第1讲Hadoop介绍

辜希武
IDC实验室
1694551702@qq.com

Hadoop是什么

- Hadoop : a distribution file system framework, lead by Apache
- Instance of GFS and MapReduce
  - Hadoop Distributed File System, HDFS
  - MapReduce programming model
- Hadoop was named after its creator's (Doug Cutting's) child's stuffed elephant.

Features in Hadoop

- Scalable
  - Hadoop can reliably store and process petabytes.
- Economical
  - It distributes the data and processing across clusters of commonly available computers.
    - These clusters can number into the thousands of nodes.
- Efficient
  - By distributing the data, Hadoop can process it in parallel on the nodes where the data is located.
- Reliable
  - Hadoop automatically maintains multiple copies of data and automatically redeploys computing tasks based on failures.

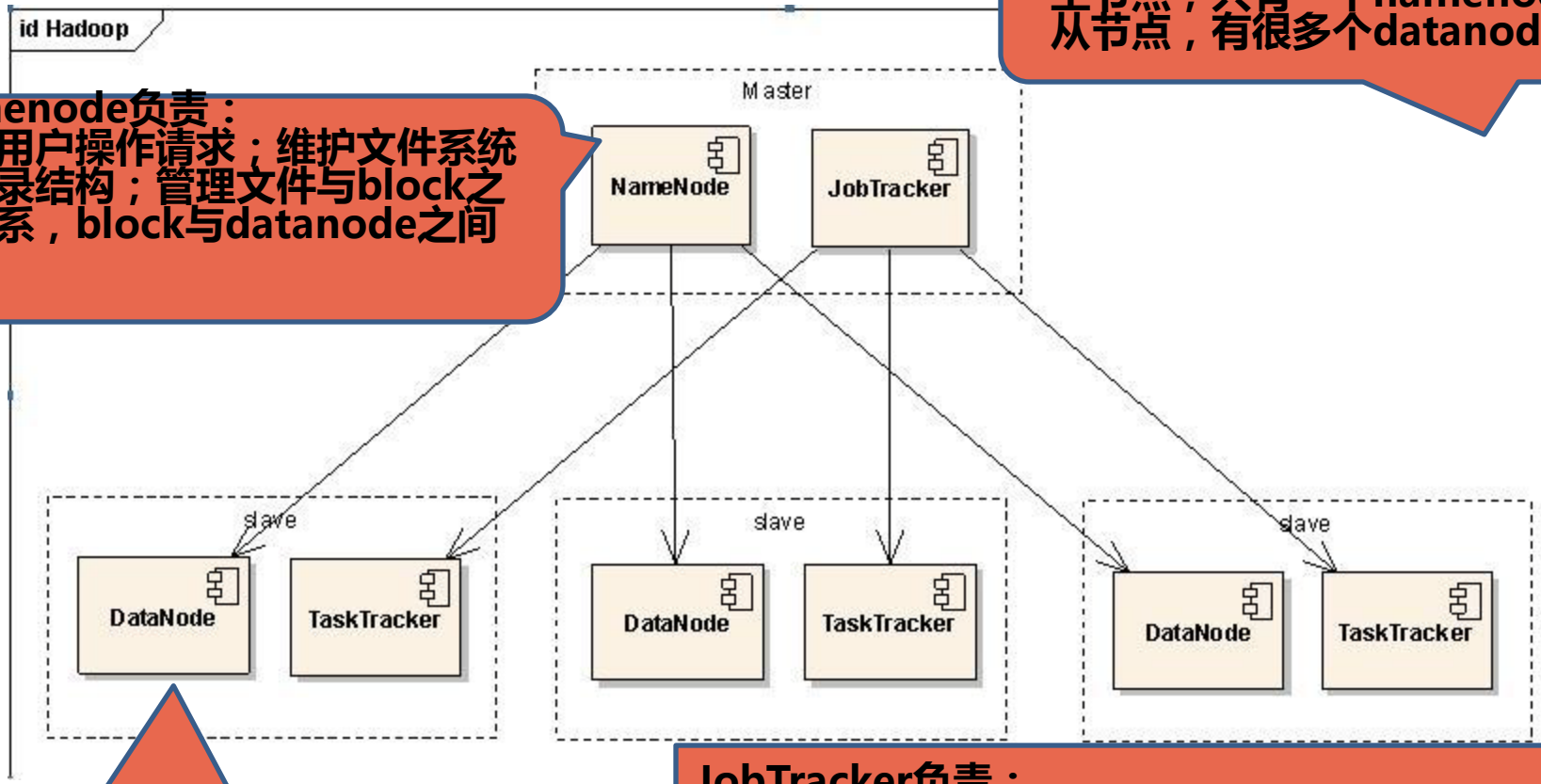
Challenges

- **Cheap nodes fail, especially if you have many**
  - Mean time between failures for 1 node = 3 years
  - MTBF for 1000 nodes = 1 day
  - Solution: Build fault-tolerance into system
- **Commodity network = low bandwidth**
  - Solution: Push computation to the data
- **Programming distributed systems is hard**
  - Solution: Data-parallel programming model: users write "map" and "reduce" functions, system handles work distribution and fault tolerance

物理分布的Hadoop集群

主从结构
主节点，只有一个namenode
从节点，有很多个datanodes

namenode负责：
接收用户操作请求；维护文件系统的目录结构；管理文件与block之间关系，block与datanode之间关系



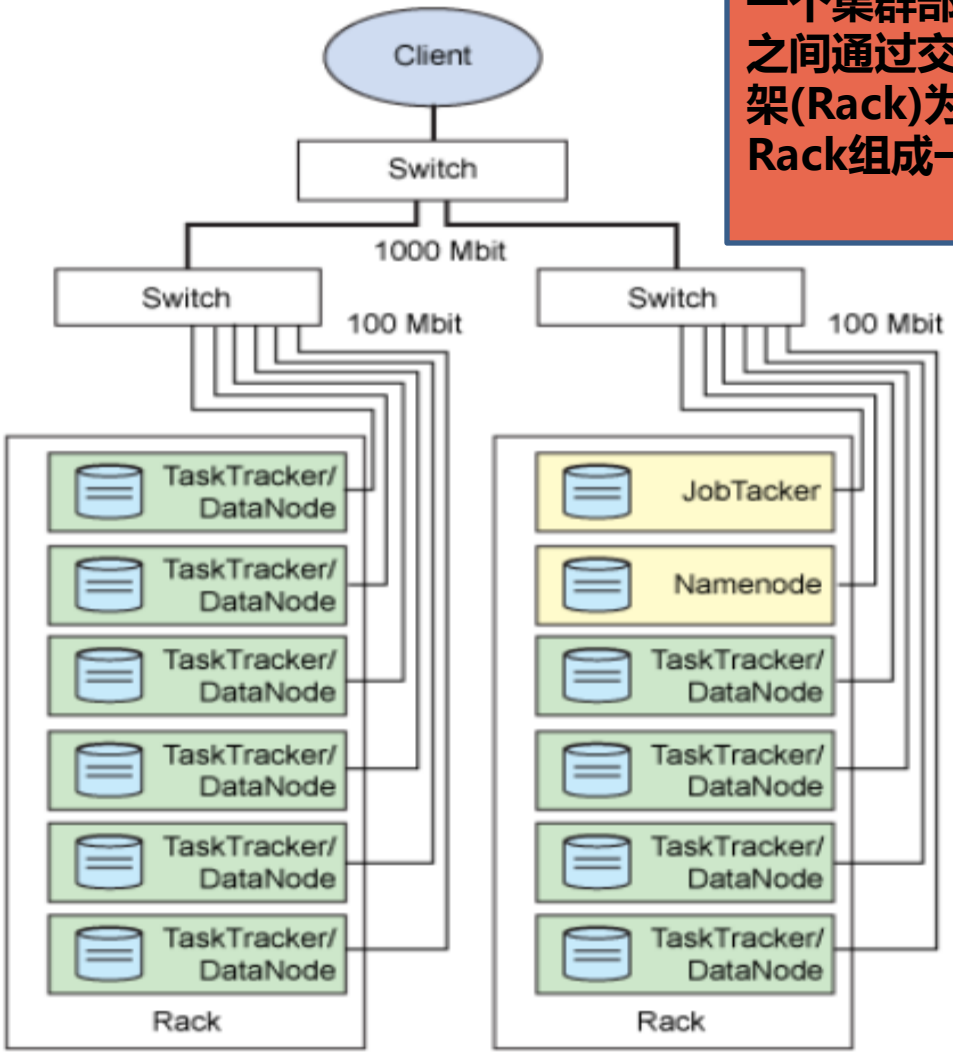
datanode负责：
存储文件；文件被分成block存储在磁盘上；为保证数据安全，文件会有多个副本

JobTracker负责：
- 接收客户提交的计算任务
- 把计算任务分给TaskTrackers执行
- 监控TaskTracker的执行情况

TaskTrackers负责：
- 执行JobTracker分配的计算任务

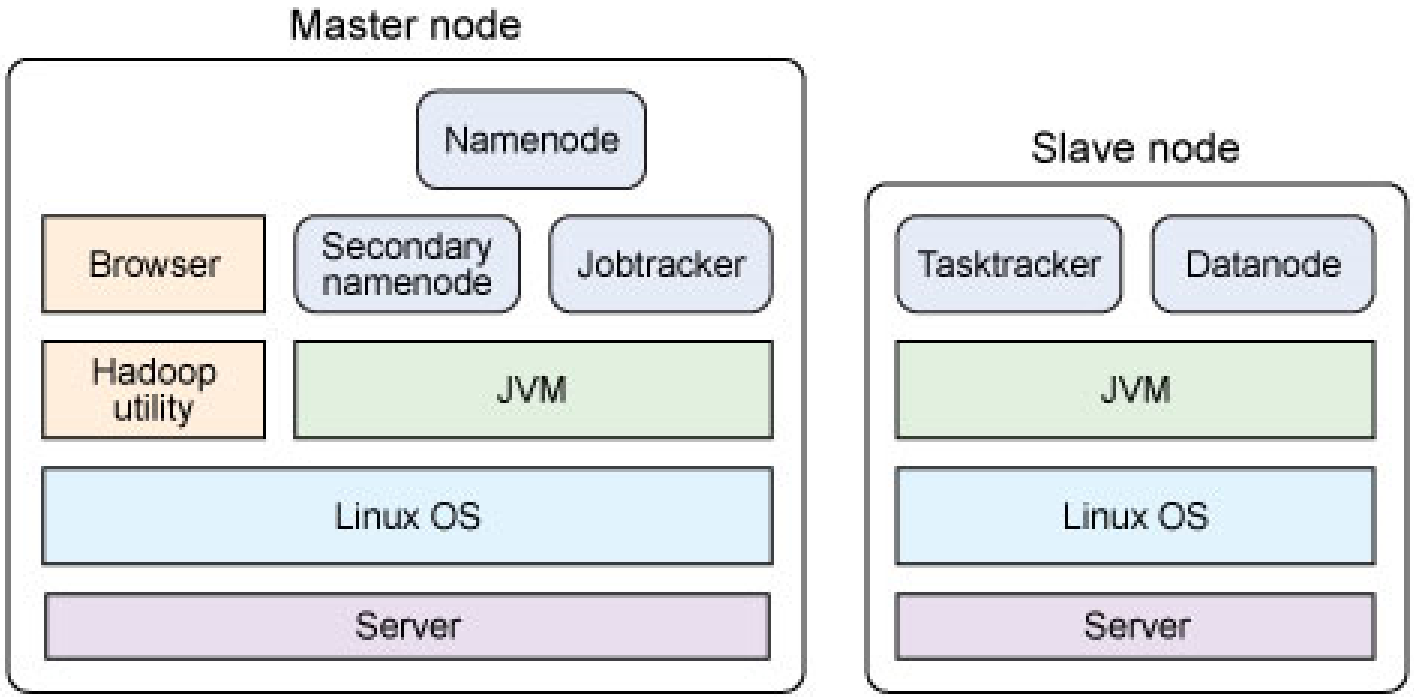
物理分布的Hadoop集群

一个集群部署在局域网环境中，节点之间通过交换机连接。多台节点以机架(Rack)为单位组织起来。多个Rack组成一个数据中心（DC）



物理部署

实际的Hadoop系统必须运行在Linux上。Namenode会有一个离线备份：SecondaryNamenode



Typical Hadoop Cluster

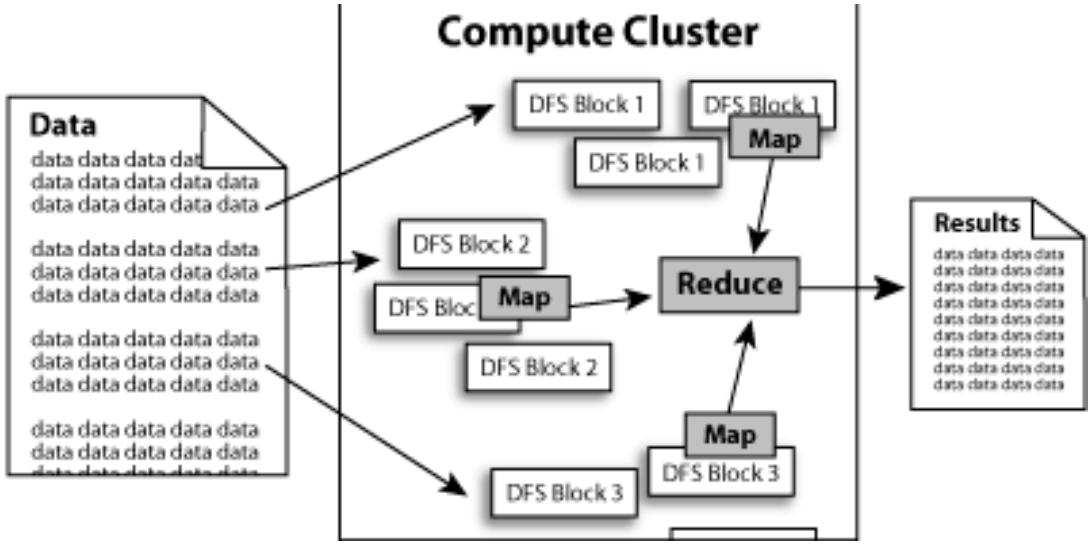
Image from http://wiki.apache.org/hadoop-data/attachments/HadoopPresentations/attachments/aw-apachecon-eu-2009.pdf

什么是HDFS?

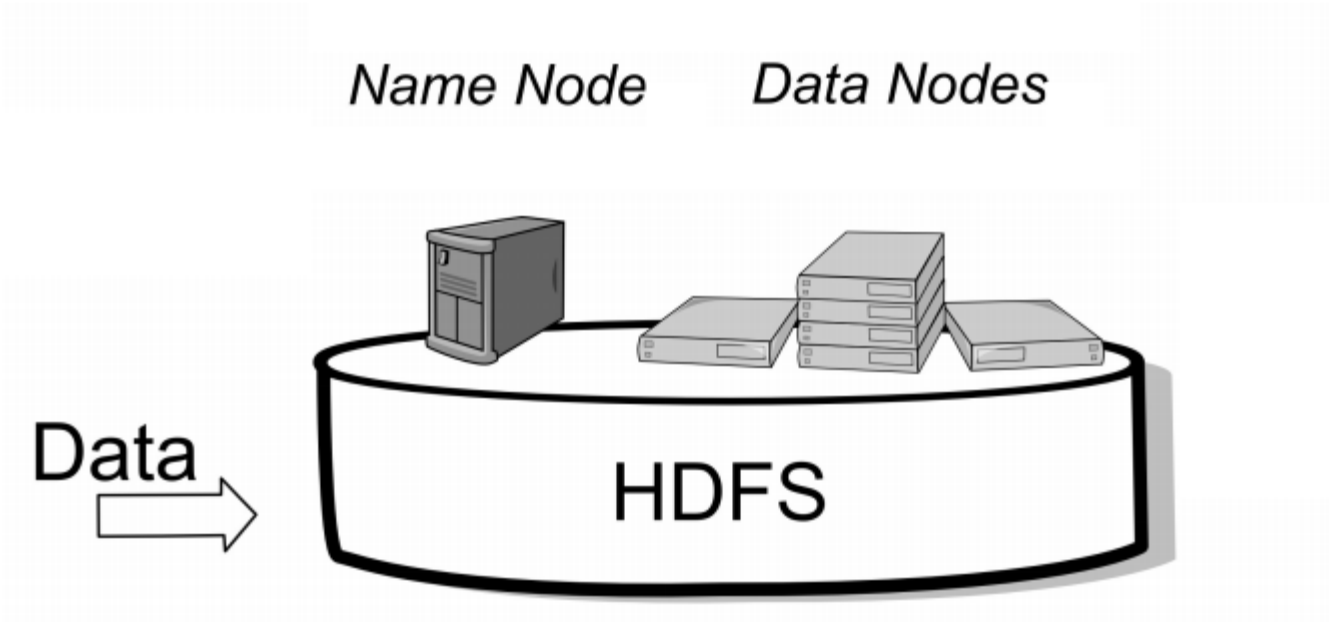
- Hadoop Distributed File System
  - Hadoop Distributed File System (HDFS) is the primary storage system used by Hadoop applications. HDFS creates multiple replicas of data blocks and distributes them on compute nodes throughout a cluster to enable reliable, extremely rapid computations.

HDFS简介

- HDFS为了做到可靠性（reliability）创建了多份数据块（data blocks）的复制（replicas），并将它们放置在服务器群的计算节点中（compute nodes），MapReduce就可以在它们所在的节点上处理这些数据了。



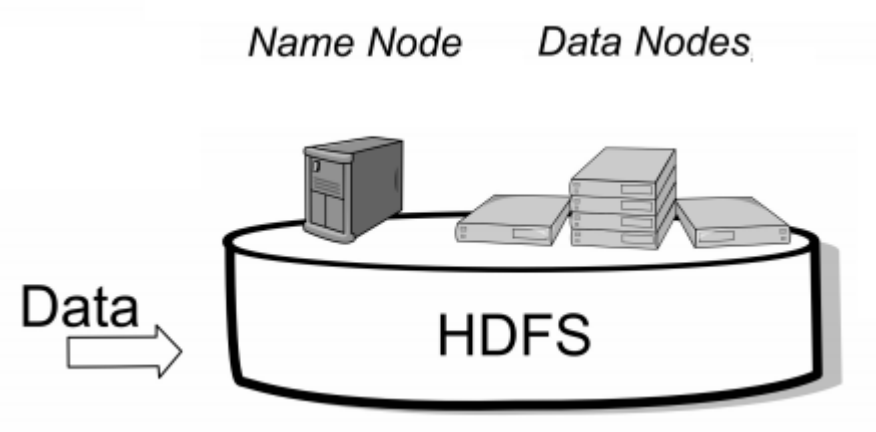
HDFS主要组件



HDFS的架构

- 主从结构
  - 主节点，只有一个: namenode
  - 从节点，有很多个: datanodes
- namenode负责：
  - 接收用户操作请求
  - 维护文件系统的目录结构
  - 管理文件与block之间关系，block与datanode之间关系
- datanode负责：
  - 存储文件
  - 文件被分成block存储在磁盘上
  - 为保证数据安全，文件会有多个副本

HDFS主要组件的功能

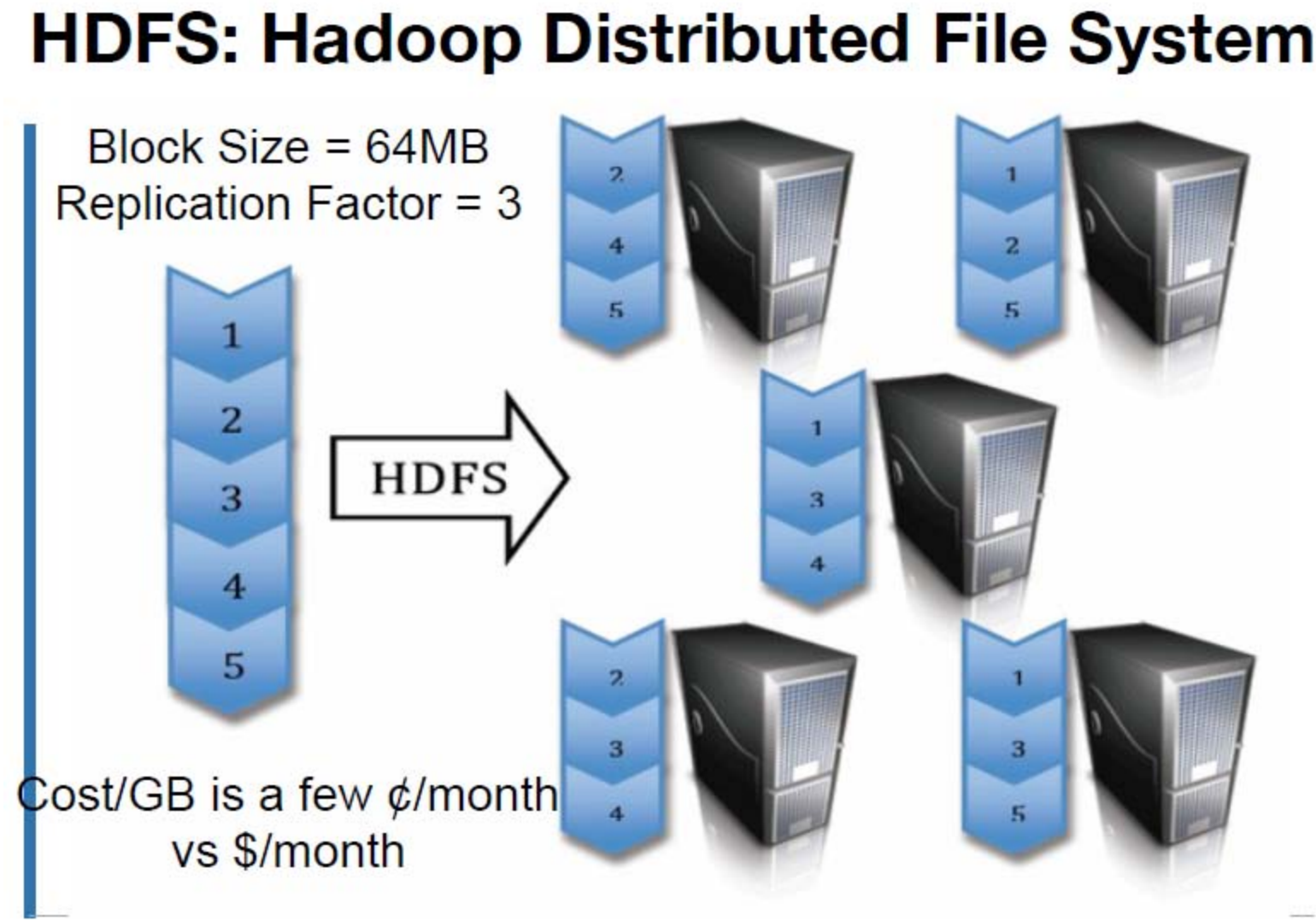


NameNode	DataNode
• 存储元数据	• 存储文件内容
•元数据保存在内存中	•文件内容保存在磁盘
• 保存文件,block ，datanode之间的映射关系	•维护了block id到datanode本地文件的映射关系

HDFS中的文件

- 文件切分成块（block，默认大小64M），以块为单位，每个块有多个副本存储在不同的机器上，副本数可在文件生成时指定（默认3）
- NameNode是主节点，存储文件的元数据如文件名，文件目录结构，文件属性（生成时间,副本数,文件权限），以及每个文件的块列表以及块所在的DataNode等等
- DataNode在本地文件系统存储文件块数据，以及块数据的校验和
- 可以创建、删除、移动或重命名文件，当文件创建、写入和关闭之后不能修改文件内容。

Hadoop中的文件以块为单位来管理



NameNode

- Namenode是一个中心服务器，单一节点（简化系统的设计和实现），负责管理文件系统的名字空间(namespace)以及客户端对文件的访问。
- 文件操作，NameNode负责文件元数据的操作，DataNode负责处理文件内容的读写请求，跟文件内容相关的数据流不经NameNode，只会询问它跟那个DataNode联系，否则NameNode会成为系统的瓶颈
- 副本存放在哪些DataNode上由NameNode来控制，根据全局情况做出块放置决定，读取文件时NameNode尽量让用户先读取最近的副本，降低带块消耗和读取时延
- Namenode全权管理数据块的复制，它周期性地从集群中的每个Datanode接收心跳信号和块状态报告(Blockreport)。接收到心跳信号意味着该Datanode节点工作正常。块状态报告包含了一个该Datanode上所有数据块的列表

DataNode

- 一个数据块在DataNode以文件（本地）存储在磁盘上，包括两个文件，一个是数据本身，一个是元数据包括数据块的长度，块数据的校验和，以及时间戳
- DataNode启动后向NameNode注册，通过后，周期性（1小时）的向NameNode上报所有的块信息。
- 心跳是每3秒一次，心跳返回结果带有NameNode给该DataNode的命令如复制块数据到另一台机器，或删除某个数据块。如果超过10分钟没有收到某个DataNode 的心跳，则认为该节点不可用。
- 集群运行中可以安全加入和退出一些机器

什么MapReduce Programming Model

- MapReduce is a programming model and an associated implementation for processing and generating large data sets
- Put forward by Google
- Schema of map and reduce functions

  map: input → list(k, v)

  reduce: (k,list(v)) → output
- This model can express the work of realistic world
- More than 1000 MapReduce procedures are run on Google racks everday

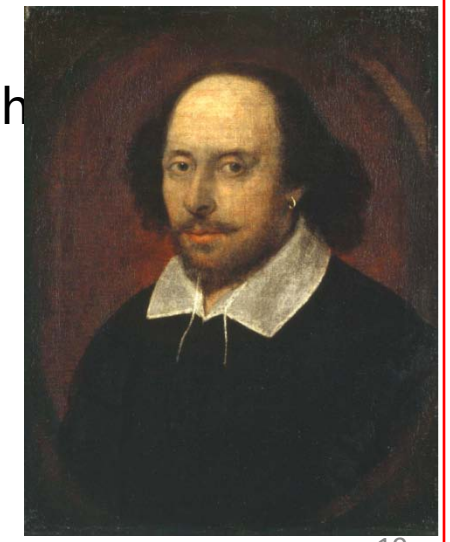
Distributed Indexing： MapReduce Example
Information Retrieval： Answers to query

☐ Antony and Cleopatra, Act III, Scene ii

Agrippa [Aside to DOMITIUS ENOBARBUS]: Why, Enobarbus,
When Antony found Julius **Caesar** dead,
He cried almost to roaring; and he wept
When at Philippi he found **Brutus** slain.

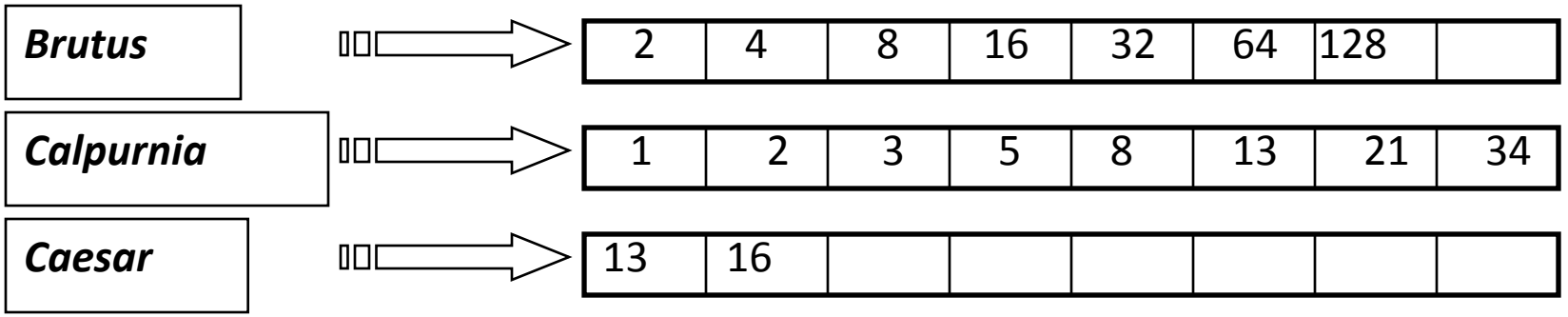
☐ Hamlet, Act III, Scene ii

Lord Polonius: I did enact Julius **Caesar** I was killed i' th
Capitol; **Brutus** killed me.



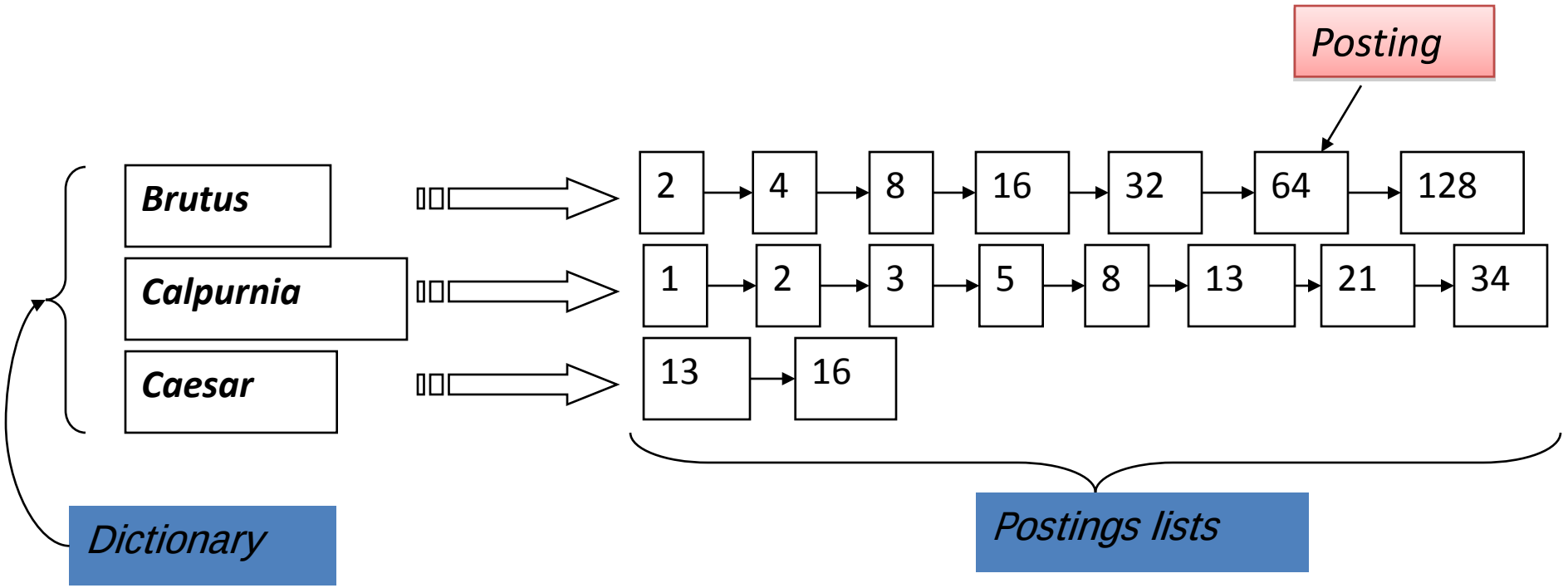
Distributed indexing: MapReduce Example Inverted index

- For each term T, we must store a list of all documents that contain T.



What happens if the word ***Caesar*** is added to document 14?

Distributed indexing: MapReduce Example Inverted index



The posting entries are sorted by docId in increasing order

Distributed indexing: MapReduce Example

- Maintain a master machine directing the indexing job – considered "safe".
- Break up indexing into sets of (parallel) tasks.
- Master machine assigns each task to an idle machine from a pool.

Distributed indexing: MapReduce Example Parallel tasks

- We will use two sets of parallel tasks
  - Parsers
  - Inverters
- Break the input document corpus into splits
- Each split is a subset of documents

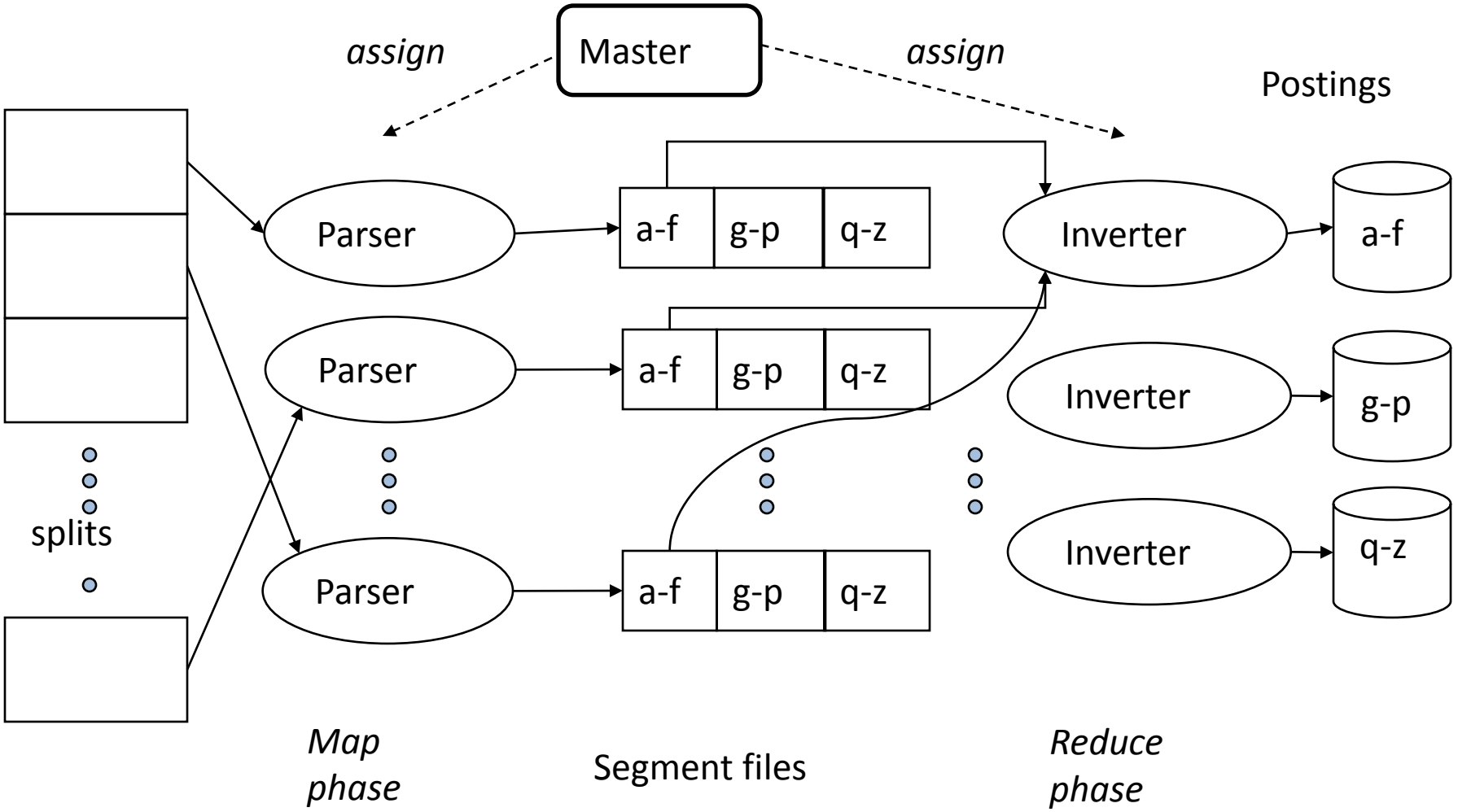
Distributed indexing: MapReduce Example Parsers

- Master assigns a split to an idle parser machine
- Parser reads a document at a time and generate (term, doc) pairs
- Parser writes pairs into j partitions
- Partition is based on the terms' first letters
- Each partition is for a range of terms' first letters
  - (e.g., a-f, g-p, q-z) – here j=3.
- Now to complete the index inversion

Distributed indexing: MapReduce Example Inverters

- An inverter collects all (term,doc) pairs (= postings) for one term-partition.
- Sorts and writes to postings lists

Distributed indexing: MapReduce Example Data flow



MapReduce

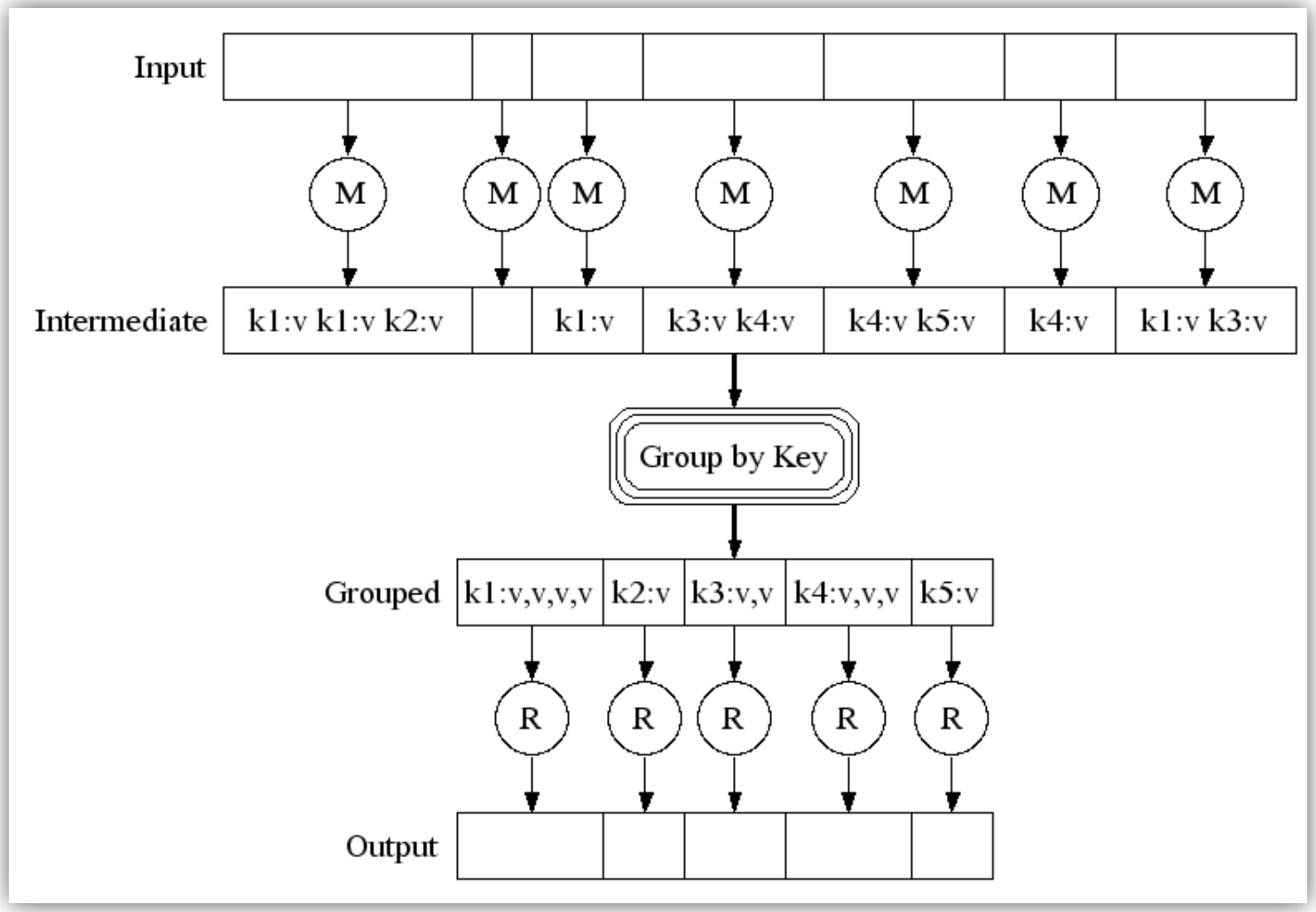
- The index construction algorithm we just described is an instance of MapReduce.
- MapReduce (Dean and Ghemawat 2004) is a robust and conceptually simple framework for
- distributed computing ...
- ... without having to write code for the distribution part.
- Google indexing system (ca. 2002) consists of a number of phases, each phase is implemented in MapReduce.

Schema for index construction in MapReduce

- **Schema of map and reduce functions**
- map: input → list(k, v) reduce: (k,list(v)) → output
- **Instantiation of the schema for index construction**
- map: web collection → list(termID, docID)
- reduce: (<termID1, list(docID)>, <termID2, list(docID)>, …) → (postings list1, postings list2, …)
- **Example for index construction**
- map: d2 : C died. d1 : C came, C c’ ed. → (<C, d2>, <died,d2>, <C,d1>, <came,d1>, <C,d1>, <c’ ed, d1>
- reduce: (<C,(d2,d1,d1)>, <died,(d2)>, <came,(d1)>, <c’ ed,(d1)>) → (<C,(d1:2,d2:1)>, <died,(d2:1)>, <came,(d1:1)>, <c’ ed,(d1:1)>)

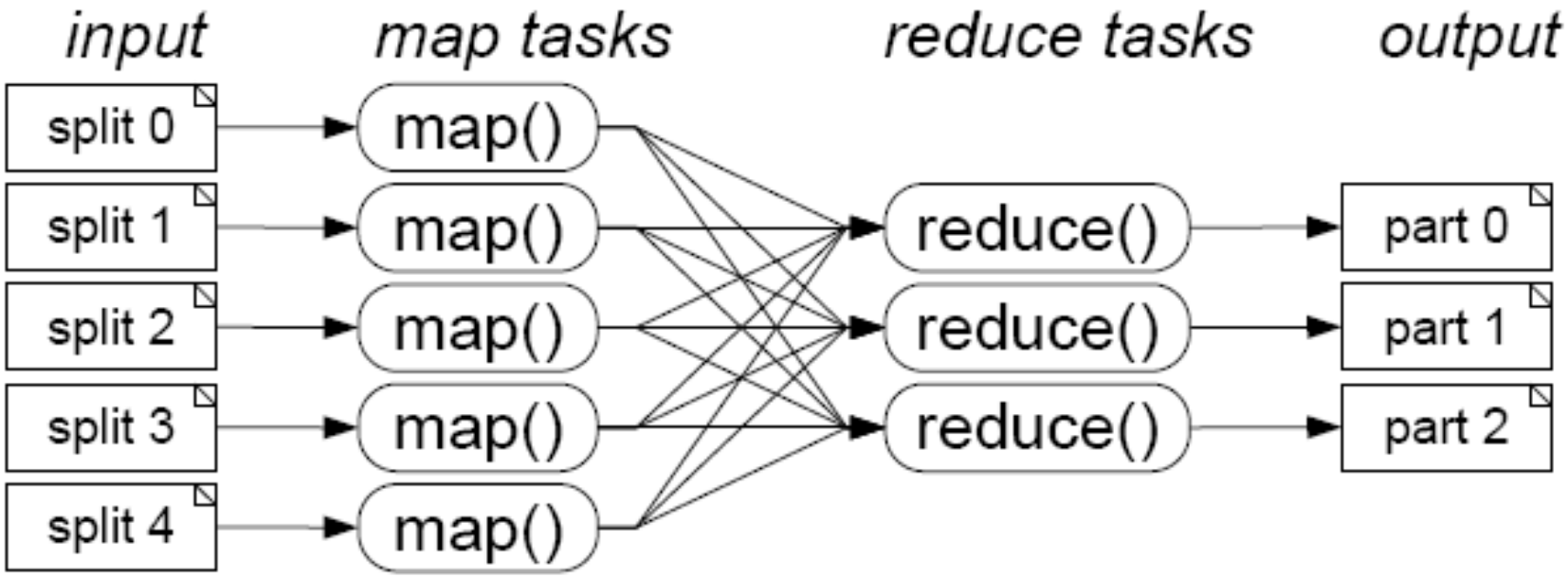
MapReduce Data flow

M：Mapper
R：Reducer
Input被分成Split



MapReduce Job Processing

用户以Job(作业)为单位提交计算任务
JobTracker会将Job划分成多个并行执行的Task分发到各个Datanode执行



Experience: Rewrite of Production Indexing System

- **Rewrote Google's production indexing system using MapReduce**
  - New code is simpler, easier to understand
  - MapReduce takes care of failures, slow machines
  - Easy to make indexing faster by adding more machines

Usage: MapReduce jobs run in August 2004

Number of jobs	29,423	
Average job completion time	634	secs
Machine days used	79,186	days
Input data read	3,288	TB
Intermediate data produced	758	TB
Output data written	193	TB
Average worker machines per job	157	
Average worker deaths per job	1.2	
Average map tasks per job	3,351	
Average reduce tasks per job	55	

MapReduce是什么

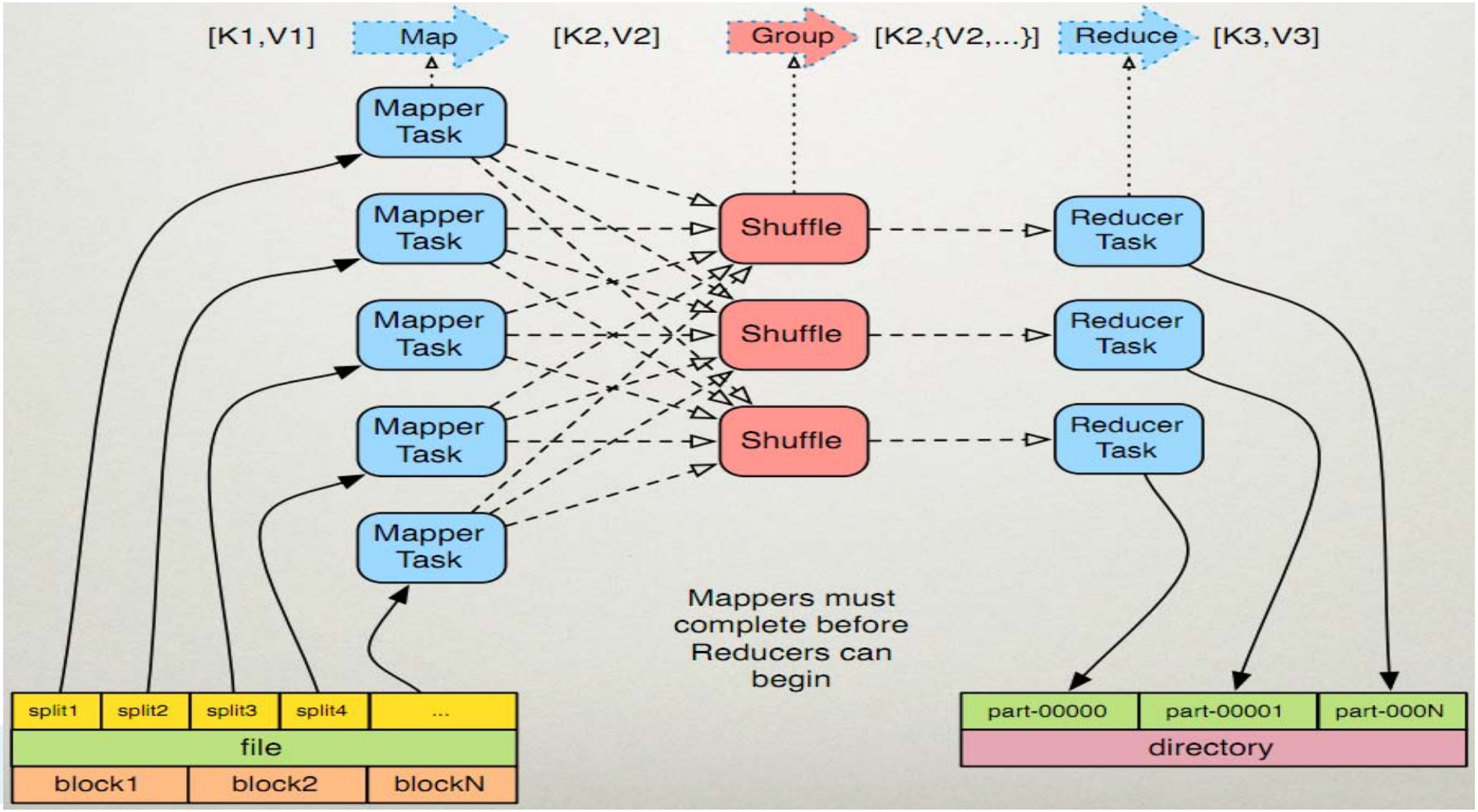
□MapReduce是一种分布式计算模型，由Google提出，主要用于搜索领域，解决海量数据的计算问题.

□MR由两个阶段组成：Map和Reduce，用户只需要实现map()和reduce()两个函数，即可实现分布式计算，用户只需专注于领域知识，解决领域问题。无需考虑分布式计算底层细节如：网络通信、任务分发、任务调度等

MapReduce的架构

- 主从结构
  - 主节点，只有一个: JobTracker
  - 从节点，有很多个: TaskTrackers
- JobTracker负责：
  - 接收客户提交的计算任务
  - 把计算任务分给TaskTrackers执行
  - 监控TaskTracker的执行情况
- TaskTrackers负责：
  - 执行JobTracker分配的计算任务

Mapreduce原理



Map和Reduce任务处理步骤

◆执行步骤：

1. map任务处理

1.1 读取输入文件内容，解析成key、value对。对输入文件的每一行，解析成key、value对。每一个键值对调用一次map函数。

1.2 写自己的逻辑，对输入的key、value处理，转换成新的key、value输出。

1.3 对输出的key、value进行分区。

1.4 对不同分区的数据，按照key进行排序、分组。相同key的value放到一个集合中。

1.5 (可选)分组后的数据进行归约。

2.reduce任务处理

2.1 对多个map任务的输出，按照不同的分区，通过网络copy到不同的reduce节点。

2.2 对多个map任务的输出进行合并、排序。写reduce函数的逻辑，对输入的key、value处理，转换成新的key、value输出。

2.3 把reduce的输出保存到文件中。

Map、Reduce键值对格式

函数	输入键值对	输出键值对
map()	<k1,v1>	<k2,v2>
reduce()	<k2,{v2}>	<k3,v3>

From CS Foundations to MapReduce

Consider a large data collection:

{

web weed green sun moon land part web green ,

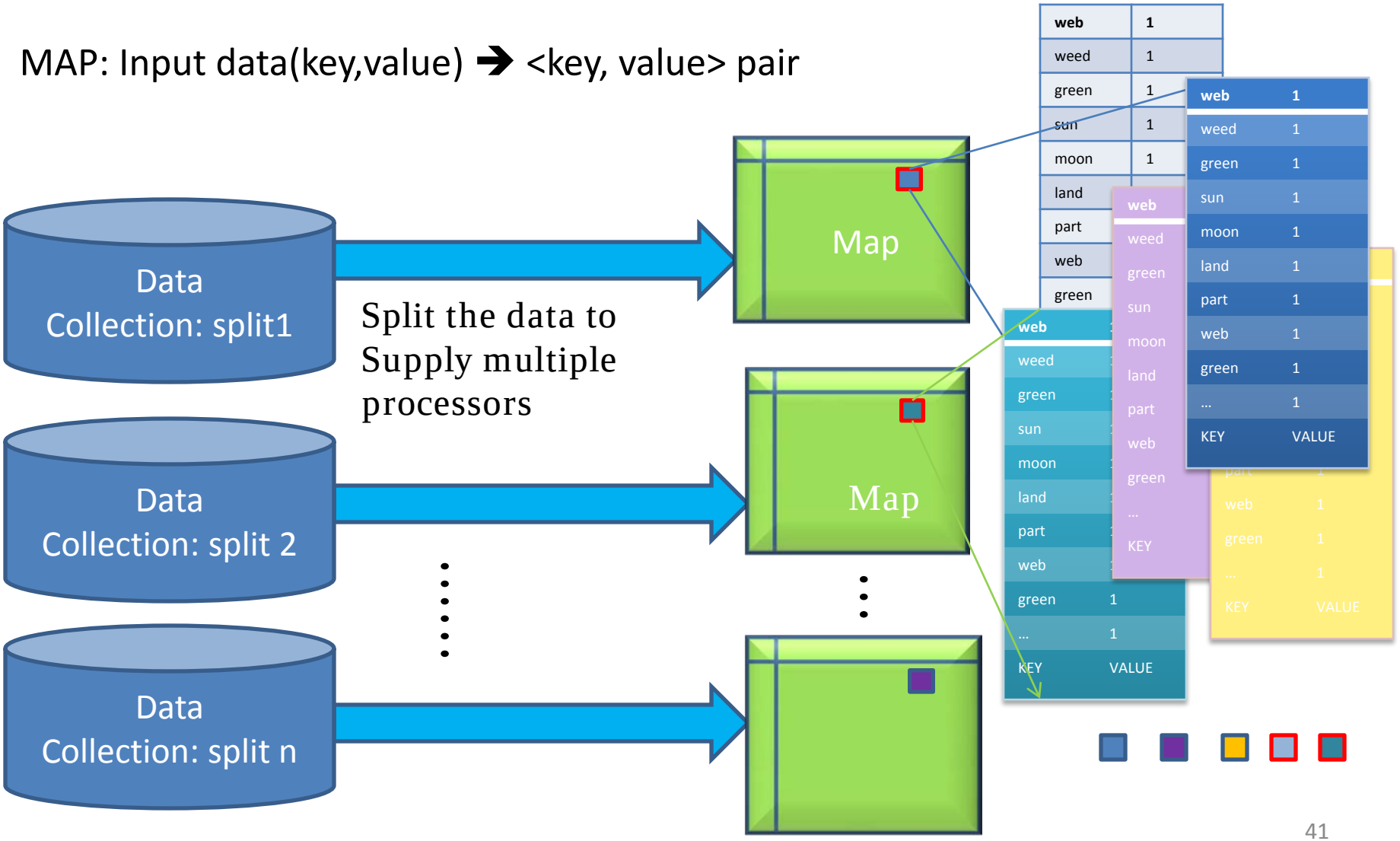
…

}

Problem: Count the occurrences of the different words in the collection.

Map Operation

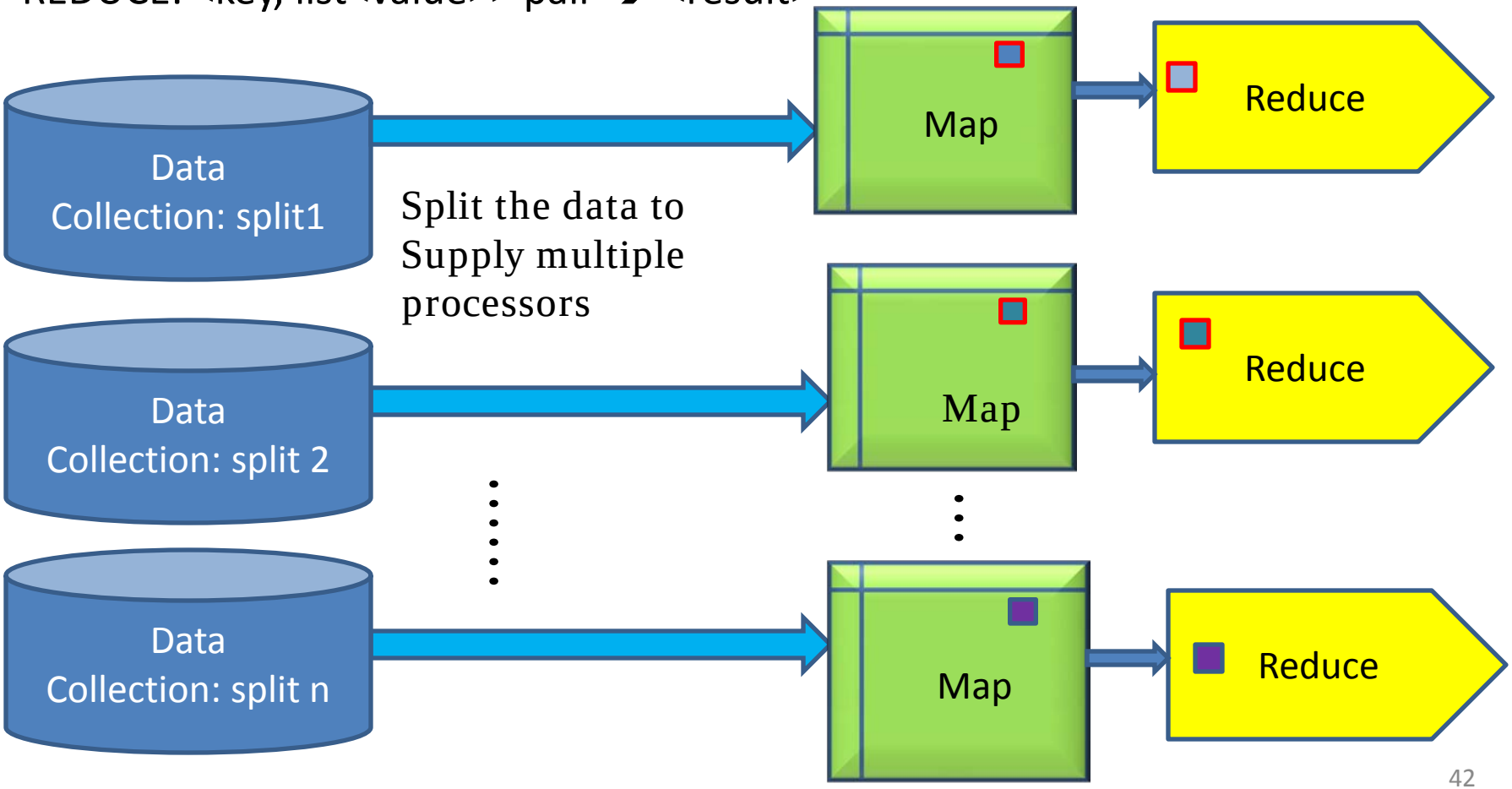
MAP: Input data(key,value) ➔ <key, value> pair



Reduce Operation

MAP: Input data(key,value) ➔ <key, value> pair

REDUCE: <key, list<value>> pair ➔ <result>



Lifecycle of a MapReduce Job

File Edit Options Buffers Tools Java Help

```
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.hasNext()) { sum += values.next().get(); }
      output.collect(key, new IntWritable(sum));
  }}

  public static void main(String[] args) throws Exception {
    JobConf conf = new JobConf(WordCount.class);
    conf.setJobName("wordcount");
    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);
  }}
```

--:--- **mapreduce.java** All L9 (Java/l Abbrev)----------

Wrote /home/shivnath/Desktop/mapreduce.java

Map function

Reduce function

Run this program as a MapReduce job

实现Mapper

Mapper类要继承Mapper<Object, Text, Text, IntWritable>，Mapper为抽象类，这里用到了泛型

```
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);
            }
        }
    }
    //其它代码
}
```

map函数原型，其中key，value是输入context为用户代码与MR系统交互的上下文

StringTokenizer将字符串分成一个个的单词

将token写入word

由于token出现一次，因此将键值对<token,1>写入context。MR框架会将context中的键值对交给Reducer处理

IntWritable、Text是Hadoop API里定义的数据类型
Text 相当于Java 里的String
IntWritable相当于Java里的Integer

实现Reducer

Reducer类要继承Reducer<Text,IntWritable,Text,IntWritable> ，Reducer为抽象类，这里用到了泛型

```
public class WordCount {
    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();     //每个val=1，进行累加
                result.set(sum);      //得到token的词频
                context.write(key, result);  //将<token,词频>写入context，由contex写到文
                                             //HDFS文 件里
            }
        }
    //其它代码
    }
}
```

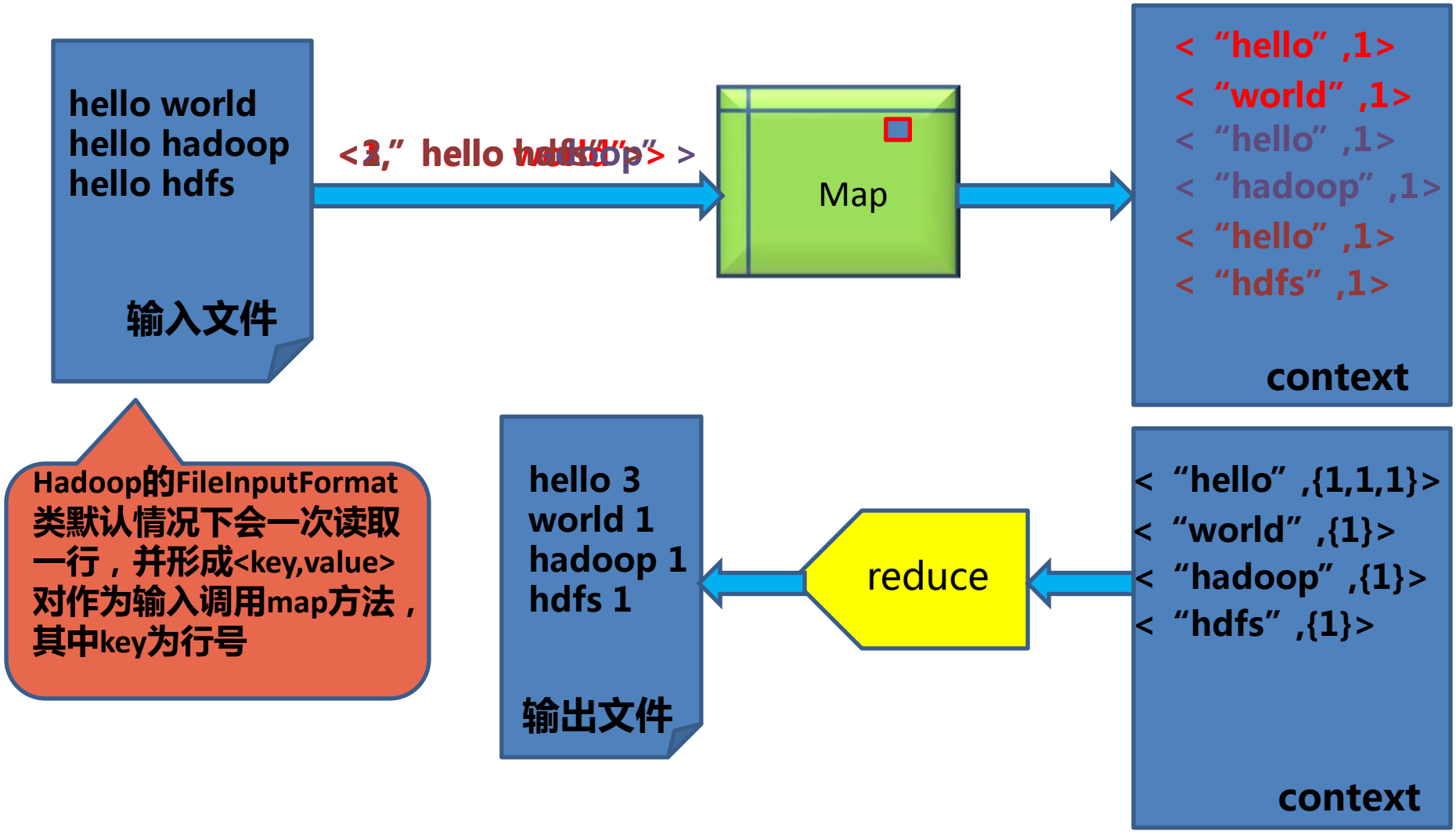
reduce函数原型，其中key，values是输入context为用户代码与MR系统交互的上下文

遍历集合里的每个value

Iterable<IntWritable>是Hadoop定义的集合类型，集合里元素类型是IntWritable

map的输出通过context交给reduce函数前，要把相同key的value都合并到一个集合里，因此reduce的第二个参数是集合类型。第一个参数为key，即token

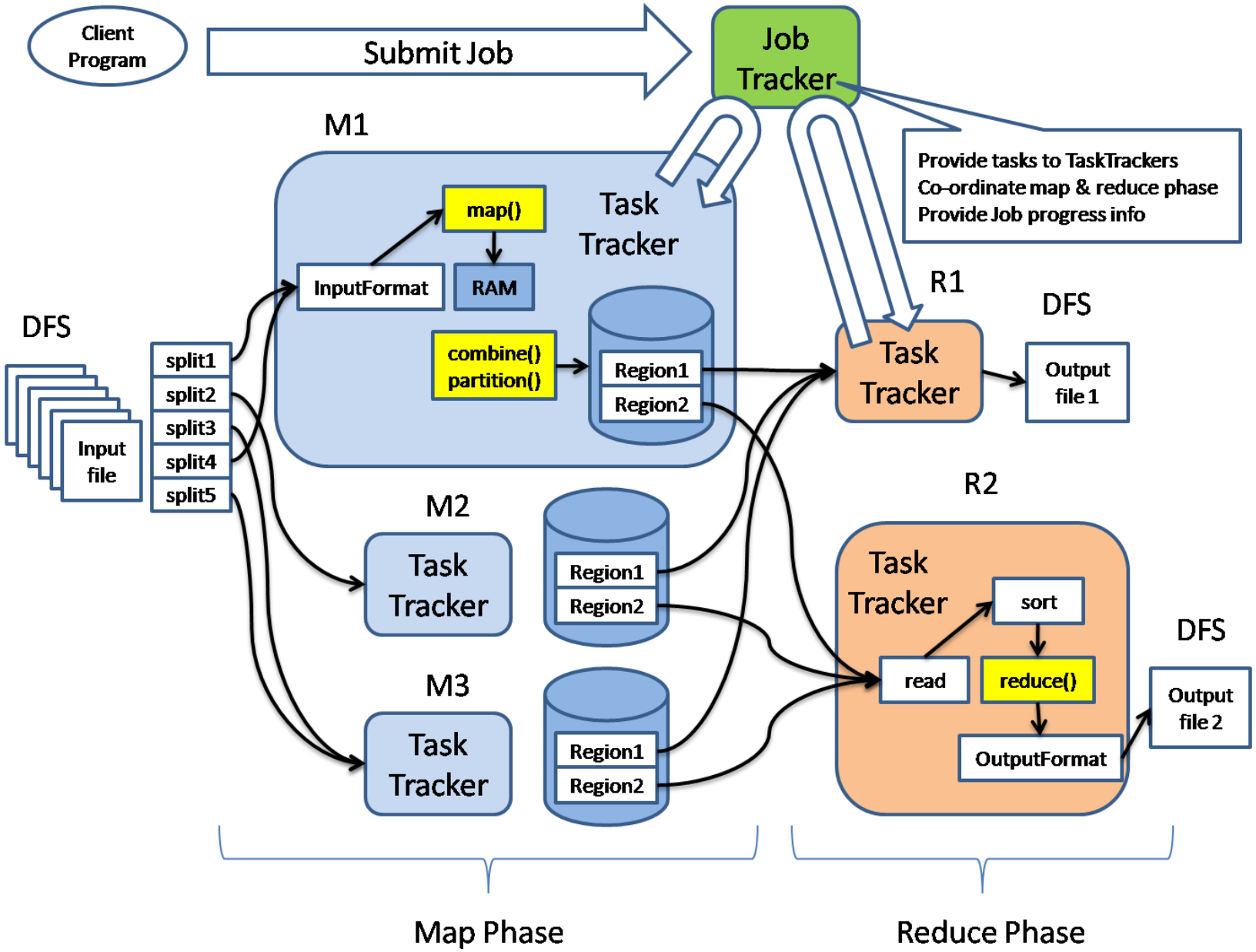
map和reduce方法之间的数据传递示例



启动MapReduce Job

```
public class WordCount {
    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();    //读取Hadoop配置信息
        String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
        if (otherArgs.length != 2) {
            System.err.println("Usage: wordcount <in> <out>");
            System.exit(2);
        }
        Job job = new Job(conf, “word count”);    //创建MR Job
        job.setJarByClass(WordCount.class);          //设置启动类
        job.setMapperClass(TokenizerMapper.class);   //设置Mapper类
        job.setReducerClass(IntSumReducer.class);    //设置Reducer类
        job.setOutputKeyClass(Text.class);           //设置输出Key的类型
        job.setOutputValueClass(IntWritable.class);  //设置输出值的类型
        FileInputFormat.addInputPath(job, new Path(otherArgs[0]));   //设置输入文件目录
        FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));   // 设置输出文件目录
        System.exit(job.waitForCompletion(true) ? 0 : 1);  //等待Job完成
    }
}
```

MapReduce执行流程



Hadoop大事记

07年1月-- 研究集群到达900个节点。

07年4月-- 研究集群达到两个1000个节点的集群。

08年4月-- 赢得世界最快1 TB数据排序在900个节点上用时209秒。

08年10月-- 研究集群每天装载10 TB的数据。

09年3月-- 17个集群总共24 000台机器。

09年4月-- 赢得每分钟排序，59秒内排序500 GB(在1400个节点上)和173分钟内排序100 TB数据(在3400个节点上)。

Hadoop/MapReduce应用

- MapReduce的应用：

  并行计算处理引擎

  【1】日志分析

  【2】排序

  【3】搜索

  【4】广告计算，广告优化、分析，点击流分析，链接分析

  【5】搜索关键字进行内容分类

  【6】搜索引擎，创建索引

  【7】word 计数，统计值计算，统计数据，过滤，分析，查询

  【8】垃圾数据分析

  【9】数据分析

  【10】机器学习

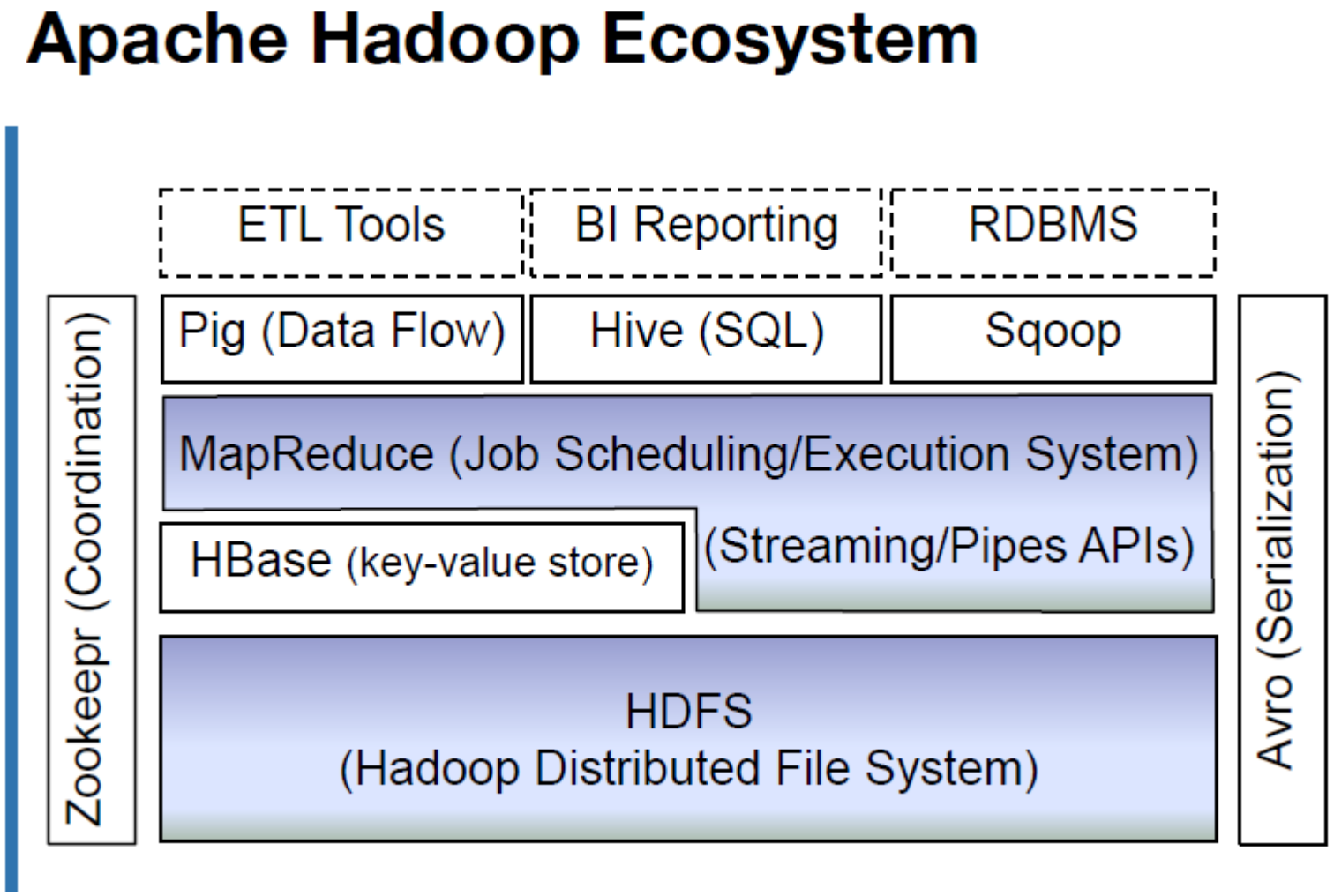
  【11】数据挖掘

  【12】大规模图像转换（纽约时报使用Hadoop 和EC2在36个小时内将4TB的TIFF图像—包括405K大TIFF图像，3.3M SGML文章和405K XML文件 — 转换为800K适合在Web上使用的PNG图像。）

Example Applications and Organizations using Hadoop

- A9.com – Amazon: To build Amazon's product search indices; process millions of sessions daily for analytics, using both the Java and streaming APIs; clusters vary from 1 to 100 nodes.
- Yahoo! : More than 100,000 CPUs in ~20,000 computers running Hadoop; biggest cluster: 2000 nodes (2*4cpu boxes with 4TB disk each); used to support research for Ad Systems and Web Search
- AOL : Used for a variety of things ranging from statistics generation to running advanced algorithms for doing behavioral analysis and targeting; cluster size is 50 machines, Intel Xeon, dual processors, dual core, each with 16GB Ram and 800 GB hard-disk giving us a total of 37 TB HDFS capacity.
- Facebook: To store copies of internal log and dimension data sources and use it as a source for reporting/analytics and machine learning; 320 machine cluster with 2,560 cores and about 1.3 PB raw storage;

Hadoop生态系统


Apache Hadoop Ecosystem
ETL Tools
BI Reporting
RDBMS
Pig (Data Flow)
Hive (SQL)
Sqoop
MapReduce (Job Scheduling/Execution System)
(Streaming/Pipes APIs)
HBase (key-value store)
HDFS
(Hadoop Distributed File System)
Zookeepr (Coordination)
Avro (Serialization)

Hadoop生态系统

- **Avro**
  A data serialization system with scripting languages.
- **HBase**
  A scalable, distributed database for large tables.
- **Hive**
  Data summarization and ad hoc querying.
- **Pig**
  A high-level data-flow language for parallel computation.
- **Sqoop**
  Sqoop is a tool designed for efficiently transferring bulk data between Apache Hadoop and structured datastores such as relational databases.
- **Zookeeper**
  Coordination service for distributed applications.

References

1. Apache Hadoop Tutorial: http://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html
2. Dean, J. and Ghemawat, S. 2008. **MapReduce: simplified data processing on large clusters.** *Communication of ACM* 51, 1 (Jan. 2008), 107-113.
3. Cloudera Videos by Aaron Kimball: http://www.cloudera.com/hadoop-training-basic
4. http://www.cse.buffalo.edu/faculty/bina/mapreduce.html