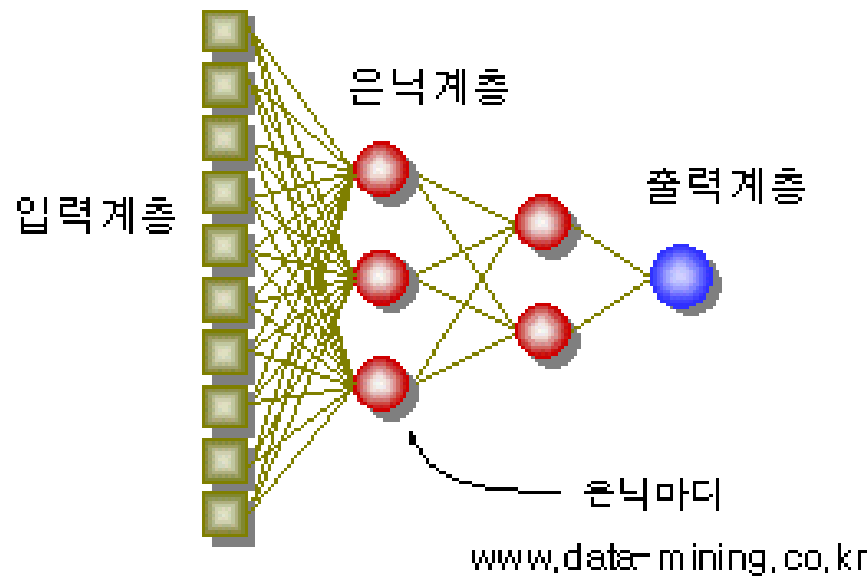


Age=40, CarType=Family ⇒ Class=Low

- 1) Age < 27.5 ⇒ High
- 2) Age ≥ 27.5 and CarType = Sports ⇒ High
- 3) Age ≥ 27.5 and CarType ≠ Sports ⇒ Low

## Neural Network

### 다계층인식인자( Multilayer Perceptron )





## Kernel Estimation

### CNN model reduction for KNN classifiers

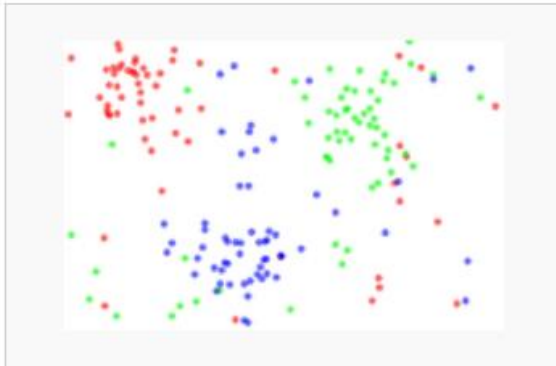


Fig. 1. The dataset.

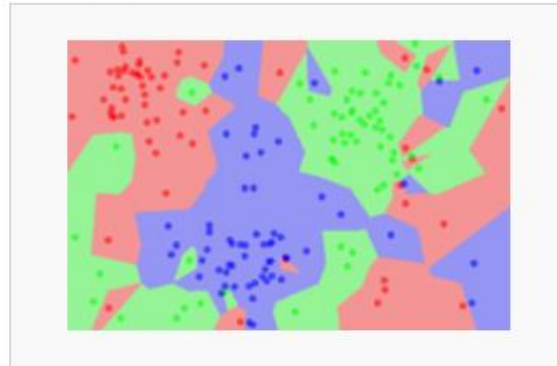


Fig. 2. The 1NN classification map.



Fig. 3. The 5NN classification map.

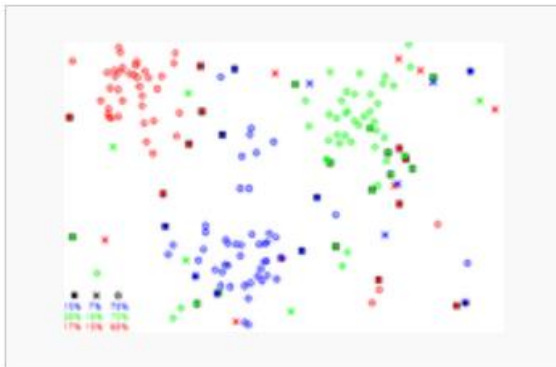


Fig. 4. The CNN reduced dataset.



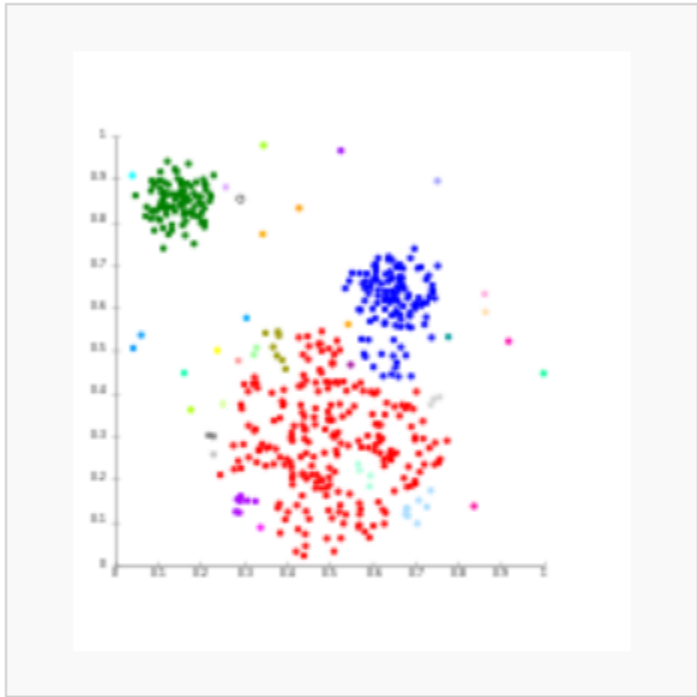
Fig. 5. The 1NN classification map based on the CNN extracted prototypes.

### 군집화 규칙(clustering rules)

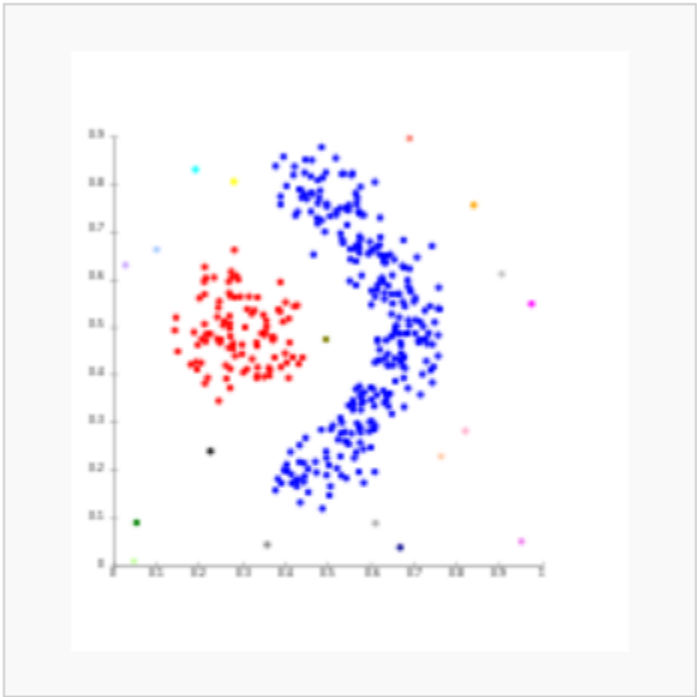
- ◆ Connectivity based methods
  - Hierarchical clustering : Linkage clustering
  - CURE(Clustering Using REpresentatives) : 비 구형 모형
  - Chameleon : 동적인 모델을 이용한 군집
    - ✓ CURE와 DBSCAN보다 좋은 성능으로 임의적인 형태의 군집, 다차원 시  $O(n^2)$  모델
- ◆ Centroid-based methods
  - k-means(평균값), k-medoids(중앙값), k-modes(최빈값)
- ◆ Distribution-based methods
  - EM(Expectation maximization)
- ◆ Density-based methods
  - OPTICS by using an R-tree index : 군집 구조 식별을 위한 순서화
  - DBSCAN(Density-Based Spatial Clustering of Applications with Noise) : 밀도 기반
  - DENCLUE(DENSity-based CLUstEring) : 밀도분포 함수 이용
- ◆ Grid-based methods
  - STING(STatistical INformation Grid) : 통계정보 격자 이용
  - WaveCluster : 웨이블릿 변환을 이용
  - CLIQUE(Clustering In QUEst) : 고차원 공간 군집화

## Connectivity based

### Linkage clustering examples



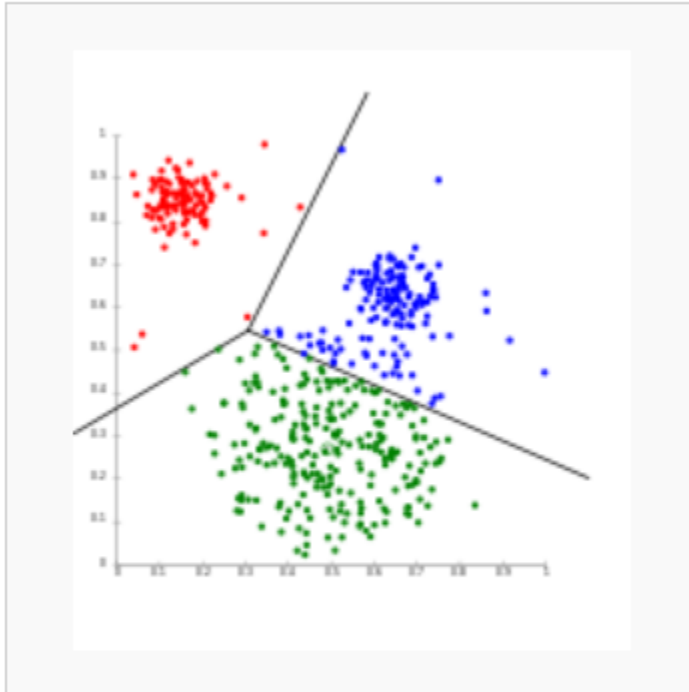
Single-linkage on Gaussian data. At 35 clusters, the biggest cluster starts fragmenting into smaller parts, while before it was still connected to the second largest due to the single-link effect.



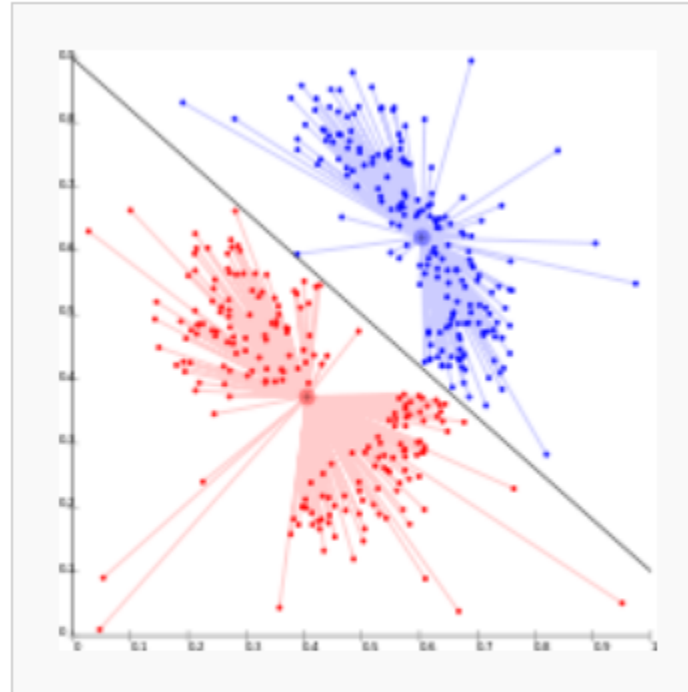
Single-linkage on density-based clusters. 20 clusters extracted, most of which contain single elements, since linkage clustering does not have a notion of "noise".

## Centroid based

### k-Means clustering examples



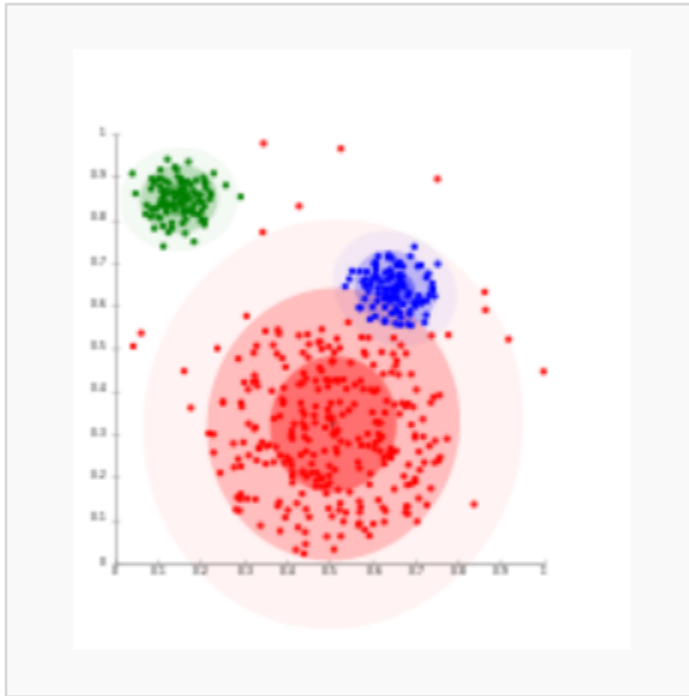
K-means separates data into Voronoi-cells, which assumes equal-sized clusters (not adequate here)



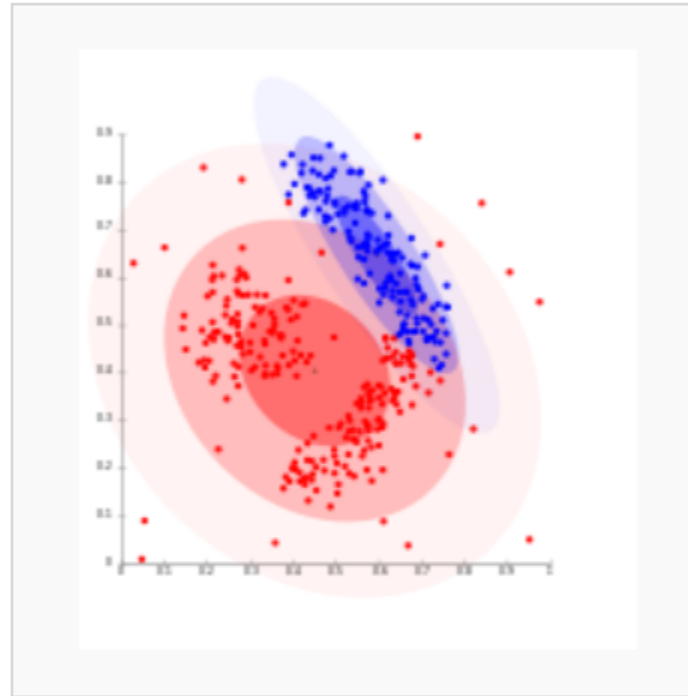
K-means cannot represent density-based clusters

## Distribution based

### EM clustering examples



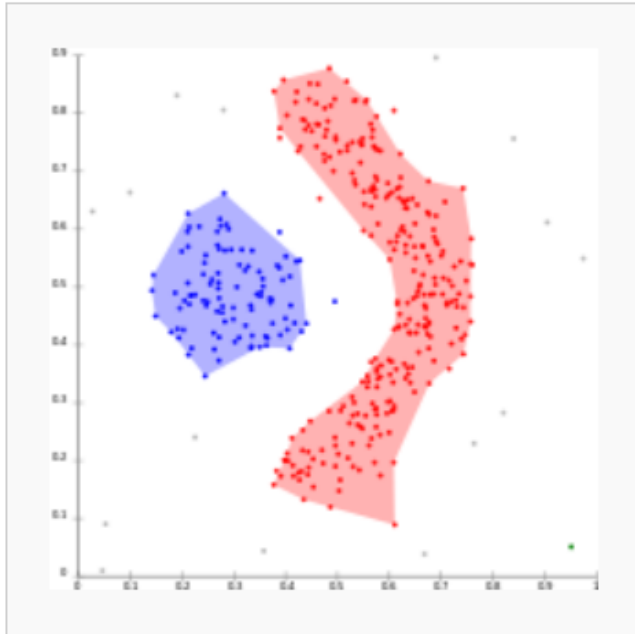
On Gaussian-distributed data, EM works well, since it uses Gaussians for modelling clusters



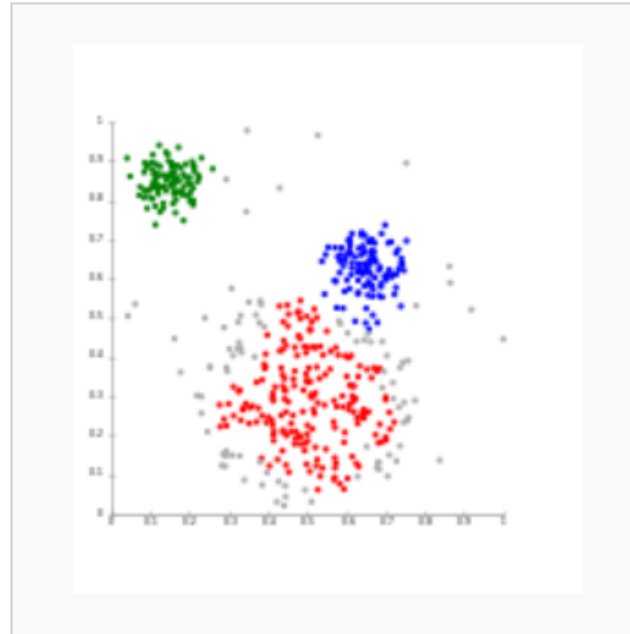
Density-based clusters cannot be modeled using Gaussian distributions

## Density based

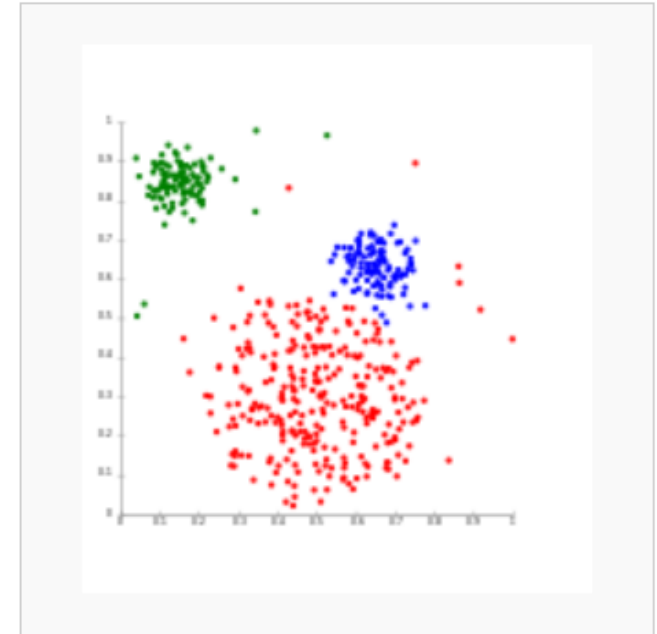
### density-based clustering examples



Density-based clustering with **DBSCAN**.



**DBSCAN** assumes clusters of similar density, and may have problems separating nearby clusters



**OPTICS** is a DBSCAN variant that handles different densities much better

### 연관 규칙(Association Rules)

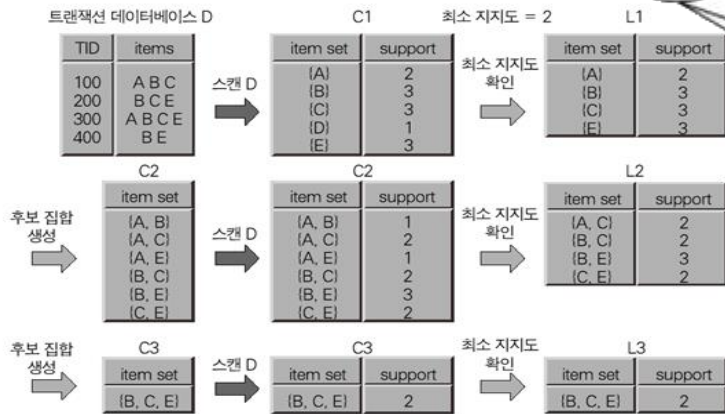
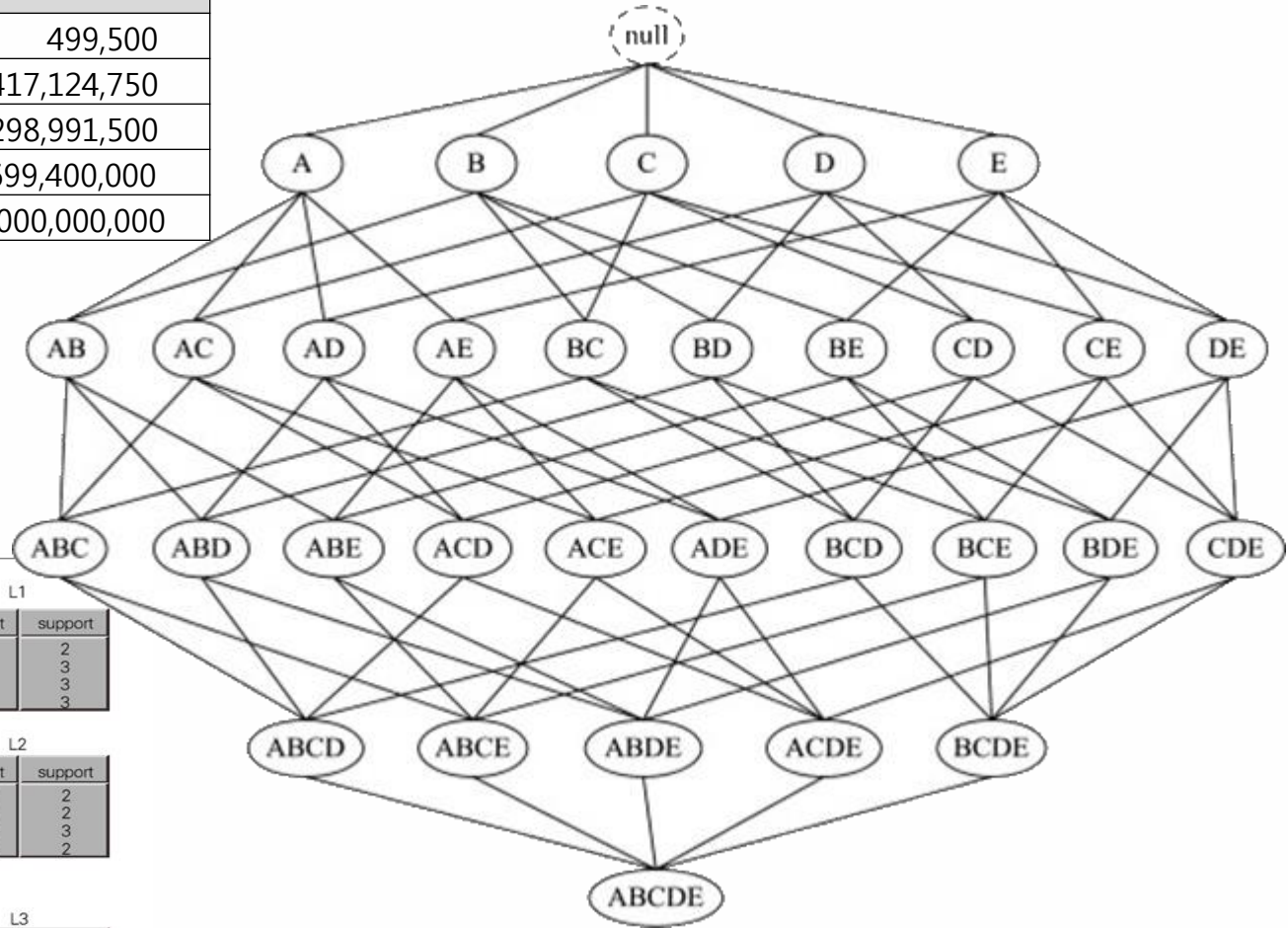
- ◆ Apriori Algorithm
  - Apriori Algorithm
  - AprioriTid Algorithm, AprioriHybrid Algorithm
  - Eclat Algorithm (depth-first search algorithm)
  - RElim Algorithm (Recursive Elimination Algorithm)
- ◆ Pattern-Growth Algorithm
  - FP-Growth Algorithm (Frequent Pattern Growth Algorithm)

### 순차 패턴(sequential patterns)

- ◆ Apriori Algorithm
  - AprioriAll, AprioriSome
  - DynamicSome
  - GSP(Generalized Sequential Patterns)
- ◆ Pattern-Growth Algorithm
  - FreeSpan(Frequent Pattern-Projected Sequential PAttern mining)
  - PrefixSpan(Prefix-projected Sequential PAtterrN mining)

## Apriori Algorithm

항목수	조합수	조합의 결과값
1,000	2	499,500
1,000	4	41,417,124,750
1,000	6	1,368,173,298,991,500
1,000	8	24,115,080,524,699,400,000
1,000	10	263,409,560,461,970,000,000,000





## PARALLEL FP-GROWTH : Mahout

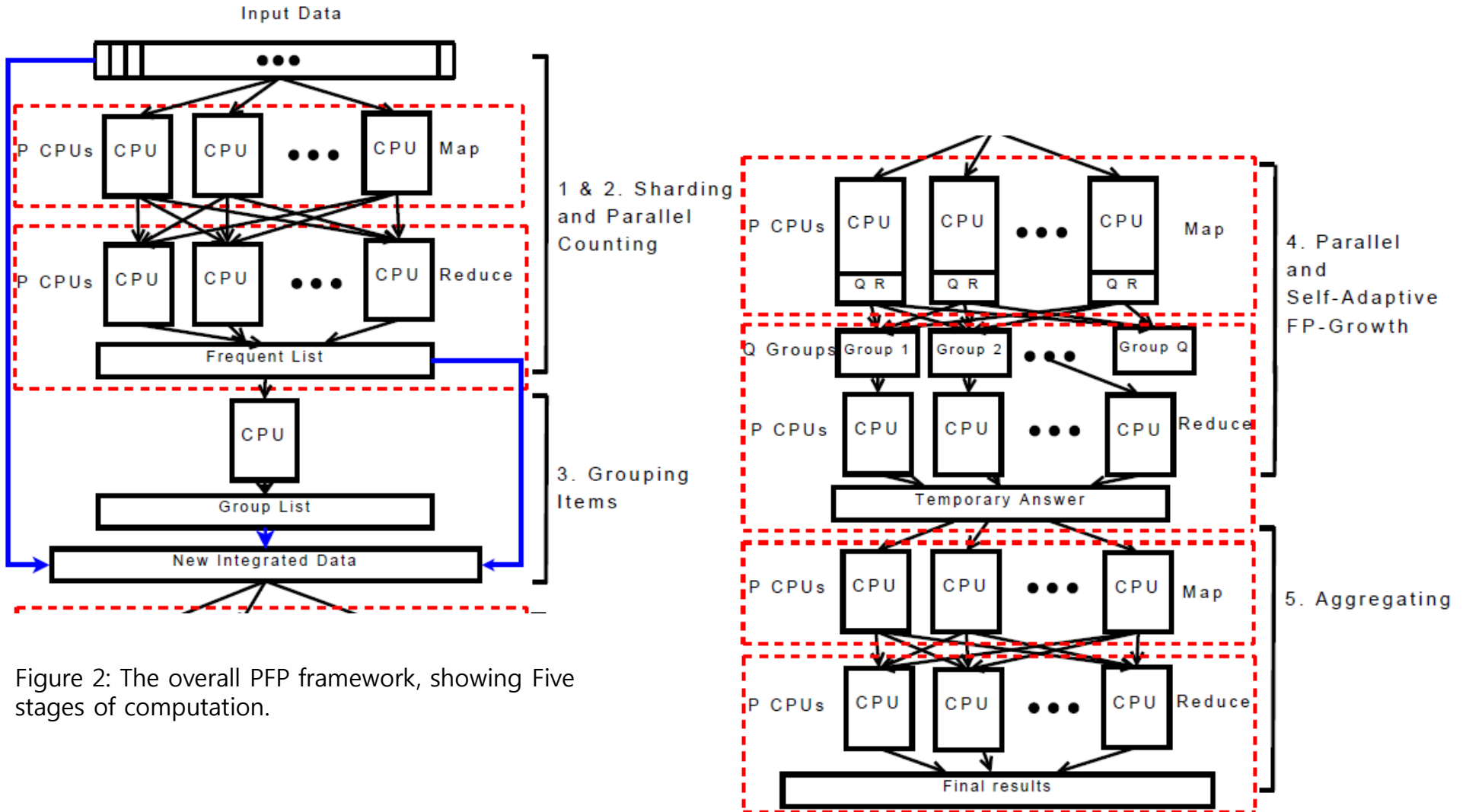


Figure 2: The overall PFP framework, showing Five stages of computation.

### PARALLEL FP-GROWTH : Mahout

```

Procedure: FPGrowth( $DB, \xi$ )
Define and clear F-List :  $F[]$ ;
foreach Transaction  $T_i$  in  $DB$  do
    foreach Item  $a_j$  in  $T_i$  do
         $F[a_i] ++$ ;
    end
end
Sort  $F[]$ ;
Define and clear the root of FP-tree :  $r$ ;
foreach Transaction  $T_i$  in  $DB$  do
    Make  $T_i$  ordered according to  $F$ ;
    Call  $ConstructTree(T_i, r)$ ;
end
foreach item  $a_i$  in  $I$  do
    Call  $Growth(r, a_i, \xi)$ ;
end
    
```

Algorithm 1: FP-Growth Algorithm

```

Procedure: Growth( $r, a, \xi$ )
if  $r$  contains a single path  $Z$  then
    foreach combination (denoted as  $\gamma$ ) of the nodes in
         $Z$  do
        Generate pattern  $\beta = \gamma \cup a$  with support =
            minimum support of nodes in  $\gamma$ ;
        if  $\beta.support > \xi$  then
            Call  $Output(\beta)$ ;
        end
    end
else
    foreach  $b_i$  in  $r$  do
        Generate pattern  $\beta = b_i \cup a$  with support =
             $b_i.support$ ;
        if  $\beta.support > \xi$  then
            Call  $Output(\beta)$ ;
        end
        Construct  $\beta$ 's conditional database ;
        Construct  $\beta$ 's conditional FP-tree  $Tree_\beta$ ;
        if  $Tree_\beta \neq \phi$  then
            Call  $Growth(Tree_\beta, \beta, \xi)$ ;
        end
    end
end
end
    
```

Algorithm 2: The FP-Growth Algorithm.

## PARALLEL FP-GROWTH : Mahout

```

Procedure: Mapper(key, value= $T_i$ )
  foreach item  $a_i$  in  $T_i$  do
    Call Output(( $a_i$ , '1'));
  end
Procedure: Reducer(key= $a_i$ , value= $S(a_i)$ )
   $C \leftarrow 0$ ;
  foreach item '1' in  $T_i$  do
     $C \leftarrow C + 1$ ;
  end
  Call Output((null,  $a_i + C$ ));
  
```

Algorithm 3: The Parallel Counting Algorithm

```

Procedure: Mapper(key, value= $v + \text{supp}(v)$ )
  foreach item  $a_i$  in  $v$  do
    Call Output(( $a_i$ ,  $v + \text{supp}(v)$ ));
  end
Procedure: Reducer(key= $a_i$ , value= $S(v + \text{supp}(v))$ )
  Define and clear a size  $K$  max heap :  $HP$ ;
  foreach pattern  $v$  in  $v + \text{supp}(v)$  do
    if  $|HP| < K$  then
      insert  $v + \text{supp}(v)$  into  $HP$ ;
    else
      if  $\text{supp}(HP[0].v) < \text{supp}(v)$  then
        delete top element in  $HP$ ;
        insert  $v + \text{supp}(v)$  into  $HP$ ;
      end
    end
  end
  Call Output((null,  $a_i + C$ ));
  
```

Algorithm 5: The Aggregating Algorithm

```

Procedure: Mapper(key, value= $T_i$ )
  Load G-List;
  Generate Hash Table  $H$  from G-List;
   $a[] \leftarrow \text{Split}(T_i)$ ;
  for  $j = |T_i| - 1$  to 0 do
     $HashNum \leftarrow \text{getHashNum}(H, a[j])$ ;
    if  $HashNum \neq \text{Null}$  then
      Delete all pairs which hash value is  $HashNum$ 
      in  $H$ ;
      Call
        Output(( $HashNum$ ,  $a[0] + a[1] + \dots + a[j]$ ));
    end
  end
Procedure: Reducer(key= $gid$ , value= $DB_{gid}$ )
  Load G-List;
   $nowGroup \leftarrow G\text{-List}_{gid}$ ;
   $LocalFPtree \leftarrow \text{clear}$ ;
  foreach  $T_i$  in  $DB(gid)$  do
    Call insert - build - fp - tree( $LocalFPtree, T_i$ );
  end
  foreach  $a_i$  in  $nowGroup$  do
    Define and clear a size  $K$  max heap :  $HP$ ;
    Call TopKFPGrowth( $LocalFPtree, a_i, HP$ );
    foreach  $v_i$  in  $HP$  do
      Call Output((null,  $v_i + \text{supp}(v_i)$ ));
    end
  end
  
```

Algorithm 4: The Parallel FP-Growth Algorithm

### 비정형 데이터 분석

- 콘텐츠 분석(Content Analysis)
  - 디지털 환경에서 생성되는 정형 및 비정형을 포함하여 여러 수준의 콘텐츠를 비즈니스 인텔리전스와 비즈니스 전략의 가치를 높이기 위한 하나의 방법
  - 보다 향상된 의사결정을 위한 Trend 및 Pattern을 발견하는 것
- 텍스트 분석(Text Analytics)
  - 비정형 데이터로부터 의미 있는 정보를 추출하기 위하여 언어적 혹은 통계적 기술 자연어 처리 등 방법을 통해 분석에 활용될 수 있는 형태의 데이터로 변환
- 실시간 분석(Real-time Analytics)
  - 분석에 필요한 모든 데이터를 활용하여 사용자가 분석을 수행하고 하는 시점에 빠르고 적시에 지식을 제공해 줄 수 있는 분석 기법
  - 결과의 정확도 및 신뢰도보다는 사용자에게 분석결과를 적시에 제공하는 것에 주안점을 가짐

○ 웹 마이닝(Web Mining)	인터넷상에서 수집된 정보를 데이터 마이닝 방법으로 분석하는 기법
○ 소셜 마이닝(Social Mining)	소셜 미디어의 글과 사용자간 관계를 수집하여 소비자의 성향과 패턴 등을 분석함으로써 판매 및 홍보에 이용, 여론변화나 사회적 흐름을 파악
○ 현실 마이닝(Reality Mining)	사람들의 행동패턴을 예측하기 위해서 사회적 행동과 관련된 정보를 수집하여 분석하는 기법으로 현실생활에서 발생하는 정보를 기반으로 인간관계나 행동 추론

## Content Analysis의 활용

Purpose	Element	Question	Use
Make inferences about the antecedents of communications	Source	Who	Answer question of disputed authorship
	Encoding process	Why	<ul style="list-style-type: none"> <li>• Secure political &amp; military intelligence</li> <li>• Analyse traits of individuals</li> <li>• Infer cultural aspects &amp; change</li> <li>• Provide legal &amp; evaluative evidence</li> </ul>
Describe & make inferences about the characteristics of communications	Channel	How	<ul style="list-style-type: none"> <li>• Analyse techniques of persuasion</li> <li>• Analyse style</li> </ul>
	Message	What	<ul style="list-style-type: none"> <li>• Describe trends in communication content</li> <li>• Relate known characteristics of sources to messages they produce</li> <li>• Compare communication content to standards</li> </ul>
	Recipient	To whom	<ul style="list-style-type: none"> <li>• Relate known characteristics of audiences to messages produced for them</li> <li>• Describe patterns of communication</li> </ul>
Make inferences about the consequences of communications	Decoding process	With what effect	<ul style="list-style-type: none"> <li>• Measure readability</li> <li>• Analyse the flow of information</li> <li>• Assess responses to communications</li> </ul>

출처 : Ole Hoisti, Duke University

### 관계 분석(Link Analysis)

#### ◆ Social Network Analysis 및 Relational Content Analysis

사회 구조를 노드(node)와 이들 노드를 연결하는 링크로 구성되는 연결망(network) 으로 도식, 이들 간의 상호작용을 계량화해 주는 분석 기법

- 자연어 처리(Natural Language Processing)

- ☞ 형태소분석

- ☞ 구문분석

- Algorithm

- ☞ 그래프 이론(Graph Theory)

- ☞ 행렬/vector/matrix ANOVA 등

- 가시화(Visualization)

- 적용 영역

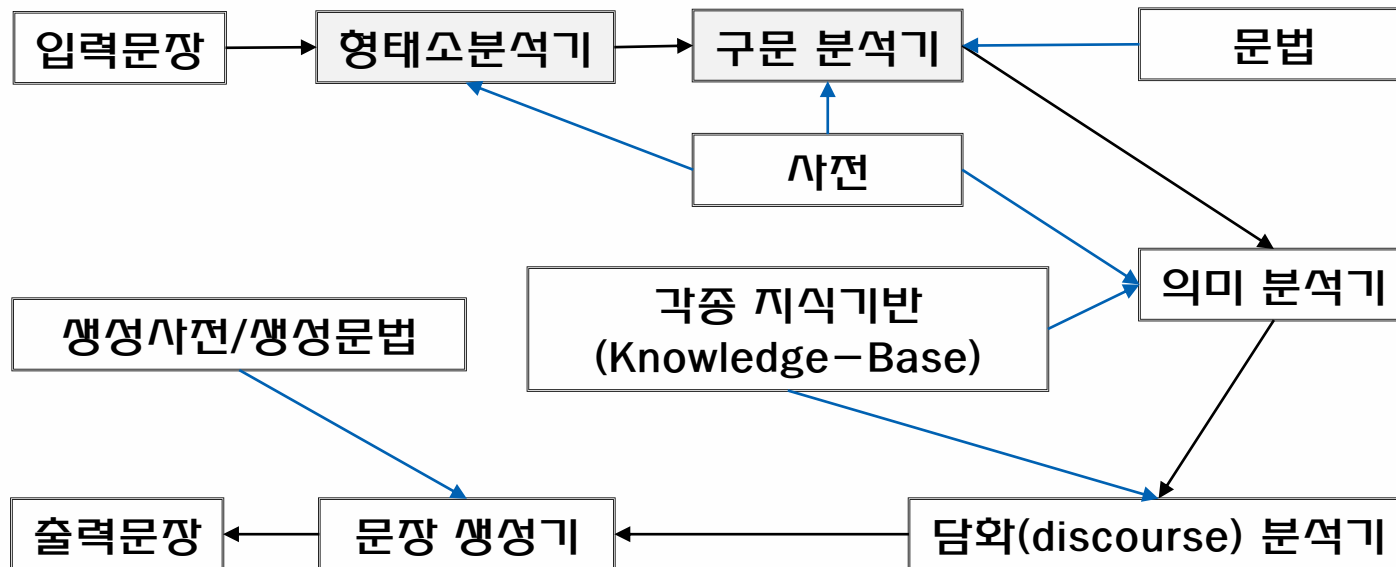
- ✓ 생물학 : 전염 경로의 분쇄 (감염자, 건강인=노드, 감염성 접촉=링크)
  - ✓ 비즈니스 : 아마존 도서 구입 안내 서비스(구매자/책=노드, 거래행위=링크)
  - ✓ 도시계획 : 도시계획 시 도로의 설계 (도시=노드, 길=링크)
  - ✓ 정치 : 테러조직 붕괴 (테러리스트=노드, 접선=링크)
  - ✓ 컴퓨터 : 인터넷 (컴퓨터, 사람=노드, 통신선=링크)
  - ✓ 사회학 : 한국 재벌가 혼맥 (재벌가 가족들=노드, 결혼=링크)
  - ✓ 경영학 : 지식경영 활성화 (개인지식=노드, 지식공유=노드)

## Natural Language Processing

### ◆ 자연어 처리란

- 인간의 언어를 기계가 이해하고 생성할 수 있도록 하기 위한 연구

## 자연언어처리 시스템의 구성도



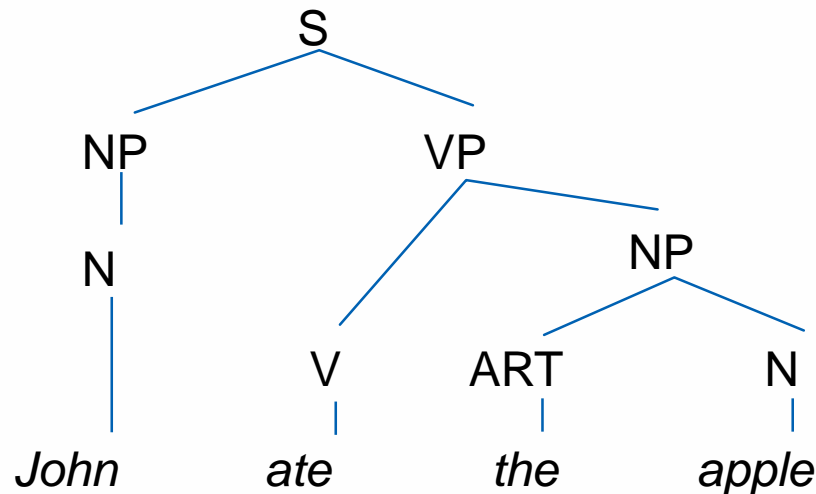
### I 어절, 단어, 형태소

- ◆ 어절 : 띄어쓰기의 단위
  - 두 단어로 된 어절 : 체언(혹은 용언 및 부사) + 조사
  - 한 단어로 된 어절 : 체언, 용언, 수식언, 감탄사
- ◆ 형태소 : 뜻을 가진 가장 작은 말의 단위
  - 자립성의 유무에 따라 :
    - ☞ 자립 형태소 : 체언, 수식언, 감탄사
    - ☞ 의존 형태소 : 조사, 어간, 어미, 접사
  - 의미, 기능에 따라 :
    - ☞ 실질 형태소 : 체언, 용언의 어근, 수식언, 감탄사
    - ☞ 형식 형태소 : 조사, 어미, 접사
- ◆ 단어 : 자립할 수 있는 말이나 자립형태소와 쉽게 분리되는 말
  - 홀로 자립하는 말 : 체언, 수식언, 감탄사
  - 자립 형태소와 쉽게 분리되는 말 : 조사
  - 의존 형태소끼리 어울려서 자립하는 말 : 용언



### Grammars and Parsing

- ◆ 문법 (Grammar) :
  - 문장의 구조적 성질을 규칙으로 표현한 것
- ◆ 구문 분석기 (Parser) :
  - 문장의 구조를 문법을 이용하여 찾아내는 process
  - 문장의 구문 구조는 Tree를 이룬다. 즉, 몇개의 형태소들이 모여서 구문 요소(phrase)를 이루고, 그 구문 요소들간의 결합구조를 Tree형태로써 구문 구조를 이루게 된다.



## I 형태소 분석

“대기업의 불공정 거래로 벤처나 중소기업이 성장하지 못하고 국가경제에 악순환을 불러오고 있다.”(관훈토론 2011년 3월22일)

대기업의	대기업/NNG+의/JKG
불공정	불/XPN+공정/NNG
거래로	거래/NNG+로/JKB
벤처나	벤처/NNG+나/JC
중소기업이	중소기업/NNG+이/JKS
성장하지	성장/NNG+하/XSV+지/EC
못하고	못하/VX+고/EC
국가경제에	국가/NNG+경제/NNG+에/JKB
악순환을	악순환/NNG+을/JKO
불러오고	불러오/VV+고/EC
있다	있/VX+다/EC

NNG	일반명사	EP	선어말어미
NNP	고유명사	EF	종결어미
NNB	의존명사	EC	연결어미
NP	대명사	ETN	명사형전성어미
NR	수사	ETM	관형형전성어미
VV	동사	XPN	체언접두사
VA	형용사	XSN	명사파생접미사
VX	보조용언	XSV	동사파생접미사
VCP	긍정지정사	XSA	형용사파생접미사
VCN	부정지정사	XR	어근
MM	관형사	SF	마침표, 물음표, 느낌표
MAG	일반부사	SP	쉼표, 가운뎃점, 콜론, 빗금
MAJ	접속부사	SS	
IC	감탄사	SE	줄임표
JKS	주격조사	SO	붙임표(물결, 숨김, 빠짐)
JKC	보격조사	SL	외국어
JKG	관형격조사	SH	한자
JKO	목적격조사	SW	기타 기호(논리 수학기호, 화폐 기호) 등)
JKB	부사격조사	NF	명사추정범주
JKV	호격조사	NV	용언추정범주
JKQ	인용격조사	SN	숫자
JX	보조사	NA	분석불능범주
JC	접속조사		

\*) 세종 계획 품사

## 구문 분석

“대기업의 불공정 거래로 벤처나 중소기업이 성장하지 못하고 국가경제에 악순환을 불러오고 있다.”(관훈토론 2011년 3월22일)

P . q:		
U 있다		W:있 E:다
_ 불러오고		V:불러오 E:고
O 악순환을		N:악순환 J:을
B 국가경제에		N:국가경제 J:에
_ 못하고		J:못하 E:고
_ 성장하지		V:성장하 E:지
S 중소기업이		N:중소기업 J:이
벤처나		N:벤처 J:이나
B 거래로		N:거래 J:으로
N 불공정		N:불공정
G 대기업의		N:대기업 J:의



## Graph Theory

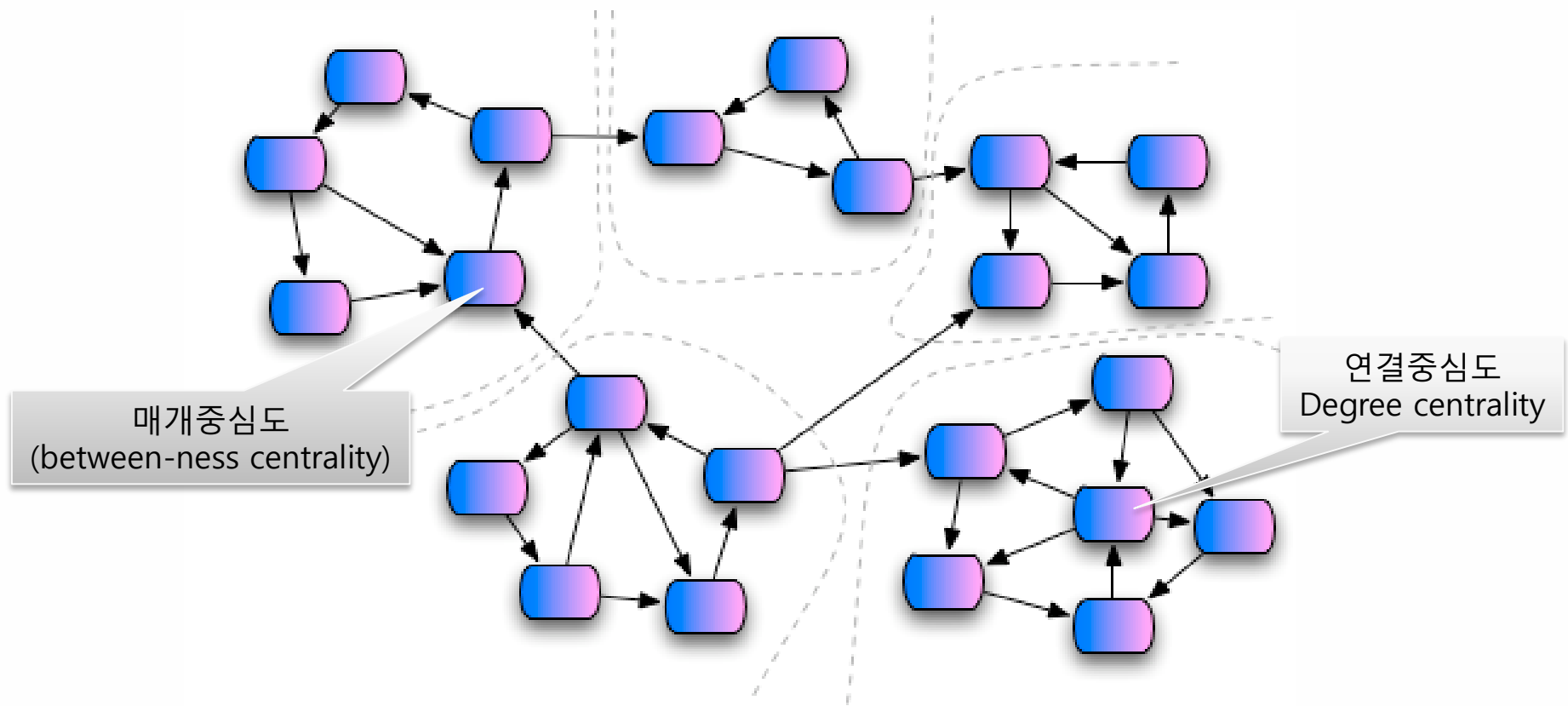
- ◆ Node와 Link
  - Node : 사람, 생물, 사물, 개념
  - Link : 작용(일방, 쌍방), 관계(우호적, 비우호적), 의사소통, 판매와 구매 등
- ◆ 행렬대수(벡터(vector)와 행렬(matrix))
  - scalar : 하나의 값으로 이루어진 데이터. Ex) Likert 3, 여성, 토익 849점
  - vector : 하나의 배열로 이루어진 데이터. Ex)  $x = \{x_1, x_2, x_3, \dots\}$
  - matrix : 여러 값들을 직사각형 모양의 2차원으로 배열한 데이터.
    - ✓ Ex)  $m(\text{행=row}) \times n(\text{열=column})$ .  $m=n$  (정방행렬=diagonal 존재)
  - 행렬 연산 기법: 행렬 간 가감승제, 행렬 간 상관관계 연산 등
- ◆ Mode of Matrix
  - 1-mode network: 동일 벡터 간의 상호작용
    - ✓ 1~n까지 학습자 간 이메일 교환 네트워크 / 콘텐츠 간 연결 네트워크
  - 2-mode network: 이질 벡터 간의 상호작용
    - ✓ 학습자와 콘텐츠 등 이질 벡터 간 상호작용

## 주요 측정값(measures)

노드(Node)
<ul style="list-style-type: none"><li>▪ 연결중심도(degree centrality)<ul style="list-style-type: none"><li>✓ 각 node가 갖고 있는 link의 개수</li></ul></li><li>▪ 거리중심도(closeness centrality)<ul style="list-style-type: none"><li>✓ 한 node와 다른 모든 node간의 평균적인 최단 경로거리</li></ul></li><li>▪ 매개중심도(betweenness centrality)<ul style="list-style-type: none"><li>✓ 한 node가 다른 모든 node들 상호간의 경로 사이에서 타인 또는 하위 그룹들간의 의사소통을 어느 정도 원활하게 연결시켜 주는가를 정도</li></ul></li><li>▪ 기타 power, effects, eigenvector, status 등</li></ul>

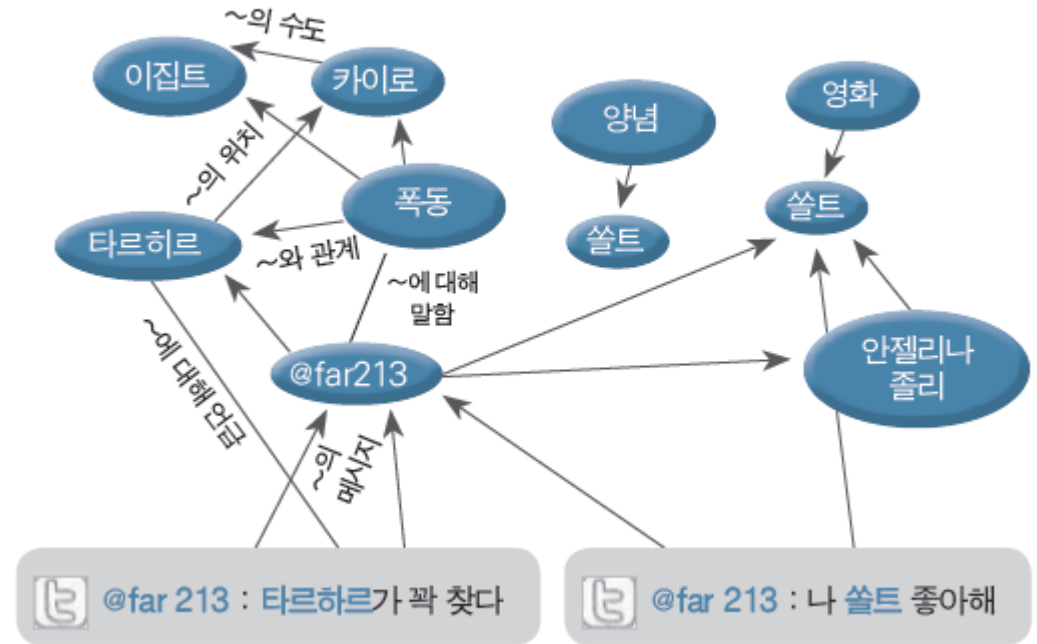
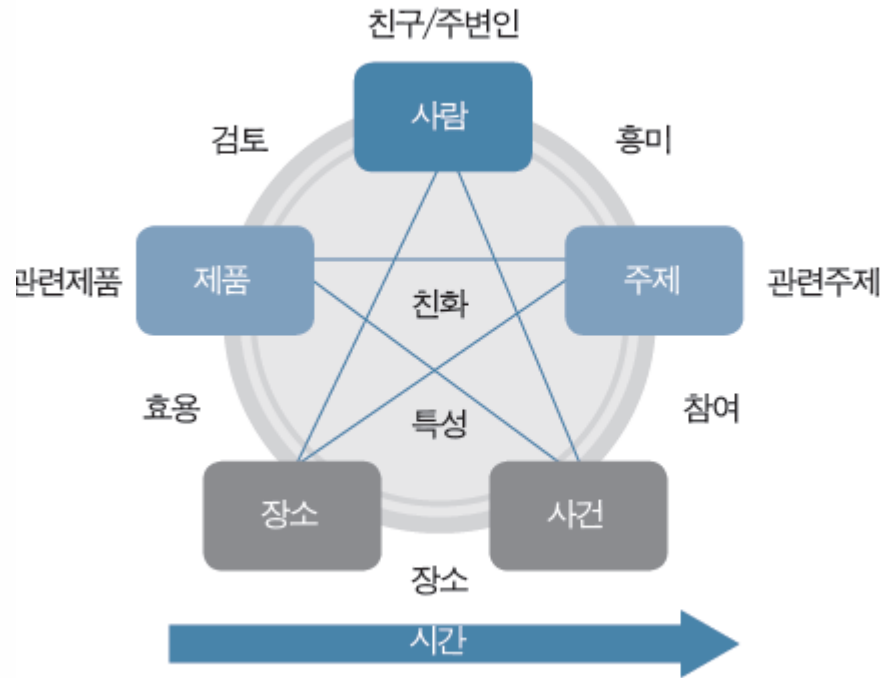
링크(Link)
<ul style="list-style-type: none"><li>▪ 응집도 (density/centralization)<ul style="list-style-type: none"><li>✓ 존재할 수 있는 가능한 총 link의 숫자 대비 실현된 link 숫자의 비율</li></ul></li><li>▪ 결속도 (cohesion)<ul style="list-style-type: none"><li>✓ 네트워크 내 모든 node들이 접근하기 위해 필요한 링크의 총 합계, 즉 경로 거리(path distance)</li></ul></li><li>▪ 측지선최단평균거리 (geodesic distance)<ul style="list-style-type: none"><li>✓ 모든 node간의 최단 경로거리 (cf. 거리중심도)</li></ul></li><li>▪ 하위집단 (component, clique 등)<ul style="list-style-type: none"><li>✓ 네트워크 내 존재하는 하위집단 규명</li></ul></li><li>▪ 구조적 유사성 (core-periphery, block model 등)<ul style="list-style-type: none"><li>✓ 두 개의 네트워크가 서로 구조적으로 유사한 정도</li></ul></li></ul>

## Graph Theory (예)



## Relational Content Analysis

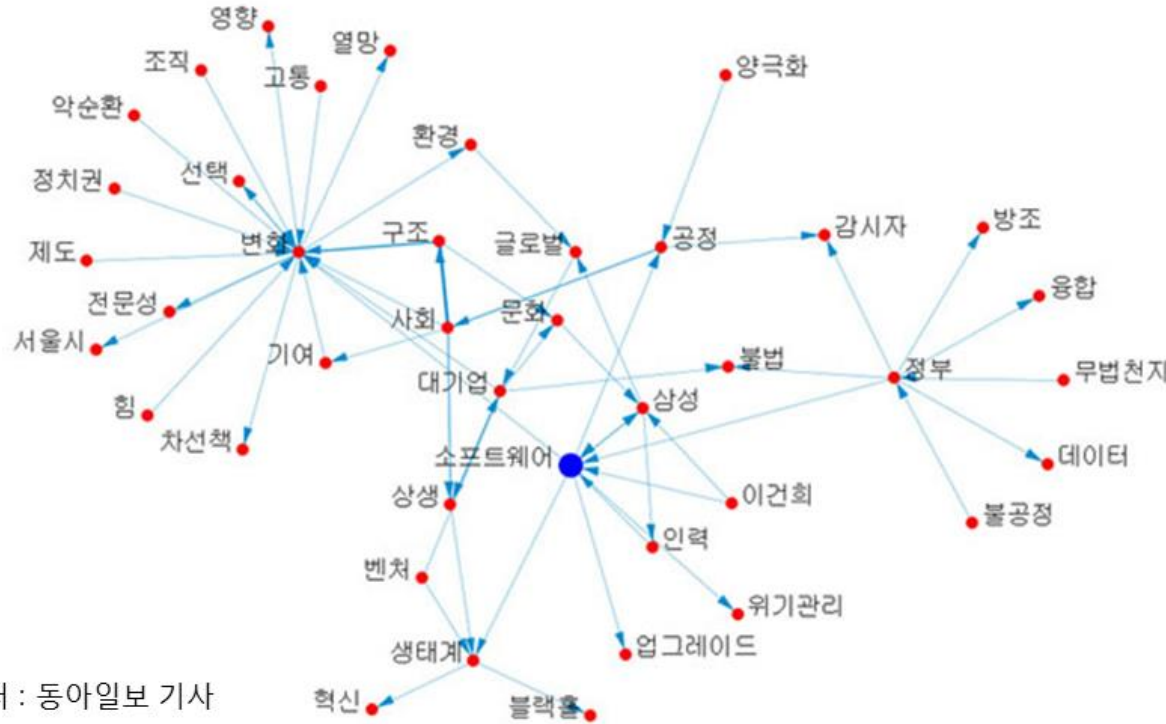
- 그림 4 월마트 랩의 Social Genome을 트위터에 응용한 체계도



※ 출처 : <http://www.walmartlabs.com/social-genome/>

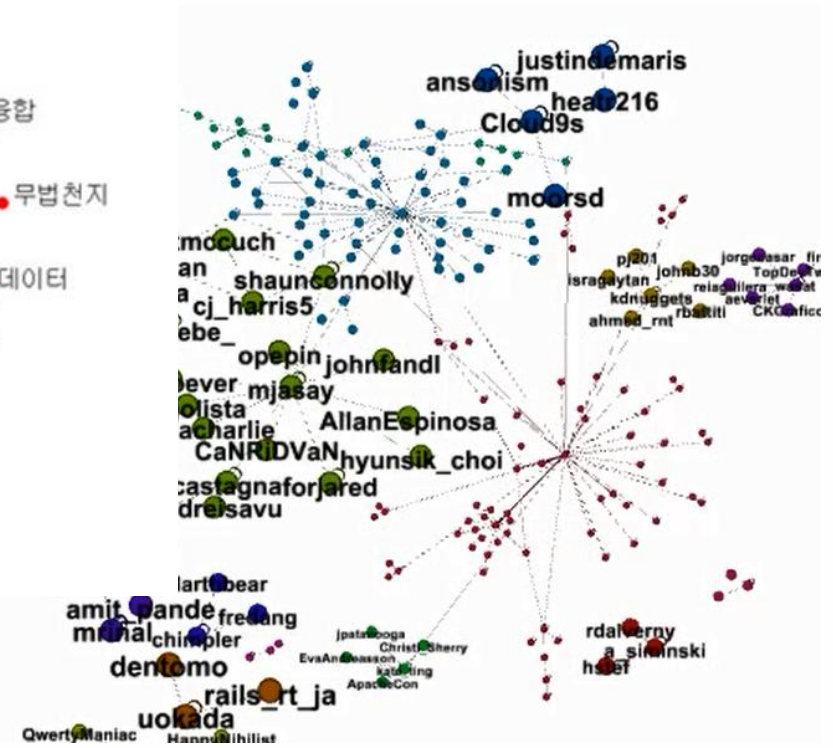
## Relational Content Analysis

<그림 5> 안철수의 소프트웨어 지도



출처 : 동아일보 기사

“대기업의 불공정 거래로 벤처나 중소기업이 성장하지 못하고 국가경제에 악순환을 불러오고 있다.”(관훈토론 2011년 3월22일)





**감사합니다**