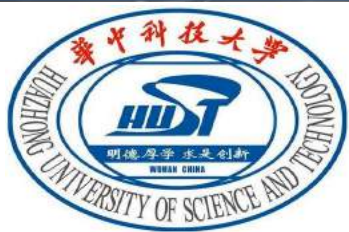


第五章 大数据处理



肖江

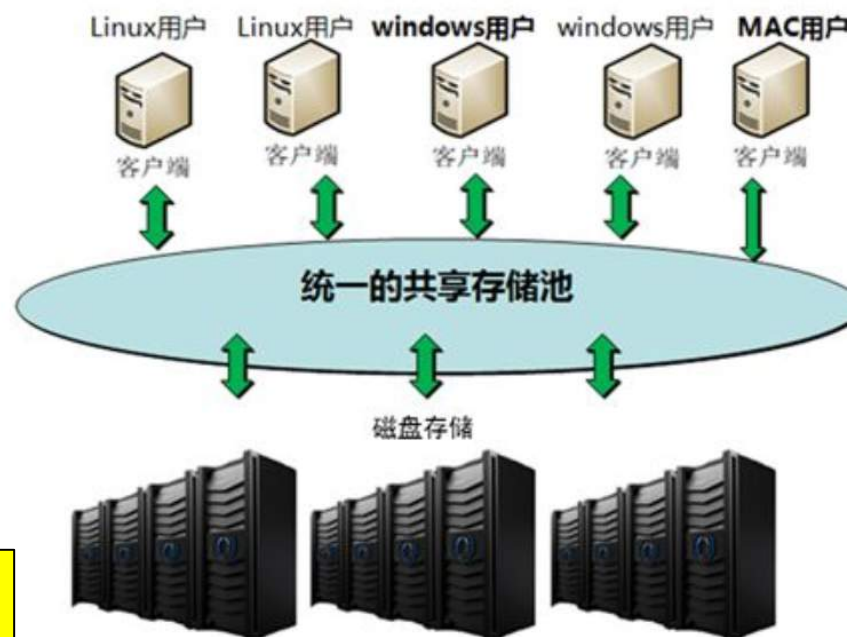
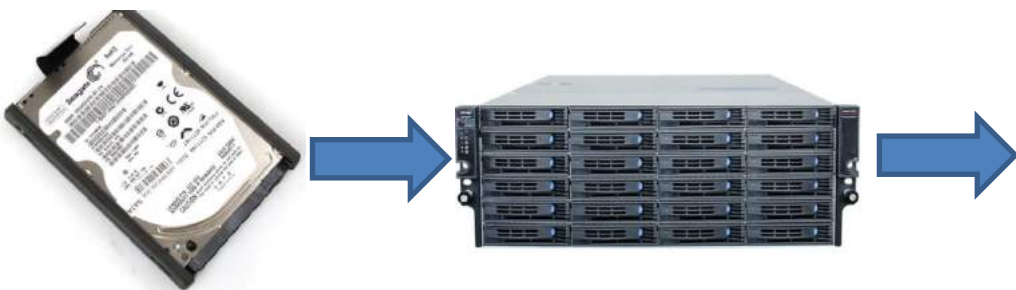
Mail : jiangxiao@hust.edu.cn

Office: 东五楼 222 室



内容回顾：大数据的存储与管理

- 分布式存储是**必然选择**



弹性容量扩张，高度并行访问





内容回顾：大数据的存储与管理

- 分布式存储不仅仅是磁盘阵列的互联
- 分布式存储的主要目标：基于大规模的普通服务器构建**可扩展、低成本、高性能、高可用**的分布式存储系统



内容回顾：大数据的存储与管理

• 分布式文件系统的关键技术问题

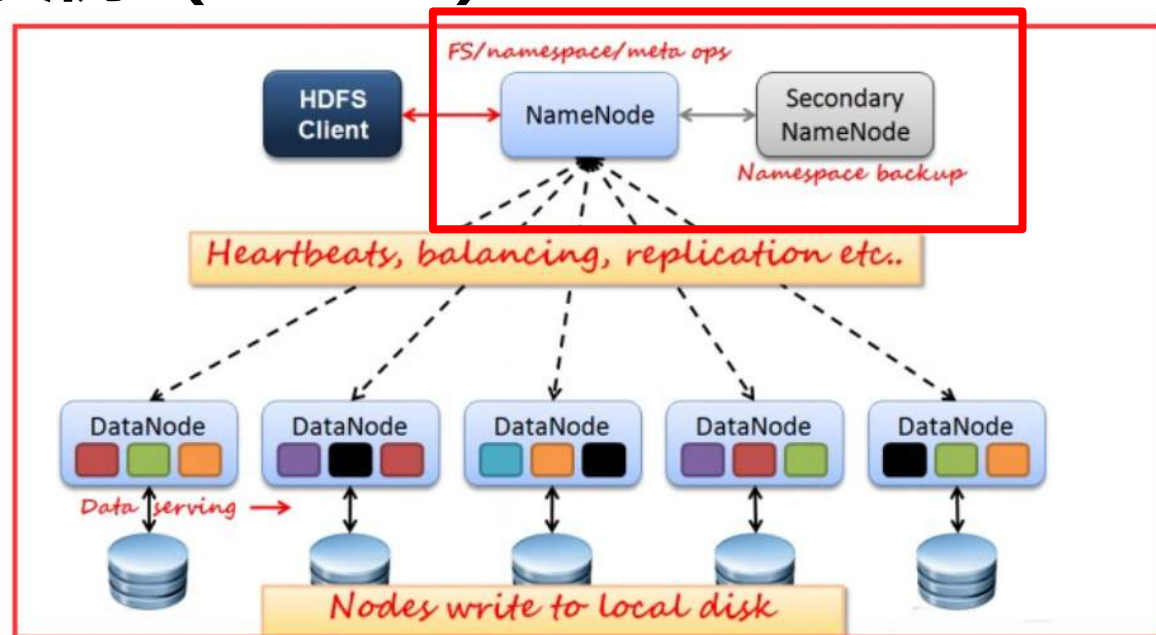
- 可扩展性  存储规模和需求，弹性收缩
- 数据分布  不同的存储节点上以实现负载均衡
- 复制和一致性  保证分布式存储系统的一致性
- 容错机制  提高存储系统的可靠性和可用性

内容回顾：大数据的存储与管理

• 分布式文件系统实例（HDFS）

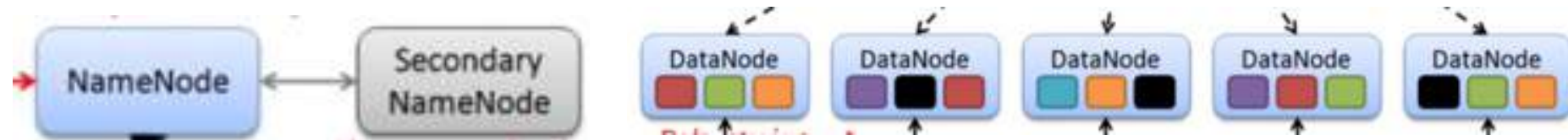
• 主从结构

元数据节点为主节点
数据节点为奴隶节点，
主节点控制着所有的
奴隶节点



内容回顾：大数据的存储与管理

• HDFS主要组件功能



- 存储元数据
- 管理数据块映射
- 处理客户端的读写请求
- 管理HDFS的命名空间

NameNode 可实现弹性的
容量扩展

- 存储数据块
- 执行数据块的读写操作
- 维护block 与DataNode
本地文件的映射关系

内容回顾：大数据的存储与管理

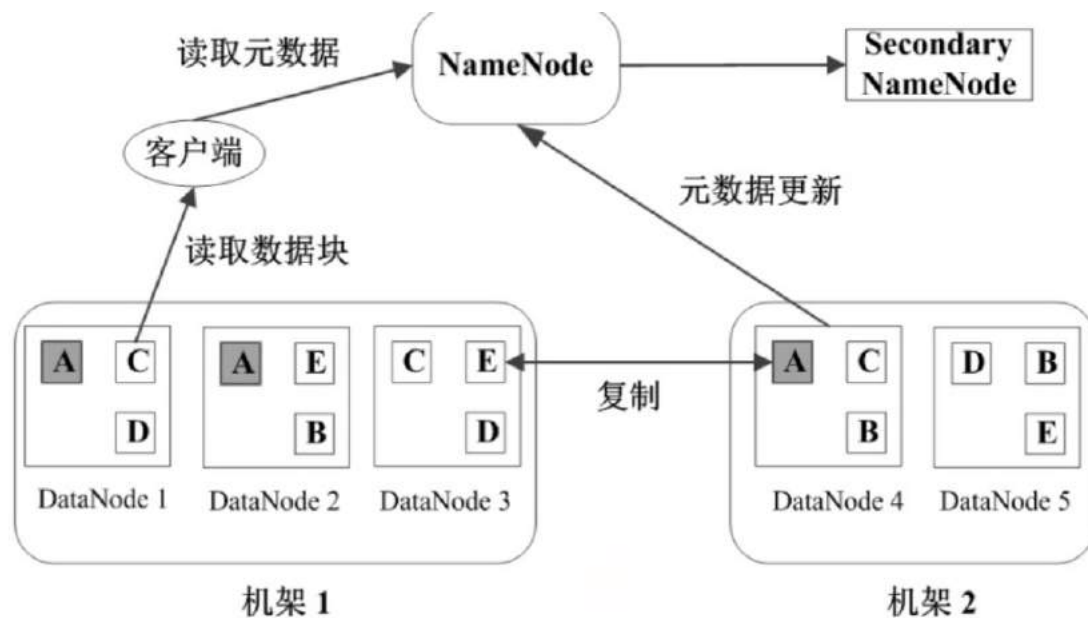
- 分布式文件系统实例（HDFS）

- HDFS中的容错

通过多副本

保证数据的

容错和高可用

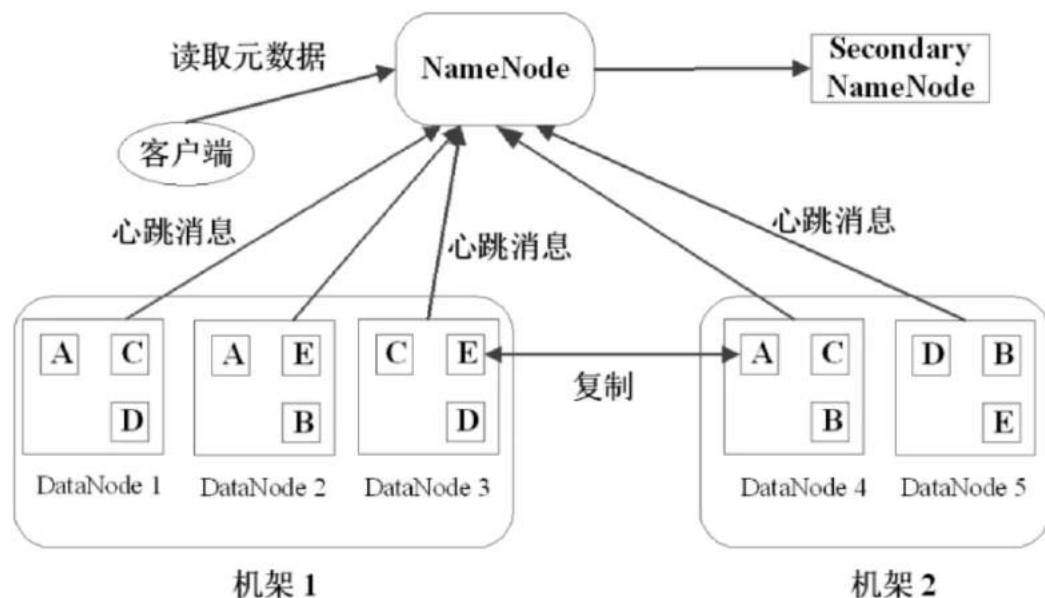


每个数据块3个副本，分布在两个机架内的三个节点

内容回顾：大数据的存储与管理

- 分布式文件系统实例（HDFS）
- 心跳机制

通过定时向主节点发送节点状态，让主节点作出反应，提供可用性。



DataNode定期向NameNode发送心跳消息

内容回顾：大数据的存储与管理

大数据管理

- 传统的数据管理系统（如关系数据库）

- 事务管理制约了数据库性能
- 互联网应用一般不涉及多个数据表
- 关系数据库要求规范的数据格式，而互联网数据多样且稀疏



内容回顾：大数据的存储与管理

大数据管理

- **NoSQL** 数据库通过**不同程度去掉**关系数据库的**某些特性**，来应对大数据的某种管理挑战

Not Only SQL



内容回顾：大数据的存储与管理

大数据管理

• NoSQL的四大分类

- 键值对存储系统
- 文档数据库
- 列族存储系统
- 图数据库



目录

5.1 引言

5.2 集中式计算架构

5.3 分布式计算架构

5.4 处理加速技术 (GPU/TPU/FPGA)

5.5 本章小结

5.1 引言

为什么需要大数据处理生态系统？

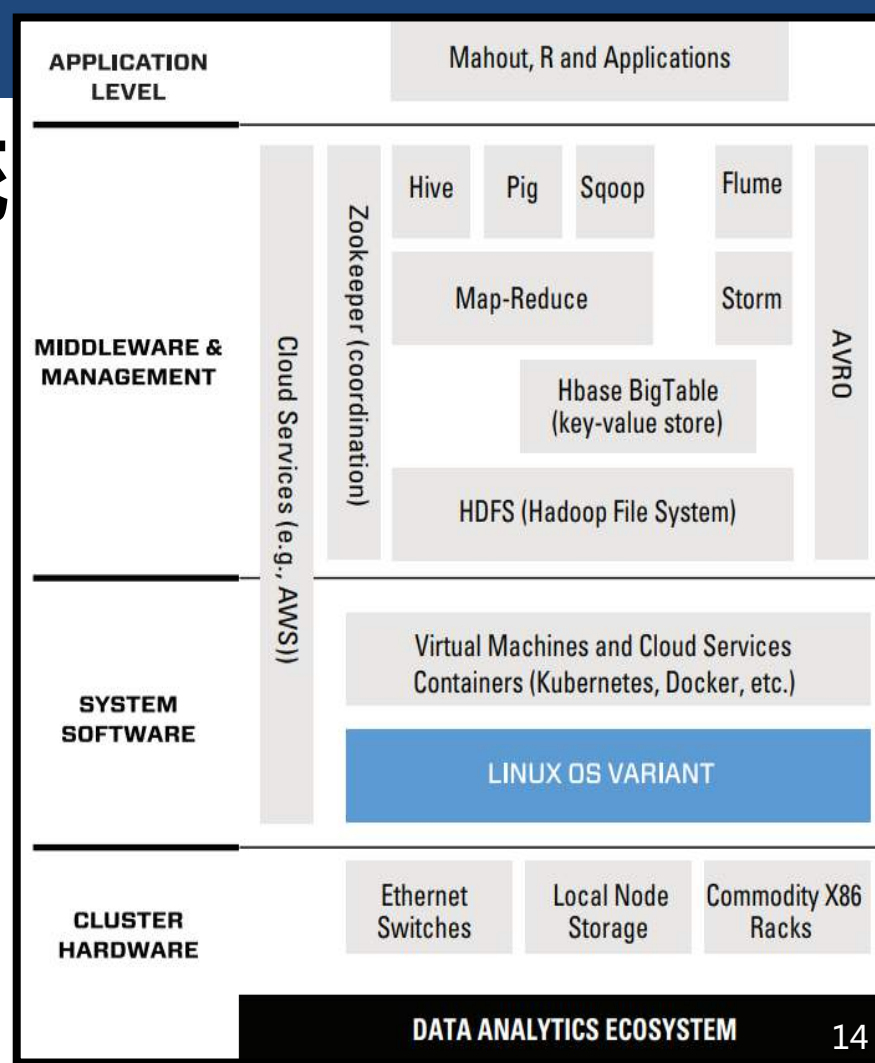
- 大数据自身特征使得传统数据架构**无法有效处理**
- 大数据处理涉及**数据层、算法层、计算层、应用开发层**等多个方面

5.1 大数据处理生态系统

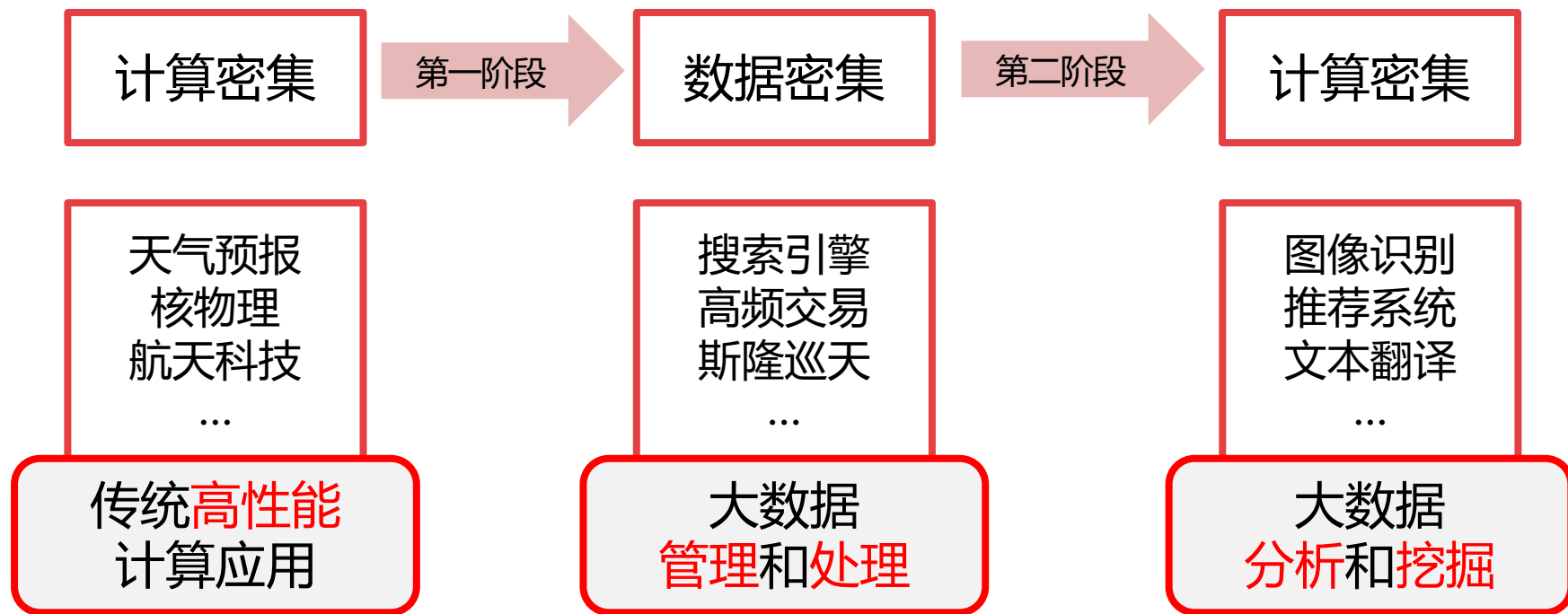
5.1.2 大数据处理

生态系统架构图

- 为支持大数据应用的多样性，大数据处理系统采用**层次化、模块化的设计思想**，实现独立的数据存储、管理和处理，所有的模块共同构成大数据处理生态系统。



计算衍化



目录

5.1 引言

5.2 集中式计算架构

5.3 分布式计算架构

5.4 处理加速技术 (GPU/TPU/FPGA)

5.5 本章小结

5.2 集中式计算架构

存储需求与计算需求不断提升

- 单机无法容纳、处理如今的数据

数据总量	• 100~1000PB
数据处理量	• 10~100PB/天
网页	• 千亿~万亿
索引	• 百亿~千亿
更新量	• 十亿~百亿/天
请求	• 十亿~百亿/天
日志	• 100TB~1PB/天



以百度最小的日志数据来看。以一般PC 1T 的硬盘容量来计算，那么需要电脑装配 100~1024 个硬盘。

另外以单机硬盘读写速度 100MB/S 来计算，则顺序读完100T 日志数据至少需要 12.5天

数据密集型应用实例

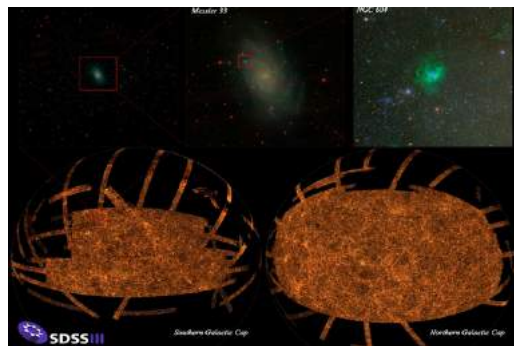
谷歌搜索

- 截至2005年6月，Google已存储超过**80亿**的网页，**1亿3千万**张图片，以及超过**1亿**的新闻组消息
- Google每日通过不同的服务，处理来自世界各地超过**2亿**次的查询
- Google开发了一套名为**GFS**的文件系统，用于存储海量搜索信息，并使用**PageRank算法**配合搜索字符串来对网页进行排名，确保将最重要的搜索结果首先呈现给用户



斯隆巡天

- 斯隆数字化巡天绘制出**人类历史上最大的天体图**，覆盖全天**三分之一**星空，记录**超过一百万**个天体
- 斯隆数字巡天从2000年开启第一阶段，2014年进入第四阶段，目前每晚产生**20TB**原始数据！
- 截止2004年，SDSS数据库的Skyserver每月要处理**超过一千万**查询



5.2 集中式计算架构

5.2.1 大型主机的特点和优势

RAS

- reliability, availability, serviceability

高I/O吞吐量

ISA系统指令架构

5.2 集中式计算架构

5.2.2 大型主机所面临的问题

人才培养
成本高

价格昂贵

扩容困难

市场份额
大幅度减少

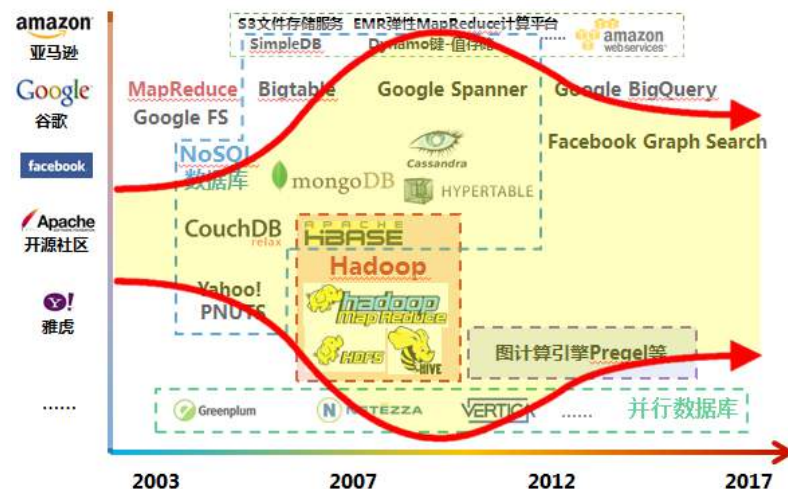
大数据技术的第一阶段：数据密集型

什么是数据密集型计算？

- 处理海量数据
- 关注数据的获取、存储和管理
- 利用廉价存储集群
- 侧重高I/O、高网络带宽和高可扩展性

超算为何在数据密集型应用中被边缘化？

- 没有容错保障
- 存储有限且成本高
- 扩展困难，通常仅由数十至数百个节点组成



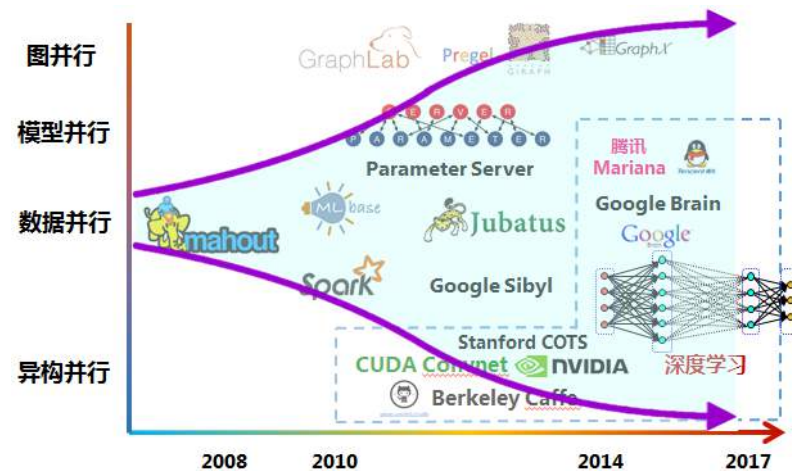
大数据管理与处理技术发展

过去十年，超算多被运用在核物理仿真等**计算密集的应用**上，
使用大规模廉价集群是数据密集应用的主流

大数据技术的第二阶段：计算密集型

新阶段的计算需求：

- 分析和挖掘进行**迭代计算**，重复调用相同数据以拟合**复杂模型**
- 计算量是大数据管理的**几万甚至几百万倍**！
- 传统的廉价集群已经无法满足计算密集型需求



大数据分析、挖掘与大规模机器学习技术

大数据技术的**目标**由存储和处理变为**分析和挖掘**

计算密集型应用实例

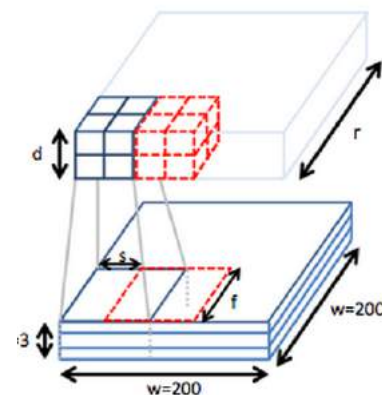
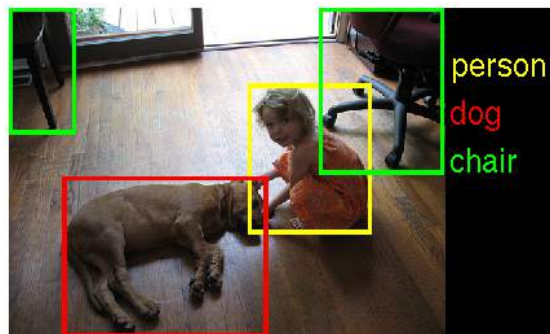
IBM 沃森 (Watson)

- 2011年，沃森参加综艺节目危险边缘**打败所有挑战者**，获得100万美元奖金
- 沃森由**90个节点**组成包含**2880颗CPU**、**16TB内存**和**4TB硬盘**存有**2亿**页结构化和非结构化的信息，包括维基百科。在比赛中沃森没有链接到互联网
- 其80TeraFLOPs的算力在2011年超级电脑世界500强排名第**94**



深度卷积神经网络的训练 (COTS)

- COTS是斯坦福大学推出的高性能深度卷积神经网络系统
- COTS包括**3台高性能服务器**，每台服务器挂载**四块GPU计算卡**
- COTS可以训练含超过**110亿参数**的深度神经网络



5.2 集中式计算架构

5.2.3 超级计算机的发展历史

超级计算机是与通用计算机相比具有极高计算性能的计算机。

性能以每秒浮点运算（ FLOPS ）而不是每秒百万条指令（ MIPS ）来衡量。

现代超算实例-天河2号

世界最快！



分布式中央处理器
+
协处理器异构架构

处理器：
16,000个运算
节点，累计
32,000颗CPU
和48,000个协
处理器，共
312万计算核
心

内存：
每节点
88GB内存，
总计内存
1.34PB

磁盘：
总计
12.4PB
磁盘阵
列

网络：
光电混合
传输技术，
数据发送
速率
6.36GB/s

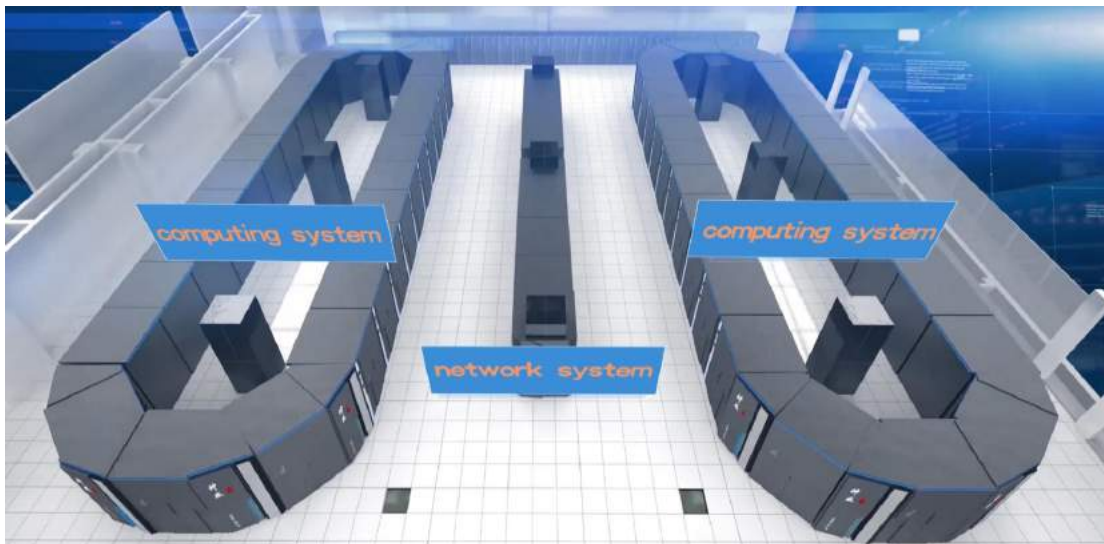
峰值速度 (Rpeak) 每秒54,902.4TFLOPS
(万亿次浮点运算)

- ✓ 分布式异构架构
- ✓ 大规模计算核心
- ✓ 海量内存
- ✓ 有限外存

**适合在内存中进行大规模
机器学习、大规模图计算
的大数据计算任务！**

现代超算实例-太湖之光

- 2017年11月，神威·太湖之光超过天河2号，摘得Top500桂冠。



现代超算实例-太湖之光

- 美国能源部公布了新一代超级计算机Summit



5.2 集中式计算架构

5.2.4 大数据对超级计算机提出的挑战

	大数据应用	高性能计算应用
应用领域	政府、商业、金融	科学与工程
密集型	数据	计算
数据存储	计算节点附近	与计算分离
并行性	隐式并行（数据并行）	显示并行
粒度	粗粒度	细粒度
耦合度	松耦合	紧耦合
软件平台	Hadoop, Spark, Storm 等	MPI, OpenMP, Lustre等

5.2 集中式计算架构

5.2.4 大数据对超级计算机提出的挑战

	传统超算	现代超算
CPU	单个CPU，单核或多核	多CPU，多核（通常单CPU超过十核）
异构计算卡	无	MIC和GPU
规模	几十至几百个节点	几千到几万个节点
网络带宽	未优化	支持10GB/s级别数据传输
容错性	耦合度高，一个节点损坏整个超算无法正常工作	耦合度低，每个节点都是一个单独服务器，容错性与大规模分布式集群相当

目录

5.1 引言

5.2 集中式计算架构

5.3 分布式计算架构

5.4 处理加速技术 (GPU/TPU/FPGA)

5.5 本章小结

5.3 分布式计算架构

Typical Big Data Problem

- Iterate over a large number of records
- Extract something of interest from each record
- Shuffle and sort intermediate results
- Aggregate intermediate results
- Generate final output

数据并行

并行批处理

Map

Reduce

Key idea: provide a functional abstraction for these two operations

5.3 分布式计算架构

5.3.1 基于MapReduce 批处理计算架构

- 批处理应用是一类常见的大数据应用
- 离线处理所有数据
- 用户共享硬件资源，能避免资源闲置，提高资源利用率
- 最常用的批处理编程模型是Google提出的MapReduce



5.3 分布式计算架构

5.3.1 基于MapReduce 批处理计算架构

- MapReduce的主要特点

构建抽象并行的
编程模型

提供统一的并行
计算框架

搭建高性能并行
计算平台

5.3 分布式计算架构

5.3.1 基于MapReduce 批处理计算架构

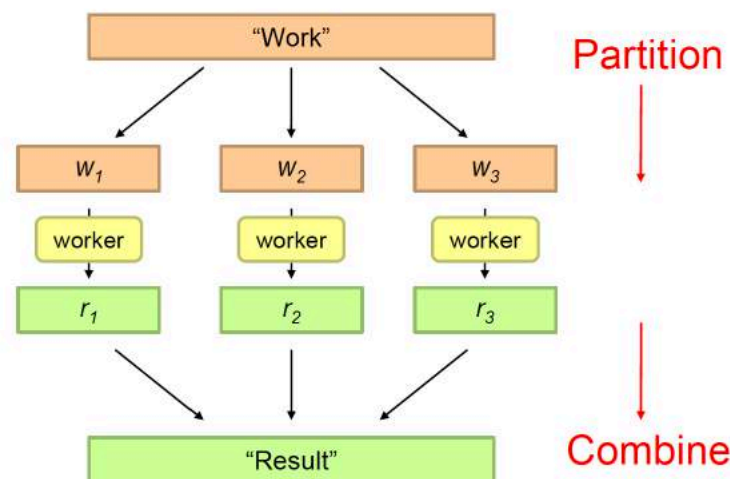
- 通过map操作获取海量网页的内容并建立索引，利用reduce操作根据网页索引处理关键词。
- 目前有上万个不同的算法问题和程序都使用MapReduce进行处理，包括大规模图形处理、文字处理、数据挖掘和机器学习等。

5.3 分布式计算架构

5.3.1 基于MapReduce 批处理计算架构

Divide and Conquer

采用“**分而治之**”的思想，把对大规模、相互间不具有计算依赖关系的数据集的操作，分发给多个节点共同完成，然后通过整合各个节点的中间结果，得到最终结果。

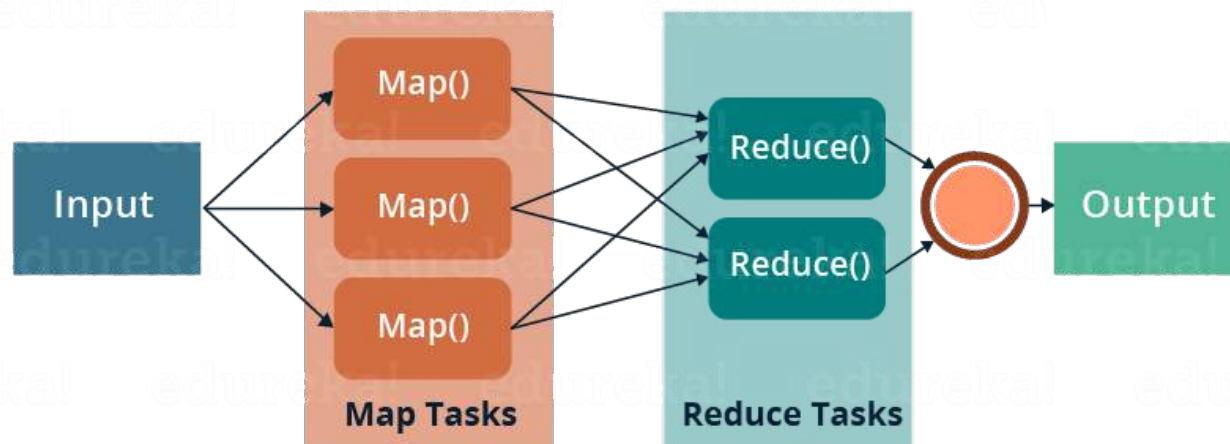


5.3 分布式计算架构

5.3.1 基于MapReduce 批处理计算架构

- MapReduce 编程模型

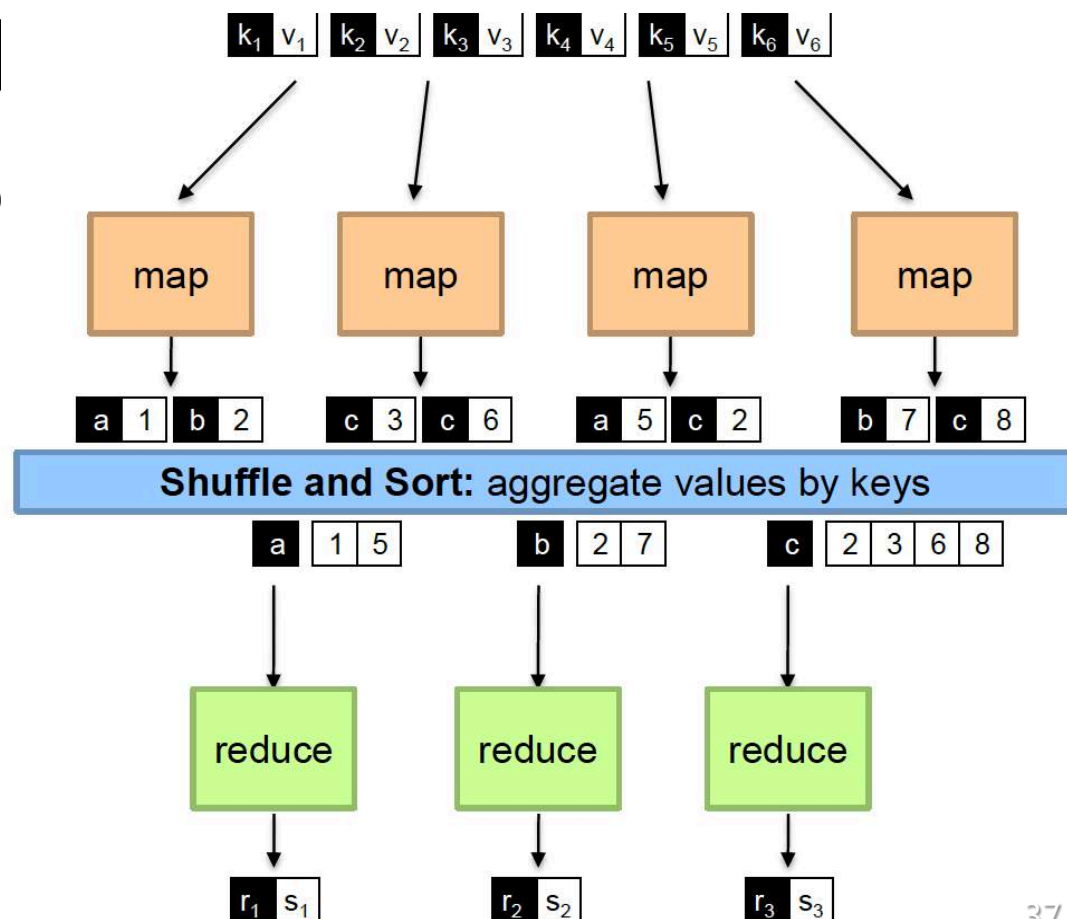
- 把对大规模、相互间不具有计算依赖关系的数据集的操作，分发给多个节点共同完成，然后通过整合各个节点的中间结果，得到最终结果。



5.3 分布式计算架构

5.3.1 基于MapReduce

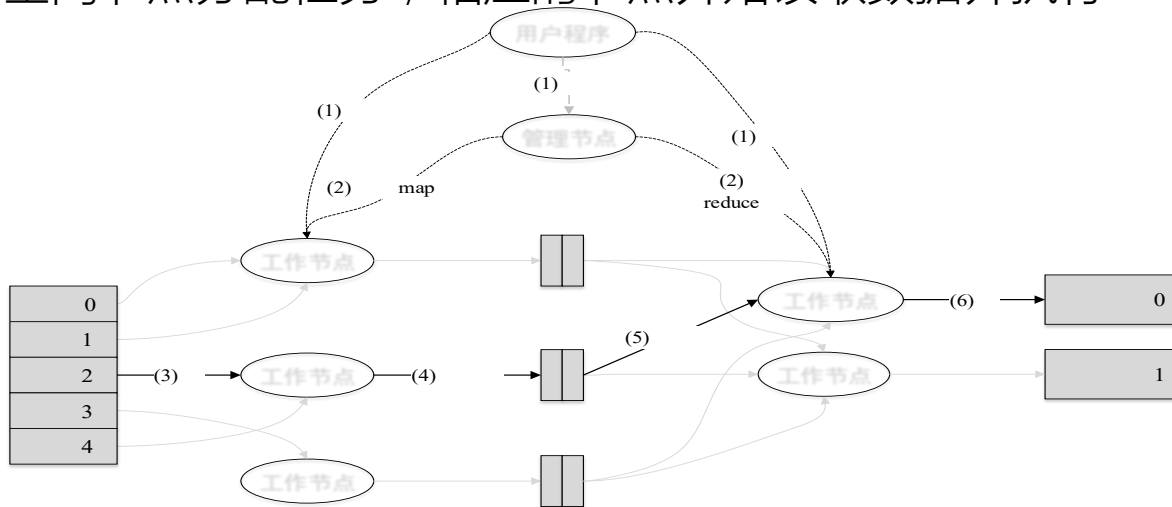
- 将集群节点分为**管理节点**和**工作节点**
- 执行过程分为**Map**、**Shuffle**和**Reduce**三个阶段



5.3 分布式计算架构

• Map阶段

- 执行分片 (Split) 和映射 (Map) 两个操作。
- Split将用户输入数据切分为多个逻辑分片，再将用户程序拷贝到多个节点上并创建相应的执行进程。
- 管理节点为空闲节点分配任务，相应的节点开始读取数据并执行map函数体功能。



5.3 分布式计算架构

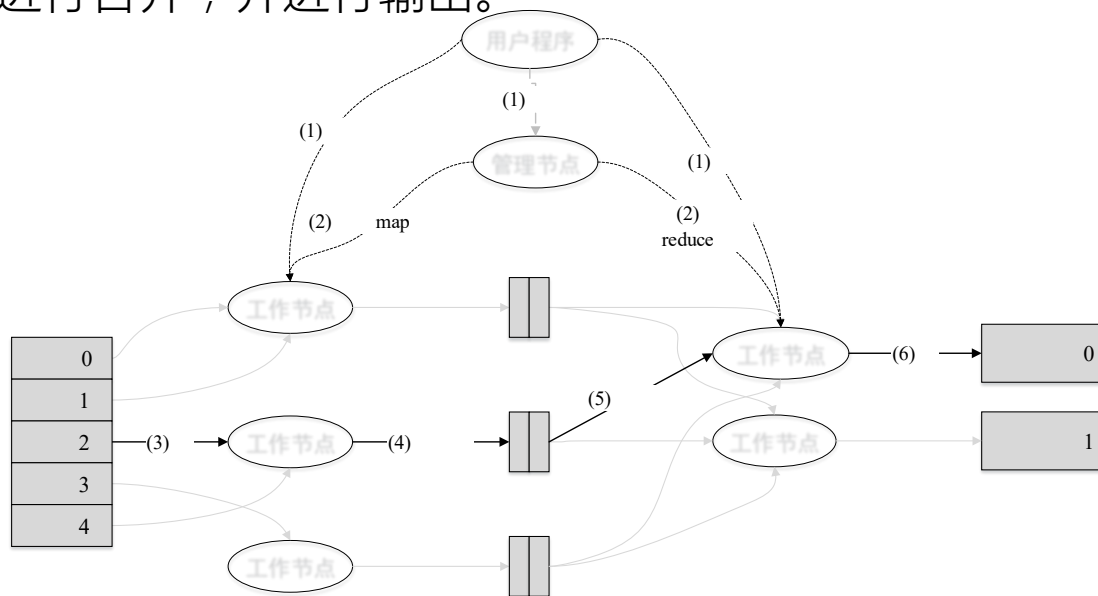
• Shuffle阶段

- 分为Map端的Shuffle操作和Reduce端的Shuffle操作两类。
- Map端shuffle
 - 将执行map任务的工作结点计算得到的中间键值数据写入磁盘。
 - 将中间键值数据划分为多个partition，其数量与reduce任务数量相同。
 - 每个partition中的数据按照key进行排序，并合并key值相同的键值对，再写入本地磁盘。
- Reduce端shuffle
 - 根据管理节点提供的地址信息，从执行map任务的工作节点的磁盘上领取属于自己的数据，并放入内存中。
 - 如果数据过多，会被写入本地磁盘。

5.3 分布式计算架构

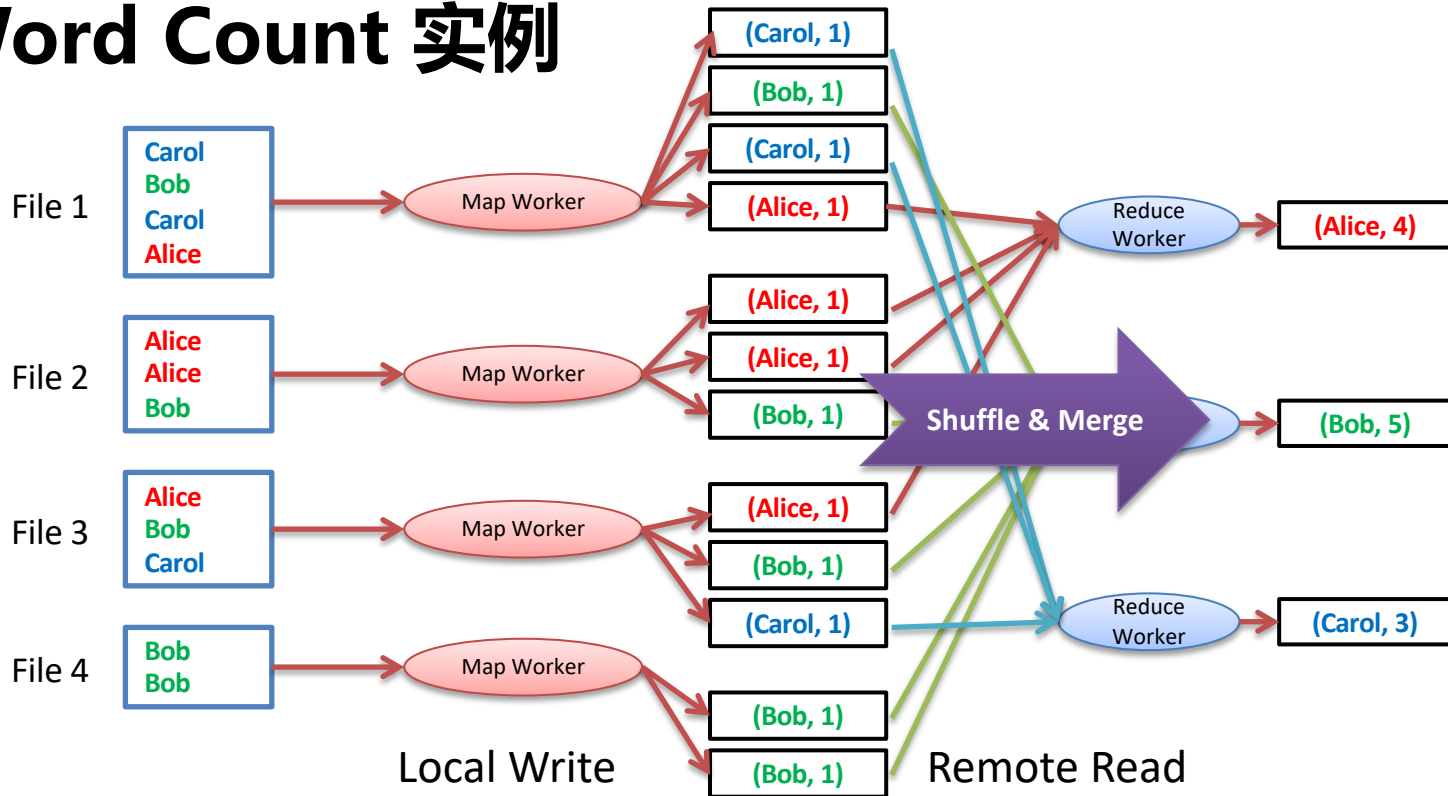
• reduce阶段

- 遍历shuffle阶段生成的有序键值数据文件，将具有相同key的键值对数据传递给reduce函数进行合并，并进行输出。



5.3 分布式计算架构

• Word Count 实例



5.3 分布式计算架构

- Word Count 实例

```
//Pseudo-code for "word counting"
map(String key, String value):
    // key: document name,
    // value: document contents
    for each word w in value:
        EmitIntermediate(w, "1");

reduce(String key, Iterator values):
    // key: a word
    // values: a list of counts
    int word_count = 0;
    for each v in values:
        word_count += ParseInt(v);
    Emit(key, AsString(word_count));
```

流处理

- **流数据**是指连续的、无边界的数据元素的序列，并且随着时间的不断的产生。



热搜榜



更多 >

🔴 关注重庆公交坠江事件 热

1 金庸去世 爆 480万

2 这两天怎么了 新 270万

3 超会挑联盟 停 264万

4 王光英去世 新 249万

5 唐嫣罗晋婚礼主持 热 176万

6 香港四大才子只剩蔡澜和... 新 139万

7 陈小春悼念金庸 新 122万

8 公交坠江发现9名遇难者 热 75万

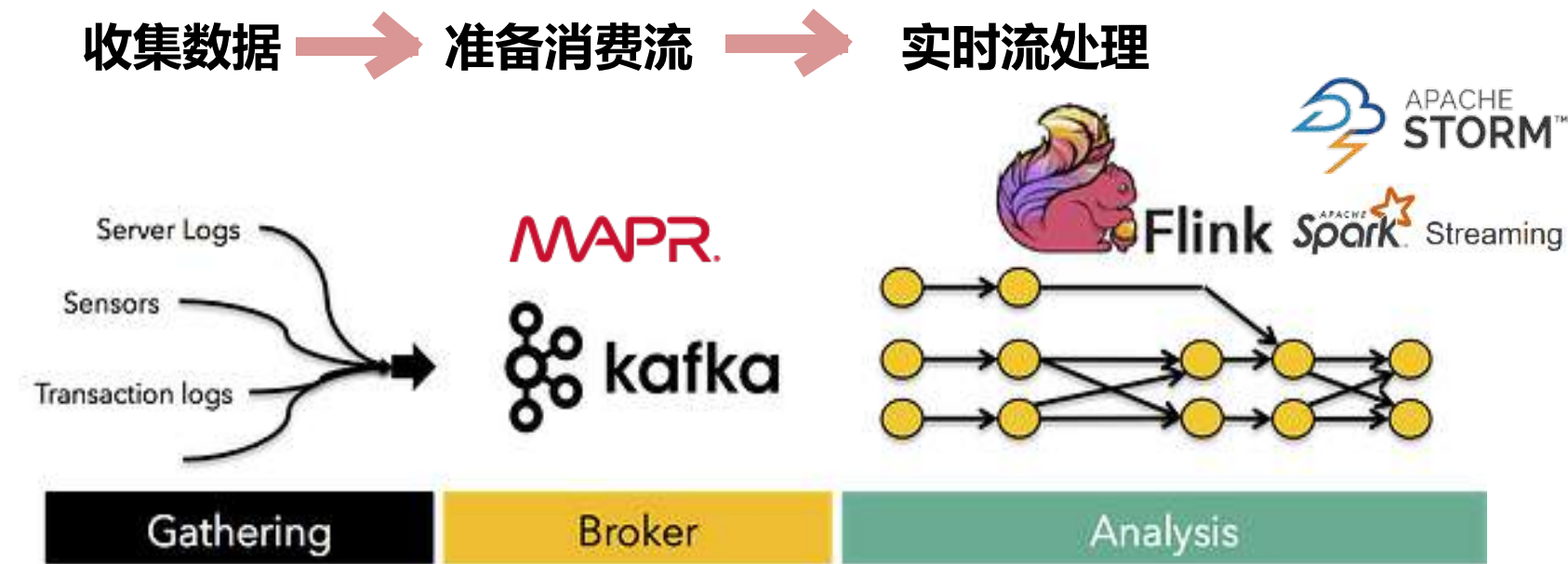
9 悲伤的十月 新 72万

10 哈文朋友圈 好友辟谣 热 71万

5.3 分布式计算架构

流处理

- 流处理的一般架构：



5.3 分布式计算架构

流处理

- 流处理的基本数据模型：元组
- 元组是最小的可被处理的数据单元，每一个元组可以表示成值的列表。

有什么新鲜事想告诉大家？ 已输入6字

Hello world!

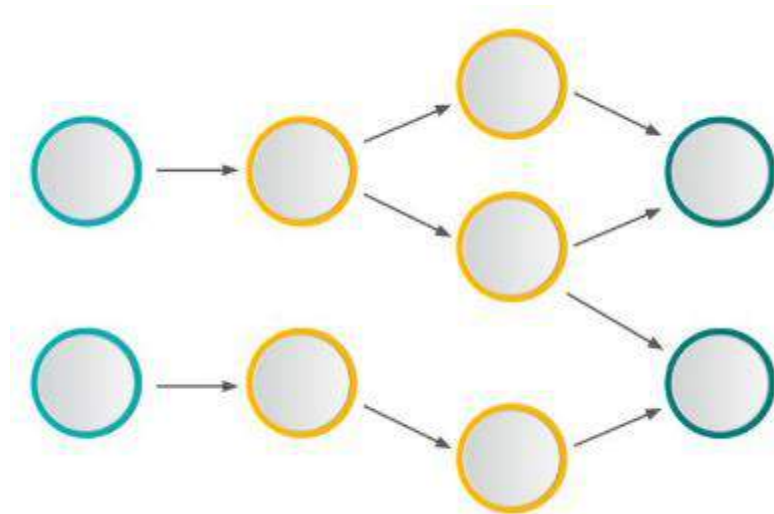
 表情  图片  视频  话题  头条文章 ... 公开 发布

```
{  
  "user_id": 17055506,  
  "timestamp": 1421777578,  
  "status": "Hello world!"  
}
```

5.3 分布式计算架构

流处理

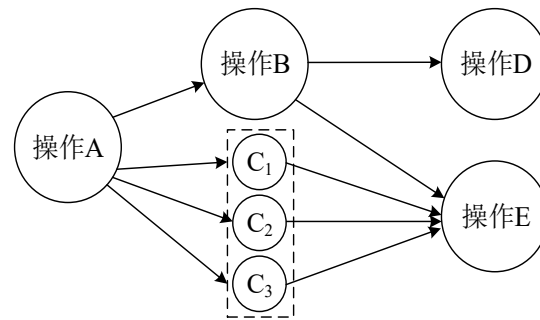
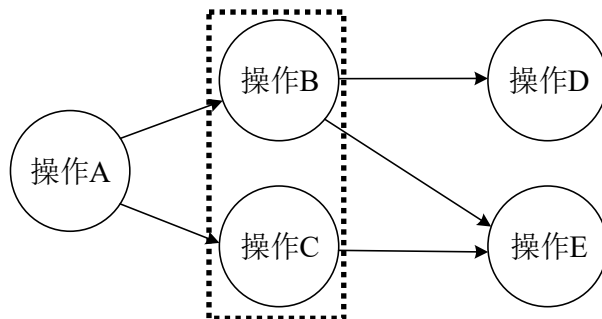
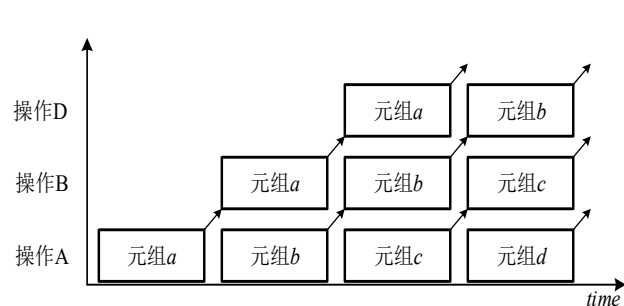
- 流处理的应用模型：DAG
- 流应用视作一个有向无环图
- **节点**：一个简单的操作
- **边**：元组的流动方向



5.3 分布式计算架构

流处理

- 流处理的并行
- 流水线并行，DAG的任务并行，数据并行



5.3 分布式计算架构

流处理



5.3 分布式计算架构

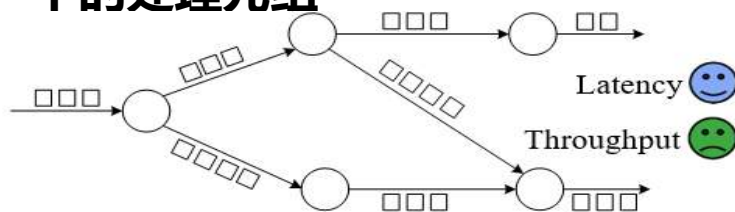
流处理

- 两种典型的流处理编程模型

不间断的数据流：一个接一个的处理元组

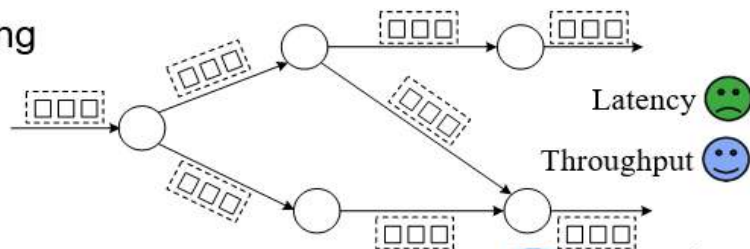
➤ Apache Storm

➤ Apache Flink



微量批数据：以微小批量的数据的方式处理元组

➤ Spark Streaming



5.3 分布式计算架构

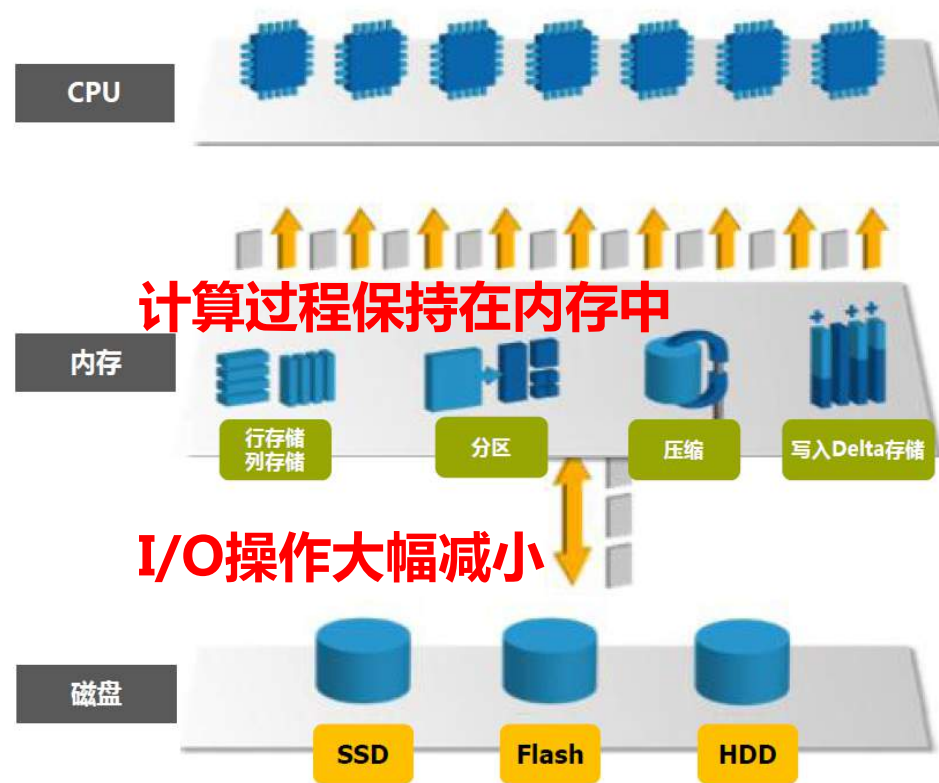
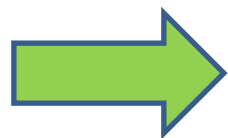
内存计算系统

- 内存计算是以大数据为中心，依靠新型的软件体系结构，通过对计算机系统结构、系统软件、编程模型等进行重大革新，将**数据装入内存中处理**，进行**避免I/O操作**的一种新型的**以数据为中心的并行计算**模式。



5.3 分布式计算架构

内存计算系统



5.3 分布式计算架构

内存计算系统



- 典型**分布式内存计算框架**：Spark

Spark最初由美国加州伯克利大学（UC Berkeley）的AMP 实验室于2009年开发。是基于内存计算的大数据并行计算框架，可用于构建大型的、低延迟的数据分析应用程序

2013年Spark加入Apache孵化器项目后发展迅猛，如今已成为Apache软件基金会最重要的三大分布式计算系统开源项目之一（Hadoop、Spark、Storm）

具备高度并行化、低延迟、可扩展特性

5.3 分布式计算架构

内存计算系统

- 典型**分布式内存计算框架**：Spark
- Spark 提供了一个内存范式，将 Hadoop 扩展到流式传输、迭代 MapReduce 和图形分析操作。换句话说，Spark 支持通用并行计算。

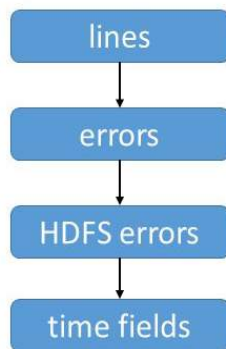


5.3 分布式计算架构

- Spark 的数据模型是**弹性分布式数据集 RDD**
Resilient Distributed Datasets(RDDs)

- An RDD is an **immutable, in-memory, partitioned** logical collection of records
- RDDs maintain **lineage** information that can be used to reconstruct lost partitions

“血缘”
依赖关系



```
lines = spark.textFile("hdfs://...")
```

```
errors=lines.filter(_.startsWith("ERROR"))
```

```
HDFS_errors=filter(_.contains("HDFS"))
```

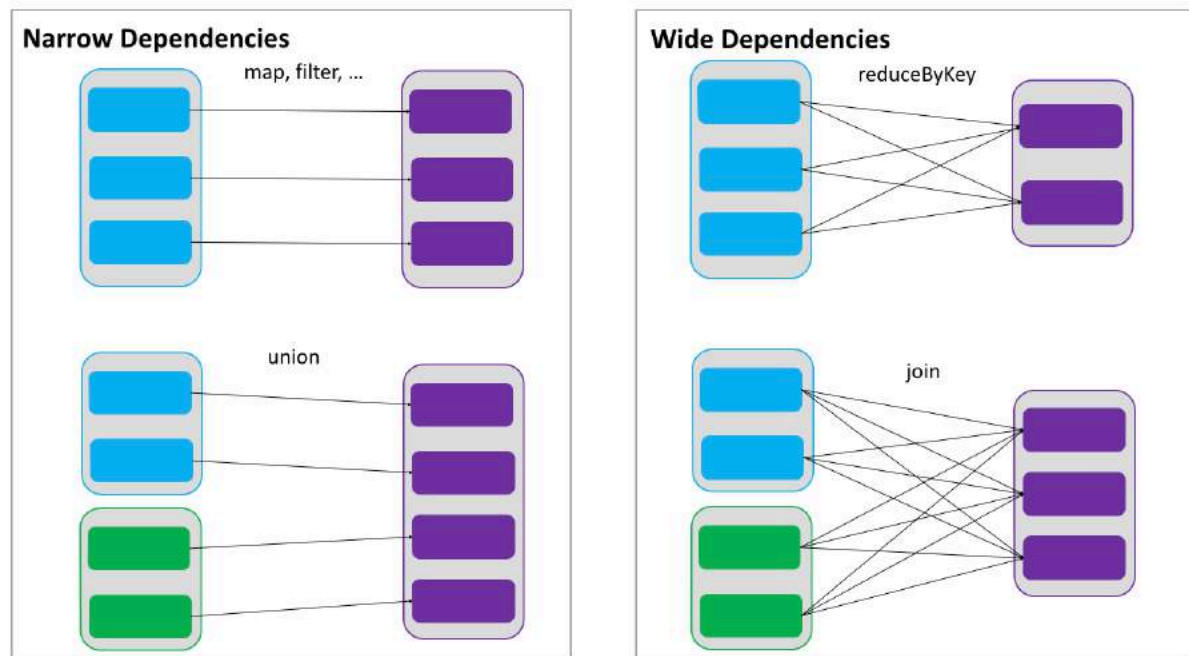
```
HDFS_errors=map(_.split('\t')(3))
```

表示一个只读的
记录分区的集合，
它只能通过其他
RDD转换而创建，
为此，RDD支持
丰富的操作函数
(如flatMap,
count, filter,
distinct等)

5.3 分布式计算架构

• RDD 窄依赖与宽依赖

- **窄依赖**，RDDs之间分区是**一一对应**的
- **宽依赖**，下游RDD每个分区与上游RDD(也称之为父RDD)的每个分区都有关，是**多对多**的关系

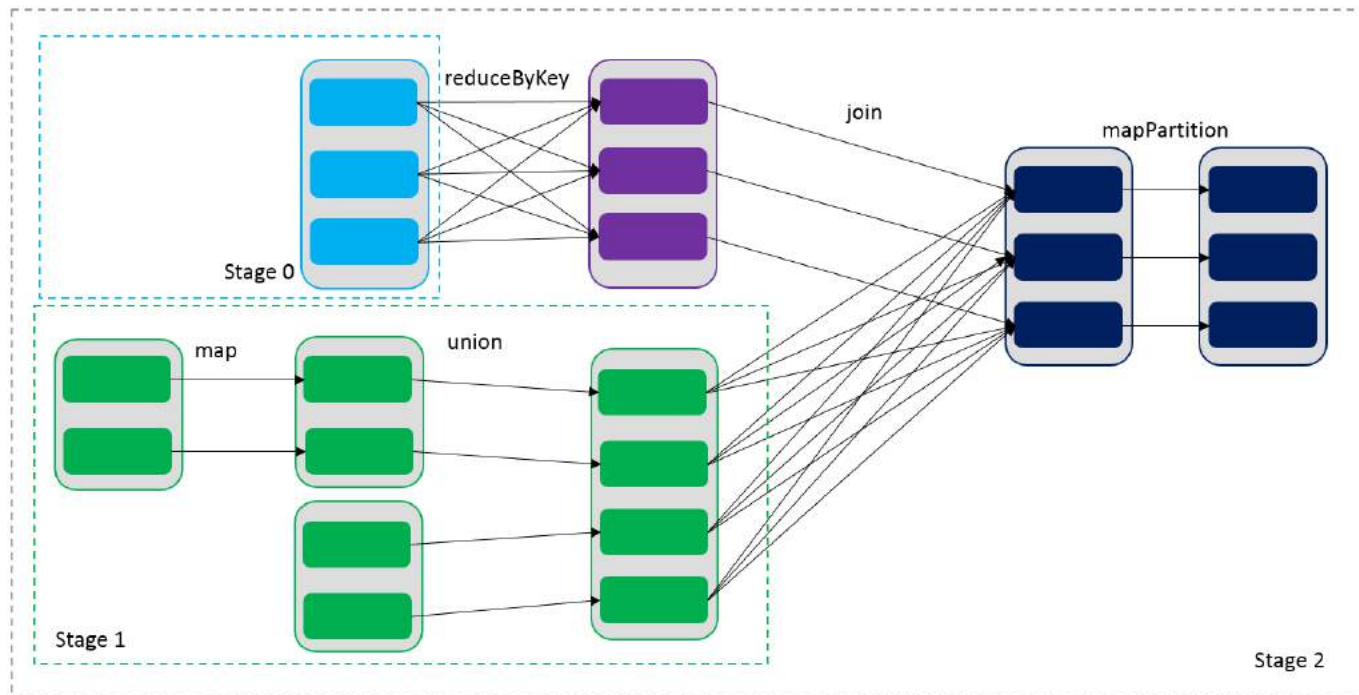


RDD之间依赖关系

5.3 分布式计算架构

• RDD 窄依赖与宽依赖

- 宽依赖对应于 Shuffle(图中的 reduceByKey和join)
- 窄依赖中的所有转换操作可以通过类似于管道的方式一气呵成执行(图中map和union可以一起执行)



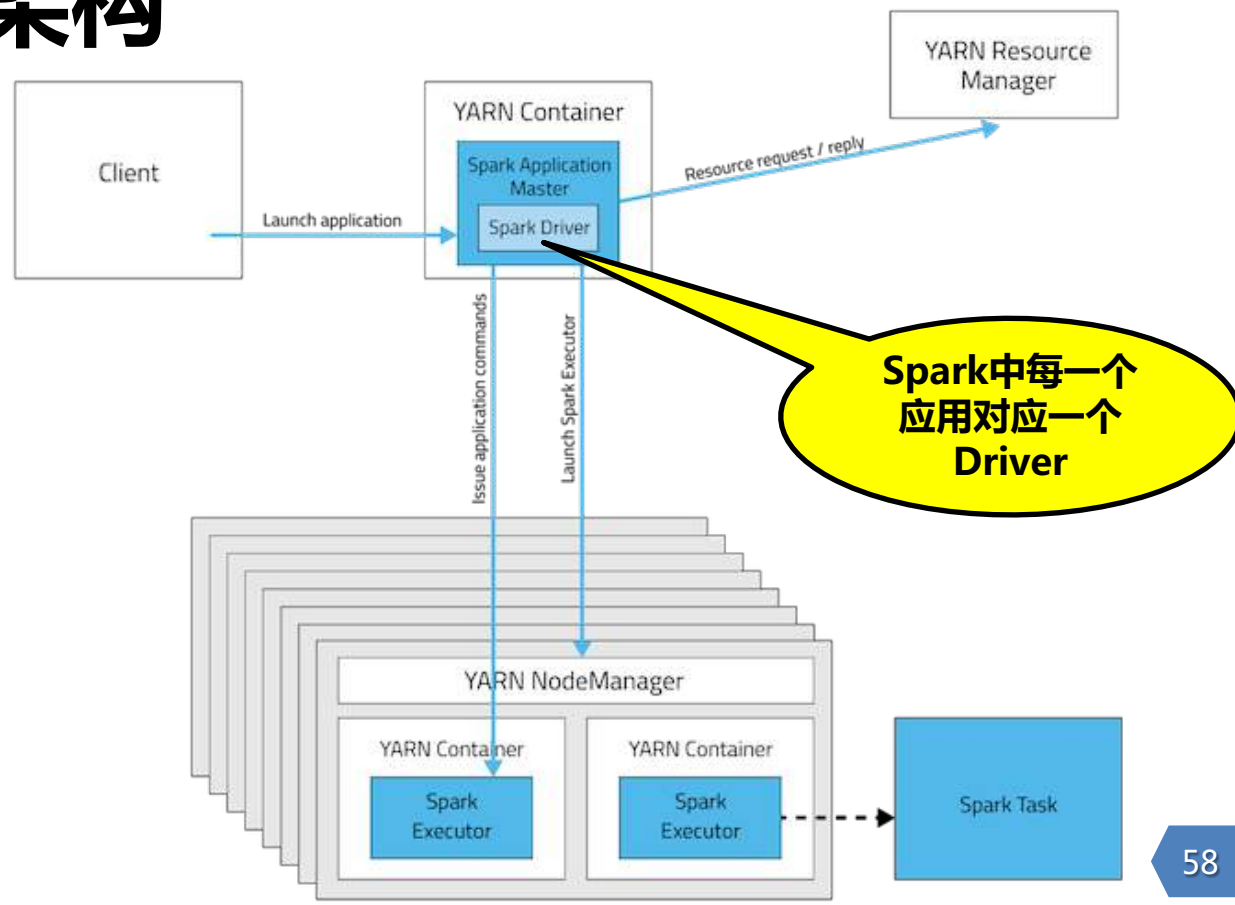
5.3 分布式计算架构

- RDD 特性

- 将不同来源、不同类型的数据进行统一
- 对 RDD 的操作可以叠加到一起计算
- RDD 提供了更丰富的数据集操作函数
- 提供了简洁的编程界面

5.3 分布式计算架构

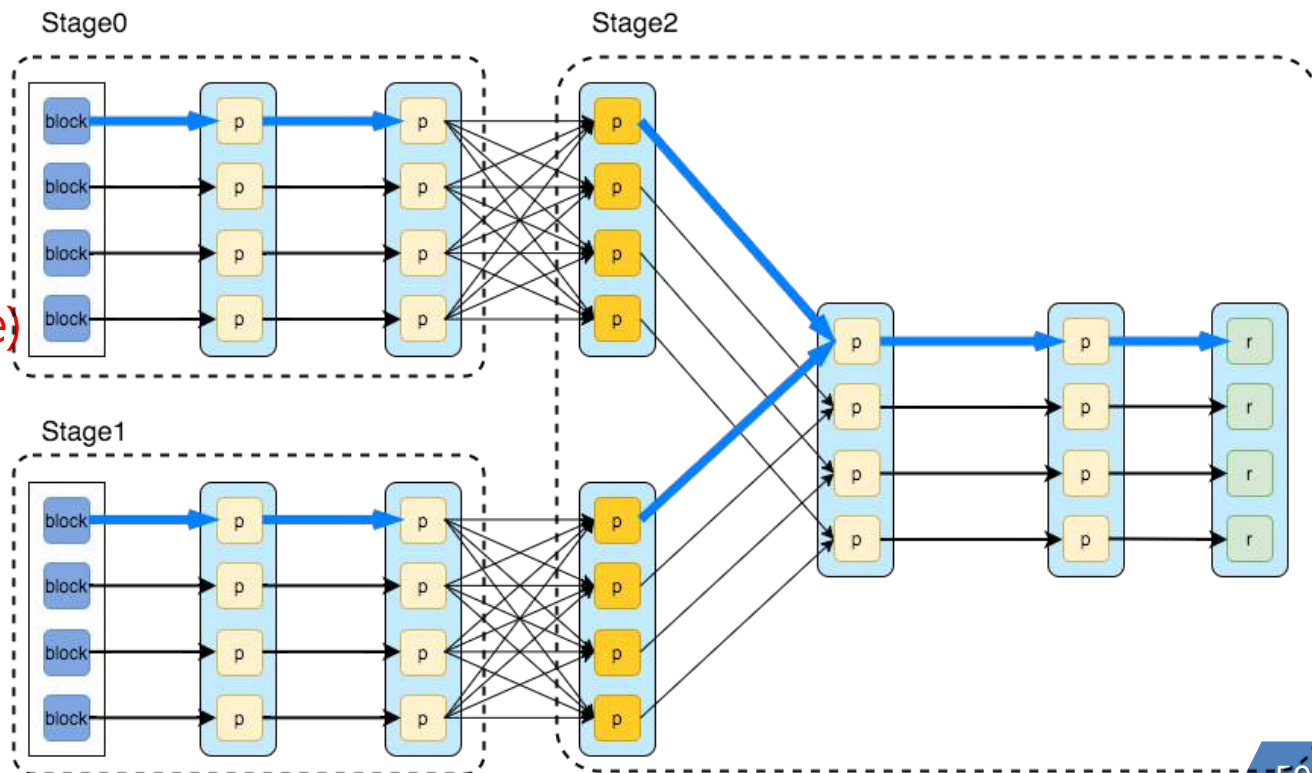
- Spark on YARN
- Spark 自身只负责计算
- 计算资源的管理和调度由第三方框架实现
 - YARN , Mesos 等



5.3 分布式计算架构

- Spark DAG

- Spark 根据 RDD 的依赖关系构建计算阶段(stage)的有向无环图(DAG)



5.3 分布式计算架构

- Spark 计算过程

- RDD 构建：构建 RDD 之间的依赖关系，将 RDD 转换为阶段的有向无环图
- 
- ```
graph TD; A[RDD 构建：构建 RDD 之间的依赖关系，将 RDD 转换为阶段的有向无环图] --> B[任务调度：根据空闲计算资源情况进行任务提交，并对任务的运行状态进行检测和处理]; B --> C[任务计算：搭建任务运行环境，执行任务并返回任务结果]; C --> D[Shuffle 过程：两个阶段之间有宽依赖时，需要进行 Shuffle 操作];
```

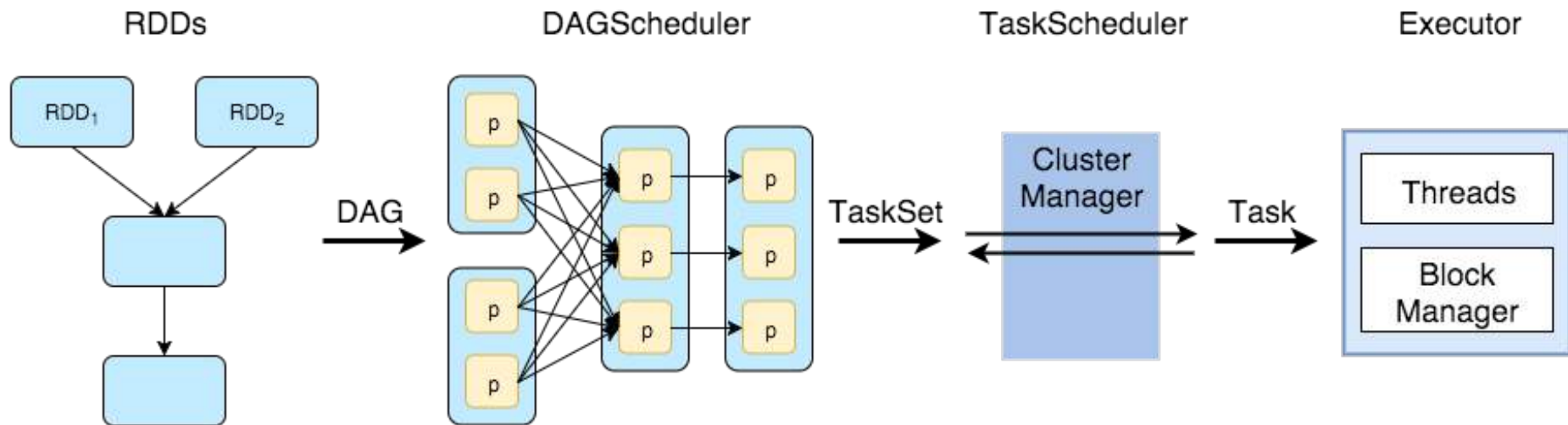
- 任务调度：根据空闲计算资源情况进行任务提交，并对任务的运行状态进行检测和处理

- 任务计算：搭建任务运行环境，执行任务并返回任务结果

- Shuffle 过程：两个阶段之间有宽依赖时，需要进行 Shuffle 操作

## 5.3 分布式计算架构

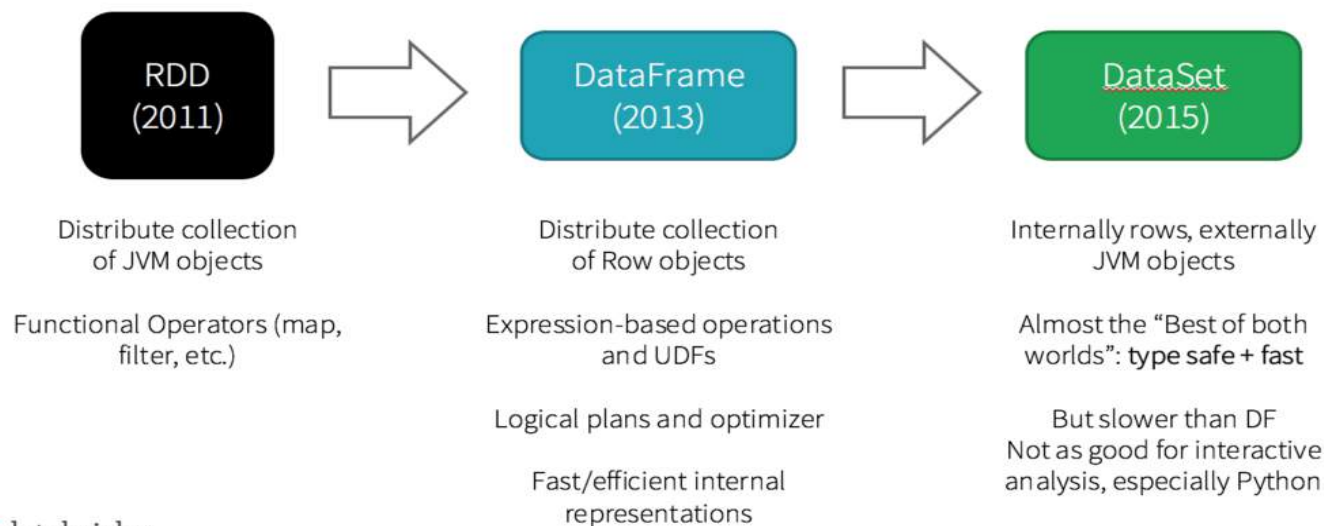
- Spark 计算过程



## 5.3 分布式计算架构

### 5.3.2 基于Spark分布式内存计算架构

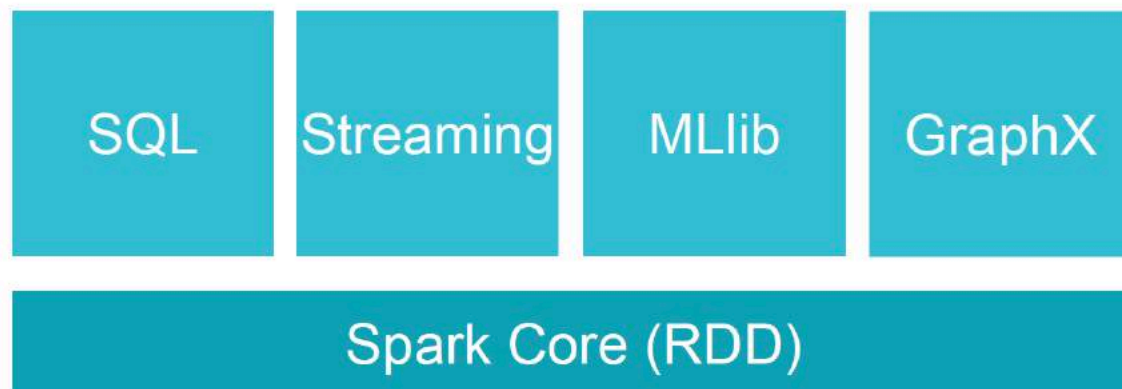
History of Spark APIs



## 5.3 分布式计算架构

### 5.3.2 基于Spark分布式内存计算架构

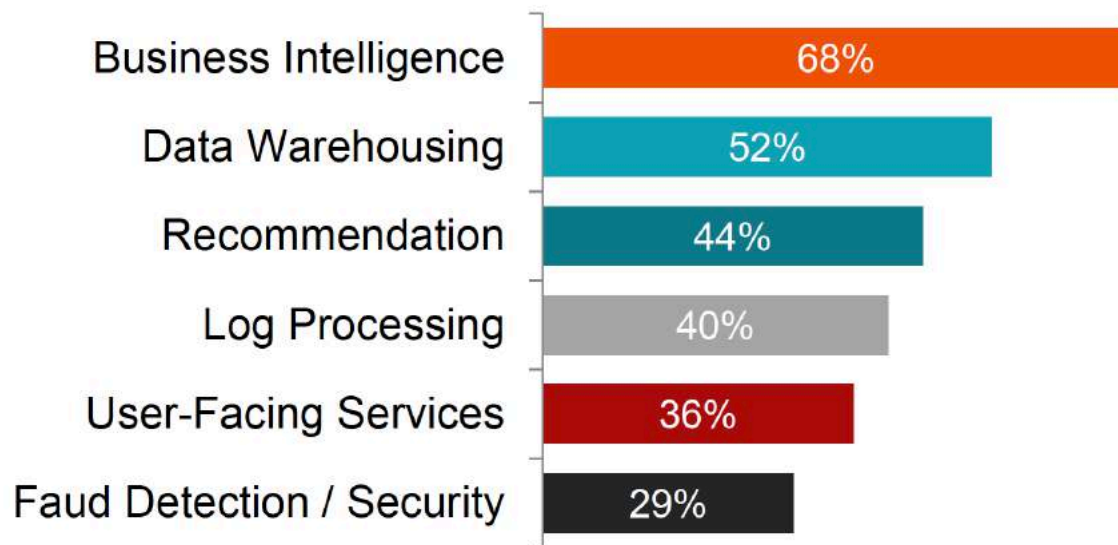
Spark stack diagram



## 5.3 分布式计算架构

### 5.3.2 基于Spark分布式内存计算架构

Top Applications

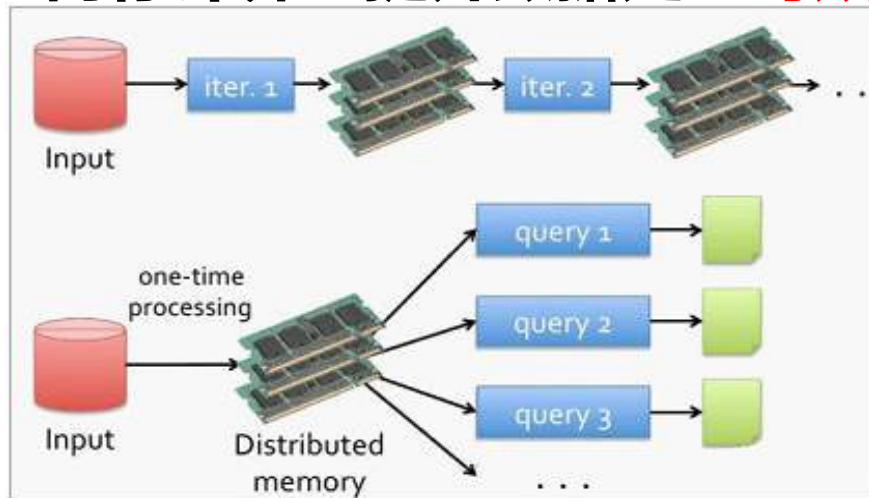




## 5.3 分布式计算架构

### 内存计算系统

- 内存计算：提升数据处理**时效性**的重要手段

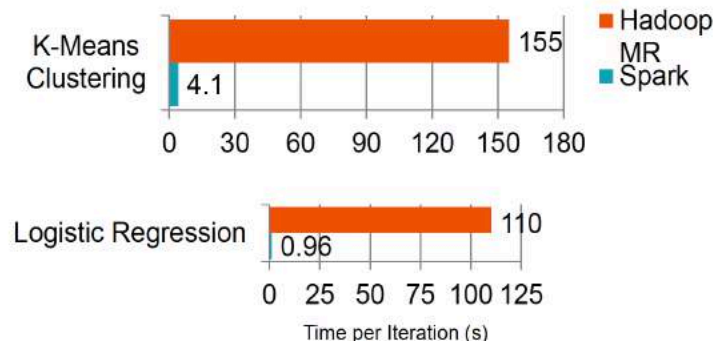


Spark比Hadoop性能提升**10-100**倍

- Spark SQL: Relational Data Processing in Spark**

Armbrust, Reynold S. Xin, Cheng Lian, Yin Huai, Davies Liu, Joseph K. Bradley, Xiangrui Meng, Tomer Kaftan, Michael J. Franklin, Ali Ghodsi, Matei Zaharia. SIGMOD 2015. June 2015.

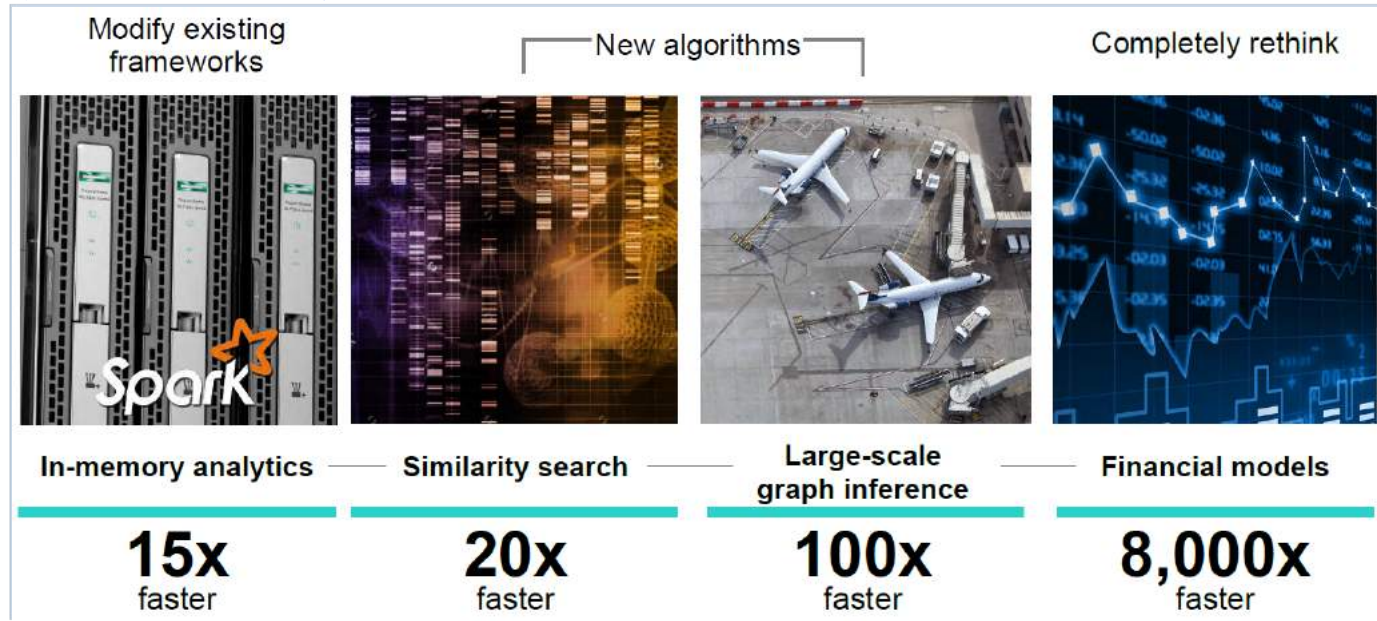
Performance



# 5.3 分布式计算架构

## 内存计算系统

- 新型内存计算系统

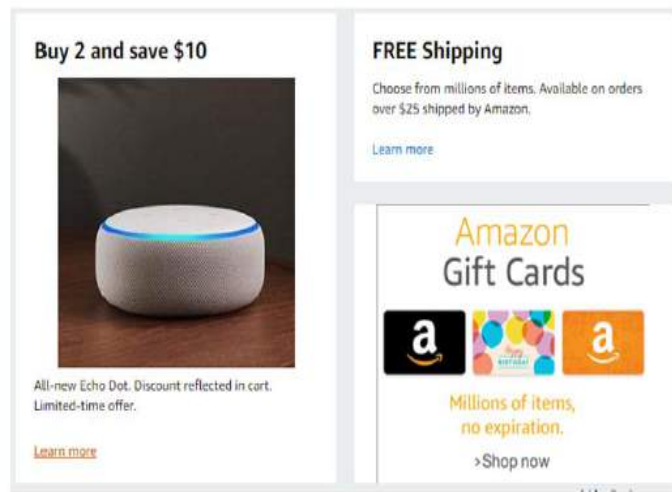


源自：HP高级工程师Dr. Kimberly Keeton在MSST'17上的特邀报告

# 5.3 分布式计算架构

## 图处理框架

- 图处理普遍存在



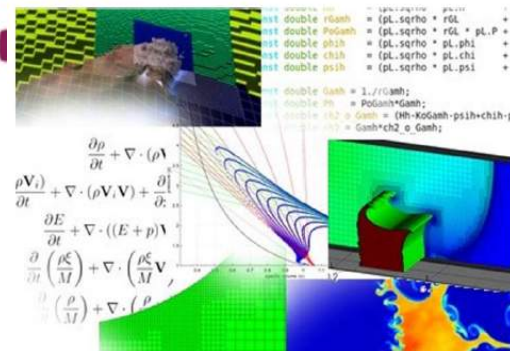
scientific calculation



scientific calculation



social network analysis



scientific calculation

## 5.3 分布式计算架构

### 图处理框架

- 图数据越来越大



> 1 M vertices  
> 100 M edges  
\*Distributed  
graphLab-2012

advertising network

> 1 B vertices  
> 1 T edges  
\*Facebook  
Engineering Blog



social media



> 50 B vertices  
> 1 T edges  
\*NSA Big Graph  
Experiment -  
2013

internet

> 100 B vertices  
> 100 T edges  
\*NSA Big graph  
Experiment -  
2013



scientific calculation

## 5.3 分布式计算架构

### 图处理框架

- 图数据面临的问题

#### 数据驱动计算

- 图形计算任务与顶点和边密切相关
- 计算结构运行前难以预测，难以并行

#### 不良的局部访问性

- 顶点/边的非连续访问
- 现有缓存机制只能加速具有良好局部性的访问

#### 不规则的结构

- 图的结构不规则，很难有效得分割

#### 访问/计算占比率高

- 大量的访问导致CPU等待

## 5.3 分布式计算架构

### 图处理框架

- 面向图数据的编程模型

#### 图编程模型应该：

- 简洁，易于使用
- 灵活，易于表示不同图形任务
- 有效，易于适应计算架构

#### 现有编程模型：

- 以顶点为中心
- 以边为中心
- 以路径为中心
- 以子图为中心

## 5.3 分布式计算架构

### 图处理框架

- **以顶点为中心的编程模型**

该模型把计算范围限定在图数据中的单个点，即**从单个点的角度**考虑图算法执行过程，包括每个点上的计算及相邻点之间的消息传递。这样各个点可以实现相互独立的计算，从而进行细粒度的并行。

-- "Think Like a Vertex" philosophy



## 5.3 分布式计算架构

### 图处理框架

- **以顶点为中心的编程模型**

该编程模型处理图数据中的**每个点**都需要采取**三种操作**：

#### 获取信息

点获取所有邻接点更新的状态信息，并为点的状态更新做准备

#### 更新信息

点根据收取邻接点的状态信息来更新自身的状态

#### 分发信息

点把更新的状态信息通过边传送出去。

**缺点**：在计算过程中存在大量的随机访存操作，这将导致较大的时间开销。

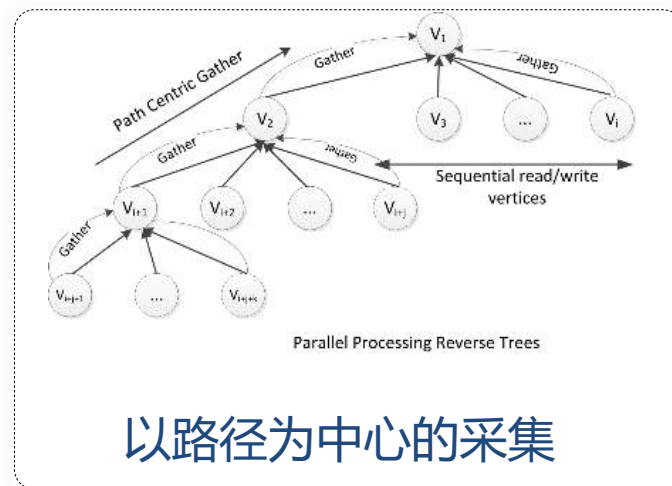
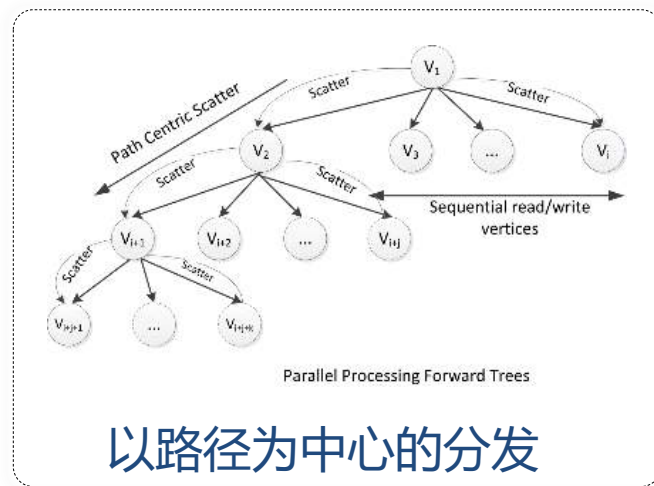
**相关图计算框架**：Pregel 等



## 5.3 分布式计算架构

### 图处理框架

- 以路径为中心的编程模型



**特点：**顺序读写不仅减少了数据在定位时所耗费的时间，同时减轻了系统在I/O上面的负载。

**相关图计算框架：**TripleGraph等

# 目录

5.1 引言

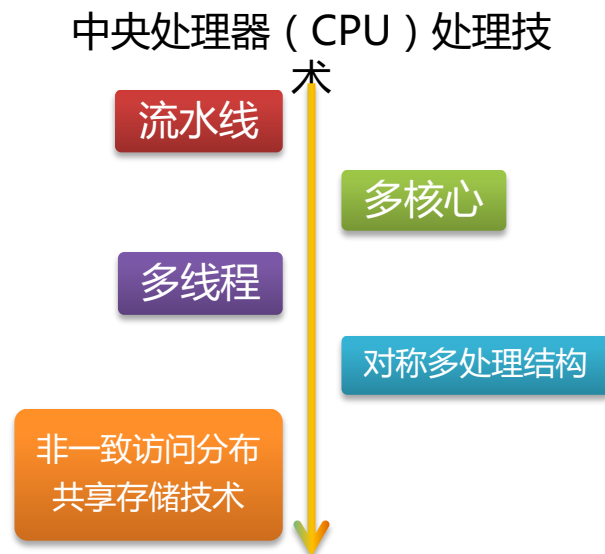
5.2 集中式计算架构

5.3 分布式计算架构

**5.4 处理加速技术 ( GPU/TPU/FPGA )**

5.5 本章小结

# 针对CPU的计算优化



## 指令集优化

- 同样的计算采用内置的特殊指令集可以实现的更加高效
- 常用CPU扩展指令集：
  - MMX
  - SSE系列
  - 3DNow!

## 缓存优化

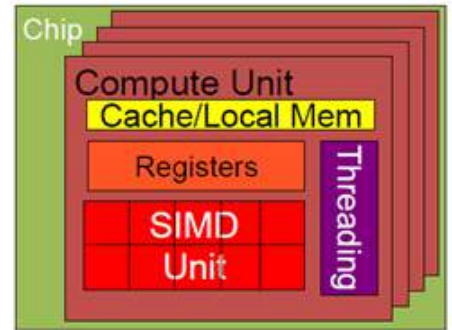
- 缓存容量的增大，可以大幅度提升CPU内部读取数据的命中率，避免访问内存或硬盘，以此提高系统性能
- 多个核心同时使用缓存是CPU计算优化的挑战

尽管CPU经历了从**单核**到**众核**的巨大变革，但针对CPU的计算优化已经非常成熟，优秀的**编译器能够自动完成**这项任务

# 5.4.1 GPU技术原理与大数据应用

## 图形处理器（GPU）

- 最初用于图形处理，后用于通用计算
- GPU的结构域CPU完全不同
  - 指令集简单，众多线程，并行度极高
  - 每个计算单元都是一个核心，适合做大规模浮点运算和矩阵运算



CPU与GPU性能对比：

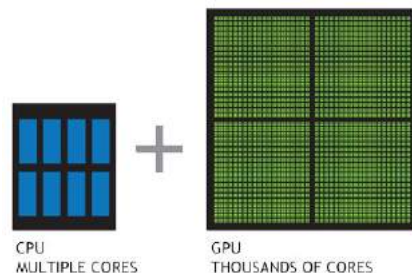
|              | Intel Xeon E7<br>CPU处理器 | Nvidia Tesla<br>K40 GPU处理器 |
|--------------|-------------------------|----------------------------|
| 核心数量         | 15                      | 2880                       |
| 内存（GB）       | 1536                    | 12                         |
| 传输带宽（GB/s）   | 85                      | 288                        |
| 计算能力（Tflops） | 0.2                     | 1.4                        |
| 价格（美元）       | 6841                    | 3875                       |

**GPU最初用于图形处理，后由于在大规模矩阵运算上的优异表现，被广泛用于通用计算中**

# 针对GPU的计算优化

## GPU计算优化

- 更多的芯片面积用于计算单元而不是缓存
- 更简单的逻辑处理部件，但提供更高效率的轻量级线程处理
- GPU编程环境区别于CPU，需要针对其设计程序



CPU任务调度+GPU并行计算

## GPU程序接口实例：Caffe

- 利用GPU的深度学习架构，用于训练深层卷积神经网络
- 一行代码无缝切换CPU-GPU编程模型：

```
Caffe::set_mode(Caffe::GPU);
```



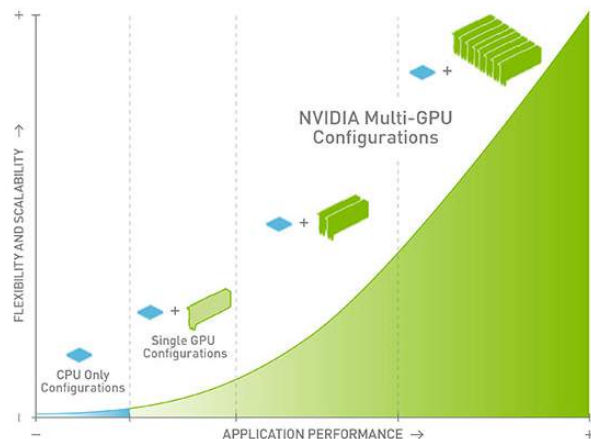
[github.com/BVLC/caffe](https://github.com/BVLC/caffe)

- 最快的卷积神经网络训练工具
  - 使用单个NVIDIA K40或Titan GPU
  - 每天可处理超过40M图像数据
  - 5毫秒/张训练图像，2毫秒/张测试图像

目前需要为GPU计算编写**特定代码**，好的**程序设计**对GPU计算性能提升巨大

# 针对GPU的计算优化

多GPU的协作计算是未来针对GPU计算优化的发展方向



多GPU计算实例：COTS

- 斯坦福大学推出的高性能深度卷积神经网络系统
- 包括3台高性能服务器，每台服务器挂载四块GPU计算卡
- COTS可以训练含超过**110亿参数**的深度神经网络

多GPU计算优化的挑战

- 合理的任务分配
- 弹性的负载均衡
- 高效的数据传输

**多GPU的协作计算将会成为异构计算的常态，需要投入大量精力进行优化**

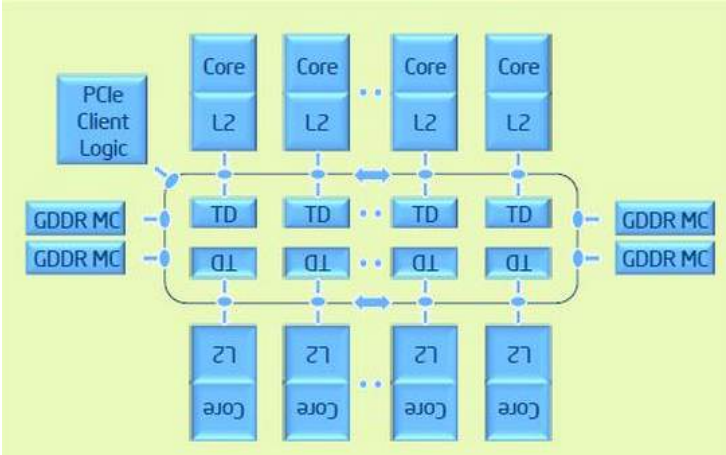
# 针对MIC的计算优化

众核处理器（MIC）

- MIC整合多个（至少超过30个）CPU到一块芯片，是Intel最新架构，与GPU相似
- MIC技术特点
  - 基于x86架构
  - 512bit向量宽度VPU
  - 每个核有4个硬件线程
  - 所有核心共享二级缓存

|               | Intel Xeon E7 CPU处理器 | Nvidia Tesla K40 GPU处理器 | Intel Xeon Phi 7120A 众核处理器 |
|---------------|----------------------|-------------------------|----------------------------|
| 核心数量          | 15                   | 2880                    | 61                         |
| 计算能力 (Tflops) | 0.2                  | 1.4                     | 1.2                        |
| 价格 ( \$ )     | 6941                 | 3875                    | 4235                       |

MIC可以**直接运行CPU源代码**，但是**直接使用效率低**，  
需要进一步优化才能达到最优效率



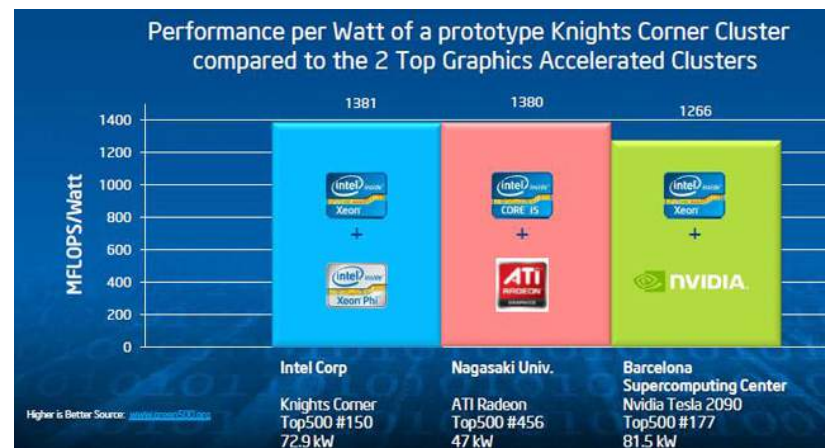
# 针对MIC的计算优化

## MIC计算优化

- 硬件线程数多（最高可达244），并行度高
- 512位SIMD向量计算单元，专门处理向量计算
- 向量计算单元与标量计算单元分离，同时利用可进一步加速

## MIC应用实例：MrPhi

- 基于MIC计算卡的MapReduce编程模型，充分利用MIC强大的向量计算单元和SIMD技术
- 相对于未经优化的多核MapReduce版本（Phoenix++）加速1.2到38倍
  - 针对向量化计算优化的map端
  - SIMD哈希算法
  - 流水线处理map和reduce任务
  - 取消局部数据，改用全局数组（利用MIC的原子操作优化）



虽然MIC支持大部分CPU指令集，比GPU计算优化简单，但要达到与GPU相当的加速效果，挑战仍很大；目前MIC尚未广泛应用于异构计算中



# 现代超算多采用异构体系结构

- 异构体系结构利用不同类型指令集和体系架构的计算单元组成高性能系统



天河一号A：配备16,384颗CPU和7,168块GPU计算卡



天河二号一期：配备32,000颗CPU和48,000块MIC计算卡

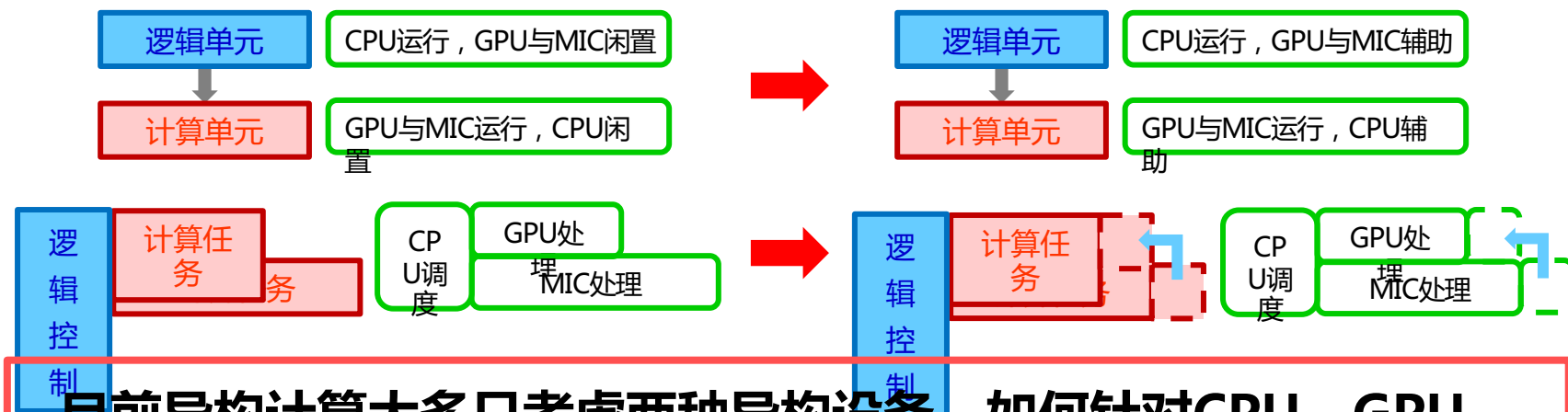
现代超算系统大多是基于CPU、GPU和MIC的**异构计算环境**，  
如何针对各自的特性进行**协同优化**，是目前的一项重大挑战

# 异构体系结构下的计算优化

CPU+GPU异构体系下的计算优化需要额外考虑动态资源调度等问题

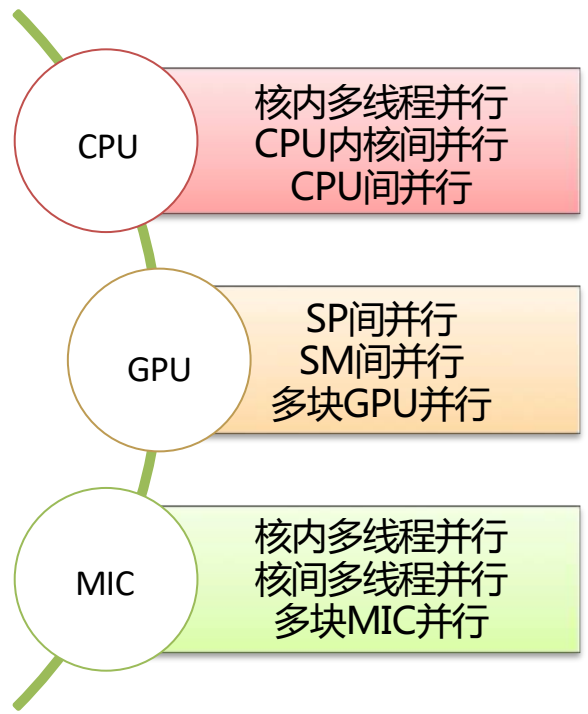
- 单纯GPU计算优化仅考虑提高并行度加速任务计算
- 针对单纯逻辑复杂或计算复杂的任务，利用闲置GPU或CPU分担计算可以最大化计算效率
- 同一任务可能同时包括逻辑复杂和计算复杂部分，拆分计算任务可以进一步提升计算效率

CPU+GPU+MIC的协同优化可以达到比单纯使用CPU或GPU或MIC更高的效率

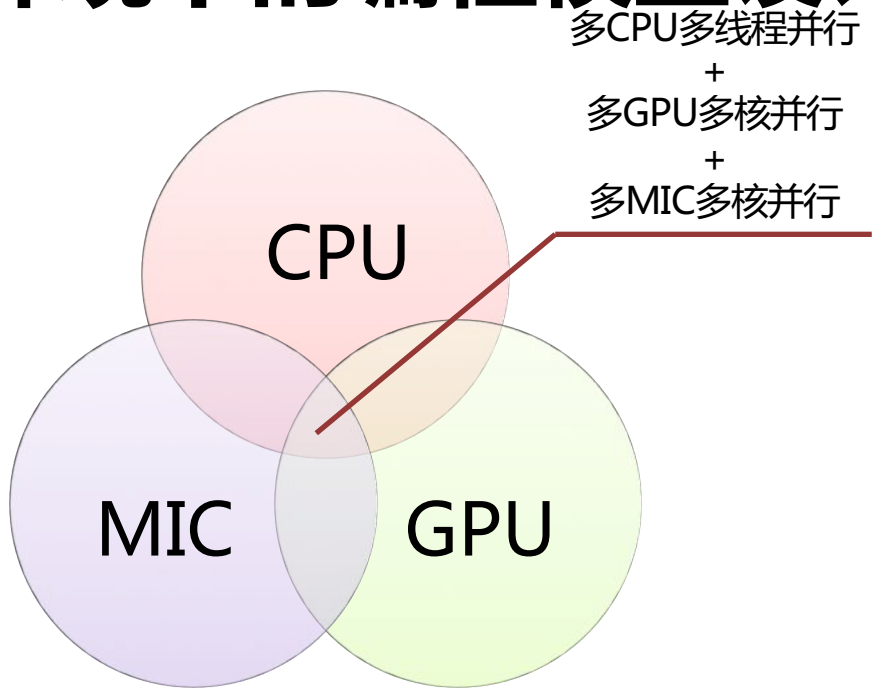


目前异构计算大多只考虑两种异构设备，如何针对CPU、GPU、MIC各自的特性进行**协同优化**，目前还没有相关成果

# 多粒度并行与分布式环境下的编程模型设计



多粒度并行编程模型



多粒度混合编程模型

# 多粒度并行与分布式环境下编程模型

## 天河一号编程模型

- ✓ CPU核内多线程并行
  - 四核至强CPU
- ✓ CPU核间并行
  - 14336颗至强CPU
  - 2048颗飞腾处理器
- ✓ GPU核间并行
  - 7168块Tesla计算卡
- ✓ CPU+GPU并行

## 天河二号一期编程模型

- ✓ CPU核内多线程并行
  - 十二核至强CPU
- ✓ CPU核间并行
  - 32000颗至强CPU
  - 每节点配备两颗
- ✓ MIC核间并行
  - 48000颗MIC协处理器
- ✓ CPU+MIC并行

## 天河二号X期编程模型

- ✓ CPU核内多线程并行
  - 十六核至强CPU
- ✓ CPU核间并行
  - 超过六万颗至强CPU
  - 每节点配备四颗
- ✓ MIC核间并行
  - 48000颗MIC协处理器
  - 48000块TeslaK40计算卡
- ✓ CPU+MIC+GPU并行

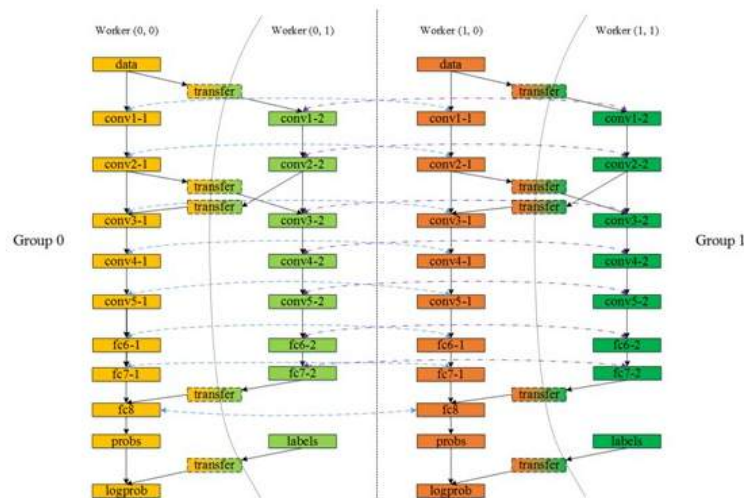
# 多粒度环境下的新编程模型目标

- 面向大数据分析
  - 支持大规模矩阵计算
  - 支持非凸优化计算
  - 支持大规模机器学习训练任务
  - 支持内存计算
- 效率与容错的统一
  - 多粒度并行计算容错机制
  - 高性能消息传递
  - 良好的资源调度
- 统一API，良好的用户使用接口
- .....

# 最新分布式异构环境下编程模型

## 腾讯Mariana

- Mariana系统包括深度神经网络的**GPU数据并行框架**，深度卷积神经网络的**GPU数据并行和模型并行框架**，以及深度神经网络的**CPU集群框架**
- 深度卷积神经网络模型并行和数据并行框架对GPU计算卡**分组**，组内的两个GPU卡做模型并行，组间做数据并行
- 针对Hinton在2012年获得ImageNet竞赛冠军用的网络，双卡模型并行取得**1.71倍**的加速比，4块GPU计算卡数据并行加模型并行时比单卡**2.52倍**的加速比

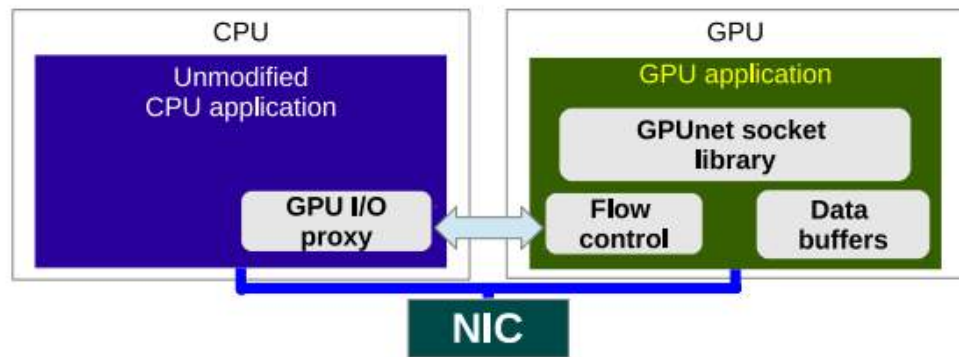


对ImageNet网络的模型并行和数据并行划分

# 最新分布式异构环境下编程模型

## GPUnet

- 是一个通用GPU网络编程模型，为GPU编程提供**统一高效的网络通信接口**
- 对多个GPU之间的网络通信进行优化，**减少网络开销**
- 提供经典的数据分析应用扩展包
  - **矩阵相乘服务**
  - **基于GPU内存的MapReduce框架**
  - **交互式面部识别服务**



GPUnet模型结构

## 5.4.2 TPU技术原理与大数据应用

- 张量处理器(tensor processing unit, TPU)是为机器学习定制的专用芯片(ASIC), 专为深度学习框架 TensorFlow 设计。

### Tensor Processing Unit (TPU)

- 30-80x TOPS/watt vs. 2015 CPUs and GPUs.
- 8 GiB DRAM.
- 8-bit fixed point.
- 256x256 MAC unit.
- Support for data reordering, matrix multiply, activation, pooling, and normalization.



**Figure 3.** TPU Printed Circuit Board. It can be inserted in the slot for an SATA disk in a server, but the card uses PCIe Gen3 x16.

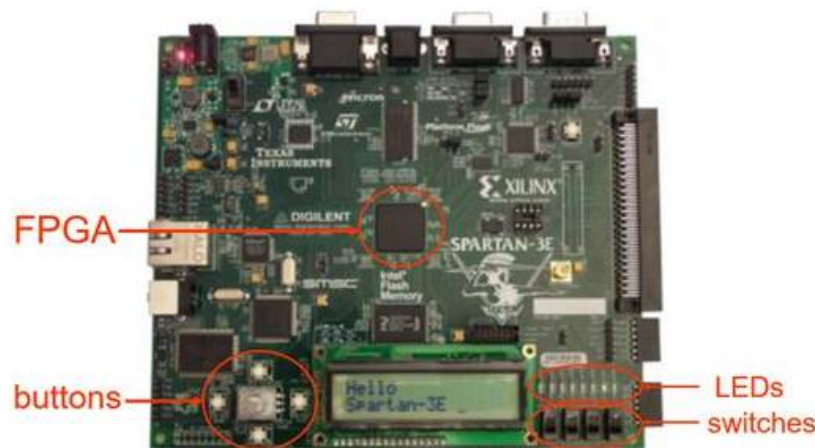


## 5.4.3 FPGA技术原理与大数据应用

### 现场可编程逻辑阵列(field programmable gate array, FPGA)

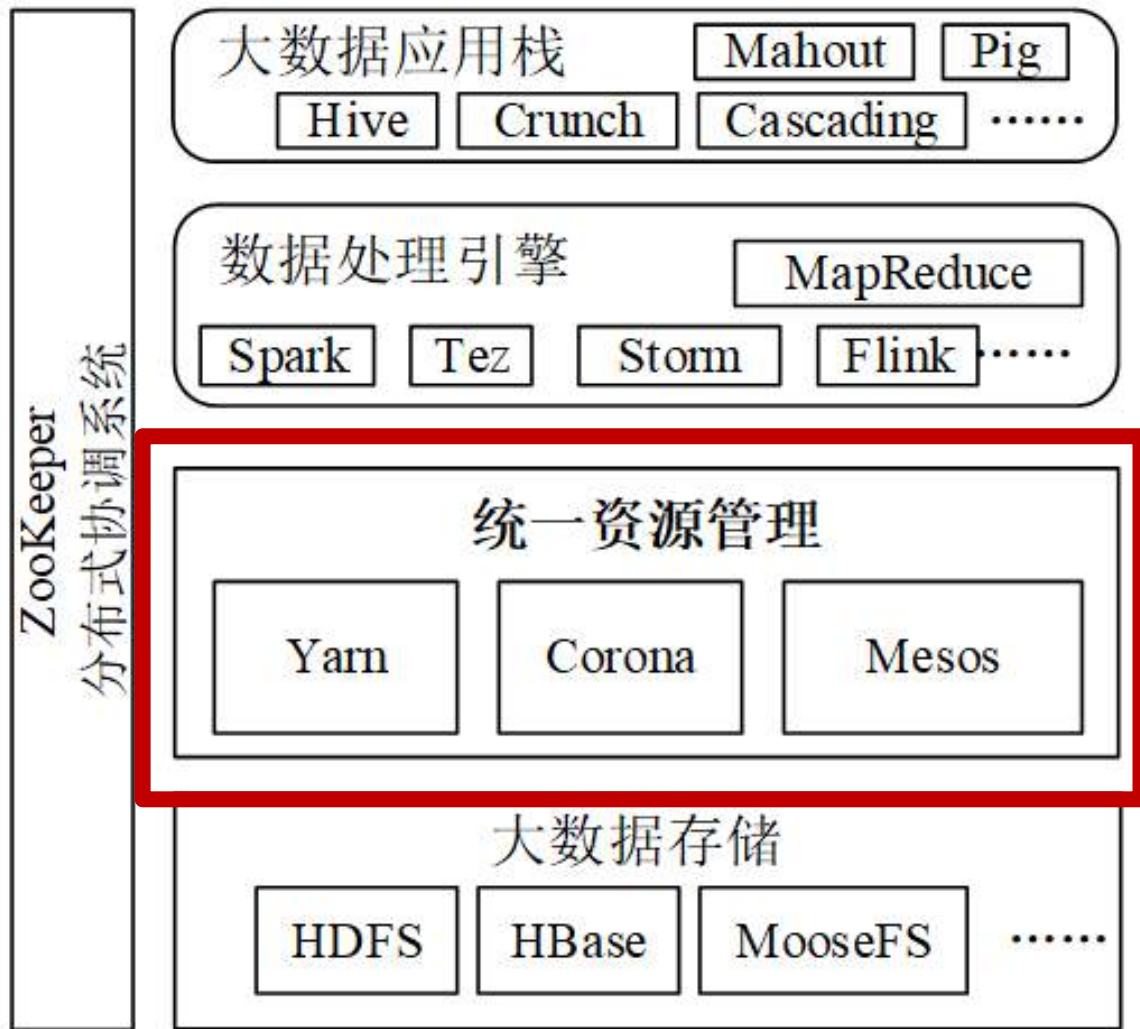
- Field Programmable Gate Array
- Integrated circuit that can be configured by the user to emulate any digital circuit as long as there are enough resources
- Array of configurable logic blocks (CLBs) connected through programmable interconnects (switch boxes)

Xilinx Spartan-3E Starter Kit



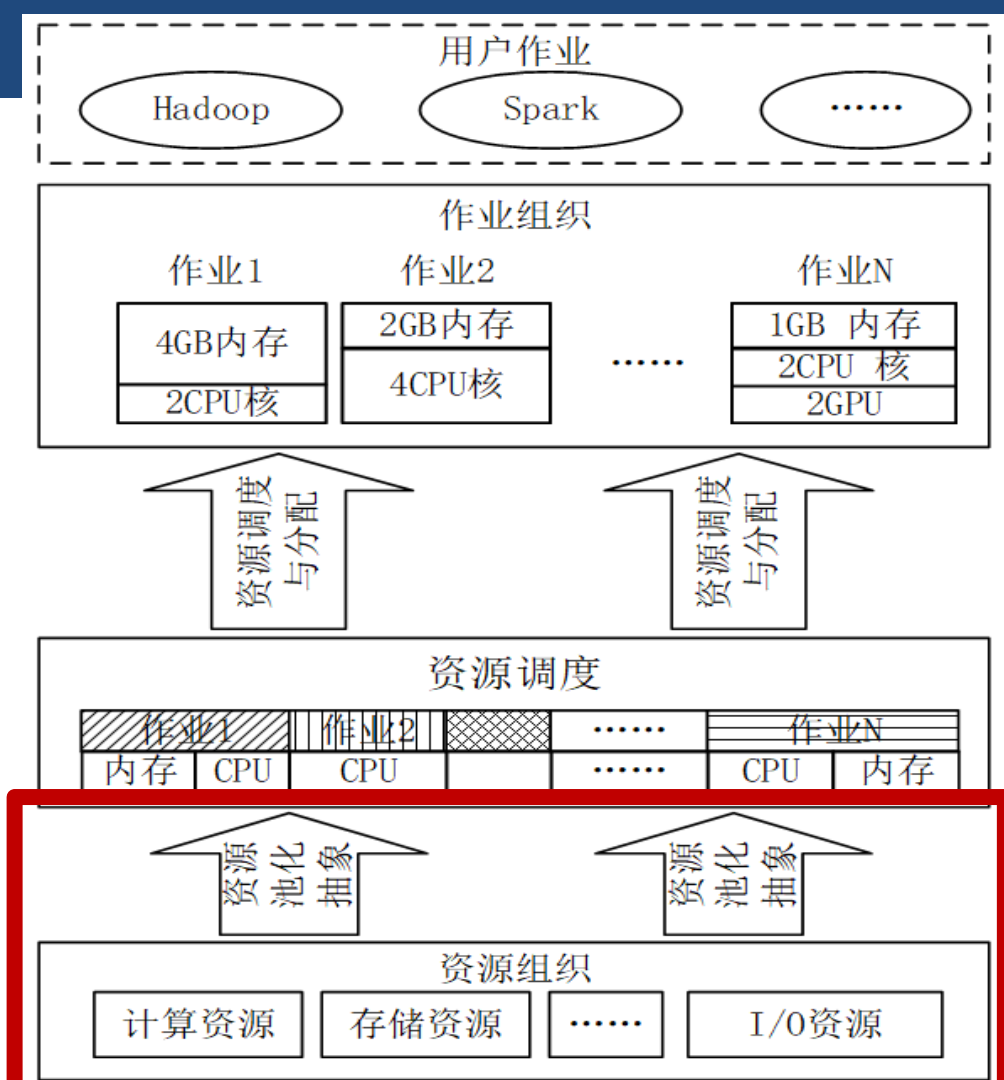
# 大数据通用处理平台

- 大数据应用和数据都具有**多样性**，为每一类大数据应用设计一套处理系统耗费巨大。
- 设计一个通用的大数据处理基础平台，**管理各种底层硬件设备和用户作业，进行资源调度以提高系统执行效率和吞吐量，同时支持多种大数据计算框架（例如批处理、流处理和图处理等）。**



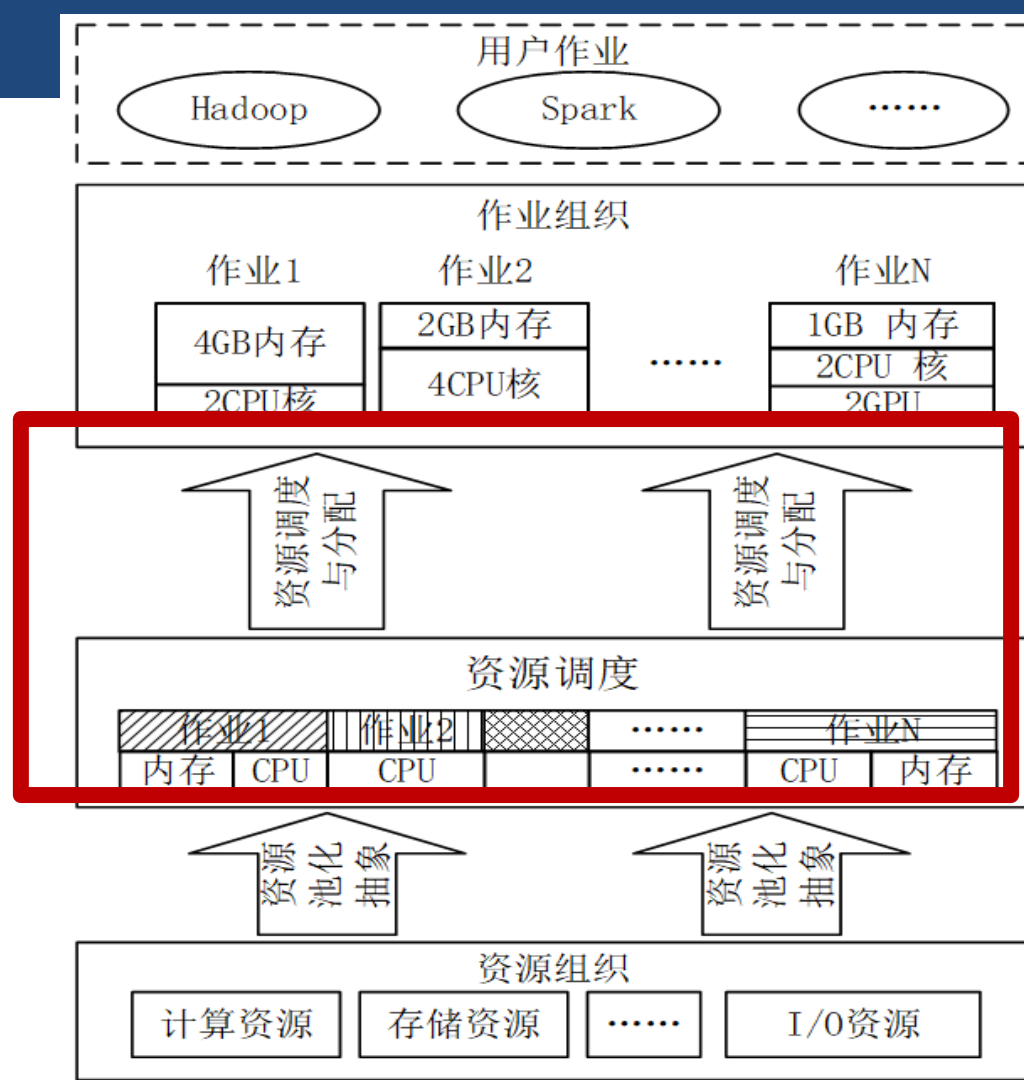
## 大数据通用处理平台

- 统一资源管理系统提供了**用户作业与零散的系统资源之间桥梁**，统一资源管理系统首先对系统软硬件资源进行**抽象和池化**，并根据不同的作业需求进行资源的分配与调度。



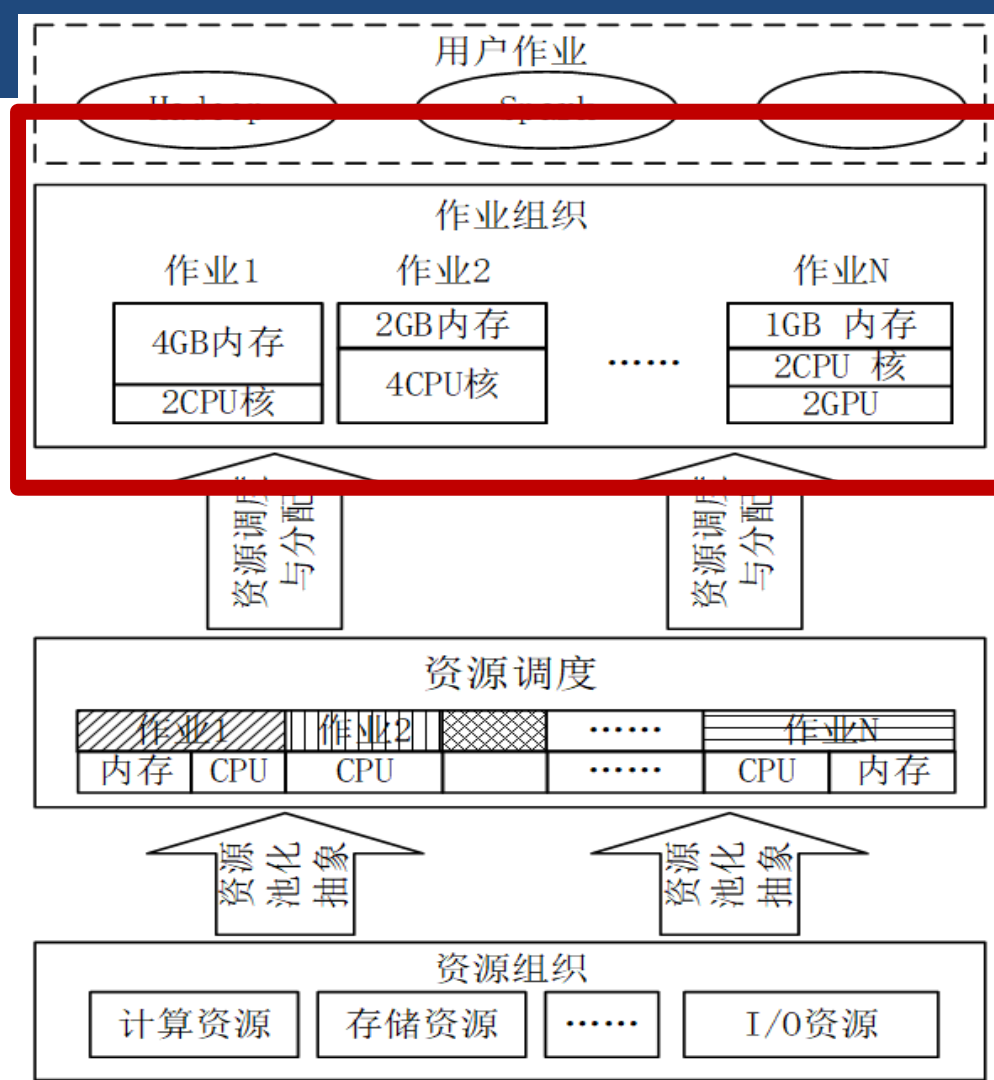
# 大数据通用处理平台

- 资源调度负责接收资源组织组件发送的资源状态信息并回收空闲资源，同时更新资源池中的相关信息。



# 大数据通用处理平台

- 作业组织组件负责**接收、组织并管理所有用户提交的作业**，并根据其特点进行分类存放。



## 5.6 小结

- ❖ 首先介绍了传统的集中式数据计算架构。
- ❖ 其次，重点介绍了超级计算机的发展历史和大数据特点。
- ❖ 然后，重点介绍了 MapReduce 和 Spark 两种分布式计算架构和近年来出现的针对流式数据的计算机架构。
- ❖ 最后，介绍对大数据计算进行加速的三种技术。