网络信息检索

第九讲 搜索引擎（2）

董守斌
sbdong@scut.edu.cn

华南理工大学计算机学院
广东省计算机网络重点实验室
Communication & Computer Network Laboratory (CCNL)

主要内容

- 体系结构
- 排序算法
- 元搜索引擎

搜索引擎的性质

- 密集并行（**Embarrassingly parallel**）
  - 无状态（**Stateless**）
  - 大量的只读（**Read-only**）操作

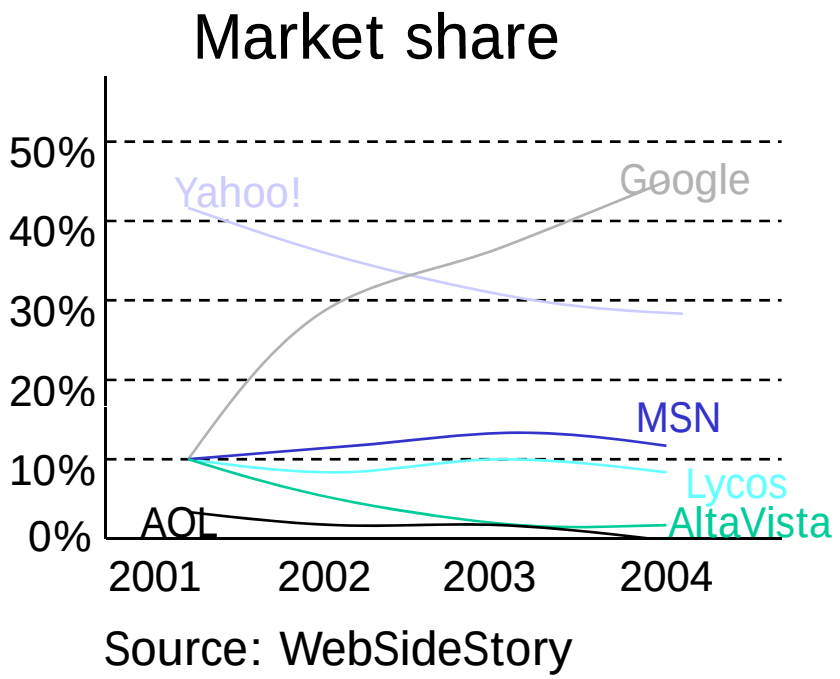
- 需要很大的存储
- 需要很多的计算
- 需要极短的响应时间

软件体系结构

- 系统的设计目标：能耗小、性能比高
  - 便宜的 **PC集群**
- 软件的可靠性
  - 容错（**Fault tolerance**）
- 解决方案：高度的复制（**High degree of replication**）

大规模搜索引擎—Google

- **Google： 10^{100}，“体现了公司整合网上海量信息的远大目标”**
- **Google将数以千计的低成本计算机联网到一起，制造出了一部超高速搜索引擎**
  - **超过80亿索引页面**
  - **超过10亿索引图像**
  - **超过80种语言**
  - **112 个国际域名**



Google体系结构（1998年）

Implemented in C and C++ on Solaris and Linux

索引2400万网页

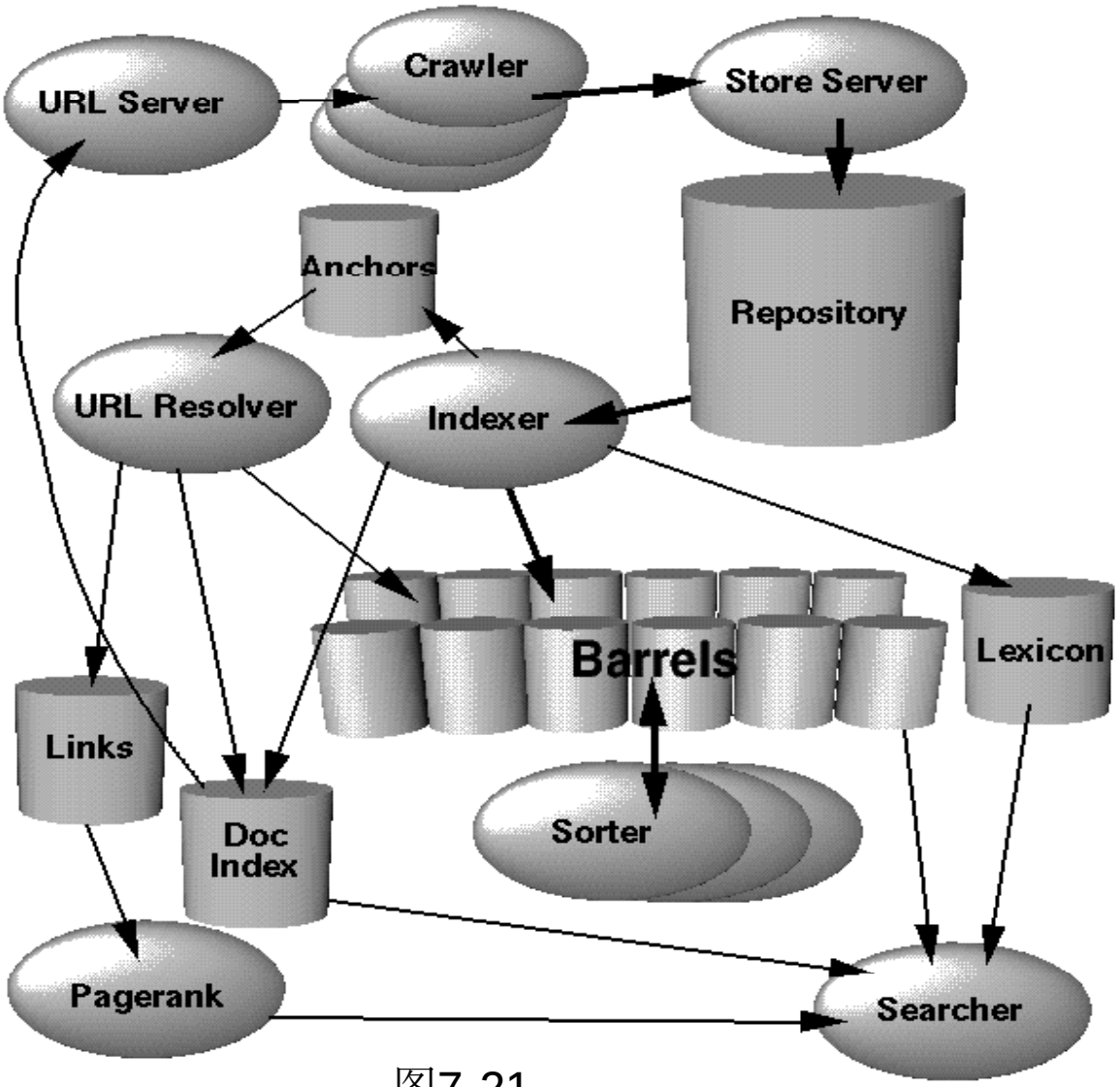


图7-21

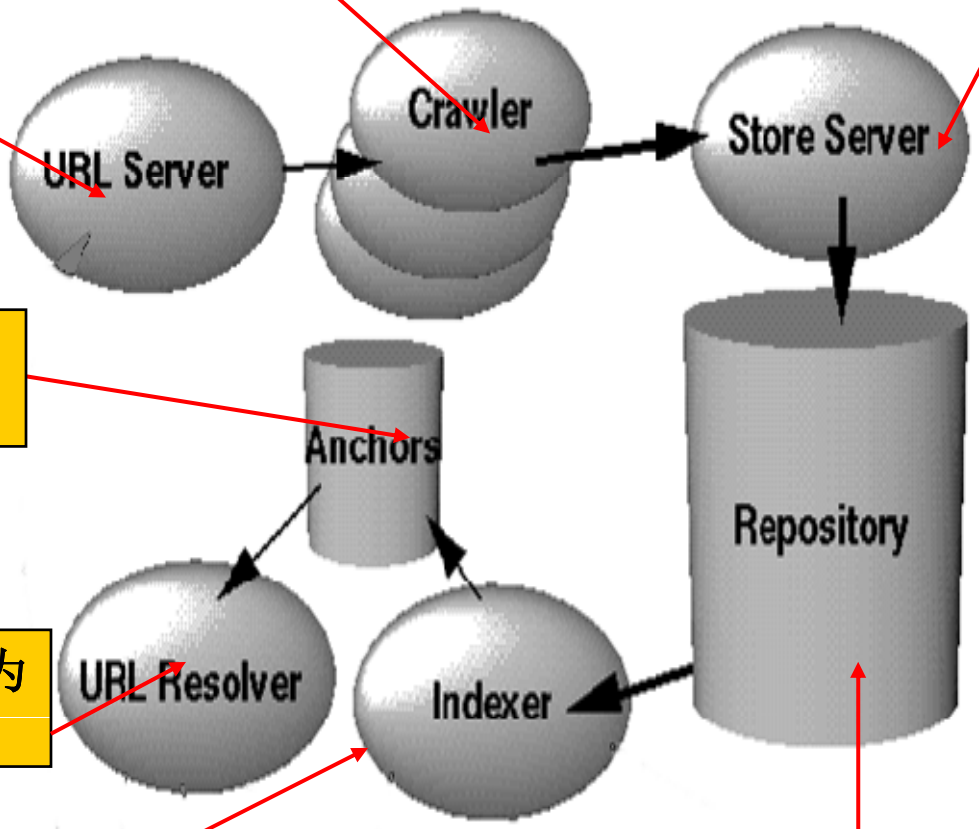
"Anatomy of a Large-Scale Hypertext Web Search Engine", Brin & Page, 1998.

Google体系结构（1）

跟踪需要采集的 URLs

并行运行多个 crawler，一次 大约300个连接

压缩和存储网页



存储每个链接和链接周围的文字

将相对路径的 URLs 转换为绝对路径的URLs.

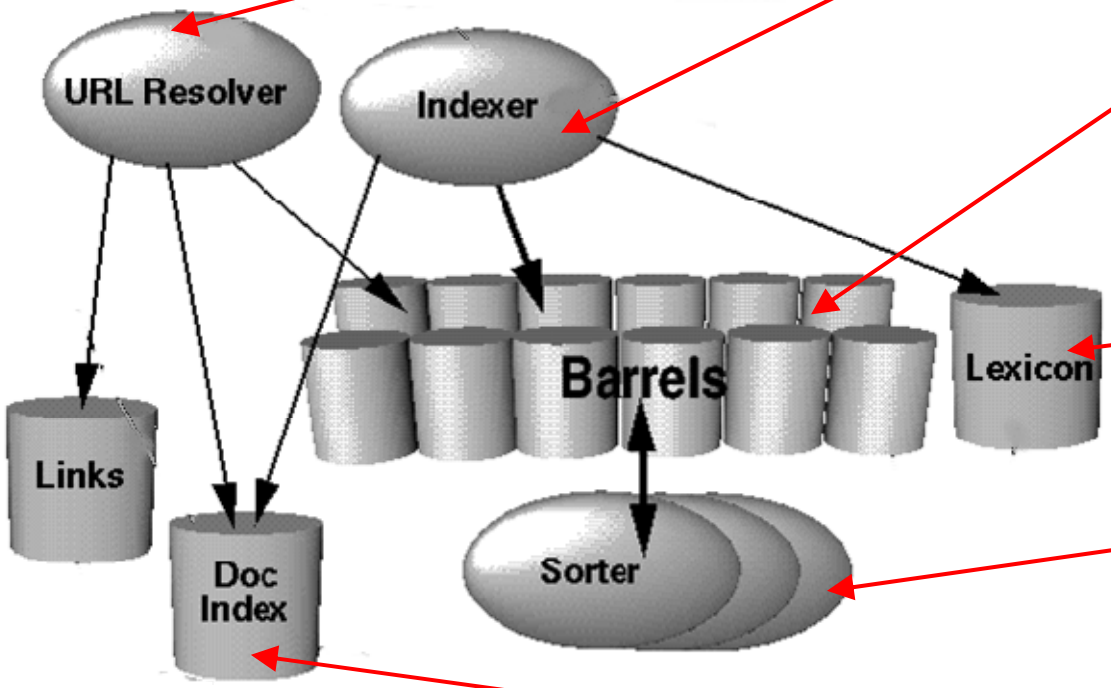
压缩和解析文档。将链接(link)信息存储在锚(anchor)文件中

存储每一个网页的HTML，每个文档前追加文档标识(docID)，文档长度(length)和 URL

Google体系结构（2）

将绝对路径的URLs映射为 docIDs，存储在文档索引 (Doc Index)，将锚文本存储在“barrels”，生成Link数据库 (docIds值对)。

解析生成命中表(hitlist)，命中表记录了关键词、该词在文档中的位置、字体大小和大小写状态等信息，并分发到 “barrels”，，生成以docID排序的前向索引



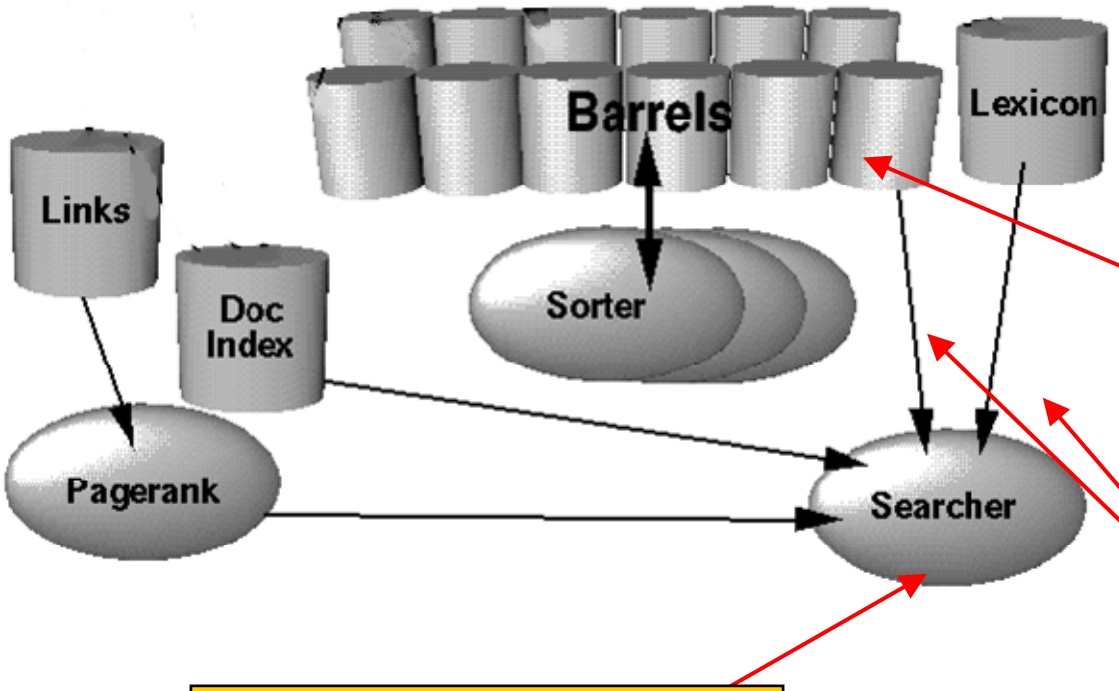
按docID 对前向索引进行部分排序。 每个 barrel存储一定范围内的wordID 的命中列表

驻留内存的hash表将词映射为 wordIds，并包含指向 barrel 的 doclist的指针

创建倒排索引，使得包含docID和hitlist的文档列表可以基于wordID被提取

按DocID组织的索引，每项包含repository 中文档的信息，如指向文档的指针，校验和（checksum），统计信息，状态等，还包含URL的信息。

Google体系结构（3）



两类的**barrels**，短的**barrell**只包含 **title** 或**anchor**的**hitlist**， 长的**barrell**包含所有**hitlist.**

Searcher利用**Sorter** 生成**wordIds**列表和**Indexer**生成**lexicon**创建新的**Lexicon** 。 **Lexicon**存储了**1400**万个字

新的**lexicon**按 **wordID**排序，倒排的文档索引按**docID**排序，和**PageRanks**一起用来回答用户查询

数据结构

- 优化的数据结构，使得海量文档可以较低的开销被抓取、索引和检索
  - 文件系统**bigFiles**
  - 网页库（**Repository**）
  - 文档索引（**Document index**）
  - 词典（**Lexicon**）
  - 命中表（**Hit lists**）
  - 前向索引（**Forward index**）
  - 倒排索引（**Inverted index**）

文件系统BigFiles

- 操作系统提供的文件系统通常不能满足搜索引擎的要求
- **BigFiles**文件系统
  - 在建在多个文件系统之上，以**64**位整数进行寻址的虚拟文件系统
  - 文件系统之间的分配由系统自动完成
  - 一般操作系统不提供足够的描述符号，所以**BigFiles**文件系统要自己处理文件描述符的分配与回收
  - **BigFiles**文件系统还直接支持文件的压缩功能

网页库（Repository）

- 网页库（**Repository**）包含了每个网页的完整**HTML**文档，每个网页都是用**zlib**进行压缩的（压缩要综合考虑存储和速度）
- 文档以**docID**，长度和**URL**作为前缀，一个接一个地存储

Repository: 53.5 GB = 147.8 GB uncompressed

Sync	Length	Compressed packet
Sync	Length	Compressed packet

…

Packet(在repository中压缩存放)

Docid	Ecode	Urllen	Pagelen	Url	Page

文档索引（Document Index）

- 文档索引按照一定的次序来保存关于每个文档的信息，它是按**docID**组织的，每个条目包含
  - 指向**repository**中文档的指针
  - 文档校验和（**checksum**）
  - 统计信息
  - 当前文档统计信息
  - **URL**指针
    - 如果该网页已经被抓取下来了，则它还包含一个指针，指向一个可变宽度、被称为**docinfo**的文件，该文件中包含文档的**URL**及标题
    - 如果该网页未被抓取，指针只是指向一个仅仅包含**URL**的**URLlist**
- 在设计时，要考虑压缩数据结果以减少存储

词典（Lexicon）

- 不同搜索引擎采用的词典不一样，在Google中，词典可以驻留在内存中，占大约256M内存，包含14,000,000个单词（一些稀有的单词没有加入到词典中），由两部分组成：
  - 其一是通过空格分隔的单词表（a list of the words, concatenated together but separated by nulls）
  - 其二是由指针组成的散列表（ a hash table of pointers ）
- 为了提高性能，除了基本词典，每个索引器（indexer）还维护一个额外的文件

命中表（Hit Lists）

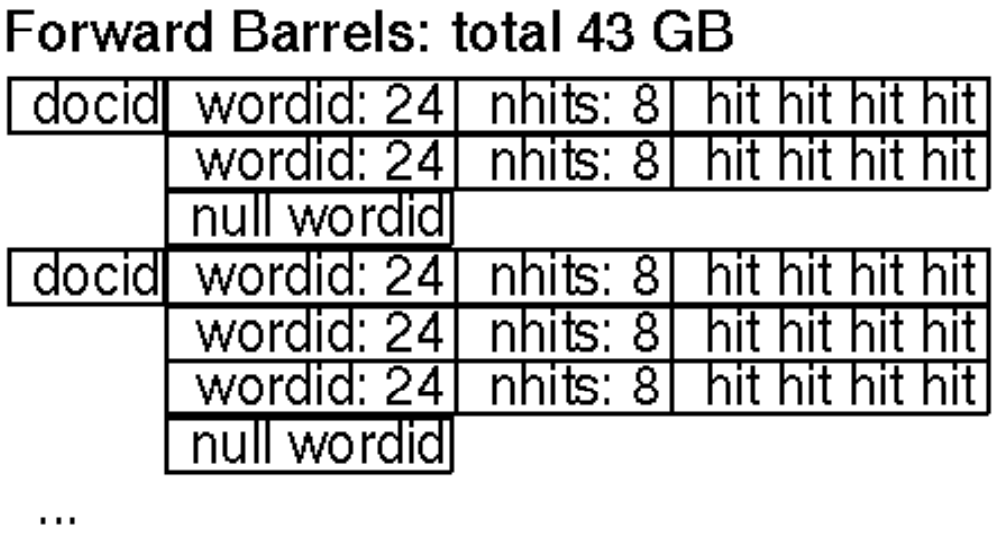
- 命中表的每一项包含某词在某文档中的出现信息：
  - 位置
  - 字体大小
  - 大小写信息等
  - 描述子类型（ **descriptor type** ），如**title**、**anchor**等
- 命中表占据了前向和倒排索引的绝大部分空间，因此，如何有效地表达它们是一个很重要的问题

Hit: 2 bytes

plain:	cap:1	imp:3	position: 12		
fancy:	cap:1	imp = 7	type: 4	position: 8	
anchor:	cap:1	imp = 7	type: 4	hash:4	pos: 4

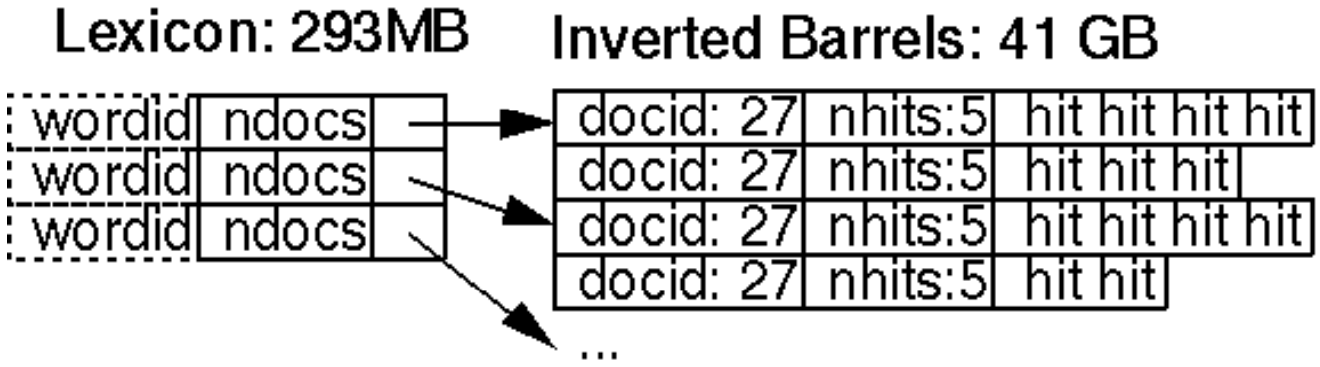
前向索引（Forward Index）

- 前向索引是文档到词的索引，在Google中，前向索引存放在64个存储桶（barrel）中，每个桶容纳一定范围内的wordID，如果一个文档包含的单词（用wordID表示）属于某个桶的话，那么该桶首先记录该文档的docID，紧接其后的是文档中的一串单词、对应于这些单词的命中列表



倒排索引（Inverted Index）

- 提高文档检索的速度，要建立词到文档的索引，即倒排索引。倒排索引也包含与前向索引一样的存储桶
- 对每一个有效的**wordID**，词典包含一个指针指向 包含该**worID**的存储桶
  - 指向由**docID**组成的**doclisys**以及相应的命中表
  - **Doclists**表示所有文档中词的所有出现



Google的主要数据结构总结

Hit: 2 bytes

Plain:	Cap:1	Imp:3	Position:12		
Fancy:	Cap:1	Imp=7	Type:4	Position:8	
Anchor:	Cap:1	Imp=7	Type:4	Hash:4	Position:4

前向Barrels:总计43G

Docid	Wordid:24	nhits:8	hit hit hit hit
	Wordid:24	nhits:8	hit hit hit hit
	Null wordid		

词典： 293MB

Wordid	ndocs	
Wordid	ndocs	
Wordid	ndocs	

倒排Barrels:总计41G

Docid：27	nhits:5	hit hit hit hit
Docid：27	nhits:5	hit hit hit
Docid：27	nhits:5	hit hit hit hit
Docid：27	nhits:5	hit hit hit

…

Google检索算法

1. 解析查询;
2. 把检索词转换为**wordID**;
3. 利用**Lexicon**，在短的存储桶中为每个词寻找包含该词的文档列表的起始位置;
4. 扫描文档列表，直到发现有一个文档包含所有的检索词;
5. 对该查询计算文档的得分;
6. 如果还在短的存储桶处，并已经达到任何文档列表的结尾，对每个词在长的存储桶中寻找文档列表的起始位置，并转到步骤**4**;
7. 如果还没有到达文档列表的终点，转到步骤**4**;
8. 将文档按分数，结合相应网页的**PageRank**情况进行排序，并返回最前面的***k***个。

单个检索词的查询排序

- 对每个词提取命中列表（Hitlist）
- 每个命中可以是以下几种类型之一：题目，锚文本，URL，大字体，小字体等
- 每个命中类型赋予一定的权重
- 类型－权重构成一个权重的矢量（weight vector）
- 每个类型命中个数被计算，并构成频率矢量（count vector）
- 两个矢量的点积（dot product）用来计算IR（Information Retrieval）的得分（内容相关度）
- IR分数与PageRank（网页重要性）结合起来计算最后的得分

多个检索词的查询排序

- 与单个检索词的排序类似，只是必须分析相近性（**proximity**）
- 匹配词越接近，权重越高
- 每个邻近关系被赋予**1**到**10**的权重，从“词组关系”到“毫无关系”
- 对每个类型的命中和相近度计算出现的次数

扩展性

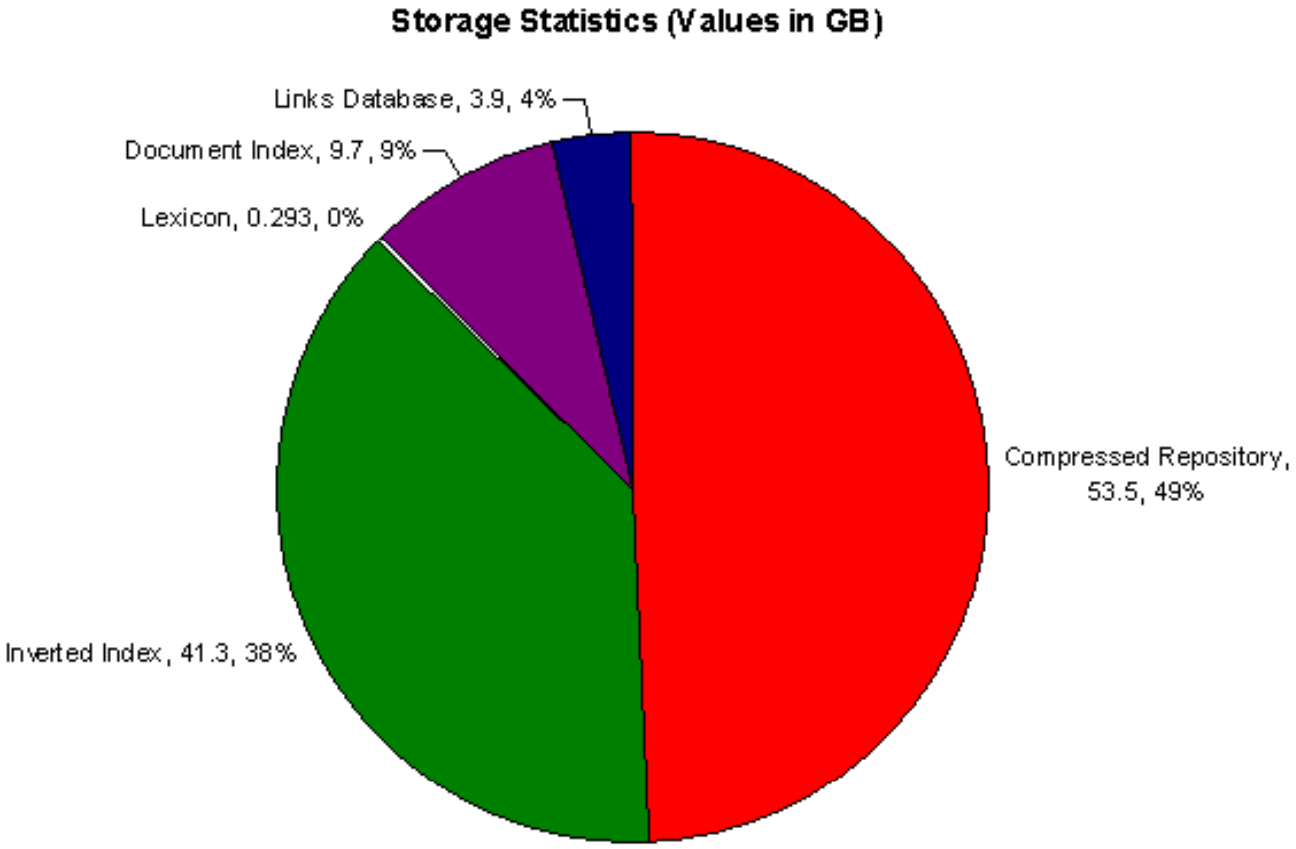
- 采用高可扩展的集群架构，当时（1998年）：
  - 大约2400万文档，在一个星期内索引完成
  - 索引了大约5亿多个超链
  - 4个crawlers每秒抓取100个文档

关键的优化技术

- 每个**crawler**维护一个自己的 **DNS**查找缓存
- 用**flex**来产生 词法分析器（**lexical analyzer**）以解析文档
- 并行化索引
- 词典的内存管理（**In-memory lexicon**）
- 网页库的压缩（**Compression of repository**）
- 对命中表（**hitlists**）的编码压缩可节省很大的存储空间
- 索引器优化的很好，比**crawler**略快，因此**crawler**是瓶颈
- 文档索引批量更新（**updated in bulk**）
- 关键的数据结果放在本地硬盘
- 全局化的架构设计，尽可能避免磁盘扫描

存储要求

- 当时 (1998)， Google对存储的需求如下：



小结：Google早期体系结构

- **Brin和Page的论文“ Anatomy of a Large-Scale Hypertext Web Search Engine”（1998年）**
  - **IR领域最经常被引用的论文之一**
  - **PageRank链接分析算法仍然是目前最好的算法**
  - **Google的成功基于这篇论文所阐述的概念**
- 对中等规模（企业级）的搜索引擎系统仍有借鉴意义

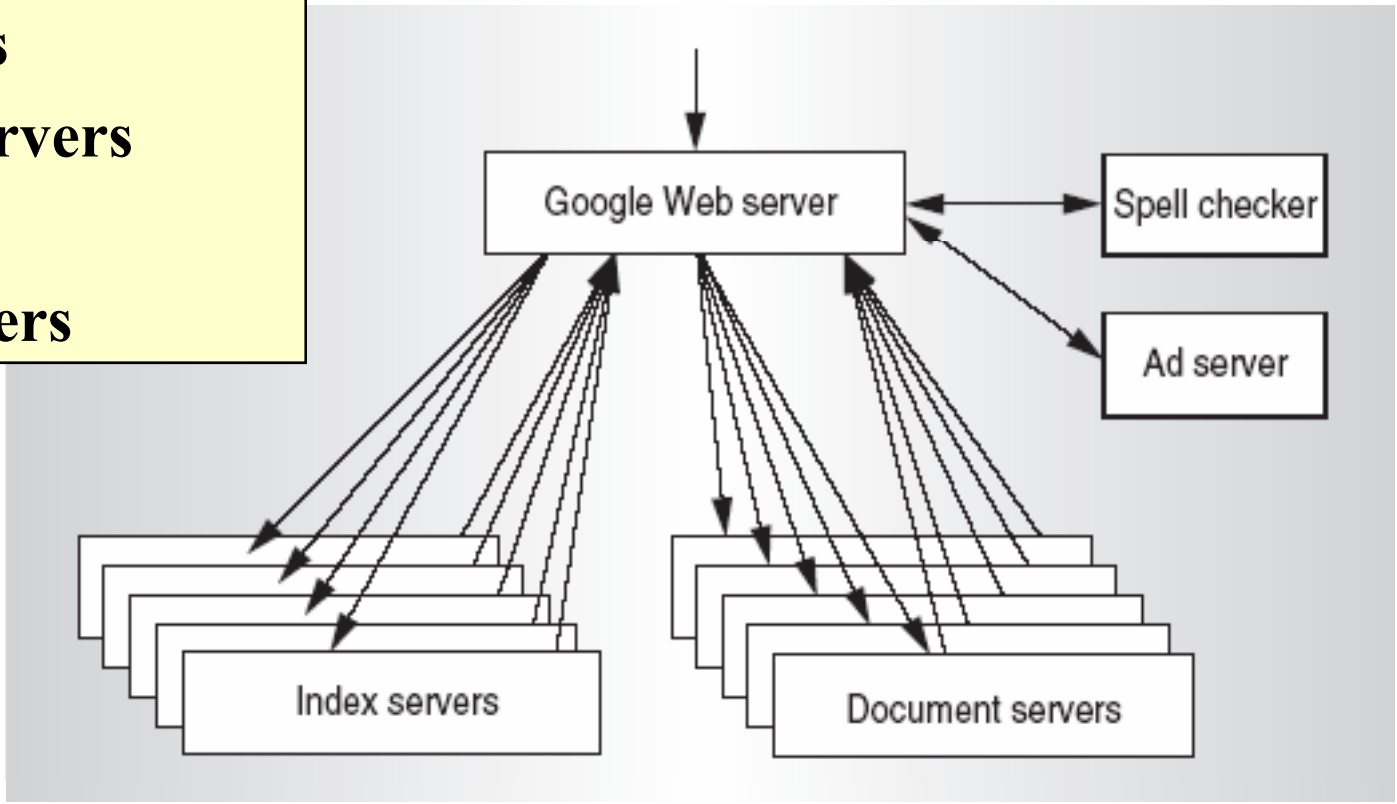
现在Google的计算体系结构

- 大约300万个处理器分布在多个集群，每个集群有2000个处理器
- 商用系统：
  - X86处理器、IDE硬盘， Ethernet通信
  - 通过冗余和软件管理获得可靠性
- 对工作负载的分割
  - 数据（data）：网页、索引分布到处理器上
  - 功能（Function）： 数据抓取、索引生成，索引检索、文档提取，广告放置（Ad placement）等

Barroso, Dean, Hölzle, Web Search for a Planet: The Google Cluster Architecture, IEEE Micro 2003

服务器类型

- **Web Servers**
- **Data-gathering servers**
- **Index servers**
- **Document servers**
- **Ad servers**
- **Spelling servers**



Web服务器

- Web服务器协调用户输入查询的执行，然后将结果组织为HTML页面
- 执行过程包括将查询送给索引服务器（index servers），合并结果，计算结果排序，通过文档服务器（document server）提取每个命中文档的摘要，征询拼写服务器（spelling servers）的意见，从广告服务器（ad server）得到广告列表

数据搜集服务器和索引服务器

- 数据搜集服务器（**Data-gathering servers**）专职于在**Web**上抓取数据，并更新索引和文档数据库，应用**PageRank**算法为每个页面赋予**rank**值
- 每个索引服务器（**Index servers**）包含一系列索引碎片（ **index shards**），他们返回包含查询词的文档的**ID (“docid”)**。索引服务器对硬盘空间要求不高，但需要最多的**CPU**能力

文档服务器

- 文档服务器（**Document servers**）存储文档
- 每个文档存在**12**个文档服务器上
- 当检索时，一个文档服务器对每个文档返回一个基于查询词的摘要
- 如需要，可返回整个文档（网页快照）
- 这些服务器需要很多的硬盘空间

广告服务器和拼写服务器

- 广告服务器（**Ad servers**）管理**AdWords**和**AdSense**提供的广告服务
- 拼写服务器（**Spelling servers make**）对查询的拼写提供建议

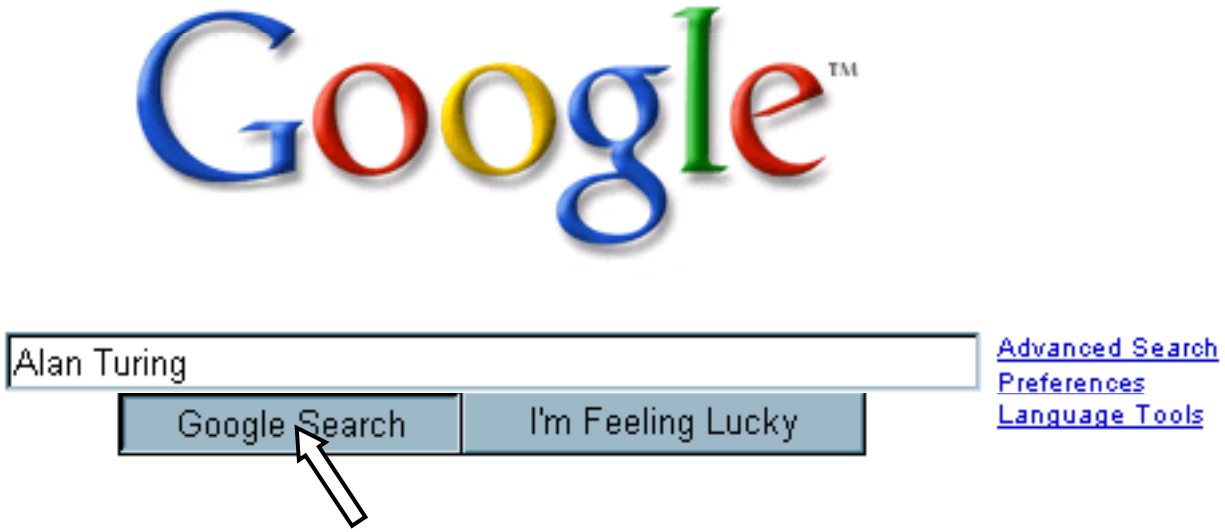
服务器软硬件

- 服务器采用商业的x86 PCs，运行商用的Linux系统
- 目标：购买可在单位能耗中提供最佳性能的CPU，而不是绝对性能
- Google面临的最大开销是能耗（power consumption）

Google的统计数据

- 基于2004年4月发布的Google IPO S-1表，Google拥有：
  - 719个机架（racks）
  - 63,272台机器
  - 126,544个CPUs
  - 253 THz的处理能力
  - 126,544 GB的RAM
  - 5,062 TB 的硬盘空间
- 根据这个估计，Google服务器构成了世界上最强大超级计算机
- 计算能力在126–316 teraflops，相当于 Blue Gene的1/3

一个简单的检索涉及到.....



- **200+ 处理器**
- **200+ TB数据库**
- **10^{10} 总的时钟周期**
- **0.1秒响应时间**
- **5¢ 广告收入**

对体系结构的挑战

- 系统设计
  - 存储和计算的配置
  - 分布式系统的挑战：可扩展性（**Scalability**）、可靠性（**Reliability**）、安全（**Security**）、协商（**Consensus**）
- 编程模型
  - 关于管理的资源的简单的、全局视图

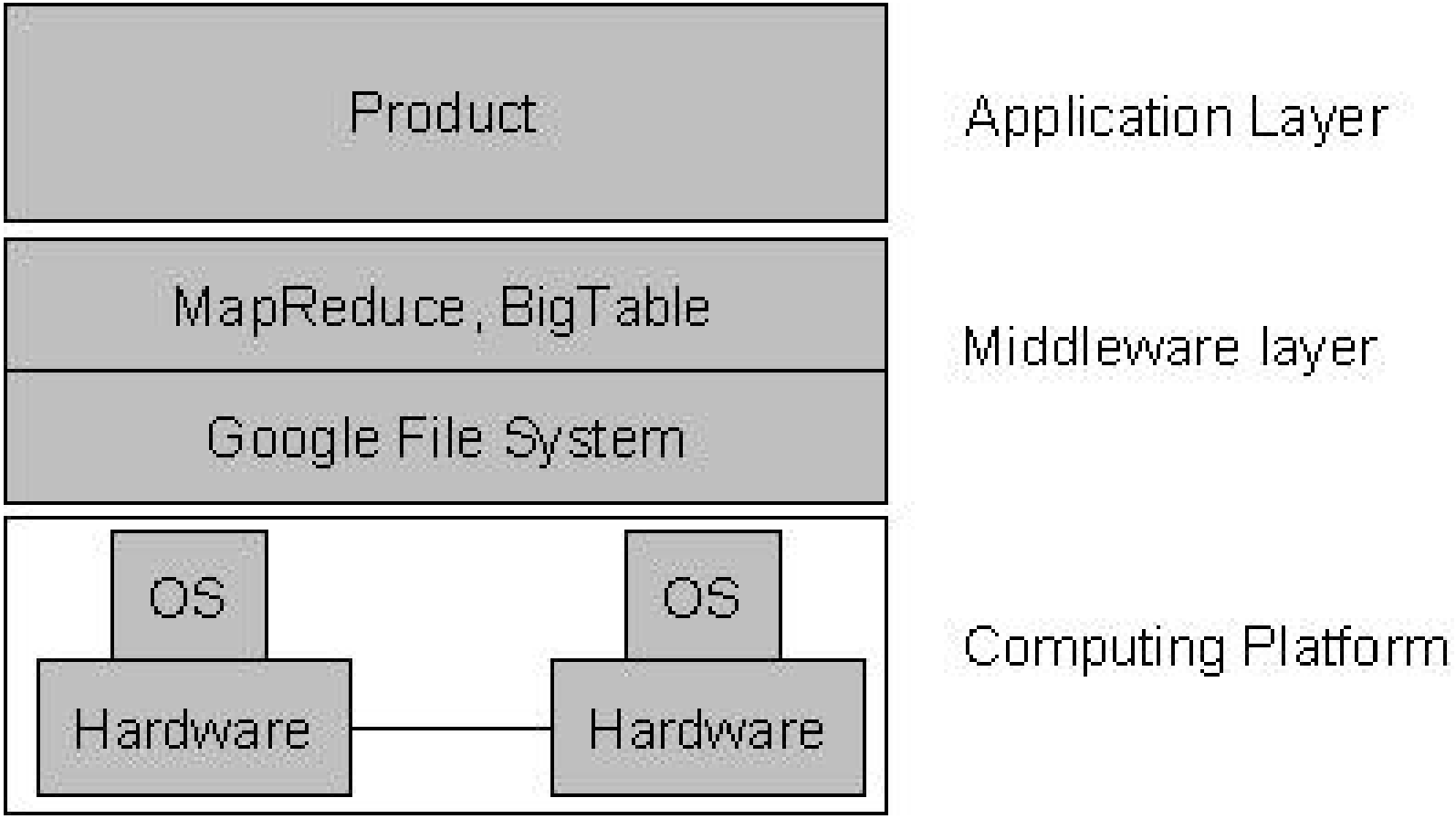
云计算（Cloud Computing）

- 利用互联网连接的数据中心和服务器进行高效计算、信息存取以及编写程序的系统
- 不同于以往的超级计算机
  - 它除了提供大规模分布式计算外，
  - 还以组织和管理数据为核心工作之一，它获取并且维护持续变化的数据集
  - 提供存储以及方便操作数据的编程模式

解决方案

- 分布式文件系统
  - **Google File System（GFS）**
  - **Hadoop Distributed File System（HDFS）**
    - **http://hapdoop.apache.org**
- 编程模型
  - **Functional Language**
    - **MapReduce, HadoopMapReduce**
  - **Column oriented database**
    - **BigTable, HadoopBase**

Google体系结构


Product
MapReduce, BigTable
Google File System
OS
OS
Hardware
Hardware
Application Layer
Middleware layer
Computing Platform

搜索引擎对文件系统的需求

- 文件系统是建立在廉价通用主机平台上的，文件系统必须能够一直监视自己的状态并从灾难中恢复过来
- 文件系统存储着许多大文件。数目大概是几百万个，大小为**100M**或者更大，几**G**
- 需要支持两种类型的读取方式：大规模流读取方式和小规模随机读取方式。在大规模流读取中，一个操作通常会从一个连续的文件区域中读取几百**K**或者更多的数据。随机读取则可能在任意位置读取几**K**的数据
- 写的方式和读的方式也类似。大量的数据通常以追加的方式添加到文件最后，因为这些数据一次写入后通常就不再更改了
- 系统必须能够保证多个客户端能并发地对同一个文件进行追加记录
- 稳定的高带宽比低延时更重要。大多数应用程序，需要在高传输速率下，大块地操作数据。

GFS体系结构

- 集群有一个单个的**master**和好多个**chunkserver.** 为避免单**master**单点失效，**Google**允许一个**GFS**集群中有多个**backup master**存在
- 主节点管理文件系统的所有元数据信息，包括命名空间，访问控制，文件到块的映射表，块的位置等。它同时控制文件系统的全局性操作
- 主节点会定时通过心跳信息与**chunkserver**进行交互，发送指令同时收集块服务器的状态信息

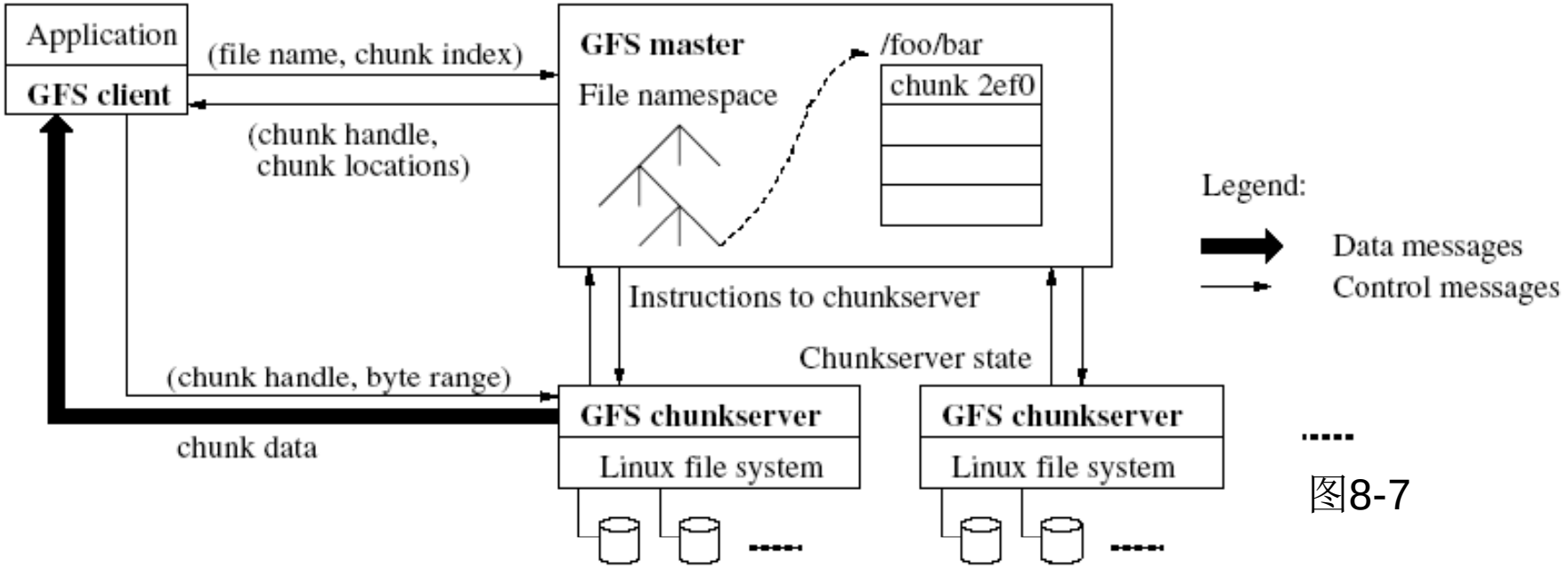


图8-7

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung, The Google File System, 19th ACM Symposium on Operating Systems Principles, 2003.

Chunk及Chunk Server

- 在GFS下，每一个文件都拆成固定大小（64MB）的chunk（块）
  - 每一个块都由master根据块创建的时间产生一个全局唯一的64位chunk handle标志
- Chunkservers在本地磁盘上用Linux文件系统保存这些块，并且根据chunk handle和字节区间，通过Linux文件系统读写这些块的数据
- 出于可靠性的考虑，每个块都会在不同的chunkserver上保存备份（replica）。缺省情况下，保存3个备份

客户读（Client Read）

- 客户端发送请求给master:
  - read(file name, chunk index)
- Master回应：
  - chunk ID, chunk version number, locations of replicas
- 客户选择最近的chunkserver读replica：
  - read(chunk ID, byte range)
  - “最近”是指简单机架内网络拓扑所决定的IP地址
- Chunkserver回应，传送数据

MapReduce：大规模数据处理

- 许多任务都需要处理很多的数据，需要能用到成百上千的CPUs
  - ... 这并不容易
- MapReduce是一个编程模型，提供：
  - 自动的并行化和分布
  - 容错
  - I/O调度
  - 状态和监控
- 运行时系统处理输入数据的的分割、执行调度、处理机器故障，以及管理所需的机器内部通信等细节
  - 这使得没有任何并行和分布式系统经验的程序员能够容易地利用一个大的分布式系统的资源。

Dean & Ghemawat: “MapReduce: Simplified Data Processing on Large Clusters”, OSDI 2004

MapReduce任务处理流程

- 用户指定一个映射函数（Map）处理一个键/值对（key/value pair）来产生中间的键/值对集合，还指定一个归并函数（reduction）来合并所有的与同一中间键相关的中间值
  - **map (in_key, in_value) -> list(out_key, intermediate_value)**
  - **reduce (out_key, list(intermediate_value)) -> list(out_value)**

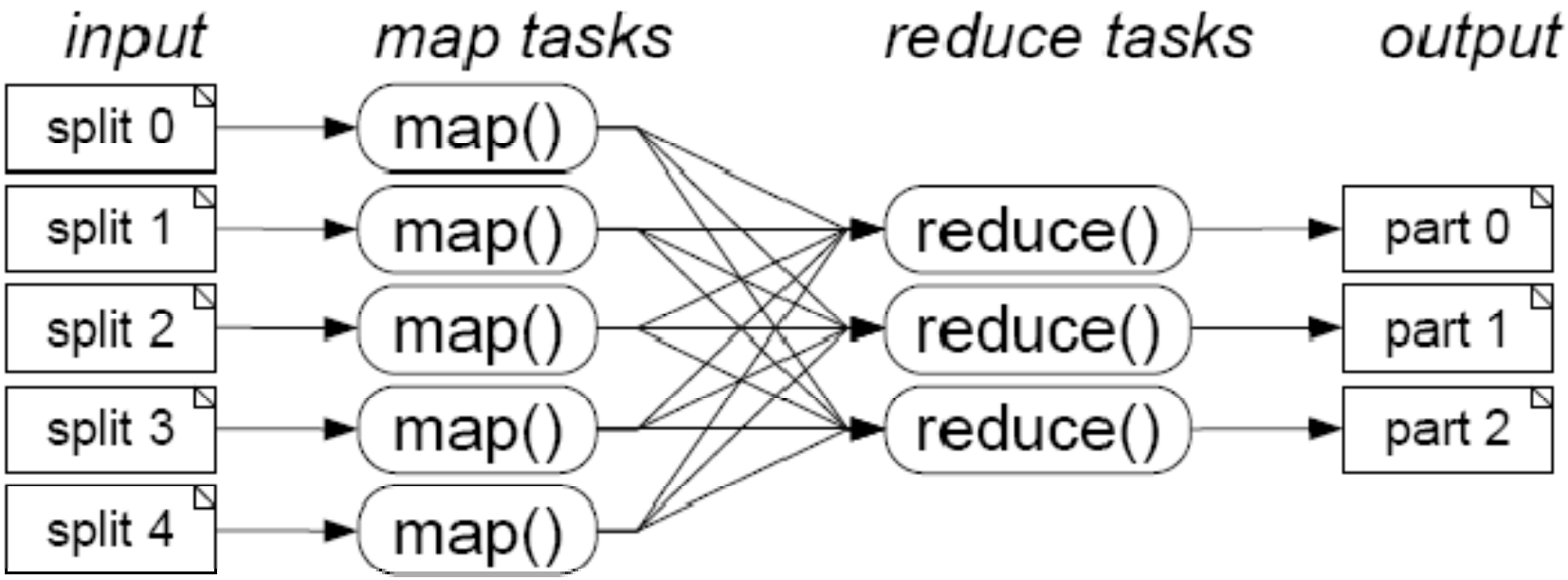


图8-4

MapReduce编程例子

- 统计一个文档集合中各个词出现的次数

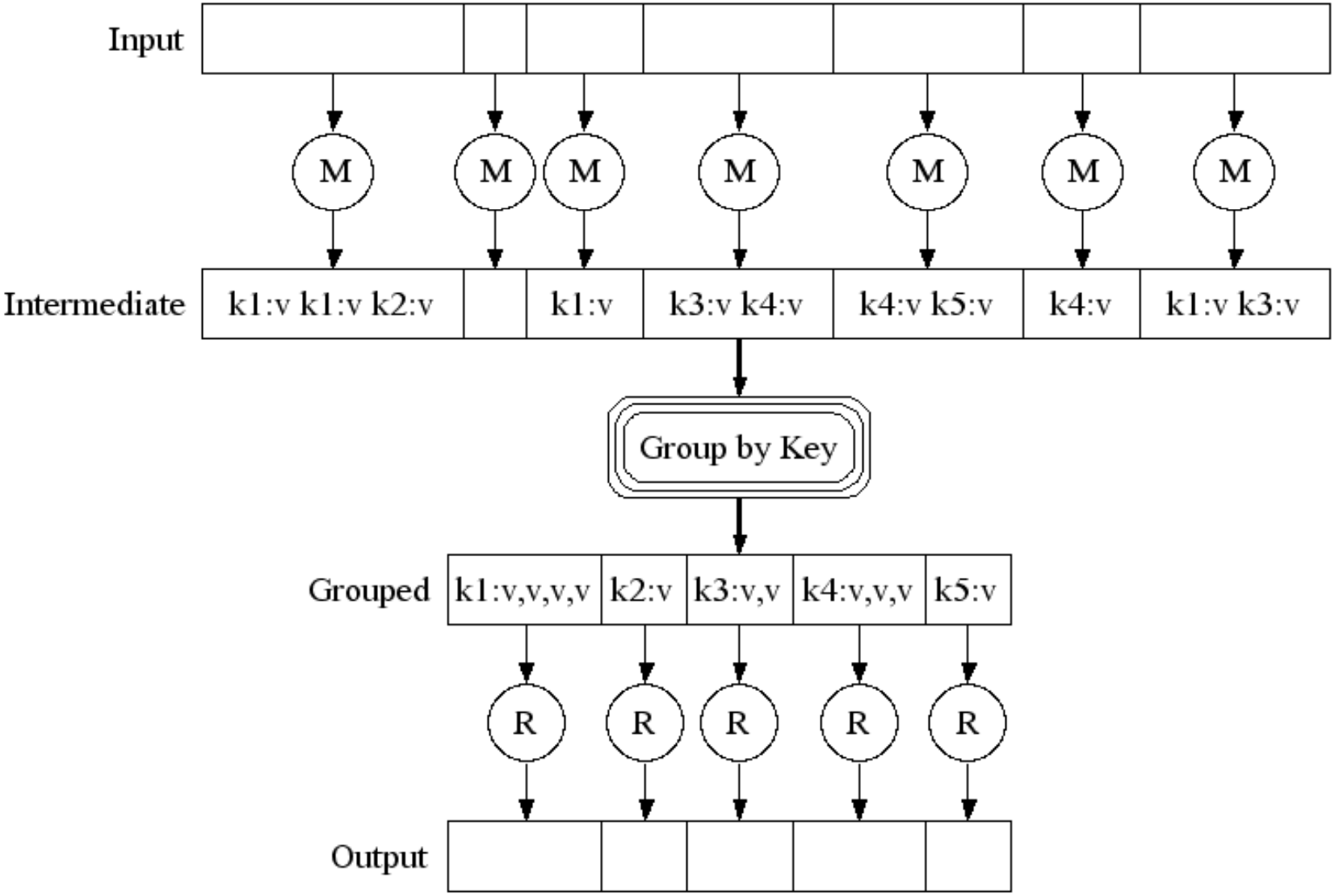
```
map(String key, String value):
  //key: document name
  //value: document contents
  For each word in value:
     EmitIntermedicte(w, “1”)
```

```
reduce(String key, Iterator values):
  //key: a word
  //values: a list of counts
  Int result = 0;
  for each v in values:
       result += ParseInt(v);
  Emit(AsString(result));
```

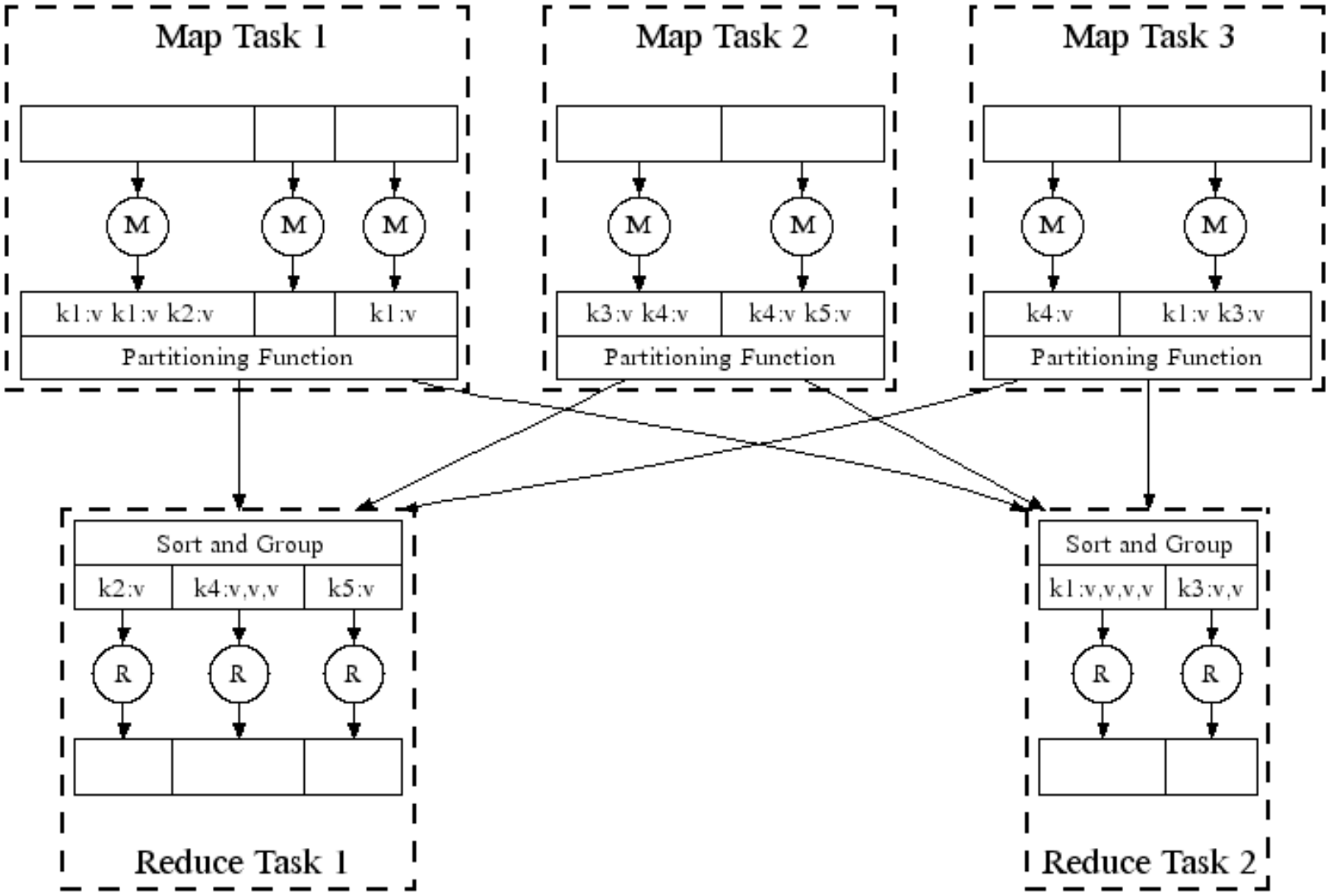
图8-5

http://code.google.com/edu/parallel/mapreduce-tutorial.html

执行


Input
M
M
M
M
M
M
M
Intermediate
k1:v k1:v k2:v
k1:v
k3:v k4:v
k4:v k5:v
k4:v
k1:v k3:v
Group by Key
Grouped
k1:v,v,v,v
k2:v
k3:v,v
k4:v,v,v
k5:v
R
R
R
R
R
Output

并行执行


Map Task 1
M
M
M
k1:v k1:v k2:v
k1:v
Partitioning Function
Map Task 2
M
M
k3:v k4:v
k4:v k5:v
Partitioning Function
Map Task 3
M
M
k4:v
k1:v k3:v
Partitioning Function
Sort and Group
k2:v
k4:v,v,v
k5:v
R
R
R
Reduce Task 1
Sort and Group
k1:v,v,v,v
k3:v,v
R
R
Reduce Task 2

BigTable

- 在BigTable中存储结构化数据
- BigTable 是一个大型的具有容错和自治特性的系统
  - 它包括TB（terabytes）量级的内存和PB的磁盘存储空间
  - 每秒可以处理几百万的读写
- BigTable是一个构建在GFS之上
  - 采用分布式散列机制实现
  - 它不是一个关系数据库，不支持SQL类型查询
  - 提供查找机制，可以通过键值来访问结构化的数据

Fay Chang, et al., “Bigtable: A Distributed Storage System for Structured Data”, OSDI 2006

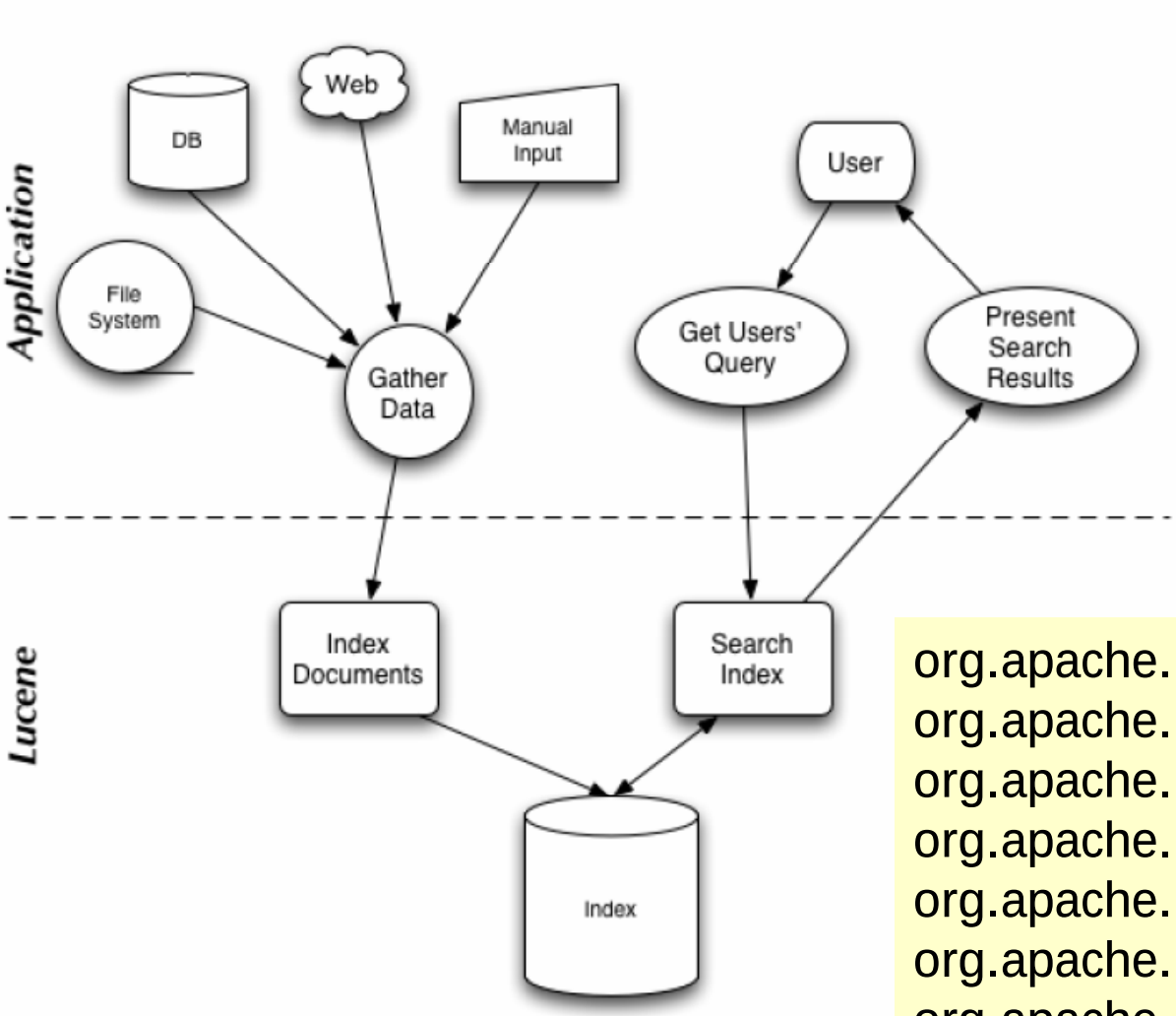
主要内容

- 体系结构
- 排序算法
  - **Lucene**
  - **Nutch**
- 元搜索引擎

Luence简介

- 一个成熟的Apache开源项目；
- 提供文本索引和搜索的Java 类库/包；
- 最新版本2.4.1
- 也有C语言接口
- http://lucene.apache.org

Lucen的典型应用



org.apache.lucene.**index**
org.apache.lucene.**search**
org.apache.lucene.**analysis**
org.apache.lucene.**document**
org.apache.Lucene.**queryParser**
org.apache.Lucene.**store**
org.apache.Lucene.**util**

Lucene得分算法

- 余弦相似度计算

$$Sim(d_i, q) = \frac{\sum_{k=1}^{t}(w_{k,i} \times w_{k,q})}{\sqrt{\sum_{k=1}^{t} w_{k,i}^{2}} \times \sqrt{\sum_{k=1}^{t} w_{k,q}^{2}}}$$

$$w_{k,i} = tf_{k,i} \times idf_k \qquad w_{k,q} = boost_k \times idf_k$$

其中 t 表示查询集中关键词的数量，$tf_{k,i}$ 表示关键词 k 在文档 i 中的权重，idf_k 表示关键词 k 在整个文档集中的逆文档频率，$boost_k$ 表示关键词的权重。

$$idf_k = \ln \frac{\text{总文档数}}{\text{含有关键词k的文档数+1}} + 1$$

$$tf_{k,i} = \sqrt{\text{关键词k在文档i中的数量}}$$

得分公式中的因子

评分因子	描　述
tf(t in d)	文档**d**中出现搜索项**t**的频率
idf(t)	搜索项**t**在倒排文档中出现的频率
fieldBoost(t.field in d)	域的加权因子（**boost**），它的值在索引过程中进行设置
fieldNorm(t.field in d)	域的标准化值（**normalization value**），反比于某一域中索引项个数（**fieldLength**）的平方根，通常在索引时计算该值并将其存储到索引中
Doc.Boost	文档权重， 调用**doc.setBoost()**在索引时进行设置
coord(q, d)	协调因子（**Coordination factor**），该因子的值基于文档中包含查询的项的个数
queryNorm(q)	每个查询的标准化值，指每个查询项权重的平方和

TermQuery的得分公式

- **TermQuery为Lucene支持的最简单的查询方式。查询只有一个关键词*k*。TermQuery的计算公式:**

$$score(d_i, q) = tf_{k,i} * idf_k * boost_k * fieldNorm$$

$$\sqrt{\sum_{k=1}^{t} w_{k,i}^2} = \sqrt{fieldLength} \qquad fieldNorm = \frac{doc.Boost * fieldBoost}{\sqrt{fieldLength}}$$

- ***fieldlength*表示查询所在的域的长度，或者简单理解为某篇文档中查询范围内词汇的数量**
- ***idf*$_k$和*boost*$_k$值与文档无关，不影响排名**
- ***fieldBoost*，*doc.Boost*是人为赋予的经验值，缺省为1.0**
- **因此排名因子： *tf* / sqrt(*fieldlength*)**
  - **单位长度的文档包含的关键词个数的平方根**

BooleanQuery

- **BooleanQuery是一种复合式查询，支持多种不同查询词的逻辑组合**
- **BooleanQuery例子**
  - ＋俄罗斯 恐怖 事件 －美国
  - ＋（俄罗斯 美国) 恐怖 事件
- **可以对不同的查询词赋予不同的boost值表示该查询词在整个BooleanQuery中的重要程度**
  - **例如:　俄罗斯3.0 恐怖2.0 事件1.0**

BooleanQuery的得分公式

- 计算每个查询词**k**和匹配文档的分值

$$score(d_i,k) = weight_k = fieldWeight_k * queryWeight_k$$
$$= (tf_{k,i} \times idf_k \times fieldNorm) * (Boost_k \times idf_k \times queryNorm)$$

- 其中：
$$queryNorm = \frac{1}{\sqrt{\sum_{k=1}^{t} idf_k^2 boost_k^2}}$$

- 对每篇文档计算总得分：
$$score = coord \times \sum_{k=1}^{t} weight_k$$

- 其中：
$$coord = \frac{匹配词总数}{总词项数}$$

TermQuery例子

- 有三篇文档的内容如下

  document1.txt：广州华南理工大学争创国家一流大学

  document2.txt：广州华南理工大学

  document3.txt：广州华南理工大学计算机科学与工程学院

- 对这三篇文档建立索引后，求检索“大学”一词得到的排序

TermQuery例子解答

- 首先对文档进行分词，得到的结果是：
  document1：广州 华南 理工 大学 争创 国家 一流 大学
  document2：广州 华南 理工 大学
  document3：广州 华南 理工 大学 计算机 科学 工程 学院

- 令**fieldBoost=1**，**boost=1**，三篇文档的**fieldNorm**分别近似为**0.3125**，**0.5**，**0.3125**（注意 **fieldNorm**在实现时只用了**1**个字节，故误差较大）
- 得分公式为：

$$socre = tf_{k,i} \times idf_k \times fieldNorm$$

- 三篇文档的**tf**分别是：**1.414**，**1**，**1**，**idf**均是**0.7123**，所以三篇文档的得分分别是**0.3147**，**0.3561**，**0.2226**。

BooleanQuery例子

- 有三篇文档的内容如下：

  Document1：广州华南理工大学争创国家一流大学

  Document2：广州华南理工大学

  Document3：广州华南理工大学计算机科学与工程学院

- 对这三篇文档建立**Lucene**索引后，求检索“大学”、“计算机”两词得到的排序。

BooleanQuery例子解答（1）

$$coord_{document1} = 0.5$$

$$coord_{document2} = 0.5$$

$$coord_{document3} = 1$$

$$idf_{大学} = 0.7123$$

$$idf_{计算机} = 1.4055$$

$$queryNorm = \frac{1}{\sqrt{idf_{大学}^{2} + idf_{计算机}^{2}}} = 0.6346$$

$$tf_{大学,document1} = 1.414$$

$$tf_{大学,document2} = 1$$

$$tf_{大学,document3} = 1$$

$$tf_{计算机,document1} = 0$$

$$tf_{计算机,document2} = 0$$

$$tf_{计算机,document3} = 1$$

BooleanQuery例子解答（2）

- 三篇文档的**fieldNorm**依然分别为**0.3125**，**0.5**，**0.3125**，所以：

$$score_{document1} = 0.0711$$
$$score_{document2} = 0.0805$$
$$score_{document3} = 0.4923$$

- 所以，查询的排序结果应该是： **document3**，**document2**，**document1**

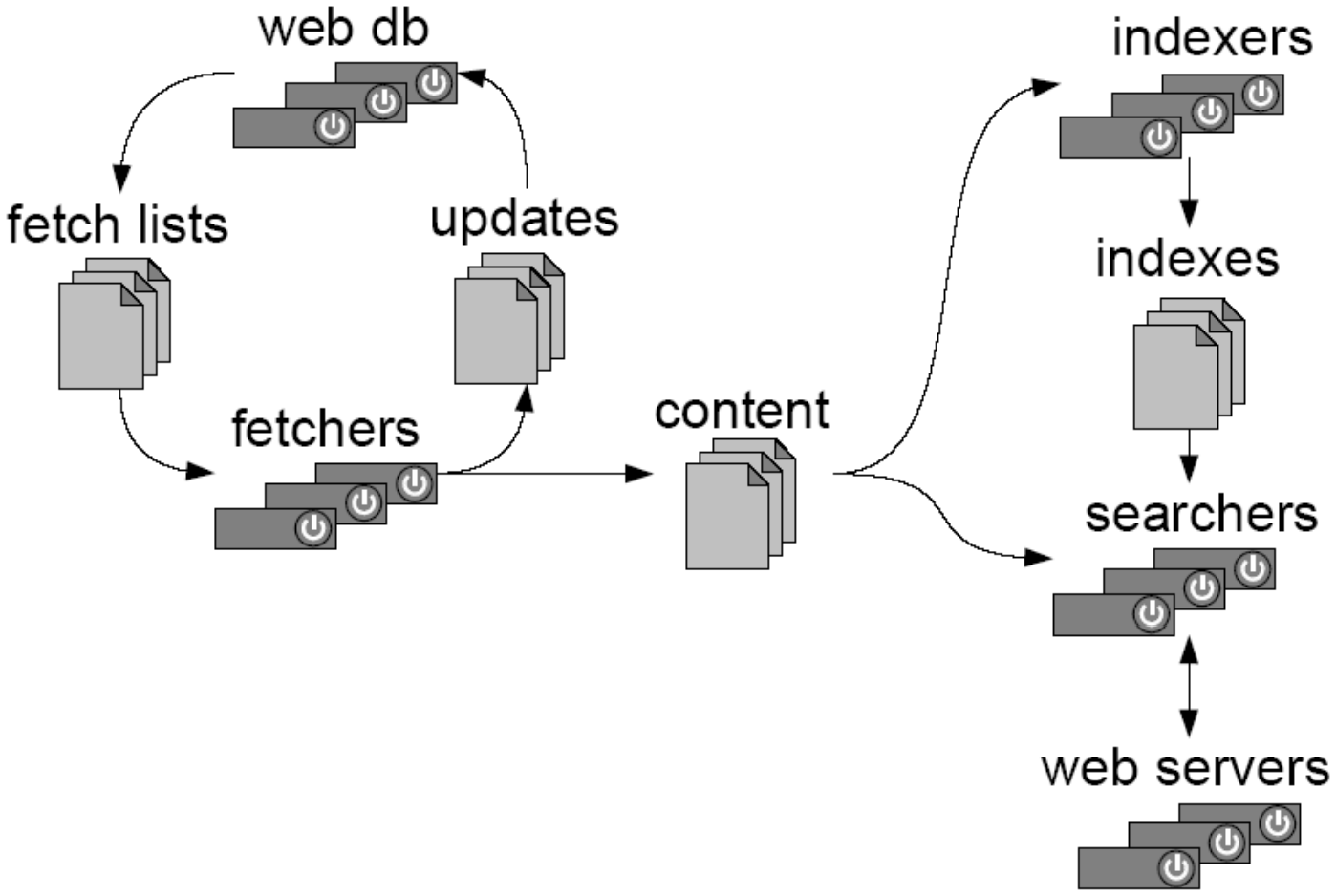
主要内容

- 体系结构
- 排序算法
  - **Lucene**
  - **Nutch**
- 元搜索引擎

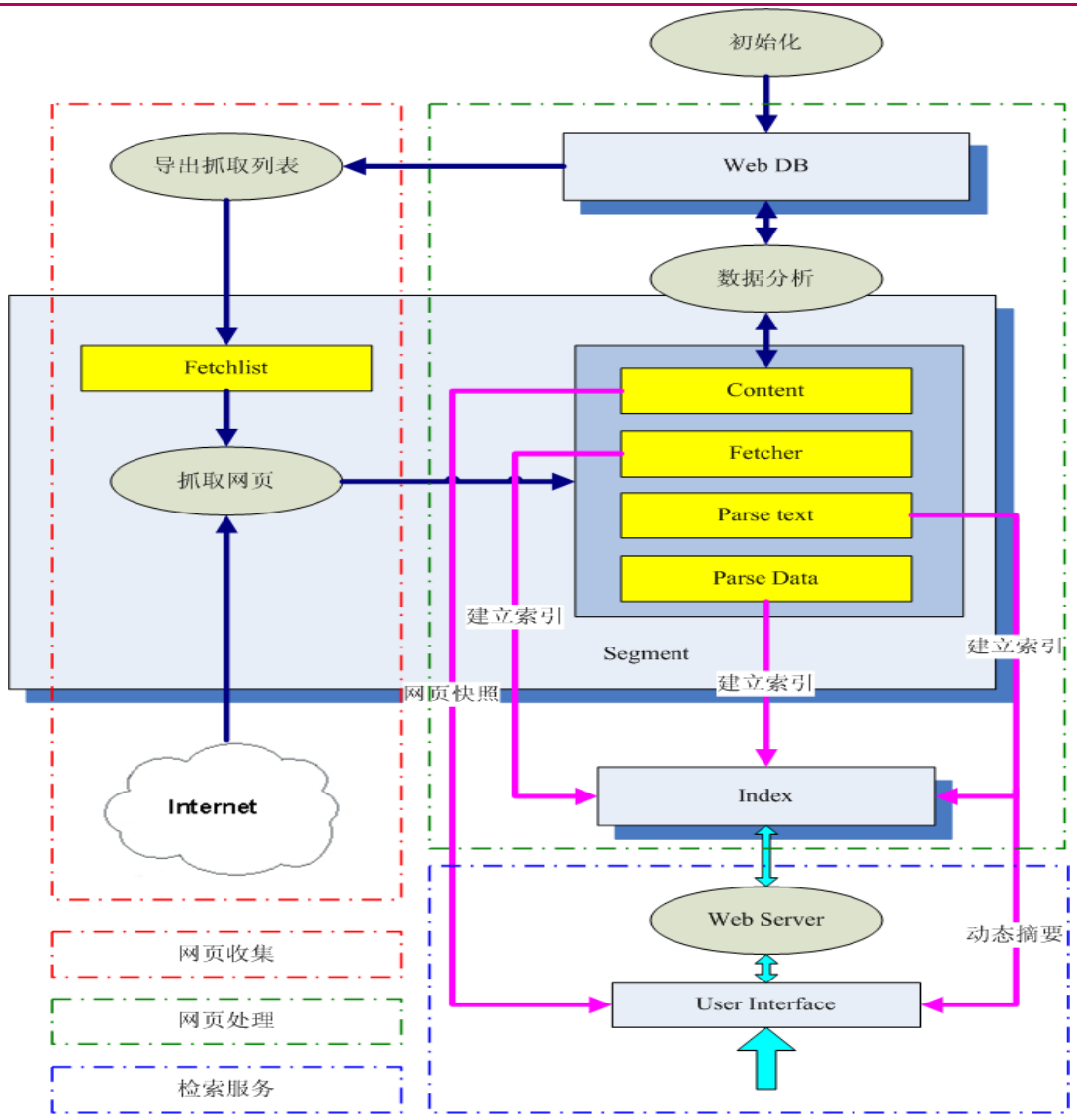
Nutch简介

- Nutch 是基于Lucene的开源搜索引擎
  - http://lucene.apache.org/nutch/
  - 目前最新版本0.9
- 增加了面向web的处理模块
  - crawler
  - 链接图（link graph）
    - 链接分析
    - 锚文本
  - HTML和其他文档格式的识别和解析
  - 语言、字符集的识别和处理
  - 扩展的索引和检索功能

Nutch架构


web db
fetch lists
updates
fetchers
content
indexers
indexes
searchers
web servers

Nutch工作流程


初始化
Web DB
导出抓取列表
数据分析
Fetchlist
Content
Fetcher
抓取网页
Parse text
Parse Data
建立索引
Segment
建立索引
网页快照
建立索引
Internet
Index
Web Server
动态摘要
User Interface
网页收集
网页处理
检索服务

核心模块

数据模块	作用
Web DB	用于保存所有以抓取和未抓取的网页的基本信息和完整的连接信息
Fetchlist	用于保存某次要抓取的网页的URL信息
Fetcher	用于保存抓取后的网页的基本信息和链入该网页的网页信息
Parse data	用于保存该网页所有链出锚点信息
Parse text	用于保存该网页去掉网页标签后的文字信息
Content	用于保存该网页的原始HTML代码
Segment	Nutch的一个工作目录，用于保存Fetchlist、Fetcher、Parse data、Parse text、Content文件
Index	通过Segment建立的索引文件

Web DB的组成

- 网页数据库（Page Database）
  - 用于抓取调度（fetch scheduling）
- 链接数据库（Link Database）
  - 表示全部链接图（link graph）
  - 存储与每个链接关联的锚文本（anchor text）
  - 用于：
    - 链接分析（Link analysis）
    - 锚文本索引（Anchor text indexing）

Nutch排序算法

$$score = score(url) + score(anchor) + score(title) + score(content)$$

$$score(field) = weight(field)$$
$$= fieldWeight(field) \times queryWeight(field)$$

$$field \in \{url, anchor, title, content\}$$

域名称	说明	权重
url	当前网页的url	4
anchor	链向当前网页的链接的锚文本	2
title	当前网页的Title标签中的文本	1.5
content	当前网页去掉HTML标签后的文本	1

Nutch排序算法（2）

$$fieldNorm = \begin{cases} \dfrac{doc.Boost}{fieldLength} & \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots field = url \\ \dfrac{doc.Boost}{\ln(fieldLength + e)} & \ldots\ldots\ldots\ldots\ldots\ldots field = anchor \\ \dfrac{doc.Boost}{\sqrt{fieldLength}} & \ldots\ldots\ldots\ldots\ldots\ldots\ldots field = title \\ \dfrac{doc.Boost}{\sqrt{\max(1000, fieldLength)}} & \ldots\ldots\ldots\ldots field = content \end{cases}$$

- ***doc.boost***是文档的权重，表示文档的重要性，它与网页之间的连接关系有关，与具体的查询词无关

$$doc.boost = 1$$

$$doc.boost = PageRank$$

$$doc.boost = \ln Count(inlink)$$

$$queryNorm = \frac{doc.Boost^2}{\sqrt{\sum(idf^2 \, fieldBoost^2)}}$$

例子文档

sports.html：

```
<html>
<head>
<title>Sports Page</title>
</head>
<body>
THIS IS THE SPORTS PAGE FOR TEST <br>
<a href = "football.html"> FOOTBALL </a> <br>
<a href = "basketball.html"> BASKETBALL </a> <br>
<a href = "golf.html"> GOLF </a> <br>
</body>
</html>
```

basketball.html：

```
<html>
<head>
<title>BASKETBALL Page</title>
</head>
<body>
THIS IS THE BASKETBALL PAGE FOR TEST <br>
<a href = "football.html"> FOOTBALL </a> <br>
<a href = "golf.html"> GOLF </a> <br>
<a href = "sports.html"> SPORTS HOME </a>
</body>
</html>
```

football.html：

```
<html>
<head>
<title>FOOTBALL Page</title>
</head>
<body>
THIS IS THE FOOTBALL PAGE FOR TEST <br>
<a href = "basketball.html"> BASKETBALL </a> <br>
<a href = "golf.html"> GOLF </a> <br>
<a href = "sports.html"> SPORTS HOME </a>
</body>
</html>
```

golf.html：

```
<html>
<head>
<title>GOLF Page</title>
</head>
<body>
THIS IS THE GOLF PAGE FOR TEST    <br>
<a href = "football.html"> FOOTBALL </a> <br>
<a href = "basketball.html"> BASKETBALL </a> <br>
<a href = "sports.html"> SPORTS HOME </a>
</body>
</html>
```

分析网页

- 以football.html为分析对象，对football各项进行分析
  - **url：http http-127 127 0 0 1 8080 football html**
  - **title：football page**
  - **content：football page this is is-the the the-football football page page-for for for-test test basketball golf sports home**
  - **anchor：football**

计算tf值和idf值

域	含有关键词**football**的数量	***tf***
url	1	1
Title	1	1
Content	2	1.4142135
Anchor	1	1

域	总文档数	含有**football**的文档数	***idf***
url	4	1	1.6931472
title	4	1	1.6931472
content	4	4	0.7768564
anchor	4	1	1.6931472

计算queryNorm和fieldNorm

$$queryNorm = \frac{doc.Boost^2}{\sqrt{\sum(idf^2\ fieldBoost^2)}}$$

$$= \frac{1}{\sqrt{1.6931472^2 \times 4^2 + 1.6931472^2 \times 2^2 + 1.6931472^2 \times 1.5^2 + 0.7768564^2 \times 1^2}}$$

$$= 0.124622196$$

fieldNorm

域	**Doc.boost**	含有词汇的数量	理论计算	实际量化
url	1	9	0.11111111	0.109375
title		2	0.70710677	0.625
content		17	0.031622775	0.03125
anchor		1	0.76146287	0.75

分别计算各部分得分（1）

$$score(url) = queryWeight(url) \times fieldWeight(url)$$
$$= fieldBoost(url) \times idf(url) \times queryNorm(url)$$
$$\times tf(url) \times idf(url) \times fidldNorm(url)$$
$$= 4 \times 1.6931472 \times 0.124622196 \times 1 \times 1.6931472 \times 0.109375$$
$$= 0.1563014$$

$$score(title) = queryWeight(title) \times fieldWeight(title)$$
$$= fieldBoost(title) \times idf(title) \times queryNorm(title)$$
$$\times tf(title) \times idf(title) \times fidldNorm(title)$$
$$= 1.5 \times 1.6931472 \times 0.124622196 \times 1 \times 1.6931472 \times 0.625$$
$$= 0.3349316$$

分别计算各部分得分（2）

$$
\begin{aligned}
score(content) &= queryWeight(content) \times fieldWeight(content) \\
&= fieldBoost(content) \times idf(content) \times queryNorm(content) \\
&\times tf(content) \times idf(content) \times fidldNorm(content) \\
&= 1 \times 0.7768564 \times 0.124622196 \times 1.4142135 \times 0.7768564 \times 0.03125 \\
&= 0.003323854
\end{aligned}
$$

$$
\begin{aligned}
score(anchor) &= queryWeight(anchor) \times fieldWeight(anchor) \\
&= fieldBoost(anchor) \times idf(anchor) \times queryNorm(anchor) \\
&\times tf(anchor) \times idf(anchor) \times fidldNorm(anchor) \\
&= 2 \times 1.6931472 \times 0.124622196 \times 1 \times 1.6931472 \times 0.75 \\
&= 0.5358905
\end{aligned}
$$

综合计算结果

$$score = score(url) + score(title) + score(content) + score(anchor)$$

$$= 0.1563014 + 0.3349316 + 0.003323854 + 0.5358905$$

$$= 1.0304474$$

计算结果示例


football
搜索
help
第1-4项 (共有 4 项查询结果):
FOOTBALL Page
FOOTBALL Page THIS IS THE ...
http://127.0.0.1:8080/football.html (网页快照) (评分详解) (anchors)
Sports Page
... PAGE FOR TEST FOOTBALL
http://127.0.0.1:8080/sports.html (网页快照) (评分详解) (anchors)
BASKETBALL Page
... PAGE FOR TEST FOOTBALL
http://127.0.0.1:8080/basketball.html (网页快照) (评分详解) (anchors)
GOLF Page
... PAGE FOR TEST FOOTBALL
http://127.0.0.1:8080/golf.html (网页快照) (评分详解) (anchors)

Nutch得分详解

- **1.0304474 = sum of:**
  - **0.1563014 = weight(url:football^4.0 in 3), product of:**
    - 0.8440149 = queryWeight(url:football^4.0), product of:
      - 4.0 = boost
      - 1.6931472 = idf(docFreq=1)
      - 0.124622196 = queryNorm
    - 0.18518797 = fieldWeight(url:football in 3), product of:
      - 1.0 = tf(termFreq(url:football)=1)
      - 1.6931472 = idf(docFreq=1)
      - 0.109375 = fieldNorm(field=url, doc=3)
  - **0.5358905 = weight(anchor:football^2.0 in 3), product of:**
    - 0.42200744 = queryWeight(anchor:football^2.0), product of:
      - 2.0 = boost
      - 1.6931472 = idf(docFreq=1)
      - 0.124622196 = queryNorm
    - 1.2698604 = fieldWeight(anchor:football in 3), product of:
      - 1.0 = tf(termFreq(anchor:football)=1)
      - 1.6931472 = idf(docFreq=1)
      - 0.75 = fieldNorm(field=anchor, doc=3)
  - **0.003323854 = weight(content:football in 3), product of:**
    - 0.09681355 = queryWeight(content:football), product of:
      - 0.7768564 = idf(docFreq=4)
      - 0.124622196 = queryNorm
    - 0.03433253 = fieldWeight(content:football in 3), product of:
      - 1.4142135 = tf(termFreq(content:football)=2)
      - 0.7768564 = idf(docFreq=4)
      - 0.03125 = fieldNorm(field=content, doc=3)
  - **0.3349316 = weight(title:football^1.5 in 3), product of:**
    - 0.31650558 = queryWeight(title:football^1.5), product of:
      - 1.5 = boost
      - 1.6931472 = idf(docFreq=1)
      - 0.124622196 = queryNorm
    - 1.058217 = fieldWeight(title:football in 3), product of:
      - 1.0 = tf(termFreq(title:football)=1)
      - 1.6931472 = idf(docFreq=1)
      - 0.625 = fieldNorm(field=title, doc=3)

总结：影响排序的因素

- 包括：
  - 页面包含的匹配的检索词的个数
  - 匹配词的邻近程度
  - 词在页面的位置
  - 词在某些tag的位置，如 <title>，<h1>，链接文本（link text），正文文本（body text）
  - 指向该页面的锚文本
  - 在页面中出现的检索词的频率，以及检索词在整个集合中出现的频率（TF*IDF）
  - 链接分析：哪些页面指向该页面
  - 点击分析：该页面被访问的频率
  - 网页的“崭新”程度
  - ……
- 设计复杂的公式将以上各因素综合起来
- 对每个搜索引擎而言，复杂的排序算法都是高度的商业秘密

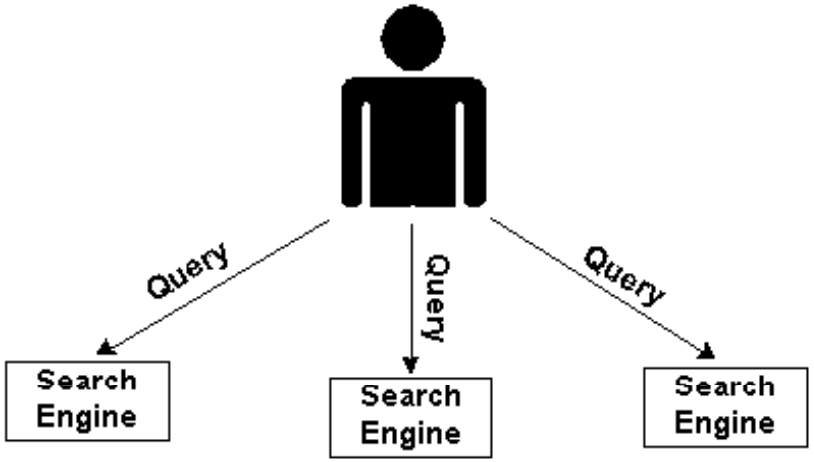
主要内容

- 体系结构
- 排序算法
- 元搜索引擎

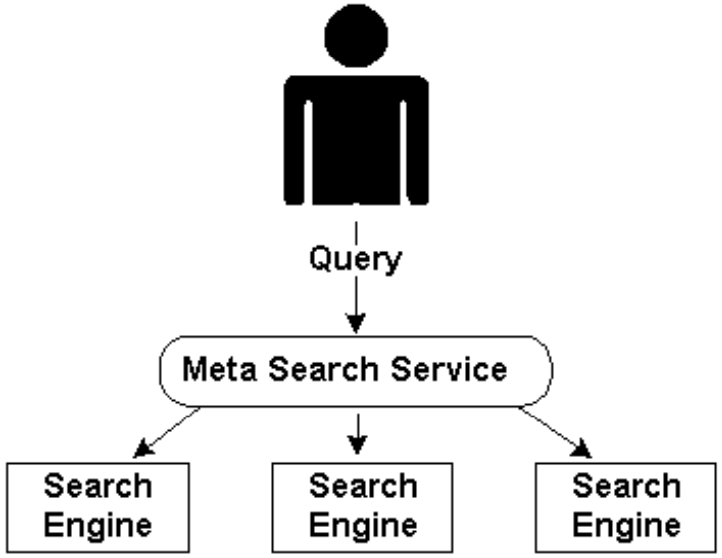
什么是元搜索引擎？

- 元搜索引擎通过将用户查询提交为多个搜索引擎（SE）来增强结果的覆盖面
- 元搜索引提供擎一个统一的接口，并将用户查询同时提交给多个搜索引擎，并将结果以统一的形式返回
- 无需抓取Web网页，创建索引等

元搜索引擎



Search using multiple search engines



Search using a meta search engine

典型的元搜索引擎

- **Dogpile (http://www.dogpile.com/)**
  - 最早的元搜索引擎之一，至今仍然受欢迎
- **Vivisimo (http://www.vivisimo.com/)**
  - 利用聚类技术，自动组织结果，允许用户选择SE
- **Kartoo (http://www.kartoo.com/)**
  - 最好玩的元搜索引擎，用可视的mind map显示结果
- **Mamma (http://www.mamma.com/)**
  - 1996年诞生于一篇硕士论文，可能是第一个元搜索引擎
- **SurfWax (http://www.surfwax.com)**
  - 用 “SiteSnaps”特性，用户可以看到查询词是否在结果文档中

Vivisimo


Vivísimo
Bio MetaCluster
cancer
Search
PubMed @NIH
MerckManual
Google
Harrison
TRIPDatabase
Clustered Results
cancer (571)
Breast cancer (129)
Colorectal cancer (62)
Early Detection (19)
Colorectal Cancer Screening (9)
Predictors, Colorectal adenomas (5)
Advanced colorectal cancer (5)
Chemotherapy For Advanced Colorectal Cancer (2)
Guidance, Irinotecan, Oxaliplatin & Raltitrexed (2)
T lymphocytes isolated from patients with advanced colorectal c
Risk of Colorectal Cancer (5)
Screening for colorectal cancer (3)
Liver metastases (2)
Ursodeoxycholic acid (2)
Colonoscopy, Neoplasia (2)
Nice, Reply (2)
Other Topics (13)
Lung cancer (34)
Gastrointestinal Tract (31)
Prostate cancer (30)
Ovarian cancer (19)
Cervical cancer (19)
Principles Of Cancer Therapy (18)
Cancer Institute (16)
Mortality (16)
More
Category cancer > Colorectal cancer > Advanced colorectal cancer contains 5 documents.
1. Combination chemotherapy for advanced colorectal cancer. [New Window] [Full Window] [Preview]
British Journal of Cancer (2003) 88, 1152-1153. doi:10.1038/sj.bjc.6600848
www.bjcancer.com
Authors: Mason, M - Johnson, P - Rudd, R -
URL: www.ncbi.nlm.nih.gov:80/entrez/query.fcgi?cmd=Retrieve&db=Pu...
Source: PubMedXML 41th
2. Comparison of intermittent and continuous palliative chemotherapy for advanced colorectal cancer: a multicentre randomised trial [New Window] [Full Window] [Preview]
URL: www.tripdatabase.com/redirect.cfm?id=181233&criteria=cancer...
Source: TRIPDatabase 91th
3. Full guidance on irinotecan, oxaliplatin & raltitrexed for Advanced Colorectal Cancer [New Window] [Full Window] [Preview]
URL: www.tripdatabase.com/redirect.cfm?id=181100&criteria=cancer...
Source: TRIPDatabase 7th
4. T lymphocytes isolated from patients with advanced colorectal cancer are suitable for gene immunotherapy approaches. [New Window] [Full Window] [Preview]
Despite improvements in treatment, the 5-year survival for metastatic colorectal cancer remains poor. Novel approaches such as gene immunotherapy are being investigated to improve treatment. Retroviral gene transfer methods have been shown to transduce primary human T lymphocytes effectively resulting in the expression of therapeutic genes. However, a number of defects have been identified in T lymphocytes isolated fror patients bearing tumour, which may have critical implications for the development of gene-targeted T cells as an anticancer therapy. To address this issue, primary T lymphocytes were isolated from patients with advanced colorectal cancer and tested fo

元搜索引擎的优缺点

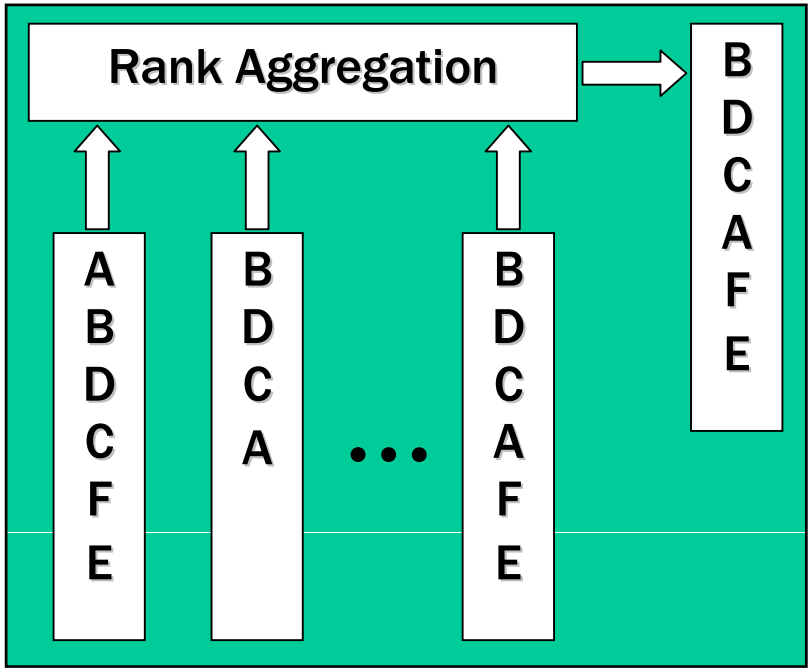
- 优点：
  - 查询可以在多个搜索引擎运行
  - 用户只需了解元搜索工具的界面，无需熟悉每个SE
  - 更好的检索结果：从不同的SE获取排在最前面的文档
- 缺点：
  - 丢失每个SE的独特个性
  - 无法穷尽：只用到每个SE的最前面的一些结果

关键技术

- 关键技术：
  - 重复结果的去除
  - 排序结果
  - 选择要使用的SE（可以让用户选择）
- 如何合并从不同源返回的结果是元搜索引擎要解决的核心问题
  - 排序聚合（Rank Aggregation）

排序聚合

- 排序聚合：将不同SE返回的排序表合理地融合成总的排序列表
- 类似社会选择理论（**social choice theory**）要解决的问题
- 排序聚合方法：
  - 博尔达计数（**Borda count**）
  - **Spearman's Footrule**
  - **Kental's Tau**



社会选择理论

- 社会选择理论：研究帮助一组人进行集合决定（如投票）的协议

- 给定一个代理（**agent**）集合，例如**voters**
  - 对不同的选择（***alternatives***）有不同的倾向**(allocations, outcomes)**
- 如何评估这些选择并作出最后的决定？
  - 有些决定与一些**voter**的愿望类似，有些则完全相反

群决策（Group Decision）

- 如何进行群决策？
  - 扔硬币（Flip a coin）？
  - 独裁（Dictatorship）？
  - 民主（Democracy ）？
- 群决策规则（Group Decision Rules）
  - 多数规则（Majority rule）
    - 孔多塞悖论（Condorcet paradox）
  - 博尔达计数（Borda count）

数学模型

- 一系列投票者（ voters ）， $V=\{V_1,V_2,V_3,\ldots,V_n\}$
- 一系列候选者（alternatives），$S=\{S_1,S_2,S_3,\ldots S_m\}$，$|S|=m$;
- 一系列偏好关系（ preference relation ），$P=\{R_1,R_2,R_3\ldots R_n\}$,，称为偏好概要（preference profile ）
  - 每个投票者i的偏好关系R_i for 是S的一个（顺序）置换

过半数规则（Majority Rule）

- 3 个理智的人对2个候选者{x,y}有合理的选择

Person / Pref.	1	2	3
1st	X	Y	X
2nd	Y	X	Y

- 由于有超过 ½ 的人 选择 x 而不是 y
- 因此这个群选择x而不是y

孔多塞悖论（Condorcet Paradox）

- 如果选择如下：

Person / Pref.	1	2	3
1st	X	Y	Z
2nd	Y	Z	X
3rd	Z	X	Y

成对的比较

- 对于(x,y)

  1: X>Y

  2: Y>X ➔ X>Y

  3 : X>Y

- 类似地，对于 (Y,Z)，可以得到 Y>Z；对于(Z,X)，可以得到 Z>X.
- 则：X>Y>Z>X（循环），不合理
- 18世纪孔多塞（Condorcet）悖论

博达规则（Borda Rule）

- 每个投票者
  - 对最偏好的候选者加上权重1，第二的权重为2，以此类推
- 对每个候选者赋予一个数值，等于
  - 每个投票者赋予的权重值的总和

Borda规则的例子

Person / Pref.	1	2	3
1st	X (1)	Y (1)	X (1)
2nd	Y (2)	X (2)	W (2)
3rd	Z (3)	W (3)	Z (3)
4th	W (4)	Z (4)	Y (4)

- 可以得到:
  - X: 1+2+1=4
  - Y: 2+1+4=7
  - Z: 3+4+3=10
  - W: 4+3+2=9

➔ X>Y>W>Z

用Borda规则解释Condorcet谬论

Person / Pref.	1	2	3
1st	X (1)	Y (1)	Z (1)
2nd	Y (2)	Z (2)	X (2)
3rd	Z (3)	X (3)	Y (3)

- 可以看到用Borda规则，每个候选者得到的总值均为6，因此无法区分胜负

一些变形

- 当有相关分数（**relevant scores**）
  - 输入系统的相关文档有相关度分数

- 加权的**Borda**规则（**Weighted Borda-rule**）
  - 每个投票者对最终结果的影响力不同
  - 可以对质量好的输入系统赋予更多的权重

排序聚合算法

- 最大、最小和平均值模型（**Min, Max and Average Models**）
  - **[Fox and Shaw,1995]**
- 线性融合模型（**Linear Combination Model**）
  - **[Bartell 1995]**

- 搜索引擎的排序聚合

Cynthia Dwork, Ravi Kumar, Moni Naor, D.Sivakumar, Rank Aggregation Methods for the Web, WWW10， 2001

最大、最小和平均值模型

- 文档*d*的最终分数是每个搜索引擎（投票者）的分数的最大值、最小值、平均值或中位值等

融合算法	最终分数
CombMin	所有相关度的最小值
CombMed	所有相关度的中位数
CombMax	所有相关度的最大值
CombSum	所有相关度的总和
CombANZ	**CombSum** / 非零相关度的数目
CombMNZ	**CombSum** * 非零相关度的数目

线性融合模型

$$S_{LC}(d) = \sum_i a_i s_i(d)$$

- 对所有搜索引擎对文档***d***的相关度加权求和，即为文档的相关度
- **LC**模型比其他的模型更灵活。大多数都是由单一参数组成的，例如，得分和或是最大得分，或是基于独立系统性能的固定权重，**LC**模型也可以看作是最简单的神经网络

搜索引擎的排序聚合方法

- 新的挑战：
  - 不同搜索引擎的覆盖（**coverage**）是不同的
  - 一些高度相关的文档可能没有被一些搜索引擎排序
  - 因此，每个搜索引擎可能只排序了部分候选列表（**partial candidate list**）
- 例如：任何两对搜索引擎的前100个结果的平均重叠页面**[Cynthia 2001]**
  - 搜索引擎：**Altavista (AV),Alltheweb (AW), Excite (EX), Google (GG), Hotbot (HB), Lycos (LY), and Northernlight (NL).**

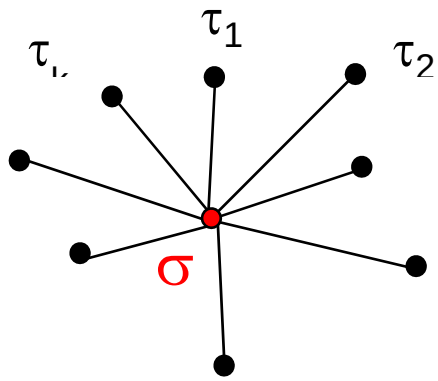
# engines	1	2	3	4	5	6	7
# pages	284.5	84.0	27.2	12.9	8.1	4.7	1.8

预备知识

- 给定一个universe U，一个与U相关的排序列表 τ，是U的子集S的排序，如：$\subseteq\tau = [x_1 \geq x_2 \geq \ldots \geq x_d]$，这里$x_i \in S$，而且 $\geq$是S的一个排序关系
- 如果τ包含：
  - U中的所有元素，则称为全列表（full list）
  - 否则称为部分列表（partial list）

优化的排序聚合

- 定义： $\tau_1, \tau_2, \ldots \tau_k$ 的优化排序聚合（optimal rank aggregation） σ，是一个可最小化 $\sum_i d(\sigma, \tau_i)$的排序



d: 排序间的距离测度
  - **Kendall tau距离**
  - **Spearman footrule距离**

Kendall Tau距离

- **Kendall tau距离（bubble sort distance）：** 与集合***S***相关的两个全列表的距离测度
- 计算两个列表的不匹配对的个数
- 定义为：

$$K(\sigma,\tau)=\left|\{(i,j)\mid i<j,\sigma(i)<\sigma(j),but\quad \tau(i)>\tau(j)\}\right|$$

- 例如： ***K*((a b c d), (a d c b)) = 0 + 2 + 1 = 3**

Kemeny优化聚合

- 问题：
  - 给定（全或部分）列表 $\tau_1, \tau_2, \ldots \tau_k$，要找到一个$\sigma$使得 σ是与 $\tau_1, \tau_2, \ldots \tau_k$的元素并集相关的全列表，而且$\sigma$最小化：

$$K(\sigma, \tau_1, \tau_2, \ldots, \tau_k)$$

  - 由优化Kendall距离得到的聚合称为 Kemeny优化聚合（Kemeny Optimal Aggregation)

Kemeny优化聚合的例子

- 2 排序列表τ_1 =abc, τ_2 =bac
  - $K(\tau_1,\tau_2) = 1$ (τ_1 : a > b, τ_2 : a < b)
  - Kemeny优化聚合：聚合列表 $\tau = \text{argmin} \sum K(\tau, \tau_i)$
  - 最终的聚合列表？

- 当 k>=4，计算Kemeny优化距离是一个NP-hard问题（非多项式问题）
  - 可以用Spearman footrule距离来逼近 Kendall距离

Spearman Footrule距离

- 给定两个全列表σ和τ， Spearman Footrule如下定义：

$$F(\sigma,\tau)=\sum_{i=1}^{|s|}\left|\sigma(i)-\tau(i)\right|$$

- 例如： F((a b c d), (a d c b)) = 0 + 2 + 0 + 2 = 4

- Spearman's footrule距离和Kemeny距离的关系：

$$K(\sigma\ ,\tau)\le F(\sigma,\tau)\le 2K(\sigma,\tau)$$

多个2个列表的距离测度

- 给定几个全列表$\tau_1, \tau_2, \ldots \tau_k$，　τ到σ的规范化的**Footrule距离**如下式：

$$F(\sigma, \tau_1, \tau_2, \ldots, \tau_k) = (1/k)\sum_{i=1}^{k} F(\sigma, \tau_i)$$

- 一个优化的**Spearman's footrule距离**σ使得上式最小化

例子

R_1	
1	A
2	B
3	C
4	D

R_2	
1	B
2	A
3	D
4	C

R_3	
1	B
2	C
3	A
4	D

R	
1	B
2	A
3	C
4	D

A: (1 , 2 , 3)
B: (1 , 1 , 2)
C: (3 , 3 , 4)
D: (3 , 4 , 4)

在TREC上的实验结果

- **TREC的Data Fusion Track**
- **排序聚合的性能由取查询的平均precision来衡量**
- **加权的borda融合Score-based borda-fuse通常是集中borda系列算法中最好的**
- **在大多数数据集（如TREC3, TREC5）上的表现说明融合结果通常好过单一最好的系统**

本讲小结

- 大规模搜索引擎体系结构（7.5）
  - 数据结构设计 （7.5.2）
  - GFS （8.1.4）
  - MapReduce （8.1.3）
- 排序算法
  - Lucene排序算法 （7.4.1）
  - Nutch排序算法（7.4.2）

推荐阅读和网站

- 《网络信息检索》第七章、第八章
- **Introduction to Information retrieval**
  - **Ch19: Web search basics**
  - **Ch21: Link Analysis**

- **Welcome to Nutch，http://lucene.apache.org/nutch/**
- **http://code.google.com/**
- **http://hapdoop.apache.org**

- **S. Brin, L. Page. The anatomy of a large-scale hypertextual web search engine. In:7th International World Wide Web Conference Proceedings, Brisbane, Australia, 1998:107～117**
- **Barroso, Dean, Hölzle, Web Search for a Planet: The Google Cluster Architecture, IEEE Micro 2003**
- **Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung, The Google File System, 19th ACM Symposium on Operating Systems Principles, 2003**
- **Jeffrey Dean and Sanjay Ghemawat, MapReduce: Simplified Data Processing on Large Clusters, OSDI'04, 2004**
- **Cynthia Dwork, Ravi Kumar, Moni Naor, D.Sivakumar, Rank Aggregation Methods for the Web, WWW10, 2001**