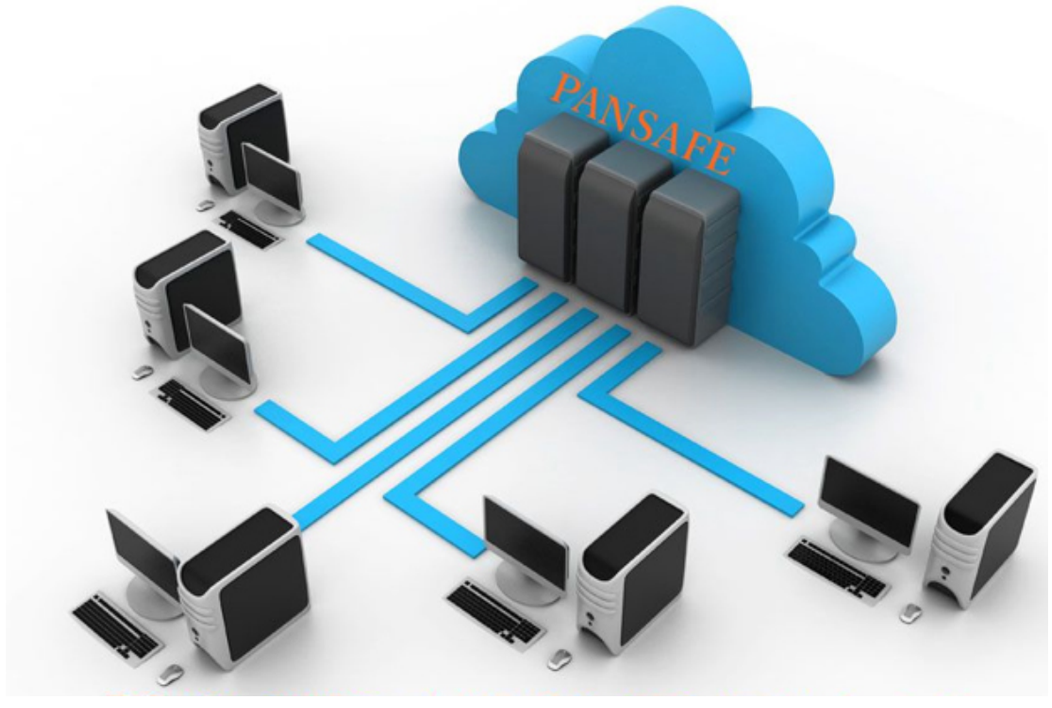
PaaS及Hadoop技术

赵来平
天津大学软件学院
laiping@tju.edu.cn

目录

- PaaS
- Hadoop
  - 分布式文件系统——GFS
  - 分布式数据处理——Mapreduce

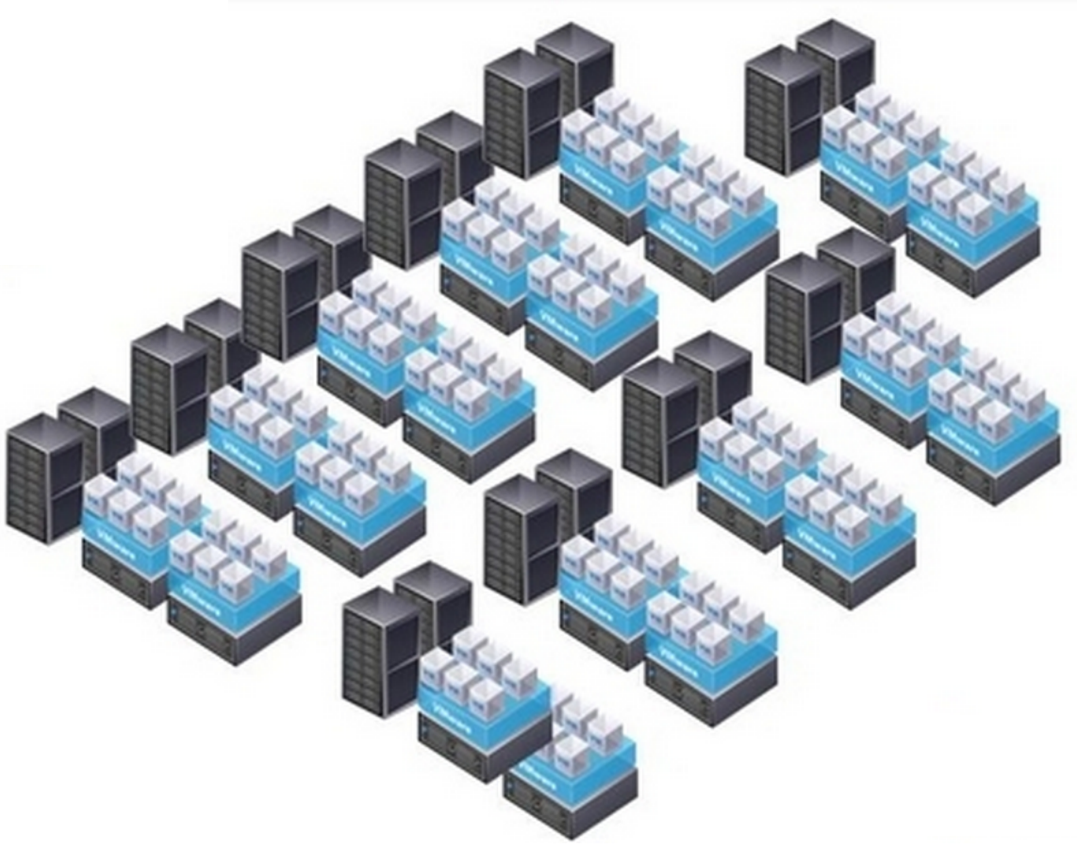


有IaaS还不够?

- IaaS为客户提供了大量的虚拟机资源，但是并不主动帮助用户根据实际需求量而弹性的增减资源。
  - 资源的弹性增减量，完全由用户自主决定。
- 用户需求：
  - 任务向计算节点的自动调度和派发；
  - 应用程序运行时环境；
  - 应用程序开发平台、测试平台。

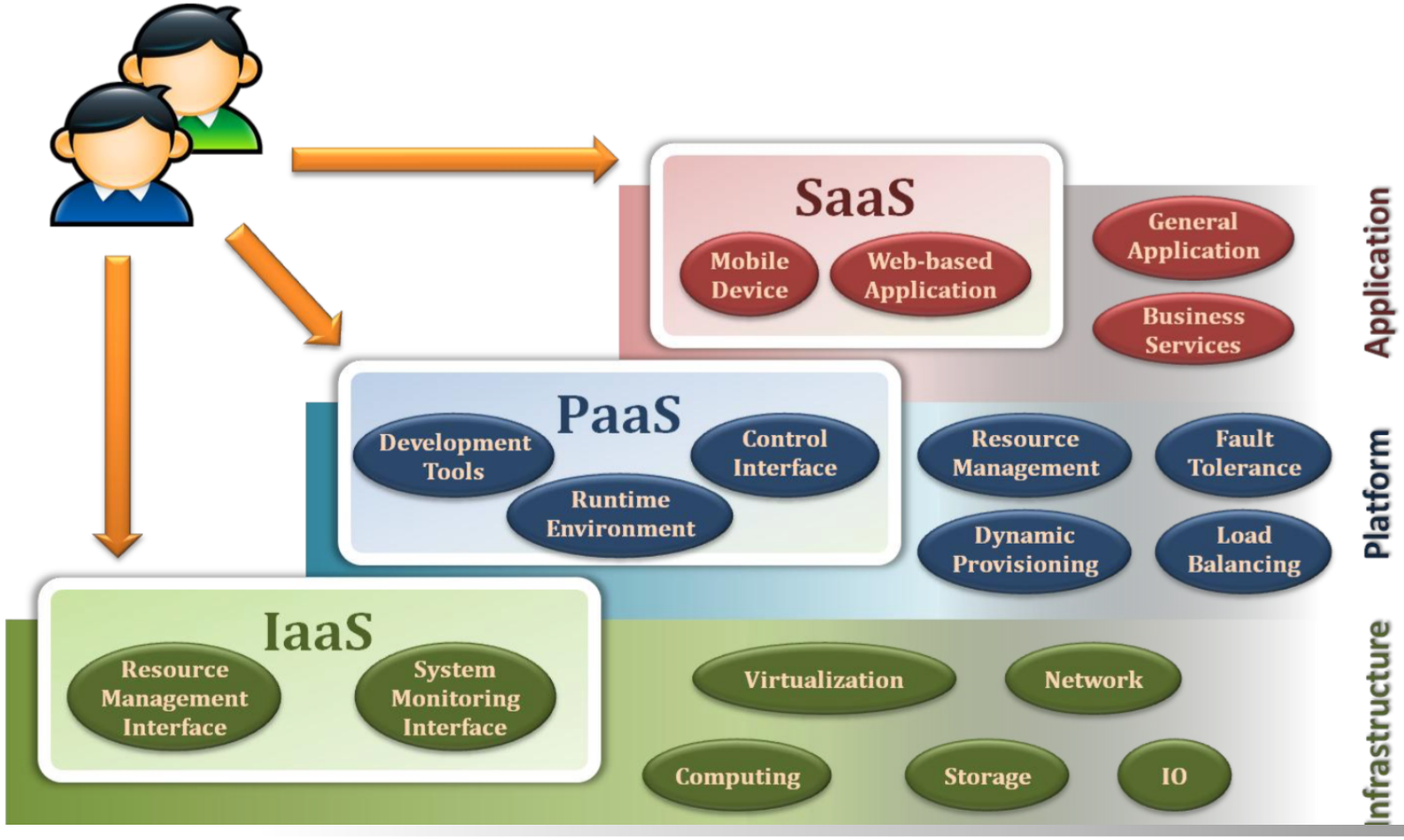
更多用户需求

- 更多需求..
  - 大规模资源量的抽象及管理；
  - 定制的运行环境；
  - 自动化部署机制；
  - 任务的分发、跟踪及管理；
  - 访问控制、认证
  - ...



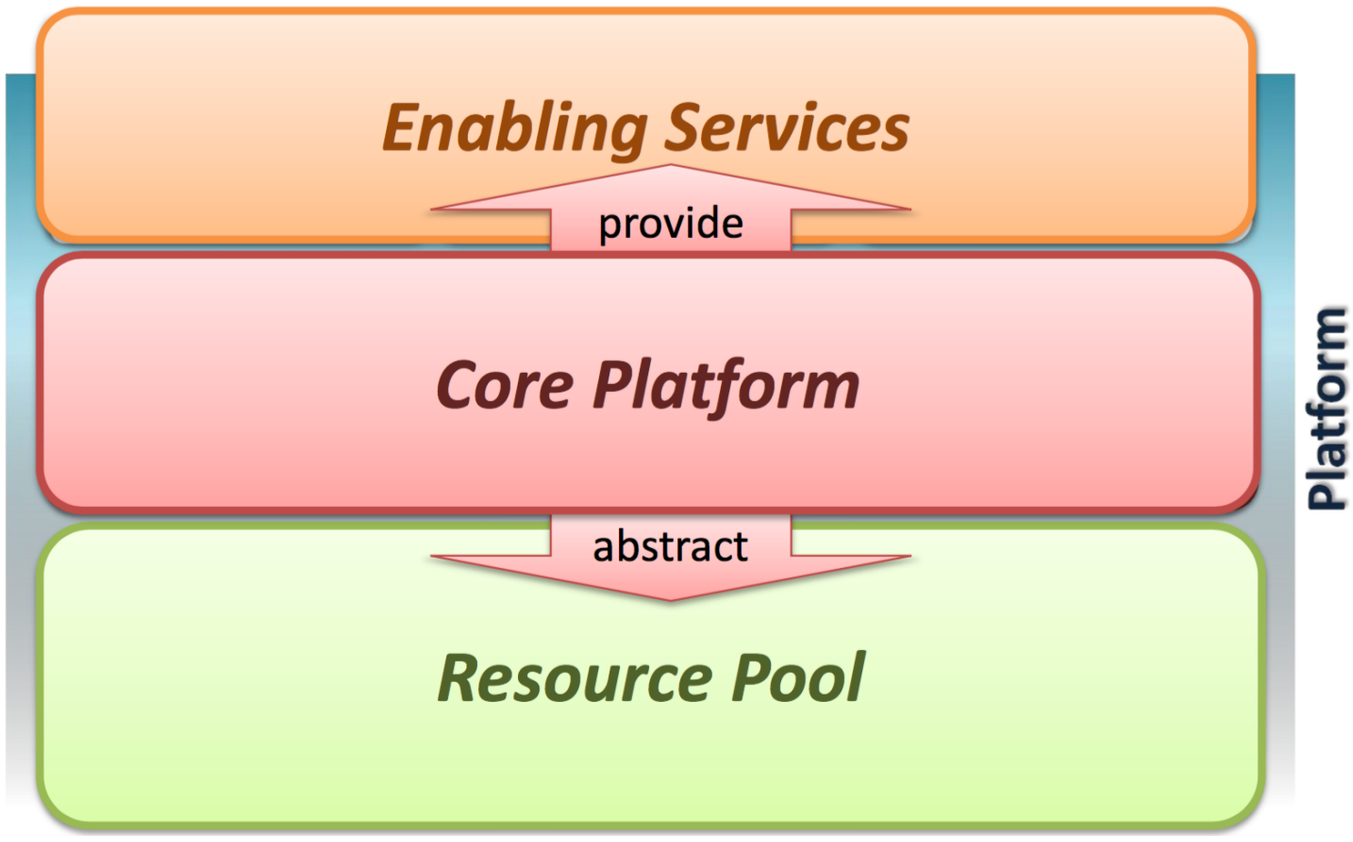
平台即服务PaaS

- Platform as a Service (PaaS)：
  - 抽象了硬件基础设施、OS和中间件，以驱动软件开发的计算平台。



平台即服务PaaS

- 计算平台服务内容：
  - 使用PaaS所支持的编程语言或工具作应用程序开发。
  - 并向云平台部署所开发的应用程序。



核心平台

- PaaS服务提供商为开发人员提供程序运行时环境。
- 连接客户与硬件，实现对云平台底层资源的自动控制管理，使得开发人员可以专注于服务本身：
  - 动态资源供给
    - 根据应用程序的实际需求动态的资源供给。
  - 负载均衡
    - 均衡的将工作负载分发到节点。
  - 容错能力
    - 即使发生硬件故障，也保证服务不受影响。
  - 系统状态监控
    - 监控系统运行状态，测算资源消耗量。

服务支持

- 提供平台接口和服务，以驱动应用程序开发：
- 客户可利用以下功能开发自己的应用程序：
  - 集成开发环境；
  - 系统控制接口；
  - 编译测试环境。
  - 等等。

PaaS

- PaaS 服务提供商

Windows Azure

Google App Engine

force.com platform as a service

IBM Pangoo

AppScale

hadoop

- Hadoop技术
  - 分布式文件系统— GFS vs HDFS
  - 分布式数据处理 — Mapreduce

文件系统

- 文件系统
  - 用于持久的存储数据的系统。
  - FAT， FAT32, NTFS, EXT等。
    - 通常覆盖在底层的物理介质上。
      - 硬盘，CD，优盘等。
- 数据组织的基本单元：文件
  - 具有文件名（1.txt）
  - 通常支持层次化嵌套（目录结构）。

文件系统

- 文件路径
  - 文件与目录的结合，用于定位文件。
  - 绝对路径，/home/aaron/foo.txt
  - 相对路径，docs/someFile.doc
  - 所有规范路径的集合构成了文件系统的目录结构。

文件系统

- 文件系统的存储内容：
  - 主要内容：用户的实际数据。
  - 元数据：驱动器元数据和文件元数据。

驱动器元数据	文件元数据
• 可用空间 • 格式信息 • 字符集 • ……	• 文件名 • 创建者 • 修改者 • 创建时间 • 修改时间 • ……

文件系统

- 文件的分块存储：

A	B	C	(free space)

A	B	C	A	(free space)

A	(free space)	C	A	(free space)

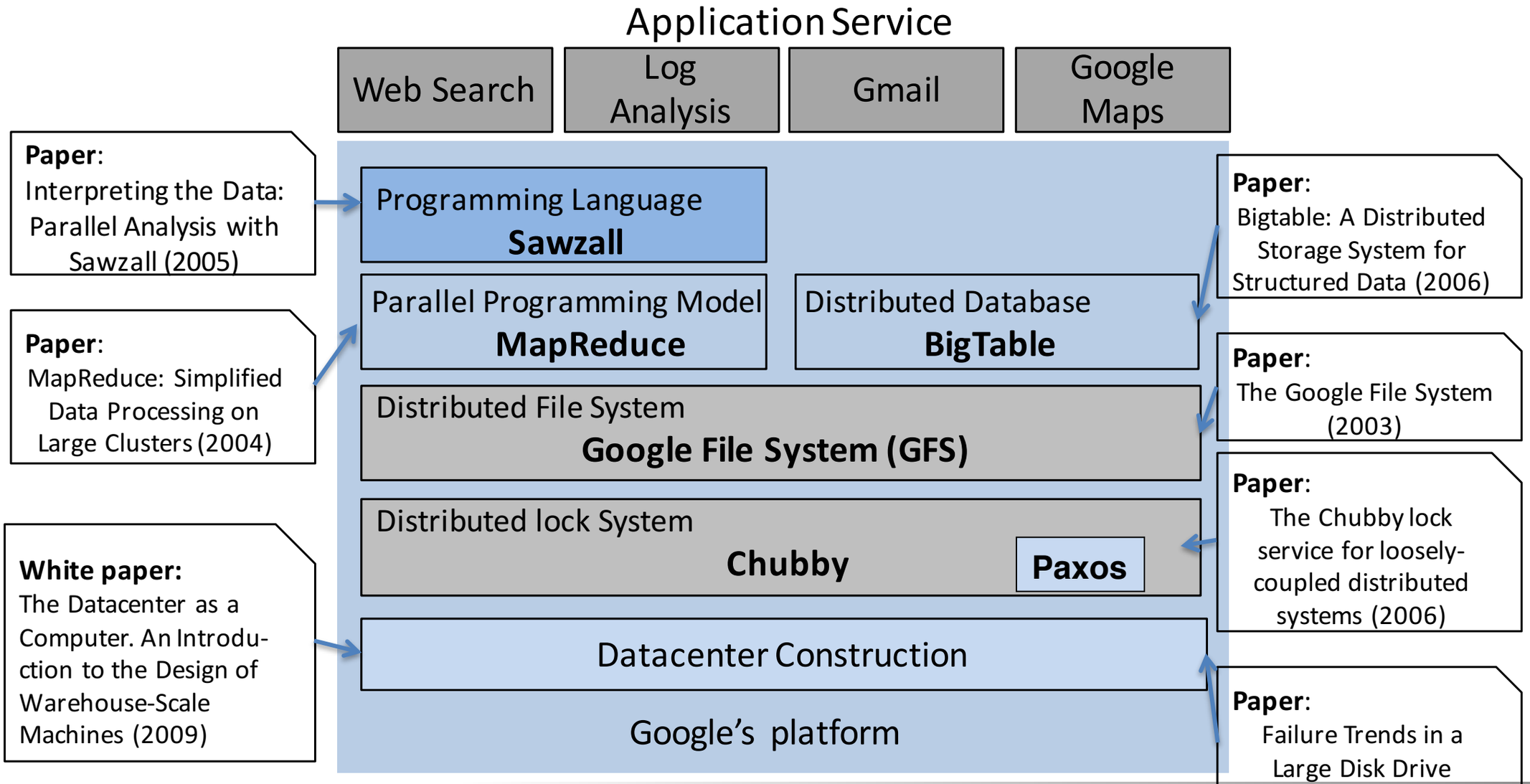
A	D	C	A	D	(free)

文件系统

- 文件系统设计的考虑因素
  - 最小存储单元
    - 较小可减少浪费空间，较大则可提高文件顺序读取速度。
  - 文件系统的设计目标是提高访问速度还是提高使用率?
- 文件系统的安全性
  - 多用户环境下的文件安全
  - 读/写权限分配
  - 文件附带访问控制列表（ACL）
- 文件系统缓存
  - 提高文件系统读写效率。

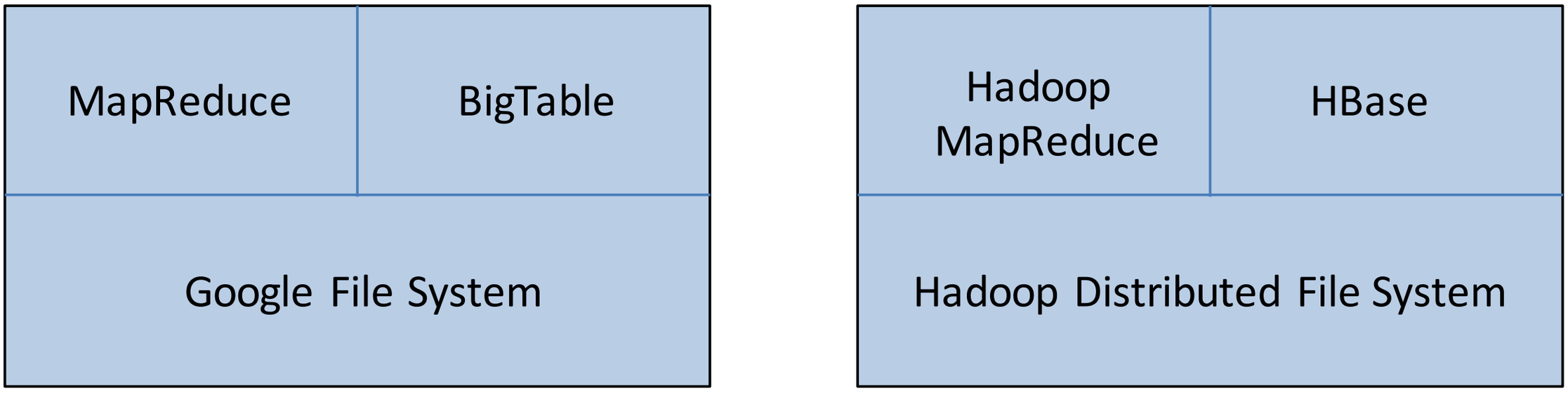
谷歌云计算技术

- GFS (Google File System)：
  - 大规模分布式文件系统



谷歌云计算技术

- Google vs 开源社区



设计动机

- Google需要一个支持海量存储的文件系统
  - 购置昂贵的分布式文件系统与硬件?

是否可以在一堆廉价且不可靠的硬件上
构建可靠的分布式文件系统?

设计动机

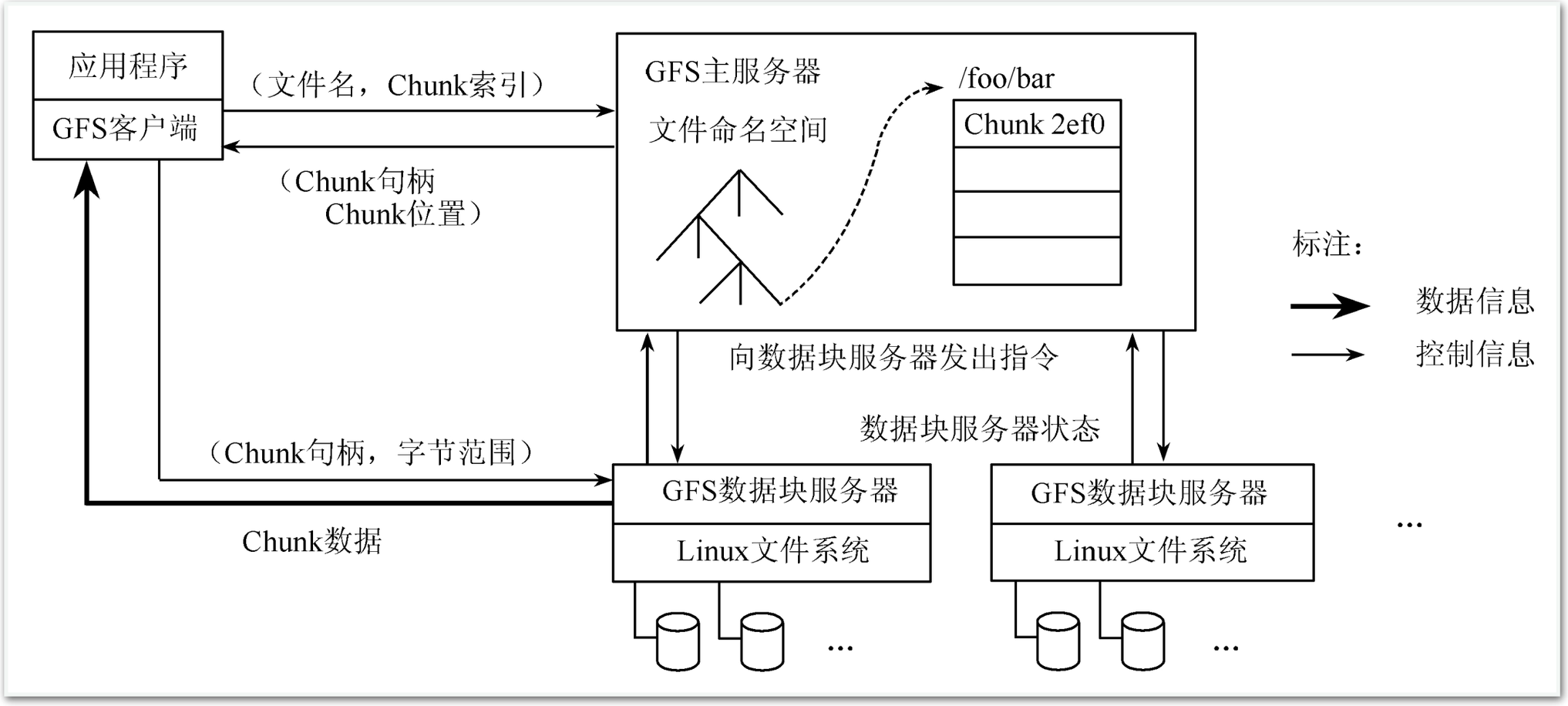
- 为什么不使用当时现存的文件系统?
- 当时的主流分布式文件系统
  - RedHat的GFS（Global File System）
  - IBM的GPFS
  - Sun的Lustre
- Google所面临的问题与众不同
  - 不同的工作负载，不同的设计优先级（廉价、不可靠的硬件）
- 需要设计与Google应用和负载相符的文件系统。

设计动机

- GFS的假设与目标：
  - 硬件出错是正常而非异常
    - 系统应当由大量廉价、易损的硬件组成。
    - 必须保持文件系统整体的可靠性。
    - 主要负载是流数据读写
  - 主要用于程序处理批量数据，而非与用户的交互或随机读写。
    - 数据写主要是“追加写”，“插入写”非常少。
  - 需要存储大尺寸的文件
    - 存储的文件尺寸可能是GB或TB量级，而且应当能支持存储成千上万的大尺寸文件。

谷歌文件系统GFS

- GFS设计



- Client（客户端）：应用程序访问接口，与主服务器和数据块服务器通信。
- Master（主服务器）：管理节点，控制中枢，存储元数据。
- Chunk Server（数据块服务器）：以普通linux文件的形式存储数据。

谷歌文件系统GFS

- 实现机制：
  - 客户端首先访问Master节点，获取交互的Chunk Server信息，然后访问这些Chunk Server，完成数据存取工作。这种设计方法实现了控制流和数据流的分离。
  - Client与Master之间只有控制流，而无数据流，极大地降低了Master的负载。
  - Client与Chunk Server之间直接传输数据流，同时由于文件被分成多个Chunk进行分布式存储，Client可以同时访问多个Chunk Server，从而使得整个系统的I/O高度并行，系统整体性能得到提高。

谷歌文件系统GFS

- 最小化主服务器的参与度：
  - 数据传输和控制流分离，提高网络利用效率。
- 数据块大小：64MB
  - 大数据块 —> 减少寻址开销。
  - 大数据处理过程中，程序通常需要处理大块数据，使用大数据块可保持较长时间的TCP连接通信，减少网络开销。
  - 减少主服务上的元数据的存储量。

谷歌文件系统GFS

- 设计特点：
  - 采用中心服务器模式
  - 数据副本提高可靠性（默认为3副本）
  - 在用户态下实现
  - 只提供专用接口

这个名字是我的孩子给一个棕黄色的大象玩具命名的。

Doug Cutting

- Hadoop Distributed File System - HDFS

HDFS

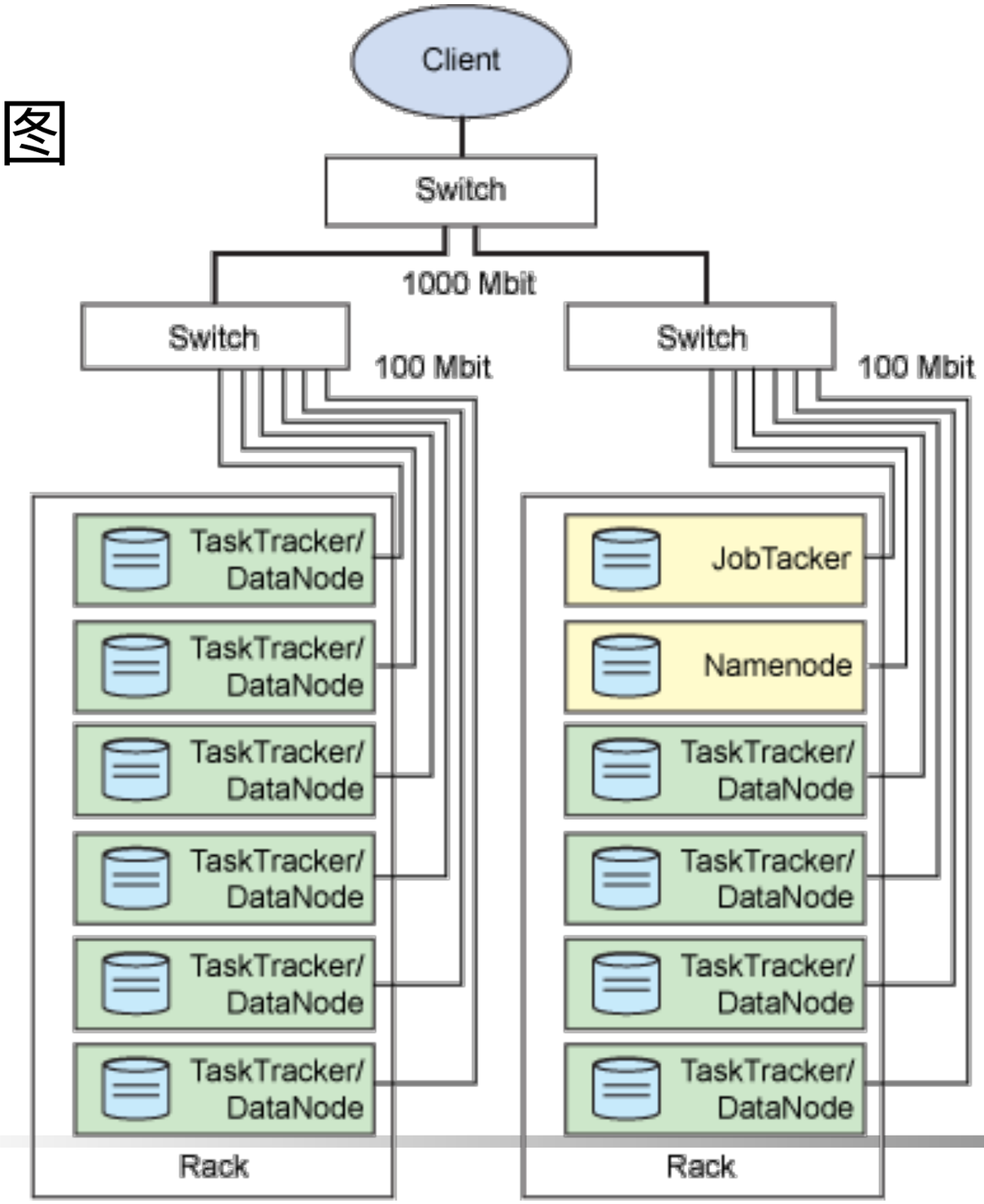
- Apache Hadoop:
  - 一个分布式计算平台；
  - 可用于开发和运行大数据处理应用程序的软件框架。
- 起源于Google发表的论文。
  - GFS （SOSP2003）
  - Mapreduce（OSDI2004）
  - Bigtable （OSDI2006）
- HDFS (Hadoop Distributed File System):
  - Hadoop项目下的分布式文件系统，支持海量数据的高速访问。
  - 使用 Java 编写。

HDFS

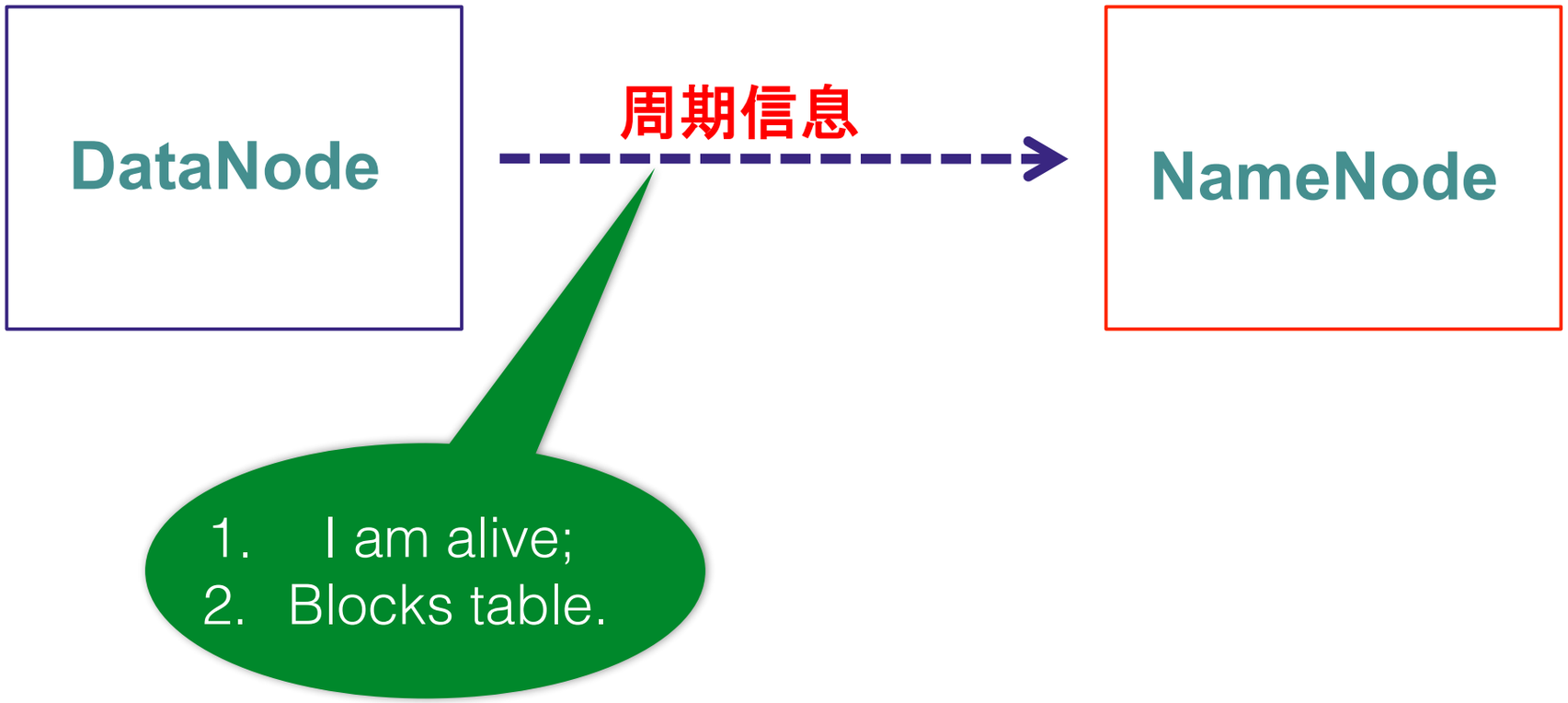
- 命名方式：
  - Namenode
    - 相当于GFS的主服务器。
  - Secondary Namenode
    - Namenode的远程实时备份。
  - Datanode
    - 相当于GFS的数据块服务器。

Hadoop集群

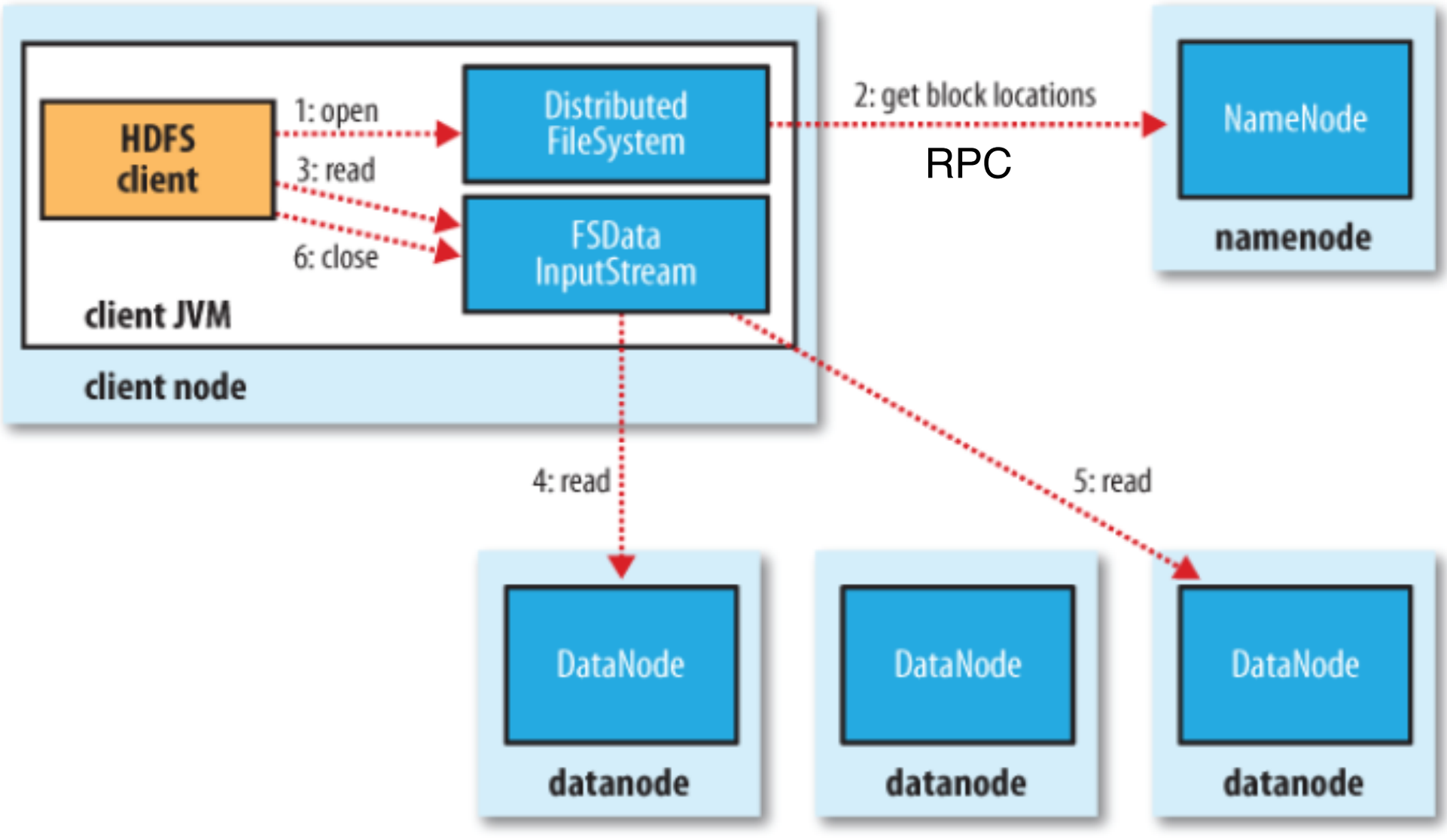
- Hadoop集群示意图



心跳机制

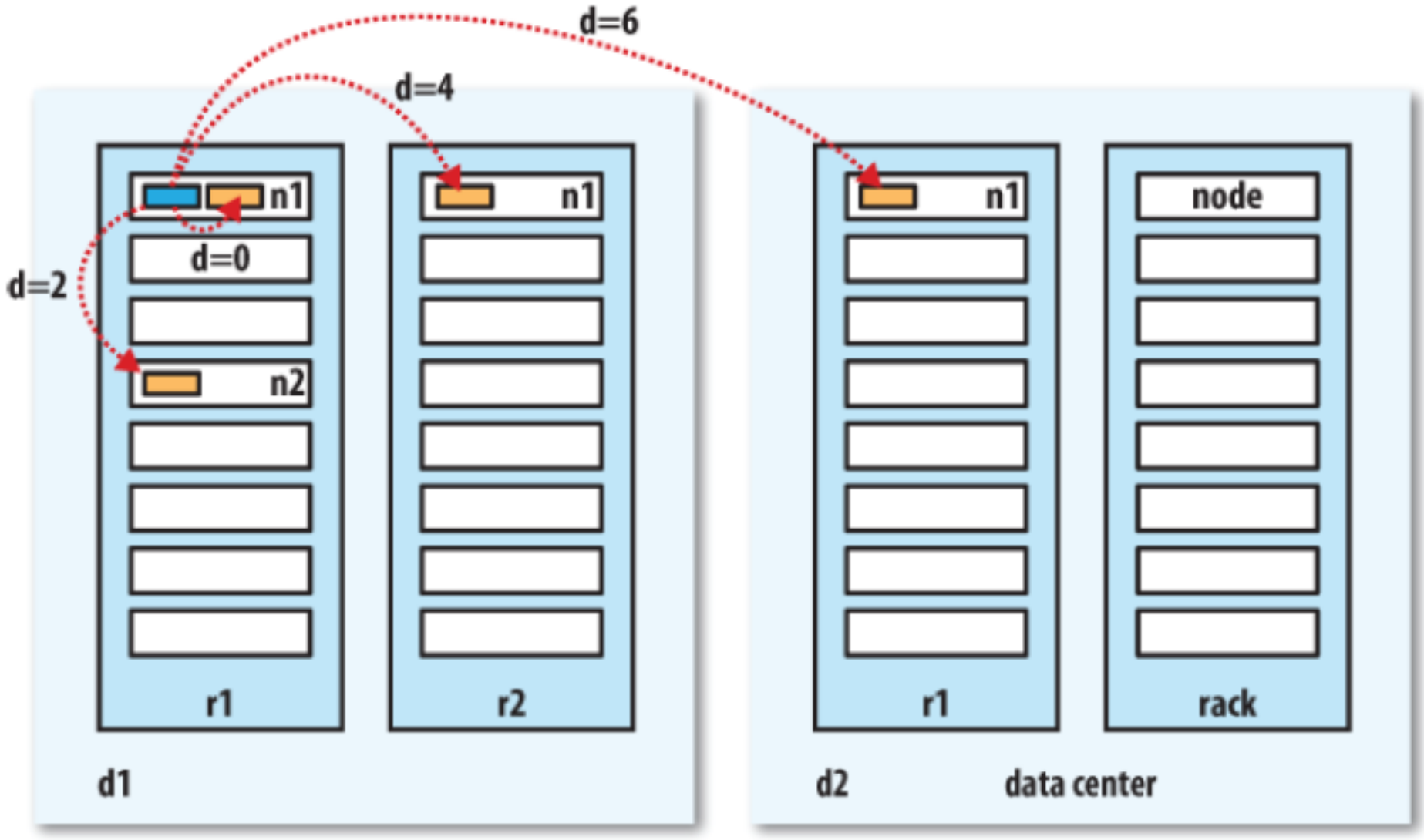


HDFS 读数据

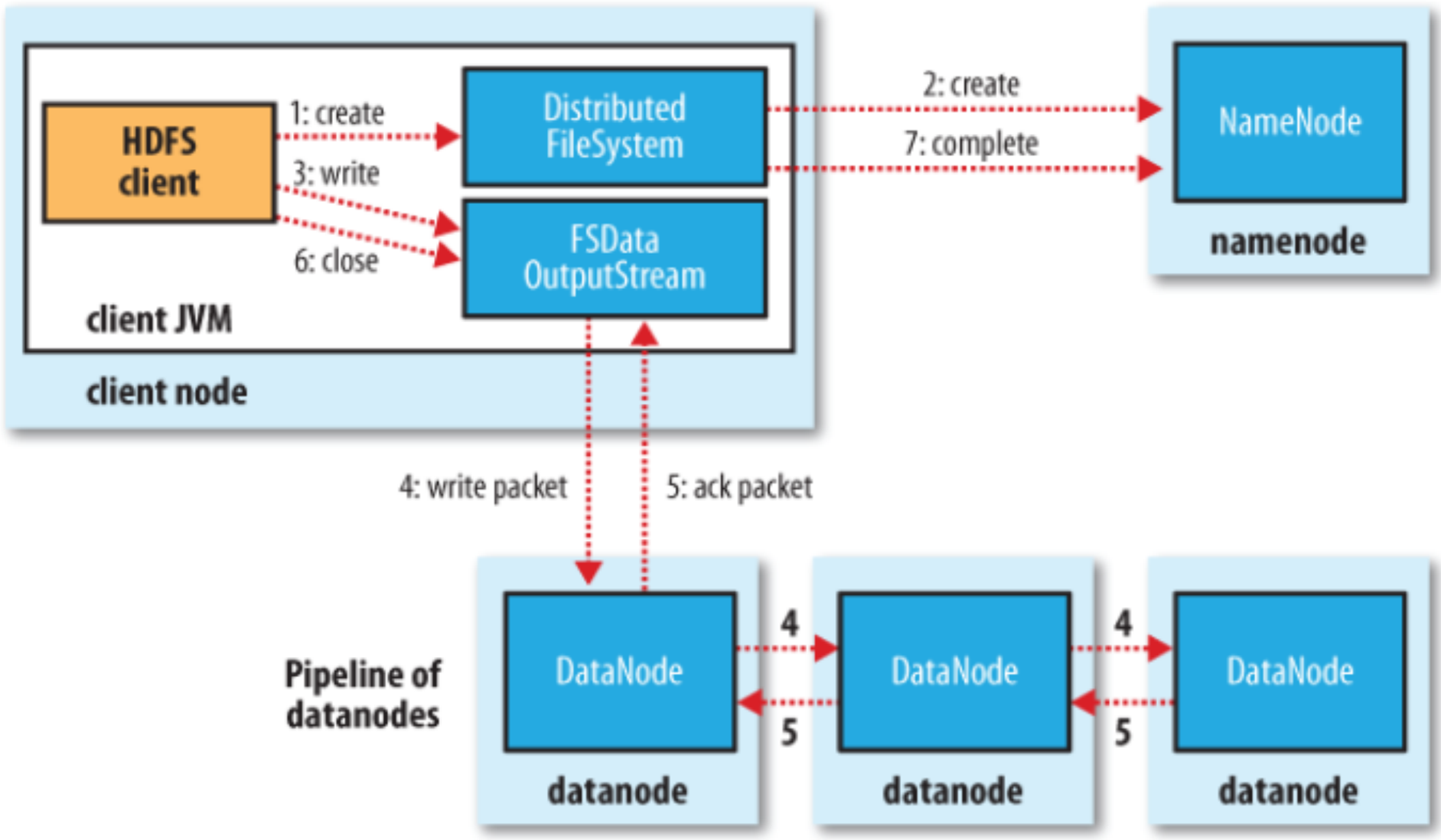

HDFS client
1: open
3: read
6: close
Distributed FileSystem
FSData InputStream
client JVM
client node
2: get block locations
RPC
NameNode
namenode
4: read
5: read
DataNode
datanode
DataNode
datanode
DataNode
datanode

HDFS读数据

- 网络中的距离：

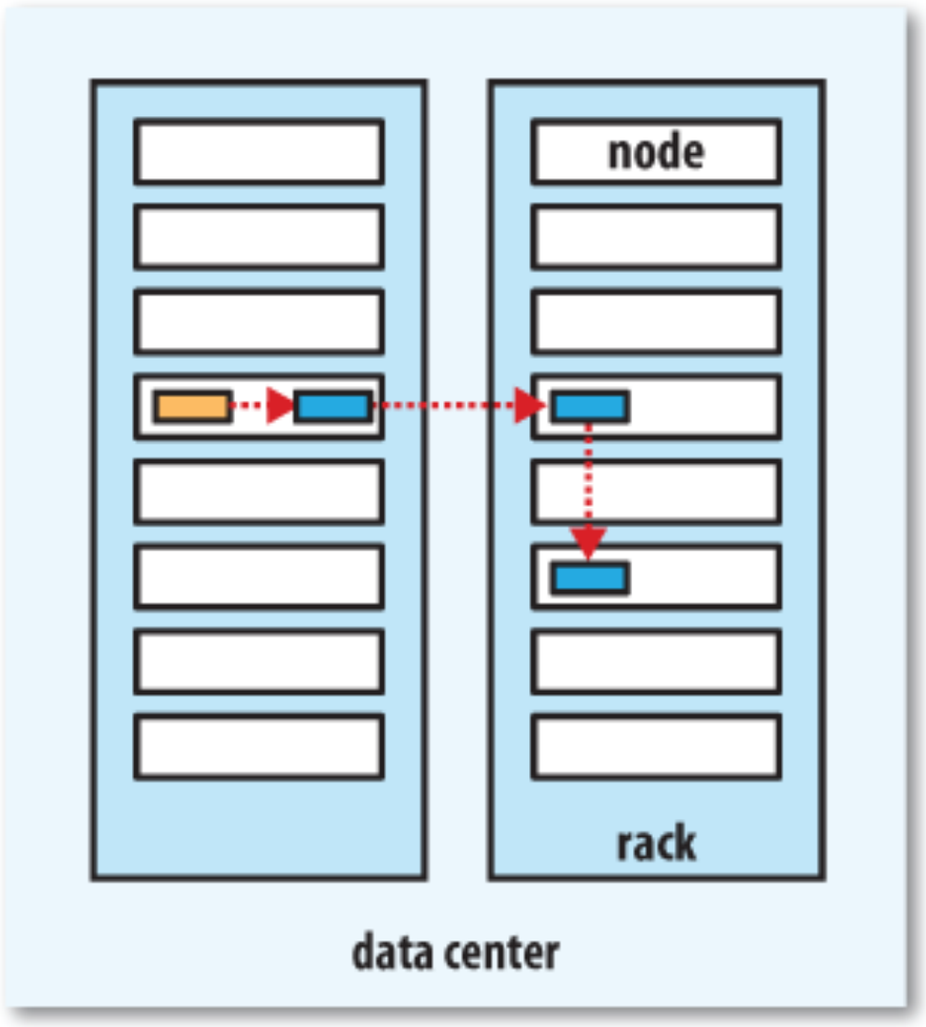


HDFS 写数据

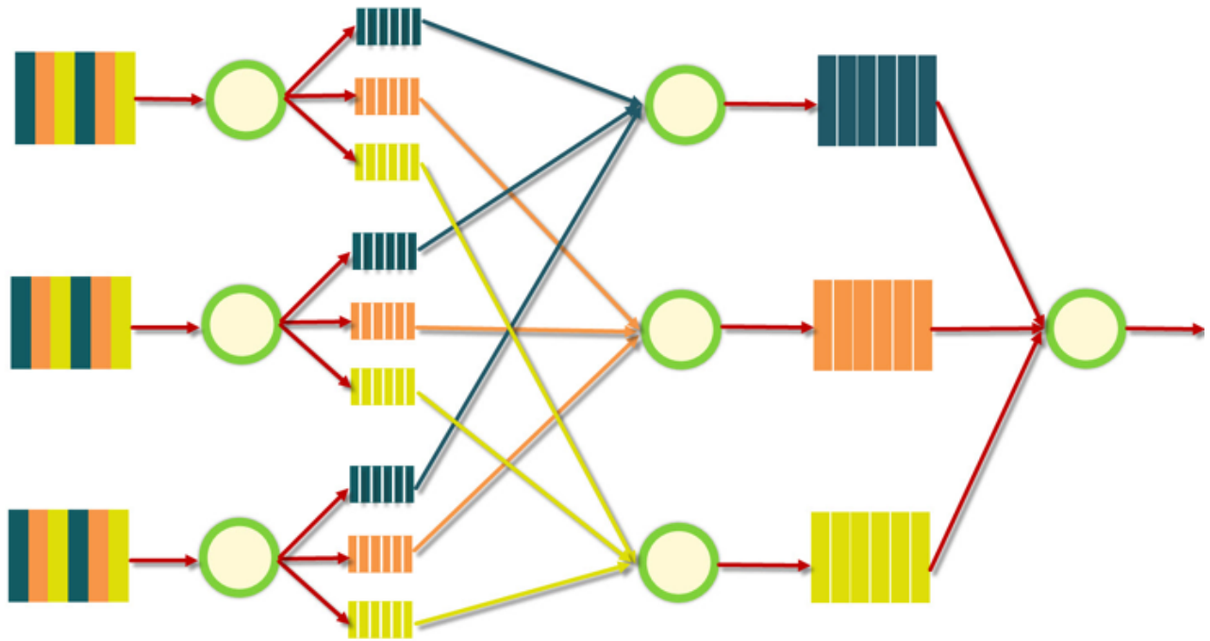

HDFS client
1: create
3: write
6: close
Distributed FileSystem
FSData OutputStream
client JVM
client node
2: create
7: complete
NameNode
namenode
4: write packet
5: ack packet
Pipeline of datanodes
DataNode
4
5
datanode
DataNode
4
5
datanode
DataNode
datanode

HDFS写数据

- HDFS下的副本安置：



- 考虑可靠性和读写数据带宽之间的平衡：
- 安置策略：
  - 第一个副本与客户所在放置同一节点。
    （客户端不在系统中时，随机选择）
  - 第二个副本放在与第一个副本不同的另一机架的随机节点。
  - 第三副本放置与第二副本同一机架，但是不同节点上。



- Mapreduce数据处理

Jeff Dean

谷歌服务

网络

网页搜索
在数十亿网页中搜索所需内容

Google Chrome
高速、简约、安全的浏览器

工具栏
将搜索框添加到浏览器中

书签
访问书签和加星标的项

移动

移动
在手机上使用Google产品

Google地图（移动版）
可在手机上查看地图、您的位置，并获取路线指引

移动版搜索
随时随地在Google上搜索

企业

AdWords
可吸引到更多客户，却只需按成效付费

Google Apps for Work
为您的企业提供电子邮件、文档、存储空间等各种定制产品和服务

Google 我的商家
确保在 Google 搜索、地图和 Google+ 上免费充分展示你的商家。

AdSense
马上在线创收

AdMob
通过您的应用获利

媒体

YouTube
观看、上传和分享视频

Google图书
搜索图书全文

图片搜索
在网络上搜索图片

新闻
搜索数以千计的新闻报道

视频搜索
在网络上搜索视频

Picasa
查找、编辑和分享照片

Motivation

- 每次搜索：
  - 200+ CPU
  - 200TB以上数据量
  - 0.1 秒内响应时间
  - 5¢ 收益

设计动机

- 网页数据集大小：
  - 上百TB；
  - 无法使用单机处理如此多数据。
- 数据处理需求：
  - 计数：Word Count
  - Google趋势
  - PageRank算法
  - …

设计动机

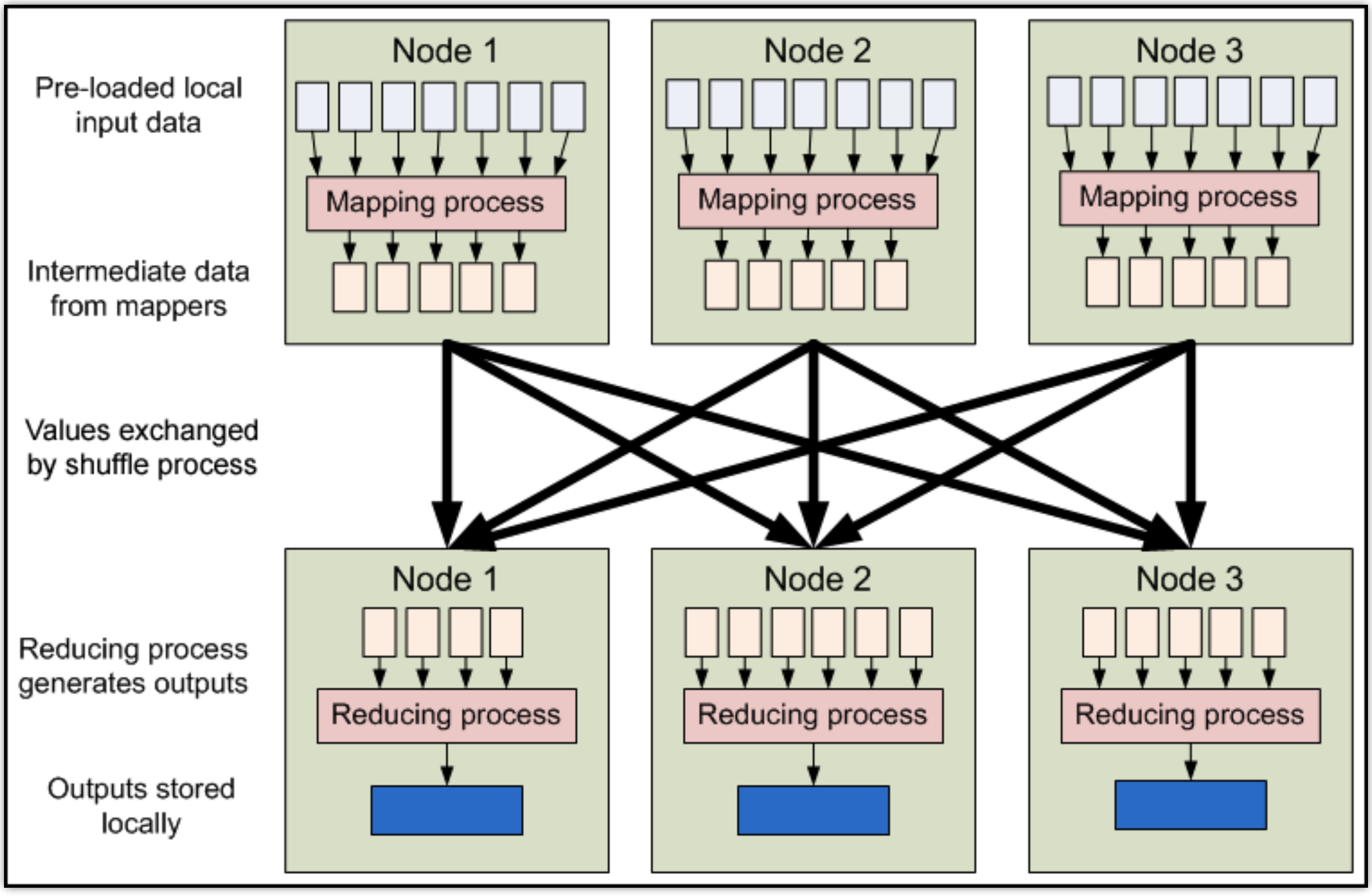
- 问题简答，解决困难:
  - 如何在有限时间内解决问题?

使用分布式系统，对大数据量分而治之!

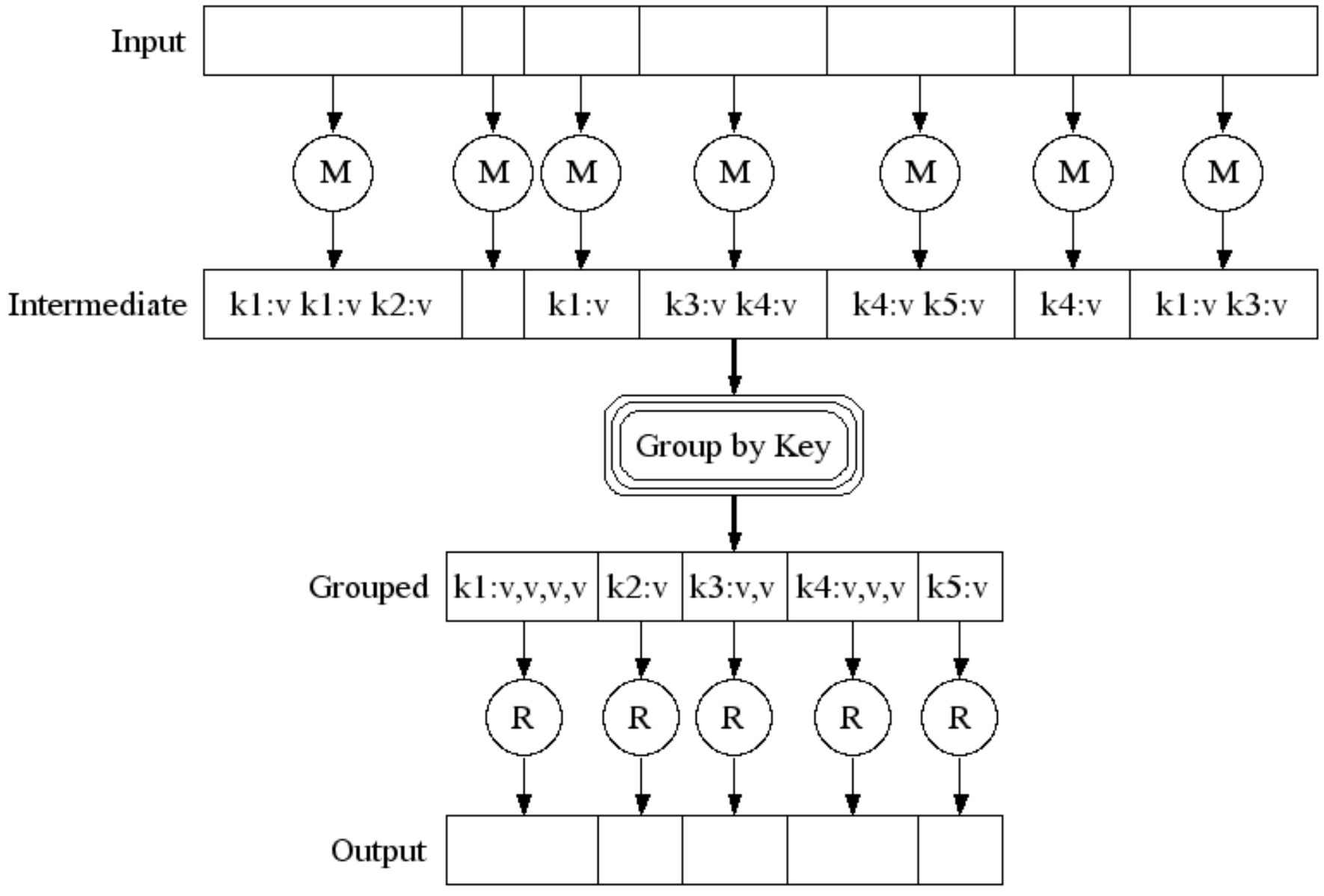
Mapreduce

- Mapreduce：
  - 一个软件架构，是一种处理海量数据的并行编程模式。
  - 用于大规模数据集（通常大于1TB）的并行运算。
- MapReduce的两个功能：
  - Map函数：把一个函数应用于集合中的所有成员，然后返回一个基于这个处理的结果集。
  - Reduce函数：对结果集进行分类和归纳。
  - Map()和 Reduce() 两个函数可能会并行运行，即使不是在同一的系统的同一时刻。

Mapreduce


Pre-loaded local input data
Intermediate data from mappers
Values exchanged by shuffle process
Reducing process generates outputs
Outputs stored locally
Node 1
Node 2
Node 3
Mapping process
Mapping process
Mapping process
Node 1
Node 2
Node 3
Reducing process
Reducing process
Reducing process

Mapreduce


Input
M
M
M
M
M
M
M
Intermediate
k1:v k1:v k2:v
k1:v
k3:v k4:v
k4:v k5:v
k4:v
k1:v k3:v
Group by Key
Grouped
k1:v,v,v,v
k2:v
k3:v,v
k4:v,v,v
k5:v
R
R
R
R
R
Output

Example: WordCount

- Input:

Page 1:
the weather is good
Page 2:
today is good
Page 3:
good weather is good

Example: WordCount

- map output：
  - Worker 1:
    - (the 1), (weather 1), (is 1), (good 1).
  - Worker 2:
    - (today 1), (is 1), (good 1).
  - Worker 3:
    - (good 1), (weather 1), (is 1), (good 1).

Page 1:
the weather is good
Page 2:
today is good
Page 3:
good weather is good

Example: WordCount

- Input of Reduce:
  - Worker 1:
    - (the 1)
  - Worker 2:
    - (is 1), (is 1), (is 1)
  - Worker 3:
    - (weather 1), (weather 1)
  - Worker 4:
    - (today 1)
  - Worker 5:
    - (good 1), (good 1), (good 1), (good 1)

Example: WordCount

- Reduce output：
  - Worker 1:
    - (the 1)
  - Worker 2:
    - (is 3)
  - Worker 3:
    - (weather 2)
  - Worker 4:
    - (today 1)
  - Worker 5:
    - (good 4)

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

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class MapperClass extends Mapper<Object,Text, Text, IntWritable>{//四个参数的含义，前两个输入，后两个输出
，类型需一致
    public Text keytext =new Text("text");//定义一个text对象，用来充当中间变量，存储词
    public IntWritable intvalue=new IntWritable(1);//词的个数，刚开始都为1，也可以不定义，直接context.write(keytext, 1);
    @Override
    protected void map(Object key, Text value,
            Context context)//key即行偏移量，作用不大，主要是value，根据value进行拆分
        throws IOException, InterruptedException {
      //获取值
      String str =value.toString();
      //分隔
      StringTokenizer stringTokenizer=new StringTokenizer(str);//StringTokenizer根据空格等分隔字符串到stringTokenizer
      while (stringTokenizer.hasMoreElements()) {//返回是否还有分隔符，判断是否还有单词
          keytext.set(stringTokenizer.nextToken());//nextToken(): 返回从当前位置到下一个分隔符的字符串。
          context.write(keytext, intvalue);//context.write("hello",1)

      }

    }
}
```

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

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class ReducerClass extends Reducer<Text, IntWritable, Text, IntWritable> {// 前两个输入：例：（hello，1），后两个输出（hello，2）

    public IntWritable intValue = new IntWritable(0);

    @Override
    protected void reduce(Text key, Iterable<IntWritable> values,// 这里声明了一个实现Iterator接口的匿名内部类，并返回了内部类的实例
        Context context)// 它用来与MapReduce系统进行通信，如把map的结果传给reduce处理
        throws IOException, InterruptedException {
    // step1
    int sum = 0;
    Iterator<IntWritable> itr = values.iterator();// 迭代器，访问容器中的元素，为容器而生
    while (itr.hasNext()) {

        sum += itr.next().get();// 如果有，则加入迭代器中的个数

    }
    intValue.set(sum);// 对于hello，sum是2
    context.write(key, intValue);// hello，2
  }
}
```

```java
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;

public class WordCount {
  public static void main(String[] args) throws Exception{
    Configuration conf =new Configuration();//指定作业执行规范
    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");//指定job名称，及运行对象
    job.setJarByClass(WordCount.class);
    job.setMapperClass(MapperClass.class);//指定map函数
    job.setCombinerClass(ReducerClass.class);//是否需要conbiner整合
    job.setReducerClass(ReducerClass.class);//指定reduce函数
    job.setOutputKeyClass(Text.class);//输出key格式
    job.setOutputValueClass(IntWritable.class);//输出value格式
    FileInputFormat.addInputPath(job, new Path(otherArgs[0]));//处理文件路径
    FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));//结果输出路径
    System.exit(job.waitForCompletion(true)?0:1);
  }
}
```

Hadoop Mapreduce编程

- Class Mapper<KEYIN,VALUEIN,KEYOUT,VALUEOUT>
  - 将输入“键值对”作Map处理，输出中间数据“键值对”。
  - map() 函数：
    - 输入数据分片，每片数据含多个键值对，被一个Map处理。
    - 一般情况下，需要重写map函数。

```
protected void map(KEYIN key,
        VALUEIN value,
        org.apache.hadoop.mapreduce.Mapper.Context context)
             throws IOException,
                    InterruptedException
```

Hadoop Mapreduce编程

- Context 对象:
  - 允许Map或Reduce类与Hadoop系统交互，获得任务处理的配置参数，以及处理结果的存储位置等信息。
- 应用程序可使用Context类：
  - 报告进展速度；
  - 设置应用层状态消息；
  - 更新计数器；
  - 读取任务配置参数。

Hadoop Mapreduce编程

- Class Reducer<KEYIN,VALUEIN,KEYOUT,VALUEOUT>
  - 对Map输出的中间结果作Reduce函数处理，将key值相同的中间数据结合到一起。
  - Reduce的执行包括3阶段：
    - Shuffle阶段:
      - 将Map的输出拷贝到reduce所在节点。
    - Sort:
      - 对中间数据按照key值作排序。
    - Reduce
      - 调用reduce()函数，处理结果。

Hadoop Mapreduce编程

```
protected void reduce(KEYIN key,
          Iterable<VALUEIN> values,
          org.apache.hadoop.mapreduce.Reducer.Context context)
               throws IOException,
                      InterruptedException
```

- reduce() 函数:
  - 每个键对应的数据，只允许被一个reduce处理。
  - 一般情况下，需要重写reduce函数。

总结

- PaaS
- Hadoop
  - 分布式文件系统——GFS
  - 分布式数据处理——Mapreduce