MapReduce基础算法程序设计 (II)

摘要

- MapReduce排序算法
- MapReduce二级排序
- MapReduce单词同现分析算法
- MapReduce文档倒排索引算法
- MapReduce专利文献数据分析

MapReduce排序算法

- Data Size
  - 10MB? 10GB? 1000GB?
- Sort Algorithm in MapReduce
  - map(k1, *) -> (k1, *)   // Identity function
  - shuffle and sort
    - (1) total-order partitioning
    - (2) local sorting
  - reduce(k1, *) -> (k1, *)   // Identity function
- A customized total-order *Partitioner*
  - recall that shuffle phase needs a *Partitioner* to partition the key space
- InputFormat, OutputFormat
  - that depends on your data format

MapReduce排序算法

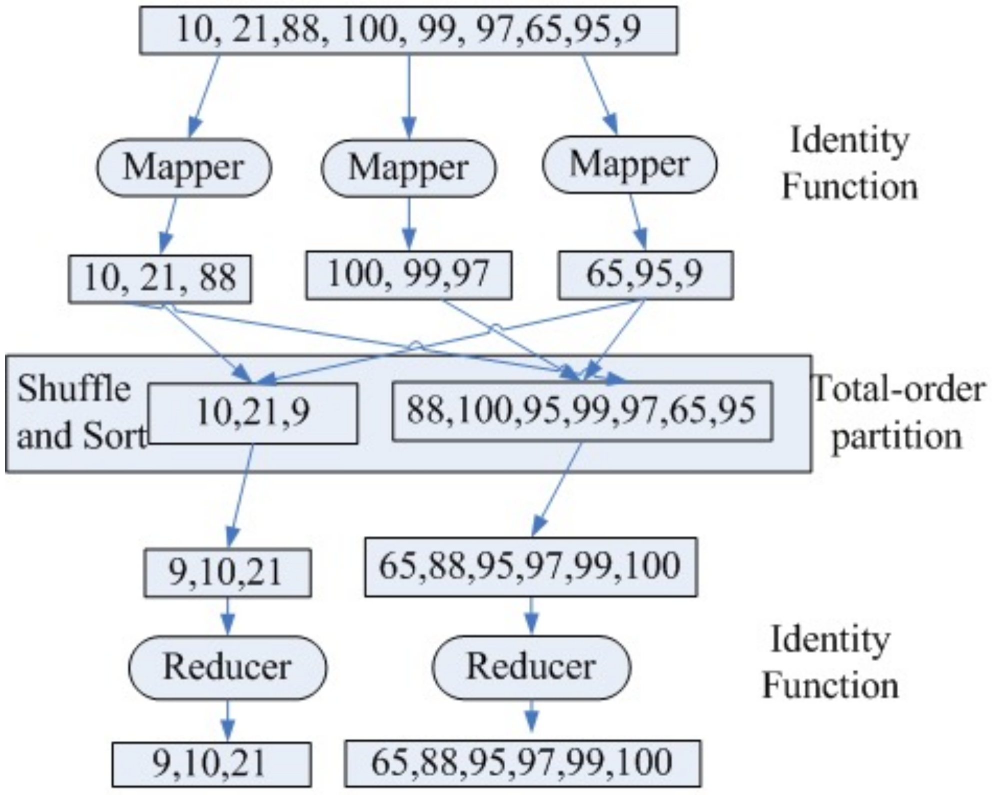


Figure: A simple MapReduce sorting example with 3 mappers, 2 reducers

Is there any problem here?

MapReduce排序算法

- Partitioner
  - 两个问题
    - (1) 如何避免在某些Reducer上聚集过多的数据而拖慢了整个程序
    - (2) 当有大量的key要分配到多个partition（也就是Reducer）时，如何高效地找到每个Key所属的partition
  - 对Partitioner的要求
    - 划分均匀
    - 查找快速
  - Thank God ☺
    - There exists a class, ***TotalOrderPartitioner*** in hadoop libs, which was originally used in TeraSort.

MapReduce排序算法

- TeraSort
  - In May 2008, running on a 910-node cluster, Hadoop sorted the 10 billion records (1 TB in size) in 209 seconds (3.48 minutes) to win the annual general purpose terabyte sort benchmark.
  - The cluster statistics were:
    - 910 nodes
    - 4 dual core Xeons @ 2.0ghz per a node
    - 4 SATA disks per a node
    - 8G RAM per a node
    - 1 gigabit Ethernet on each node
    - Red Hat Enterprise Linux Server Release 5.1 (kernel 2.6.18)
    - Sun Java JDK 1.6.0_05-b13
  - In May 2009, it was announced that a team at Yahoo! used Hadoop to sort one terabyte in 62 seconds.
  - Package org.apache.hadoop.examples.terasort

MapReduce排序算法

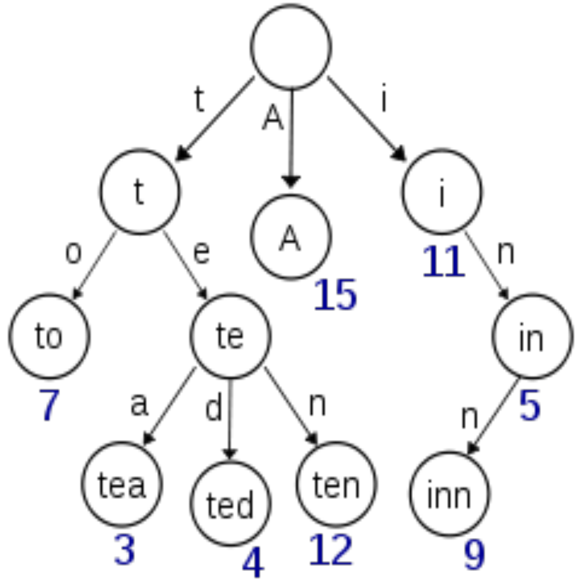
- TotalOrderPartitioner for TeraSort
  - TotalOrderPartitioner
    - 一个提供全序划分的*Partitioner*
    - 从Hadoop v0.19.0开始正式发布在库类中
    - org.apache.hadoop.mapreduce.lib.partition.TotalOrderPartitioner<K,V>
  - 为满足两个要求所采用的策略
    - 通过采样获取数据的分布
    - 构建高效的划分模型

MapReduce排序算法

- TotalOrderPartitioner
  - 获取数据分布作均匀划分
    - Key 的分布未知
    - 预读一小部分数据采样(sample)
    - 对采样数据排序后均分，假设有N个reducer，则取得N-1个分割点
    - uses a sorted list of *N-1* sampled keys that define the key range for each reduce.
    - In particular, all keys such that ***sample[i-1] <= key < sample[i]*** are sent to reduce *i*. This guarantees that the output of reduce *i* are all less than the output of reduce *i+1*.
  - Example
    - 设reduce数目为3，采到9条记录：1,22,55,60,62,66,68,70,90
    - 取两个分割点60,68；划分区间为：[*,60), [60, 68), [68,*)

MapReduce排序算法

- TotalOrderPartitioner
  - 高效的划分模型
    - 若Key 的数据类型是BinaryComparable的，即两个对象的可以直接按字节比较大小（如Text），则以key构造Trie Tree；否则以二分查找来确定key的所属区间
    - Trie Tree，一种高效的适于查找的数据结构
    - The partitioner builds **a two level trie** that quickly indexes into the list of sample keys based on the first two bytes of the key. (ref: hadoop docs)
    - 两级的Trie可以最多对应大约256*256个reducer，通常是足够的



A trie for keys "A","to", "tea", "ted", "ten", "i", "in", and "inn".

MapReduce二级排序

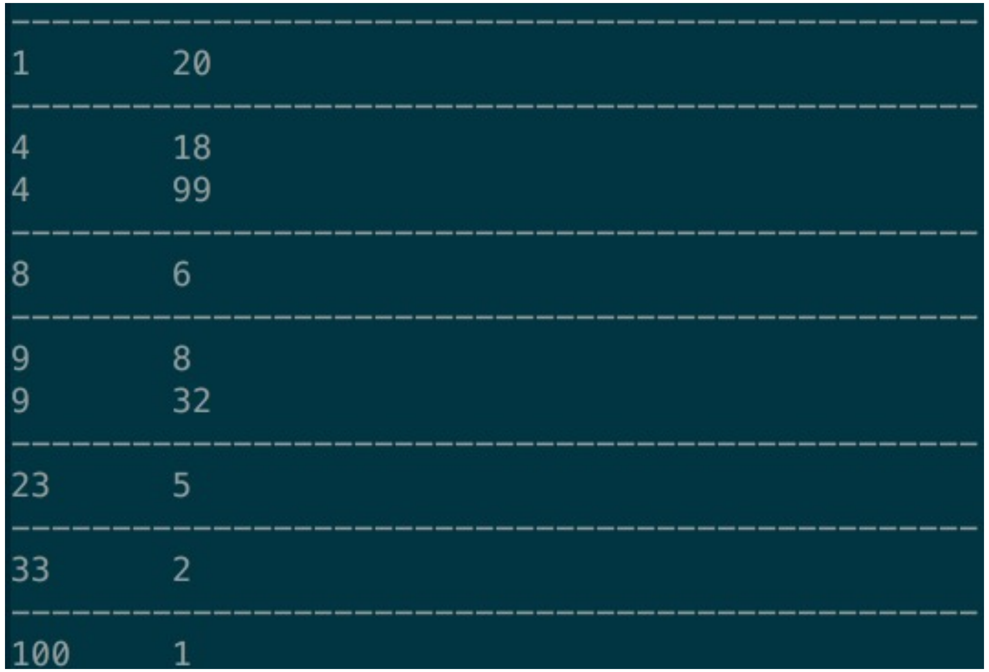
- SecondarySort
  - Hadoop在将Mapper产生的数据输送给Reducer之前，会自动对它们进行排序
  - 那么，如果我们还希望按值排序，应该怎么做呢?
    - 二级排序。通过对key对象的格式进行小小的修改，二级排序可以在排序阶段将值的作用也施加进去。

MapReduce二级排序

- SecondarySort
  - 方法1：Reducer将给定key的所有值都缓存起来，然后对它们再做一个Reducer内排序。但是，由于Reducer需要保存给定key的所有值，可能会导致出现内存耗尽的错误。
  - 方法2：将值的一部分或整个值加入原始key，生成一个合成key。
    - 生成组合key的过程很简单。我们需要先分析一下，在排序时需要把值的哪些部分考虑在内，然后，把它们加进key里去。随后，再修改key类的compareTo方法或是Comparator类，确保排序的时候使用这个组合而成的key。

MapReduce二级排序

- 例如对如下int pair进行排序：
  - 1 20
  - 9 8
  - 9 32
  - 33 2
  - 4 99
  - 4 18
  - 8 6
  - 100 1
  - 23 5
- 设计的(key value)对为((left, right),right)



```
--------------------------------------------
1        20
--------------------------------------------
4        18
4        99
--------------------------------------------
8        6
--------------------------------------------
9        8
9        32
--------------------------------------------
23       5
--------------------------------------------
33       2
--------------------------------------------
100      1
```

MapReduce二级排序

- 主要工作：
  - 1. 自定义key
    - Define a pair of integers that are writable. They are serialized in a byte comparable format.
    - class IntPair implements WritableComparable<IntPair>
  - 2. 自定义Partitioner类
    - Partition based on the first part of the pair.
    - class FirstPartitioner extends Partitioner<IntPair, IntWritable>
  - 3. 自定义Key的比较类
    - A Comparator that compares serialized IntPair.
    - class Comparator extends WritableComparator

MapReduce二级排序

- 主要工作：
  - 4. 自定义分组比较类
    - Compare only the first part of the pair, so that reduce is called once for each value of the first part.
    - class FirstGroupingComparator extends WritableComparator
  - 5. 定义Mapper类
    - Read two integers from each line and generate a key, value pair as ((left, right), right).
  - 6. 定义Reducer类
    - A reducer class that just emits the sum of the input values.
- 实现： org.apache.hadoop.examples.SecondarySort

构建单词同现矩阵算法

- Word Co-occurrence Matrix
  - 语料库的单词同现矩阵是一个二维 N×N矩阵
  - N是语料库的词汇量（即，不同单词的数目）
  - 矩阵元素M[i, j] 代表单词W[i] 与单词W [j]在一定范围内同现的次数（一个语句中，一个段落中，一篇文档中，或文本串中一个宽度为M个单词的窗口中，这些都依具体问题而定）
- Building word co-occurrence matrices from large corpora
  - a common task in text processing, and provides the starting point to many other algorithms.

构建单词同现矩阵算法

- A Word Co-occurrence Matrix Example

	agent	mining	communication	audio	cognition	...
Mitsuru Ishizuka	454	143	414	382	246	...
Koiti Hasida	412	156	1020	458	1150	...
Yutaka Matsuo	129	112	138	89	58	...
Nobuaki Minematsu	227	22	265	648	138	...
Yohei Asada	6	6	6	2	0	...
...	...	...	...	...	...	...

Figure: Example of person-to-word co-occurrence matrix

Figure taken from: Yutaka Matsuo, …, POLYPHONET: An Advanced Social Network Extraction System from the Web, 2006

构建单词同现矩阵算法

- Building the words co-occurrence matrix
  - 如果内存足够大，把整个矩阵放在内存中，矩阵元素的计算会非常简单
  - 实际上，web-scale的文档的词汇量可能有数十万，甚至数亿
  - 同现矩阵的空间开销为 $O(n^2)$
  - 简单地在单机上的实现，内存与磁盘之间的换页会使任务的执行十分缓慢

构建单词同现矩阵算法

M.R. Algorithm (“pairs” approach) pseudo-code:

```
class Mapper
  method Map(docid a, doc d)
     for all term w ∈ doc d do
          for all term u ∈ Neighbors(w) do
               //Emit count for each co-occurrence
               Emit(pair (w, u), count 1)
------------------------------------------------------------
class Reducer
  method Reduce(pair p; counts [c1, c2,…])
     s ← 0
        for all count c 2 counts [c1, c2,…] do
             s ← s + c              //Sum co-occurrence counts
        Emit(pair p, count s)
```

构建单词同现矩阵算法

- A simple “pairs” approach example
  - 语料

    we are not what we want to be but at least we are not what we used to be
  - 同现定义 Neighbors (w)
    - words that co-occur with w within a 2-word window

构建单词同现矩阵算法

- A simple “Pairs” approach example (cont.)
  - after map
    - (<we, are>, 1)
    - (<are, not>, 1)
    - (<not, what>, 1)
    - (<we, want>, 1)
    - (<want, to>, 1)
    - (<to, be>, 1)
    - (<but, at>,1)
    - (<at, least>,1)
    - (<we, are>,1)
    - (<are, not>,1)
    - (<not, what>,1)
    - (<we , used>,1)
    - (<used, to>,1)
    - (<to, be>,1)

构建单词同现矩阵算法

- A simple “Pairs” approach example (cont.)
  - after shuffle and sort
    - (<we, are>,[1,1])
    - (<are, not>,[1,1])
    - (<not, what>,[1,1])
    - (<we, want>,[1])
    - (<want, to>,[1])
    - (<to, be>,[1,1])
    - (<but, at>, [1])
    - (<at, least>, [1])
    - (<we, used>, [1])
    - (<used, to>, [1])

构建单词同现矩阵算法

- A simple “Pairs” approach example (cont.)
  - after reduce
    - (<we, are>,2)
    - (<are, not>,2)
    - (<not, what>,2)
    - (<we, want>,1)
    - (<want, to>,1)
    - (<to, be>,2)
    - (<but, at>,1)
    - (<at, least>,1)
    - (<we, used>,1)
    - (<used, to>,1)

构建单词同现矩阵算法

- A simple “Pairs” approach example (cont.)

	we	are	not	what	want	to	be	but	at	least	used
we		2			1						1
are	2		2								
not		2		2							
what			2								
want	1					1					
to					1		1				1
be						1					
but									1		
at								1			
least											
used	1					1					

Figure: the co-occurrence matrix

构建单词同现矩阵算法

- 算法的扩展
  - 同现定义 Neighbors(w)为其他形式时该怎么实现
    - 根据同现关系的不同，可能需要实现和定制不同的FileInputFormat和RecordReader，
    - 如同现关系为一个英文句子，则需要实现以一个英文句子为单位的FileInputFormat和RecordReader
    - 如同现关系为一个段落，则需要实现以一个段落为单位的FileInputFormat和RecordReader
  - 同现关系可扩展为从大量观察数据中进行任意离散关联事件的分析和数据挖掘
  - 类似应用问题
    - 零售商通过分析大量的交易记录，识别出关联的商品购买行为（如：“啤酒和纸尿裤”的故事）
    - 从生物医学文献中自动挖掘基因交互作用关系

文档倒排索引算法

- 文档倒排算法简介
  - Inverted Index(倒排索引)是目前几乎所有支持全文检索的搜索引擎都要依赖的一个数据结构。基于索引结构，给出一个词(**term**)，能取得含有这个term的文档列表(the list of **documents**)
  - Web Search中的问题主要分为三部分：
    - crawling(gathering web content)
    - indexing(construction of the inverted index)
    - retrieval(ranking documents given a query)
  - crawling和indexing都是离线的，retrieval是在线、实时的

文档倒排索引算法

- 简单的文档倒排算法

doc1：
one fish
two fish

doc2：
red fish
blue fish

doc3：
one red bird

倒排索引：
one: doc1, doc3
fish: doc1, doc2
two: doc1
red: doc2, doc3
blue: doc2
bird: doc3

基于以上索引的搜索结果：
fish → doc1, doc2
red → doc2, doc3
red fish → doc2

文档倒排索引算法

简单的文档倒排算法

改进：map输出的key除了文件名,还给出了该词所在行的偏移值：
格式：filename#offset

```
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
public class InvertedIndexMapper extends Mapper<Text, Text, Text, Text>
{           @Override
            protected void map(Text key, Text value, Context context)
                                    throws IOException, InterruptedException
            // default RecordReader: LineRecordReader; key: line offset; value: line string
            {           FileSplit fileSplit = (FileSplit)context.getInputSplit();
                        String fileName = fileSplit.getPath().getName();
                        Text word = new Text();
                        Text fileName_lineOffset = new Text(fileName+"#"+key.toString());
                        StringTokenizer itr = new StringTokenizer(value.toString());
                        for(; itr.hasMoreTokens(); )
                        {     word.set(itr.nextToken());
                              context.write(word, fileName_lineOffset);
                        }
            }
}
```

文档倒排索引算法

简单的文档倒排算法

```
import java.io.IOException;
import java.util.Collections;
import java.util.Iterator;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class InvertedIndexReducer extends Reducer<Text, Text, Text, Text>
{           @Override
            protected void reduce(Text key, Iterable<Text> values, Context context)
                              throws IOException, InterruptedException
            {           Iterator<Text> it = values.iterator();
                        StringBuilder all = new StringBuilder();
                        if(it.hasNext())  all.append(it.next().toString());
                        for(; it.hasNext(); )
                        {           all.append(“;");
                                    all.append(it.next().toString());
                        }
                        context.write(key, new Text(all.toString()));
            } //最终输出键值对示例：(“fish", “doc1#0; doc1#8;doc2#0;doc2#8 ")
}
```

文档倒排索引算法

- 简单的文档倒排算法

```
public class InvertedIndexer
{
    public static void main(String[] args)
    {
        try {       Configuration conf = new Configuration();
                    job = new Job(conf, "invert index");
                    job.setJarByClass(InvertedIndexer.class);
                    job.setInputFormatClass(TextInputFormat.class);
                    job.setMapperClass(InvertedIndexMapper.class);
                    job.setReducerClass(InvertedIndexReducer.class);
                    job.setOutputKeyClass(Text.class);
                    job.setOutputValueClass(Text.class);
                    FileInputFormat.addInputPath(job, new Path(args[0]));
                    FileOutputFormat.setOutputPath(job, new Path(args[1]));
                    System.exit(job.waitForCompletion(true) ? 0 : 1);
           } catch (Exception e) {    e.printStackTrace();                 }
    }
}
```

文档倒排索引算法

- 输出

```
bird    doc3.txt#0
blue    doc2.txt#9
fish    doc2.txt#9;doc2.txt#0;doc1.txt#10;doc1.txt#0
one     doc1.txt#0;doc3.txt#0
red     doc3.txt#0;doc2.txt#0
two     doc1.txt#10
```

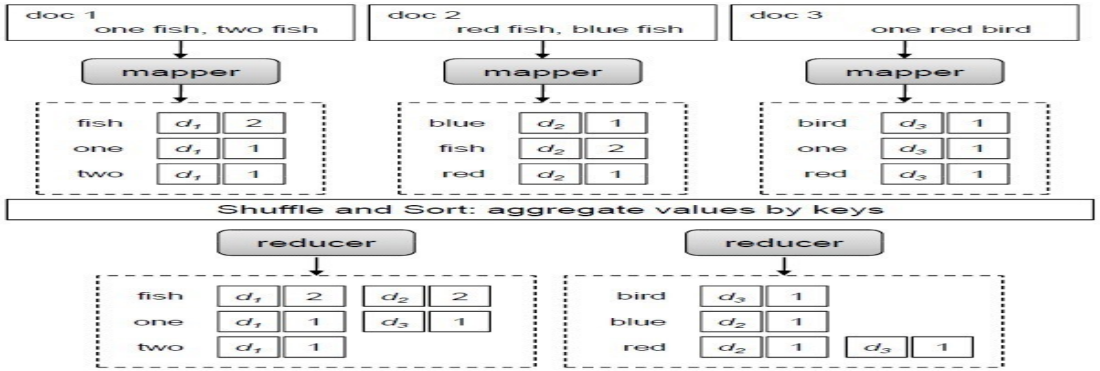
文档倒排索引算法

- 带词频等属性的文档倒排算法
  - 如果考虑单词在每个文档中出现的词频、位置、对应Web文档的URL等诸多属性，则前述简单的倒排算法就不足以有效工作。我们把这些词频、位置等诸多属性称为有效负载（Payload）

注：以下的算法内容引自Jimmy Lin，Data-Intensive Text Processing with MapReduce， 2010，College Park, 以及其课件

文档倒排索引算法

- 带词频等属性的文档倒排算法
  - 基本的倒排索引结构



  - 一个倒排索引由大量的postings list构成
  - 一个postings list由多个posting构成(按doc id排序)
  - 一个postings list与一个term关联
  - 一个posting 包含一个document id和一个payload
  - payload上载有term在document中出现情况相关的信息(e.g. term frequency, positions, term properties)
  - 同时还有对应Web文档到其URL的映射doc_id→URL

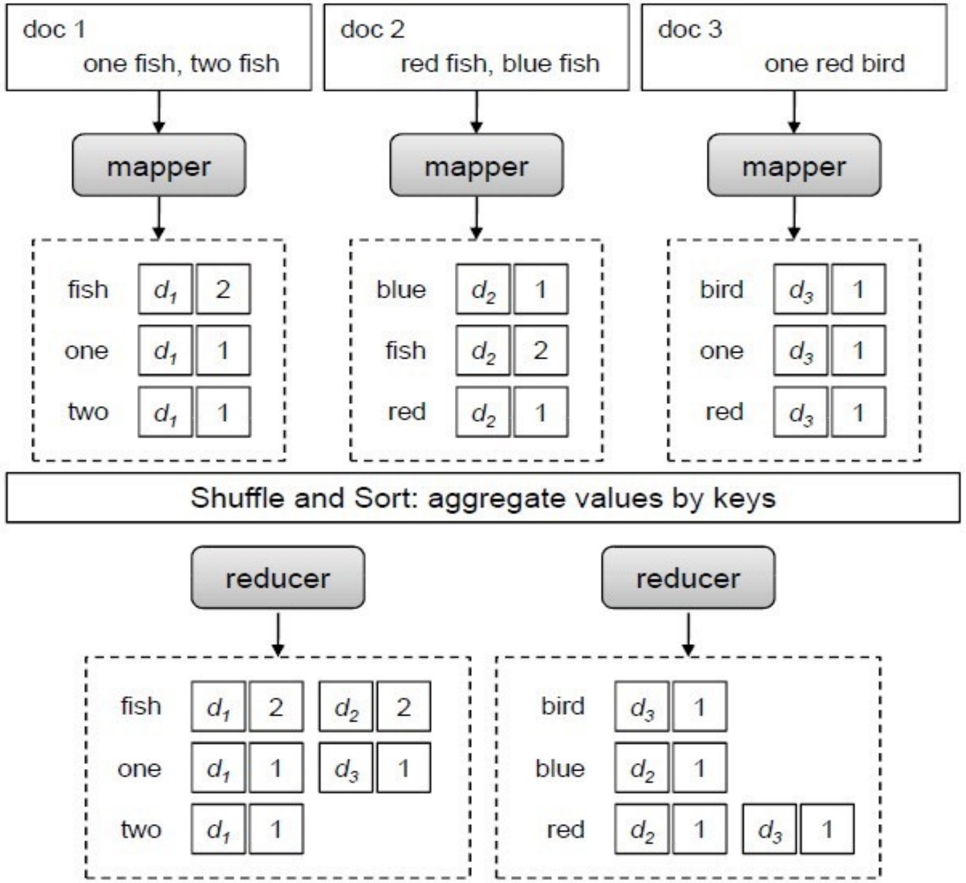
文档倒排索引算法

- 带词频等属性的文档倒排算法
  - Map和Reduce实现伪代码

```
class Mapper
    procedure Map(docid n, doc d)
        H ← new AssociativeArray
        for all term t ∈ doc d do
            H{t} ← H{t} + 1
        for all term t ∈ H do
            Emit(term t, posting <n, H{t}>)
------------------------------------------------------------
class Reducer
    procedure Reduce(term t, postings [<n1, f1>, <n2, f2>…])
        P ← new List
        for all posting <a, f> ∈ postings [<n1, f1>, <n2, f2>…] do
            Append(P, <a, f>)
        Sort(P)
        Emit(term t; postings P)
```

文档倒排索引算法

- 带词频等属性的文档倒排算法



A simple example posting(docid, tf)

文档倒排索引算法

- 带词频等属性的文档倒排算法
  - Scalability bottleneck
    - The algorithm assumes that there is sufficient memory to hold all postings associated with the same term.
    - The reducer first buffers all postings and then performs an in-memory sort.
    - As collections grow larger, reducers will run out of memory.
  - Solution
    - let the MapReduce runtime do the sorting
    - Emit the intermediate key-value pairs like this:

      (tuple <term, docid>, tf f)

      (design trick: value-to-key conversion)

文档倒排索引算法

- 带词频属性的文档倒排算法

Mapper

doc1：
one fish
two fish

(fish, doc1) 2
(one, doc1) 1
(two, doc1) 1

doc2:
red fish
blue fish

(fish, doc2) 2
(blue, doc2) 1
(red, doc2) 1

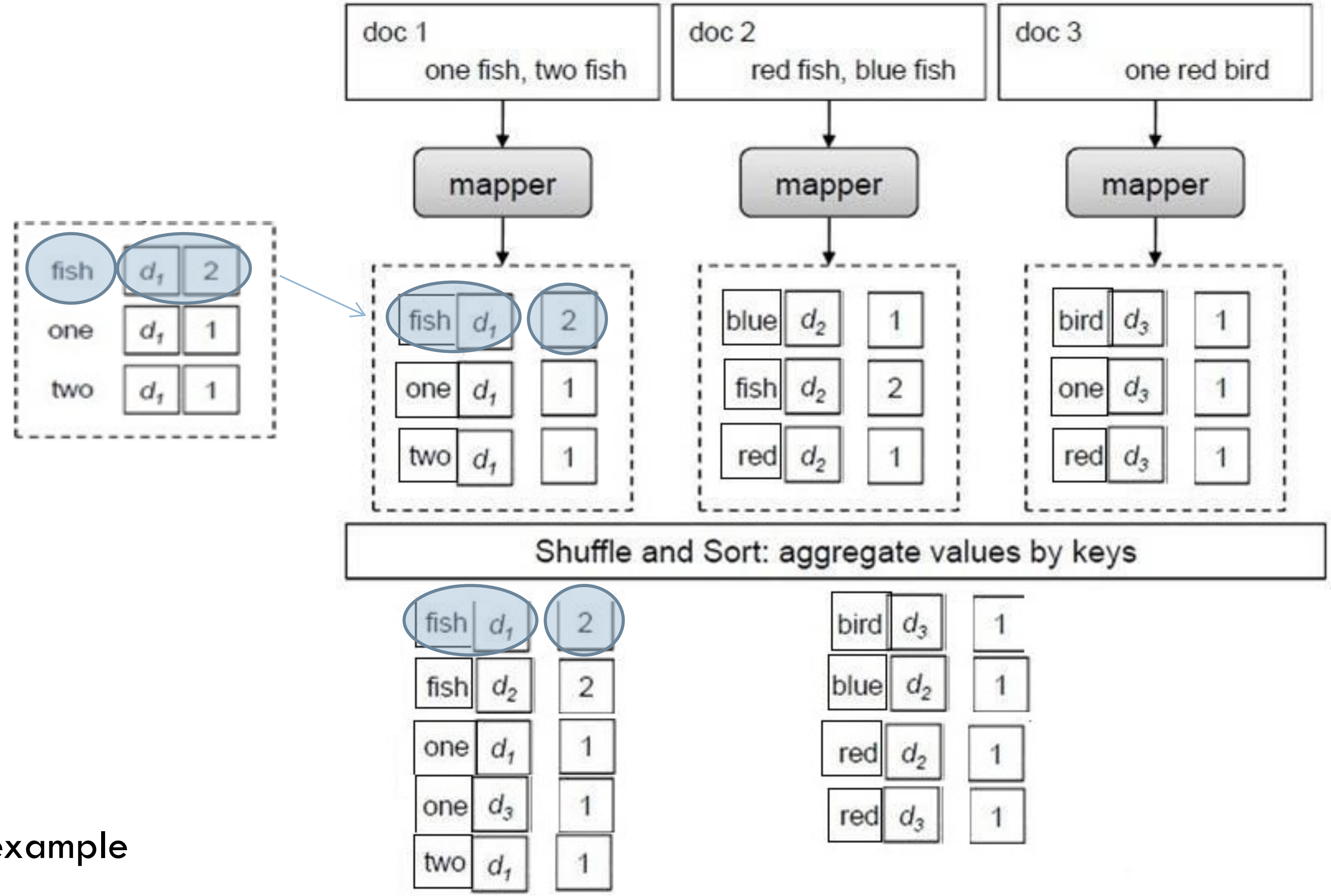
doc3:
one red bird

(bird, doc3) 1
(one, doc3) 1
(red, doc3) 1

<(term, docid), P>

Reducer

one: (doc1,1)(doc3,1)
fish: (doc1,2)(doc2,2)
two: (doc1,1)
red: (doc2,1)(doc3,1)
blue: (doc2,1)
bird: (doc3,1)



A revised example

文档倒排索引算法

- 可扩展的带词频属性的文档倒排算法
  - **Mapper**

```
class Mapper
    method Map(docid n; doc d)
        H ← new AssociativeArray
        for all term t ∈ doc d do
            H{t} ← H{t} + 1
        for all term t ∈ H do
            Emit(tuple<t, n>, tf H{t})
```

文档倒排索引算法

- 可扩展的带词频属性的文档倒排算法
  - A customized partitioner
    - **Why?**
      - To ensure that all tuples with the same term are shuffled to the same reducer(notice that the new key is a <term, docid> tuple)

文档倒排索引算法

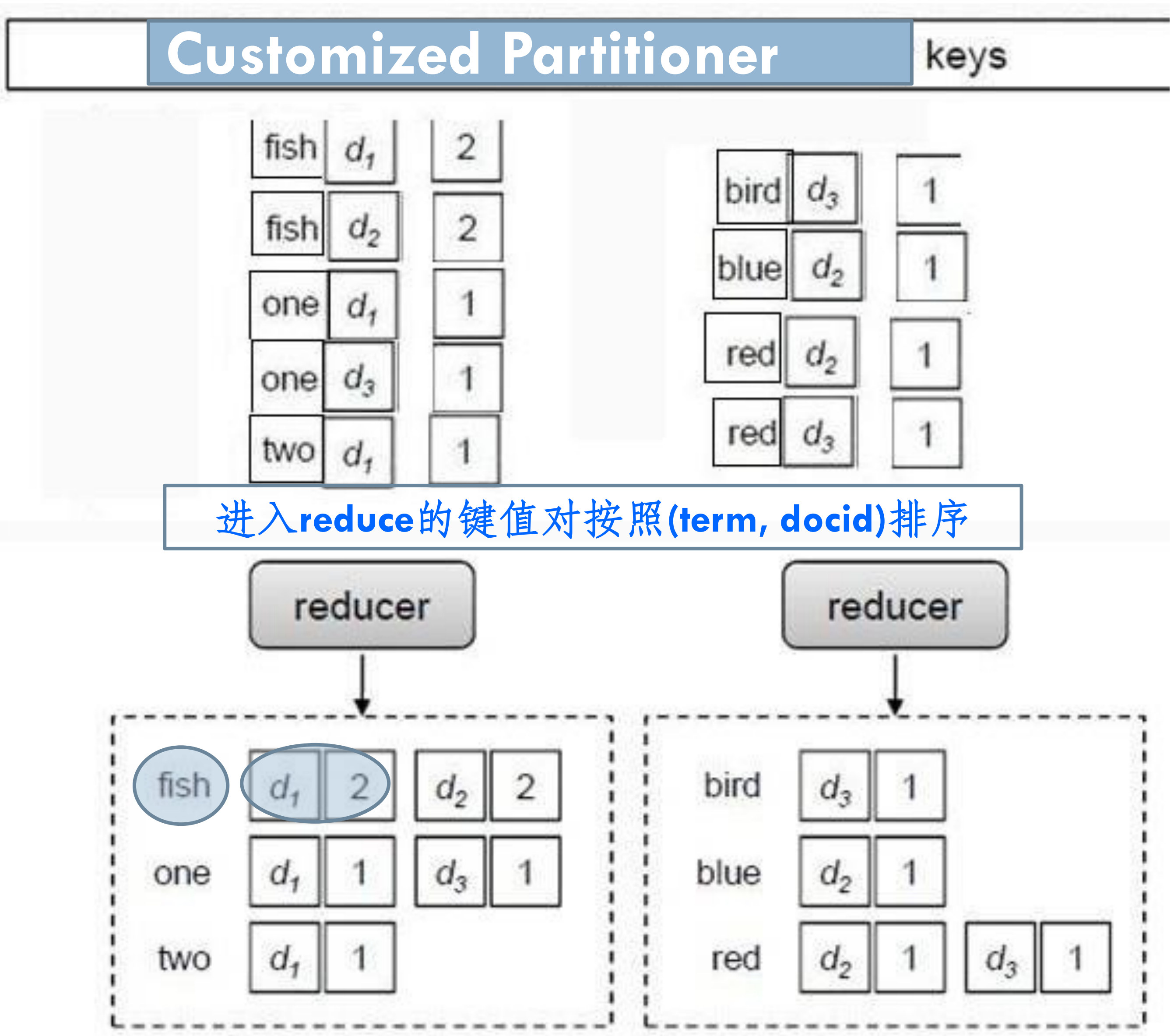
- 可扩展的带词频属性的文档倒排算法
  - **How?**
    - Class **NewPartitioner** extends HashPartitioner<K,V>

```
// org.apache.hadoop.mapreduce.lib.partition.HashPartitioner
{   // override the method
    getPartition(K key, V value, int numReduceTasks)
    {   term = key. toString().split(“,”)[0]; //<term, docid>=>term
        super.getPartition(term, value, numReduceTasks);
    }
}
```

    - Set the customized partitioner in job configuration

Job.setPartitionerClass(NewPartitioner)

Customized Partitioner



A revised example(cont.)

文档倒排索引算法

- 可扩展的带词频属性的文档倒排算法
  - **Reducer**

class Reducer
    method Setup // 初始化
        $t_{prev} \leftarrow \varnothing$;
        $P \leftarrow$ new PostingsList
    method Reduce(tuple $<t, n>$, tf [f])
        if $t \neq t_{prev} \wedge t_{prev} \neq \varnothing$ **then**
            Emit(t_{prev}, P)
            P.Reset()
        P.Add($<n, f>$)
        $t_{prev} \leftarrow t$
    method Cleanup
        Emit(t, P)

用于输出最后一次未得到输出的$<t, P>$

文档倒排索引算法

- 可扩展的带词频属性的文档倒排算法
  - Extensions
    - 单词形态还原(e.g. ‘books’ -> ‘book’, …)
    - removing stop-words (common words such as ‘the’, ‘a’, ‘of’, etc.)

专利文献数据分析

数据源：美国专利文献数据

- Available from the National Bureau of Economic Research at http://www.nber.org/patents/
- The data sets were originally compiled for the paper “The NBER Patent Citation Data File: Lessons, Insights and Methodological Tools.”
- Two data sets：
  - Citation data set “cite75_99.txt”
  - Patent description data set “apat63_99.txt”

专利文献数据分析

数据源：美国专利文献数据

Citation data set “cite75_99.txt”

“CITING”,”CITED”

3858241,956203

3858241,1324234

3858241,3398406

3858241,3557384

3858241,3634889

3858242,1515701

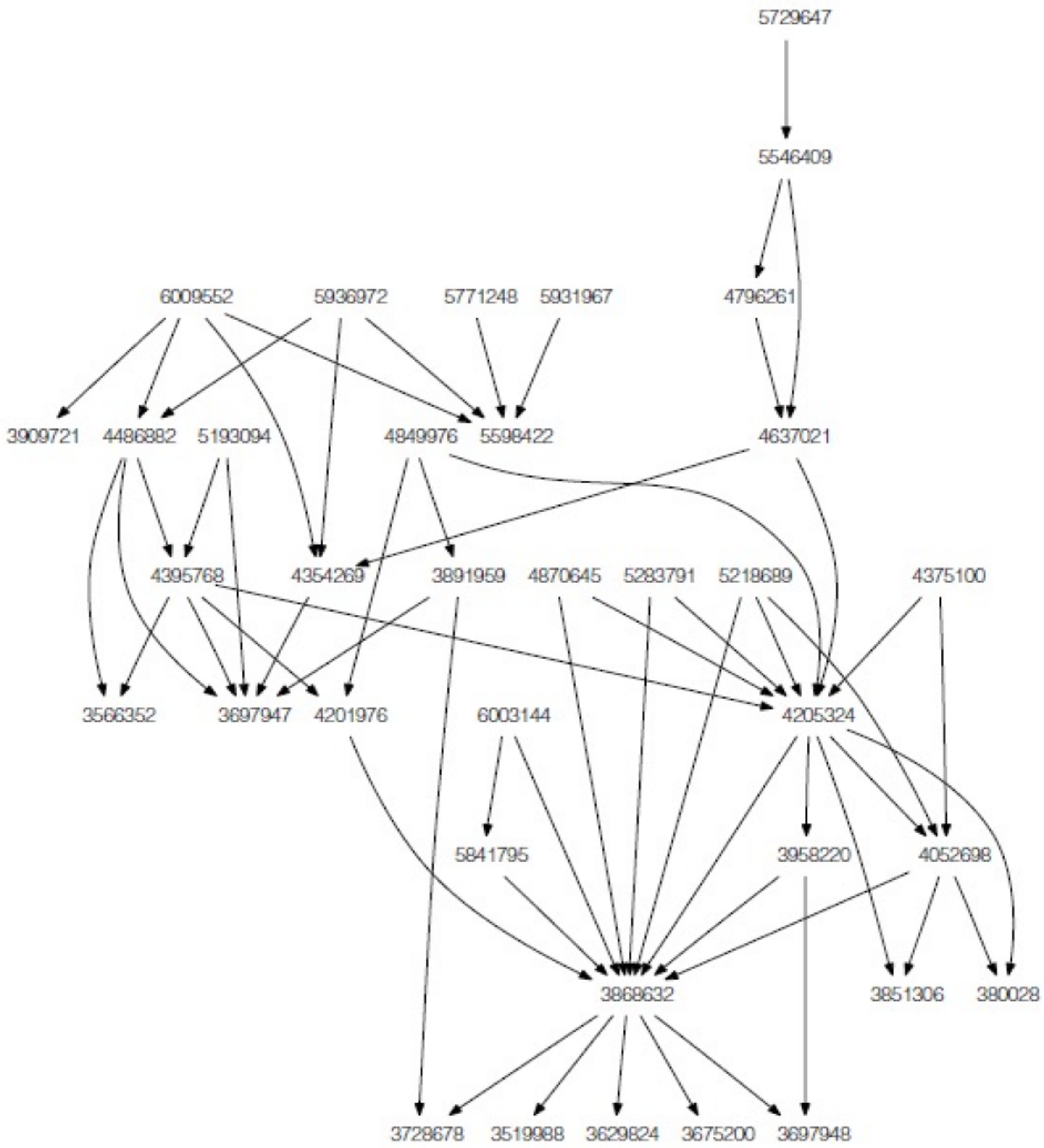
3858242,3319261

3858242,3668705

3858242,3707004

…

3858241引用了
956203, 1324234, 3398406,
3557384, 3634889



A partial view of the patent citation data set as a graph. Each patent is shown as a vertex (node), and each citation is a directed edge (arrow).

专利文献数据分析

数据源：美国专利文献数据

Patent description data set “apat63_99.txt”

“PATENT”,”GYEAR”,”GDATE”,”APPYEAR”,”COUNTRY”, ”POSTATE”,”ASSIGNEE”, ”ASSCODE”,”CLAIMS”,”NCLASS”,”CAT”,”SUBCAT”,”CMADE”,”CRECEIVE”, ”RATIOCIT”,”GENERAL”,”ORIGINAL”,”FWDAPLAG”,”BCKGTLAG”,”SELFCTUB”, ”SELFCTLB”,”SECDUPBD”,”SECDLWBD”

3070801,1963,1096,,”BE”,””,,1,,269,6,69,,1,,0,,,,,,

3070802,1963,1096,,”US”,”TX”,,1,,2,6,63,,0,,,,,,,,

3070803,1963,1096,,”US”,”IL”,,1,,2,6,63,,9,,0.3704,,,,,,

3070804,1963,1096,,”US”,”OH”,,1,,2,6,63,,3,,0.6667,,,,,,

3070805,1963,1096,,”US”,”CA”,,1,,2,6,63,,1,,0,,,,,,

专利文献数据分析

数据源：美国专利文献数据

Patent description data set “apat63_99.txt”

Attribute name	Content
PATENT	Patent number
GYEAR	Grant year
GDATE	Grant date, given as the number of days elapsed since January 1, 1960
APPYEAR	Application year (available only for patents granted since 1967)
COUNTRY	Country of first inventor
POSTATE	State of first inventory (if country is U.S.)
ASSIGNEE	Numeric identifier for assignee (i.e., patent owner)
ASSCODE	One-digit (1-9) assignee type. (The assignee type includes U.S. individual U.S. government, U.S. organization, non-U.S. individual, etc.)
CLAIMS	Number of claims (available only for patents granted since 1975)
NCLASS	3-digit main patent class

- 专利被引列表(citation data set 倒排)
- Map

```
public static class MapClass extends Mapper<LongWritable, Text, Text, Text>
{
    public void map(LongWritable key, Text value, Context context)
                        throws IOException, InterruptedException
    // 输入key: 行偏移值；value: "citing专利号, cited专利号" 数据对
    {       String[] citation = value.toString().split(",");
            context.write(new Text(citation[1]), new Text(citation[0]));
    } // 输出key: cited 专利号；value: citing专利号
}
```

- 专利被引列表(citation data set 倒排)
  - Reduce

```
public static class ReduceClass extends Reducer<Text, Text, Text, Text>
{
    public void reduce(Text key, Iterable<Text> values, Context context)
                        throws IOException, InterruptedException
    {       String csv = "";
            for (Text val:values)
            {   if (csv.length() > 0) csv += ",";
                 csv += val.toString();
            }
            context.write(key, new Text(csv));
    } // 输出key: cited专利号; value: "citing专利号1, cited专利号2,..."
}
```

专利文献数据分析

- 专利被引列表(citation data set 倒排)
  - 专利被引列表输出结果

1	3964859,4647229
10000	4539112
100000	5031388
1000006	4714284
1000007	4766693
1000011	5033339
1000017	3908629
1000026	4043055
1000033	4190903,4975983
1000043	4091523
1000044	4082383,4055371
1000045	4290571
1000046	5918892,5525001, 5609991

……

专利文献数据分析

- 专利被引次数统计
- Map Class

```
public static class MapClass extends Mapper<LongWritable, Text, Text, Text>
{
    private IntWritable one = new IntWritable(1);
    public void map(LongWritable key, Text value, Context context)
                    throws IOException, InterruptedException
    // 输入key: 行偏移值； value: "citing专利号, cited专利号" 数据对
    {       String[] citation = value.toString().split(",");
            context.write(new Text(citation[1]), one);
    } // 输出key: cited 专利号； value: citing专利号
}
```

- 专利被引次数统计
- Reduce Class

```
public static class ReduceClass extends Reducer<Text, Text, Text, Text> {
    public void reduce(Text key, Iterable<Text> values, Context context)
                          throws IOException, InterruptedException {
            int count = 0;
        while (values.hasNext()) {
                  count += values.next().get();
        }
        context.write(key, new IntWritable(count));
    } // 输出key: 被引专利号; value: 被引次数
}
```

专利被引次数统计

- 专利被引次数统计输出结果

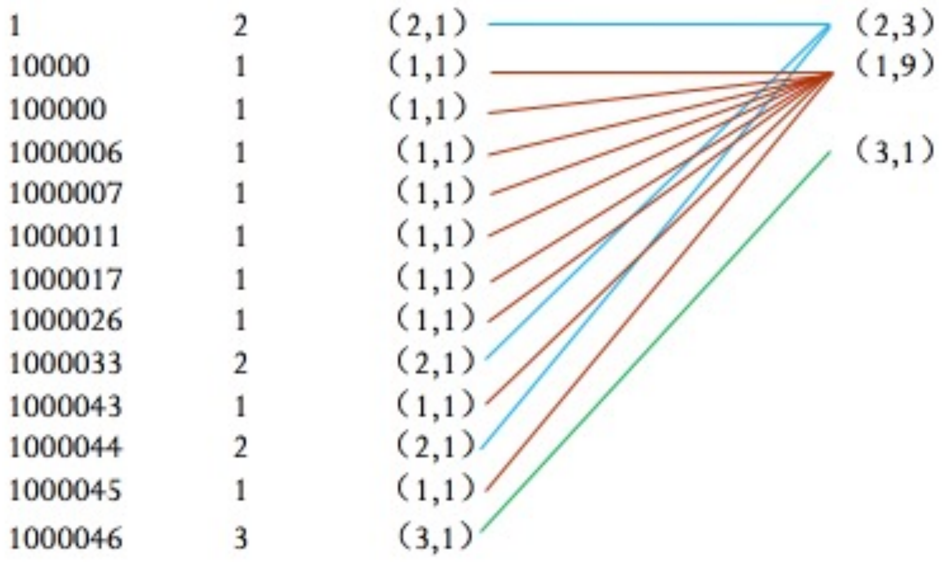
1	2
10000	1
100000	1
1000006	1
1000007	1
1000011	1
1000017	1
1000026	1
1000033	2
1000043	1
1000044	2
1000045	1
1000046	3

……

专利文献数据分析

专利被引次数统计

- 目的：有的专利被引用一次，有的可能上百次，可以进行引用次数分布统计，最后可画出统计图。
- 基本思想是：扫描刚才产生的被引次数统计数据，忽略每一行中的专利号，仅考虑右侧的被引次数，看每种被引次数分别有多少次出现



- 专利被引次数统计
- Map Class

```
public static class MapClass extends Mapper<Text, Text, LongWritable, LongWritable>
{
    private final static IntWritable one = new IntWritable(1);
    private IntWritable citationCount = new IntWritable();
    public void map(Text key, Text value, Context context)
                        throws IOException, InterruptedException {
            citationCount.set(Integer.parseInt(value.toString()));
            context.write (citationCount, one);
    }
}
```

被引次数

出现1次

- 专利被引次数统计
- Reduce Class

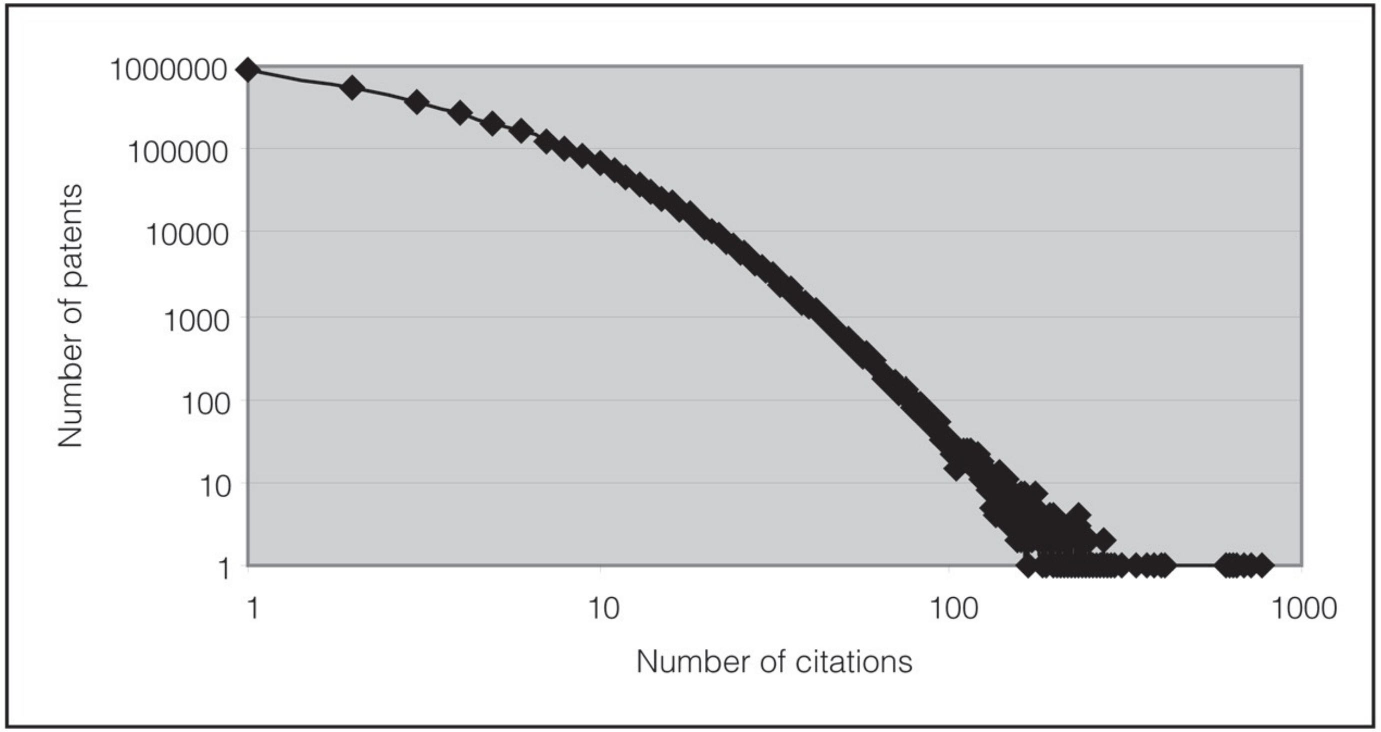
```
public static class ReduceClass extends Reducer
                    < IntWritable,IntWritable,IntWritable,IntWritable >
{
    public void reduce(IntWritable key, Iterable<IntWritable> values,    Context context) throws
    IOException, InterruptedException
    {       int count = 0;
            while (values.hasNext()) {    count += values.next().get(); }
            context.write(key, new IntWritable(count));
     } // 输出key：被引次数； value：总出现次数
}
```

- 主类-- CitationHistogram

```
public class CitationHistogram{
    public static void main(String[] args) {
        Configuration conf = new Configuration();
        JobConf job = new JobConf(conf, CitationHistogram.class);
        Path in = new Path(args[0]);
        Path out = new Path(args[1]);
        FileInputFormat.setInputPaths(job, in);
        FileOutputFormat.setOutputPath(job, out);
        job.setJobName("CitationHistogram");
        job.setMapperClass(MapClass.class);
        job.setReducerClass(ReduceClass.class);
        job.setInputFormat(KeyValueTextInputFormat.class);
        job.setOutputFormat(TextOutputFormat.class);
        job.setOutputKeyClass(IntWritable.class);
        job.setOutputValueClass(IntWritable.class);
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

专利文献数据分析

- 专利被引次数统计结果



1	921128
2	552246
3	380319
4	278438
5	210814
6	163149
7	127941
8	102155
9	82126
10	66634
…	
411	1
605	1
613	1
631	1
633	1
654	1
658	1
678	1
716	1
779	1

专利文献数据分析

- 年份/国家专利数统计

Patent description data set "apat63_99.txt"

"PATENT","**GYEAR**","GDATE","APPYEAR","COUNTRY", "POSTATE","ASSIGNEE", "ASSCODE","CLAIMS","NCLASS","CAT","SUBCAT", "CMADE","CRECEIVE", "RATIOCIT","GENERAL","ORIGINAL","FWDAPLAG","BCKGTLAG","SELFCTUB", "SELFCTLB","SECDUPBD", "SECDLWBD"

3070801,1963,1096,,"BE","",,1,,269,6,69,,1,,0,,,,,,

3070802,1963,1096,,"US","TX",,1,,2,6,63,,0,,,,,,,,

3070803,1963,1096,,"US","IL",,1,,2,6,63,,9,,0.3704,,,,,,

3070804,1963,1096,,"US","OH",,1,,2,6,63,,3,,0.6667,,,,,,

3070805,1963,1096,,"US","CA",,1,,2,6,63,,1,,0,,,,,,

……

主要设计思想是：分析以上的专利描述数据集，根据要统计的列名(年份或国家等)，取出对应列上的年份(col_idx=1)或国家(col_idx=4)，然后由Map发出(year，1)或(country，1)，再由Reduce累加。

- 年份/国家专利数统计
- Map Class

```
public static class MapClass extends Mapper<Text, Text, Text, LongWritable> {
    private final static IntWritable one = new IntWritable(1);
    private int col_idx = 1;  // 1：年份；  4：国家
    public void map(Text key,  Text value, Context context)
                  throws IOException, InterruptedException {
        String[] cols = value.Split(',');  // value：读入的一行专利描述数据记录
        String col_data = cols[col_idx];
        context.write (new Text(col_data), one);
    }
}
```

年份或国家

出现1次

专利文献数据分析

- 年份/国家专利数统计
- Reduce Class

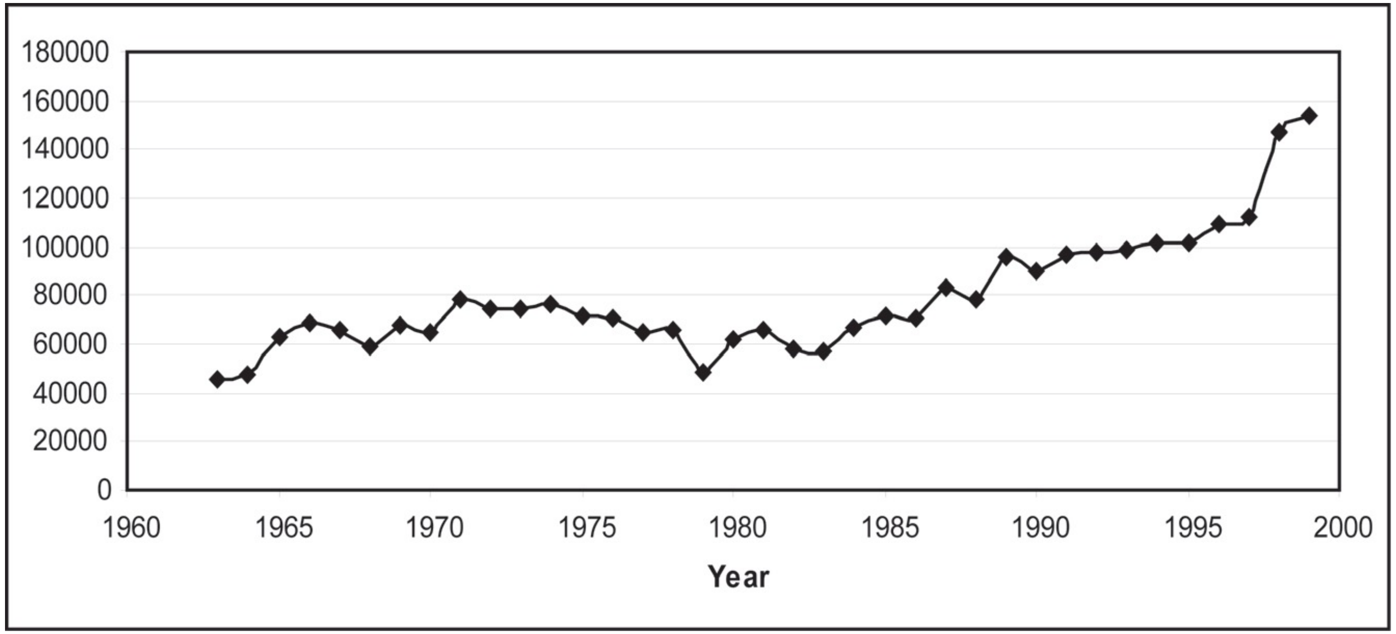
```
public static class ReduceClass extends Reducer
                        < Text, IntWritable, Text, IntWritable > {
    public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException,
    InterruptedException
    {       int count = 0;
            while (values.hasNext()) {    count += values.next().get(); }
            context.write(key, new IntWritable(count));
     } // 输出key: 年份或国家； value: 总的专利数
}
```

专利文献数据分析

- 年份专利统计输出

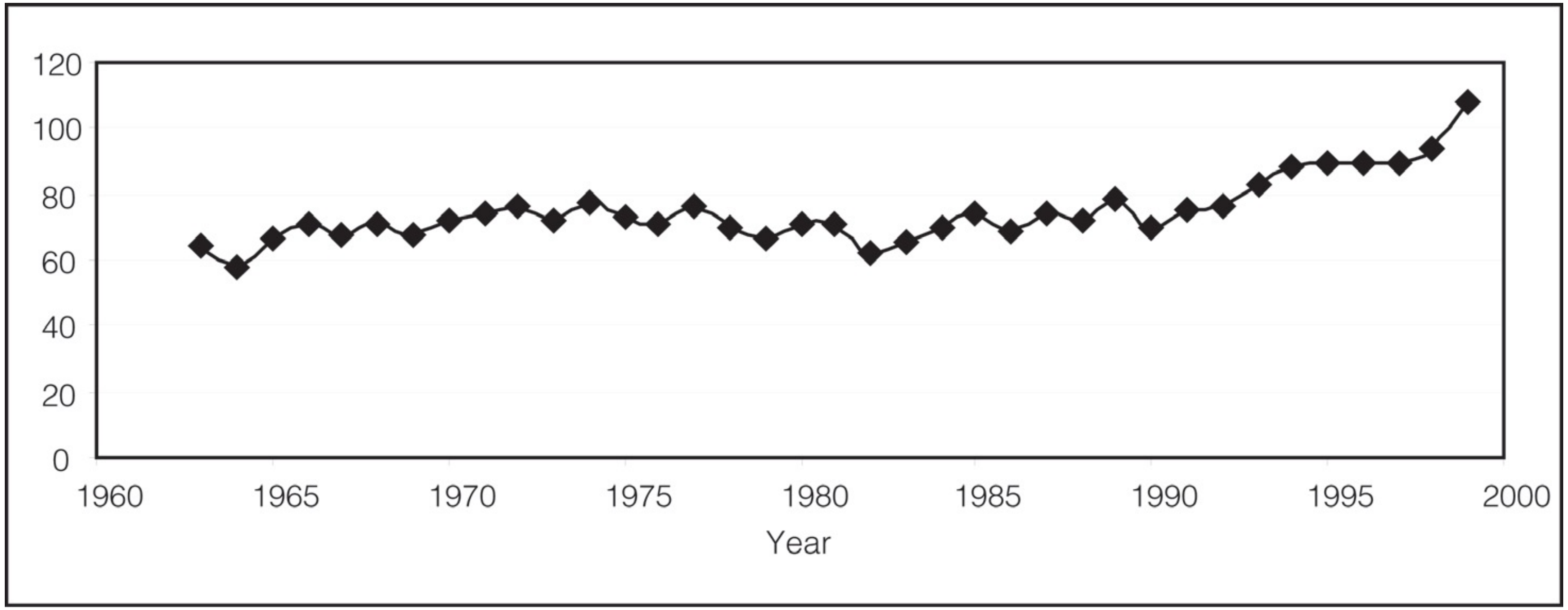
"GYEAR" 1

1963	45679
1964	47375
1965	62857
…	
1996	109645
1997	111983
1998	147519
1999	153486



专利文献数据分析

- 每年申请美国专利的国家数统计
  - 假如我们需要从专利描述数据集中统计每年有多少个国家申请了美国专利，并得出如下的统计图，该如何实现Map和Reduce？



专利文献数据分析

- 每年申请美国专利的国家数统计
- Solution 1

1. Map中用<year, country>作为key输出， Emit(<year, country>,1)

   (<1963, BE>, 1), (<1963, US>, 1), (<1963, US>, 1), …

2. 实现一个定制的Partitioner，保证同一年份的数据划分到同一个Reduce节点
3. Reduce中对每一个(<year, country>, [1, 1,1,…])输入，忽略后部的出现次数，仅考虑key部分：<year, country>

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

解决办法：在Reduce中仅计数同一年份下不同的国家个数

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

- 每年申请美国专利的国家数统计
- Solution 2

1. Map中用一个数据结构保存<year, country>，并检查一个新的<year, country>是否已经出现，若未出现则emit(year, country)并将<year, country>加入数据结构；否则跳过。如此在每个Map节点上可保证一个年份下一个国家仅出现一次

2. Reduce中对每一个(year, [country, country, country…])输入直接计数其中的country个数有问题吗?

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

解决办法：在Reduce中仅计数同一年份下不同的国家个数

专利文献数据分析

- 更多的专利文献数据分析问题
  - 如何统计一个国家占全球的专利申请比例?
  - 如何统计一个国家的专利引用率?
  - ……

MapReduce算法设计小结

- A few design tricks (“Design Patterns”)
  - Local aggregation
    - use combiner
  - Complex structures
    - such as “pairs” and “stripes”
  - value-to-key conversion

THANK YOU