MapReduce原理与应用

李轩　19212010022

2019/12/18

MapReduce介绍

1

为什么需要MapReduce

搜索引擎的存储和计算需求：

- 200亿个网页 x 网页内容20KB = 400+ TB
- 数据的存储
  - 存储这些网页需要大约1000台计算机
- 数据的计算
  - IO瓶颈：一台计算机的读取速度大约30 MB/sec

分布式大规模数据的计算的需求举例：

1. BI商务数据分析
2. 银行和运营商等企业的用户信息与历史数据的挖掘
3. YouTube、谷歌、淘宝等网站的搜索和推荐

。。。

这些任务大多都是不难实现的任务，但是由于输入的数据量很大，因此要想在可接受的时间内完成运算，只有将这些计算分布在成百上千的主机上。如何处理并行计算？

- **Idea:**
  - 即使我们有一台高性能服务器，但在硬盘读写数据的速度上还是硬伤，所以如果有多台电脑同时做读写，那可省下不少时间
  - 每台机器都做一部分本地的数据处理，然后再做汇总

- **Map-reduce**
  - MapReduce是Google提出的分布式并行计算框架
  - 能够将分布在多个节点的海量数据进行整合，充分利用每个节点的计算能力，每个节点并行处理数据，可完成PB级数据批处理。

大多数的运算都可以划分成若干次的Map-Reduce：在输入数据上应用 Map 操作得出一个 key-value pair 集合，然后在所有具有相同 key 值的 value 值上应用 Reduce 操作，得到结果。

使用 MapReduce 模型，只需用户实现根据自己业务设计 Map 和 Reduce 函数，我们就可以轻松实现大规模并行化计算，**而不必关心并行计算、容错、数据分布、负载均衡等复杂问题，只需要实现Map和Reduce接口。**

MapReduce原理

2

主要步骤

- Mapper
- Partitioner
- Combiner
- Reducer

Mapper

为每一个InputSplit产生一个map task

定义一个类继承 Mapper类

```
@Override
protected void map(LongWritable key, Text value, Context context)
```

其中 key是每一行的行数， value是每一行的内容
根据需要,完成对value的处理
在WordCount任务中，就是对每一行的value做一个根据分隔符分割

```
File 1 内容:
My name is Tony
My company is pivotal

File 2 内容:
My name is Lisa
My company is EMC
```

map

split 0:
My 1
name 1
is 1
Tony 1
My 1
company 1
is 1
Pivotal 1

split 1:
My 1
name 1
is 1
Lisa 1
My 1
company 1
is 1
EMC 1

Partitioner

Partitioner是MapReduce中非常重要的组件。Partitioner的作用是针对Mapper阶段的中间数据进行分片，然后将相同分片的数据交给同一个reducer处理。Partitioner过程其实就是Mapper阶段shuffle过程中关键的一部分。

hadoop中默认的partition是HashPartitioner。根据Mapper阶段输出的key的hashcode做划分：

```
1  public class HashPartitioner<K, V> extends Partitioner<K, V> {
2
3    /** Use {@link Object#hashCode()} to partition. */
4    public int getPartition(K key, V value,
5                            int numReduceTasks) {
6      return (key.hashCode() & Integer.MAX_VALUE) % numReduceTasks;
7    }
8
9  }
```

在很多场景中，我们是需要通过重写Partitioner来实现自己需求的。例如，我们有全国分省份的数据，我们经常需要将相同省份的数据输入到同一个文件中。这个时候，通过重写Partitioner就可以达到上面的目的。

Combiner

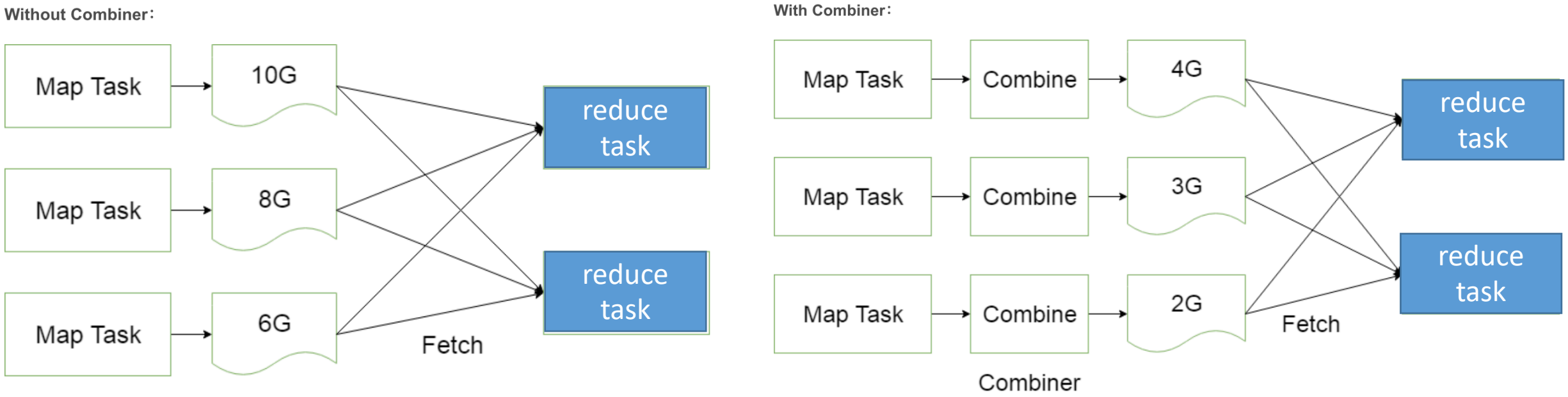
MapReduce框架中使用Mapper将数据处理成一个<key,value>键值对，在网络节点间对其进行整理(shuffle)，然后使用Reducer处理数据并进行最终输出。

问题：
1. 分布式系统中的传输代价高
2. 对Reduce端负载能力要求高

如果我们有10亿个数据，Mapper会生成10亿个键值对在网络间进行传输，但如果我们只是对数据求最大值，那么很明显的Mapper只需要输出它所知道的最大值即可。这样做不仅可以减轻网络压力，同样也可以大幅度提高程序效率。

Combiner

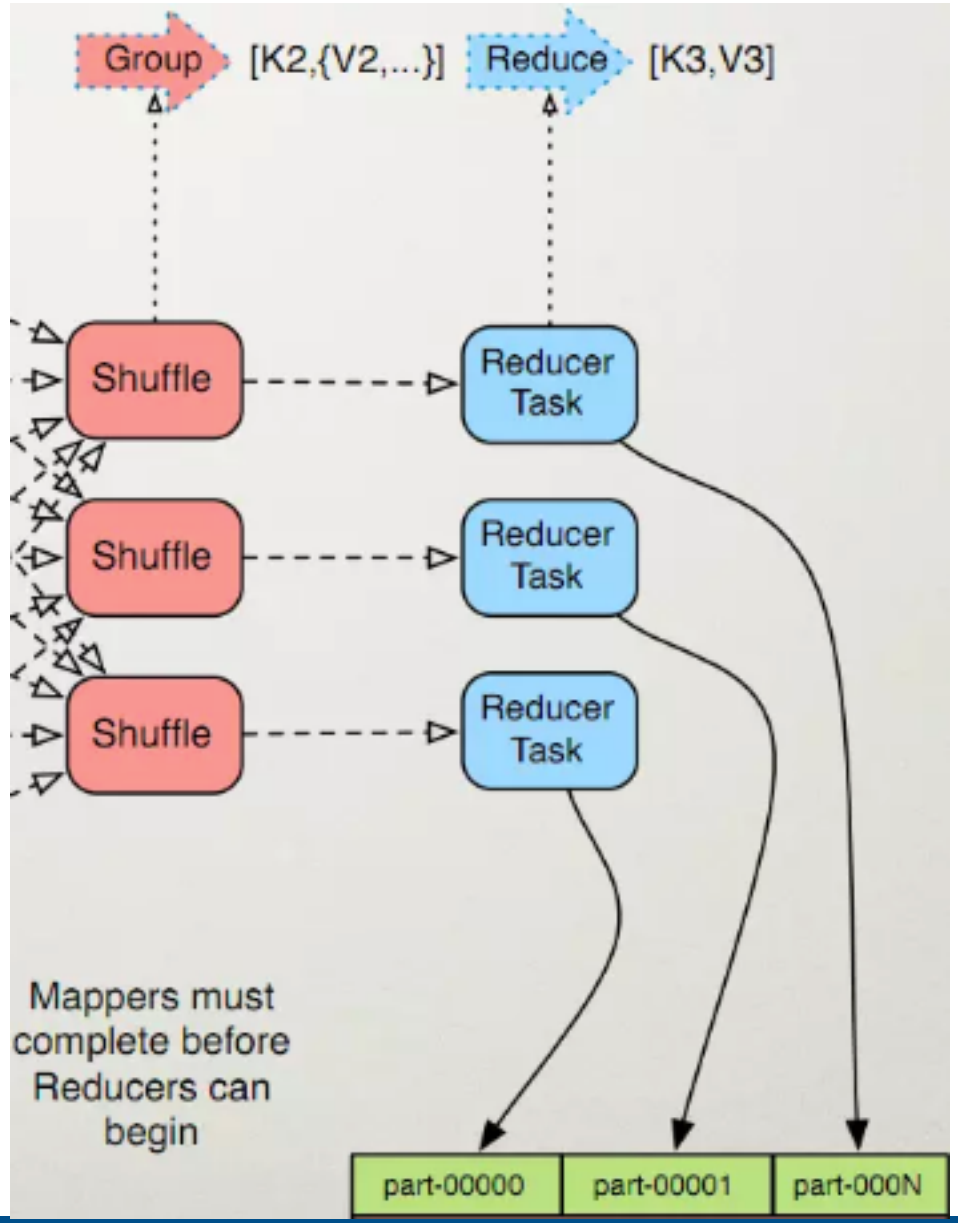
MapReduce中的Combiner就是为了避免map任务和reduce任务之间的海量数据传输而设置的，Hadoop允许用户针对map task的输出指定一个合并函数。即为了减少传输到Reduce中的数据量。它主要是为了合并Mapper的输出从而减少网络带宽和Reducer之上的负载。



combiner 相当于是在map端的reducer。

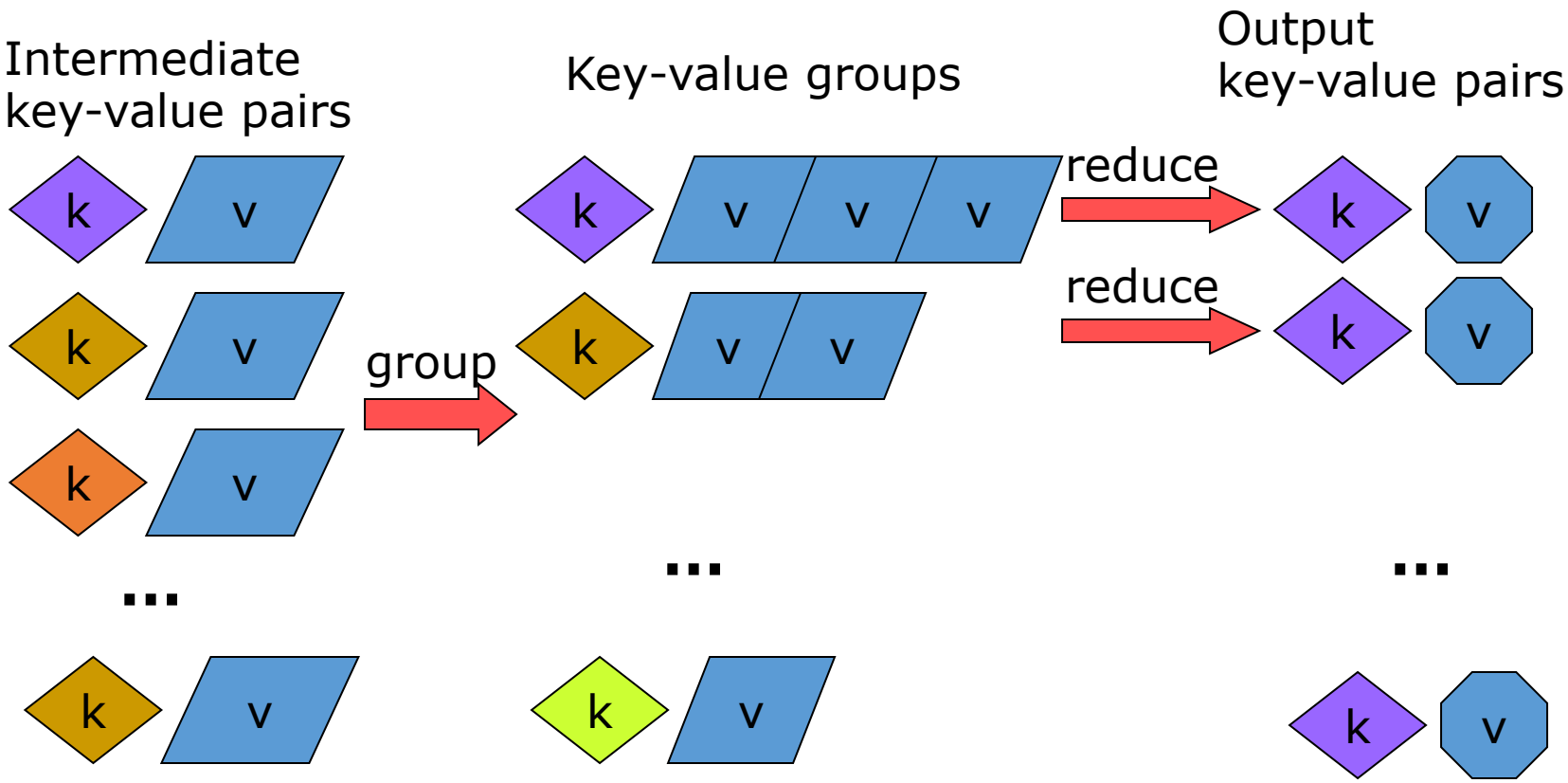
使用combiner需要自己调用， job.setCombinerClass(Combine.**class**)
如果combiner和reducer的功能是一样的，那么可以直接将reducer作为combiner

Reducer



框架为已分组的输入数据中的每个 <key, (list of values)>对调用一次Reduce。

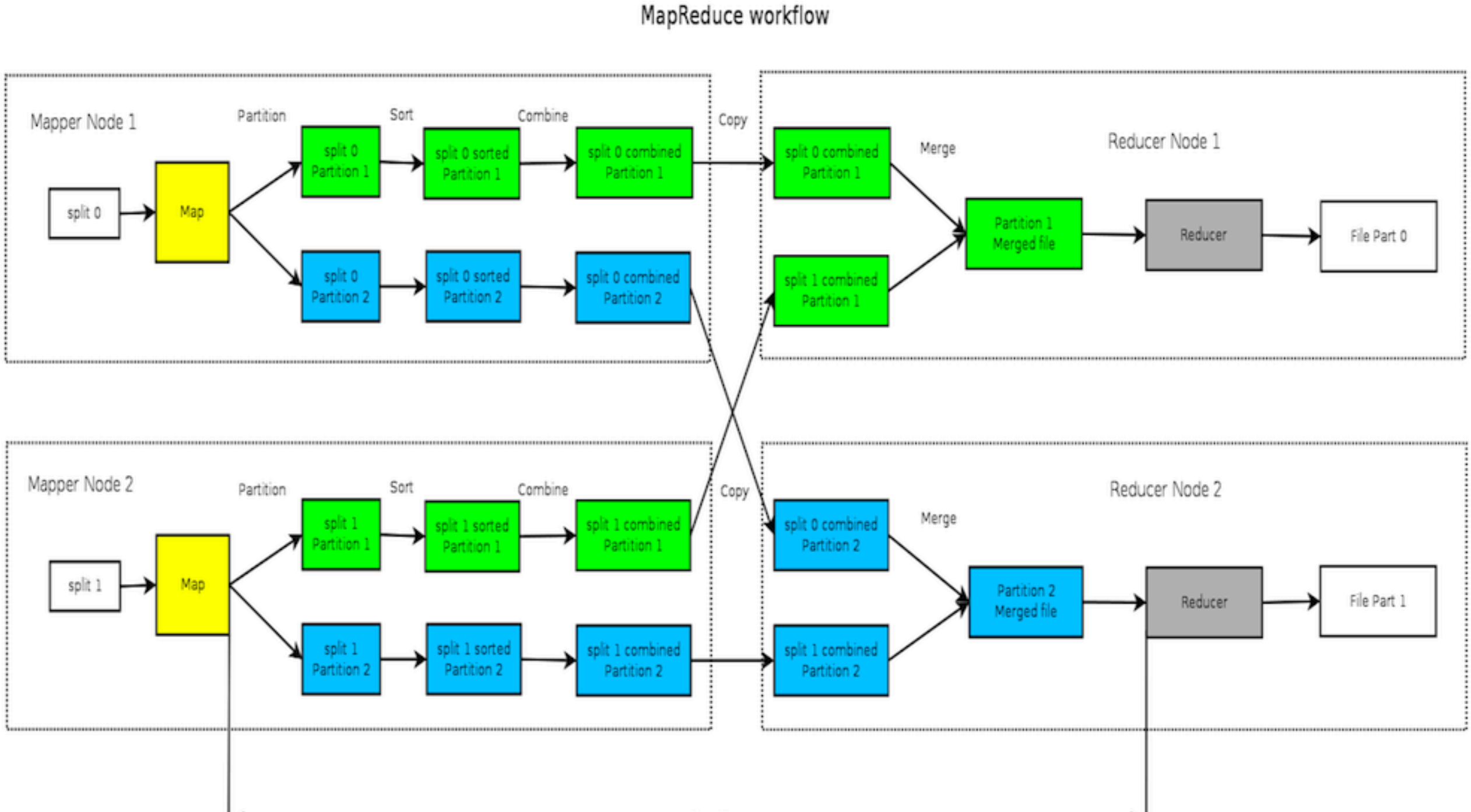
比如在wordcount中，reducer的输入是< word , [1,1,1,1,1,1] >，然后reduce根据需要完成的功能做相应的处理，比如求和，求平均，求最大值等等。



E.g.
(word, wordcount-in-a-doc)

(word, list-of-wordcount)
~ SQL Group by

(word, final-count)
~ SQL aggregation


MapReduce workflow
Mapper Node 1
Partition
Sort
Combine
Copy
split 0
Map
split 0 Partition 1
split 0 sorted Partition 1
split 0 combined Partition 1
split 0 Partition 2
split 0 sorted Partition 2
split 0 combined Partition 2
Reducer Node 1
Merge
split 0 combined Partition 1
split 1 combined Partition 1
Partition 1 Merged file
Reducer
File Part 0
Mapper Node 2
Partition
Sort
Combine
Copy
split 1
Map
split 1 Partition 1
split 1 sorted Partition 1
split 1 combined Partition 1
split 1 Partition 2
split 1 sorted Partition 2
split 1 combined Partition 2
Reducer Node 2
Merge
split 0 combined Partition 2
split 1 combined Partition 2
Partition 2 Merged file
Reducer
File Part 1

MapReduce应用

3

常见应用举例

分布式排序
分布式Grep
Top K
去重，计数统计
求数据的最大值、最小值和平均值等
数据表的关联
倒排索引
等

WordCount

```
File 1 内容:
My name is Tony
My company is pivotal

File 2 内容:
My name is Lisa
My company is EMC
```

map

split 0:

My 1
name 1
is 1
Tony 1
My 1
company 1
is 1
Pivotal 1

split 1:

My 1
name 1
is 1
Lisa 1
My 1
company 1
is 1
EMC 1

partition

split 0:

Partition 1:

company 1
is 1
is 1

Partition 2:

My 1
My 1
name 1
Pivotal 1
Tony 1

split 1:

Partition 1:

company 1
is 1
is 1
EMC 1

Partition 2:

My 1
My 1
name 1
Lisa 1

sort

split 0:

Partition 1:		Partition 2:	
company	1	My	1
is	1	My	1
is	1	name	1
		Pivotal	1
		Tony	1

split 1:

Partition 1:		Partition 2:	
company	1	Lisa	1
EMC	1	My	1
is	1	My	1
is	1	name	1

combine

split 0:

Partition 1:		Partition 2:	
company	1	My	2
is	2	name	1
		Pivotal	1
		Tony	1

split 1:

Partition 1:		Partition 2:	
company	1	Lisa	1
EMC	1	My	2
is	2	name	1

copy

Partition 1 :

split 0 :

company 1
is 2

split 1:

company 1
EMC 1
is 2

Partition 2:

split 0 :

My 2
name 1
Pivotal 1
Tony 1

split 1:

Lisa 1
My 2
name 1

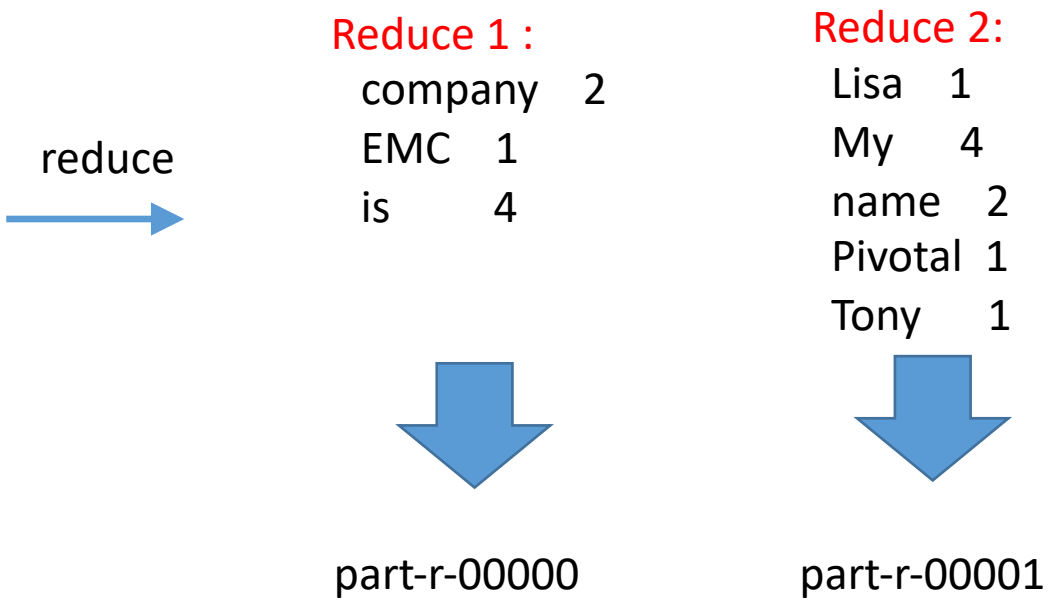
merge

Partition 1 :

company 1
company 1
EMC 1
is 2
is 2

Partition 2:

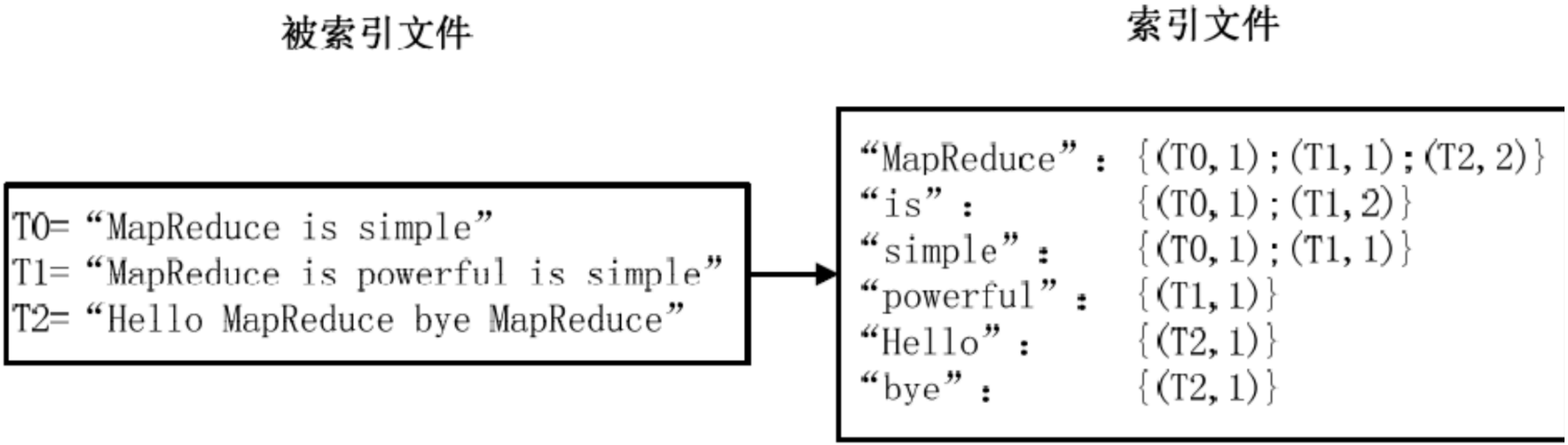
Lisa 1
My 2
My 2
name 1
name 1
Pivotal 1
Tony 1


reduce
Reduce 1 :
company 2
EMC 1
is 4
Reduce 2:
Lisa 1
My 4
name 2
Pivotal 1
Tony 1
part-r-00000
part-r-00001

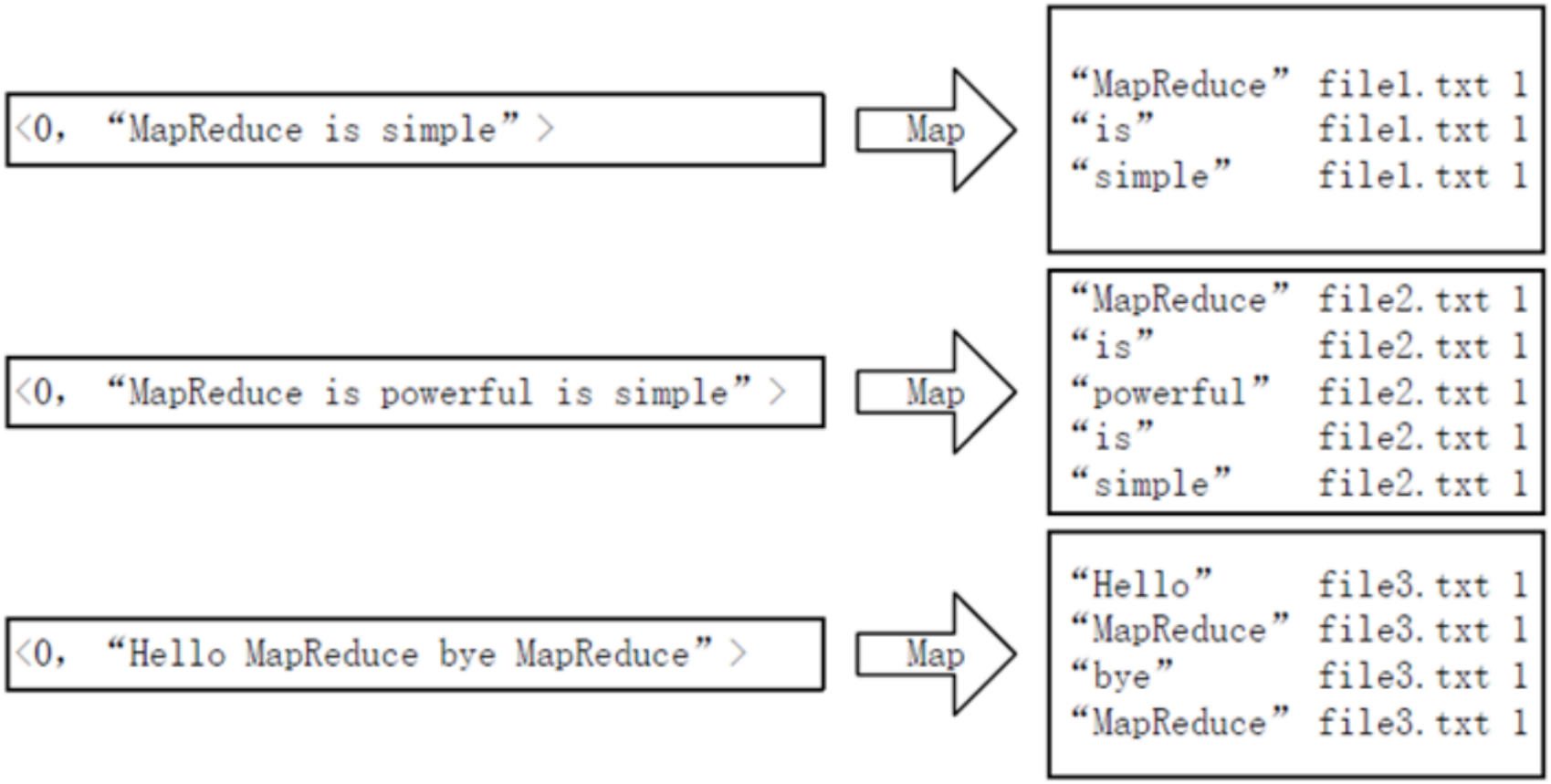
代码示例

倒排索引

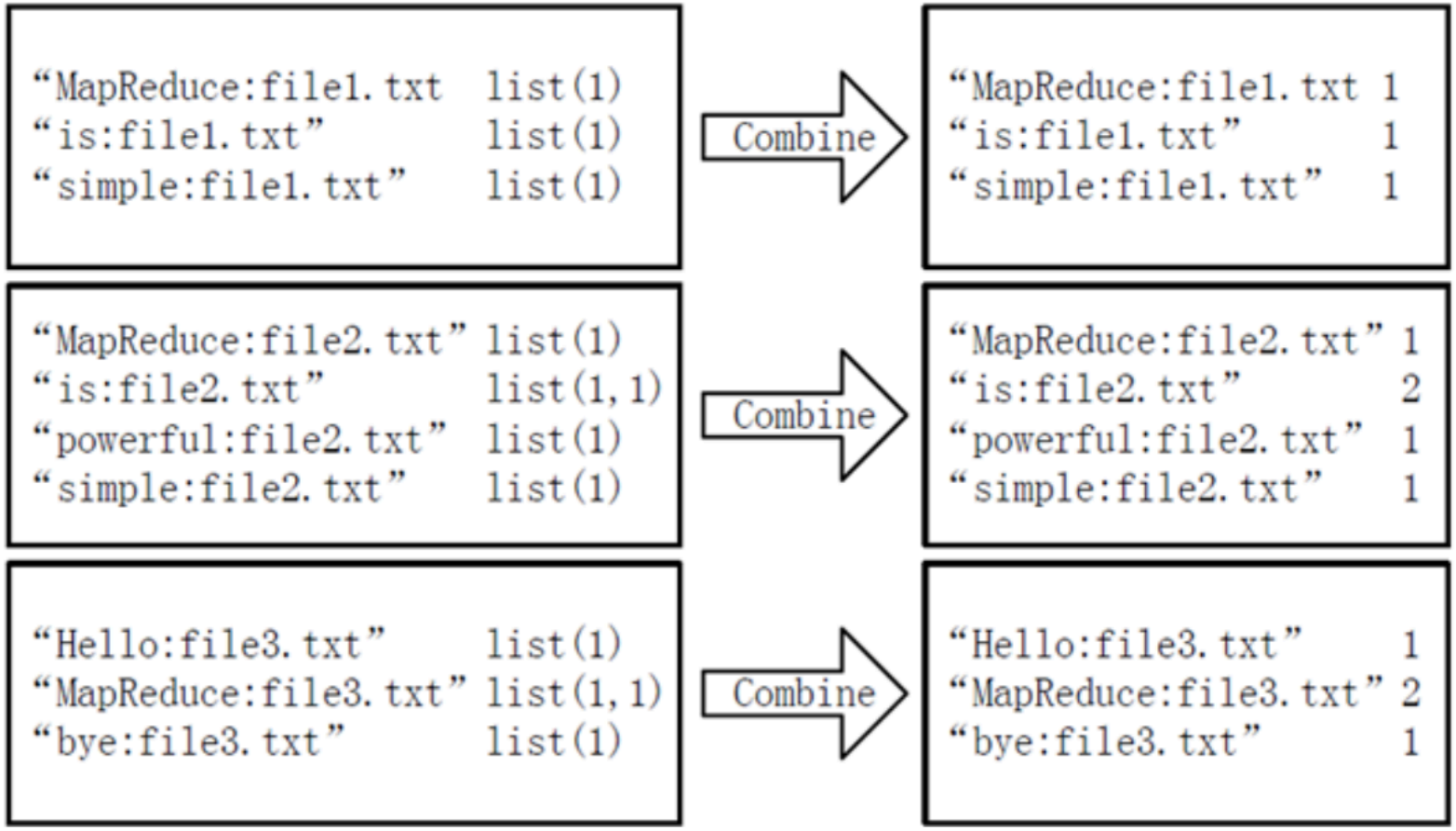
“倒排索引”是文档检索系统中最常用的数据结构，被广泛地应用于全文搜索引擎。它主要是用来存储某个单词（或词组）在一个文档或一组文档中的存储位置的映射，即提供了一种根据内容来查找文档的方式。由于不是根据文档来确定文档所包含的内容，而是进行相反的操作，因而称为倒排索引。



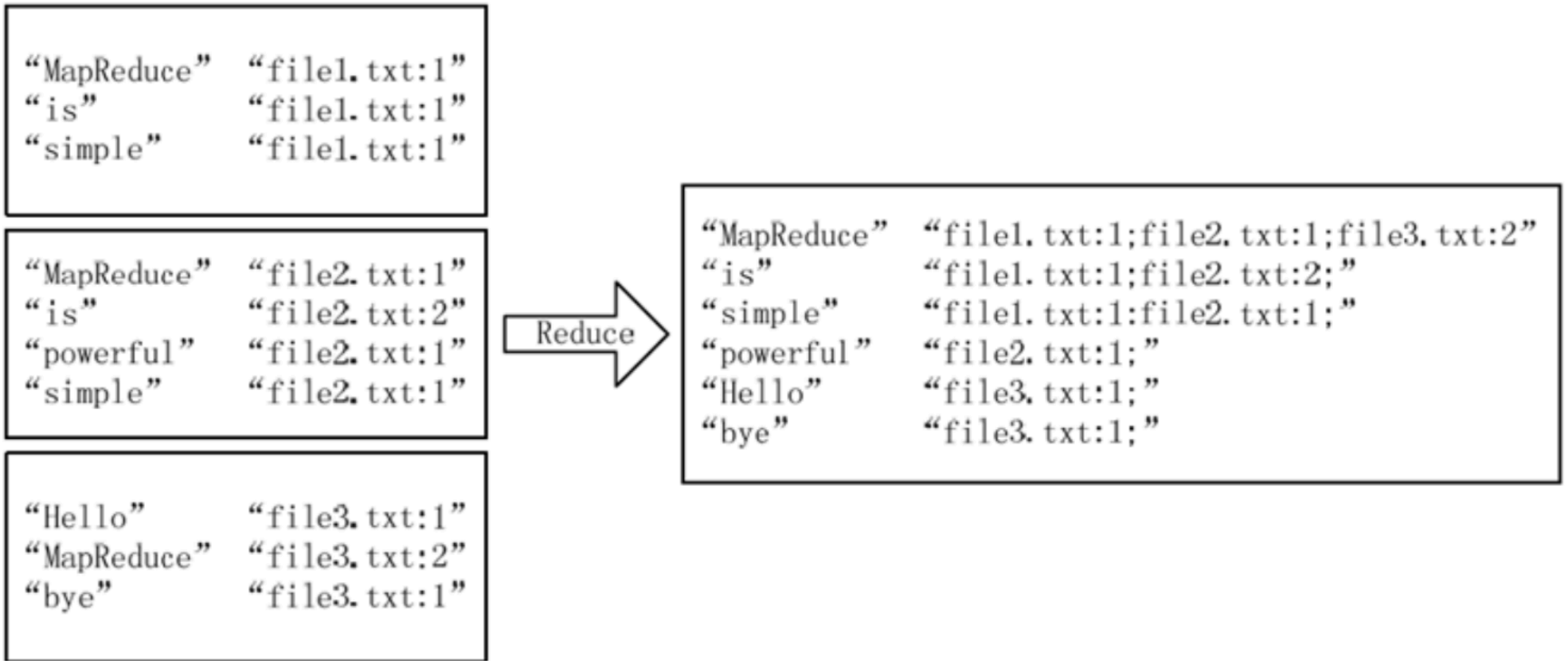
Map


<0, “MapReduce is simple”>
Map
“MapReduce” file1.txt 1
“is” file1.txt 1
“simple” file1.txt 1
<0, “MapReduce is powerful is simple”>
Map
“MapReduce” file2.txt 1
“is” file2.txt 1
“powerful” file2.txt 1
“is” file2.txt 1
“simple” file2.txt 1
<0, “Hello MapReduce bye MapReduce”>
Map
“Hello” file3.txt 1
“MapReduce” file3.txt 1
“bye” file3.txt 1
“MapReduce” file3.txt 1

Combine

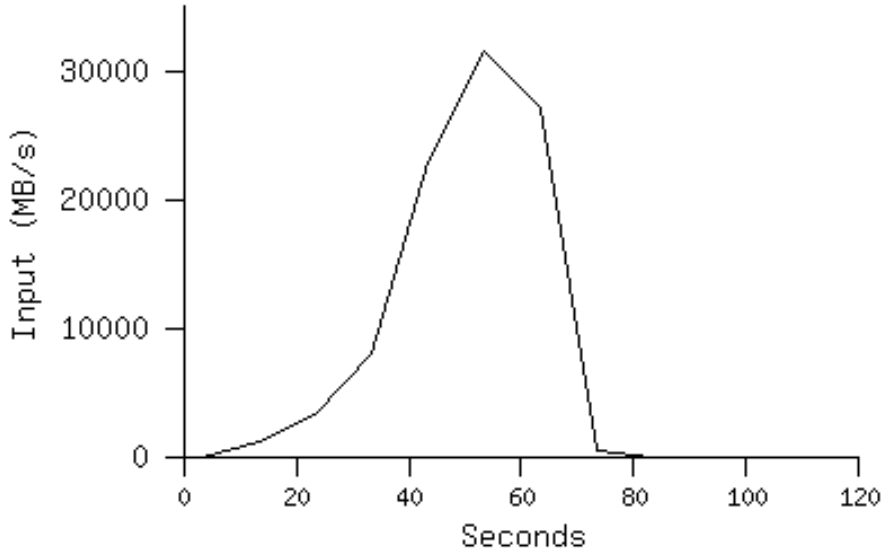

"MapReduce:file1.txt list(1)
"is:file1.txt" list(1)
"simple:file1.txt" list(1)
Combine
"MapReduce:file1.txt 1
"is:file1.txt" 1
"simple:file1.txt" 1
"MapReduce:file2.txt" list(1)
"is:file2.txt" list(1,1)
"powerful:file2.txt" list(1)
"simple:file2.txt" list(1)
Combine
"MapReduce:file2.txt" 1
"is:file2.txt" 2
"powerful:file2.txt" 1
"simple:file2.txt" 1
"Hello:file3.txt" list(1)
"MapReduce:file3.txt" list(1,1)
"bye:file3.txt" list(1)
Combine
"Hello:file3.txt" 1
"MapReduce:file3.txt" 2
"bye:file3.txt" 1

Reduce


"MapReduce" "file1.txt:1"
"is" "file1.txt:1"
"simple" "file1.txt:1"
"MapReduce" "file2.txt:1"
"is" "file2.txt:2"
"powerful" "file2.txt:1"
"simple" "file2.txt:1"
"Hello" "file3.txt:1"
"MapReduce" "file3.txt:2"
"bye" "file3.txt:1"
Reduce
"MapReduce" "file1.txt:1;file2.txt:1;file3.txt:2"
"is" "file1.txt:1;file2.txt:2;"
"simple" "file1.txt:1:file2.txt:1;"
"powerful" "file2.txt:1;"
"Hello" "file3.txt:1;"
"bye" "file3.txt:1;"

代码示例

Performance – MR_Grep



- Locality optimization helps:
  - 1800 machines read 1 TB of data at peak of ~31GB/s
  - Without this, rack switches would limit to 10 GB/s
- Startup overhead is significant for short jobs

Apache Hadoop Ecosystem

Ambari
Provisioning, Managing and Monitoring Hadoop Clusters

Sqoop
Data Exchange

Flume
Log Collector

Zookeeper
Coordination

Oozie
Workflow

Pig
Scripting

Mahout
Machine Learning

R Connectors
Statistics

Hive
SQL Query

Hbase
Columnar Store

YARN Map Reduce v2
Distributed Processing Framework

HDFS
Hadoop Distributed File System

Conclusion

优点：

- 可以使用廉价的商用机器搭建集群完成大规模分布式计算
- MapReduce 隐藏分布式计算编程中各种复杂问题，不必关心并行计算、容错、数据分布、负载均衡等细节
- MapReduce 提供一个简单的编程接口
  - 一般只需实现Map 和 Reduce接口
  - Hive :SQL -> MapReudce Task

缺点：

- 某些复杂的运算无法分解成Map和Reduce task实现
- 基于磁盘的MapReduce较慢，一般用于离线计算
- 每一个task都要来回读写数据


THANKS