# 第四讲　批处理系统MapReduce

徐 辰
cxu@dase.ecnu.edu.cn
華東師範大學

## Hadoop简介

2

- Hadoop是Apache软件基金会旗下的一个开源分布式计算平台，为用户提供了系统底层细节透明的分布式基础架构
- Hadoop是基于Java语言开发的，具有很好的跨平台特性，并且可以部署在廉价的计算机集群中
- Hadoop的核心是分布式文件系统HDFS（Hadoop Distributed File System）和MapReduce
- Hadoop被公认为行业大数据标准开源软件，在分布式环境下提供了海量数据的处理能力

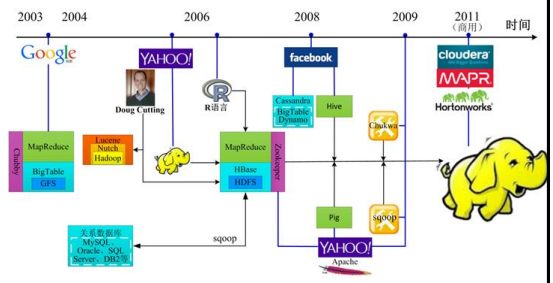
## Hadoop发展简史

3

- Hadoop最初是由Apache Lucene项目的创始人Doug Cutting开发的文本搜索库。Hadoop源自始于2002年的Apache Nutch项目——一个开源的网络搜索引擎并且也是Lucene项目的一部分
- 2004年，Nutch项目也模仿GFS开发了自己的分布式文件系统NDFS（Nutch Distributed File System），也就是HDFS的前身
- 2004年，谷歌公司又发表了另一篇具有深远影响的论文，阐述了MapReduce分布式编程思想
- 2005年，Nutch开源实现了谷歌的MapReduce
- 2006年2月，Nutch中的NDFS和MapReduce开始独立出来，成为Lucene项目的一个子项目，称为Hadoop，同时，Doug Cutting加盟雅虎
- 2008年1月，Hadoop正式成为Apache顶级项目，Hadoop也逐渐开始被雅虎之外的其他公司使用
- 2008年4月，Hadoop打破世界纪录，成为最快排序1TB数据的系统，它采用一个由910个节点构成的集群进行运算，排序时间只用了209秒
- 在2009年5月，Hadoop更是把1TB数据排序时间缩短到62秒。Hadoop从此名声大震，迅速发展成为大数据时代最具影响力的开源分布式开发平台，并成为事实上的大数据处理标准

## Hadoop生态圈发展路线

4




## MapReduce在Hadoop中位置

5

**MapReduce:**
是google MapReduce的开源实现。是对并行计算的封装，使用户通过一些简单的逻辑即可完成复杂的并行计算。将一个大的运算任务分解到集群每个节点上，充分运用集群资源，缩短运行时间。

**Yarn:**
Yet Another Resource Negotiator(另一种资源协商)，是一个资源调度框架

Hive
HBase
Map/Reduce
Yarn
DFS

**Hive:**数据分析工具，提供类SQL语言到MapReduce的转化能力。支持JDBC接口.
**HBase:**是google bigtable的开源实现，
Master/Slave架构下的列存储系统
稀疏的，长期存储的、多维度的、排序的映射表
表的索引是行关键字、列关键字和时间戳。每一个索引和值都是字符串。没有其他数据类型。
采用hdfs做为底层存储，支持MapReduce

**DFS：分布式文件系统，支持HDFS，S3。**

## 大纲

6

- 设计思想
  - MPI与MapReduce
  - 数据模型
  - 计算模型
- 体系架构
- 工作原理
- 容错机制

## MPI(Message Passing Interface)编程简介

7

- MPI是一个信息传递应用程序接口，包括协议和和语义说明
- 常用接口
  - MPI_Init(…) 并行环境初始化
  - MPI_Comm_size(…) 获得进程个数 size
  - MPI_Comm_rank(…) 获取进程的rank值
  - MPI_Send(…) 发送消息
  - MPI_Recv(…) 获取消息
  - MPI_Finalize() 退出MPI环境

## MPI编程举例

8

```
#include <stdio.h>
#include <string.h>
#include "mpi.h"
void main(int argc, char* argv[])
{
    int numprocs, myid, source;
    MPI_Status status;
    char message[100];
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &myid);
    MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
    if (myid != 0) {  //非0号进程发送消息
        strcpy(message, "Hello World!");
        MPI_Send(message, strlen(message) + 1, MPI_CHAR, 0, 99,
            MPI_COMM_WORLD);
    }
    else {   // myid == 0, 即0号进程接收消息
        for (source = 1; source < numprocs; source++) {
            MPI_Recv(message, 100, MPI_CHAR, source, 99,
                MPI_COMM_WORLD, &status);
            printf("接收到第%d号进程发送的消息：%s\n", source, message);
        }
    }
    MPI_Finalize();
} /* end main */
```

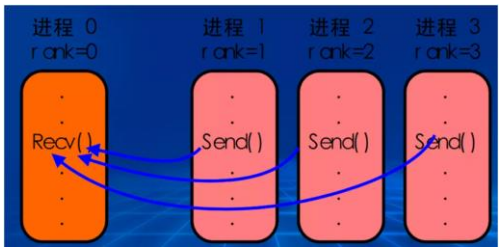
## 运行MPI程序

9

```
PS F:\并行计算学习\pi\x64\Debug> mpiexec.exe -n 4 .\send_recv.exe
接收到第1号进程发送的消息，Hello World!
接收到第2号进程发送的消息，Hello World!
接收到第3号进程发送的消息，Hello World!
PS F:\并行计算学习\pi\x64\Debug> mpiexec.exe -n 8 .\send_recv.exe
接收到第1号进程发送的消息，Hello World!
接收到第2号进程发送的消息，Hello World!
接收到第3号进程发送的消息，Hello World!
接收到第4号进程发送的消息，Hello World!
接收到第5号进程发送的消息，Hello World!
接收到第6号进程发送的消息，Hello World!
接收到第7号进程发送的消息，Hello World!
PS F:\并行计算学习\pi\x64\Debug>
```

## MPI程序工作示意

10




## MPI的局限性

11

- 从用户编程的角度来看，程序员需要考虑到进程之间的并行问题，并且进程之间的通信需要用户在程序中显式地表达，这无疑增加了程序员编程的复杂性。
- 从系统实现的角度来看，MPI程序是以多进程方式运行的。如果在运行过程中某一进程因故障导致崩溃，那么除非用户在编写程序时添加了故障恢复的功能，否则MPI编程框架本身并不能提供容错能力。

## 大纲

12

- 设计思想
  - MPI与MapReduce
  - 数据模型
  - 计算模型
- 体系架构
- 工作原理
- 容错机制

## 数据模型

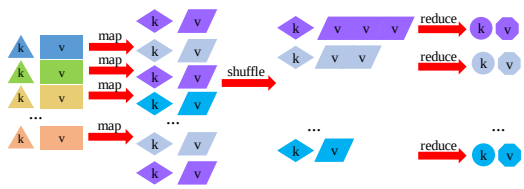
13

- 将数据抽象为一系列键值对，在处理过程中对键值对进行转换

| 函数 | 输入 | 输出 | 说明 |
|---|---|---|---|
| Map | $<k_1,v_1>$<br>如：<br><行号,"a b c"> | List($<k_2,v_2>$)<br>如：<br><"a",1><br><"b",1><br><"c",1> | 1.将小数据集进一步解析成一批<key,value>对，输入Map函数中进行处理<br>2.每一个输入的$<k_1,v_1>$会输出一批$<k_2,v_2>$。$<k_2,v_2>$是计算的中间结果 |
| Reduce | $<k_2,$List($v_2$)$>$<br>如：<br><"a",<1,1,1>> | $<k_3,v_3>$<br><"a",3> | 输入的中间结果$<k_2,$List($v_2$)$>$中的List($v_2$)表示是一批属于同一个$k_2$的value |

## 数据模型

14




## 大纲

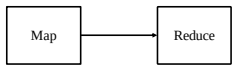
15

- 设计思想
  - MPI与MapReduce
  - 数据模型
  - 计算模型
- 体系架构
- 工作原理
- 容错机制

## 逻辑计算模型

16

- MapReduce将复杂的、运行于大规模集群上的并行计算过程高度地抽象为Map和Reduce两个过程
- 有向无环图（Directed Acyclic Graph, DAG)




## MapReduce编程范型

17

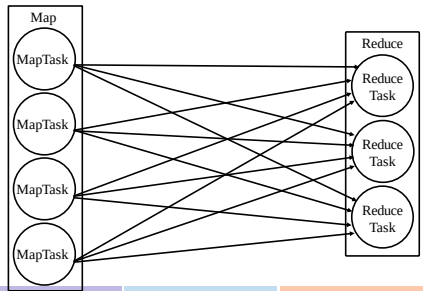
- 编程容易，不需要掌握分布式并行编程细节，很容易把程序运行在分布式系统上

```
Class X {
    map() {              //map函数的实现
      …
    }

    reduce() {           //reduce函数的实现

    }

    main(){
       Job job = …       //定义分布式作业
       job.config =      //作业参数设置
    }
}
```

## 物理计算模型

18

- 采用"分而治之"策略，由多个任务并行处理



## 大纲

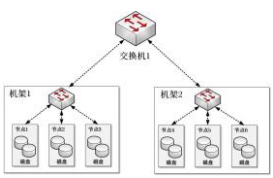
19

- 设计思想
- 体系架构
  - 架构图
  - 应用程序执行流程
- 工作原理
- 容错机制

## 集群网络拓扑

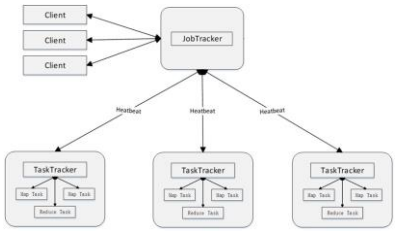
20

- 机架（Rack）间通信
- 机架内通信



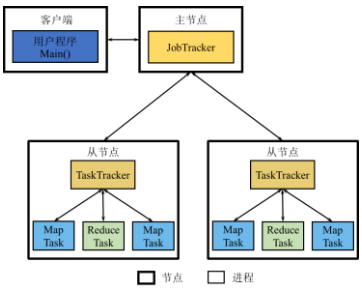

## 架构图

21




## 架构图

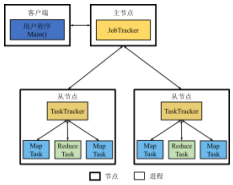
22




## JobTracker

23

- **集群资源管理**：监控TaskTracker与Job的状况
- **集群作业管理**：将Job拆分成Task，跟踪Task的执行进度、资源使用量等信息



## TaskTracker

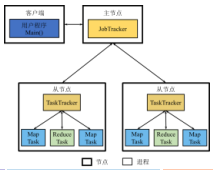
24

- **节点任务管理**
  - **执行命令**：接收JobTracker 发送过来的命令并执行（如启动新Task、杀死Task等）
  - **资源划分**：使用“slot”等量划分本节点上的资源量（CPU、内存等），一个Task 获取到一个slot 后才有机会运行
    - Map slot -> Map Task
    - Reduce slot -> Reduce Task
  - **汇报信息**：通过“心跳”将本节点上资源使用情况和任务运行进度汇报给JobTracker
    - 如何汇报?

## Task

25

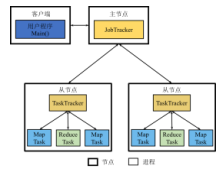
- **任务执行**
  - TaskJobTracker根据TaskTracker汇报的信息进行调度，命令存在空闲slot的TaskTracker启动Task进程执行map或reduce任务
  - 在Hadoop MapReduce的实现中该进程的名称为Child



## Client

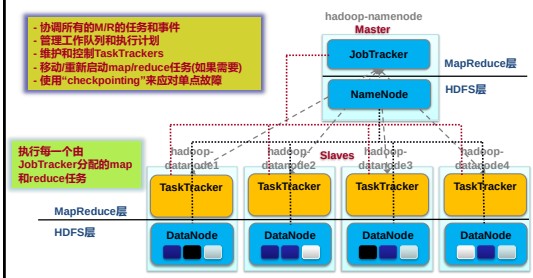
26

- **提交作业**：用户编写的MapReduce程序通过Client提交到JobTracker
- **作业监控**：用户可通过Client提供的一些接口查看作业运行状态



## MapReduce与HDFS关系

27




## MapReduce vs. HDFS关系

28

- 计算与存储相分离
- "计算向数据靠拢"，而不是"数据向计算靠拢"

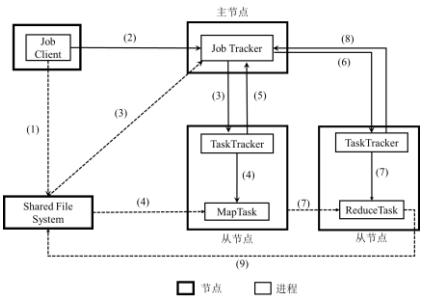
## 大纲

29

- 设计思想
- 体系架构
  - 架构图
  - 应用程序执行流程
- 工作原理
- 容错机制

## 应用程序执行流程

30



## 应用程序执行流程

31

① Client将用户编写的MapReduce作业的配置信息、jar包等信息上传到共享的文件系统，通常是HDFS。
② Client提交作业给JobTracker，即告知作业信息的位置。
③ JobTracker读取作业的信息，生成一系列Map和Reduce任务，调度给拥有空闲slot的TaskTracker。
④ TaskTracker根据JobTacker的指令启动Child进程执行Map任务，Map任务将从共享文件系统读取输入数据。
⑤ JobTracker从TaskTracker处获得Map任务进度信息。
⑥ 一旦Map任务完成后，JobTacker将Reduce任务分发给TaskTracker。
⑦ TaskTracker根据JobTacker的指令启动Child进程执行Reduce任务，Reduce任务将从Map端读取数据。
⑧ JobTracker从TaskTracker处获得Reduce任务进度信息。
⑨ 当Reduce任务完成计算并将结果写入共享文件系统，则意味着整个作业执行完毕。

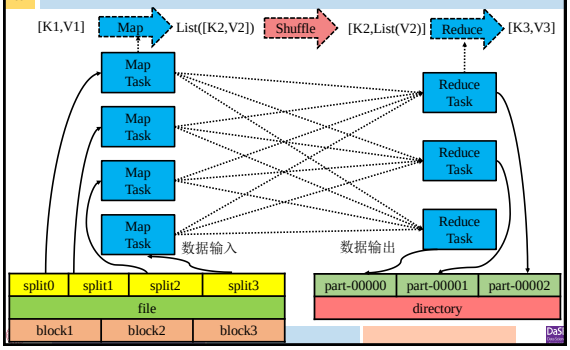
## 大纲

32

- 设计思想
- 体系架构
- 工作原理
- 容错机制

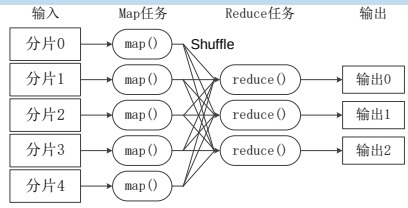
## Map-Shuffle-Reduce

33




## MapReduce工作过程

34




•不同的Map任务之间不会进行通信
•不同的Reduce任务之间也不会发生任何信息交换
•用户不能显式地从一台机器向另一台机器发送消息
•所有的数据交换都是通过MapReduce框架自身去实现的(Shuffle)

## 大纲

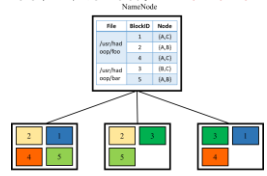
35

- 设计思想
- 体系架构
- 工作原理
  - 数据输入
  - Map阶段
  - Shuffle阶段
  - Reduce阶段
  - 数据输出
- 容错机制

## 数据输入

36

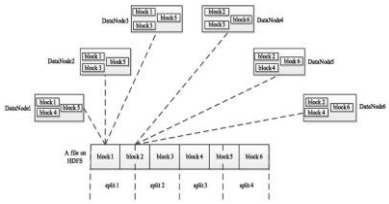
- 从存储系统中文件与Map任务可处理的键值对记录之间的映射
  - 输入文件的格式
- 问题：文件分块存储，可能存在跨块记录



## Split vs. Block

37

- 逻辑概念，包含一些元数据信息，如数据起始位置、数据长度、数据所在节点等



## InputFormat

38

- 数据逻辑划分
  - InputFormat根据预定义格式将输入数据在逻辑上划分为若干个Split（切片）
  - Map任务读取的单位是Split，而不是物理的文件块
  - Split的数量往往决定了Map任务的个数，一个Split的数据一般由一个Map任务来处理
- 键值对解析
  - 给定一个Split，InputFormat将根据分隔符、大小等元信息将Split中的数据解析为相应键值对

## 常见的InputFormat

39

- TextInputFormat
- KeyValueTextInputFormat
- NLineInputFormat
- CombineTextInputFormat
- …
- 自定义InputFormat

## 大纲

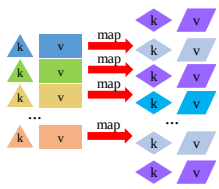
40

- 设计思想
- 体系架构
- 工作原理
  - 数据输入
  - Map阶段
  - Shuffle阶段
  - Reduce阶段
  - 数据输出
- 容错机制

## Map逻辑过程

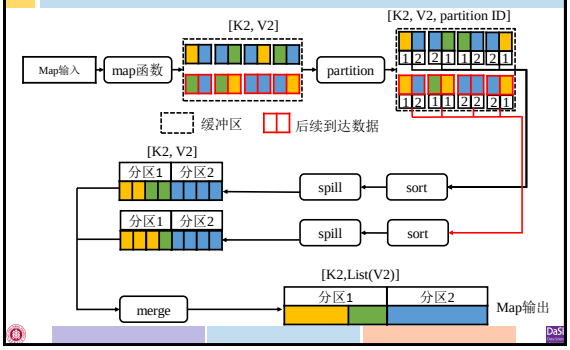
41

- [k1, v1] → List([k2, v2])



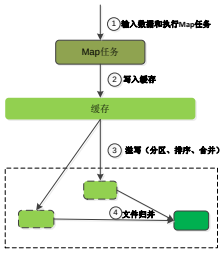

## Map物理过程

42



## Map物理过程

43




(1)输入数据和执行Map任务
•执行Map函数

(2)写入缓存
•每个Map任务分配一个缓存

(3)溢写：设置溢写比例
·分区：哈希函数，其它？
·排序：局部有序
·合并：根据用户Combine函数计算

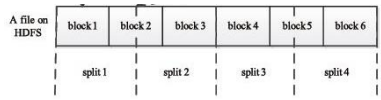
(4)文件归并
·归并：将key相同的记录拼接在一起
•归并得到的文件，放在本地磁盘
•JobTracker会一直监测Map任务的执行，并通知Reduce任务来领取数据

合并（Combine）和归并（Merge）的区别？
两个键值对<"a",1>和<"a",1>，如果合并，会得到<"a",2>，如果归并，会得到<"a",<1,1>>

## Map任务的数量

44

- MapReduce为每个split创建一个Map任务，split 的多少决定了Map任务的数目
- mapred.map.tasks设置程序员期望的map个数




## 大纲

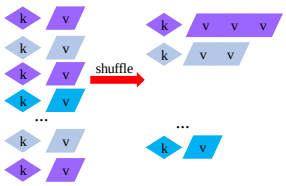
45

- 设计思想
- 体系架构
- 工作原理
  - 数据输入
  - Map阶段
  - Shuffle阶段
  - Reduce阶段
  - 数据输出
- 容错机制

## Shuffle逻辑过程

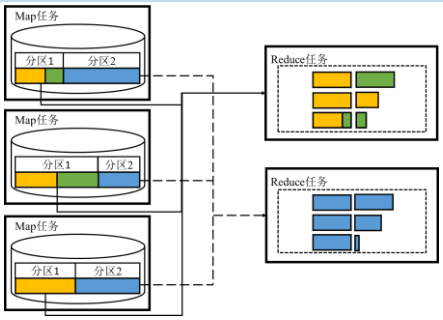
46

- List([k2, v2]) → [k2, List(v2)]




## Shuffle物理过程

47




## 何时Shuffle？

48

- 当系统中的Map任务完成率达到设定阈值时，系统将启动Reduce任务
  - 例如，阈值设定为60%意味着如果系统中共有100个Map任务，那么一旦有60个Map任务已经完成了就可以启动Reduce任务，而不必等到这100个Map任务全部完成
- Reduce任务不会等到所有的Map任务结束才复制数据，但是拉取的必然是已经完成了的Map任务的数据，即已经保存在磁盘上的文件

## 大纲

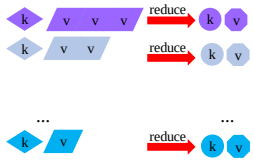
49

- 设计思想
- 体系架构
- 工作原理
  - 数据输入
  - Map阶段
  - Shuffle阶段
  - Reduce阶段
  - 数据输出
- 容错机制

## Reduce逻辑过程

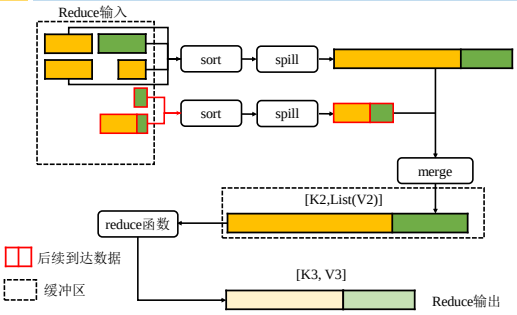
50

- [k2, List(v2)] → [k3, v3]




## Reduce物理过程

51




## Reduce物理过程

52

**(1) 领取数据**

•Map任务完成后， JobTracker通知Reduce领取数据

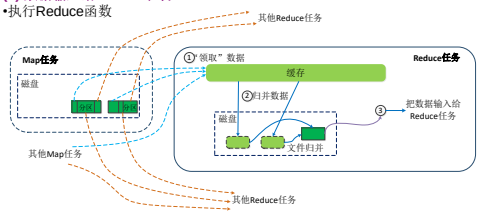
**(2) 归并数据**

•Reduce领取数据先放入缓存，来自不同Map机器，归并写入磁盘

•多个溢写文件归并成一个或多个大文件，文件中的键值对是**排序**的

**(3) 数据输入给Reduce任务**

•执行Reduce函数




## Reduce任务的数量

53

- 程序指定
- 最优的Reduce任务个数取决于集群中可用的reduce任务槽(slot)的数目
- 通常设置比reduce任务槽数目稍微小一些的Reduce任务个数（这样可以预留一些系统资源处理可能发生的错误）

## 大纲

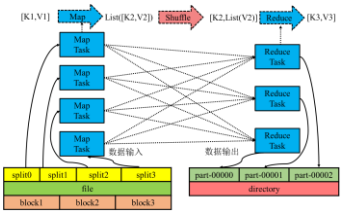
54

- 设计思想
- 体系架构
- 工作原理
  - 数据输入
  - Map阶段
  - Shuffle阶段
  - Reduce阶段
  - 数据输出
- 容错机制

## 数据输出

55

- 每个Reduce任务的输出结果将以一个文件的形式保持到指定的目录当中
  - MapReduce输出结果是一组文件




## OutputFormat

56

- 与数据输入阶段相反，MapReduce需要定义输出文件的格式，即OutputFormat
  - 包括分隔符等元信息
  - 从MapReduce程序处理的逻辑键值对数据到物理存储之间的映射
  - MapReduce系统将Reduce任务处理产生的结果按OutputFormat定义的格式写入HDFS等

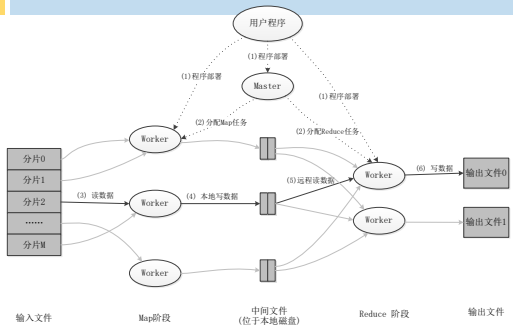
## 常见的OutputFormat

57

- TextOutputFormat
- KeyValueTextOutputFormat
- NLineOutputFormat
- CombineTextOutputFormat
- …
- 自定义OutputFormat

## 回顾：MapReduce工作过程

58




## 大纲

59

- 设计思想
- 体系架构
- 工作原理
- 容错机制

## MapReduce故障类型

60

- 主节点故障
  - JobTracker故障：如宕机引起
- 从节点故障
  - TaskTracker故障：如节点宕机引起
  - Task故障: 如JVM内存不够退出
- MapReduce容错和HDFS容错是两回事

## 大纲

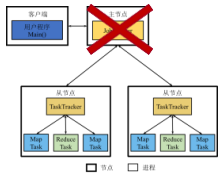
61

- 设计思想
- 体系架构
- 工作原理
- 容错机制
  - JobTracker故障
  - TaskTracker故障
  - Task故障

## JobTracker故障

62

- 对于MapReduce 1.0的架构，JobTracker故障意味着所有作业需要重新执行
- MapReduce 1.0没有处理JobTracker故障的机制，因而成为单点瓶颈



## 大纲

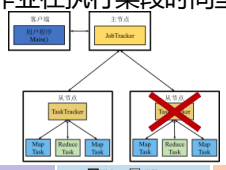
63

- 设计思想
- 体系架构
- 工作原理
- 容错机制
  - JobTracker故障
  - TaskTracker故障
  - Task故障

## TaskTracker故障

64

- JobTracker不会接收到“心跳”
- JobTracker会安排其他TaskTracker重新运行失败TaskTracker的任务
- 这个过程对于用户来说是透明的，只会感觉到该作业在执行某段时间里变慢了而已



## 大纲

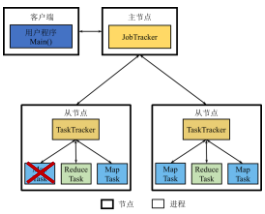
65

- 设计思想
- 体系架构
- 工作原理
- 容错机制
  - JobTracker故障
  - TaskTracker故障
  - Task故障

## Task故障

66

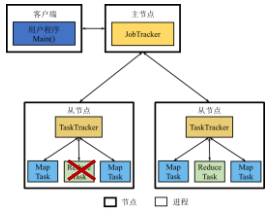
- Map Task故障
  - 重新执行Map任务
  - 去HDFS重新读入数据



## Task故障

67

- Reduce Task故障
  - 重新执行Reduce任务
  - 去哪里重新读入数据?




## Task故障

68

- 典型例子
  - Map任务或Reduce任务代码异常
- 当一个任务经过最大尝试次数运行后仍然失败，那么整个作业将被标记为失败

## 课后阅读

69

- 分布式系统概念与设计，George Coulouris等著，金蓓弘等译
  - 第21章 21.6.1
- 论文
  - Dean, J., & Ghemawat, S. (2004). MapReduce : Simplified Data Processing on Large Clusters. In OSDI (pp. 137–149).

## 本讲小结

70

- 设计思想
- 体系架构
- 工作原理
- 容错机制

**谢谢！Q&A**