深入理解大数据-大数据处理与编程实践

Ch.5. MapReduce 算法设计

南京大学计算机科学与技术系

主讲人：黄宜华，顾荣

鸣谢：本课程得到Google（北京）与Intel公司
中国大学合作部精品课程计划资助

Ch.5. MapReduce算法设计

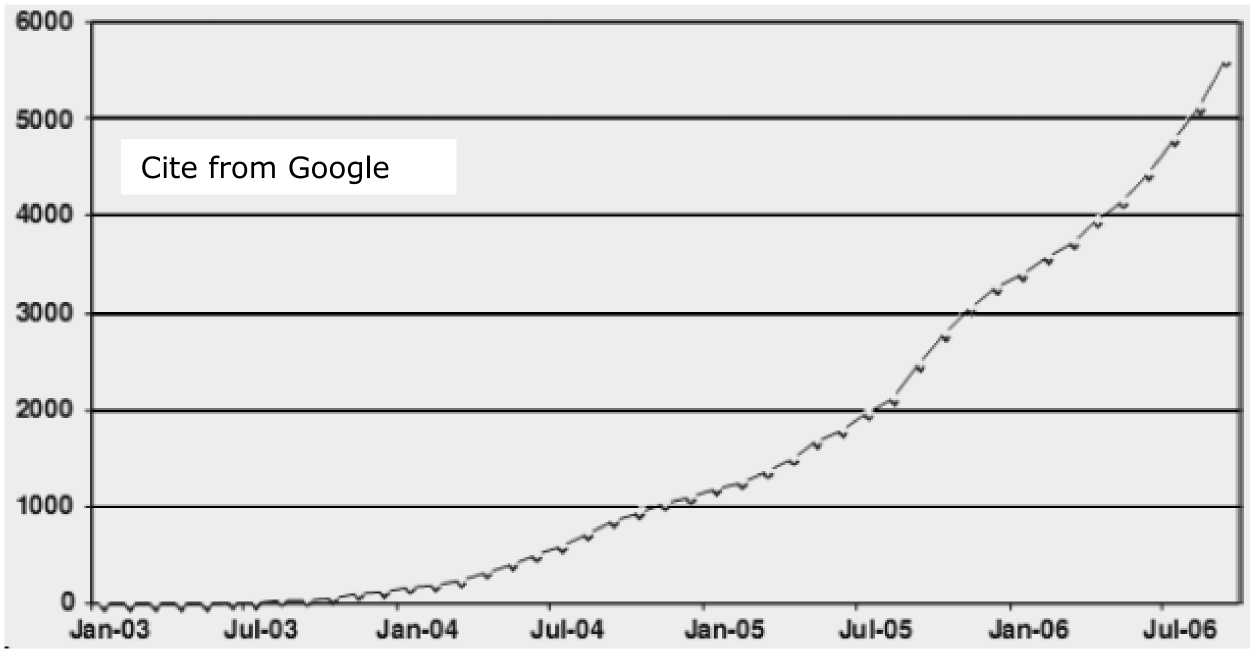
1. MapReduce可解决哪些算法问题？
2. 回顾：MapReduce处理流程
3. MapReduce排序算法
4. MapReduce单词同现分析算法
5. MapReduce文档倒排索引算法
6. 专利文献数据分析
7. 实验3：文档倒排索引实验

1. MapReduce可解决哪些算法问题？

自MapReduce发明后,Google大量用于各种海量数据处理,目前Google内部有7千以上的程序基于MapReduce实现。MapReduce可广泛应用于搜索引擎（文档倒排索引，网页链接图分析与页面排序等）、Web日志分析、文档分析处理、机器学习、机器翻译等

各种大规模数据并行计算应用领域

各类大规模数据并行处理算法。



1. MapReduce可解决哪些算法问题？

基本算法

各种全局数据相关性小、能适当划分数据的计算任务，如：

- 分布式排序
- 分布式GREP(文本匹配查找)
- 关系代数操作

  如：选择，投影，求交集、并集，连接，成组，聚合…
- 矩阵向量相乘、矩阵相乘
- 词频统计(word count)，词频重要性分析(TF-IDF)
- 单词同现关系分析

  典型的应用如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
- ……

复杂算法或应用

- Web搜索

  网页爬取、倒排索引、网页排序、搜索算法
- Web访问日志分析

  分析和挖掘用户在Web上的访问、购物行为特征、以定制个性化用户界面或投放用户感兴趣的产品广告
- 数据/文本统计分析

  如科技文献引用关系分析和统计、专利文献引用分析和统计
- 图算法

  并行化宽度优先搜索(最短路径问题，可克服Dijkstra串行算法的不足)，最小生成树，子树搜索、比对

  Web链接图分析算法PageRank，垃圾邮件连接分析
- 聚类(clustring)

  文档聚类、图聚类、其它数据集聚类

MapReduce可解决哪些算法问题？

复杂算法或应用

- 相似性比较分析算法

  字符序列、文档、图、数据集相似性比较分析
- 基于统计的文本处理

  最大期望(EM)统计模型，隐马可夫模型(HMM)，……
- 机器学习

  监督学习、无监督学习、分类算法（决策树、SVM...)
- 数据挖掘
- 统计机器翻译
- 生物信息处理

  DNA序列分析比对算法Blast：双序列比对、多序列比对

  生物网络功能模块(Motif)查找和比对
- 广告推送与推荐系统
- ……

MapReduce可解决哪些算法问题？

基于大数据集的机器学习和自然语言处理算法

信息检索、自然语言理解和机器学习的三个要素：数据, 特征与算法

2001，微软研究院的Banko and Brill*等发表了一篇自然语言理解领域的经典研究论文，探讨训练数据集大小对分类精度的影响，发现数据越大，精度越高；更有趣的发现是，他们发现当数据不断增长时，不同算法的分类精度趋向于相同，使得小数据集时不同算法在精度上的差别基本消失！

结论引起争论：看似算法不再要紧，数据更重要！看似不再需要研究复杂算法，找更多数据就行了

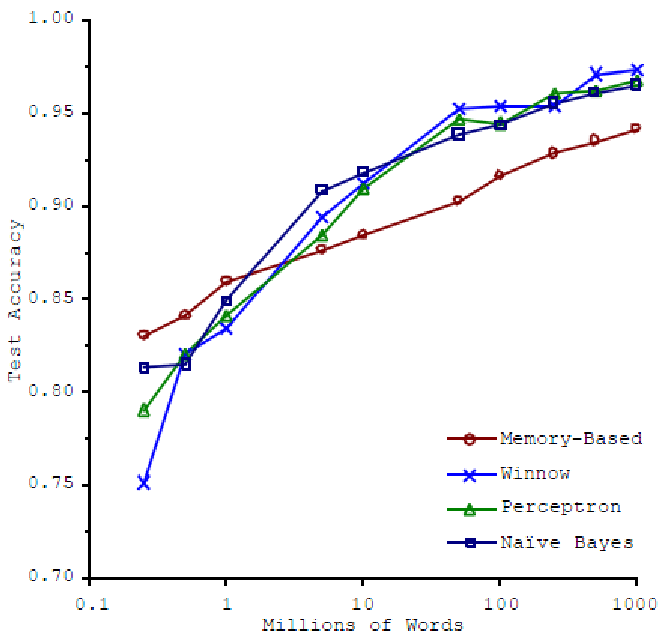


Figure 1. Learning Curves for Confusion Set Disambiguation

* M. Banko and E. Brili (2001). Scaling to very very large corpora for natural language disambiguation. *ACL 2001*

MapReduce可解决哪些算法问题？

基于大数据集的机器学习和自然语言处理算法

2007，Google公司Brants *等基于MapReduce研究了一个基于2万亿个单词训练数据集的语言模型，比较了当时最先进的Kneser-Ney smoothing 算法与他们称之为“stupid backoff “的简单算法，最后发现，后者在小数据集时效果不佳，但在大数据集时，该算法最终居然产生了更好的语言模型！

结论：大数据集上的简单算法能比小数据集上的复杂算法产生更好的结果！

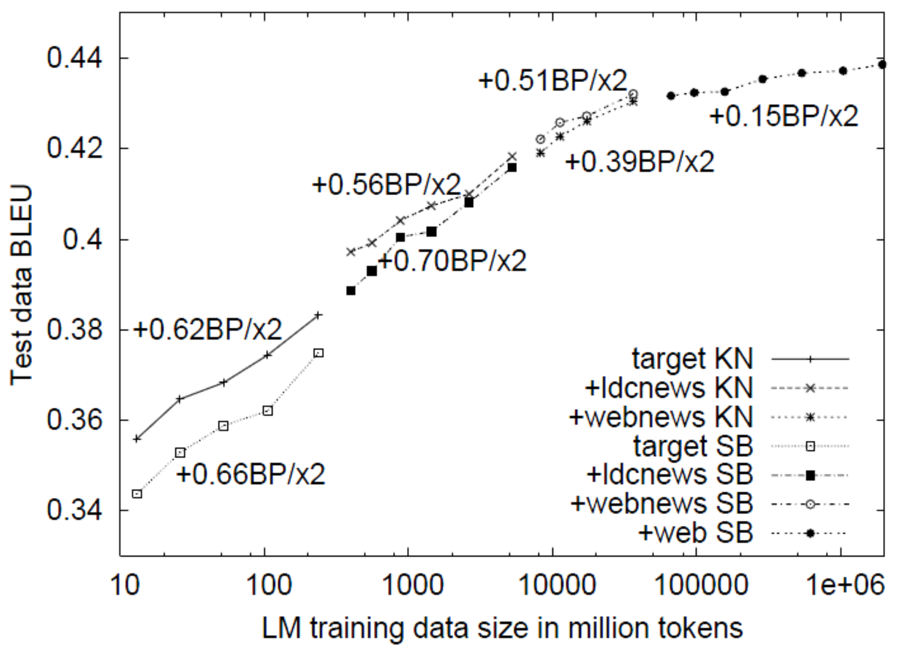


Figure 5: BLEU scores for varying amounts of data using Kneser-Ney (KN) and Stupid Backoff (SB).

* T. Brants, A. C. Popat, et al. **Large Language Models in Machine Translation.** In EMNLP-CoNLL 2007 - Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning

MapReduce可解决哪些算法问题？

机器学习与数据挖掘并行算法

本课题组基于MapReduce实现了大多数机器学习和数据挖掘算法研究

分类算法（KNN分类，贝叶斯分类，决策树分类...)

聚类算法

关联规则挖掘算法

HMM与EM算法

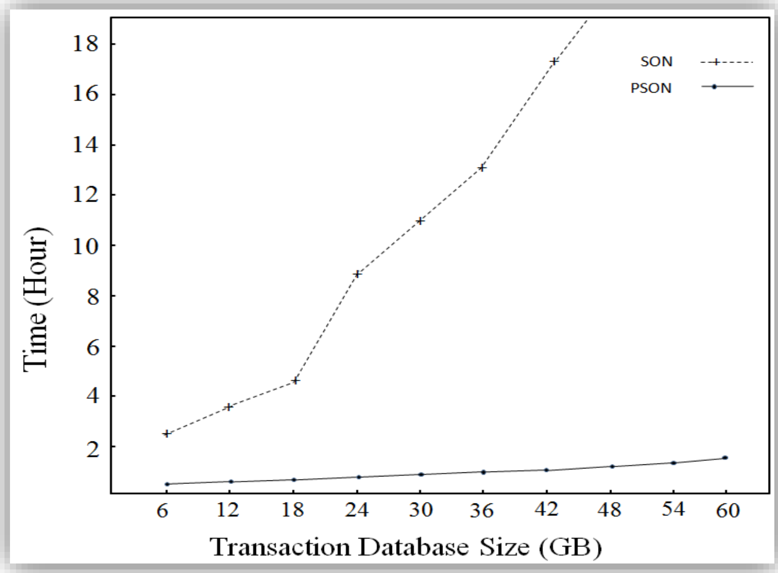
MapReduce可解决哪些算法问题？

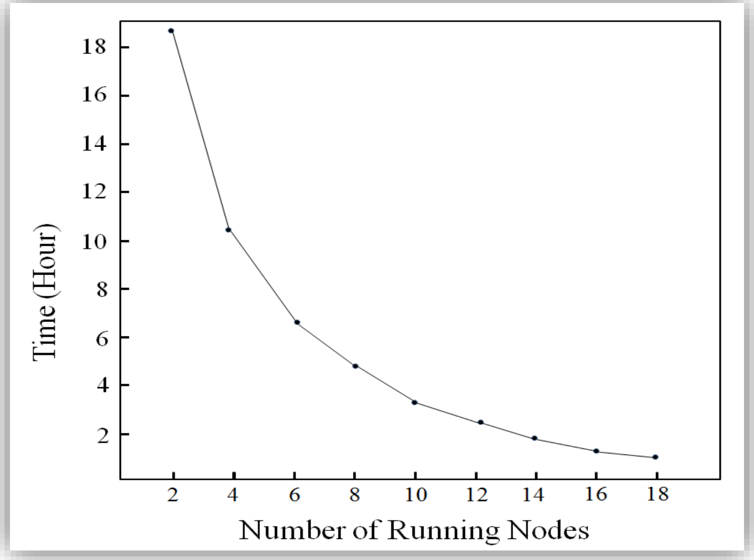
大规模数据频繁项集挖掘并行化算法

本研究组进行了基于MapReduce的频繁项集挖掘算法研究

PSON: A Parallelized SON Algorithm with MapReduce for Mining Frequent Sets
Tao Xiao, Shuai Wang, Chunfeng Yuan, Yihua Huang
The 4th International Symposium on Parallel Architectures, Algorithms and Programming (PAAP 2011), Tianjin, Dec. 9-11, 2011

根据基本的Apriori 算法和SON算法，研究实现并行化的频繁项集挖掘算法





MapReduce可解决哪些算法问题？

大规模重复文档检测并行算法

本研究组根据最新的Shingling算法[1]具有的计算高效但检测精确性低的特点，以及IMatch算法[2]检测准确较高但计算费时的特点，针对目前现有的英文文档重复检测方法在处理效果和适用性方面的不足，基于MapReduce研究并提出一种适用于中文文档重复检测和过滤的方法，有效提高了检测精度和计算性能。

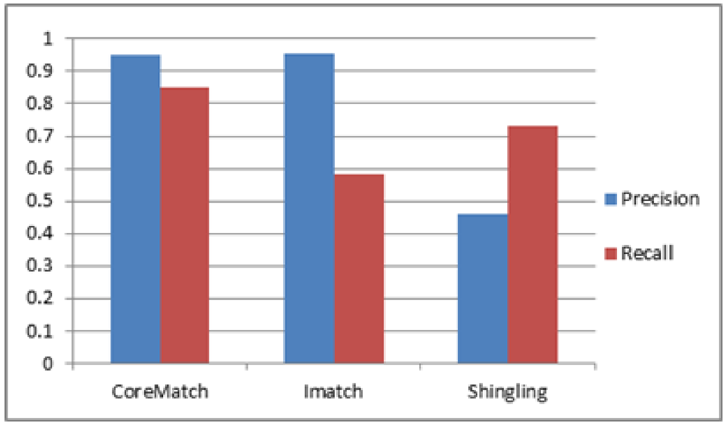


Figure 6 The precision and recall of CoreMatch, Imatch and Shingling.

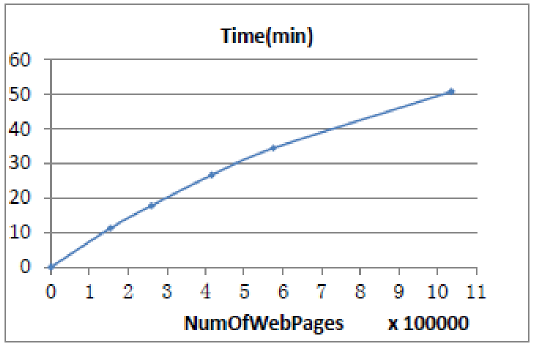


Figure 8 The time of our improved approach in processing huge number of Web pages range from 100,000 to 1,000,000.

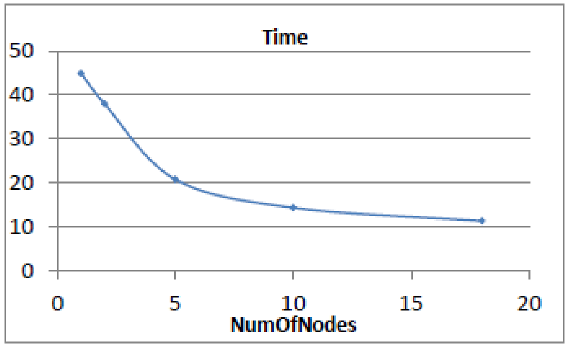
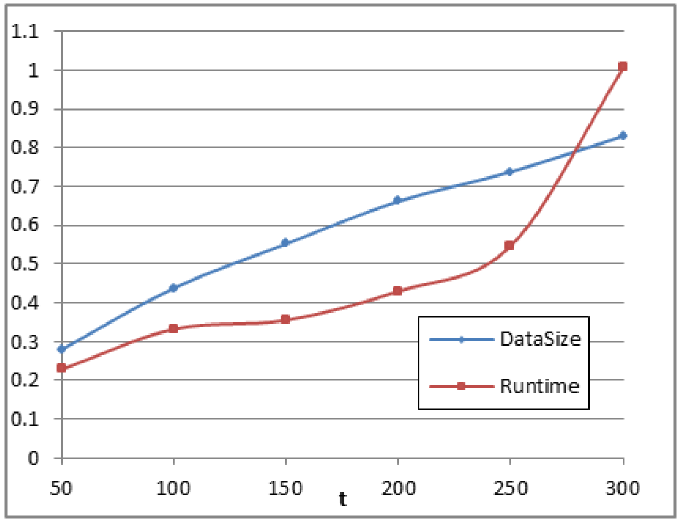


Figure 9 The processing time of Hadoop system with datanodes range from 1 to 18.



MapReduce可解决哪些算法问题？

大规模长基因序列比对算法

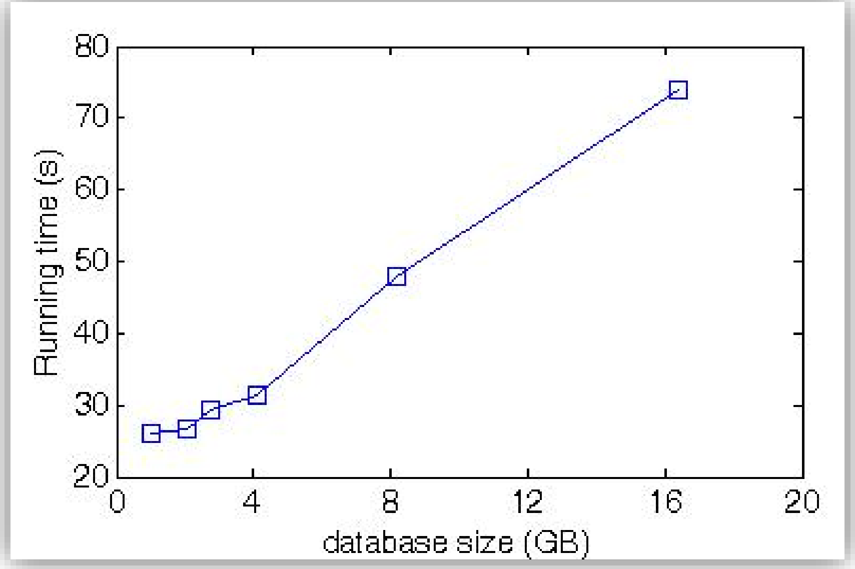
本研究组进行了基于MapReduce的大规模基因序列比对并行算法研究

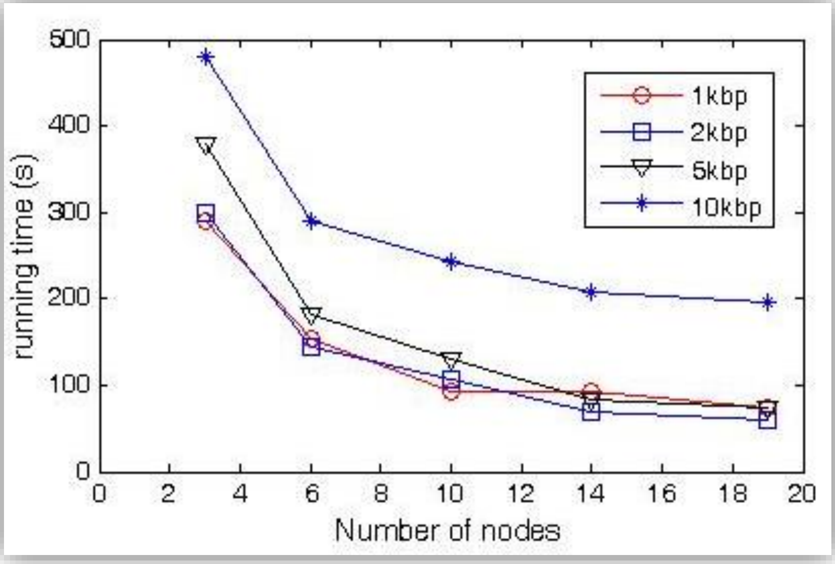
Parallization of BLAST with MapReduce

Xiaoliang Yang, Chunfeng Yuan, Yihua Huang

The 4th International Symposium on Parallel Architectures, Algorithms and Programming (PAAP 2011), Tianjin, Dec. 9-11, 2011

一个未知功能的待比对序列，需要与数据库中数十万已知基因序列逐一比对，这是一个非常耗时的计算工作。本课题组完成了基于MapReduce的并行化比对算法研究。





课程项目设计

MapReduce可解决哪些算法问题？

梁亚澜，李杰，钮鑫涛：Hadoop平台下覆盖表生成遗传算法参数配置启发式
李袁奎，刘文杰，王姜：使用Mapreduce框架进行软件代码分析

软件工程

黄刚， 陈光鹏：一种基于MapReduce的频繁闭项集挖掘算法研究及其实现
王苏琦，金龑，罗爱宝，王灵江：基于模型的协同过滤并行化算法
胡昊然，冯子陵，窦文科，刘晶晶：面向新浪微博的关注推荐系统
段轶
孙谊
刘敏
刘立 …法并行化的研究
王尧 …际关系的分析实验

金惠
…的分布式设计

机器学习
数据挖掘

社会网络
分析

机器翻译

- 选题覆盖了我系大多数研究方向
- 随着研究问题数据规模越来越大，越来越多的研究领域都需要使用并行计算技术提供新的计算方法
- 本课程的开设对推动我系各方向的研究将起到积极的作用

张旭，何良朋：P2P流媒体中的结点分簇与最短路径构造

网络通信

陈虎，笪庆小组：基于内容的图像搜索引擎EagleEye

多媒体检索

张航， 杨琬琪， 陶承恺：基于MapReduce的本体匹配技术

Web本体

江凯，顾小东，陆瑶，王团团小组：基于Hadoop的SQL查询工具

数据库

“大规模海量数据并行处理”课程项目设计

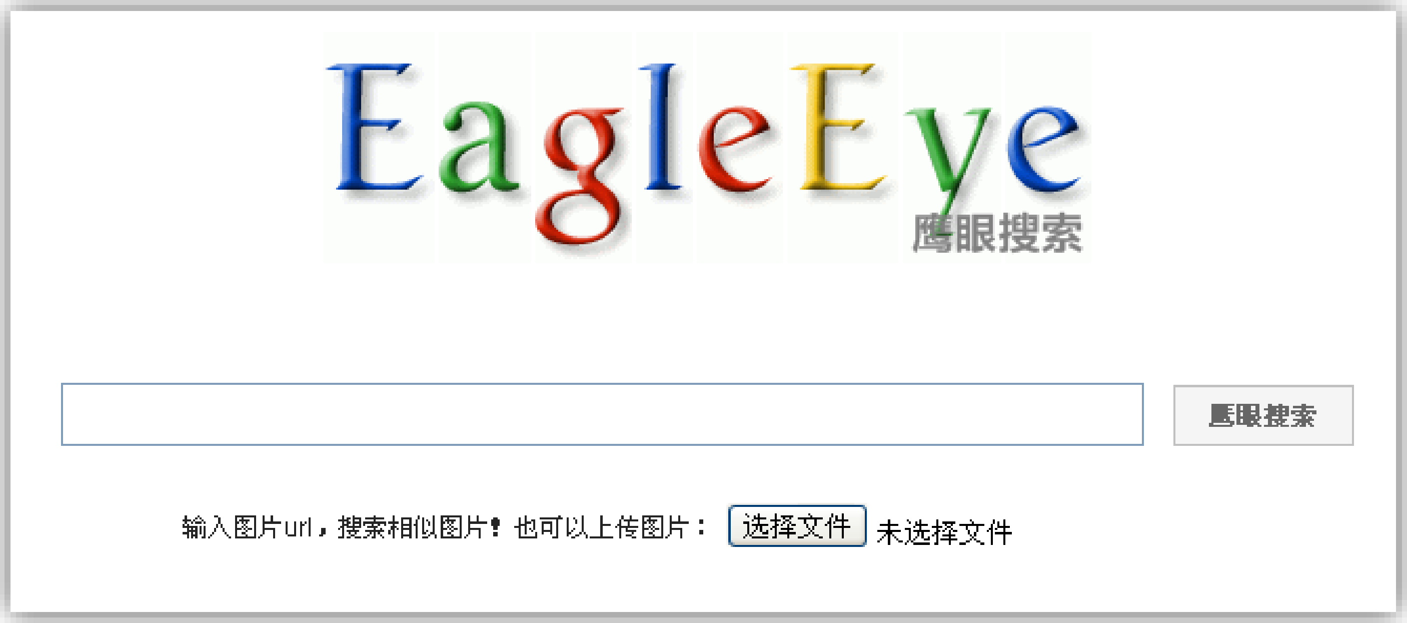
陈虎，笪庆小组：基于内容的图像搜索引擎EagleEye
—MapReduce海量数据并行处理项目

主要研究内容：

1、研究解决了有效的图像特征表示和快速提取方法：表示和提取图像的特征使其在基于内容的图像检索中能够更准确地表征不同图像之间的相似程度。

2、研究解决了基于MapReduce的海量图像特征索引和图像搜索算法

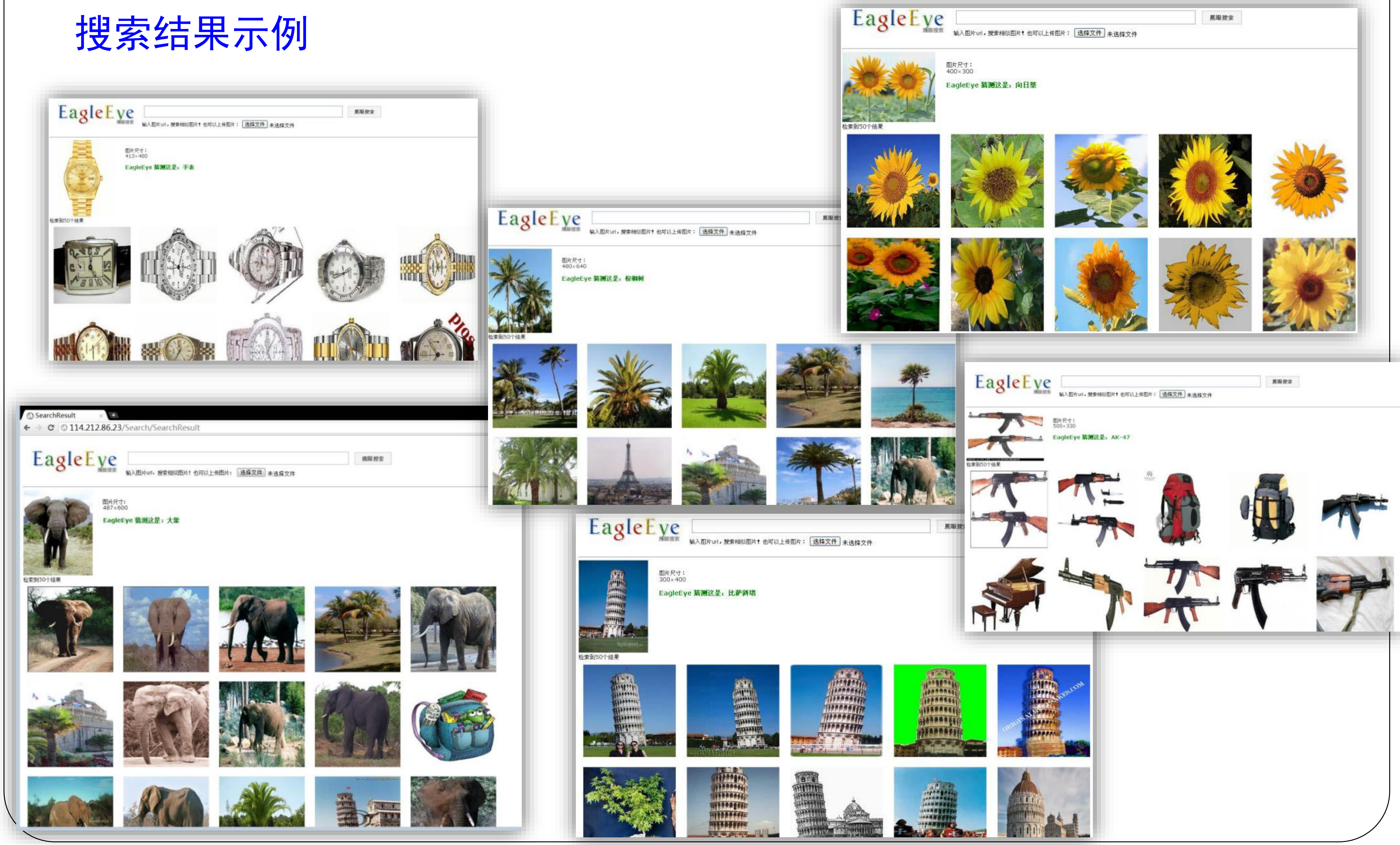
3、完成了一个基于内容的图像搜索EagleEye原型系统的设计实现



“大规模海量数据并行处理”课程项目设计

陈虎，笪庆小组：基于内容的图像搜索引擎EagleEye

搜索结果示例

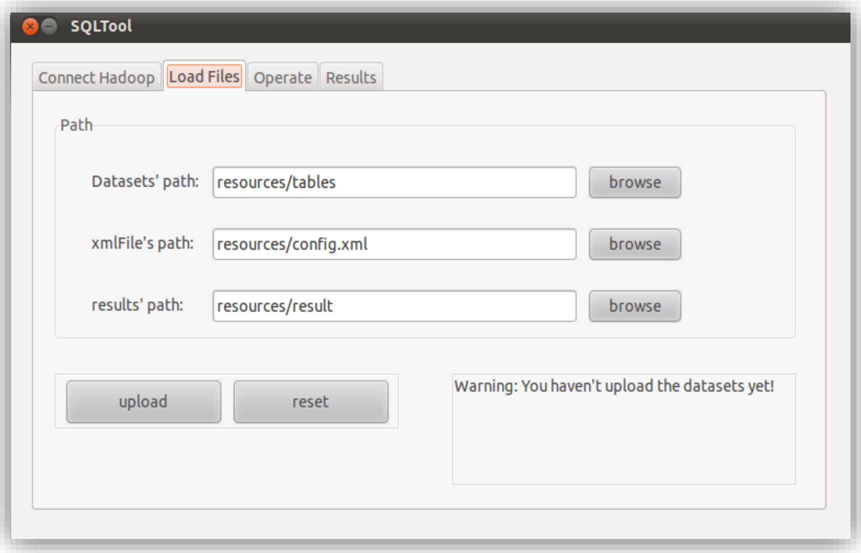


“大规模海量数据并行处理”课程项目设计

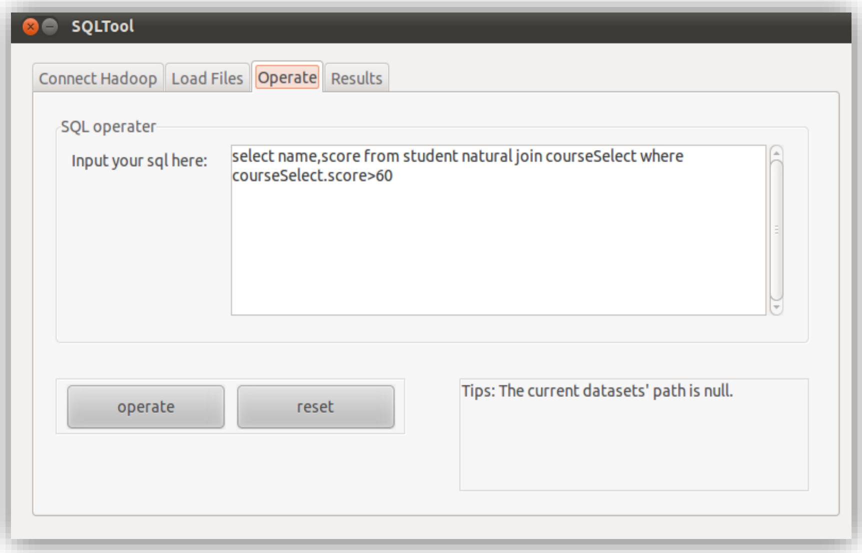
江凯，顾小东，陆瑶，王团团小组：**基于Hadoop的SQL查询工具**

主要研究了在Hadoop分布式文件系统环境下设计和模拟一个管理和查询结构化数据的原型数据库系统，主要技术内容包括：

- 设计了基于XML的数据库Schema的描述和处理方法
- 设计了基本的SQL查询语言
- 完成SQL语句的解析处理
- 完成SQL到关系代数的转换处理
- 基于MapReduce并行计算框架完成关系代数的并行化处理，提高计算效率
- 设计实现了一个原型的查询工具



Datasets' path: resources/tables
xmlFile's path: resources/config.xml
results' path: resources/result
Warning: You haven't upload the datasets yet!



Input your sql here:
```
select name,score from student natural join courseSelect where courseSelect.score>60
```
Tips: The current datasets' path is null.

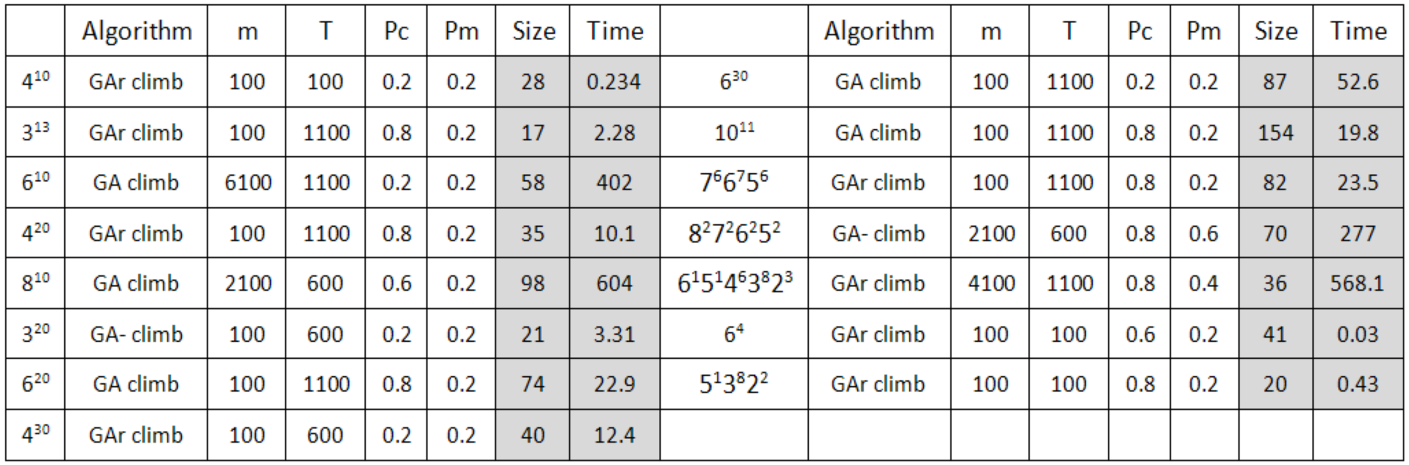
“大规模海量数据并行处理”课程项目设计

梁亚澜，李杰，钮鑫涛：Hadoop平台下覆盖表生成遗传算法参数配置启发式演化工具

主要研究内容：

1. 采用启发式演化方法对遗传算法的种群规模、进化机制、交叉概率、变异概率及其变种算法5个因素进行取值组合演化，系统地探索各个因素对遗传算法覆盖表生成效果的影响程度和性质，并以覆盖表规模和消耗时间为依据寻找出最佳配置
2. 遗传算法生成覆盖表的计算量极大，设种群规模为100，进化代数为1000，则完整的进化过程需运行遗传算法100*1000 ＝ 100,000次，以一次生成覆盖表的时间为1分钟为例，采用串行计算共需100000分钟，约71天。课题研究实现了基于Hadoop MapReduce的并行化遗传算法生成覆盖表算法，大大缩短了计算时间

表 3：各待测实例的最终最优配置和覆盖表生成结果



	Algorithm	m	T	Pc	Pm	Size	Time		Algorithm	m	T	Pc	Pm	Size	Time
4^{10}	GAr climb	100	100	0.2	0.2	28	0.234	6^{30}	GA climb	100	1100	0.2	0.2	87	52.6
3^{13}	GAr climb	100	1100	0.8	0.2	17	2.28	10^{11}	GA climb	100	1100	0.8	0.2	154	19.8
6^{10}	GA climb	6100	1100	0.2	0.2	58	402	$7^6 6^7 5^6$	GAr climb	100	1100	0.8	0.2	82	23.5
4^{20}	GAr climb	100	1100	0.8	0.2	35	10.1	$8^2 7^2 6^2 5^2$	GA- climb	2100	600	0.8	0.6	70	277
8^{10}	GA climb	2100	600	0.6	0.2	98	604	$6^1 5^1 4^6 3^8 2^3$	GAr climb	4100	1100	0.8	0.4	36	568.1
3^{20}	GA- climb	100	600	0.2	0.2	21	3.31	6^4	GAr climb	100	100	0.6	0.2	41	0.03
6^{20}	GA climb	100	1100	0.8	0.2	74	22.9	$5^1 3^8 2^2$	GAr climb	100	100	0.8	0.2	20	0.43
4^{30}	GAr climb	100	600	0.2	0.2	40	12.4								

基于本课程设计项目的研究成果作者和导师发表了两篇学术论文

1. 梁亚澜，聂长海，覆盖表生成的遗传算法配置参数优化 2011年6月，计算机学报已录用.
2. Liang Yalan, Changhai Nie, Jonathan M. Kau_man, Gregory M. Kapfhammer, and Hareton Leung. **Empirically identifying the best genetic algorithm for covering array generation**. In Proceedings of the 3rd International Symposium on Search Based Software Engineering, Szeged, Hungary, September 2011

MapReduce可解决哪些算法问题？

MapReduce应用案例

纽约时报历史文章处理

In 2007, converting 11 million image documents from the New York Times archive between 1851 and 1922 for free access

扫描存储的图像文件TIFF转换为PDF后，上网提供联机的PDF下载访问

- Derek Gottfrid, a software programmer at the Times, thought this was a perfect opportunity to use the Amazon Web Services (AWS) and Hadoop.
- Storing and serving the final set of PDFs from Amazon's Simple Storage Service (S3) was already deemed a more cost-effective approach than scaling up the storage back-end of the website. Why not process the PDFs in the AWS cloud as well?
- Derek copied the 4 TB of TIFF images into S3. He "started writing code to pull all the parts that make up an article out of S3, generate a PDF from them and store the PDF back in S3. This was easy enough using the JetS3t —Open Source Java toolkit for S3, iText PDF Library and installing the Java Advanced Image Extension ."
- After tweaking his code to work within the Hadoop framework, Derek deployed it to Hadoop running on 100 nodes in Amazon's Elastic Compute Cloud (EC2). The job ran for 24 hours and generated another 1.5 TB of data to be stored in S3.
- At 10 cents per instance per hour, the whole job ended up **costing only $240** (100 instances x 24 hours x $0.10) in computation

Cite from "Hadoop in Action", Chuck Lam, 2010 , Manning Publications

MapReduce应用案例

中国移动通信数据挖掘

China Mobile looks to data warehousing and mining of this data to extract insights for improving marketing operations, network optimization, and service optimization. Some typical applications include

- Analyzing user behavior
- Predicting customer churn
- Analyzing service association
- Analyzing network quality of service (QOS)
- Analyzing signaling data
- Filtering

- 原来使用由著名供应商提供的专用的商业数据挖掘系统，但该系统的单服务器构架严重限制了大数据量挖掘处理。
- 一个分支机构使用了8 核、32 GB 内存、一个磁盘阵列的Unix服务器，但仅能处理1.4百万个用户的行为数据，或者仅仅本分支机构10%的用户数据,而且处理时间很长

MapReduce可解决哪些算法问题？

MapReduce应用案例

中国移动通信数据挖掘

然后他们决定基于Hadoop重新做一个数据挖掘系统

- Datanode/TaskTracker —单路 4核 Xeon 2.5 GHz CPU, 8 GB RAM, 4 x 250 GB SATA disks
- Namenode/JobTracker —双路 2核 AMD Opteron 2.6 GHz CPU, 16 GB RAM, 4 x 146 GB SAS

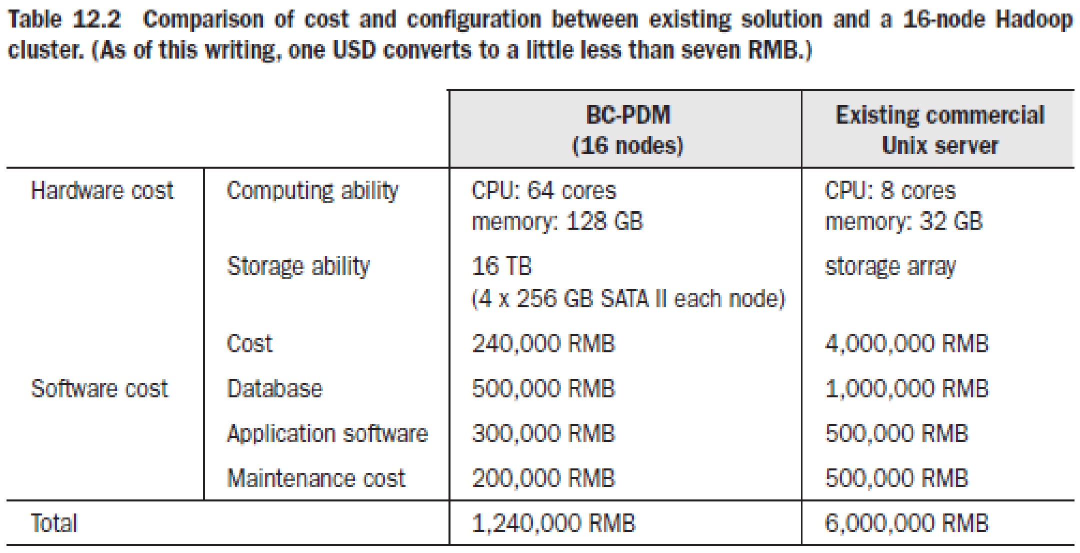
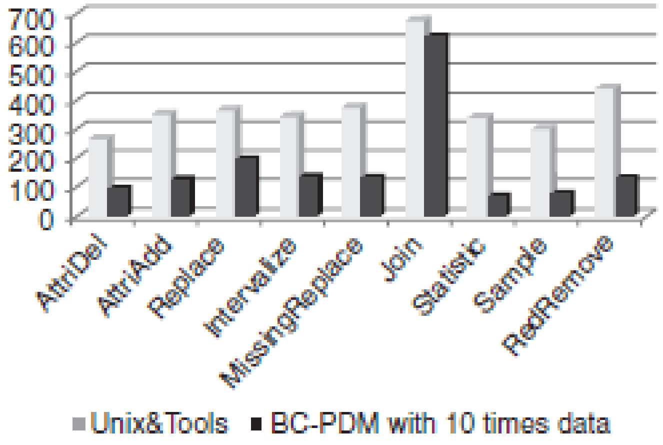


Table 12.2 Comparison of cost and configuration between existing solution and a 16-node Hadoop cluster. (As of this writing, one USD converts to a little less than seven RMB.)

		BC-PDM (16 nodes)	Existing commercial Unix server
Hardware cost	Computing ability	CPU: 64 cores memory: 128 GB	CPU: 8 cores memory: 32 GB
	Storage ability	16 TB (4 x 256 GB SATA II each node)	storage array
	Cost	240,000 RMB	4,000,000 RMB
Software cost	Database	500,000 RMB	1,000,000 RMB
	Application software	300,000 RMB	500,000 RMB
	Maintenance cost	200,000 RMB	500,000 RMB
Total		1,240,000 RMB	6,000,000 RMB



价格比较：1/5的价钱

10倍数据时处理能力
平均比原有方法快2倍以上

MapReduce应用案例

IBM智力问答竞赛机器人Waston

IBM 智力竞赛机器人Watson是一个基于MapReduce

数据并行处理和统计模型自然语言处理的成功应用。

IBM Watson

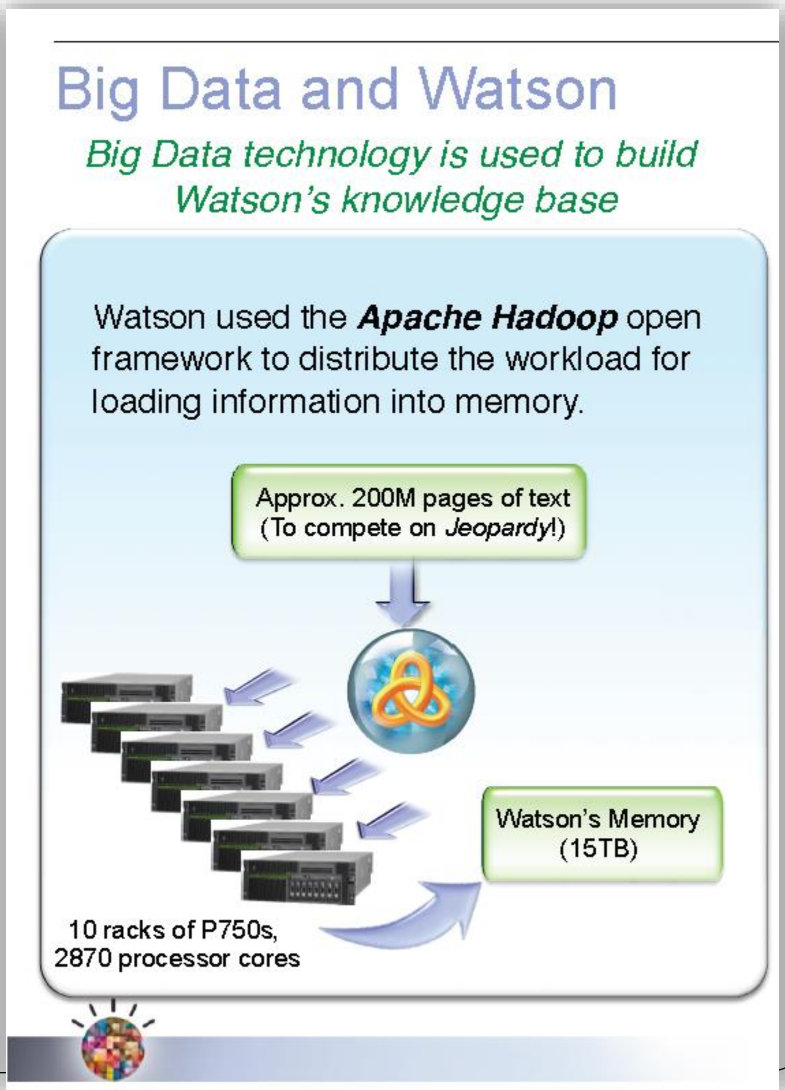
IBM Watson is a breakthrough in analytic innovation, but it is only successful because of the quality of the information from which it is working.

©2011 BW@ IBM Corporation

MapReduce应用案例

IBM智力问答竞赛机器人Waston

Watson收集了2亿页知识文本数据，并基于Hadoop MapReduce并行处理集群进行数据分析，采用了优化的并行体系结构和优化的知识和自然语言处理算法，可在1秒内完成对大量非结构化信息的检索，并实时回答知识竞赛问答题。



Big Data and Watson

Big Data technology is used to build Watson's knowledge base

Watson used the ***Apache Hadoop*** open framework to distribute the workload for loading information into memory.

MapReduce可解决哪些算法问题？

MapReduce应用算法专著

1.Mining of Massive Datasets

2010, Anand Rajaraman(Kosmix, Inc), Jeffrey D. Ullman(Stanford Univ.)

主要介绍基于MapReduce的大规模数据挖掘相关的技术和算法，尤其是Web或者从Web导出的数据

Ch3. Similarity search, including the key techniques of minhashing and localitysensitive hashing.
Ch4. Data-stream processing and specialized algorithms for dealing with data that arrives so fast it must be processed immediately or lost.
Ch5. The technology of search engines, including Google’s PageRank, link-spam detection, and the hubs-and-authorities approach(a link analysis algorithm： Hyperlink-Induced Topic Search (HITS)).
Ch6. Frequent-itemset mining, including association rules, market-baskets, the A-Priori Algorithm and its improvements (a classic algorithm for learning association rules).
Ch7. Algorithms for clustering very large, high-dimensional datasets.
Ch8. Two key problems for Web applications: managing advertising and recommendation systems.

MapReduce可解决哪些算法问题？

MapReduce应用算法专著

2. Data-Intensive Text Processing with MapReduce

Jimmy Lin and Chris Dyer，2010，University of Maryland, College Park

主要介绍基于MapReduce的大规模文档数据处理技术和算法

Ch4. Inverted Indexing for Text Retrieval
Ch5. Graph Algorithms
- Parallel Breadth-First Search
- PageRank

Ch6. EM Algorithms for Text Processing
- EM, HMM
- Case Study: Word Alignment for Statistical Machine Translation

Ch.6. MapReduce算法设计

1. MapReduce可解决哪些算法问题？
2. 回顾：MapReduce处理流程
3. MapReduce排序算法
4. MapReduce单词同现分析算法
5. MapReduce文档倒排索引算法
6. 专利文献数据分析
7. 实验3：文档倒排索引实验

2. 回顾: MapReduce 处理流程

MapReduce处理流程

1. map(K1,V1)->[(K2,V2)]
2. shuffle and sort
3. reduce(K2,[V2]) -> [(K3,V3)]

([…] denotes a list)

Any algorithm that you wish to develop must be expressed in terms of such rigidly-defined components

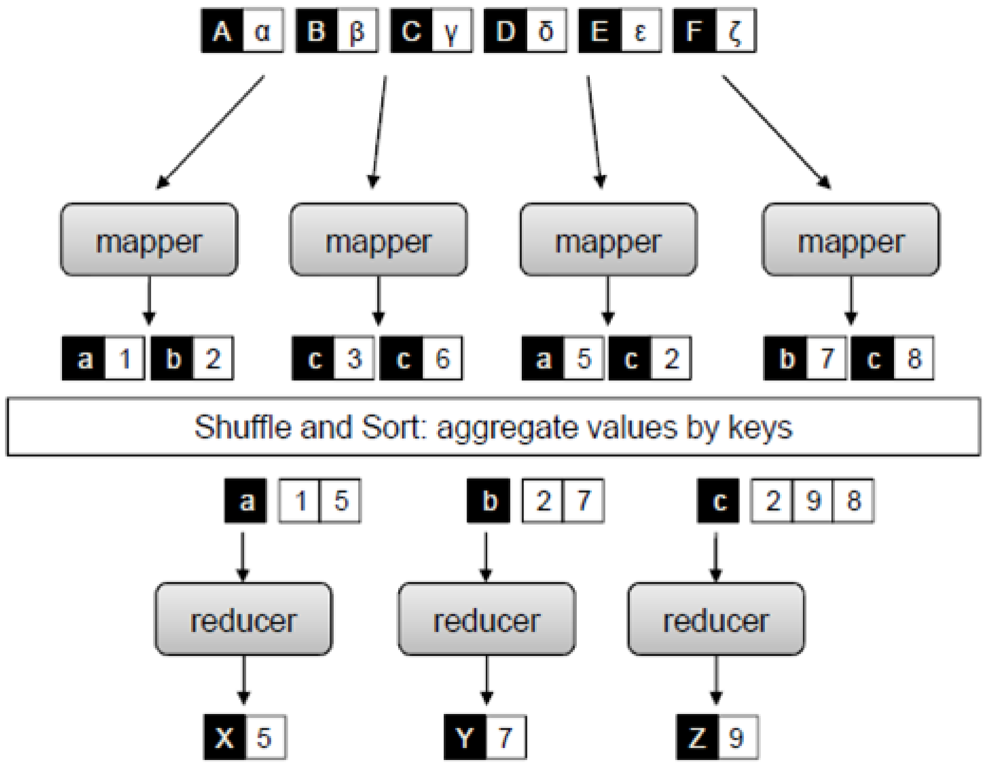


Figure taken from Jimmy Lin’ s manuscript, April 2010

回顾：MapReduce 处理流程

可编程控制的阶段

- Mapper
  - Initialize: setup()
  - map(): It is called once for each key/value pair in the input split. The default is the identity function.
  - Close: cleanup()
- Shuffle
  - shuffle phase needs the ***Partitioner*** to route the output of mapper to reducer
  - *Partitioner* controls the partitioning of the keys of the intermediate map-outputs. The key is used to derive the partition, typically by a hash function.
  - The total number of partitions is the same as the number of reduce tasks for the job.
  - *HashPartitioner* is the default Partitioner (hadoop v0.21.0)

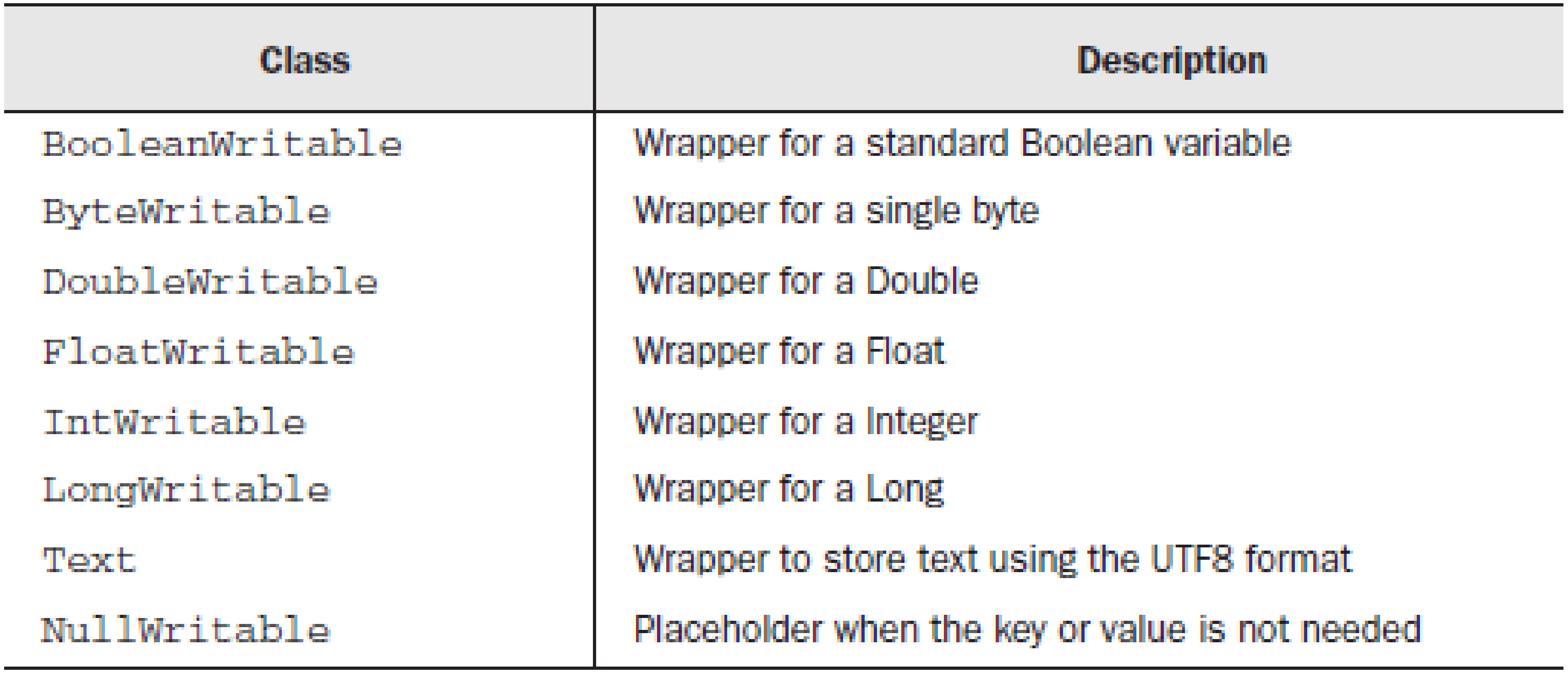
回顾：MapReduce处理流程

可编程控制的阶段

- Sort
  - we can controls how the keys are sorted before they are passed to the Reducer by using a customized comparator
- Reducer
  - Initialize: setup(), called once when creating the instance of Reducer, can be used to do some initialization work, e.g., open a file or create a dababase connection, for reduce method to use.
  - reduce(): It is called once for each key. The default implementation is an identity function.
  - Close: cleanup(), called once when whole reduce process terminates, can be used to do some clean-up work for reducer, e.g., close an open file or a database connection.

Hadoop常用的key-value对的数据类型

- 这些数据类型都实现了WritableComparable接口，以便进行网络传输和文件存储，以及进行大小比较



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
`Text`	Wrapper to store text using the UTF8 format
`NullWritable`	Placeholder when the key or value is not needed

Hadoop编程API文档

Hadoop基本类库API：
http://hadoop.apache.org/common/docs/r0.21.0/api/index.html

Hadoop HDFS类库API：
http://hadoop.apache.org/hdfs/docs/r0.21.0/api/index.html

Hadoop MapReduce类库API：
http://hadoop.apache.org/mapreduce/docs/r0.21.0/api/index.html

Ch.6. MapReduce算法设计

1. MapReduce可解决哪些算法问题？
2. 回顾：MapReduce处理流程
3. MapReduce排序算法
4. MapReduce单词同现分析算法
5. MapReduce文档倒排索引算法
6. 专利文献数据分析
7. 实验2：文档倒排索引实验

3. 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

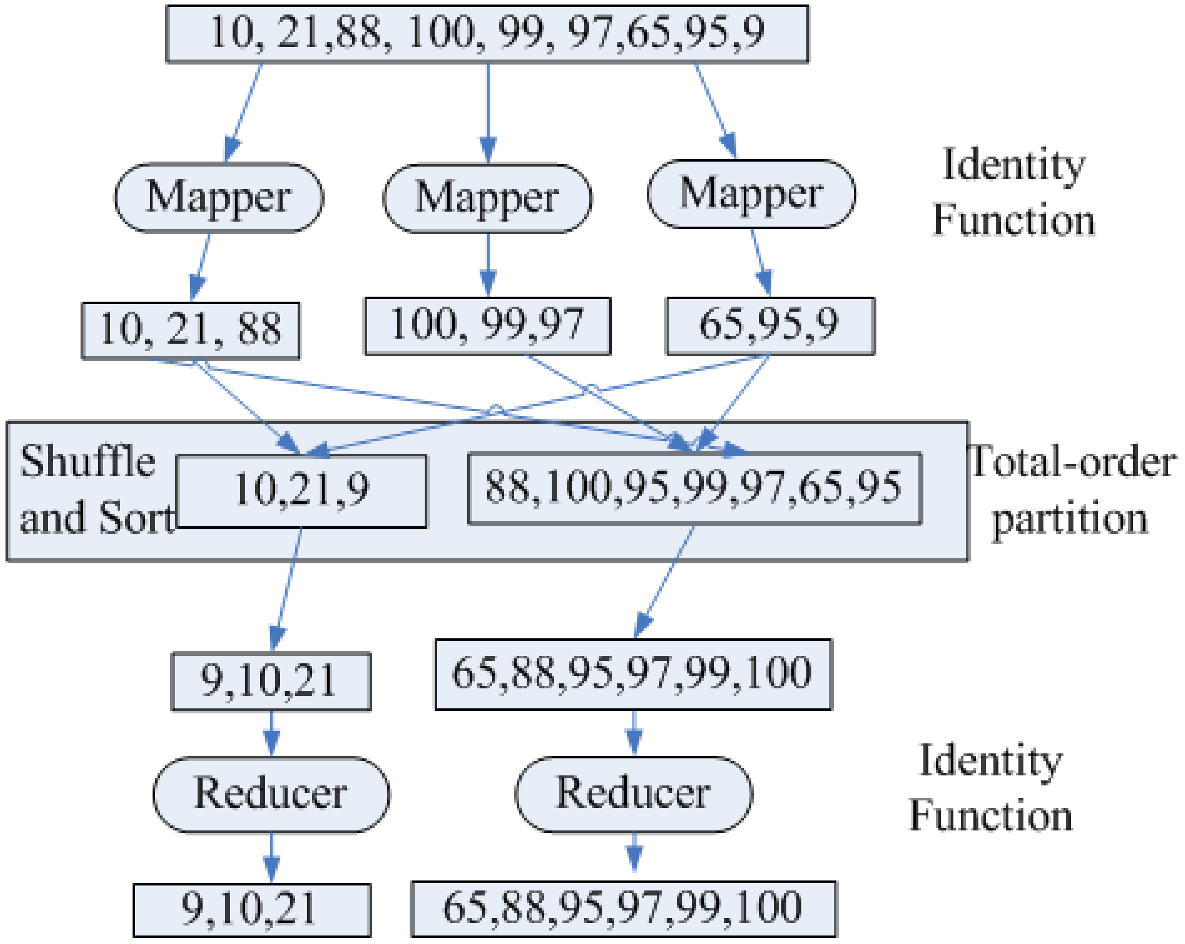


Figure: A simple MapReduce sorting example with 3 mappers, 2 reducers

Is there any problem here?

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.

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.

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
- 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

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,*)

TotalOrderPartitioner

- 高效的划分模型
  - 若Key 的数据类型是BinaryComparable的，即两个对象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

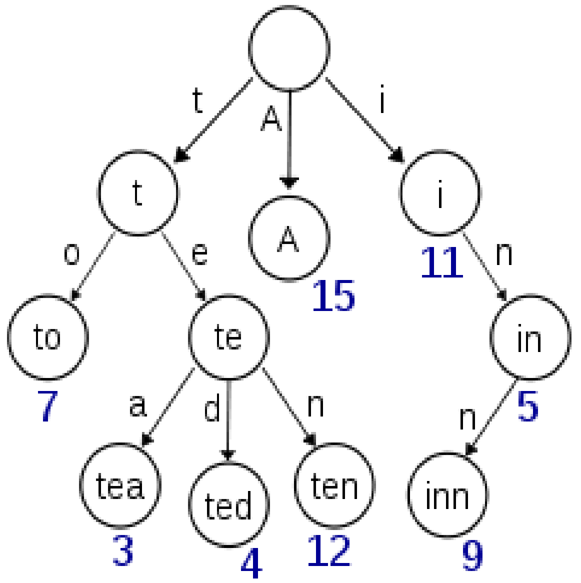


Figure: An example of Trie
Taken From wiki

Ch.5. MapReduce算法设计

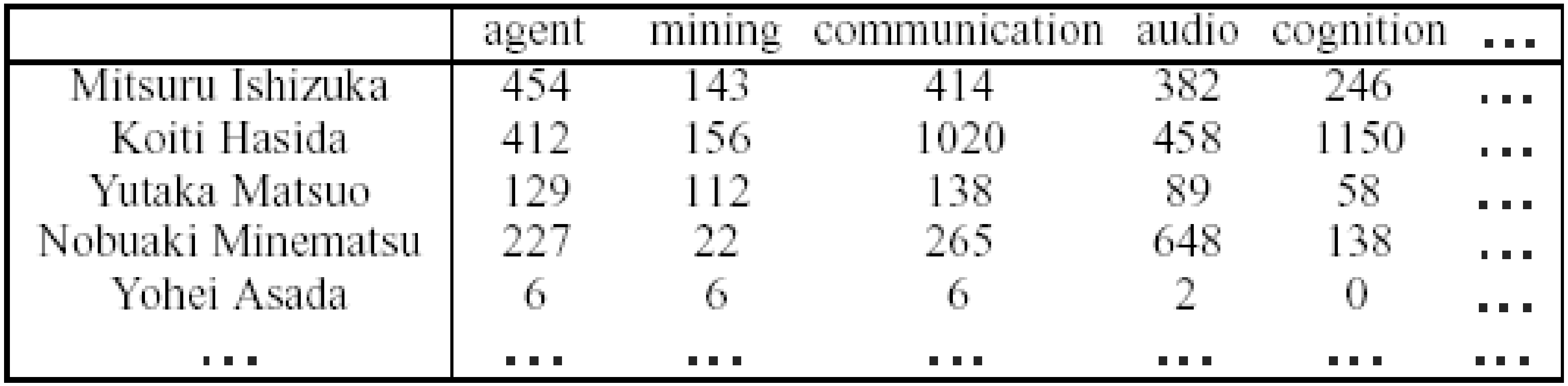
1. MapReduce可解决哪些算法问题？
2. 回顾：MapReduce处理流程
3. MapReduce排序算法
4. MapReduce单词同现分析算法
5. MapReduce文档倒排索引算法
6. 专利文献数据分析
7. 实验3：文档倒排索引实验

4. 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:

```
1: class Mapper
2:     method Map(docid a, doc d)
3:         for all term w $\in$ doc d do
4:             for all term u $\in$ 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 $\leftarrow$ 0
4:         for all count c in counts [c1, c2,…] do
5:             s $\leftarrow$ s + c            //Sum co-occurrence counts
6:         Emit(pair p, count s)
```

Algo. from: Jimmy Lin’ s manuscript ,April 2010

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)

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

Ch.6. MapReduce算法设计

1. MapReduce可解决哪些算法问题？
2. 回顾：MapReduce处理流程
3. MapReduce排序算法
4. MapReduce单词同现分析算法
5. **MapReduce文档倒排索引算法**
6. 专利文献数据分析
7. 实验2：文档倒排索引实验

5. 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

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

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);
                        }
            }
}
```

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 ")
}
```

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();          }
    }
}
```

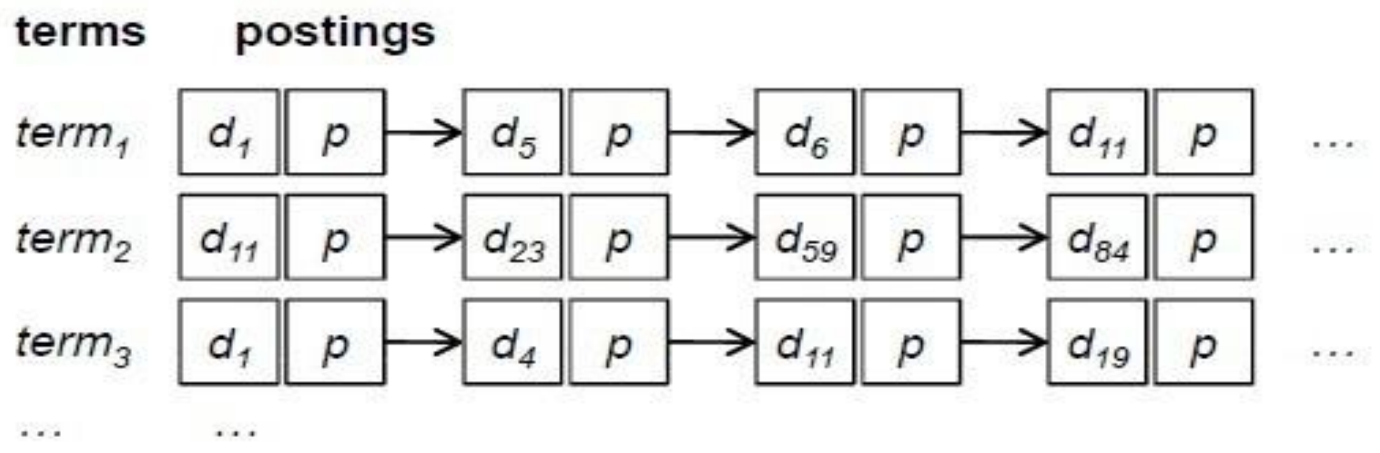
带词频等属性的文档倒排算法

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）

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

带词频等属性的文档倒排算法

基本的倒排索引结构


terms
postings
term1
term2
term3
d1 p → d5 p → d6 p → d11 p …
d11 p → d23 p → d59 p → d84 p …
d1 p → d4 p → d11 p → d19 p …
…
…

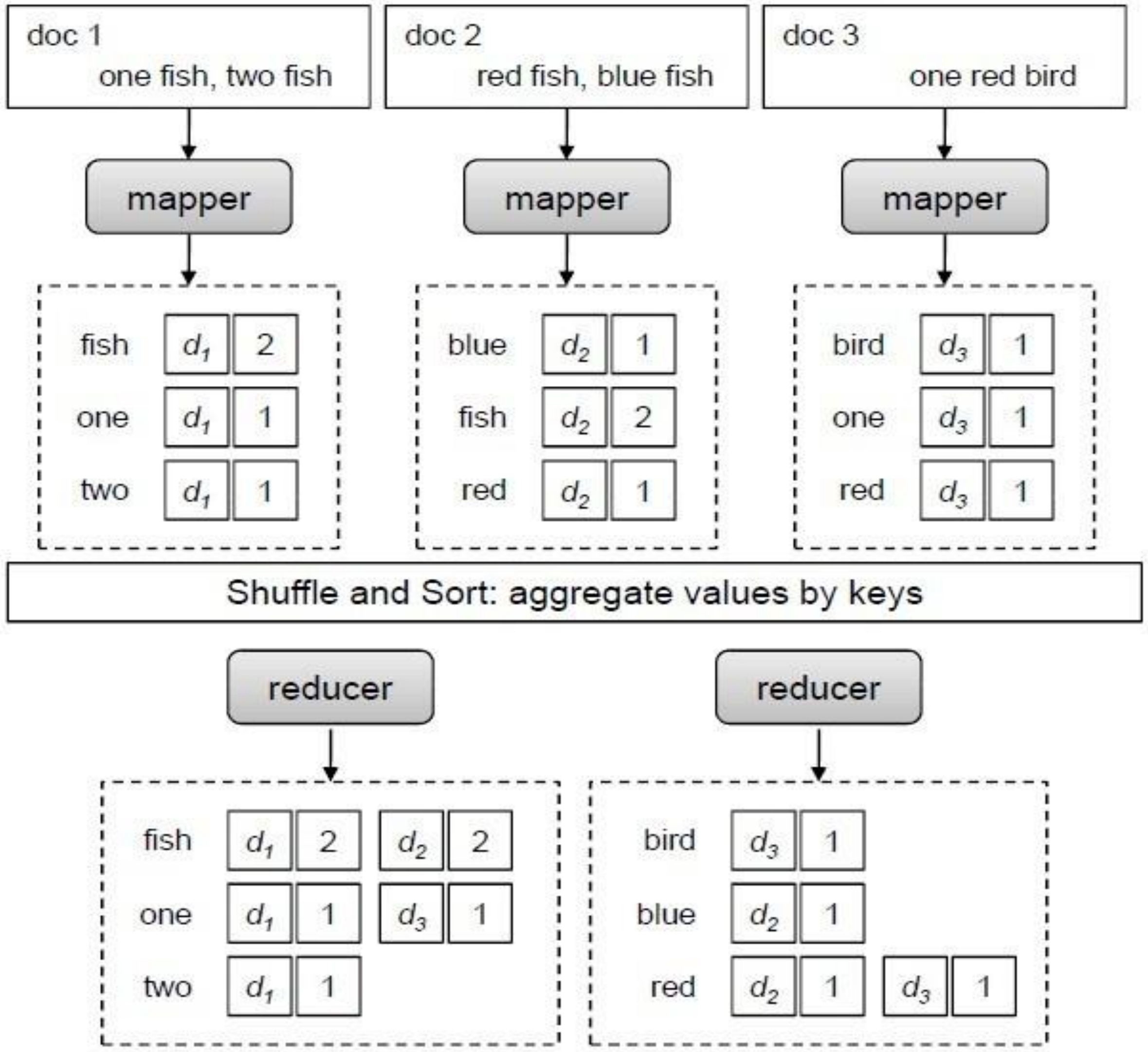
- 一个倒排索引由大量的postings list构成
- 一个postings list由多个posting构成(按doc id排序)
- 一个postings list与一个term关联
- 一个posting 包含一个document id和一个payload
- payload上载有term在document中出现情况相关的信息(e.g. term frequency, positions, term properties)
- 同时还有对应Web文档到其URL的映射doc_id→URL

带词频属性的文档倒排算法

Map和Reduce实现伪代码

```
1: class Mapper
2:     procedure Map(docid dn, doc d)
3:         F ← new AssociativeArray
4:         for all term t ∈ doc d do
5:             F{t} ← F{t} + 1
6:         for all term t ∈ F do
7:             Emit(term t, posting <dn, F{t}>)
1: class Reducer
2:     procedure Reduce(term t, postings [<dn1, f1>, <dn2, f2>…])
3:         P ← new List
4:         for all posting <dn, f> ∈ postings [<dn1, f1>, <dn2, f2>…] do
5:             Append(P, <dn, f>)
6:         Sort(P)
7:         Emit(term t; postings P)
```

带词频属性的文档倒排算法



A simple example
posting(docid, tf)

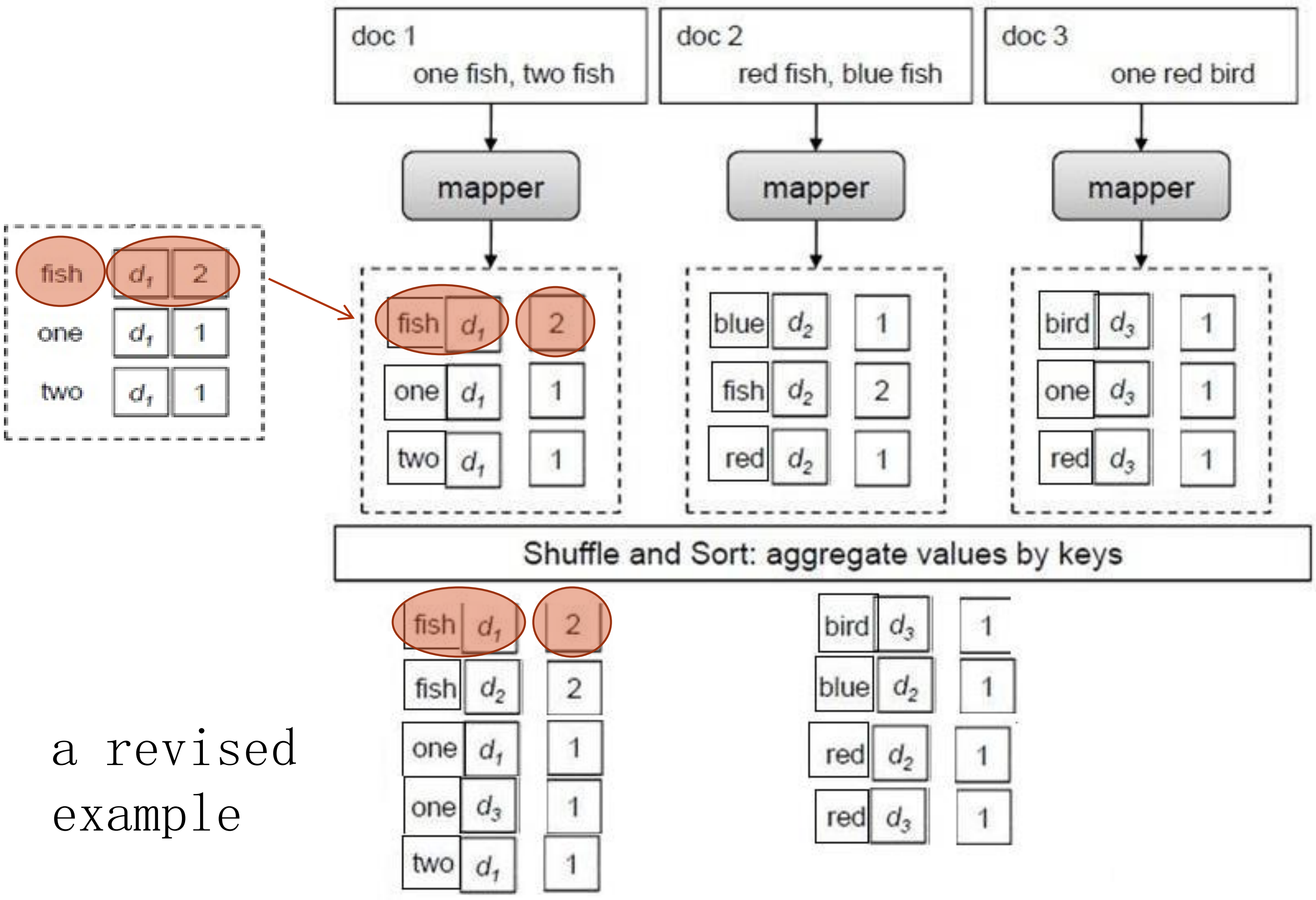
带词频属性的文档倒排算法

- 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 to 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



a revised example

可扩展的带词频属性的文档倒排算法

- Mapper

```
1: class Mapper
2:     method Map(docid dn; doc d)
3:         $F \leftarrow$ new AssociativeArray
4:         for all term t $\in$ doc d do
5:             F{t} $\leftarrow$ F{t} + 1
6:         for all term t $\in$ F do
7:             Emit(tuple<t, dn>, tf F{t})
```

问题：当对键值对进行shuffle处理以传送给合适的Reducer时，将按照新的键<t,dn> 进行排序和选择Reducer，因而同一个term的键值对可能被分区到不同的Reducer！

解决方案：定制Partitioner来解决这个问题

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)
  - **How?** 基本思想是把组合键<term, docid> 临时拆开，“蒙骗”partitioner按照<term>而不是<term, docid>进行分区选择正确的Reducer，这样可保证同一个term下的一组键值对一定被分区到同一个Reducer
    - Class **NewPartitioner** extends HashPartitioner<K,V>

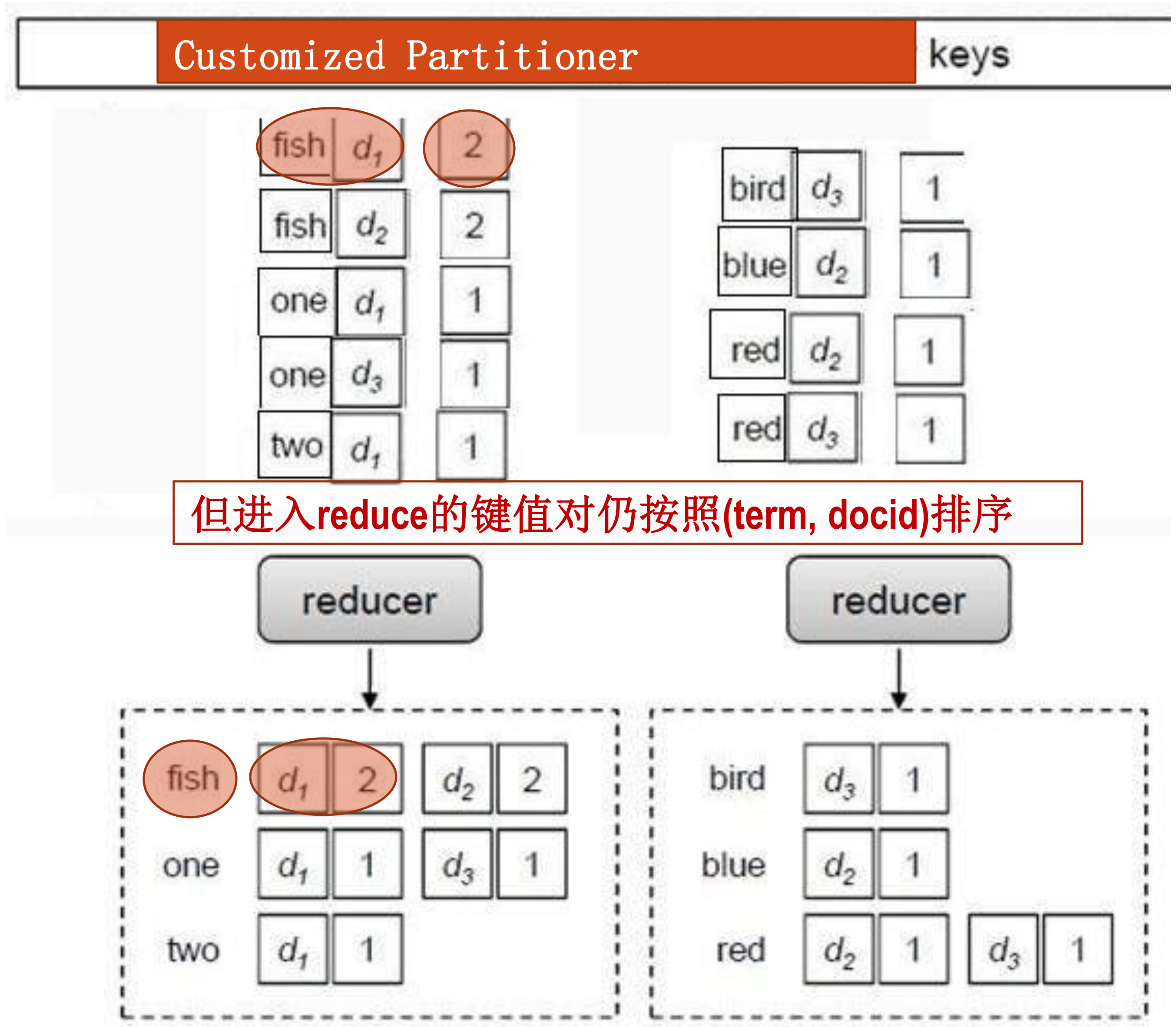
```
 // org.apache.hadoop.mapreduce.lib.partition.HashPartitioner
{   // override the method
    getPartition(K key, V value, int numReduceTasks)
    {   term = key. toString().split(“,”)[0]; //<term, docid>=>term
        super.getPartition(term, value, numReduceTasks);
    }
 }
```

    - Set the customized partitioner in job configuration

    Job. setPartitionerClass(NewPartitioner)

文档倒排索引算法

A revised example (cont.)



可扩展的带词频等属性的文档倒排算法 (cont.)

- **Reducer**

```
1: class Reducer
2:     method Setup // 初始化
3:         $t_{prev} \leftarrow \emptyset$;
4:         $P \leftarrow$ new PostingsList
5:     method Reduce(tuple <t, n>, tf [f])
6:         if $t \neq t_{prev} \wedge t_{prev} \neq \emptyset$ then
7:             Emit($t_{prev}$, $P$)
8:             $P$.Reset()
9:         $P$.Add(<n, f>)
10:        $t_{prev} \leftarrow t$
11:    method Close
12:        Emit(t, $P$)
```

用于输出最后一次未得到输出的<t, P>

可扩展的带词频等属性的文档倒排算法 (cont.)

- Extensions
  - 单词形态还原(e.g. ‘books’ -> ‘book’, …)
  - removing stopwords (common words such as ‘the’, ‘a’, ‘of’, etc)

MapReduce算法设计总结

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

7. 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”

本例引自Chuck Lam，Hadoop in Action，2010 ，Manning Publications

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
...
```

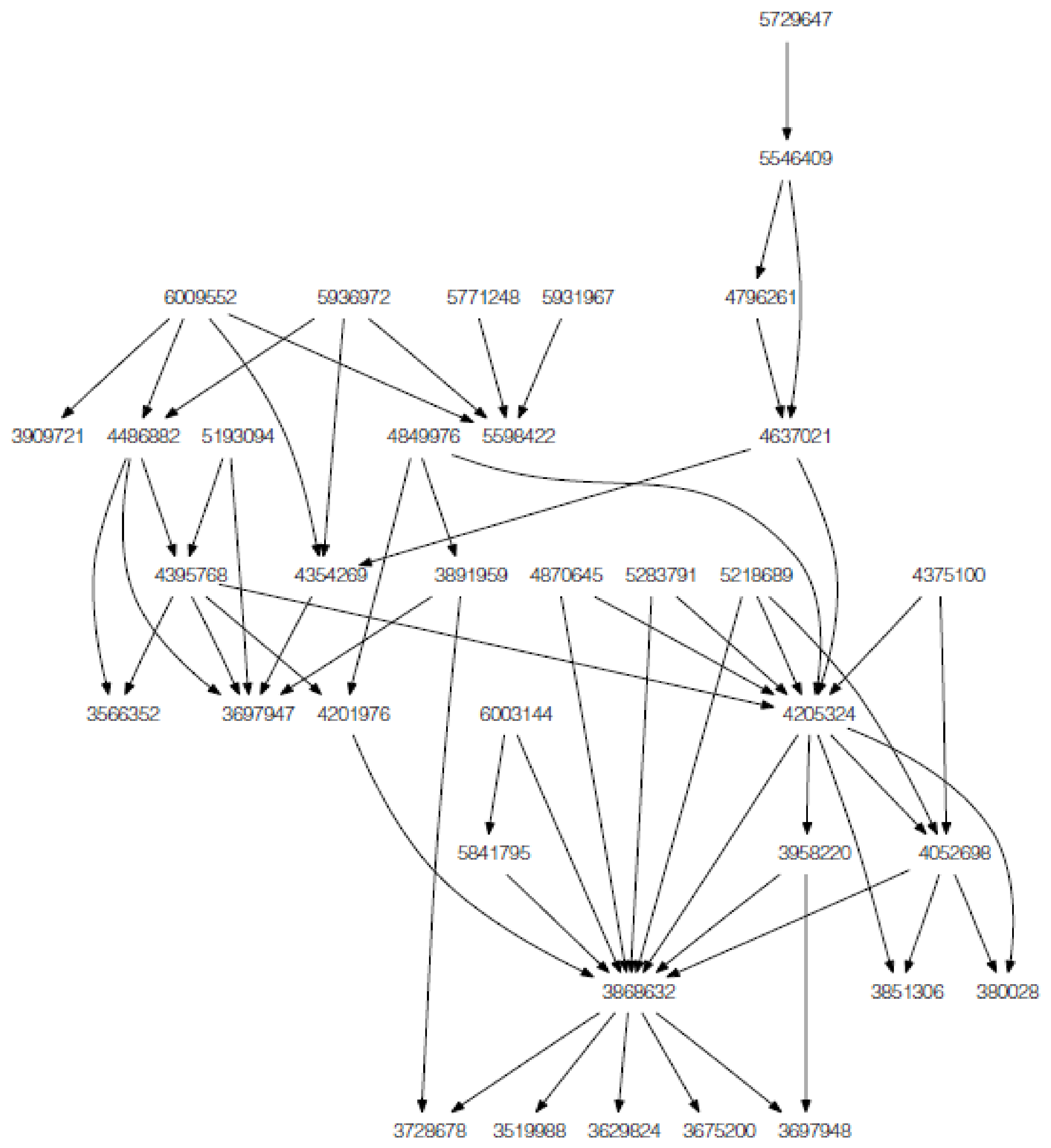
3858241引用了
956203，1324234，3398406，3557384, 3634889

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,,,,,,,
```

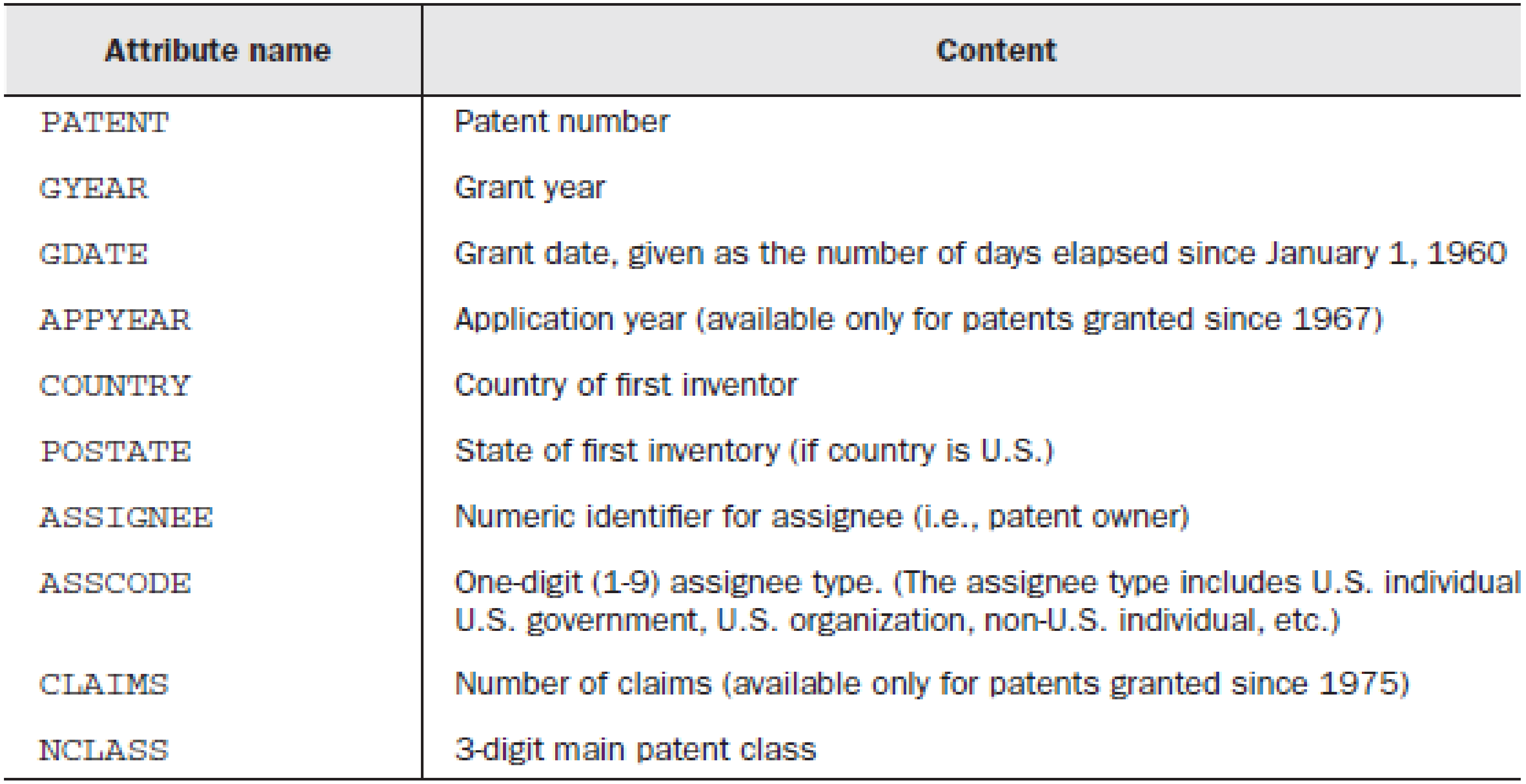
专利文献数据分析



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).

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, citing专利号2,…”
}
```

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

……

专利被引次数统计

- Map Class

```
IntWritable one = new IntWritable(1);
public static class MapClass extends Mapper<LongWritable, Text,
                                                        Text, IntWritable>
{
   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: 1
}
```

专利被引次数统计

- Reduce Class

```
public static class ReduceClass extends Reducer<Text, IntWritable, Text, Text>
{
    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: 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

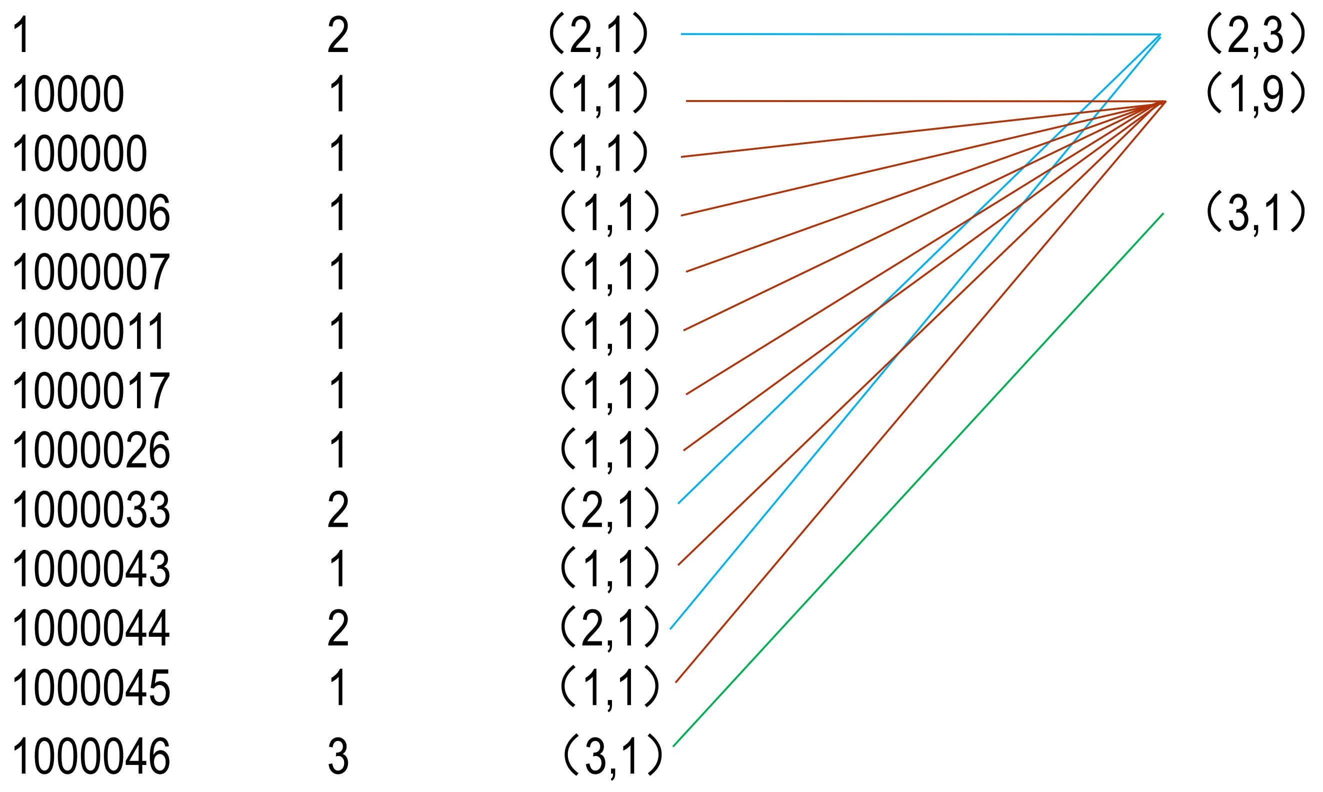
```
1           2
10000       1
100000      1
1000006     1
1000007     1
1000011     1
1000017     1
1000026     1
1000033     2
1000043     1
1000044     2
1000045     1
1000046     3
```

……

专利被引次数直方图统计

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，最后可画出统计直方图

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



1	2	（2,1）	（2,3）
10000	1	（1,1）	（1,9）
100000	1	（1,1）	
1000006	1	（1,1）	（3,1）
1000007	1	（1,1）	
1000011	1	（1,1）	
1000017	1	（1,1）	
1000026	1	（1,1）	
1000033	2	（2,1）	
1000043	1	（1,1）	
1000044	2	（2,1）	
1000045	1	（1,1）	
1000046	3	（3,1）	

专利被引次数直方图统计

- Map Class

```
public static class MapClass extends Mapper<Text, Text, LongWritable, LongWritable>
{
    private final static IntWritable uno = 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, uno);
    }
}
```

被引次数

出现1次

专利被引次数直方图统计

- 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
}
```

专利被引次数直方图统计

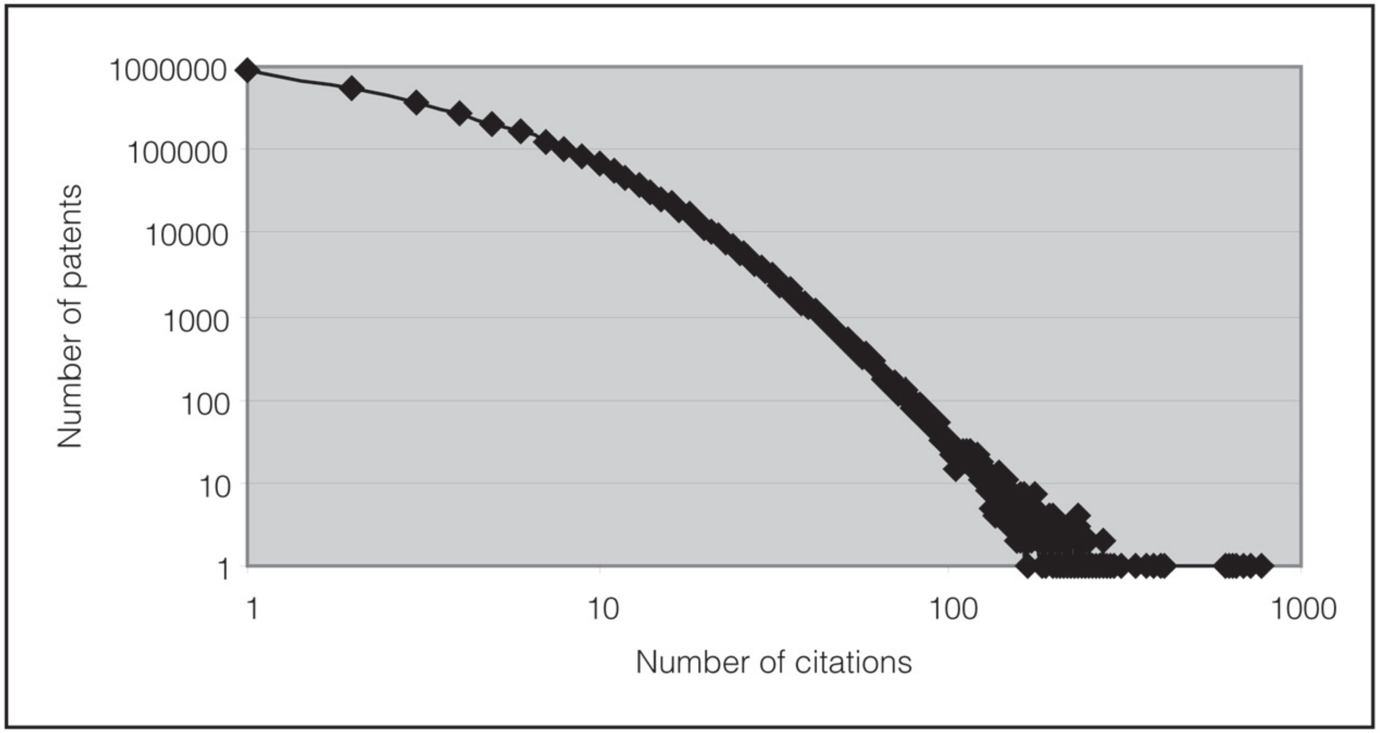
- 主类-- CitationHistogram

```
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);
    }
}
```

直接用Hadoop内置的KeyValue文本输入格式读取以key-value对形式保存的专利被引次数统计输出结果

专利被引次数直方图统计

专利被引次数直方图统计结果



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

年份或国家专利数统计

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：扫描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 uno = 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), uno);
    }
}
```

年份或国家

出现1次

年份/国家专利数统计

- Reduce Class

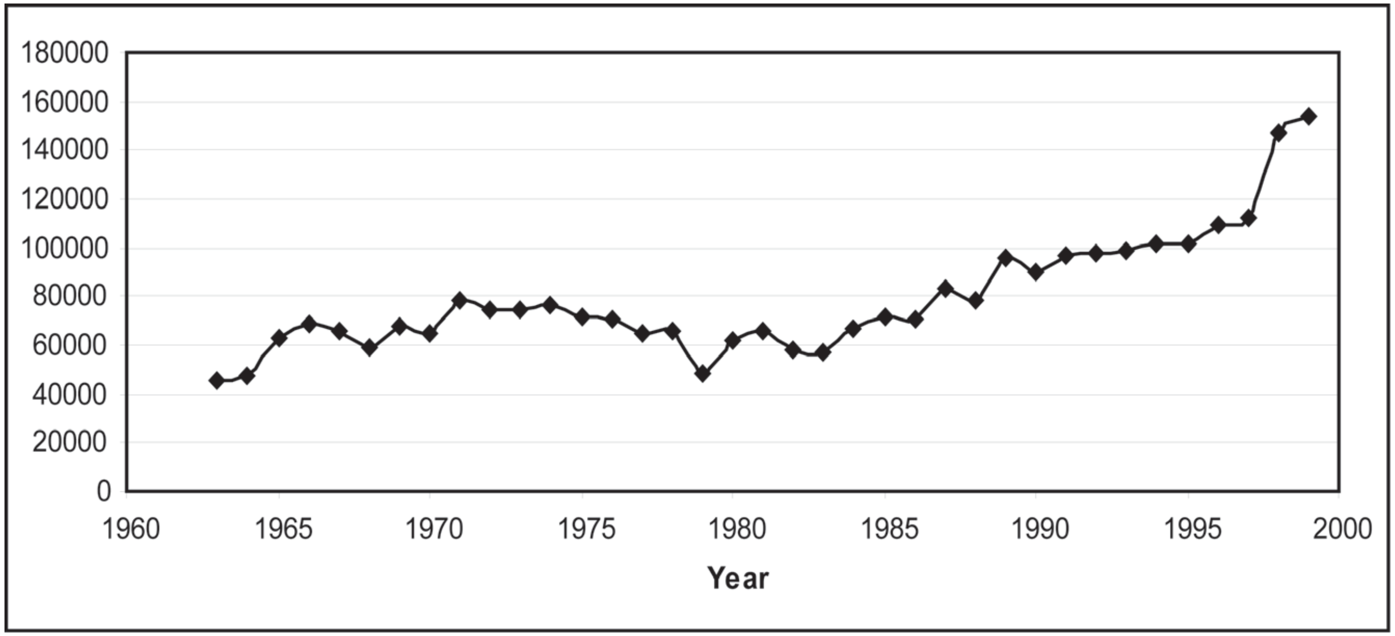
```
public static class ReduceClass extends Reducer
                                         < Text, IntWritable, Text, IntWritable >
{
    public void reduce(Text key, Iterable<IntWritable> values, Context  context)
              throws IOException, InterruptedException
    {         int count = 0;
              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
}
```

年份/国家专利数统计

- 年份专利统计输出

"GYEAR"

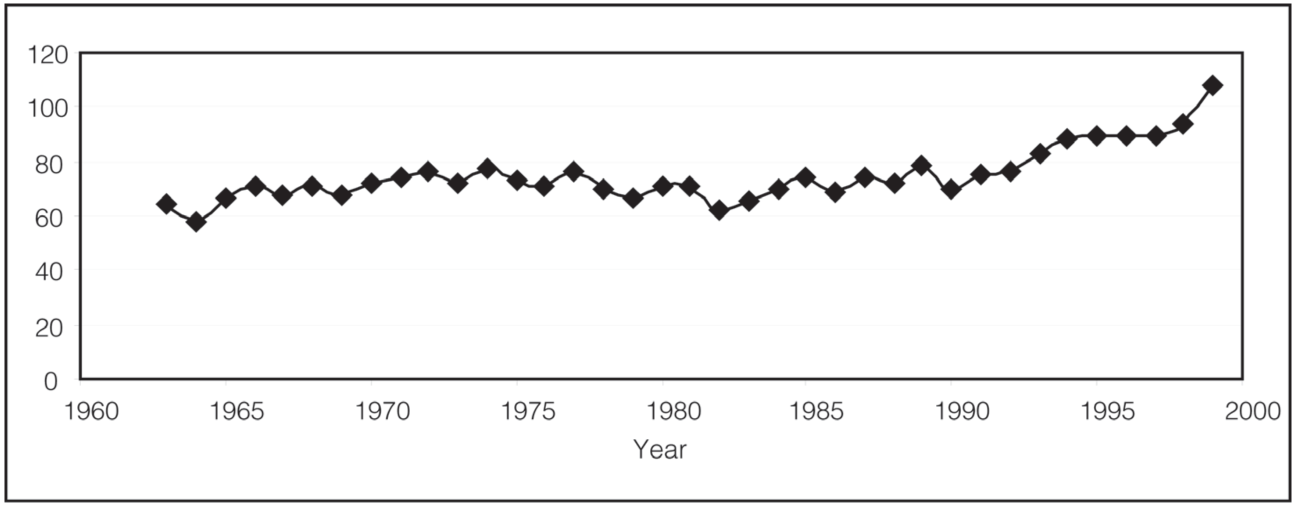
1963	45679
1964	47375
1965	62857
...	
1996	109645
1997	111983
1998	147519
1999	153486



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，并得出如下的统计直方图，该如何实现Map和Reduce？

```
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,,,,,,
```



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, 1,3,…])输入，忽略后部的出现次数，仅考虑key部分：<year, country>，然后在Reduce的输出中统计每个year下的国家数

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？
- ……

7. 实验3：文档倒排索引算法

实验内容与要求

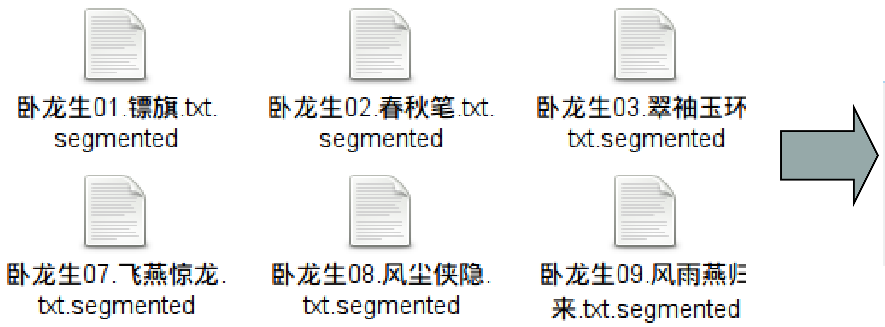
1. 请实现课堂上介绍的“带词频属性的文档倒排算法”。

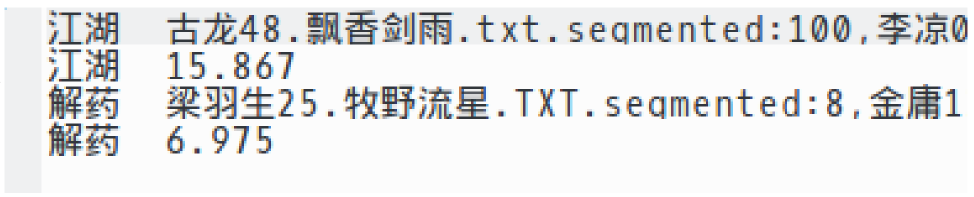
2. 在统计词语的倒排索引时，除了要输出带词频属性的倒排索引，还请计算每个词语的“平均提及次数”并输出。

平均提及次数 = 词语在全部文档中出现的频数总和 / 包含该词语的文档数

3. 两个计算任务请在同一个MapReduce Job中完成，输出时两个内容可以混杂在一起。

4. 输入输出文件的格式和其他具体要求请见FTP上“实验要求”文件夹下对应的PDF文档。


卧龙生01.镖旗.txt.segmented
卧龙生02.春秋笔.txt.segmented
卧龙生03.翠袖玉环.txt.segmented
卧龙生07.飞燕惊龙.txt.segmented
卧龙生08.风尘侠隐.txt.segmented
卧龙生09.风雨燕归来.txt.segmented



```
江湖	古龙48.飘香剑雨.txt.segmented:100,李凉0
江湖	15.867
解药	梁羽生25.牧野流星.TXT.segmented:8,金庸1
解药	6.975
```

7. 实验3：文档倒排索引算法

实验内容与要求

5. 实验结果提交：要求书写一个实验报告，其中包括：

- 实验设计说明，包括主要设计思路、算法设计、程序和各个类设计说明
- 程序运行和实验结果说明和分析
- 性能扩展性等方面可能存在的不足和可能的改进之处
- 源程序 ，执行程序
- 运行结果文件
- 实验报告文件命名规则：MPLab3-组号-组长姓名.doc
- 实验完成时间： 11 月 5 日前完成并提交报告

Thanks！