

## § Ch.1 大数据简介

大数据背景：架构的scale up&scale out，处理时间过长，成本过高，挑战：存储、计算、管理、分析

什么是数据：本质是生产资料和资产，不再是社会生产的副产物，而是可被二次乃至多次加工的原料，从中可以探索更大的价值，它变成了生产资料。结构化和半结构化

什么是大数据

大数据定义：一个超大的，难以用常规的数据库管理技术和工具处理的数据集

5V特征：大体量、多样性、时效性、准确性、大价值

大数据的类型3：

结构特征：（结构化，非结构/半结构化）

获取和处理方式：动态（流式/增量式/线上）/实时数据，静态（线下数据）/非实时数据

关联特征：无关联/简单关联数据（键值记录型数据），复杂关联数据（图数据）

大数据涉及的关键技术：5+3

海量数据存储技术(分布式文件系统)，实时处理（流计算引擎），高速传输（服务器/存储间高速通信），搜索（文本检索，智能搜索，实时搜索），数据分析（自然语言处理，文本情感分析，机器学习，聚类关联，数据模型）等

新平台技术：基于SQL，不基于SQL或mapreduce，数据流

不同范围的服务服务：数据入口/汇聚，数据平台，分析

新传输方案：传统的交付方式-单片或基于设备的解决方案，云：能够充分利用物理设施的弹性，以实现处理快速增长数据的能力，

挑战：存储量大、数据规模导致传统算法失效、复杂的数据关联性导致高复杂度计算

途径：新算法降低复杂度，降低数据尺度，并行化计算

1.何为大数据？

大数据指无法在可容忍的时间下，使用常规的数据库管理技术和软件方法完成存储、管理和处理任务的超大体量数据集。

2.大数据有哪几个特征？

大数据有5V特征，即Volume（大体量），Variety（多样性），Velocity（时效性），Veracity（准确性），Value（大价值）

3. 请解释数据和信息的区别和关系

关系：信息是数据的含义，数据是信息的载体。数据是反映客观事物属性的记录，是信息的具体表现形式；信息是经过加工处理之后的数据，需要经过数字化转变成数据才能存储和传输。

区别：数据是事实或观察的结果，是对客观事物的逻辑归纳，是用于表示客观事物的未经加工的原始素材。信息可以简单地理解为数据中包含的有用的内容，是剔除了无用冗余后的有用的数据。

4.请简述金融数据的特征。

金融数据一般具有“流数据”特征，需要在短时间内快速处理。具有逻辑关系紧密，处理实时性要求高，可展示性需求强等特征。

## Ch.2 并行计算和MPI基础编程

提高计算机硬件性能的主要手段：贯穿整个计算机技术发展的核心目标是提高计算性能，而之前是1、提高处理器字长、2、提高集成度、3、采用流水线等微体系结构技术实现指令级并行，4、提高处理器频率等等。

### § 为什么需要并行计算？

这些手段都遇到了瓶颈1、芯片集成度已经接近极限，不可能无限制提高。2、处理器的指令级并行度提升接近极限，复杂技术得以研究应用，但难以挖掘出更多潜能。3、处理器速度和存储器速度的差异越来越大。4、功耗和散热大幅增加超过芯片承受能力。

总而言之，单核处理器的性能提升已经接近极限，而应用领域的计算规模和复杂度仍在大幅提高，因此多核/众核并行计算技术的发展成为必然趋势。

### § 并行计算的分类6

系统类型，并行类型，计算特征，存储访问框架，数据和指令的处理结构，并行程序的设计模型/方法

按数据和指令处理结构（单/多指令单/多数据流）；按并行类型（位/指令/线程/数据级/任务级）；按存储访问构架（共享内存/分布共享存储体系结构/分布式存储）；按系统类型（多核并行计算系统/对称多处理器/大规模并行处理/集群/网格）；按计算特征（数据密集/计算密集/二者混合）；按并行程序设计模型/方法（共享内存变量/消息传递方式/mapreduce）

### § MPI并行程序设计的特点

在处理器之间以消息传递的方式进行数据通信和同步，以库函数形式为程序员提供一组易于使用的编程接口。

用户需要通过显式的发送和接受消息来实现处理机之间的数据交换，每个并行进程均有自己的独立的地址空间，访问不能直接进行，必须通过显式的消息传递来实现。并行计算粒度大，适合于大规模可扩展并行算法，要求用户很好的分解问题，组织不同进程间的数据交换。

### § MPI通信机制

1、点对点通信，同步/异步通信

2、节点集合通信：（同步，数据移动，数据归约）一对多广播通信，多节点计算同步控制，结果的规约计算reduce

3、用户自定义的复合数据类型传输

### § MPI的不足

1、无良好的数据和任务划分支持

2、缺少分布文件系统支持分布数据存储管理

3、通信开销大、当计算问题复杂、节点数量很大时、难以处理、性能大幅下降

4、没有节点失效恢复机制，一旦有节点失效，可能导致计算过程无效

5、缺少良好的构架支撑，程序员需要考虑以上所有的细节问题，程序设计较为复杂

## Ch.3 Google MapReduce的基本架构

MapReduce基本模型和处理思想：

不可分拆的计算任务或互相之间有依赖关系的数据无法分而治之

抽象模型Map和Reduce

map：对一组数据元素进行某种重复式的处理，获取感兴趣的中间信息( $k_1; v_1 \rightarrow [(k_2; v_2)]$ )，输入键值对( $k_1, v_1$ ) ( $txt1, "hello world!"$ )表示的数据，map函数处理这些键值对，并以另一种键值对形式输出处理的中间结果  $[(k_2; v_2)]$  ( $hello, 1)(world, 1)(!, 1)$

## 对中间结果排序和整理 (同步障)

reduce: 对map的中间结果进行某种进一步的结果整理( $k2; [v2]$ ) ->  $[(k3; v3)]$ ,  $[(k2; v2)]$ 将被进行合并处理将同样主键下的不同数值合并到一个列表[v2]中(good,1)(good,1)(is,1)(is,1)(is,1), 输入reduce函数, 进行进一步整理或处理, 输出  $[(k3; v3)]$  (good,2)(is,3)

reduce可以多个可以1个

## MapReduce伪代码 (实现Map和Reduce)

```
Class Mapper
    method map(String input_key, String input_value):
        // input_key: text document name
        // input_value: document contents
        for each word w in input_value:
            EmitIntermediate(w, "1");

Class Reducer
    method reduce(String output_key,
                  Iterator intermediate_values):
        // output_key: a word
        // output_values: a list of counts
        int result = 0;
        for each v in intermediate_values:
            result += ParseInt(v);
        Emit(AsString(result));
```

MapReduce提供一个统一的计算框架6:

- 1、计算任务的划分和调度
- 2、数据的分布存储和划分
- 3、处理数据与计算任务的同步
- 4、结果数据的收集整理 (sorting, combining对同一个map节点的中间结果的同样主键的数据合并(good,2)减少通信开销, partitioning把同样主键的数据发送到同一个reduce节点)
- 5、系统通信、负载平衡、计算性能优化处理
- 6、处理系统节点出错检测和失效恢复

MapReduce主要设计思想和特征:

- 1、Scale out, not scale up; 价格便宜易于扩展的大量低端商用服务器
- 2、失效被认为是常态;
- 3、把处理向数据迁移, 优先处理本地数据;
- 4、大数据集 批处理的并行计算, 顺序处理数据, 避免随机访问;
- 5、隐藏系统层细节;
- 6、平滑无缝的可扩展性: 数据扩展和系统规模扩展

google三驾马车: GFS, mapreduce , bigtable

## Ch.4 Google MapReduce的基本架构

### Google MapReduce

基本工作原理: 1、划分数据块以及相应的用户作业程序2、有一个负责调度的节点master以及工作节点mapper和reducer3、作业程序交给主节点4、主节点寻找和配备可用的mapper并传送程序5、reducer同理6、主节点启动mapper, mapper尽可能读取本地/本机架数据进行计算7、每个mapper处理数据、整理并将中间结果放在本地, 向主节点发送任务完成的信息和位置信息8、主节点等所有

mapper完成任务，启动reducer，reducer根据位置信息读取数据9、reducer计算结果并输出到结果文件

失效处理：主节点（周期性设置checkpoint，检查计算作业的完成情况，失效则从最近的检查点开始。）工作节点（主节点周期性发送检测命令，若无回应，则认为失效，终止其任务并将失效任务重新调度到其他节点上执行）

带宽优化：大量键值对传送给reduce时开销过大，combiner压缩同样主键的数据

## GFS

基本设计原则：Google GFS 是一个基于分布式集群的大型分布式文件系统，为 MapReduce 计算框架提供数据存储和数据可靠性支撑；GFS 是一个构建在分布节点本地文件系统之上的一个逻辑文件系统，它将数据存储在物理上分布的每个节点上，但通过 GFS 将整个数据形成一个逻辑上整体的文件。

1. 廉价本地磁盘分布存储：各节点本地分布式存储数据，优点是不需要采用价格较贵的集中式磁盘阵列，容量可随节点数增加自动增加。
2. 多数据自动备份解决可靠性：采用廉价的普通磁盘，把磁盘数据出错视为常态，用自动多数据备份存储解决数据存储可靠性问题。
3. 为上层的 Map Reduce 计算框架作为支撑：GFS 作为向上层 Map Reduce 执行框架的底层数据存储支撑，负责处理所有的数据自动存储和容错处理，因而上层框架不需要考虑底层的数据存储和数据容错问题。

基本工作原理：

- 1、在程序运行前，数据已经存储在 GFS 文件系统中，程序运行时应用程序会告诉 GFS master 所要访问的文件名或者数据块索引是什么。
- 2、GFS master 根据文件名和数据块索引在其文件目录空间中查找和定位该文件或数据块，找出数据块具体在哪些 ChunkServer 上，将这些位置信息回送给应用程序。
- 3、应用程序根据 GFS master 返回的具体 Chunk 数据块位置信息，直接访问相应的 Chunk Server
- 4、应用程序根据 GFS master 返回的具体 Chunk 数据块位置信息直接读取指定位置的数据进行计算处理。

特点：应用程序访问具体数据是不需要经过 GFS Master，因此，避免了 Master 成为访问瓶颈。由于一个大数据会存储在不同的 Chunk Server 中，应用程序可实现并发访问。

## BigTable基本工作原理

设计目标：为应用程序提供比单纯的文件系统更方便、更高层的数据操作能力。结构化数据的存储的访问管理。

存储多种数据，处理海量的服务请求，商用数据库无法适用，适用性、可扩展性、高吞吐量数据访问、高可用性和容错性，自动管理，简单性

## Data Model

见作业

基本构架：主服务器：执行元数据操作和负载平衡（新子表分配，子表服务器状态监控，负载均衡）

子表服务器：数据存储和访问操作（SSTable 对应一个 Chunk，可被多个子表共享）

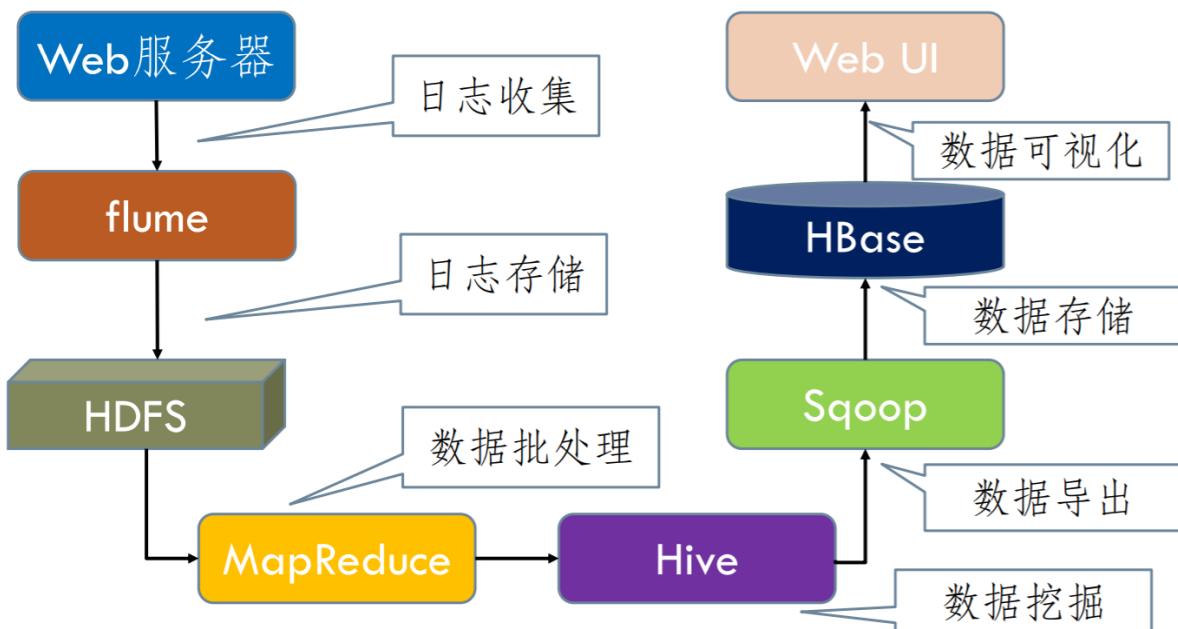
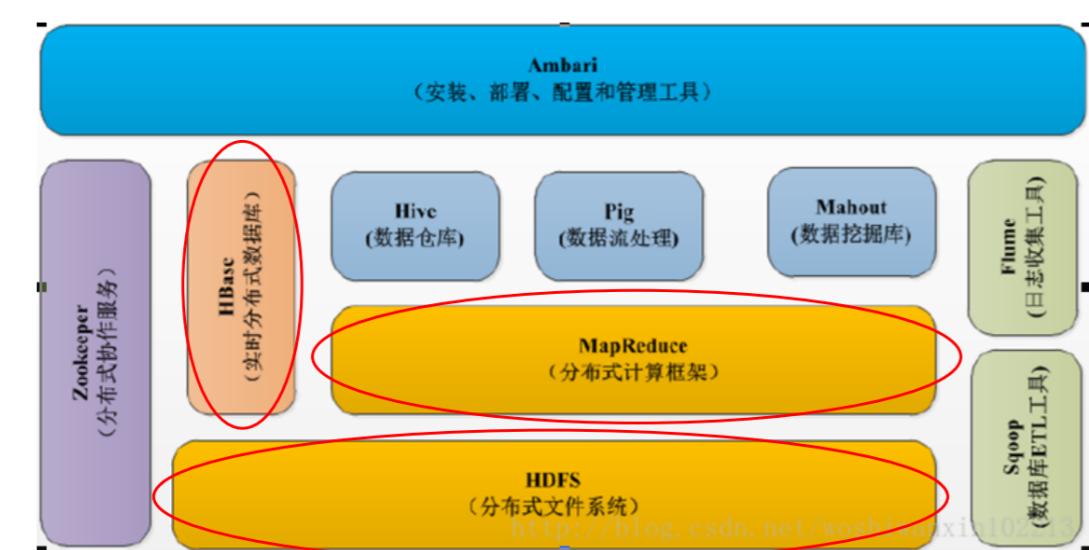
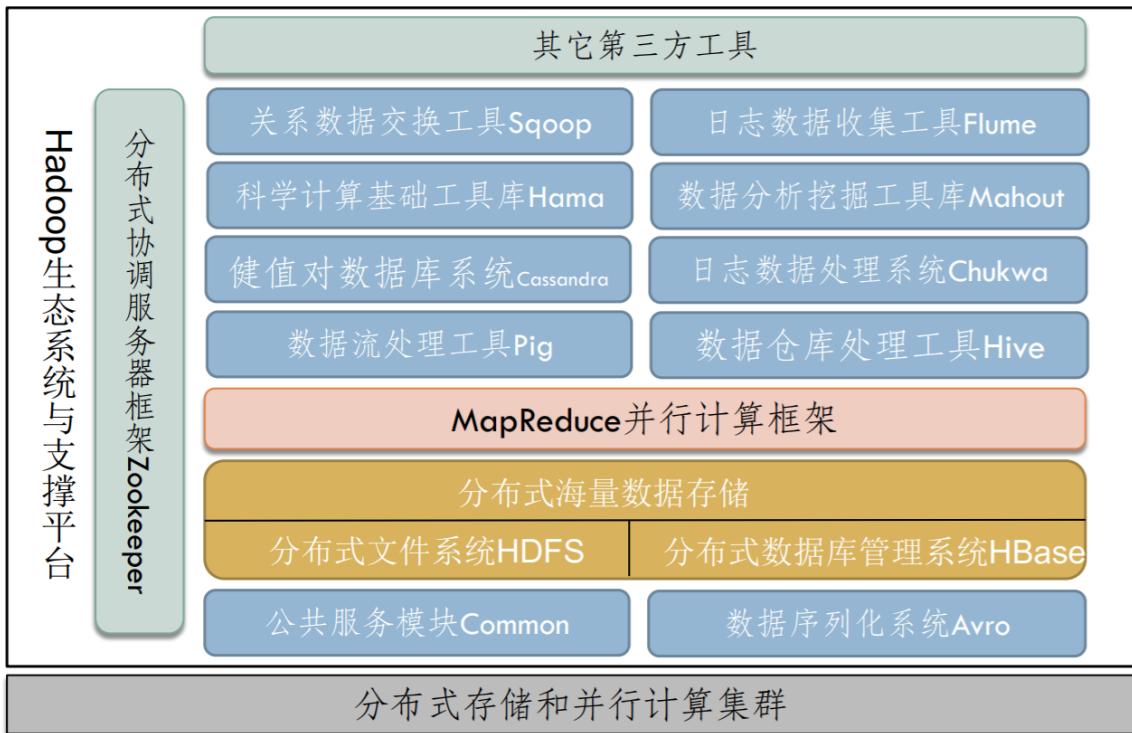
Chubby 服务器（分布式锁服务）：元数据存储及主服务器选择

GFS：子表数据和日志

GoogelWorkQueue：负责故障监控和处理

## Ch.4 Hadoop MapReduce的基本架构

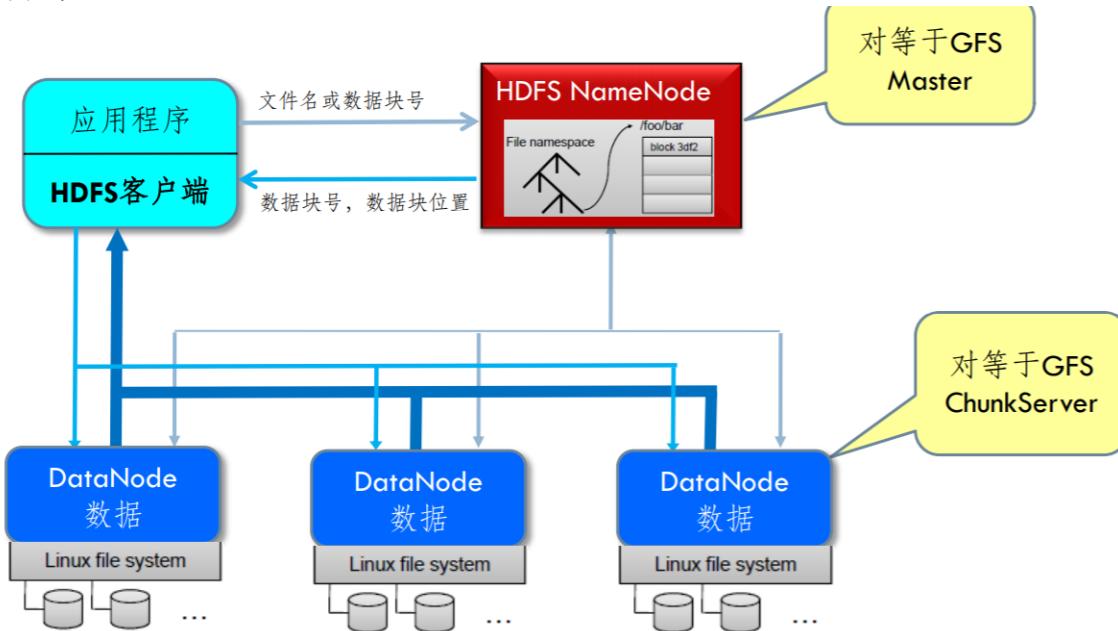
Hadoop生态系统



## HDFS

基本特征：1、模仿Google GFS 2、大量数据保存到大量节点，支持很大的单个文件 3、高可靠性高容错性（数据复制和恢复）4、提供对数据的快速访问 5、可扩展性 6、为mapreduce提供数据存储的支撑 7、顺序读出 8、支持一次写入多次读取，不支持数据更新 9、不进行本地缓存 10、多副本存储

基本构架：



数据分布设计及设计要点：

多副本数据块形式存储，按照快的方式随机选择存储节点，默认三个副本。

心跳：namenode不断检测datanode是否失效

设计要点：

命名空间，副本选择，安全模式（刚进入时等待每个datanode报告状态，推出安全模式才能复制副本），namenode有自己的fsImage和editlog，前者有自己的文件系统状态，后者是还没有更新的记录

纠删码：提高存储利用率，可靠性。对数据分块，计算校验数据，使各部分数据产生关联，当丢失一部分可通过剩余部分计算出。

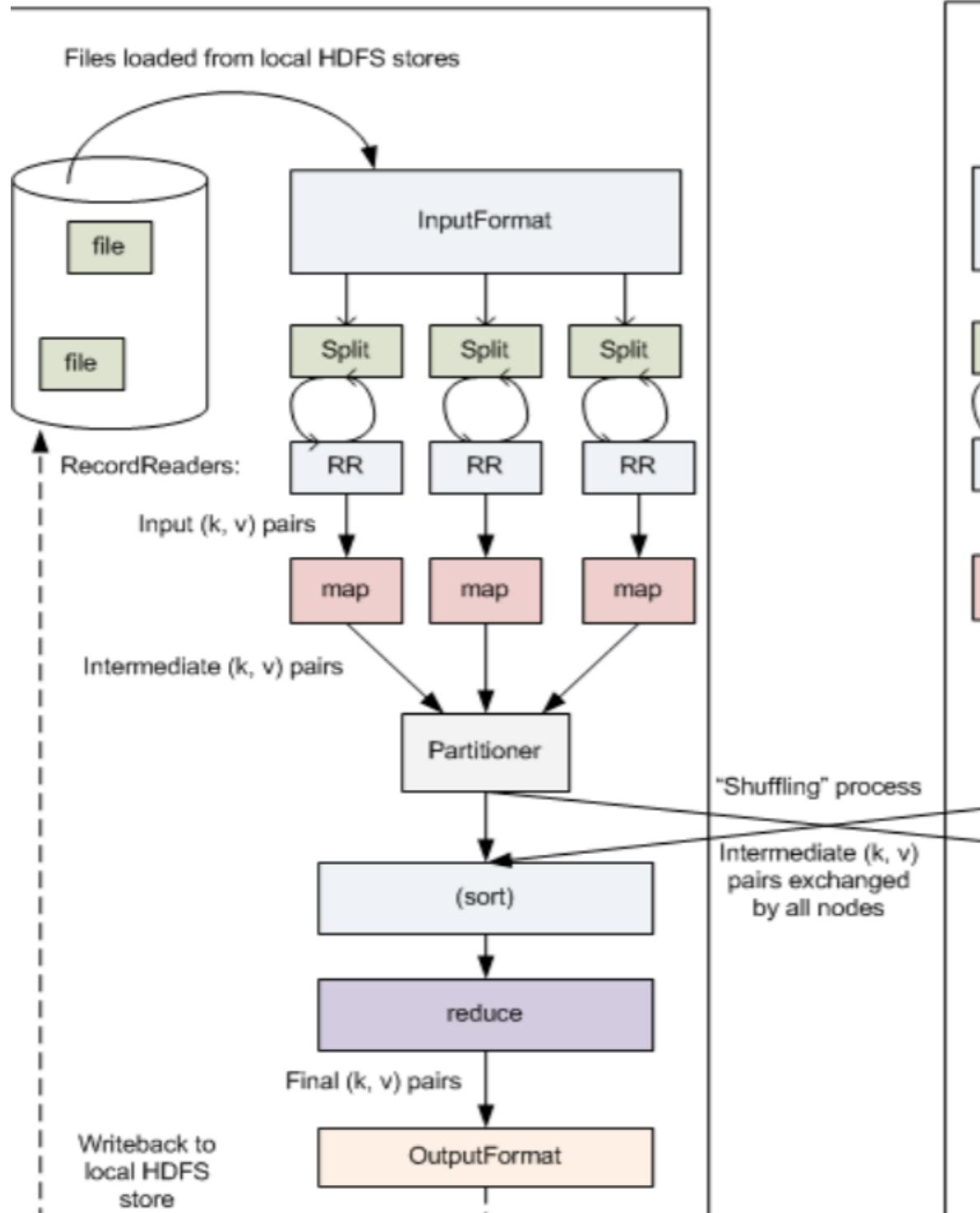
## Hadoop MapReduce

基本构架：一个job tracker 多个task tracker， job tracker资源管理和作业控制， tasktracker接受命令并执行

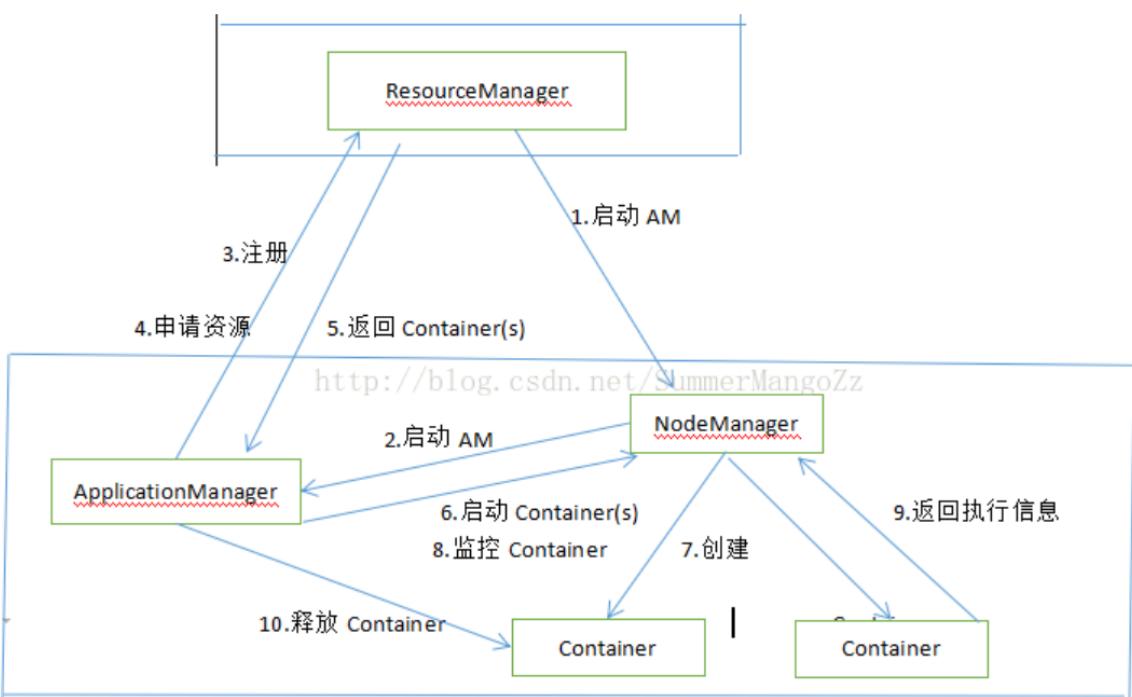
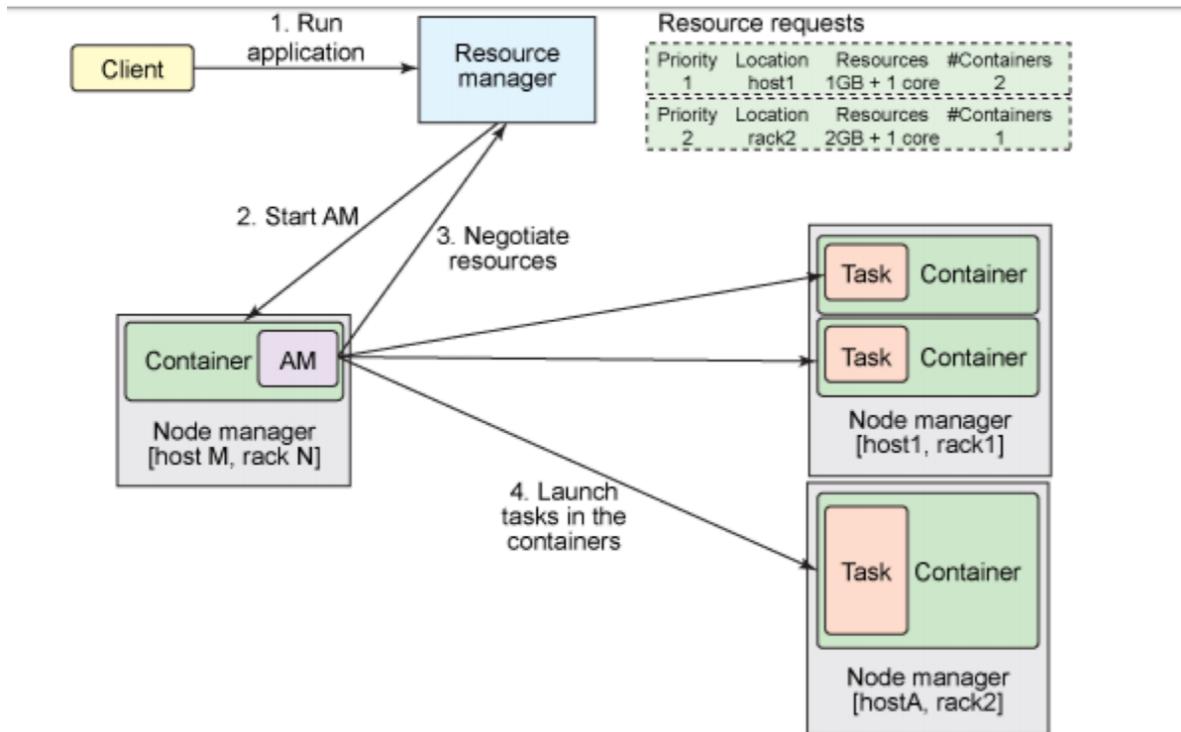
主要组件：

- 1、文件输入格式：input format定义文件如何分割和读取
- 2、输入数据分块：定义输入到每个mapper的任务输入数据，分割数据大小
- 3、数据记录读入record reader：定义了如何从数据上转化为key-value对的方法，如何读取数据
- 4、mapper
- 5、combiner：合并key相同的键值对
- 6、partitioner划分数据以及指定传输的reducer & shuffle
- 7、sort：在reduce前自动排序
- 8、reducer：
- 9、输出格式：outputformat

# Node 1



MapReduce v1.0 vs. YARN (v2.0): yarn统一的资源管理和调度（RM+AM），RMresourcemanager管理所有应用程序的计算资源的分配，全局集群资源管理器，AMapplicationmaste每个node专用的job tracker管理一个应用程序，NMnodemanager管理各个子结点代替task tracker，containers用application来替代以前的mapreduce作业，容器承载应用，为了以后的扩充



容错及优化：由Hadoop系统自己解决，把失败任务再次执行，task tracker将状态信息汇报给jobtracker后者决定重做的任务，自动重复执行取最快

HBase

逻辑模型：bigtable 的一个开源实现，是一个分布式多维表，表、行、列族、列、单元格、时间戳

物理存储：按列存储的稀疏行/列矩阵，物理存储格式上按逻辑模型中的行进行分割，并按照列族存储，值为空的列不予存储

## Ch.5/6 MapReduce基础编程

§ MapReduce流水线

§ WordCount

§ 矩阵乘法

§ 关系代数运算

§ 排序算法：totalorderpartitioner 预采样，排序后取N-1个分割点，(N个reducer),取partitioner得分割点，避免不同reducer之间的数据数量不平衡

二级排序：1、自定义key2、自定义partitioner类3、自定义key的比较类4、自定义分组比较类5、自定义mapper和reducer

§ 单词同现

```
1: class Mapper
2:   method Map(docid a, doc d)
3:     for all term w ∈ doc d do
4:       for all term u ∈ Neighbors(w) do
5:         //Emit count for each co-occurrence
          Emit(pair (w, u), count 1)
```

---

```
1: class Reducer
2:   method Reduce(pair p; counts [c1, c2,...])
3:     s ← 0
4:     for all count c 2 counts [c1, c2,...] do
5:       s ← s + c           //Sum co-occurrence counts
6:     Emit(pair p, count s)
```

§ 倒排索引： (bird:doc1,doc2)

## □ 带词频等属性的文档倒排算法

### □ Map和Reduce实现伪代码

1: **class Mapper**

2:   **procedure** Map(docid n, doc d)

3:        $H \leftarrow \text{new AssociativeArray}$

4:       **for all** term t  $\in$  doc d **do**

5:            $H\{t\} \leftarrow H\{t\} + 1$

6:       **for all** term t  $\in H$  **do**

7:           Emit(term t, posting <n, H{t}>)

-----  
1: **class Reducer**

2:   **procedure** Reduce(term t, postings [<n<sub>1</sub>, f<sub>1</sub>>, <n<sub>2</sub>, f<sub>2</sub>>...])

3:       P  $\leftarrow \text{new List}$

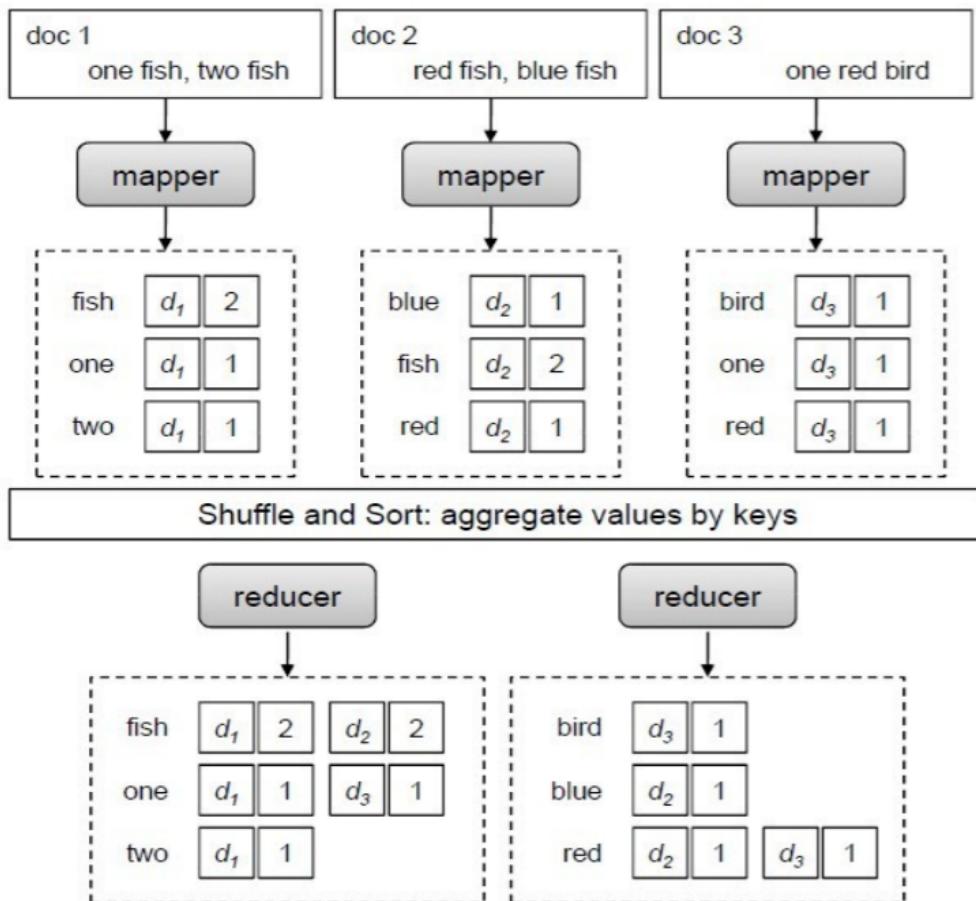
4:       **for all** posting <a, f>  $\in$  postings [<n<sub>1</sub>, f<sub>1</sub>>, <n<sub>2</sub>, f<sub>2</sub>>...] **do**

5:           Append(P, <a, f>)

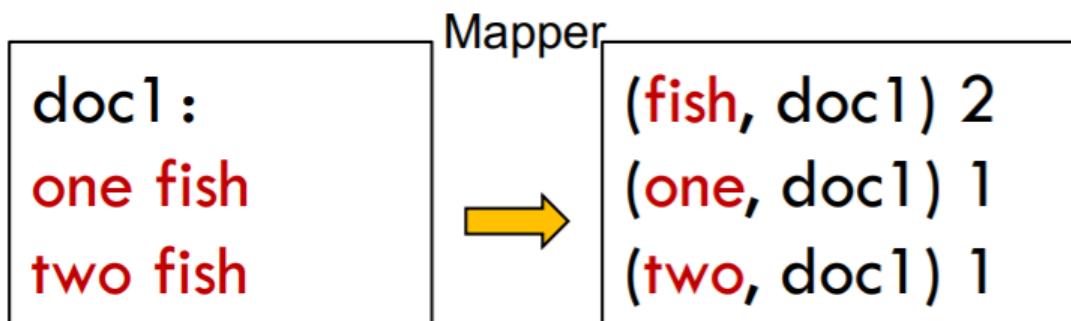
6:       Sort(P)

7:       Emit(term t; postings P)

## □ 带词频等属性的文档倒排算法



## □ 带词频属性的文档倒排算法



§ 专利文献数据分析:注意不同map节点发送来的数据, 可能有同样的关注的键

问题1: 如每碰到一个`<year, country>`, 即`emit(year, 1)`有问题吗?

答案: 有问题。因为可能会有从不同Map节点发来的同样的`<year, country>`, 因此会出现对同一国家的重复计数

问题2: Map结果(`<year, country>, [1, 1, 1, ...]`)数据通信量较大

解决办法: 实现一个Combiner将`[1, 1, 1, ...]`合并为1

## Ch.7 MapReduce高级编程

§ 复合键值对的使用: 进入reduce的键值对默认对key排序, 可以将需要排序的部分加入key, 形成复合键, 但需要自己实现partitionor保证同key的分到同reducer

## 键值合并

(a, b) → 1  
(a, c) → 2  
(a, d) → 5  
(a, e) → 3  
(a, f) → 2



a → { b: 1, c: 2, d: 5, e: 3, f: 2 }

- 然后，在Reduce阶段，把每个单词a的键值对(条)进行加：

$$\begin{array}{r} a \rightarrow \{ b: 1, \quad d: 5, e: 3 \} \\ + \quad a \rightarrow \{ b: 1, c: 2, d: 2, \quad f: 2 \} \\ \hline a \rightarrow \{ b: 2, c: 2, d: 7, e: 3, f: 2 \} \end{array}$$

## § 用户自定义数据类型

Class	Description
BooleanWritable	Wrapper for a standard Boolean variable
ByteWritable	Wrapper for a single byte
DoubleWritable	Wrapper for a Double
FloatWritable	Wrapper for a Float
IntWritable	Wrapper for a Integer
LongWritable	Wrapper for a Long
NullWritable	Placeholder when the key or value is not needed
Text	Wrapper to store text using the UTF-8 format

writable接口，如果作为key或需要比大小时需要实现writablecomparable接口

## § 用户自定义输入输出格式

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

RecordReader:	InputFormat	Description:
LineRecordReader	default reader for TextInputFormat	reads lines of text files
KeyValueLineRecordReader	default reader for KeyValueTextInputFormat	parses lines into key-val pairs
SequenceFileRecordReader	default reader for SequenceFileInput Format	User-defined methods to create keys and values

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 outputs

RecordWriter:	Description
LineRecordWriter	Default RecordWriter for TextOutputFormat writes lines in "key \t value" form

§ 用户自定义Partitioner和Combiner

§ 迭代完成MapReduce计算：需要用迭代方法求逼近结果，递归调用

$$PR(p_i) = \frac{1-d}{N} + d \sum_{p_j \in M(p_i)} \frac{PR(p_j)}{L(p_j)}$$

问题是在求解  $PR(p_i)$  时，需要递归调用  $PR(p_j)$ ，而  $PR(p_j)$  本身也是待求解的。因此，我们只能先给每个网页赋一个假定的  $PR$  值，如 0.5。但这样求出的  $PR(p_i)$  肯定不准确。然而，当用求出的  $PR$  值反复进行迭代计算时，会越来越趋近于最终的准确结果。因此，需要用迭代方法循环运行 MapReduce 过程，直至第  $n$  次迭代后的结果与第  $n-1$  次的结果小于某个指定的阈值时结束。

§ 链式 MapReduce 任务：jobconf 独立，输入输出文件路径 job3.waitForCompletion(true) 顺序化执行，

数据依赖关系：jobz.addDependingJob(jobx) 等到 x 执行完毕，Job Control 类

前处理后处理：加入map1,map2, reduce,map3,map4, chain mapper/reducer注意数据类型一致

§ 多数据源的连接

§ 全局参数/数据文件的传递：setInt设置传入的第几个参数是哪一个属性,addCacheFile添加文件到内存

§ 其它处理技术：查询任务相关信息，划分多个输出文件集合（按某一要求集合输出，比如输出到不同国家的文件目录），输入输出到关系数据库

## Ch.8 基于MapReduce的图算法

§ 图的表示：邻接矩阵， $M_{ij}$ 表示从*i*到*j*的有向边 优点：便于数学操作，遍历一行可以得到出度，一列入度，缺点：对于稀疏矩阵浪费内存

邻接表：只保存不为0的节点：1到2，4==>1: 2, 4 优点：更加紧凑，容易得到出度，缺点不易得到入度

§ PageRank的基本设计思想和设计原则：根据网页间超链接计算网页排名，搜索引擎用来标识网页的等级或重要性，1-10 10最受欢迎，优质网页：数量假设（很多网页连接到他）和质量假设（高质量网页连接到他）

T的连接指向A，T认为A比较重要，于是把T的一部分重要性得分赋予A， $PR(T)/L(T)$ ,L是出链数，PR是T的PageRank值，A的值为一系列类似比值的累加

排名泄露：一个网页没有外出链接则多次迭代后所有网页都趋于0。解决：去掉这种点，或添加一条边

排名下沉：一个网页没有入度，则它的分值在多次迭代后趋于0.解决：随机浏览模型

## 随机浏览模型的PageRank公式：

$$PR(p_i) = \frac{1-d}{N} + d \sum_{p_j \in M(p_i)} \frac{PR(p_j)}{L(p_j)}$$

随机浏览模型：1、更符合用户行为2、一定程度解决排名下沉3、保证PR存在唯一值

mapreduce: 1、graph builder: <URL,(pr,link)>初始值和出度列表，不需要reduce

2、pagerankiter: map:迭代计算pr值，直到收敛或达到预定次数。对link表中的所有网页计算  
 $val=PR/L.len,<u,value>$ 同时传递每个网页的链接信息<URL,link>

reduce:计算value和，乘上d，加上  $(1-d)/N$ ,得到PR值，输出<URL,(PR,link)>

```

1: class MAPPER
2:   method MAP(nid n, node N)
3:     p ← N.PAGERANK/|N.ADJACENCYLIST|
4:     EMIT(nid n, N)                                ▷ Pass along graph structure
5:     for all nodeid m ∈ N.ADJACENCYLIST do
6:       EMIT(nid m, p)                            ▷ Pass PageRank mass to neighbors

1: class REDUCER
2:   method REDUCE(nid m, [p1, p2, ...])
3:     M ← ∅
4:     for all p ∈ counts [p1, p2, ...] do
5:       if IsNODE(p) then
6:         M ← p                                ▷ Recover graph structure
7:       else
8:         s ← s + p                            ▷ Sums incoming PageRank contributions
9:     M.PAGERANK ← s
10:    EMIT(nid m, node M)

```

## PageRankIter伪代码

终止条件：分数不变/排序不变/次数达到多少

§ 单源最短路径的并行广度优先算法：

- 1、同一层之间的节点可并行执行
- 2、每一次mapreduce过程处理一层
- 3、初始化：起始点为0，其他为正无穷
- 4、迭代终止，没有距离为正无穷的点

```

1: class MAPPER
2:   method MAP(nid n, node N)
3:     d ← N.DISTANCE
4:     EMIT(nid n, N)                                ▷ Pass along graph structure
5:     for all nodeid m ∈ N.ADJACENCYLIST do
6:       EMIT(nid m, d + 1)                            ▷ Emit distances to reachable nodes

1: class REDUCER
2:   method REDUCE(nid m, [d1, d2, ...])
3:     dmin ← ∞
4:     M ← ∅
5:     for all d ∈ counts [d1, d2, ...] do
6:       if IsNODE(d) then
7:         M ← d                                ▷ Recover graph structure
8:       else if d < dmin then
9:         dmin ← d                            ▷ Look for shorter distance
10:        M.DISTANCE ← dmin                  ▷ Update shortest distance
11:        EMIT(nid m, node M)

```

## Ch.12 HBase基础原理与程序设计

§ CAP定理：分布式计算系统不可能同时满足三点：一致性（所有节点在同一时间有相同的数据）、可用性（保证每个请求不管成功还是失败都有相应）、分隔容忍分区容错性（系统中任何信息丢失或失败都不会影响系统的继续运作）

§ ACID vs. BASE

base: noSQL数据库对可用性和一致性的弱要求原则，无连接的软状态

## □ ACID

- 原子性(**A**tomicity)
- 一致性(**C**onsistency)
- 隔离性(**I**solation)
- 持久性(**D**urable)

## □ BASE

- 基本可用(**B**asic **A**vailable)
- 软状态/柔性事务 (**S**oft state)
- 最终一致性(**E**ventual **c**onsistency)

### § RDBMS vs. NoSQL

关系数据库不适合大表。结构不灵活可变，无停机时间的在线大表分区和动态扩容

非关系型数据库，可以存储超大规模数据，这些类型的数据存储不需要固定模式，无需多于操作就可以横向扩展

## □ RDBMS

- 高度组织化结构化数据
- 结构化查询语言**SQL**
- 数据和关系存储在单独的表中
- 数据操纵语言，数据定义语言
- 严格的一致性
- 基础事务

## □ NoSQL

- 代表着不仅仅是**SQL**
- 没有声明性查询语言
- 没有预定义的模式
- 键-值对存储，列存储，文档存储，图形数据库
- 最终一致性，而非**ACID**属性
- 非结构化和不可预知的数据
- CAP定理
- 高性能、高可用和可伸缩性

## □ 优点

- 高可扩展
- 分布式计算
- 低成本
- 架构灵活
- 半结构化数据
- 没有复杂的关系

## □ 缺点

- 没有标准化
- 有限的查询功能
- 最终一致是不直观的程序

### § HBase设计目标和功能特点

设计目标：针对HDFS缺少结构化半结构化数据存储访问能力的缺陷，提供一个分布式数据管理系统，解决大规模的结构化和半结构化数据存储访问问题。“Google BigTable的一个开源实现

- “ 提供基于列存储模式的大数据表管理能力
- “ 可存储管理数十亿以上的数据记录，每个记录可包含百万以上的数据列
- “ HBase试图提供随机和实时的数据读写访问能力
- “ 具有高可扩展性、高可用性、容错处理能力、负载平衡能力、实时数据查询能力

功能特点：2+2+4+2

#### 列式存储

- “ 表数据是稀疏的多维映射表
- “ 读写的严格一致性（区别于Cassandra的最终一致性）
- “ 提供很高的数据读写速度，为写数据进行了特别优化
- “ 良好的线形可扩展性
- “ 提供海量数据存储能力
- “ 数据会自动分片
- “ 具有自动的失效检测和恢复能力，保证数据不丢失
- “ 提供了方便的与HDFS和MapReduce集成的能力
- “ 提供Java API作为编程接口

NoSQL通常不担心冗余，推荐多行，时间按戳可以选择保存最新的几条或一段时间的几条

表设计：大表存储：查询简单，过期时有大量读写操作，活跃region在region server的分布不均匀

按时间分表：过期处理简单，活跃region可以均匀分布，但查询需要跨表性能较差

#### § 数据存储管理方法

大表分为多个子表region，子表存储在子表服务器，每个子表的数据区有很多个数据存储块store构成，每个store数据块又由存放在内存的memstore和存放在文件中的storefile构成。

当客户端需要进行数据更新时，先查到子表服务器，然后向子表提交数据更新请求。提交的数据并不直接存储到磁盘上的数据文件中，而是添加到一个基于内存的子表数据对象memStore中，当memStore中的数据达到一定大小时，系统将自动将数据写入到文件数据块StoreFile中。

- ¤ 每个文件数据块StoreFile最后都写入到底层基于HDFS的文件中。

访问：子表先查memstore再查磁盘上的storefile，store file有b+树结构支持快速查询。file定时压缩。  
子表可以合并和分割

主服务器HServer管理所有的子表服务器，维护状态，当新子表服务器注册时让其装载子表，连接超时时重启并移走任务，标注空闲

查询定位：所有元数据子表的元数据都保存在根子表中，主服务器扫描跟子表，得到所有元数据子表的位置，再扫描这些元数据子表得到寻找的子表位置。三级索引：根子表-用户表的元数据表-用户表

三层B+树保存子表位置：zookeeper-ROOT-META（保存region位置信息）-region

## § 基本操作和编程方法

- 查看有哪些表
  - list
- 创建表
  - create <table>, {NAME => <family>, VERSIONS => <VERSIONS>}
- 删除表
  - disable <table>
  - drop <table>
- 查看表结构
  - describe <table>
- 修改表结构
  - alter 't1', {NAME => 'f1'}, {NAME => 'f2', METHOD => 'delete'}

## Ch.13 Hive简介

基于Hadoop的数据仓库，对数据进行汇总、查询和分析

可扩展性，横向扩展集群不需要重启

延展性，自定函数

良好的容错性

不支持记录级别的增删改操作，查询耗时严重不是为在线事务设计的

## § RDBMS vs. Hive

对比项	Hive	RDBMS
查询语言	HQL	SQL
数据存储	HDFS	Raw Device or Local FS
执行器	MapReduce	Executor
数据插入	支持批量导入/单条插入	支持单条或者批量导入
数据操作	覆盖追加	行级更新删除
处理数据规模	大	小
执行延迟	高	低
分区	支持	支持
索引	0.8 版本之后加入简单索引	支持复杂的索引
扩展性	高（好）	有限（差）
数据加载模式	读时模式（快）	写时模式（慢）
应用场景	海量数据查询	实时查询

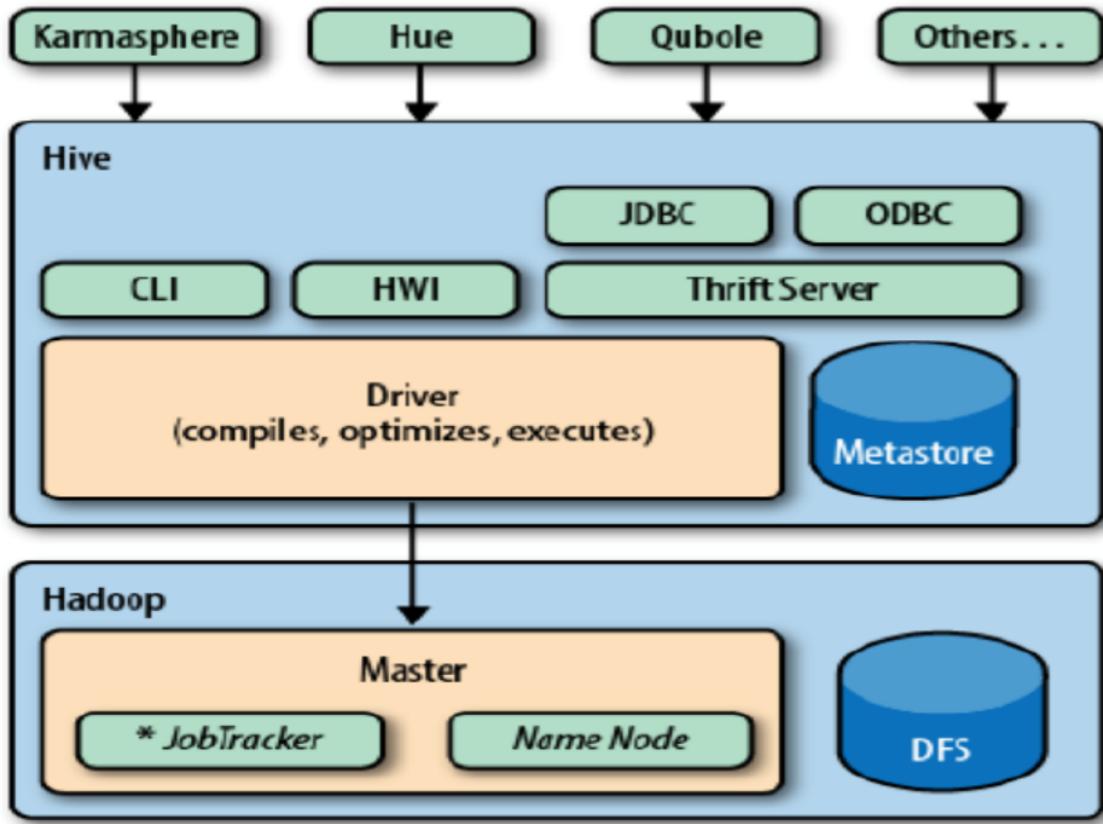
3+2+4+3

## § HBase vs. Hive：

hbase实时数据查询OLTP（联机事务处理），基于Hadoop的NoSQL数据库，主要适用于海量明细数据（十亿、百亿）的随机实时查询，如日志明细、交易清单、轨迹行为等

hive数据处理计算。OLAP（联机分析处理），基于Hadoop的数据仓库，严格来说，不是数据库，主要是让开发人员能够通过SQL来计算和处理HDFS上的结构化数据，适用于离线的批量数据计算。用途：日志分析、数据挖掘、文档索引、商业智能信息处理、即时查询及数据验证

#### § Hive的体系结构



driver:核心组件：对HQL语句进行解析编译（compiler）优化（optimizer），生成执行计划（executor）  
存在HDFS，调用底层的mapreduce

metastore:数据服务组件，存储元数据：操作的数据对象的格式信息，再HDFS中的存储位置以及其他用于数据转换的信息。元数据：database、表名、表的列及类型、存储空间、分区、表数据所在目录

CLI: 命令行接口

thrift server: 提供JDBC和ODBC接入的能力，进行可扩展且跨语言的服务的开发，

HWI: 通过网页的方式访问hive

§ Hive的数据模型：表：tables，列有类型，可能是复合类型list/map，

按一定规则划分：partition，如通过日期进行划分。

桶：buckets在一定范围内按hash方式划分

§ Hive QL:hive的数据查询语言，类似SQL，提供了数据查询语言与用户的接口，包括shell接口，可以进行用户交互以及网络接口和JDBC的接口，JDBC的接口可以直接用于编程

§ DDL: 数据定义语句，包括CREATE, ALTER, SHOW, DESCRIBE, DROP等

DML: 数据操作语句，包括LOAD DATA, INSERT。Hive的设计中没有考虑UPDATE操作。

QUERY: 数据查询语句，主要是SELECT语句。

## Ch.14 Spark简介

§ Spark特点：

基于内存计算思想提高计算性能，基于内存的弹性分布式数据集RDD，一组RDD形成可执行的有向无环图DAG构成灵活的计算流图，覆盖多种计算模式，运行速度快批处理、流式计算，在多种语言下可应用，集成了多种应用，SQL、dataframe、MLlib等用于复杂分析，可以接受多种数据源，在多种环境运行，单机，云，

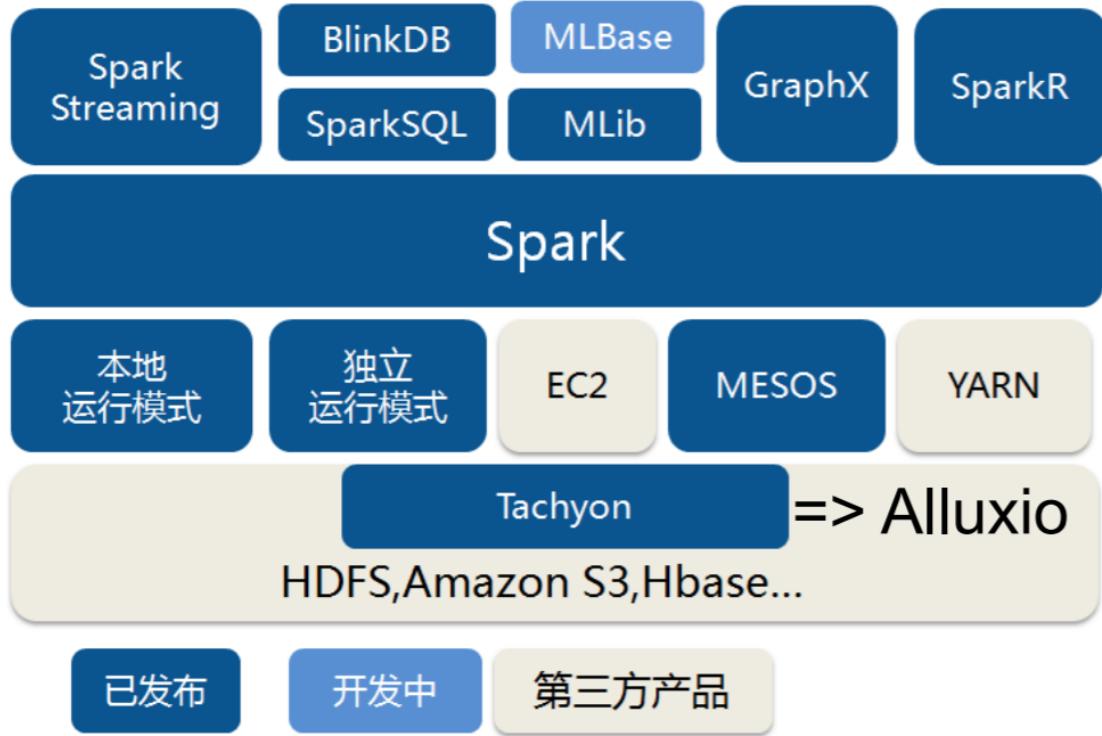
### § Spark vs. Hadoop

Hadoop	Spark
Distributed Storage + Distributed Compute	Distributed Compute Only
MapReduce framework	Generalized computation
Usually data on disk (HDFS)	On disk / <b>in memory</b>
Not ideal for iterative work	Great at Iterative workloads (machine learning ..etc)
Batch process	- Up to 2x - 10x faster for data on disk - Up to 100x faster for data in memory
	<b>Compact code</b> <b>Java, Python, Scala supported</b>

spark: 1、中间数据放内存2、迭代运算效率高3、容错性高RDDcheckpoint4、更加通用，不只有mapreduce5、提供多种数据集操作类型，和通信模型6、多种语言7、分布式运算

Hadoop: 1、中间结果落地，保存在磁盘2、只有mapreduce操作3、通信模型只有shuffle4、分布式存储和运算

### § Spark生态圈



RDDs, spark SQL, sparkstreaming,MLlib, graphX,

DAG分布式并行计算框架, cache机制支持多次迭代运算或数据共享

RDD分布在节点中的只读对象的集合，集合是弹性的，如果数据集中一部分丢失，可以通过血统对他们重建

partition就近读取分布式文件系统中的数据块到各个节点内存中进行计算

## § Spark的基本构架和组件

基本构架：Spark 架构采用了分布式计算中的Master-Slave模型。Master 是对应集群中的含有Master 进程的节点，Slave 是集群中含有Worker 进程的节点。Master 作为整个集群的控制器，负责整个集群的正常运行；Worker 相当于计算节点，接收主节点命令与进行状态汇报；Executor 负责任务的执行；Client 作为用户的客户端负责提交应用，Driver 负责控制一个应用的执行。Spark 集群部署后，需要在主节点和从节点分别启动Master 进程和Worker 进程，对整个集群进行控制。在一个Spark 应用的执行过程中，Driver 和Worker 是两个重要角色。Driver 程序是应用逻辑执行的起点，负责作业的调度，即Task 任务的分发，而多个Worker 用来管理计算节点和创建Executor 并行处理任务。在执行阶段，Driver 会将Task 和Task 所依赖的file 和jar 序列化后传递给对应的Worker 机器，同时Executor 对相应数据分区的任务进行处理。

组件：

Master node：是集群部署时的概念，是整个集群的控制器，负责整个集群的正常运行，管理Worker node。

Worker node：是计算节点，接收主节点命令与进行状态汇报。

Executors：每个Worker 上有一个Executor，负责完成Task 程序的执行。

Spark 集群部署后，需要在主从节点启动Master 进程和Worker 进程，对整个集群进行控制。

在Spark 应用程序执行过程中，Driver 和Worker 扮演着最重要的角色。

Driver：是应用执行起点，负责作业调度。

Worker：管理计算节点及创建并行处理任务。

Cache：存储中间结果等。

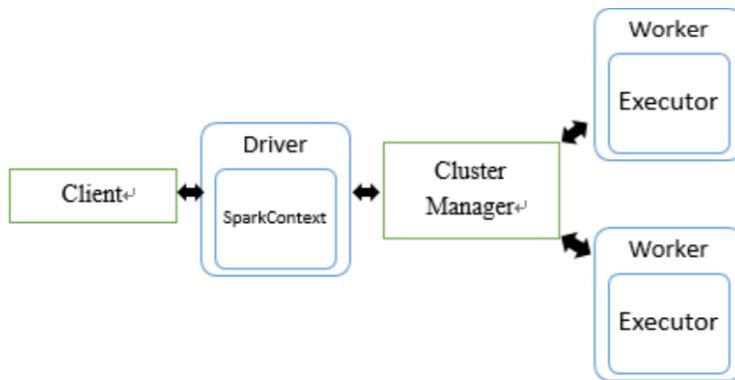
Input Data：为输入数据。

action 催生 job，job 拆分成多组 task，每组是一个 stage，一个 task 由一个 executor 执行

一组 RDD 形成有向无环图，

同一个 executor 中多个 task 共享资源 cache

worker node 上可以有多个 worker，一个 worker 一个 executor



- **Client**: 用户的客户端
- **Driver**: 负责控制一个应用的执行
- **Cluster Manager**: 在集群上获取资源的外部服务，例如 **Standalone、Mesos、YARN、Kubernetes**
- **Executor**: 负责执行Task任务

#### § Spark的技术特点

**RDD**: Spark提出的弹性分布式数据集，是Spark最核心的分布式数据抽象，Spark的很多特性都和RDD密不可分。

**Transformation & Action**: Spark通过RDD的两种不同类型的运算实现了惰性计算，即在RDD的Transformation运算时，Spark并没有进行作业的提交；而在RDD的Action操作时才会触发SparkContext提交作业。

**Lineage**: 为了保证RDD中数据的鲁棒性，Spark系统通过血统关系（lineage）来记录一个RDD是如何通过其他一个或者多个父类RDD转变过来的，当这个RDD的数据丢失时，Spark可以通过它父类的RDD重新计算。

**Spark调度**: Spark采用了事件驱动的Scala库类Akka来完成任务的启动，通过复用线程池的方式来取代MapReduce进程或者线程启动和切换的开销。

**API**: Spark使用Scala语言进行开发，并且默认Scala作为其编程语言。因此，编写Spark程序比MapReduce程序要简洁得多。同时，Spark系统也支持Java、Python语言进行开发。

**Spark生态**: Spark SQL、Spark Streaming、GraphX等为Spark的应用提供了丰富的场景和模型，适合应用于不同的计算模式和计算任务。

**Spark部署**: Spark拥有Standalone、Mesos、YARN、K8S等多种部署方式，可以部署在多种底层平台上。

Spark适用于需要多次操作特定数据集的应用场合。需要反复操作的次数越多，所需读取的数据量越大，受益越大，数据量小但是计算密集度较大的场合，受益就相对较小。由于RDD的特性，Spark不适用那种异步细粒度更新状态的应用，例如web服务的存储或者是增量的web爬虫和索引。就是对于那种增量修改应用模型不适合，适用于数据量不是特别大，但是要求实时统计分析需求的任务。

综上所述，Spark是一种为大规模数据处理而设计的快速通用的分布式计算引擎，适合于完成一些迭代式、关系查询、流式处理等计算密集型任务。

## Ch.15 Spark基础编程

#### § Spark安装与运行

#### § Spark编程模型

RDD的操作：创建：读文件，数组并行化，其他RDD的transformation

转换transformation：定义新的RDDmap,filter,flatmap,union,distinct,groupByKey,join

动作action: 计算值返回结果rreduce, collect, count, first, take, save..., foreach, countByKey

RDD的容错: 血统记录从父类的变换, 检查点

RDD的依赖: 窄依赖: 父RDD中的一个partition最多被子RDD中一个partition依赖:  
map, filter, union, join1

宽依赖: 一个partition被子的多个依赖reduceByKey, join2

划分阶段, 进行调度

RDD的持久化: 1、未序列化的java对象存在内存中, 性能最优2、序列化的数据存在内存中, 使用时需要反序列化3、磁盘存储, 重新计算RDD开销大

在action时将rdd放入缓存, 第二次action直接读取缓存

RDD: 分区, 描述血统的依赖, 父类做的函数, 元数据 (分区模式和数据存放的位置)

§ Spark编程实例

WordCount

K-Means每次操作都是在内存中完成, 迭代操作都在一个job中完成, 没有重启多次job的开销

## § Ch.20 云计算简介

§ 什么是云计算? 云计算解决什么主要问题?

通过集中式远程计算资源池, 以按需分配的方式为终端用户提供强大廉价的计算能力, 工业化部署, 商业化运作的大规模计算能力, 资源池物理上对用户透明,

把计算能力变成像水电等公用服务一样, 随用随取按需取用, 解决大粒度系统日益增多, 系统规模日益扩大; 小粒度系统资源重复, 无法共享。1) 为小粒度应用提供一个集中管理的巨大的计算资源池, 提供巨大的计算资源和能力资源共享; 2) 为大粒度应用提供大规模计算能力

§ 云计算的主要特点: 超大规模, 高可靠性, 高伸缩性, 虚拟化, 通用性, 按需取用, 极为廉价

§ 云计算的分类

1按服务层面的分类: IaaS基础设施作为服务, 硬件设备, PaaS平台作为服务, 程序运行环境, SaaS软件作为服务

2按系统类型的分类: 公用云: 面向大众, 缺少隐私, 私有云: 面向应用行业/组织内部, 私密性好, 社区云: 面向社团组织用户, 混合云: 有以上两种以上云计算类型混合

§ 云计算的关键技术: 虚拟化, 服务器、存储、网络桌面、

云计算构架系统软硬件构架、资源调度: 物理或虚拟资源、

并行计算: 大数据和复杂计算应用, 数据和任务切分, 算法设计,

大数据存储: 分布式存储, 访问共享数据备份,

云安全访问安全数据安全,

云计算应用面向各个行业不同形式,

云计算中心的节能和散热

§ 怎样才算是云计算系统?

资源虚拟化和弹性调度解决小粒度应用资源共享 (至少)

大数据存储处理和并行计算服务提供大粒度应用计算能力

§ 容器云:

以容器为资源分割和调度的基本单位, 封装整个软件运行时的环境, 为开发者和系统管理员提供用于构建发布和运行分布式应用的平台, 敏捷应用程序的创建与部署, 持续开发、继承和部署, 关注开发和运维的分离, 可观察性, 跨开发, 测试和生产环境一致性paas

§ 云原生：云原生是一种构建和运行应用程序的方法，是一套技术体系和方法论。云原生（Cloud Native）是一个组合词，Cloud + Native。Cloud表示应用程序位于云中，而不是传统的数据中心；Native表示应用程序从设计之初即考虑到云的环境，原生为云而设计，在云上以最佳姿势运行，充分利用和发挥云平台的弹性+分布式优势。属性：容器化封装，自动化管理，面向微服务松耦合提高敏捷性和可维护性四要素：持续交付，devops,微服务，容器

§ 数据湖：数据湖是一类存储数据自然/原始格式的系统或存储，通常是对象块或者文件，通常是全量数据的单一存储，包括原数据拷贝和为了各类任务产生的转换数据