MapReduce

于策

思考

- 数据：10000个文本文件，英文
  - 问题1
    - 统计各个单词在所有文件中出现的次数
  - 问题2
    - 统计各个单词分别在哪些文件中出现过
- 串行算法？
- 并行算法？

Outline

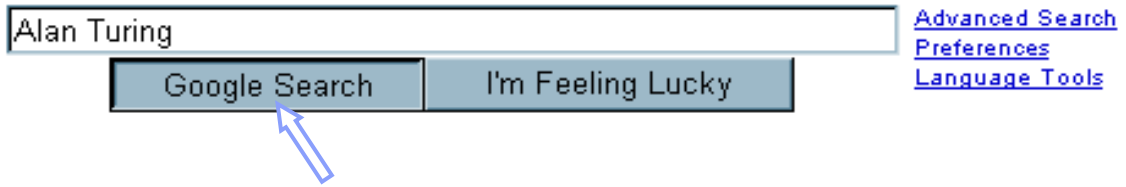
- MapReduce编程模型
  - 算法
  - Shuffle和Sort
  - 实测数据
- Hadoop
  - HDFS
  - HBase
- 实例（天文交叉证认计算）

Outline

- **MapReduce编程模型**
  - 算法
  - Shuffle和Sort
  - 实测数据
- Hadoop
  - HDFS
  - HBase
- 实例（天文交叉证认计算）

MapReduce起源：Google搜索

Alan Turing
1912—1954



- 每一次搜索
  - 200+ CPU
  - 200TB以上数据
  - 10^{10} CPU周期
  - 0.1秒内响应
  - 5¢广告收入

MapReduce：大规模数据处理

Jeff Dean

- 处理海量数据（>1TB）
- 上百/上千 CPU 实现并行处理
- 简单地实现以上目的

分而治之

Divide and Conquer

"Google Earth uses **70.5 TB**: 70 TB for the raw imagery and 500 GB for the index data."

From: http://googlesystem.blogspot.com/2006/09/how-much-data-does-google-store.html

MapReduce特性

- 自动实现分布式并行计算
- 容错
- 提供状态监控工具
- 模型抽象简洁，程序员易用

Outline

- **MapReduce编程模型**
  - 算法
  - Shuffle和Sort
  - 实测数据
- Hadoop
  - HDFS
  - HBase
- 实例（天文交叉证认计算）

MapReduce编程模型

- 借鉴了函数式编程方式（functional programming）
- 用户只需要实现两个函数接口：
  - **map (in_key, in_value) ->**
    (out_key, intermediate_value) list
  - **reduce (out_key, intermediate_value list) ->**
    out_value list

函数式编程（Functional Programming）

- Functional操作不修改数据结构
  - 创建新数据结构
- 原始数据以原形式存在，不会被修改
- 对于程序设计，数据流是隐式的
- 操作执行的次序无关

函数式编程

```
# let double x =
    x * 2
  in
  List.map double [ 1; 2; 3 ];;
- : int list = [2; 4; 6]
```

```
let multiply n list =
  let f x =
    n * x
  in
  List.map f list
  ;;
```

```
# multiply 2 [1; 2; 3];;
- : int list = [2; 4; 6]
# multiply 5 [1; 2; 3];;
- : int list = [5; 10; 15]
```

map

- 将数据源中的记录（文本中的行、数据库中条目等）作为map函数中的key*value对
  - 例如，（filename, line）
- map()将生成一个或多个中间结果，以及与input相对应的一个output key

reduce

- map操作结束后，所有与某指定out key相对应的中间结果组合为一个列表（list）。
- reduce()函数将这些中间结果组合为一个或多个对应于同一output key 的 final value
  - （实际上每一个output key通常只有一个final value）

示例：WordCount

- 源数据
  - Page 1:
    - the weather is good
  - Page 2:
    - today is good
  - Page 3:
    - good weather is good

map 输出

- 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).

reduce 的输入

- 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)

reduce输出

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

WordCount 伪代码

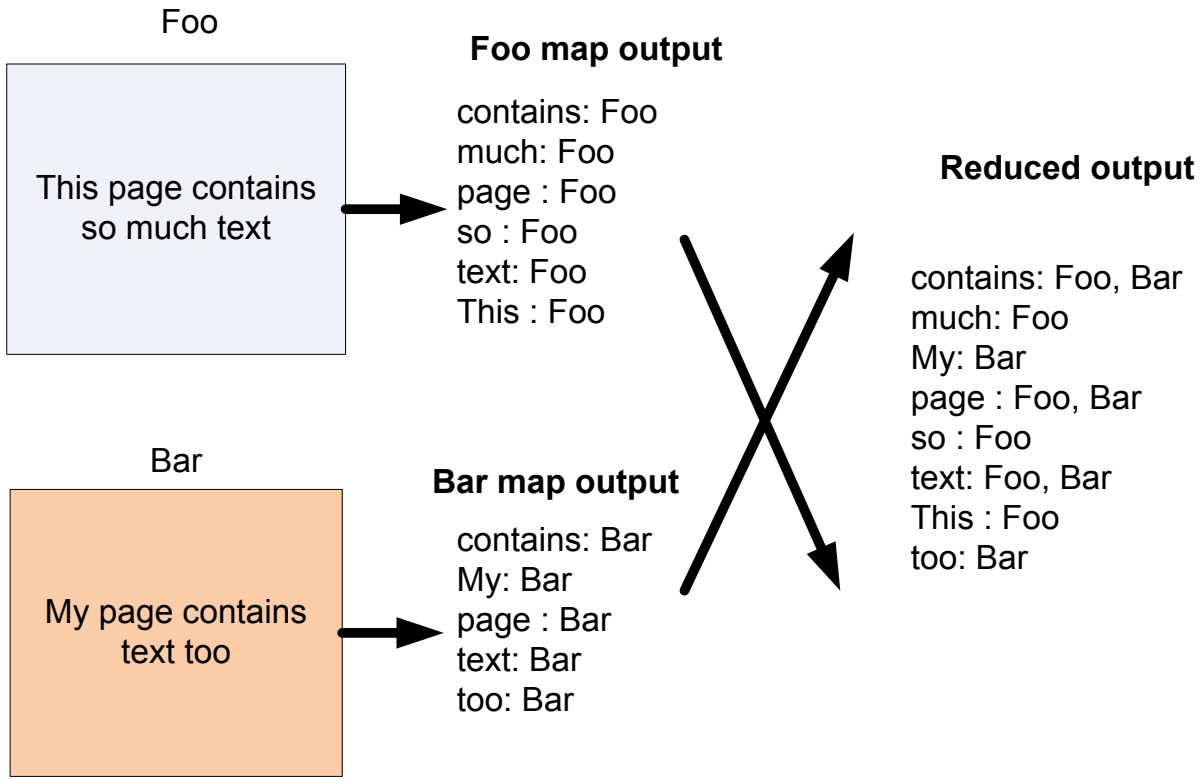
```
map(String input_key, String input_value):
  // input_key: document name
  // input_value: document contents
  for each word w in input_value:
    EmitIntermediate(w, "1");

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));
```

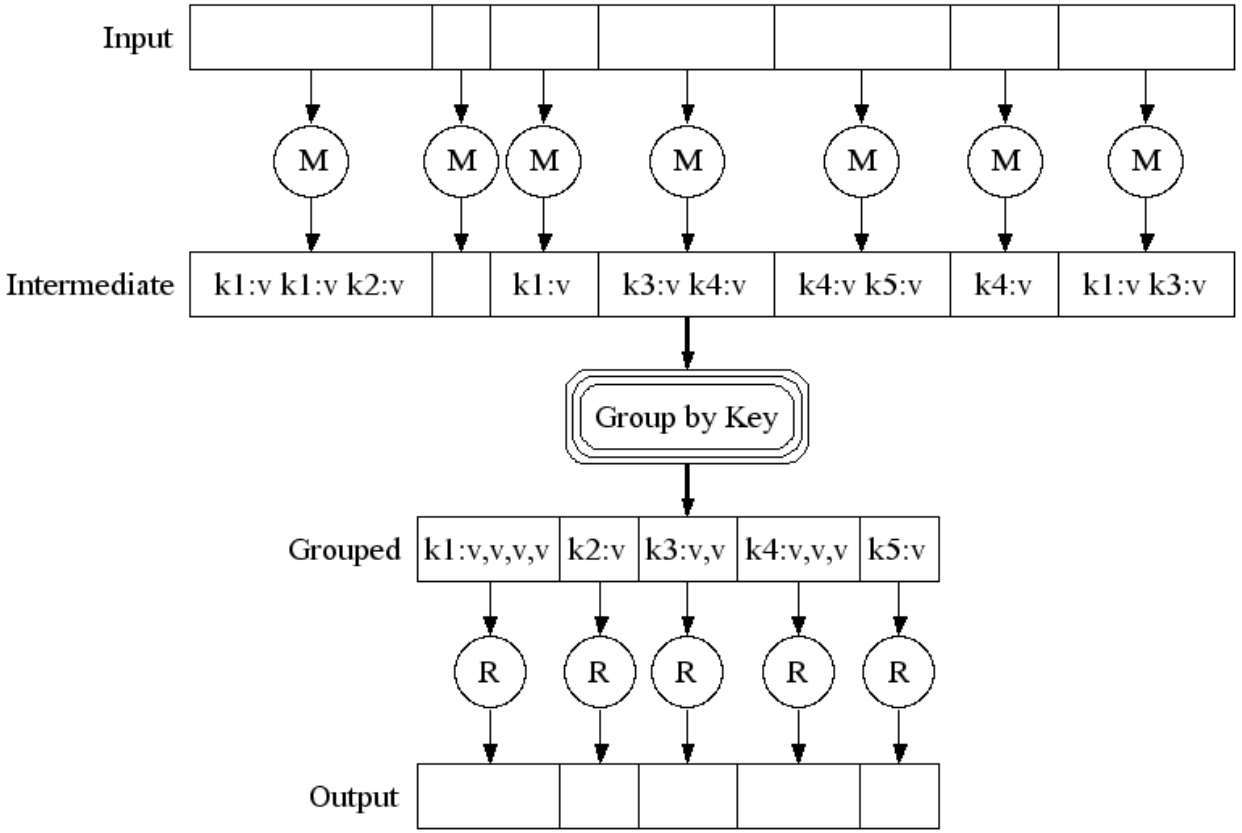
倒排索引 (Inverted Index) Algorithm

- Mapper: For each word in (file, words), map to (word, file)
- Reducer: Identity function
- 文件内容：
  - foo
    - This page contains so much text
  - bar
    - My page contains text too

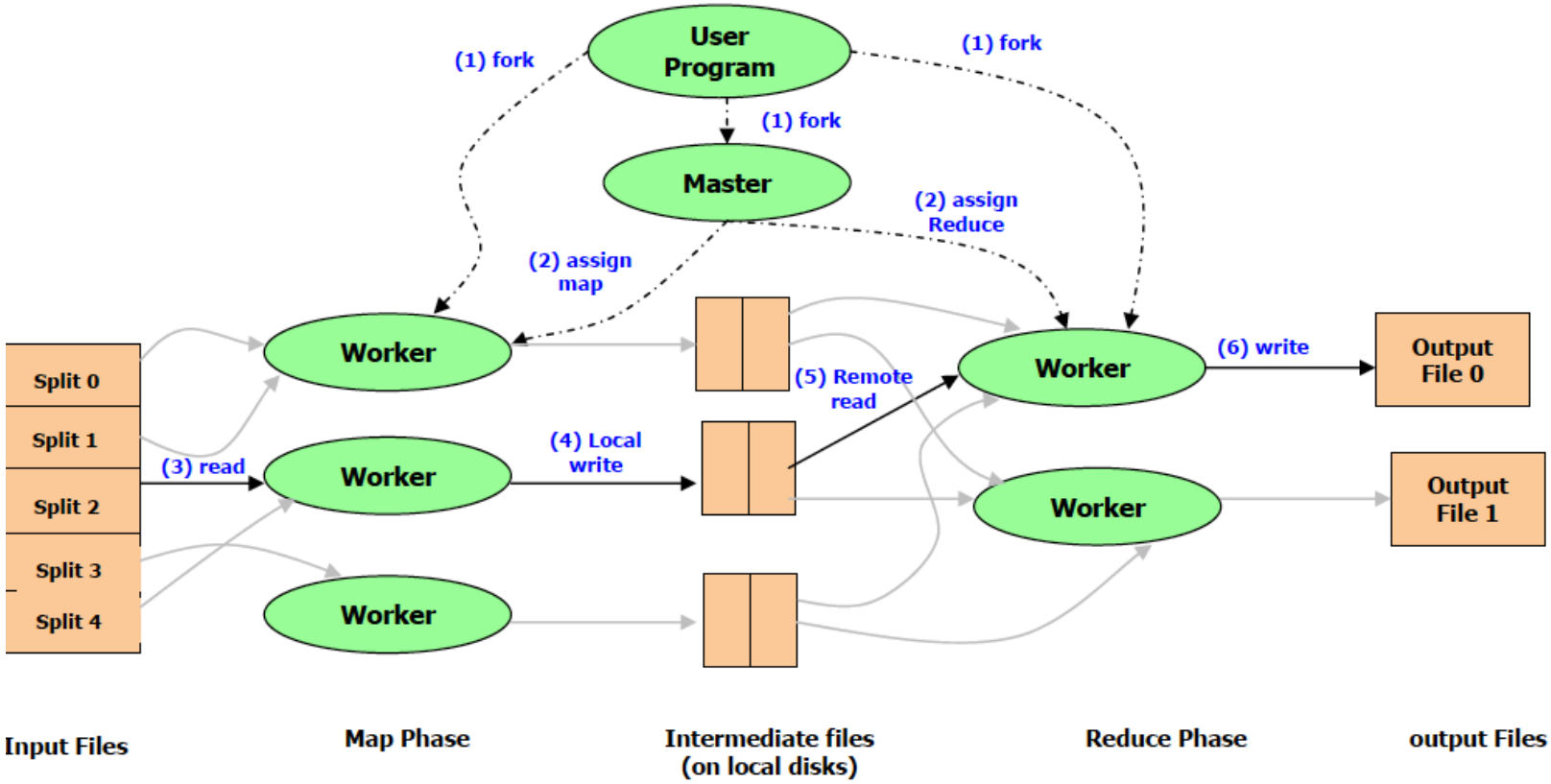
Inverted Index: Data flow



MapReduce逻辑过程



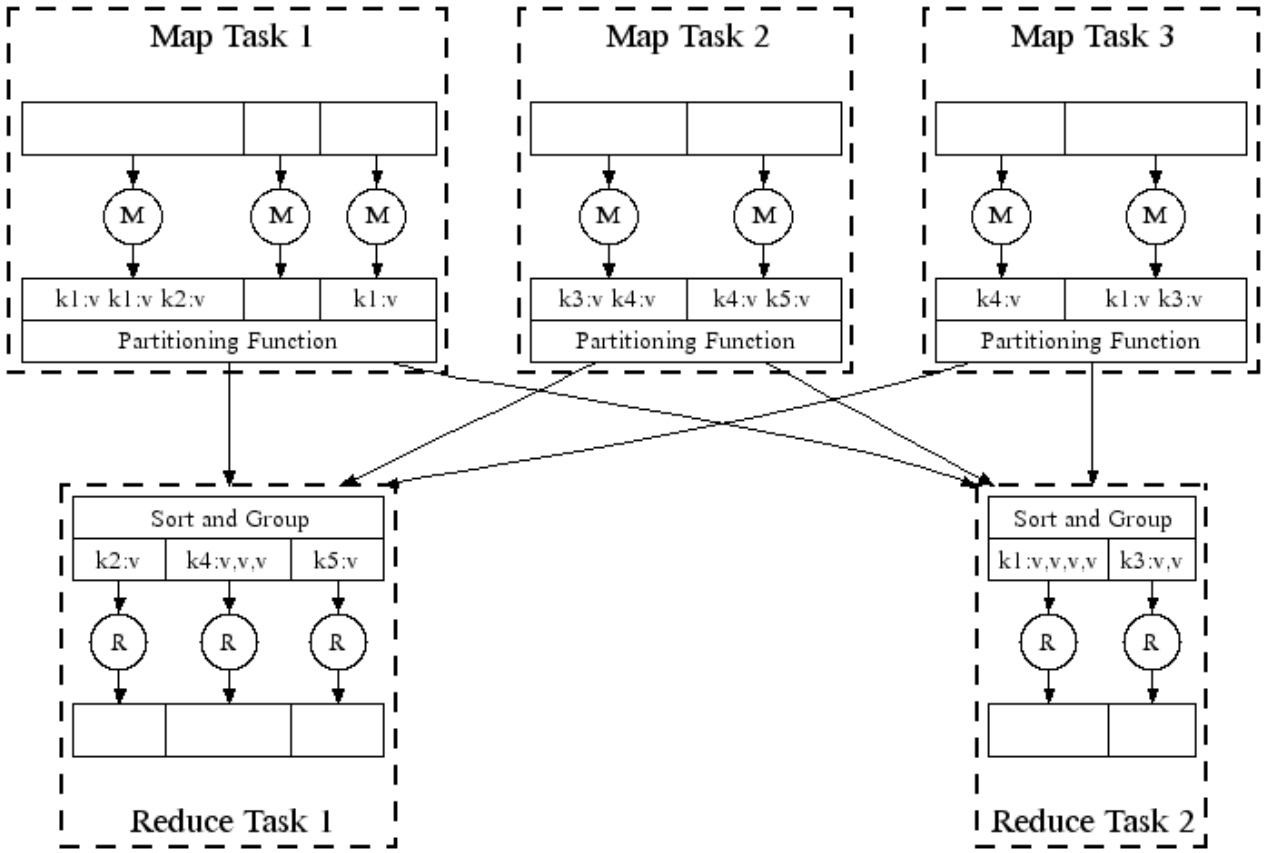
运行过程


User Program
(1) fork
(1) fork
(1) fork
Master
(2) assign Reduce
(2) assign map
Worker
Worker
Worker
Worker
Worker
Split 0
Split 1
Split 2
Split 3
Split 4
(3) read
(4) Local write
(5) Remote read
(6) write
Output File 0
Output File 1
Input Files
Map Phase
Intermediate files (on local disks)
Reduce Phase
output Files

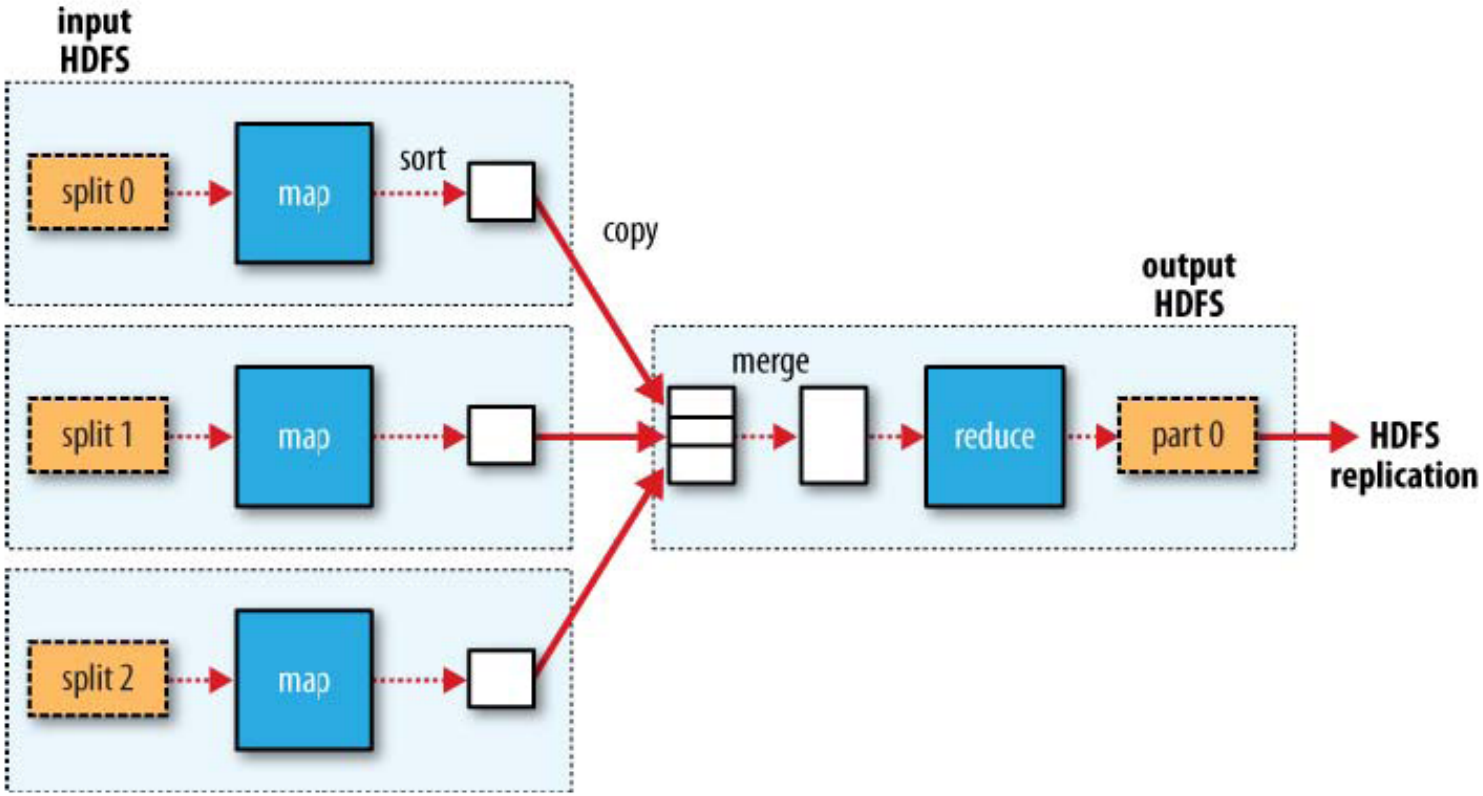
并行化

- map()函数可以并行执行，为不同的输入数据集生成不同的中间结果
- reduce()函数也可以并行执行，分别处理不同的output key
- map和reduce的处理过程中不发生通信
- 瓶颈：
  - 只有当map处理全部结束后，reduce过程才能够开始

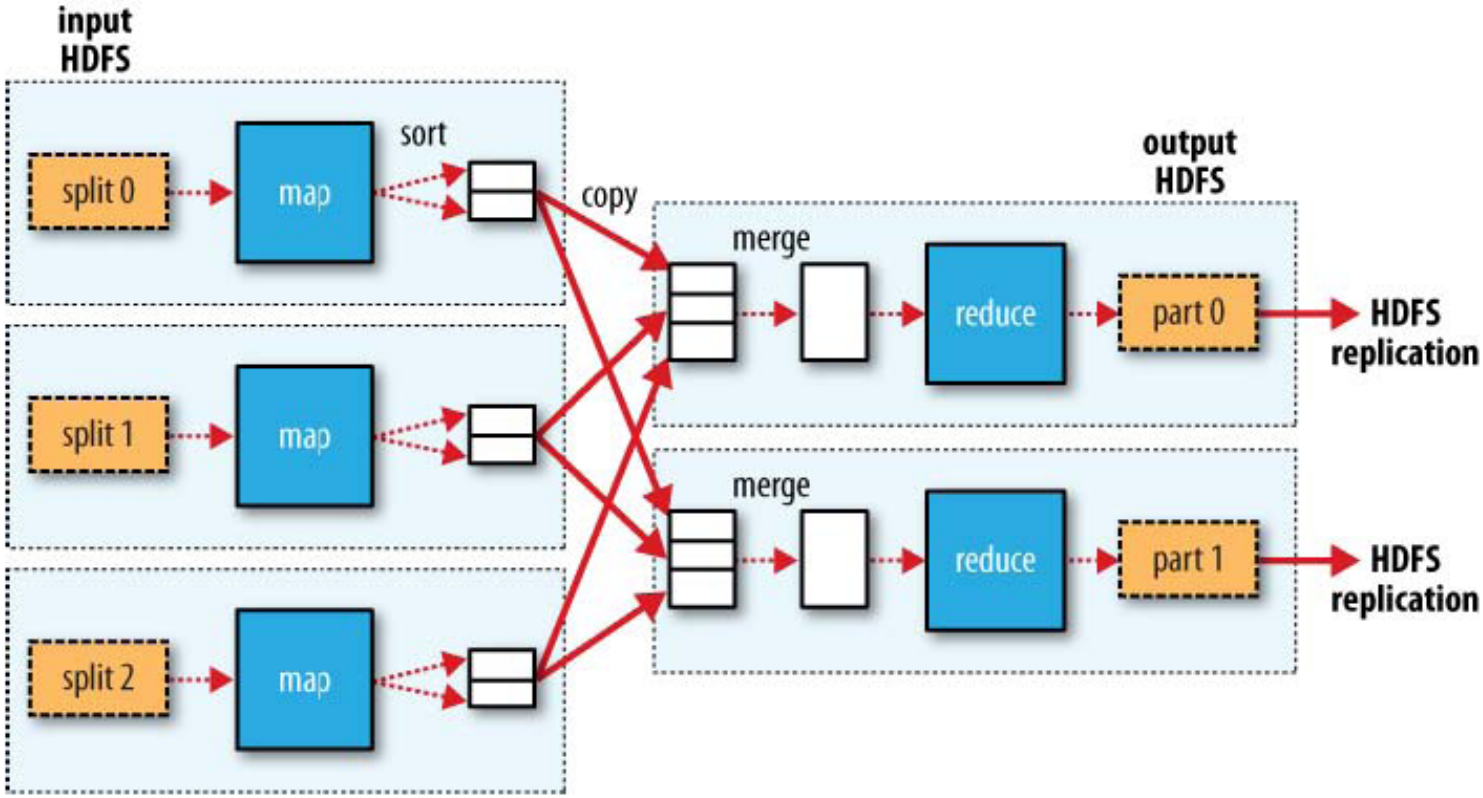
MapReduce的并行执行



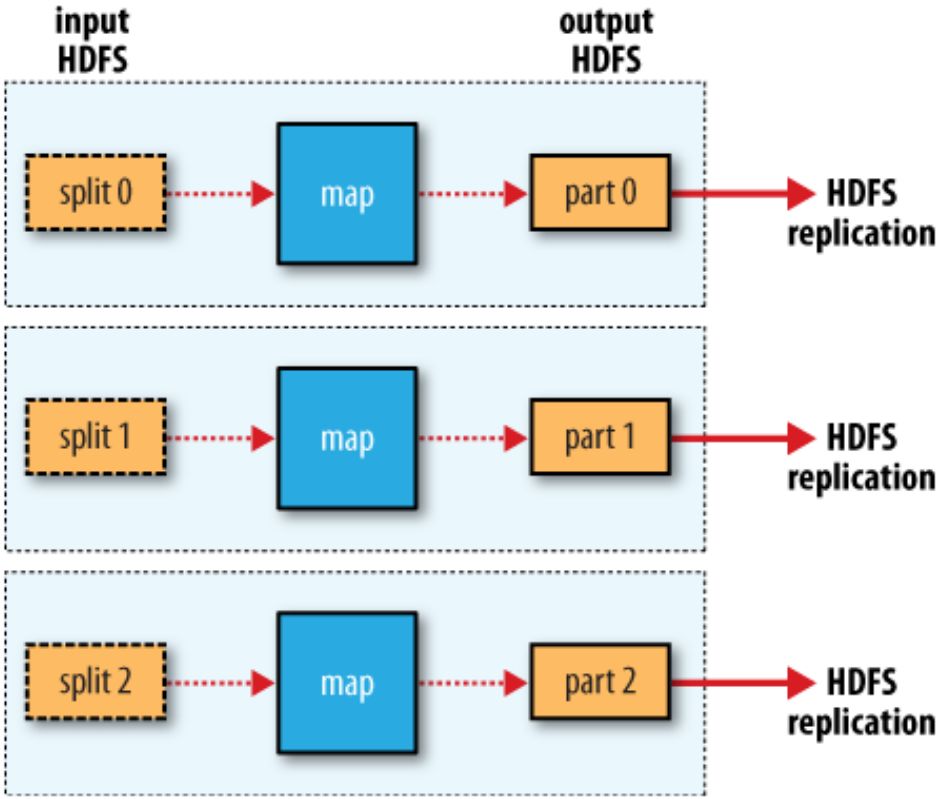
单一reduce处理


input
HDFS
split 0
map
sort
split 1
map
split 2
map
copy
output
HDFS
merge
reduce
part 0
HDFS
replication

多个reduce处理


input
HDFS
split 0
map
sort
split 1
map
split 2
map
copy
merge
reduce
part 0
output
HDFS
HDFS
replication
merge
reduce
part 1
HDFS
replication

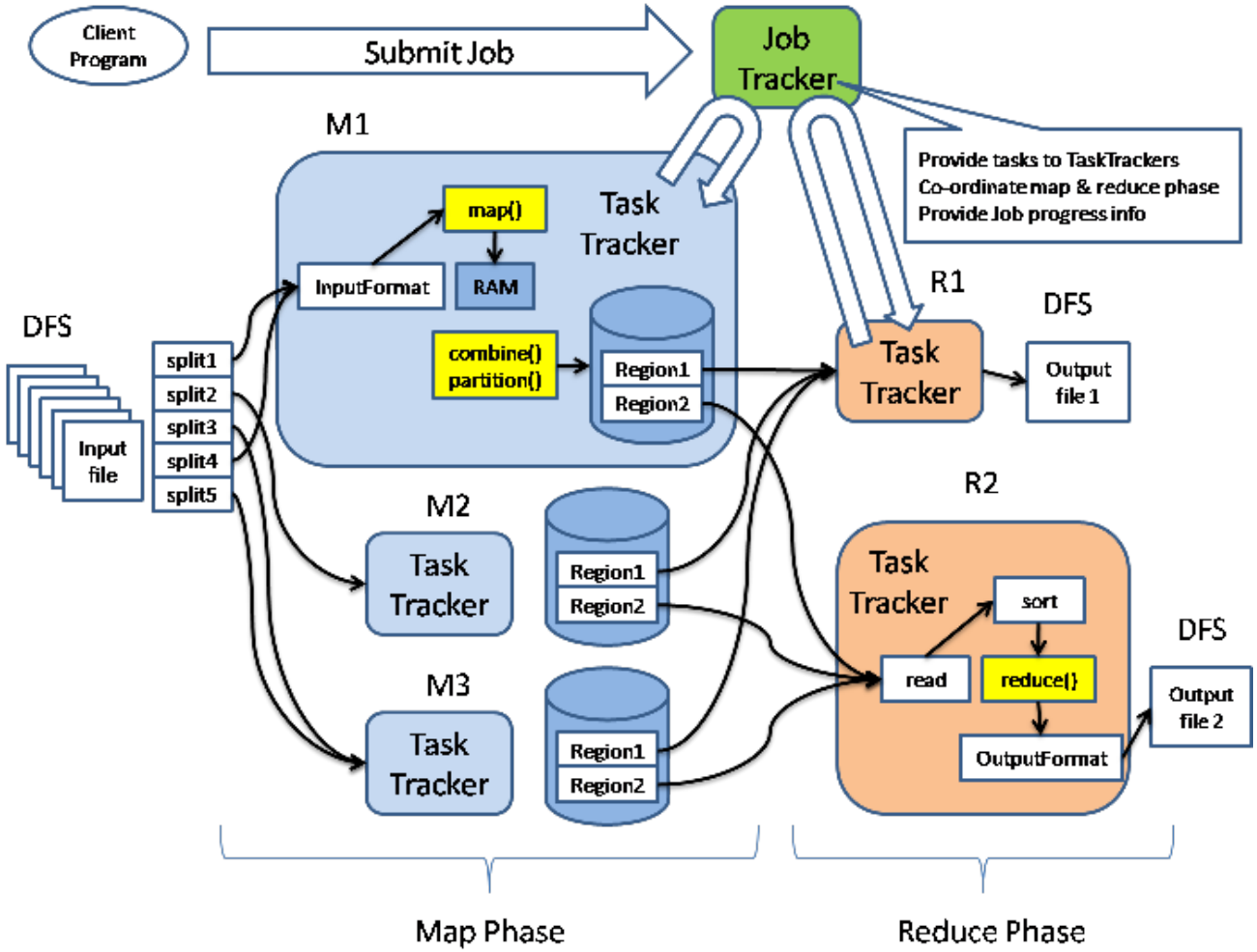
无reduce处理


input
HDFS
output
HDFS
split 0
map
part 0
HDFS
replication
split 1
map
part 1
HDFS
replication
split 2
map
part 2
HDFS
replication

Outline

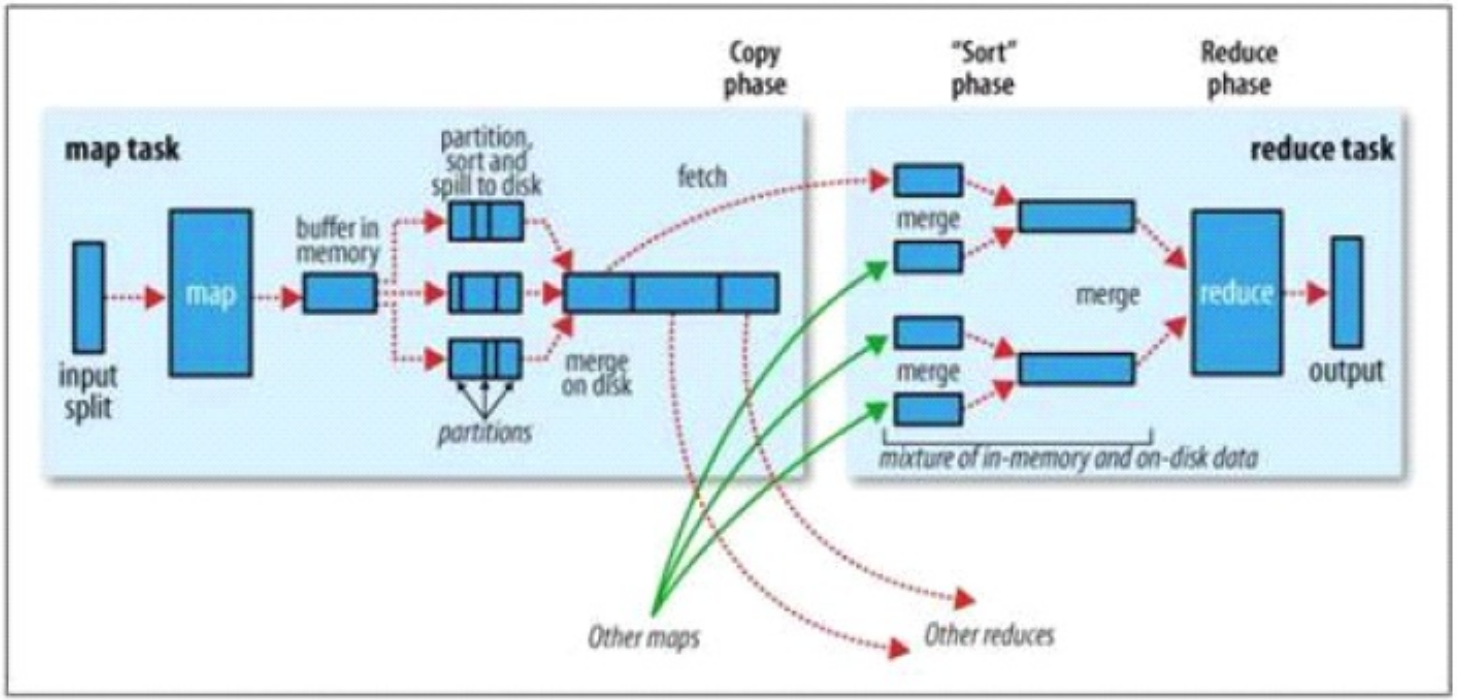
- **MapReduce编程模型**
  - 算法
  - **Shuffle和Sort**
  - 实测数据
- Hadoop
  - HDFS
  - HBase
- 实例（天文交叉证认计算）

任务执行过程


Client Program
Submit Job
Job Tracker
Provide tasks to TaskTrackers
Co-ordinate map & reduce phase
Provide Job progress info
M1
Task Tracker
map()
InputFormat
RAM
combine()
partition()
Region1
Region2
DFS
Input file
split1
split2
split3
split4
split5
R1
Task Tracker
DFS
Output file 1
M2
Task Tracker
Region1
Region2
R2
Task Tracker
sort
read
reduce()
OutputFormat
DFS
Output file 2
M3
Task Tracker
Region1
Region2
Map Phase
Reduce Phase

Shuffle 和 Sort

- 当Map 开始产生输出时，并不是简单的把数据写到磁盘，因为频繁的磁盘操作会导致性能严重下降。它的处理过程更复杂，数据首先是写到内存中的一个缓冲区，并进行预排序，以提升效率。

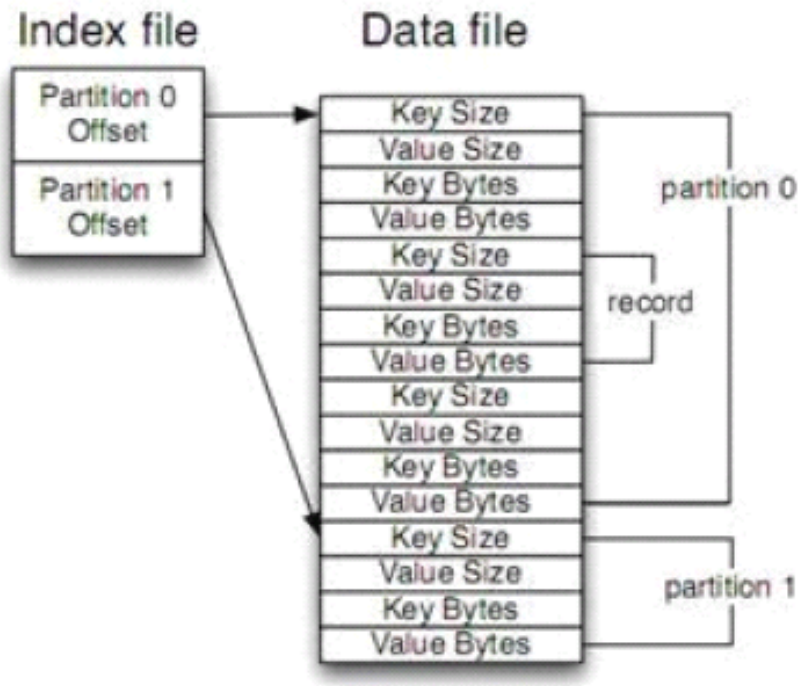


Map的结果输出

- 每个Map 任务都有一个用来写入输出数据的循环内存缓冲区。这个缓冲区默认大小是100MB。
- 当缓冲区中的数据量达到一个特定阀值(默认是0.80)，系统将会启动一个后台线程把缓冲区中的内容spill 到磁盘。在spill 过程中，Map 的输出将会继续写入到缓冲区，但如果缓冲区已满，Map 就会被阻塞直到spill 完成。

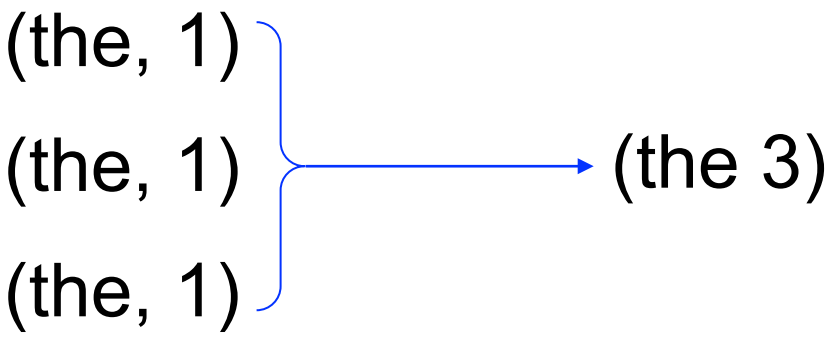
Map输出文件

- spill 线程在把缓冲区的数据写到磁盘前，会对它进行一个二次快速排序，首先根据数据所属的partition 排序，然后每个partition 中再按Key 排序。
- 输出包括一个索引文件和数据文件。



Map输出的压缩

- 如果设定了Combiner，将在排序输出的基础上运行。
- Combiner 就是一个Mini Reducer，它在执行Map 任务的节点本身运行，先对Map 的输出做一次简单Reduce，使得Map 的输出更紧凑，更少的数据会被写入磁盘和传送到Reducer。



Map输出结束

- 每当内存中的数据达到spill 阀值的时候，都会产生一个新的spill 文件，所以在Map任务写完它的最后一个输出记录时，可能会有多个spill 文件。
- 在Map 任务完成前，所有的spill 文件将会被归并排序为一个索引文件和数据文件
- 当spill 文件归并完毕后，Map 将删除所有的临时spill 文件，并告知TaskTracker 任务已完成。

数据拷贝

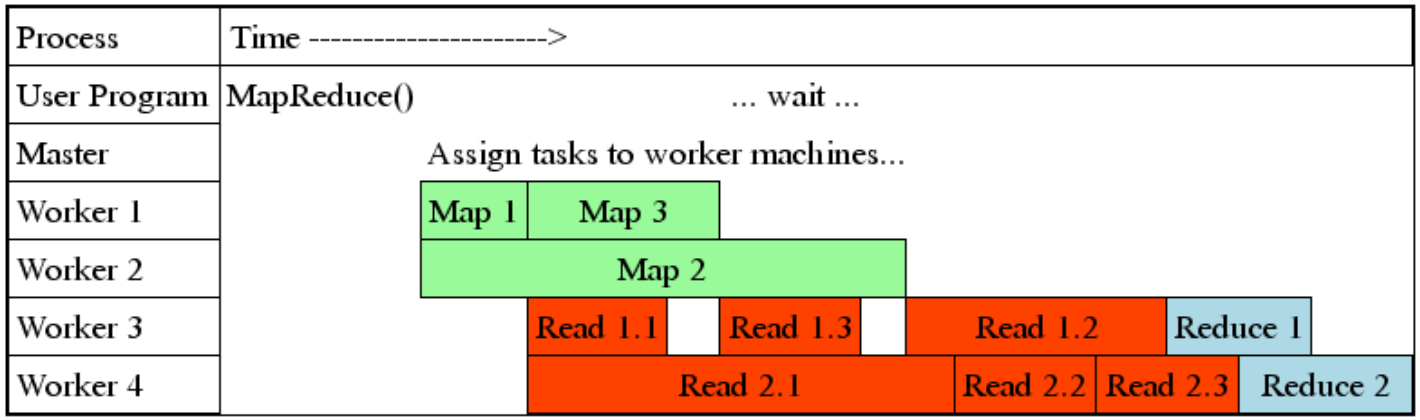
- Map 的输出文件放置在运行Map 任务的TaskTracker 的本地磁盘上，它是运行Reduce 任务的TaskTracker 所需要的输入数据。
- Reduce 任务的输入数据分布在集群内的多个Map 任务的输出中，Map 任务可能会在不同的时间内完成，只要有其中的一个Map 任务完成，Reduce 任务就开始拷贝它的输出。这个阶段称之为拷贝阶段。
- Reduce 任务拥有多个拷贝线程， 可以并行的获取Map输出。线程数默认是5。

归并

- 拷贝来的数据叠加在磁盘上，有一个后台线程会将它们归并为更大的排序文件，节省后期归并的时间。
- 当所有的Map 输出都被拷贝后，Reduce 任务进入排序阶段（归并阶段，因为排序在Map 端就已经完成），这个阶段会对所有的Map 输出进行归并排序，这个工作会重复多次。
- 假设有50 个Map 输出（可能有保存在内存中），并且归并因子是10，则最终需要5 次归并。每次归并会把10 个文件归并为一个，最终生成5 个中间文件。之后，系统不再把5 个中间文件归并成一个，而是排序后直接提交给Reduce 函数，省去向磁盘写数据这一步。

任务粒度与流水线

- 细粒度任务：Map任务数远多于机器数
  - 最小化故障恢复的时间
  - 使得shuffle操作可以与map同步进行
  - 有利于动态负载平衡
- 2000台机器，通常设置200,000 个map任务，5000 个 reduce任务



Outline

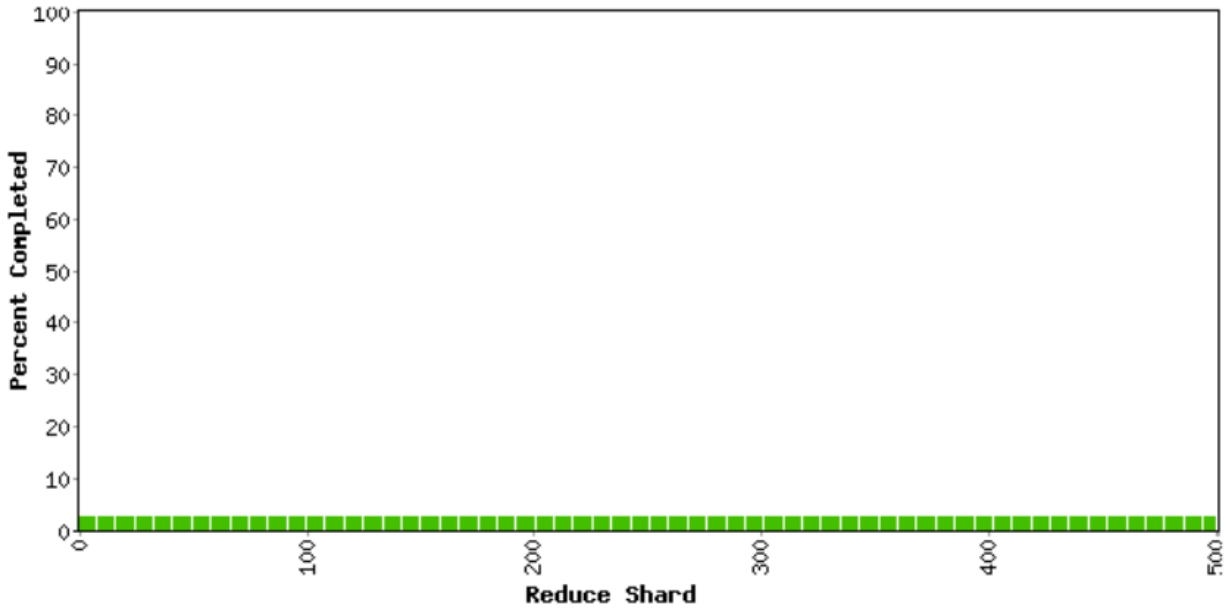
- **MapReduce编程模型**
  - 算法
  - Shuffle和Sort
  - **实测数据**
- Hadoop
  - HDFS
  - HBase
- 实例（天文交叉证认计算）

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 00 min 18 sec

323 workers; 0 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	0	323	878934.6	1314.4	717.0
Shuffle	500	0	323	717.0	0.0	0.0
Reduce	500	0	0	0.0	0.0	0.0



Counters

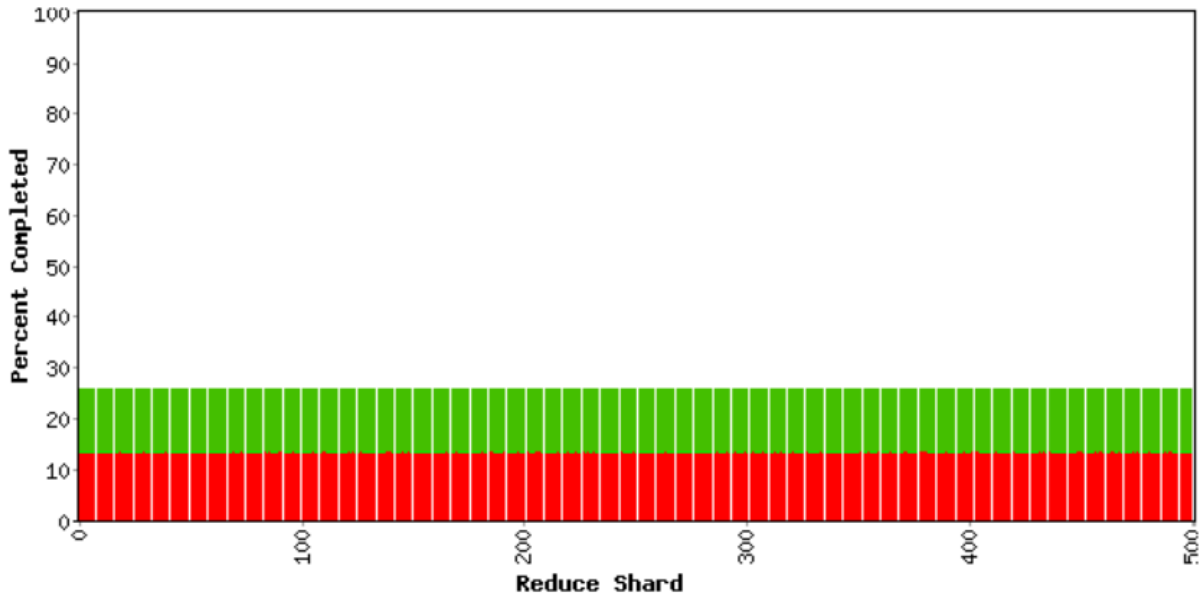
Variable	Minute
Mapped (MB/s)	72.5
Shuffle (MB/s)	0.0
Output (MB/s)	0.0
doc-index-hits	145825686
docs-indexed	506631
dups-in-index-merge	0
mr-operator-calls	508192
mr-operator-	506631

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 05 min 07 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	1857	1707	878934.6	191995.8	113936.6
Shuffle	500	0	500	113936.6	57113.7	57113.7
Reduce	500	0	0	57113.7	0.0	0.0



Counters

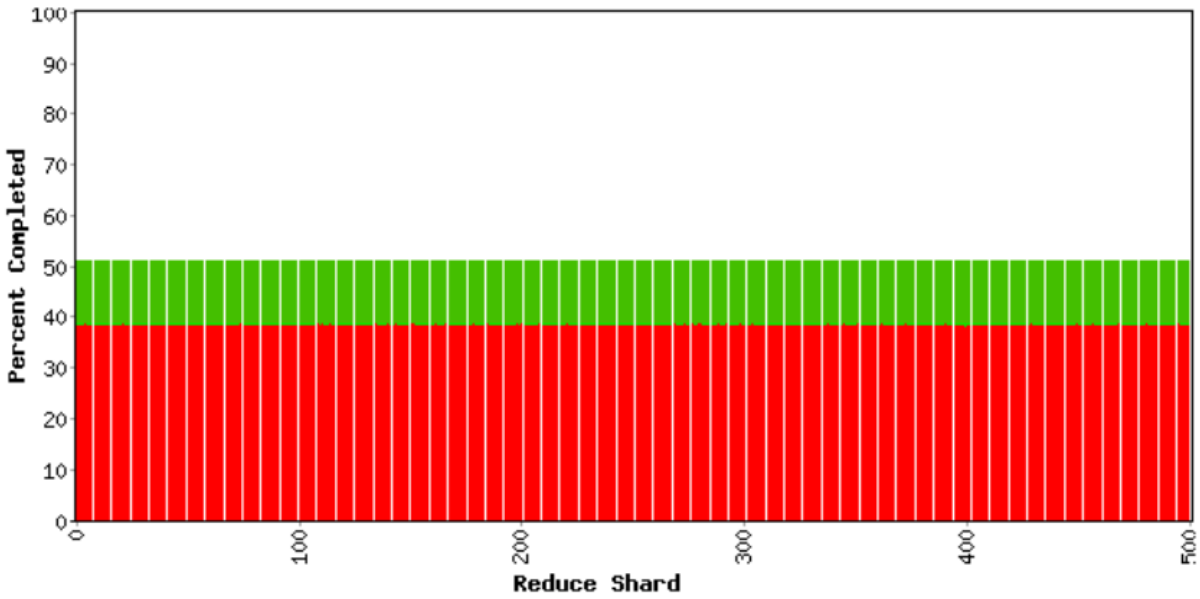
Variable	Minute
Mapped (MB/s)	699.1
Shuffle (MB/s)	349.5
Output (MB/s)	0.0
doc-index-hits	5004411944
docs-indexed	17290135
dups-in-index-merge	0
mr-operator-calls	17331371
mr-operator-outputs	17290135

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 10 min 18 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	5354	1707	878934.6	406020.1	241058.2
Shuffle	500	0	500	241058.2	196362.5	196362.5
Reduce	500	0	0	196362.5	0.0	0.0



Counters

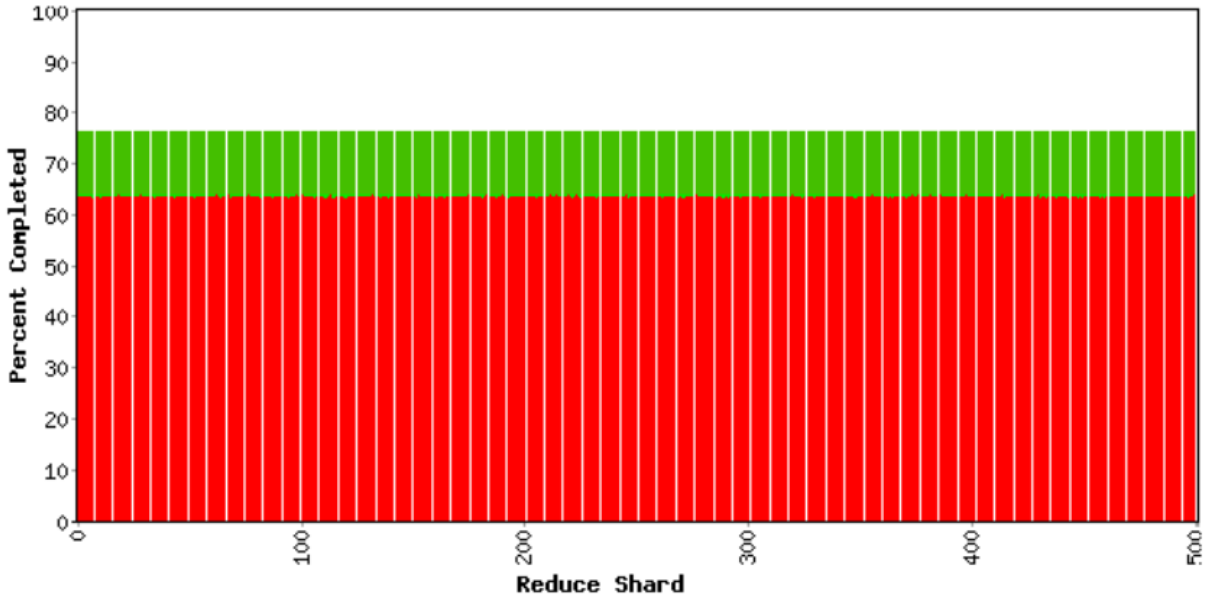
Variable	Minute
Mapped (MB/s)	704.4
Shuffle (MB/s)	371.9
Output (MB/s)	0.0
doc-index-hits	5000364228
docs-indexed	17300709
dups-in-index-merge	0
mr-operator-calls	17342493
mr-operator-outputs	17300709

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 15 min 31 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	8841	1707	878934.6	621608.5	369459.8
Shuffle	500	0	500	369459.8	326986.8	326986.8
Reduce	500	0	0	326986.8	0.0	0.0



Counters

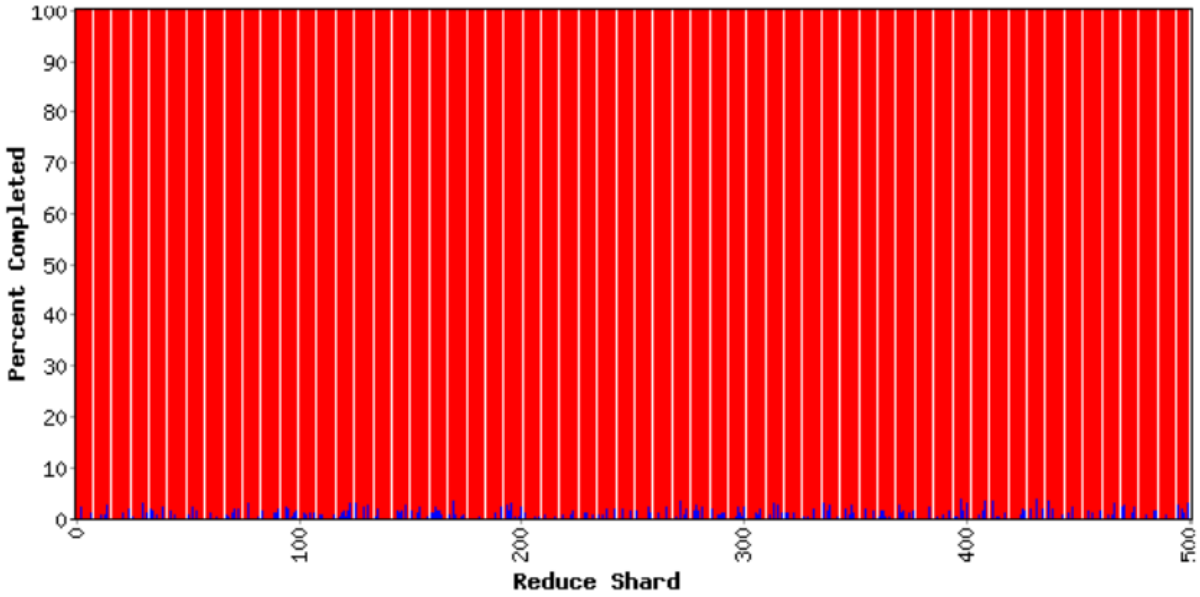
Variable	Minute
Mapped (MB/s)	706.5
Shuffle (MB/s)	419.2
Output (MB/s)	0.0
doc-index-hits	4982870667
docs-indexed	17229926
dups-in-index-merge	0
mr-operator-calls	17272056
mr-operator-outputs	17229926

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 29 min 45 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	195	305	523499.2	523389.6	523389.6
Reduce	500	0	195	523389.6	2685.2	2742.6



Counters

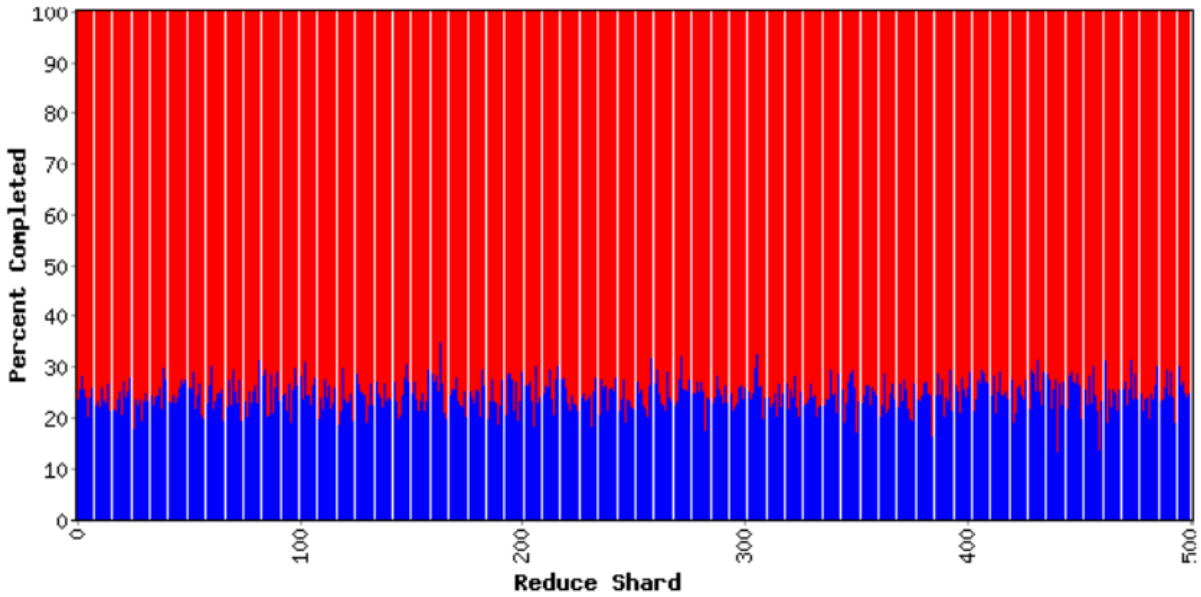
Variable	Minute	
Mapped (MB/s)	0.3	
Shuffle (MB/s)	0.5	
Output (MB/s)	45.7	
doc-index-hits	2313178	10
docs-indexed	7936	
dups-in-index-merge	0	
mr-merge-calls	1954105	
mr-merge-outputs	1954105	

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 31 min 34 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	500	0	523499.2	523499.5	523499.5
Reduce	500	0	500	523499.5	133837.8	136929.6



Counters

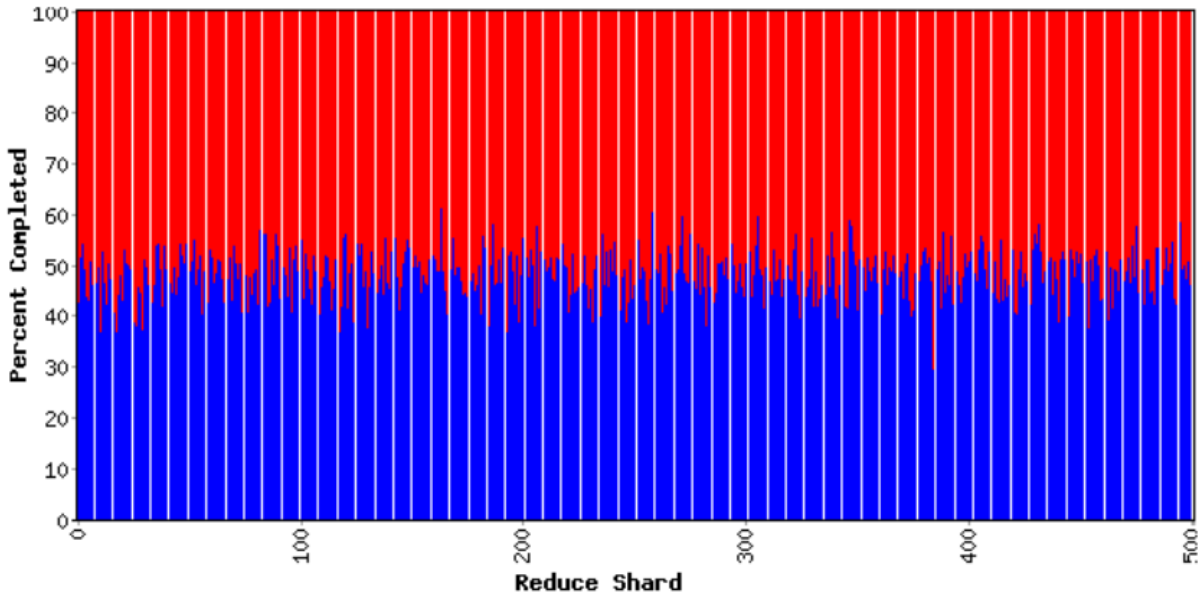
Variable	Minute
Mapped (MB/s)	0.0
Shuffle (MB/s)	0.1
Output (MB/s)	1238.8
doc-index-hits	0
docs-indexed	0
dups-in-index-merge	0
mr-merge-calls	51738599
mr-merge-outputs	51738599

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 33 min 22 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	500	0	523499.2	523499.5	523499.5
Reduce	500	0	500	523499.5	263283.3	269351.2



Counters

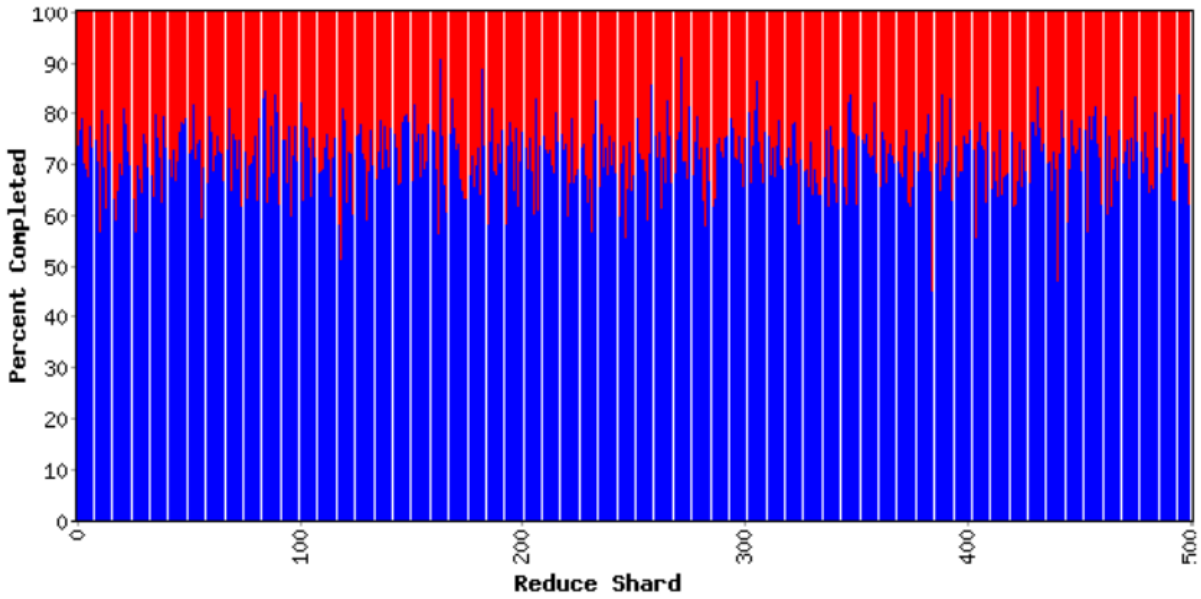
Variable	Minute	
Mapped (MB/s)	0.0	
Shuffle (MB/s)	0.0	
Output (MB/s)	1225.1	
doc-index-hits	0	1(
docs-indexed	0	
dups-in-index-merge	0	
mr-merge-calls	51842100	
mr-merge-outputs	51842100	

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 35 min 08 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	500	0	523499.2	523499.5	523499.5
Reduce	500	0	500	523499.5	390447.6	399457.2



Counters

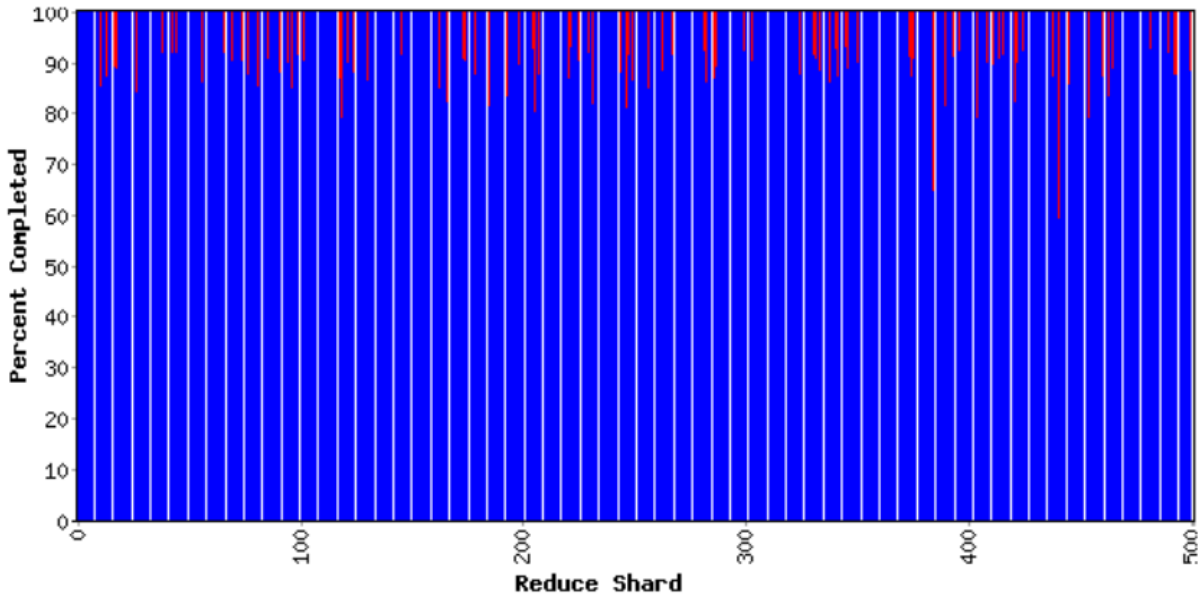
Variable	Minute
Mapped (MB/s)	0.0
Shuffle (MB/s)	0.0
Output (MB/s)	1222.0
doc-index-hits	0
docs-indexed	0
dups-in-index-merge	0
mr-merge-calls	51640600
mr-merge-outputs	51640600

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 37 min 01 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	500	0	523499.2	520468.6	520468.6
Reduce	500	406	94	520468.6	512265.2	514373.3



Counters

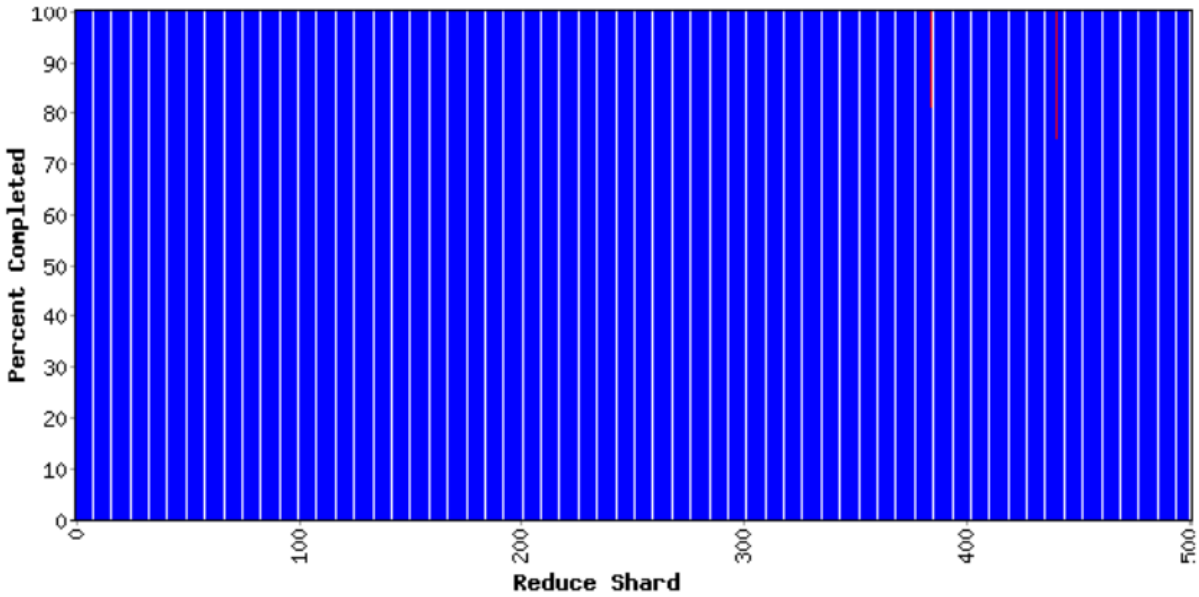
Variable	Minute	
Mapped (MB/s)	0.0	
Shuffle (MB/s)	0.0	
Output (MB/s)	849.5	
doc-index-hits	0	1(
docs-indexed	0	
dups-in-index-merge	0	
mr-merge-calls	35083350	
mr-merge-outputs	35083350	

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 38 min 56 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	500	0	523499.2	519781.8	519781.8
Reduce	500	498	2	519781.8	519394.7	519440.7



Counters

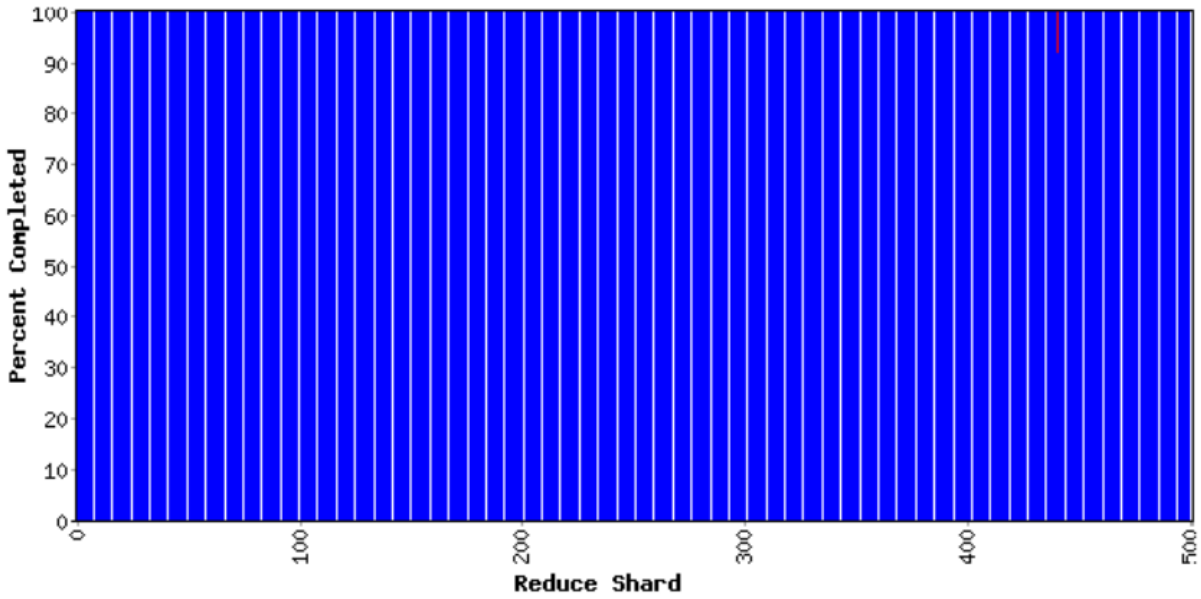
Variable	Minute	
Mapped (MB/s)	0.0	
Shuffle (MB/s)	0.0	
Output (MB/s)	9.4	
doc-index-hits	0	1056
docs-indexed	0	1
dups-in-index-merge	0	
mr-merge-calls	394792	1
mr-merge-outputs	394792	1

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 40 min 43 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	500	0	523499.2	519774.3	519774.3
Reduce	500	499	1	519774.3	519735.2	519764.0



Counters

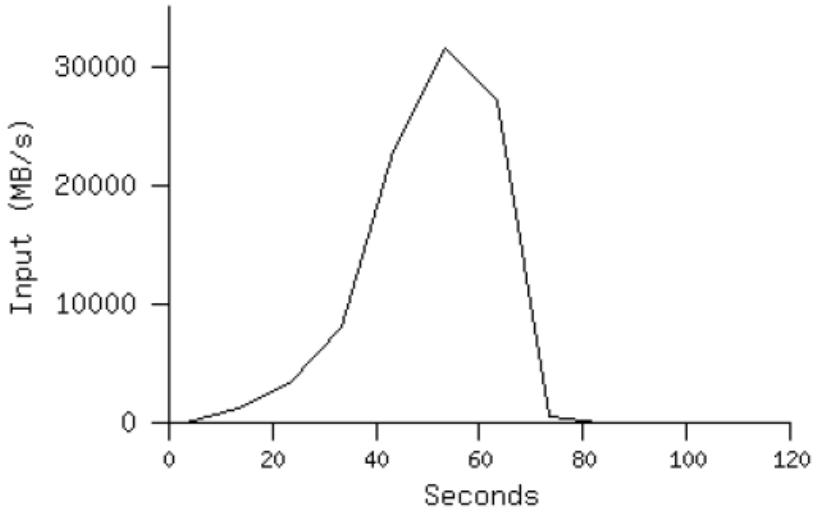
Variable	Minute	
Mapped (MB/s)	0.0	
Shuffle (MB/s)	0.0	
Output (MB/s)	1.9	
doc-index-hits	0	105
docs-indexed	0	
dups-in-index-merge	0	
mr-merge-calls	73442	
mr-merge-outputs	73442	

性能测试

- 1800个机器的集群
  - 4 GB内存
  - Dual-processor 2 GHz Xeons with Hyperthreading
  - Dual 160 GB IDE disks
  - Gigabit Ethernet per machine
- 两个基准测试程序
  - MR_GrepScan： 10^{10}个100字节的记录，从中找出符合特定模式的记录(92K个)
  - MR_SortSort： 10^{10}个100字节的记录按照TeraSort基准测试程序的方式排序

MR_Grep

在150秒之内处理1TB数据

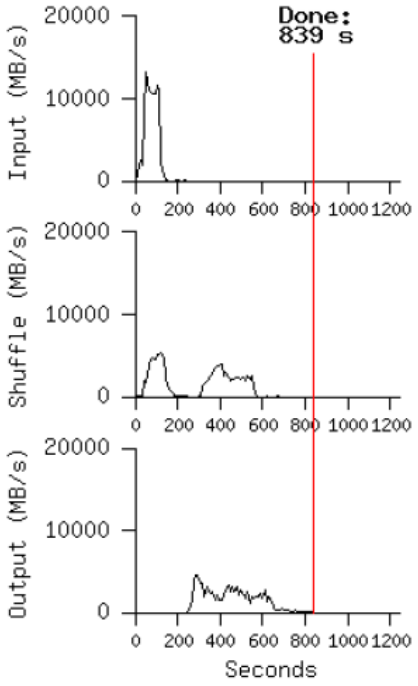


- 本地化的作用
  - 1800个机器读1TB数据的峰速为大约31GB/s
  - 如果不做本地化，只能达到网络的10GB/s限制
- 启动的额外开销很大
  - 计算总共花了150秒，其中有1分钟的启动时间

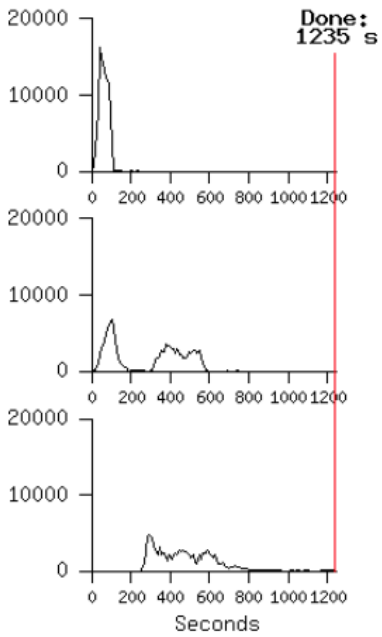
MR_Sort

在14分钟内完成了对1TB记录的排序

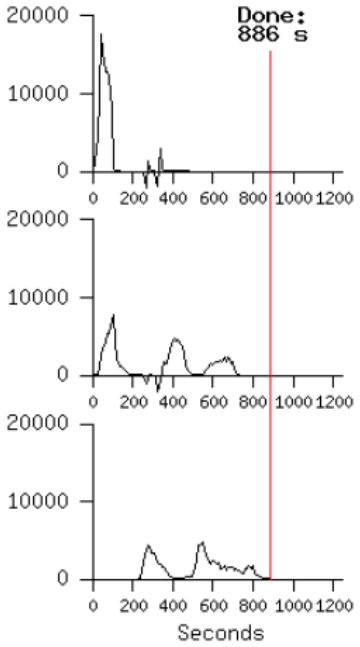
Normal



No backup tasks



200 processes killed



后备任务减少了执行时间　系统的容错性好

Usage: MapReduce jobs run in August 2004

- 作业数量 29,423
- 作业平均执行时间 634 secs
- 总计机器天数 79,186 days

- 输入数据 3,288 TB
- 产生的中间数据 758 TB
- 输出数据 193 TB

- 每个作业所调用的机器数量（平均） 157
- 每个作业遇到工作线程问题次数（平均） 1.2
- 每个作业的Map任务数（平均） 3,351
- 每个作业的Reduce任务数（平均） 55

Outline

- MapReduce编程模型
  - 算法
  - Shuffle和Sort
  - 实测数据
- **Hadoop**
  - **HDFS**
  - **HBase**
- 实例（天文交叉证认计算）

hadoop

Doug Cutting

- Hadoop是一个分布式系统基础架构，由Apache基金会开发。用户可以在不了解分布式底层细节的情况下，开发分布式程序。充分利用集群的威力高速运算和存储。
- 由Doug Cutting创建。
  - Apache Lucene: 开源搜索引擎
  - Apache Nutch: 基于Lucene的Web搜索实现
- 是对Google所提出的MapReduce、GFS、BigTable等模型的一个开源实现。

MapReduce实现：hadoop

Google calls it:	Hadoop equivalent:
MapReduce	Hadoop
GFS	HDFS
Bigtable	HBase
Chubby	(nothing yet… but planned)

Hadoop的组成部分

- Hadoop Common
  - The common utilities that support the other Hadoop subprojects.
- Avro
  - A data serialization system that provides dynamic integration with scripting languages.
- Chukwa
  - A data collection system for managing large distributed systems.
- Hbase
  - A scalable, distributed database that supports structured data storage for large tables.
- HDFS
  - A distributed file system that provides high throughput access to application data.
- Hive
  - A data warehouse infrastructure that provides data summarization and ad hoc querying.
- MapReduce
  - A software framework for distributed processing of large data sets on compute clusters.
- Pig
  - A high-level data-flow language and execution framework for parallel computation.
- ZooKeeper
  - A high-performance coordination service for distributed applications.

HDFS文件系统的特征

- 存储极大数目的信息（terabytes or petabytes），将数据保存到大量的节点当中。支持很大单个文件。
- 提供数据的高可靠性，单个或者多个节点不工作，对系统不会造成任何影响，数据仍然可用。
- 提供对这些信息的快速访问，并提供可扩展的方式。能够通过简单加入更多服务器的方式就能够服务更多的客户端。
- HDFS是针对MapReduce设计的，使得数据尽可能根据其本地局部性进行访问与计算。

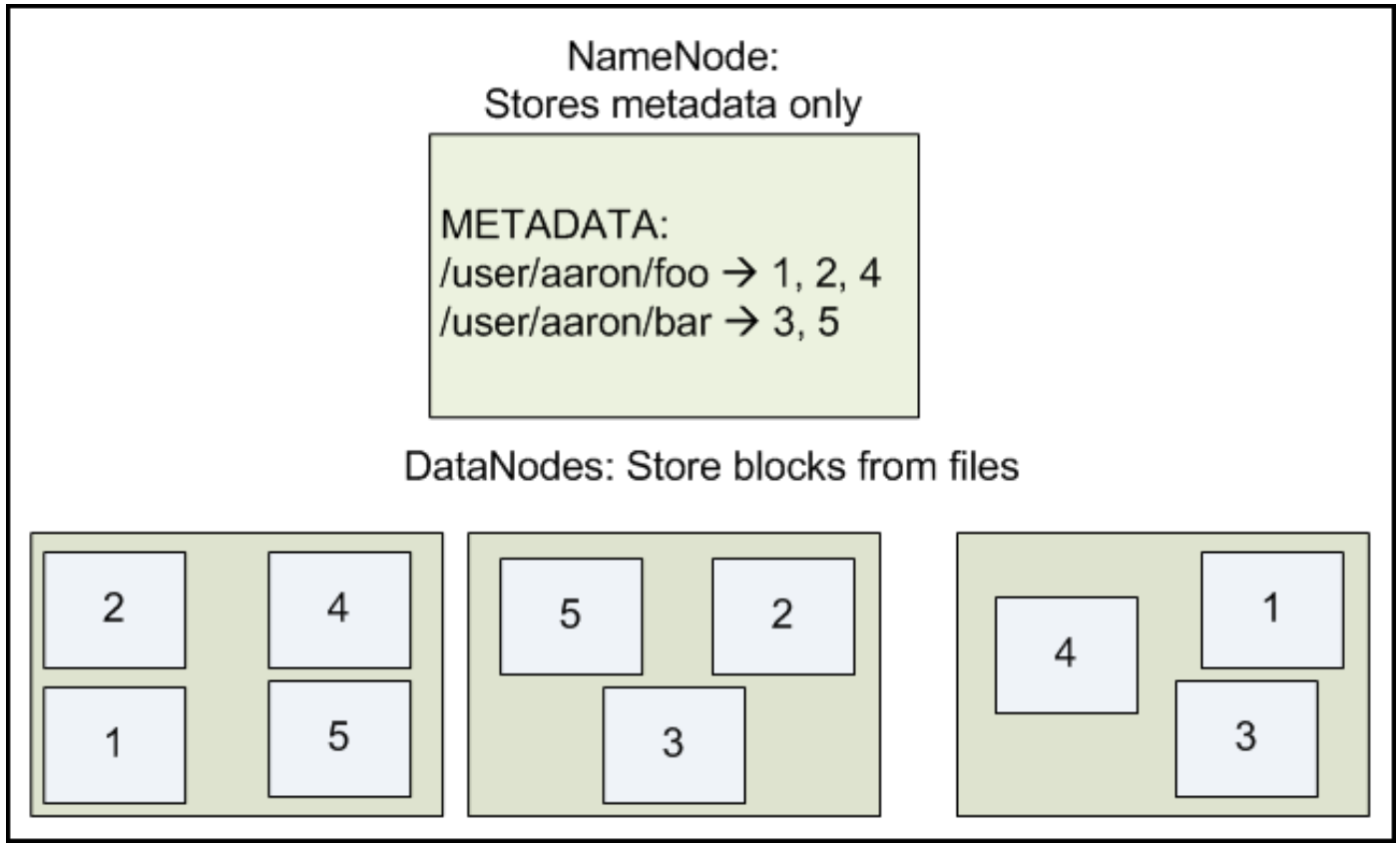
HDFS适应的场景

- 大量地从文件中顺序读。HDFS对顺序读进行了优化，代价是对于随机的访问负载较高。
- 数据支持一次写入，多次读取。对于已经形成的数据的更新不支持。
- 数据不进行本地缓存（文件很大，且顺序读没有局部性）
- 任何一台服务器都有可能失效，需要通过大量的数据复制使得性能不会受到大的影响。

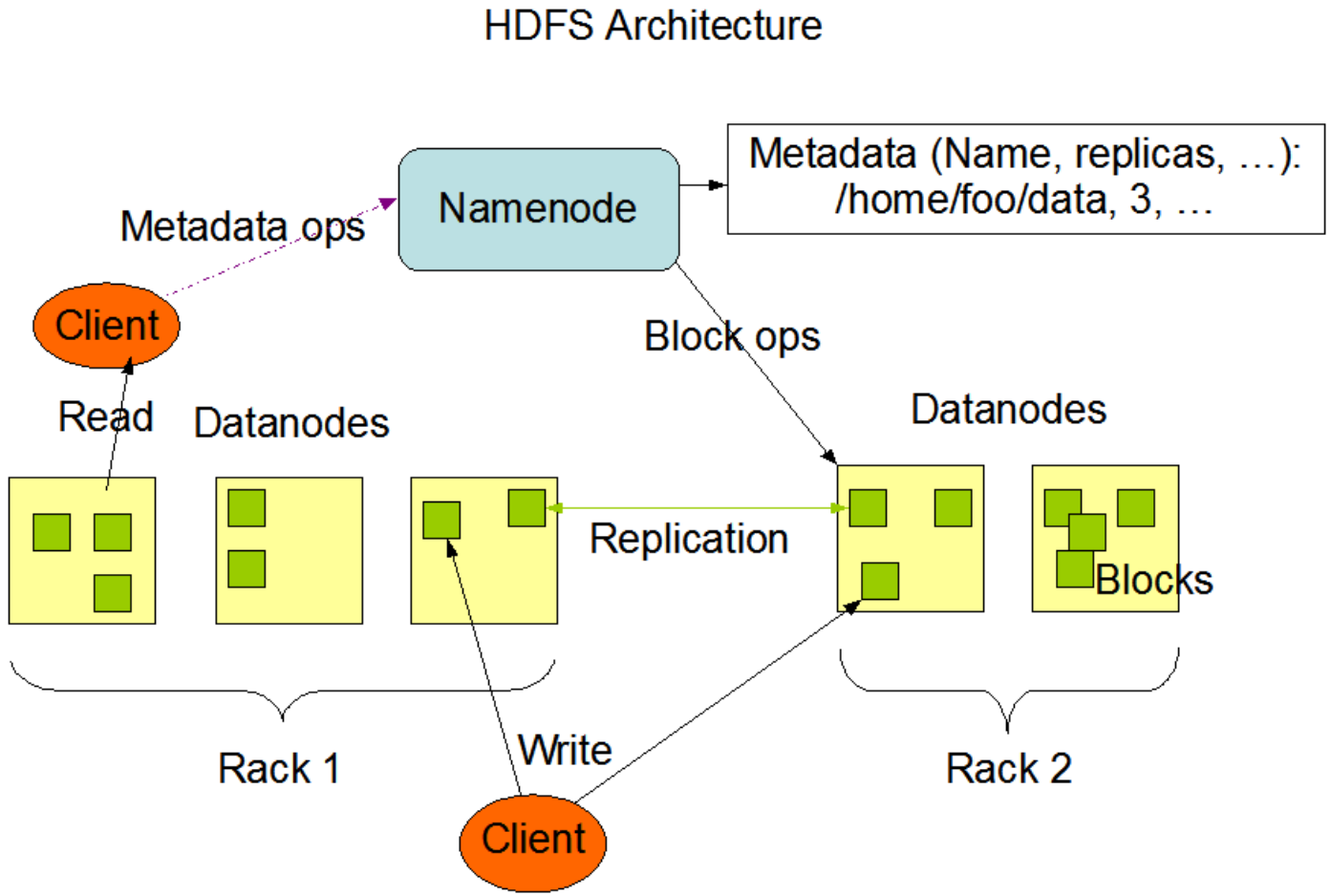
HDFS的设计

- 基于块的文件存储
- 块进行复制的形式放置，按照块的方式随机选择存储节点
- 副本的默认数目是3
- 默认的块的大小是64MB
  - 减少元数据的量
  - 有利于顺序读写（在磁盘上数据顺序存放）
- 适合于 MapReduce应用程序
- 依据Google File System的设计编写

HDFS数据分布设计


NameNode:
Stores metadata only
METADATA:
/user/aaron/foo → 1, 2, 4
/user/aaron/bar → 3, 5
DataNodes: Store blocks from files
2
4
1
5
5
2
3
4
1
3

Hadoop文件系统HDFS



HBase

- 分布式开源数据库，基于HDFS
- 可以处理非常庞大的表（数10亿行*几百万列）
- 也可以使用本地文件系统
- 等同于Google的BigTable

HBase概念视图与物理视图

Row Key	Time Stamp	Column *"contents:"*	Column *"anchor:"*		Column *"mime:"*
"com.cnn.www"	t9		"anchor:cnnsi.com"	"CNN"	
	t8		"anchor:my.look.ca"	"CNN.com"	
	t6	"<html>..."			"text/html"
	t5	"<html>..."			
	t3	"<html>..."			

Row Key	Time Stamp	Column *"contents:"*
"com.cnn.www"	t6	"<html>..."
	t5	"<html>..."
	t3	"<html>..."

Row Key	Time Stamp	Column *"mime:"*
"com.cnn.www"	t6	"text/html"

Row Key	Time Stamp	Column *"anchor:"*	
"com.cnn.www"	t9	"anchor:cnnsi.com"	"CNN"
	t8	"anchor:my.look.ca"	"CNN.com"

Outline

- MapReduce编程模型
  - 算法
  - Shuffle和Sort
  - 实测数据
- Hadoop
  - HDFS
  - HBase
- 实例（天文交叉证认计算）

实例：天文交叉证认计算

- 天文交叉证认
  - 在不同望远镜的观测数据中，或者同一望远镜不同时间的观测数据中，找到同一天体的相关数据。

计算原理及数据

- 实验数据：SDSS(1亿) 2MASS（4.7亿）
- 证认公式： Spatial Join

$$d = \sqrt{((RA\_A - RA\_B)\cos((DEC\_A + DEC\_B)/2)^2 + (DEC\_A - DEC\_B)2} < 3\sqrt{r_1^2 + r_2^2}$$

- 结果目标：

SDSS_ID	Twomass_ID	Distance
587731512617271364	02595905+0000200	5.243e-05
587731512617271365	02595905+0000200	6.55e-05
587731513154076828	02593768+0012219	3.2e-05
587731513154077269	02593768+0012219	0.0025043169

原始实现方法

- 复杂度：n*n —— unacceptable
- 解决办法：
  - 画框，过滤范围

$$|RA\_A - RA\_B| < \frac{|r_1| + |r_2|}{\cos((DEC\_A + DEC\_B)/2)}$$

$$|DEC\_A - DEC\_B| < |r_1| + |r_2|$$

  - 建立高效索引，提高读取速度

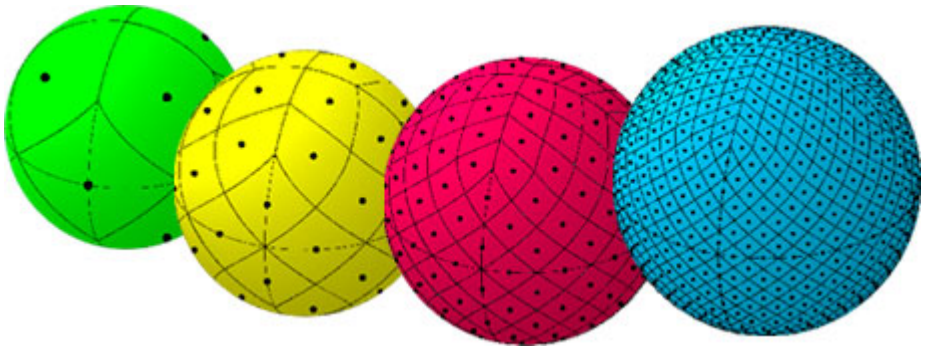
    Healpix & HTM

  - 数据划分、实现并行化

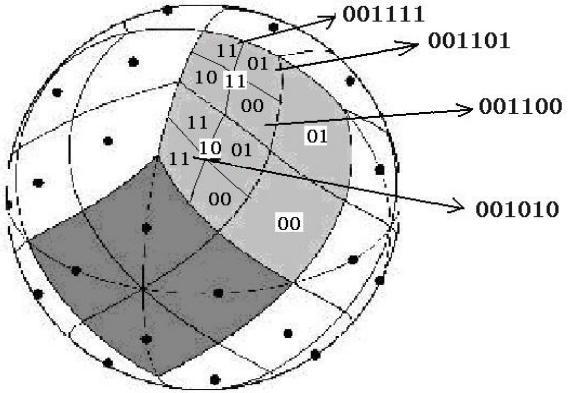
    消息传递型MPI

球面数据索引方式——HEALPix

- HEALPix —— **H**ierarchical **E**qual **A**rea iso**L**atitude **Pix**elization of a sphere.

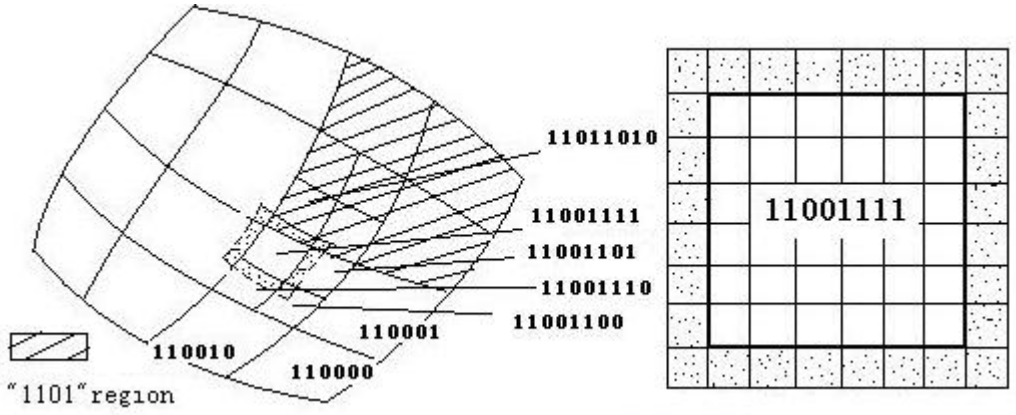


- *Quadtree* pixel numbering



解决边缘数据问题

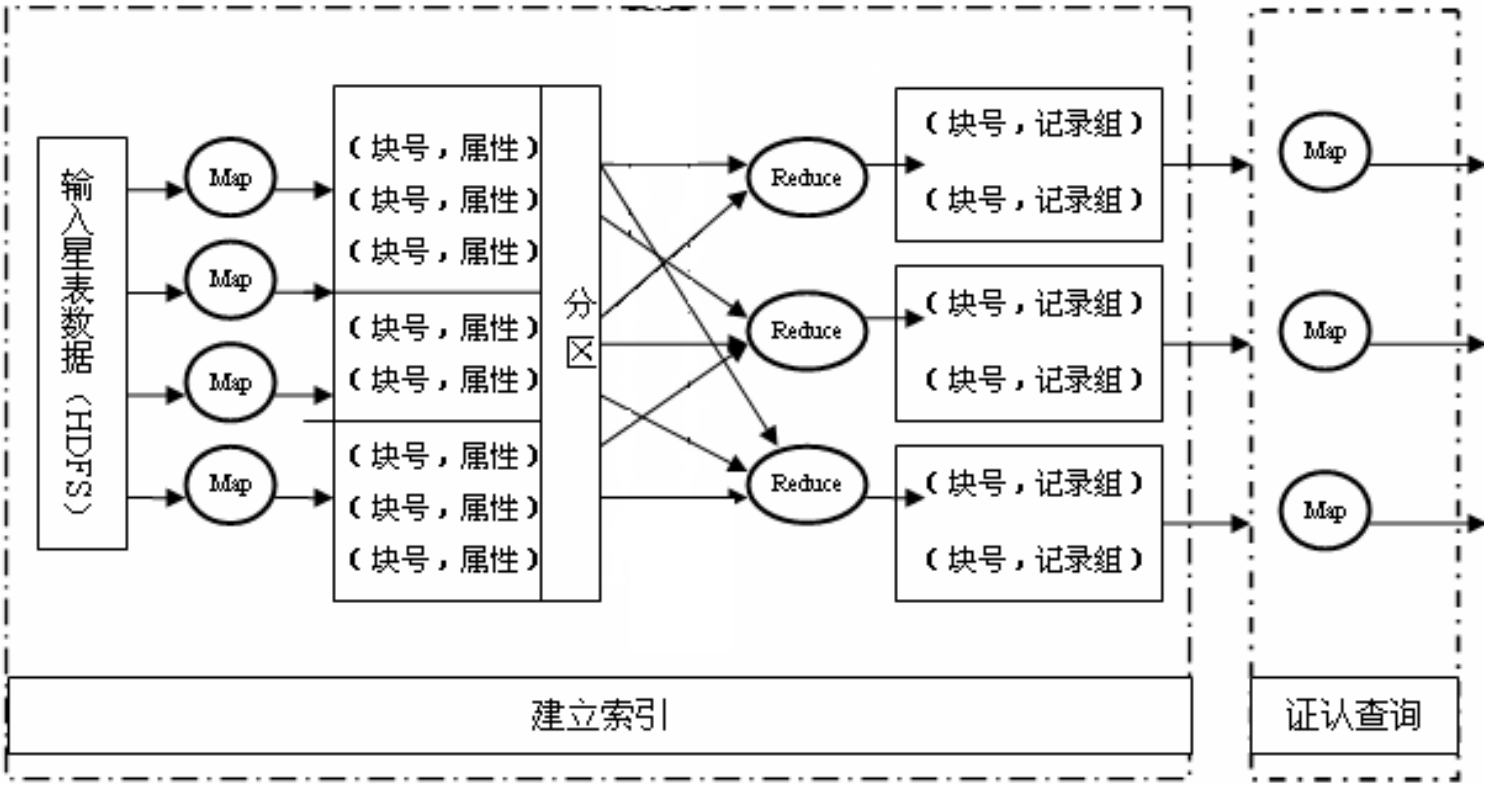
- 解决边缘数据问题
  - A fast bitwise operation algorithms to deduce the neighbor blocks' index number



  - High Efficiency

    less then 1 second!

基于MapReduce的方法



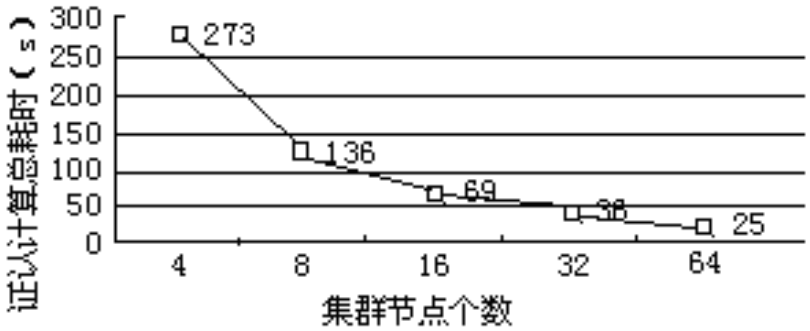
数据的分布式存放(包含Map、Reduce)

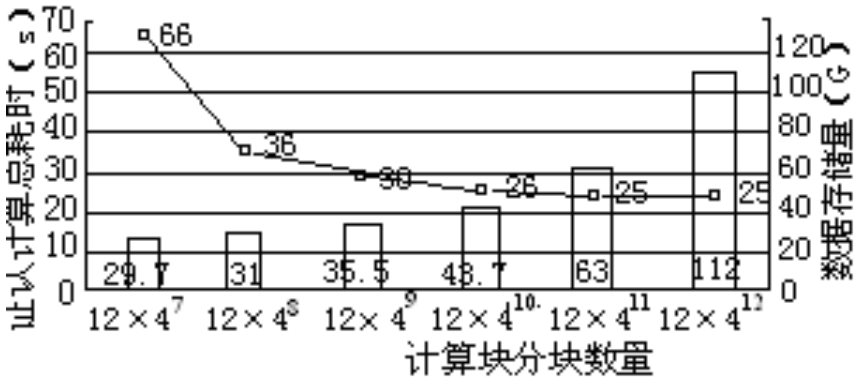
- 1. MapReduce系统自动切分输入数据，完成数据块分发；
- 2. 分布式地执行Map过程，对输入文件的每行，以“块号+代表星表来源的标志位”为key，将其他字段的信息以空格间隔作为value，输出<key, value>元组；
- 3. 按key值排序、分区，并分发给各个节点，使具有相同块号的记录连续存放，从而构成一个个计算块组。组内相同来源的数据也是连续存放的；
- 4. 各节点执行Reduce，统计各组基本信息，包括各组来自星表A和B各自的条数，作为该组的头文件输出。

证认计算（只包含一个Map过程）

- 各节点执行Map，对每一个计算块组，先读入头文件，如果来自两个星表中一方的数据量为0，则对该块数据不执行任何操作。否则，将两星表数据分别读入两个数组，计算两数组间的每两条数据间的距离，如果小于特定值则输出<ID_in_星表A +ID_in_星表B, 距离>元组。

性能测试结果





测试条件：

实验环境是64台普通PC，其中每台PC机配置如下：CPU：奔腾双核E2160，1800MHz，内存：2G
操作系统：Linux Ubuntu 9.04; Swap：2G
编程环境：JAVA、Hadoop

测试数据是美国斯隆数字巡天SDSS DR6的星表 和2微米巡天计划的星表，其数据量分别约为1亿条和4.7亿条。