

信息检索导论

An Introduction to Information Retrieval

第18讲 信息采集

Crawling

授课人：古晓艳

中国科学院信息工程研究所/国科大网络空间安全学院

*改编自“An introduction to Information retrieval”网上公开的课件，地址 <http://nlp.stanford.edu/IR-book/>

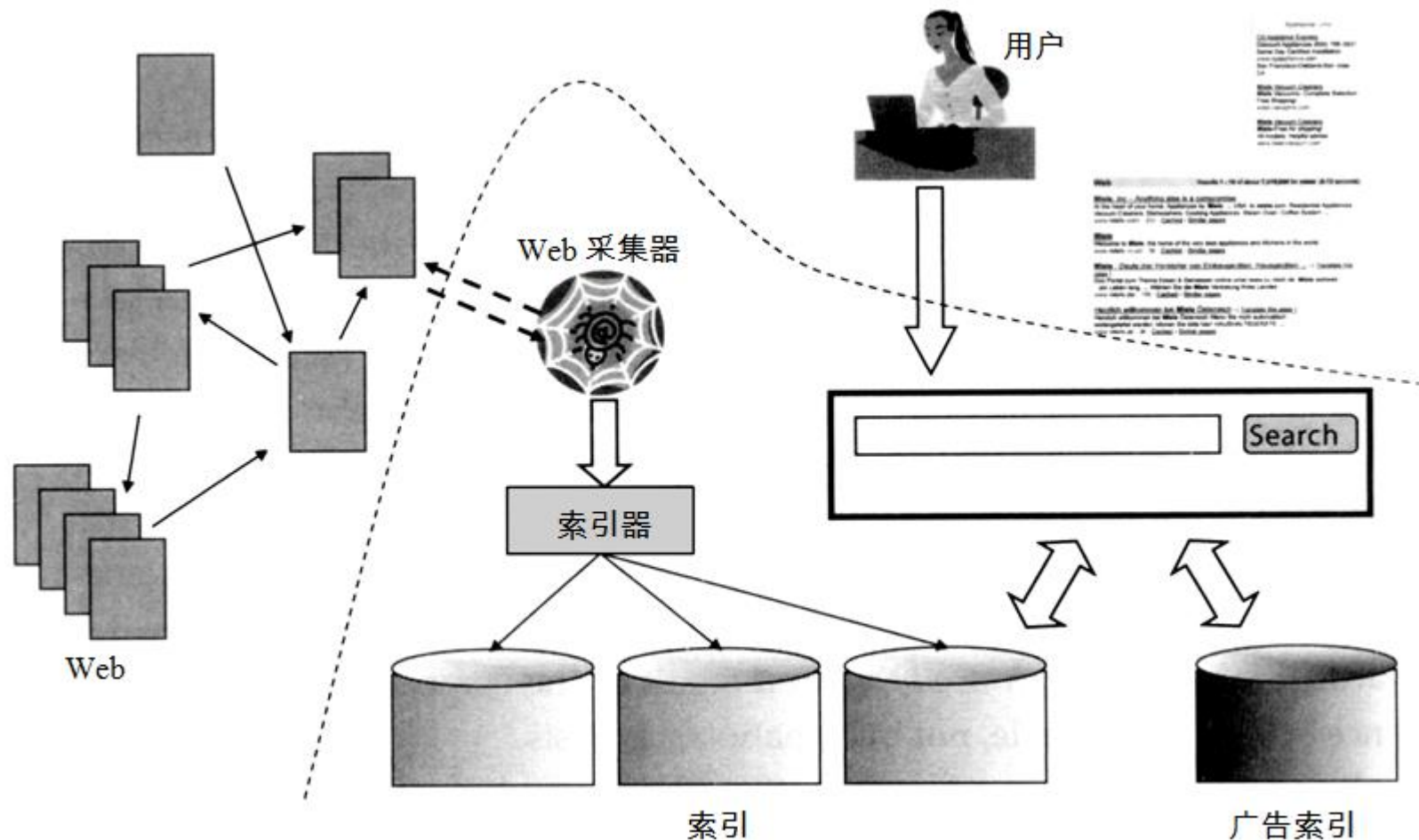
提纲

- ① 上一讲回顾
- ② 一个简单的采集器
- ③ 一个真实的采集器
- ④ 分布式索引

提纲

- ① 上一讲回顾
- ② 一个简单的采集器
- ③ 一个真实的采集器
- ④ 分布式索引

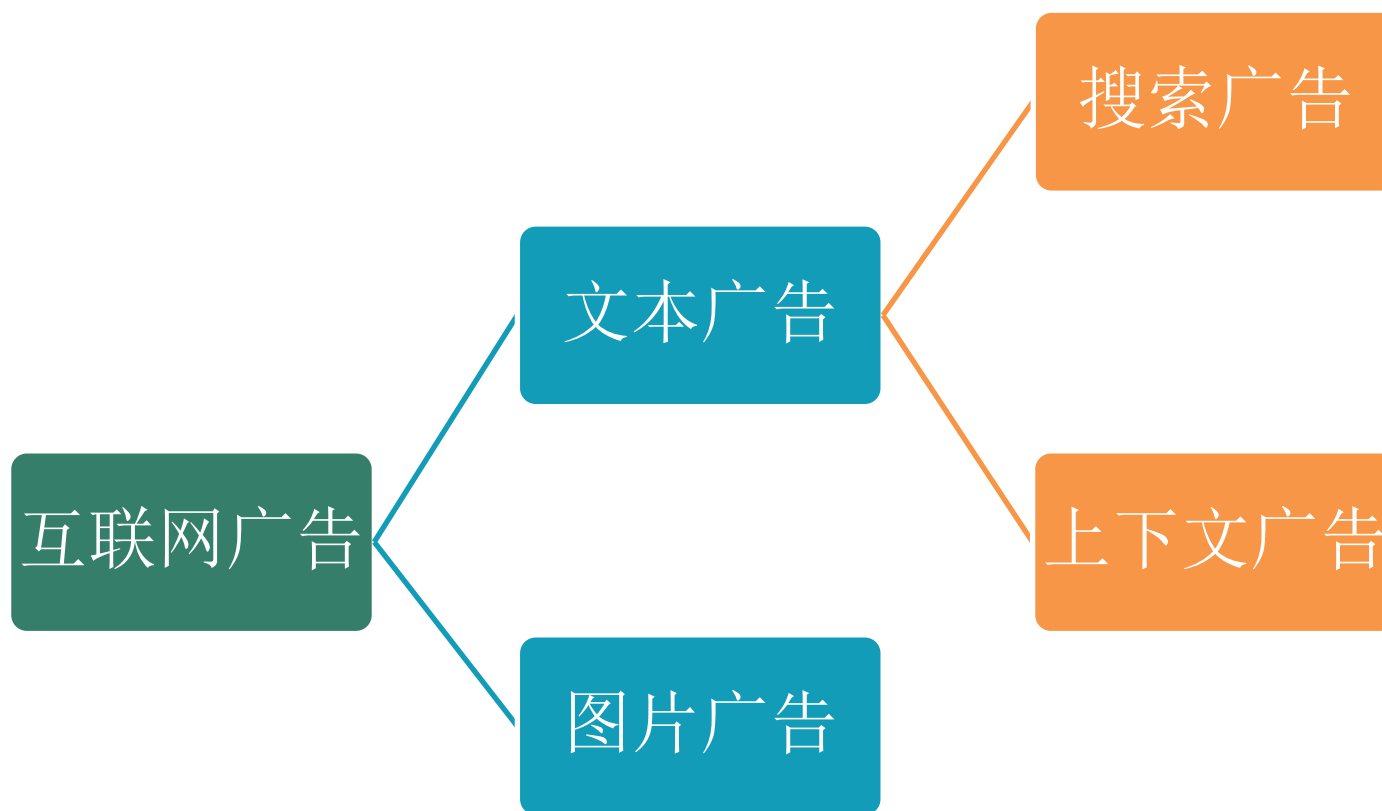
Web搜索系统组成



Web信息检索的特点

- ① 查询请求：幂次分布, 信息类、导航类、事务类
- ② 上下文：地理信息、个人信息等
- ③ 用户：统一接口，用户需求复杂，评价指标多样
- ④ 文档：动态生成、内容变化、多语种
- ⑤ 链接：Web蝴蝶结形结构
- ⑥ 大小：确定Web大小的下限、根据相对大小估算

互联网广告



搜索广告

Web Images Maps News Shopping Gmail more Sign in

Google discount broker Search [Advanced Search](#) [Preferences](#)

Web Results 1 - 10 of about 807,000 for discount broker (definition) (0.12 seconds)

Discount Broker Reviews
Information on online **discount brokers** emphasizing rates, charges, and customer comments and complaints.
www.broker-reviews.us/ - 94k - [Cached](#) - [Similar pages](#)

Discount Broker Rankings (2008 Broker Survey) at SmartMoney.com
Discount Brokers. Rank/ **Brokerage/** Minimum to Open Account, Comments, Standard Commis- sion*, Reduced Commission, Account Fee Per Year (How to Avoid), Avg. ...
www.smartmoney.com/brokers/index.cfm?story=2004-discount-table - 121k - [Cached](#) - [Similar pages](#)

Stock Brokers | Discount Brokers | Online Brokers
Most Recommended. Top 5 **Brokers** headlines. 10. Don't Pay Your **Broker** for Free Funds May 15 at 3:39 PM. 5. Don't **Discount** the Discounters Apr 18 at 2:41 PM ...
www.fool.com/investing/brokers/index.aspx - 44k - [Cached](#) - [Similar pages](#)

Discount Broker
Discount Broker - Definition of **Discount Broker** on Investopedia - A stockbroker who carries out buy and sell orders at a reduced commission compared to a ...
www.investopedia.com/terms/d/discountbroker.asp - 31k - [Cached](#) - [Similar pages](#)

Discount Brokerage and Online Trading for Smart Stock Market ...
Online stock **broker** SogoTrade offers the best in **discount brokerage** investing. Get stock market quotes from this internet stock trading company.
www.sogotrade.com/ - 39k - [Cached](#) - [Similar pages](#)

15 questions to ask discount brokers - MSN Money
Jan 11, 2004 ... If you're not big on hand-holding when it comes to investing, a **discount broker** can be an economical way to go. Just be sure to ask these ...
moneycentral.msn.com/content/Investing/StartInvesting/P66171.asp - 34k - [Cached](#) - [Similar pages](#)

Sponsored Links

Rated #1 Online Broker
No Minimums. No Inactivity Fee
Transfer to Firsttrade for Free!
www.firsttrade.com

Discount Broker
Commission free trades for 30 days.
No maintenance fees. Sign up now.
TDAMERITRADE.com

TradeKing - Online Broker
\$4.95 per Trade, Market or Limit
SmartMoney Top **Discount Broker** 2007
www.TradeKing.com

Scottrade Brokerage
\$7 Trades, No Share Limit. In-Depth
Research. Start Trading Online Now!
www.Scottrade.com

Stock trades \$1.50 - \$3
100 free trades, up to \$100 back
for transfer costs, \$500 minimum
www.sogotrade.com

\$3.95 Online Stock Trades
Market/Limit Orders, No Share Limit
and No Inactivity Fees
www.Marsco.com

INGDIRECT | ShareBuilder
Discount Broker

搜索广告排名：Google次高竞标价格拍卖机制

advertiser	bid	CTR	ad rank	rank	paid
A	\$4.00	0.01	0.04	4	(minimum)
B	\$3.00	0.03	0.09	2	\$2.68
C	\$2.00	0.06	0.12	1	\$1.51
D	\$1.00	0.08	0.08	3	\$0.51

- **bid**: 每个广告商为每次点击给出的最大投标价格
- **CTR**: 点击率，即一旦被显示后被点击的比率。CTR是一种相关性度量指标。
- **ad rank**: $\text{bid} \times \text{CTR}$: 这种做法可以在 (i) 广告商愿意支付的价格 (ii) 广告的相关度高低 之间进行平衡。
- **rank**: 拍卖中的排名
- **paid**: 广告商的次高竞标价格

Google次高竞标价格拍卖机制

advertiser	bid	CTR	ad rank	rank	paid
A	\$4.00	0.01	0.04	4	(minimum)
B	\$3.00	0.03	0.09	2	\$2.68
C	\$2.00	0.06	0.12	1	\$1.51
D	\$1.00	0.08	0.08	3	\$0.51

- 次高竞标价格拍卖：广告商支付其维持在拍卖中排名所必须的价钱(加上一分钱)(用它的下一名计算其支付价格)

$$\text{price}_1 \times \text{CTR}_1 = \text{bid}_2 \times \text{CTR}_2 \text{ (使得排名 } \text{rank}_1 = \text{rank}_2 \text{)}$$

$$\text{price}_1 = \text{bid}_2 \times \text{CTR}_2 / \text{CTR}_1$$

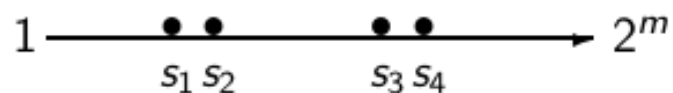
$$p_1 = \text{bid}_2 \times \text{CTR}_2 / \text{CTR}_1 = 3.00 \times 0.03 / 0.06 = 1.50$$

$$p_2 = \text{bid}_3 \times \text{CTR}_3 / \text{CTR}_2 = 1.00 \times 0.08 / 0.03 = 2.67$$

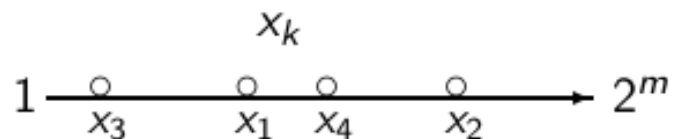
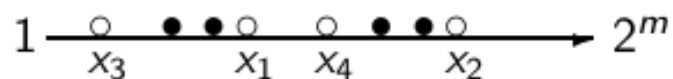
$$p_3 = \text{bid}_4 \times \text{CTR}_4 / \text{CTR}_3 = 4.00 \times 0.01 / 0.08 = 0.50$$

重复检测：置换和最小值：例子

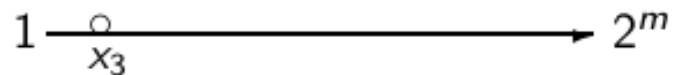
文档 1: $\{s_k\}$



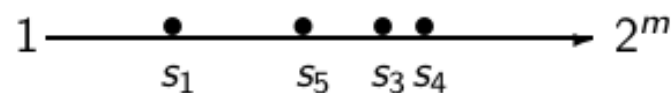
$$x_k = \pi(s_k)$$



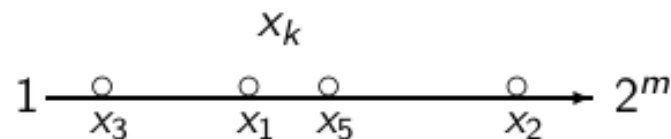
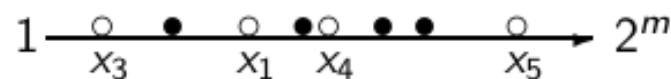
$$\min_{s_k} \pi(s_k)$$



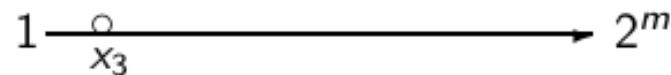
文档2: $\{s_k\}$



$$x_k = \pi(s_k)$$



$$\min_{s_k} \pi(s_k)$$



使用 $\min_{s \in d_1} \pi(s) = \min_{s \in d_2} \pi(s)$ 作为文档 d_1 和 d_2 是否近似重复的测试条件？该例子中置换 π 表明: $d_1 \approx d_2$

例子

	d_1	d_2
s_1	1	0
s_2	0	1
s_3	1	1
s_4	1	0
s_5	0	1

$$h(x) = x \bmod 5$$

$$g(x) = (2x + 1) \bmod 5$$

$$\min(h(d_1)) = 1 \neq 0 =$$

$$\min(h(d_2)) \quad \min(g(d_1)) =$$

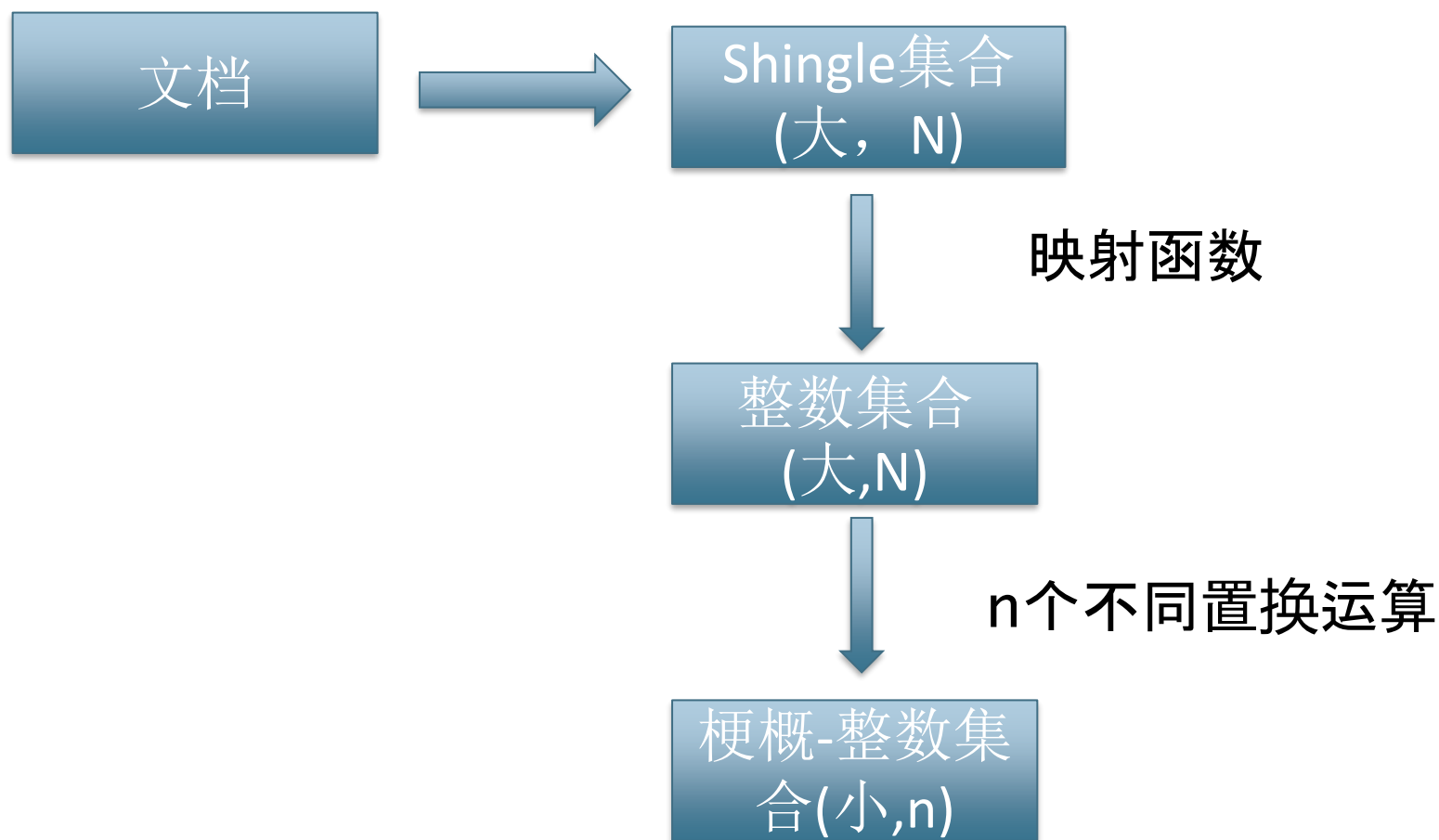
$$2 \neq 0 = \min(g(d_2))$$

$$\hat{J}(d_1, d_2) = \frac{0+0}{2} = 0$$

	d_1 slot		d_2 slot	
h	∞		∞	
g	∞		∞	
$h(1) = 1$	1	1	–	∞
$g(1) = 3$	3	3	–	∞
$h(2) = 2$	–	1	2	2
$g(2) = 0$	–	3	0	0
$h(3) = 3$	3	1	3	2
$g(3) = 2$	2	2	2	0
$h(4) = 4$	4	1	–	2
$g(4) = 4$	4	2	–	0
$h(5) = 0$	–	1	0	0
$g(5) = 1$	–	2	1	0

最终的梗概

文档的表示过程



本讲内容

- 网页采集的概念
- 一个简单的采集器
- 一个真实的采集器
- 分布式索引

提纲

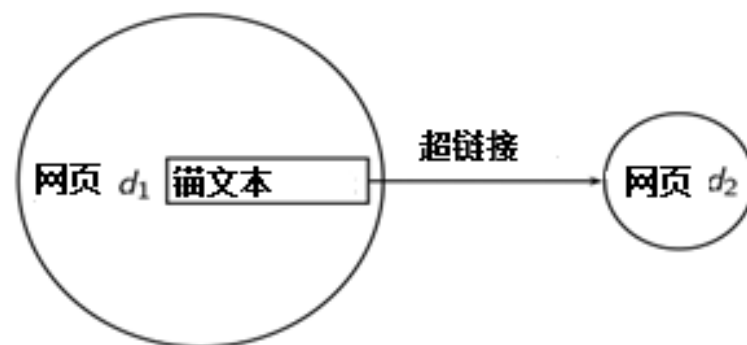
- ① 上一讲回顾
- ② 一个简单的采集器
- ③ 一个真实的采集器
- ④ 分布式索引

采集会有多难？

- Web搜索引擎必须要采集网页文档
- 其他有些IR系统获得文档内容相对容易一些
 - 比如，对硬盘上所有文档建立索引只需要基于文件系统
进行迭代式扫描即可
- 但是对于Web IR系统来说，获得文档内容需要更长的时间 ...
- ... 这是因为存在延迟
- 但是这真的是系统设计中的一个难点吗？

基本的采集过程

- 初始化采集URL种子队列;
- 重复如下过程:
 - 从队列中取出URL
 - 下载并分析网页
 - 从网页中抽取更多的UR
 - 将这些URL放到队列中
- 这里有个“Web的连通性很好”的基本假设



课堂思考题: 下列爬虫有什么问题?

```
urlqueue := (some carefully selected set of seed urls)
while urlqueue is not empty:
    myurl := urlqueue.getlastanddelete()
    mypage := myurl.fetch()
    fetchedurls.add(myurl)
    newurls := mypage.extracturls()
    for myurl in newurls:
        if myurl not in fetchedurls and not in urlqueue:
            urlqueue.add(myurl)
            addtoinvertedindex(mypage)
```

上述简单采集器的问题

- 规模问题: 必须要分布式处理
- 我们不可能索引所有网页, 必须要从中选择部分网页, 如何选择?
- 重复网页: 必须要集成重复检测功能
- 作弊网页和采集器陷阱: 必须要集成作弊网页检测功能
- 礼貌性问题: 对同一网站的访问按遵照协议规定, 并且访问的间隔必须要足够
- 新鲜度(freshness)问题: 必须要定期更新或者重采
 - 由于Web的规模巨大, 我们只能对一个小的网页子集频繁重采
 - 同样, 这也存在一个选择或者优先级问题

采集规模的数量级

- 如果要在一个月內采集20,000,000,000个页面...
- ... 那么必须要在一秒內大概采集 8000个网页!
- 由于我们采集的网页可能重复、不可下载或者是作弊网页, 实际上可能需要更快的采集速度才能达到上述指标

采集器必须做到

礼貌性

- 不要高频率采集某个网站
- 仅仅采集robots.txt所规定的可以采集的网页

鲁棒性

- 能够处理采集器陷阱、重复页面、超大页面、超大网站、动态页面等问题

Robots.txt文件

- 1994年起使用的采集器协议(即规定了采集器对网站的访问限制)
- 例子:
 - User-agent: *
Disallow: /yoursite/temp/
 - User-agent: searchengine
Disallow:
- 采集时, 要将每个站点的 robots.txt放到高速缓存中, 这一点相当重要

Example of a robots.txt (nih.gov)

```
User-agent: PicoSearch/1.0
Disallow: /news/information/knight/
Disallow: /nidcd/
...
Disallow: /news/research_matters/secure/
Disallow: /od/ocpl/wag/
User-agent: *
Disallow: /news/information/knight/
Disallow: /nidcd/
...
Disallow: /news/research_matters/secure/
Disallow: /od/ocpl/wag/
Disallow: /ddir/
Disallow: /sdminutes/
```

Example of a robots.txt



```
User-agent: *
Disallow:
Sitemap: //www.qq.com/sitemap_index.xml
Sitemap: http://news.qq.com/topic_sitemap.xml
```



```
User-agent: Baiduspider
Disallow: /

User-agent: baiduspider
Disallow: /
```

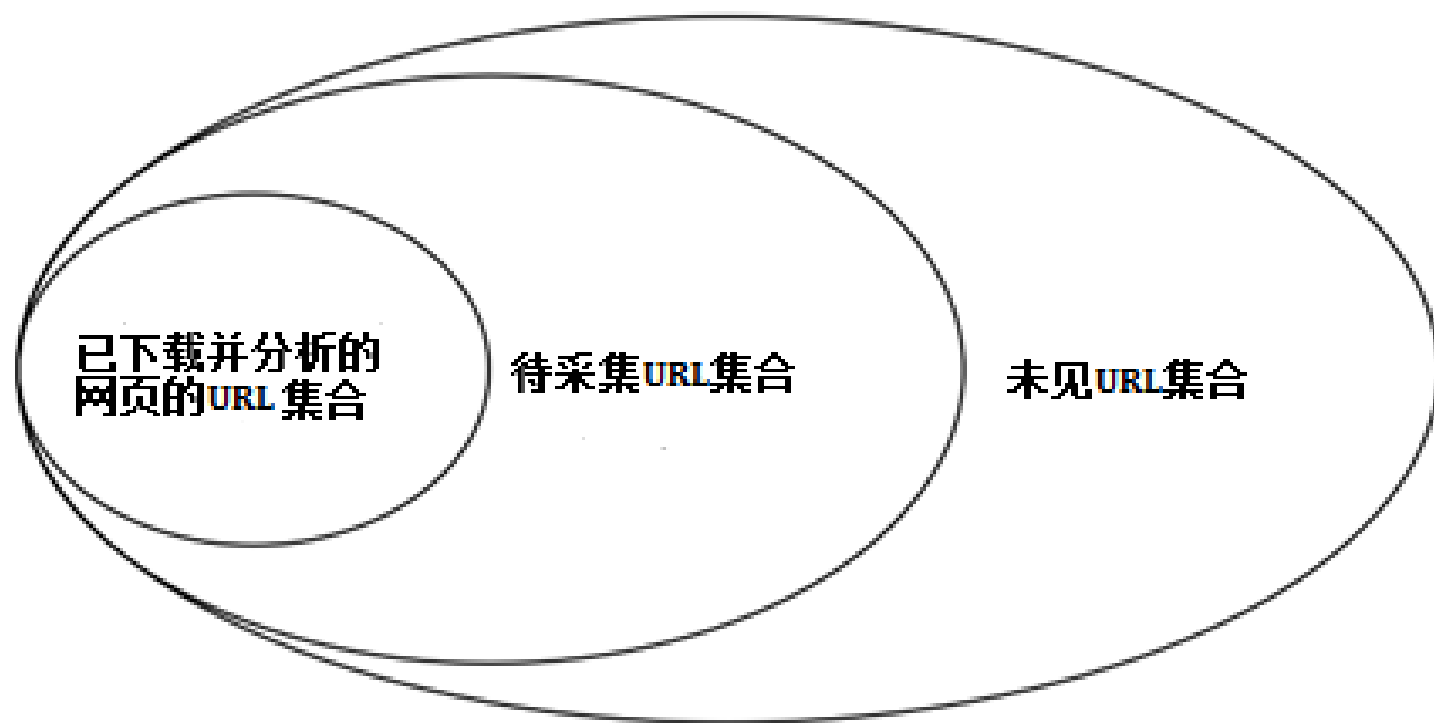
任意一个采集器应该做到

- 能够进行分布式处理
- 支持规模的扩展：能够通过增加机器支持更高的采集速度
- 优先采集高质量网页
- 能够持续运行：对已采集网页进行更新

提纲

- ① 上一讲回顾
- ② 一个简单的采集器
- ③ 一个真实的采集器
- ④ 分布式索引

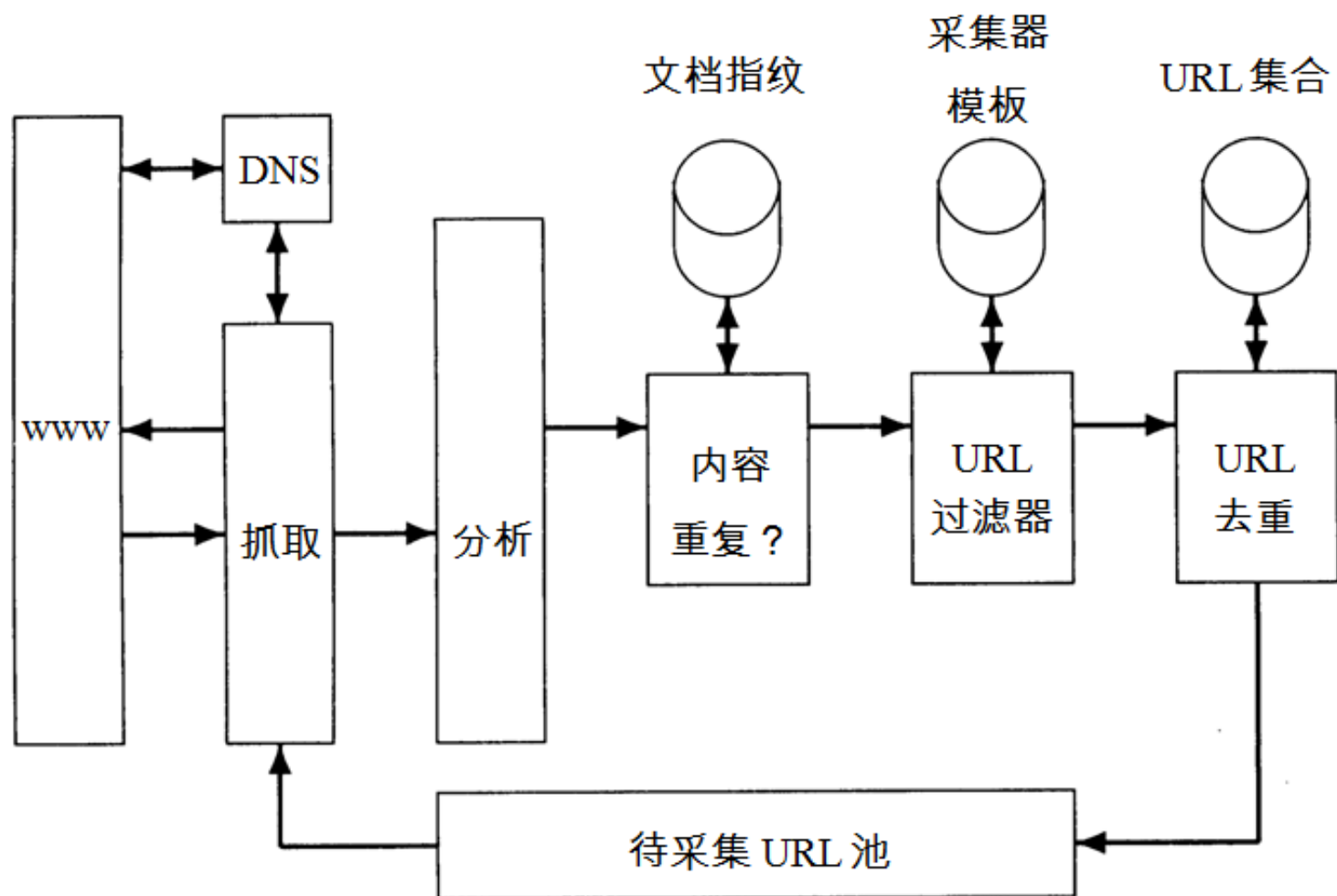
待采集URL池(URL frontier)



待采集URL池

- 待采集URL池是一个数据结构，它存放并管理那些已经看到但是还没有采集的URL集合
- 可能包含来自同一主机的不同页面
- 必须要避免在同一时间采集这些来自同一主机的页面
- 必须要保证采集线程任务饱和

基本的采集架构



URL 规范化(normalization)

- 从网页中抽取的URL有些是相对地址
- 比如，在<http://mit.edu>网站下，我们会采集页面
aboutsites.html
 - 该页面的绝对地址为： `http://mit.edu/aboutsites.html`
- 在网页分析过程中，必须要将相对URL地址规范化

内容重复判别(Content seen)

- 对每个抓取的页面，判断它是否已在索引当中
- 可以采用文档指纹或者shingle的方法判别
- 忽略那些已经在索引中的重复页面

分布式采集

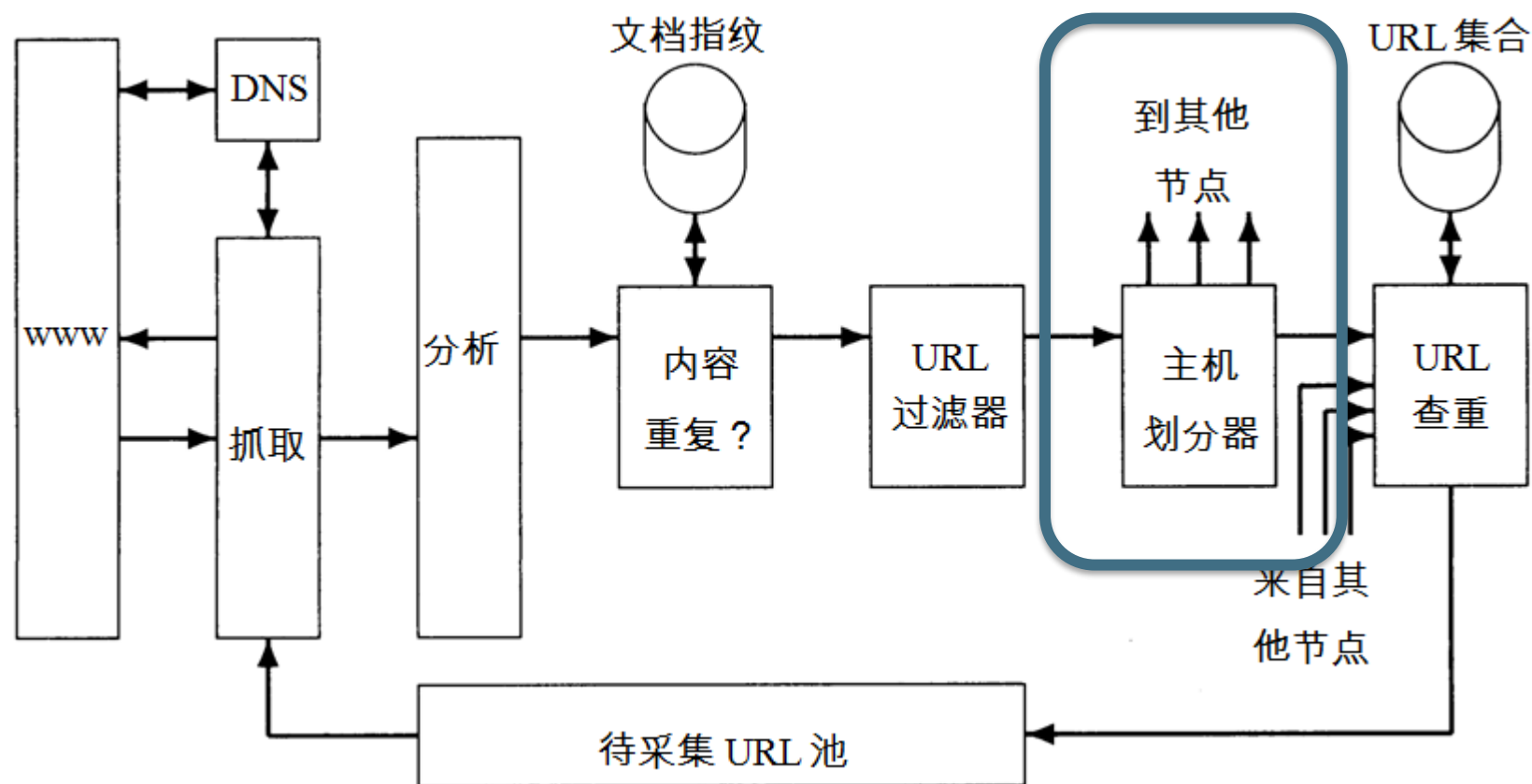
- 运行多个采集线程，这些线程可以分布在不同节点上
 - 这些节点往往在地理上分散在不同位置
- 将采集的主机分配到不同位置上

Google 数据中心



截止2010年，Google共有41个**数据中心**。其中美国有24个、欧洲12个、俄罗斯1个、南美1个和亚洲3个

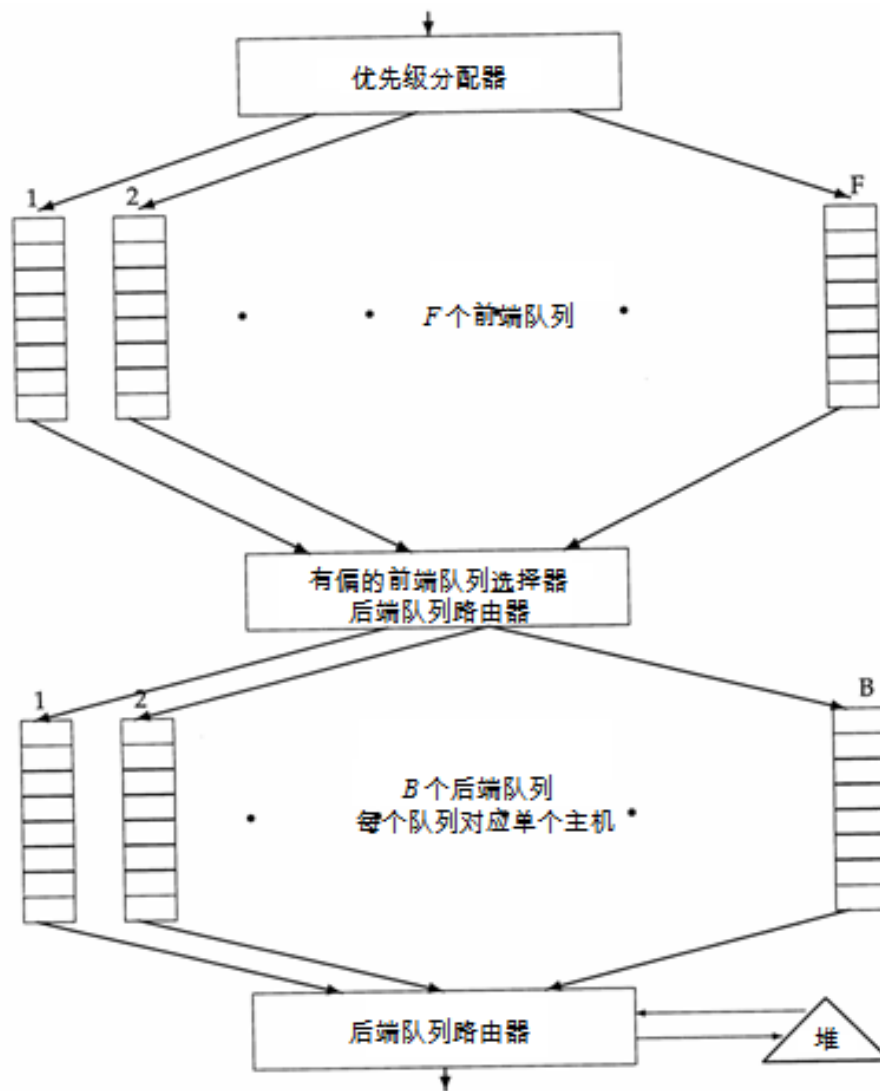
分布式采集器



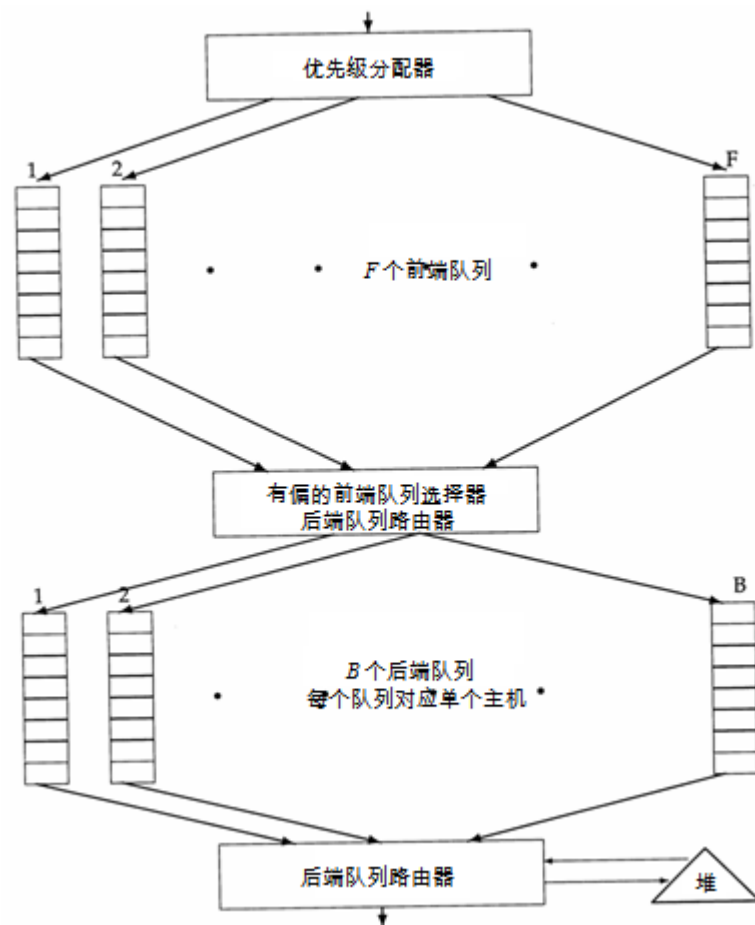
待采集URL池: 主要考虑两点

- 礼貌性: 不要非常频繁地访问某个Web服务器
 - 比如, 可以在两次服务器访问之间设置一个时间间隔
- 新鲜度: 对某些网站的采集频率(如新闻网站)要高于其他网站
- 要实现上述功能并不容易, 一个简单的优先级队列难以成功

Mercator 中的待采集URL缓冲池

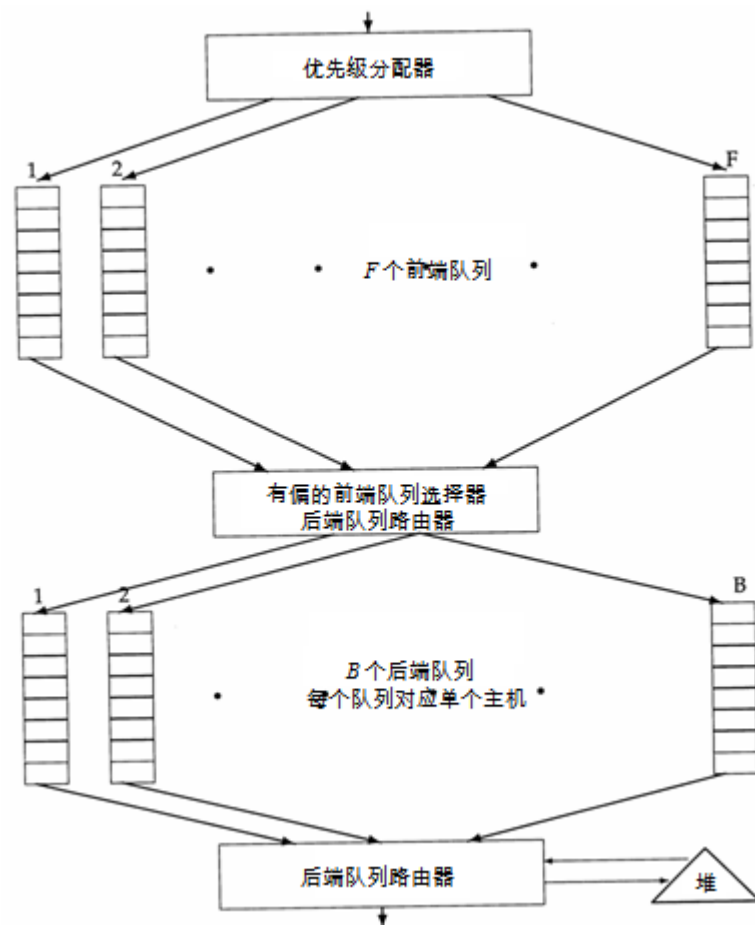


Mercator 中的待采集URL缓冲池



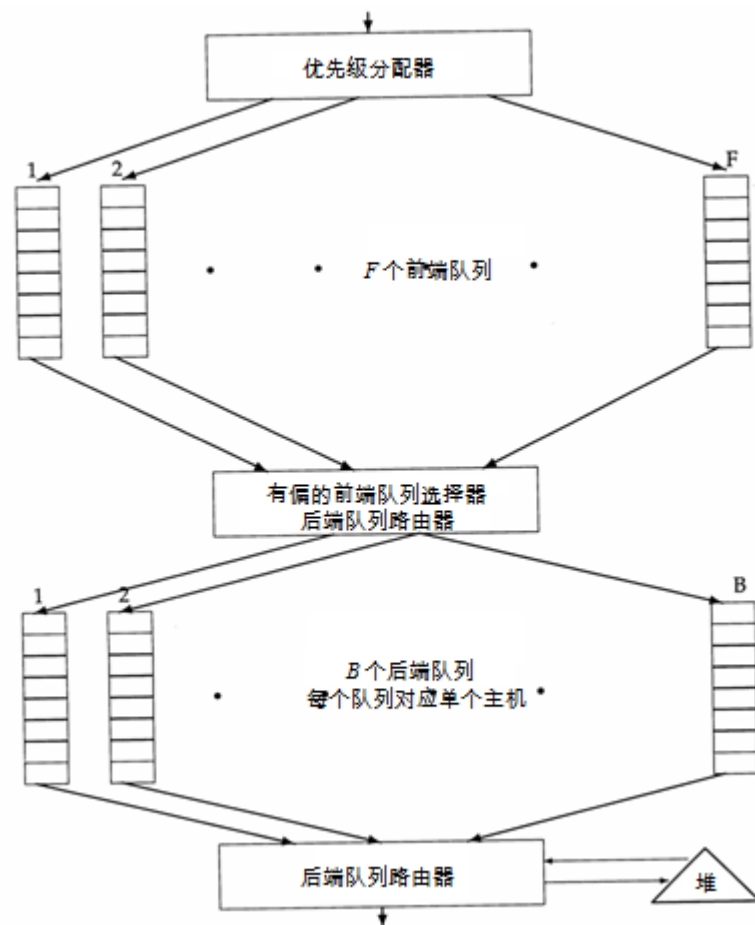
- URL从上部流入缓冲池

Mercator 中的待采集URL缓冲池



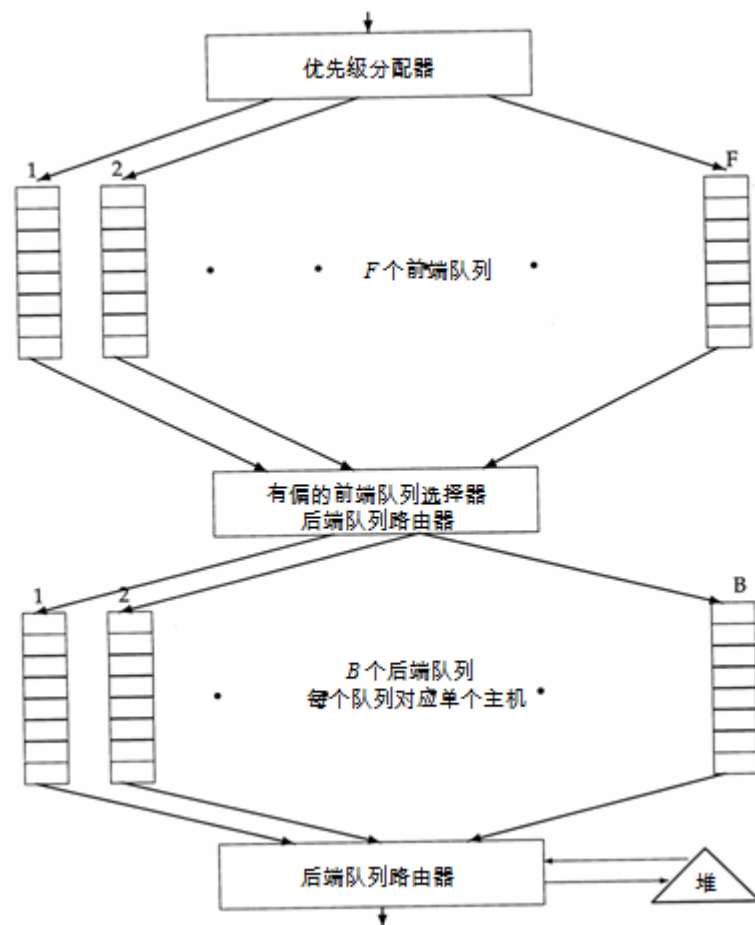
- URL从上部流入缓冲池
- 前端队列(Front queue)管理优先级

Mercator 中的待采集URL缓冲池



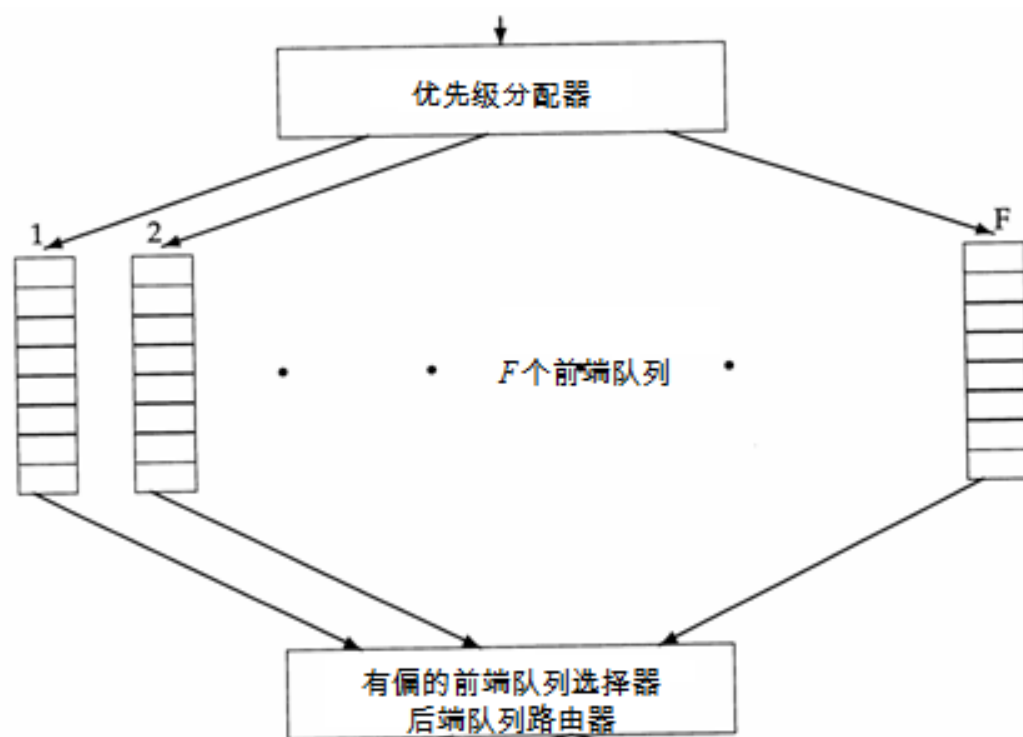
- URL从上部流入缓冲池
- 前端队列(Front queue)管理优先级
- 后端队列(Back queue)实现礼貌性

Mercator 中的待采集URL缓冲池

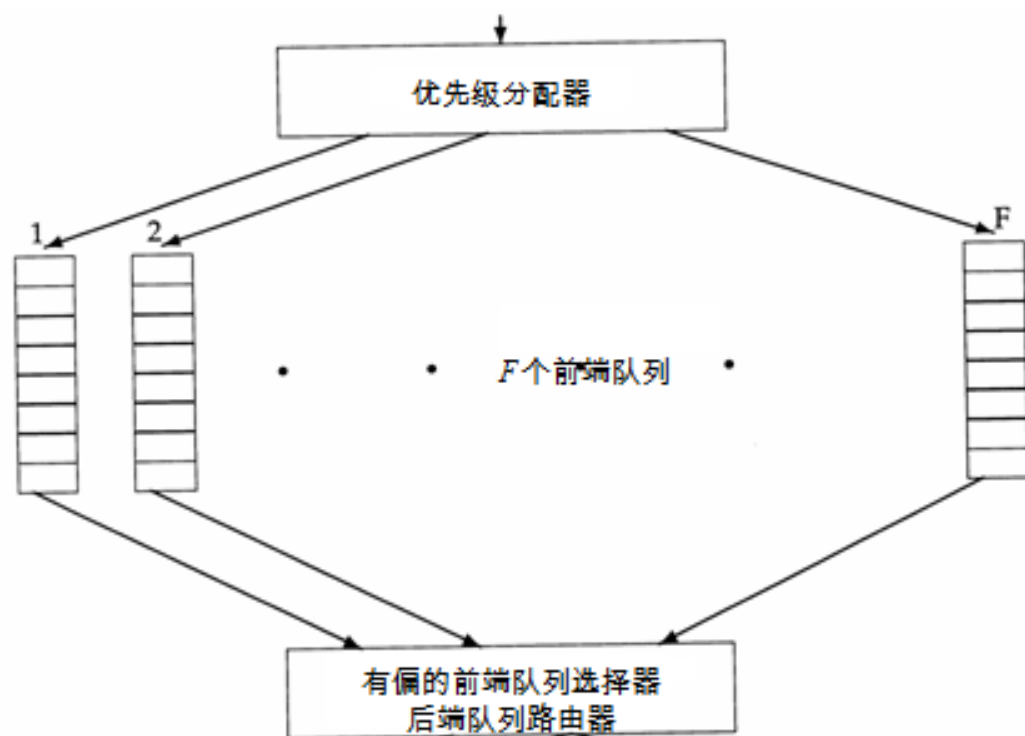


- URL从上部流入缓冲池
- 前端队列(Front queue)管理优先级
- 后端队列(Back queue)实现礼貌性
- 每个队列都是FIFO

Mercator 中的待采集URL缓冲池： 前端队列(Front queue)

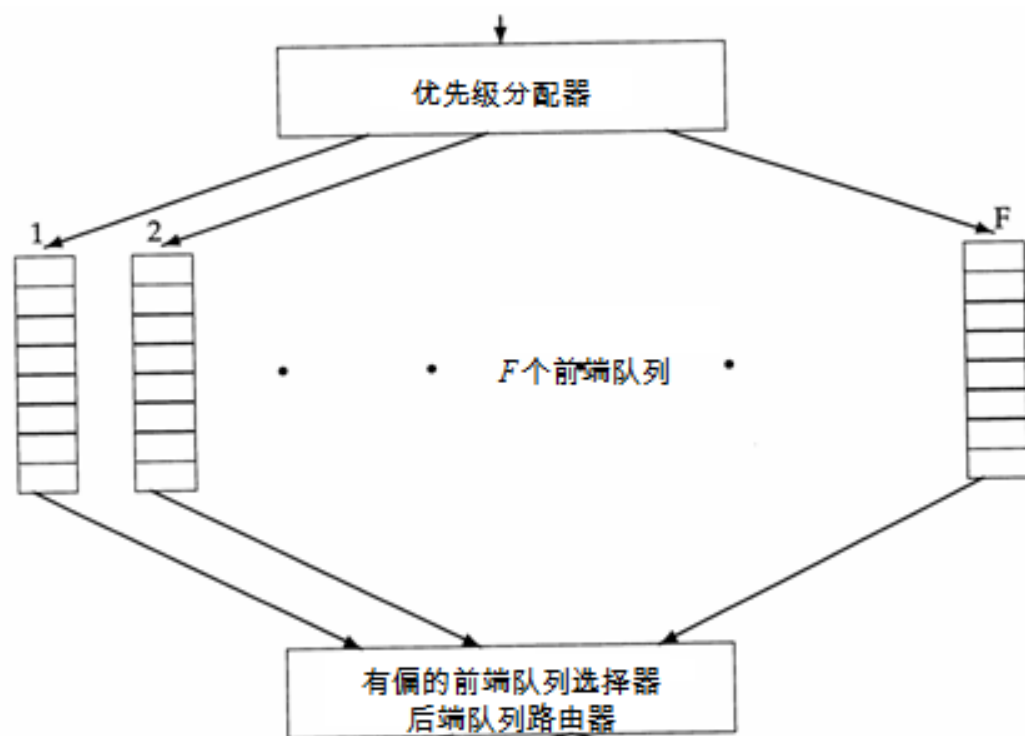


Mercator 中的待采集URL缓冲池： 前端队列(Front queue)



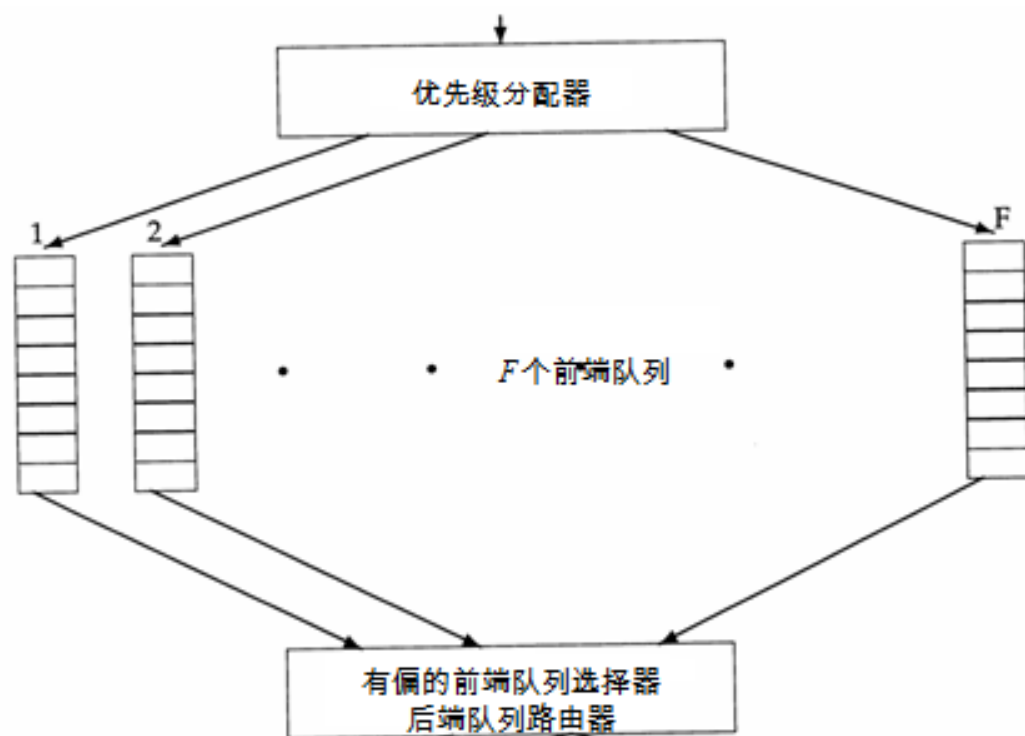
- 优先级分配器给每个URL分配一个1到 F 之间的优先级整数

Mercator 中的待采集URL缓冲池： 前端队列(Front queue)



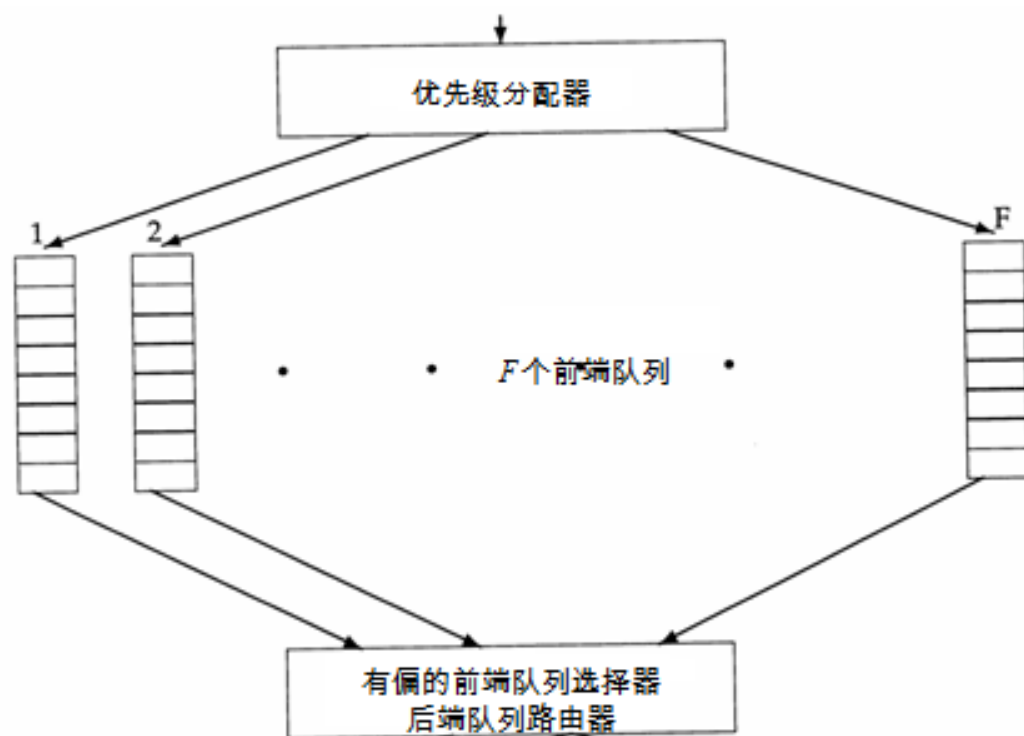
- 优先级分配器给每个URL分配一个1到 F 之间的优先级整数
- 然后将URL添加到相应的队列中

Mercator 中的待采集URL缓冲池 ：前端队列(Front queue)



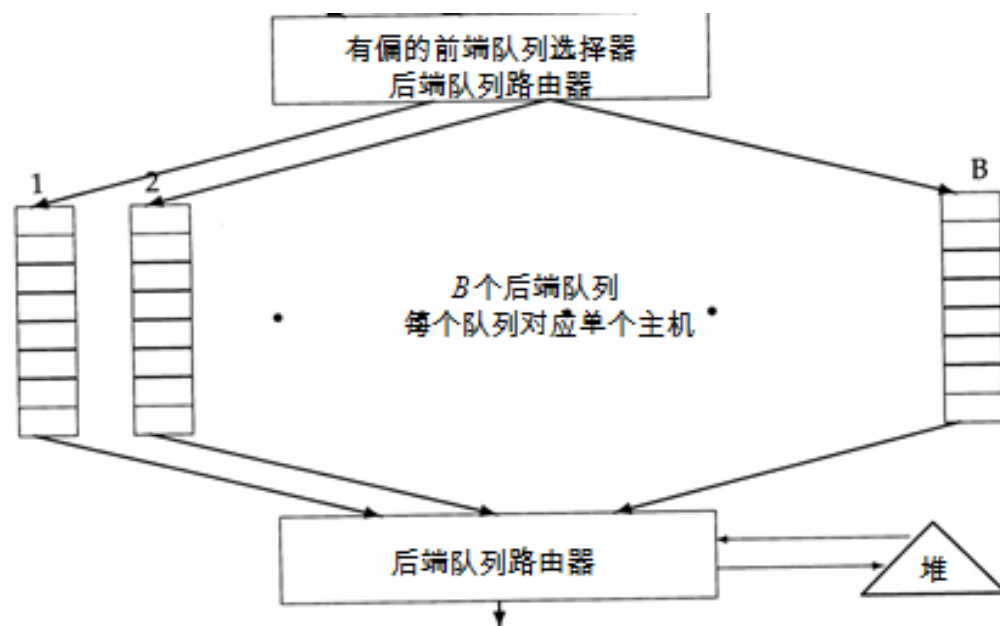
- 优先级分配器给每个URL分配一个1到 F 之间的优先级整数
- 然后将URL添加到相应的队列中
- 分配优先级可以基于启发式信息：更新率、PageRank等等

Mercator 中的待采集URL缓冲池 ：前端队列(Front queue)

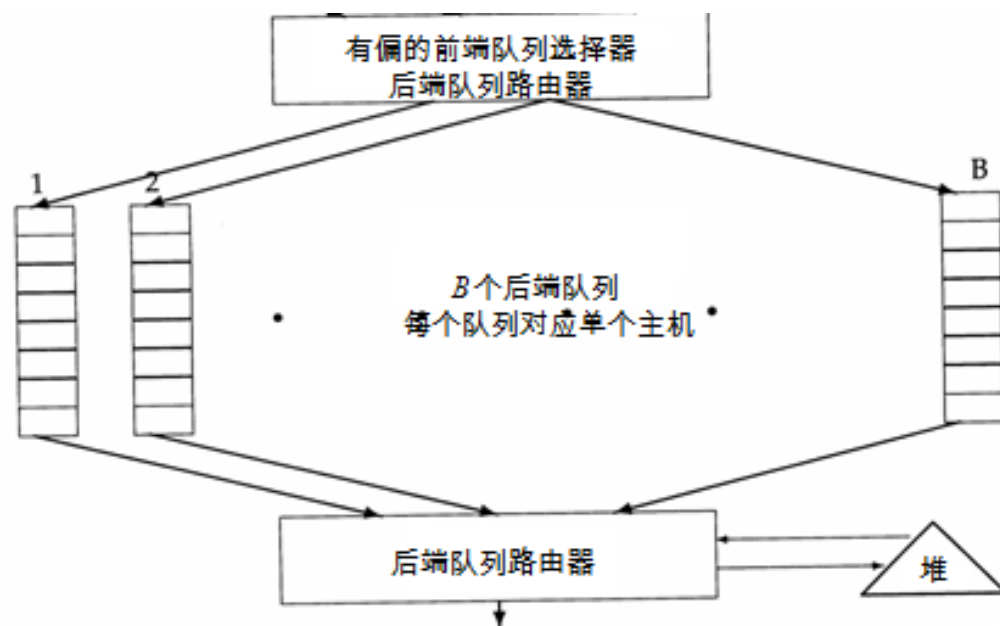


- 从前端队列中进行选择
由后端队列发起
- 选择一个前端队列来选
择下一个URL：轮询法
(Round robin)、随机法
或者更复杂的方法
- 但是上述选择过程倾向
于高优先级的前端队列

Mercator 中的待采集URL缓冲池： 后端队列(Back queue)



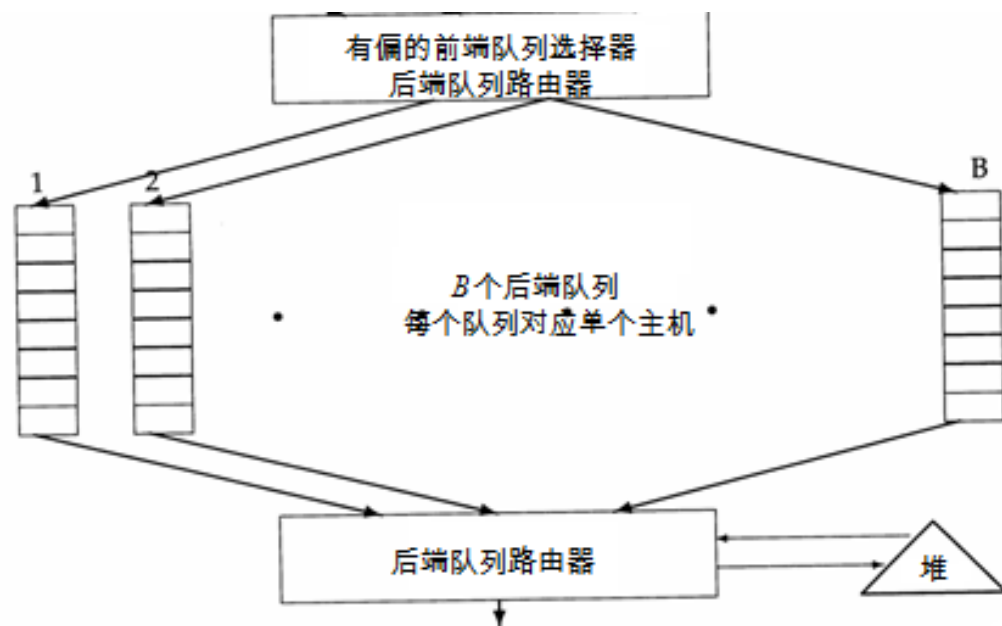
Mercator 中的待采集URL缓冲池： 后端队列(Back queue)



- **恒定情形1：** 当采集器在运行时，每个后端队列不为空
- **恒定情形2：** 每个后端队列中仅存放来自同一主机的URL
- 维护一张主机到后端队列的表

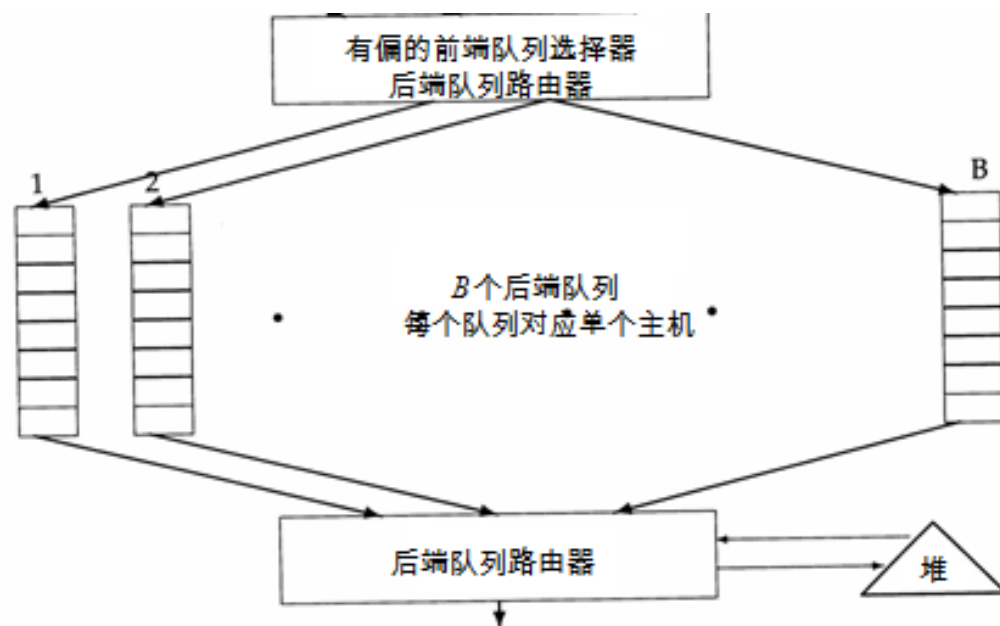
主机	后端队列
standford.edu	23
microsoft.com	47
acm.org	12

Mercator 中的待采集URL缓冲池 : 后端队列(Back queue)



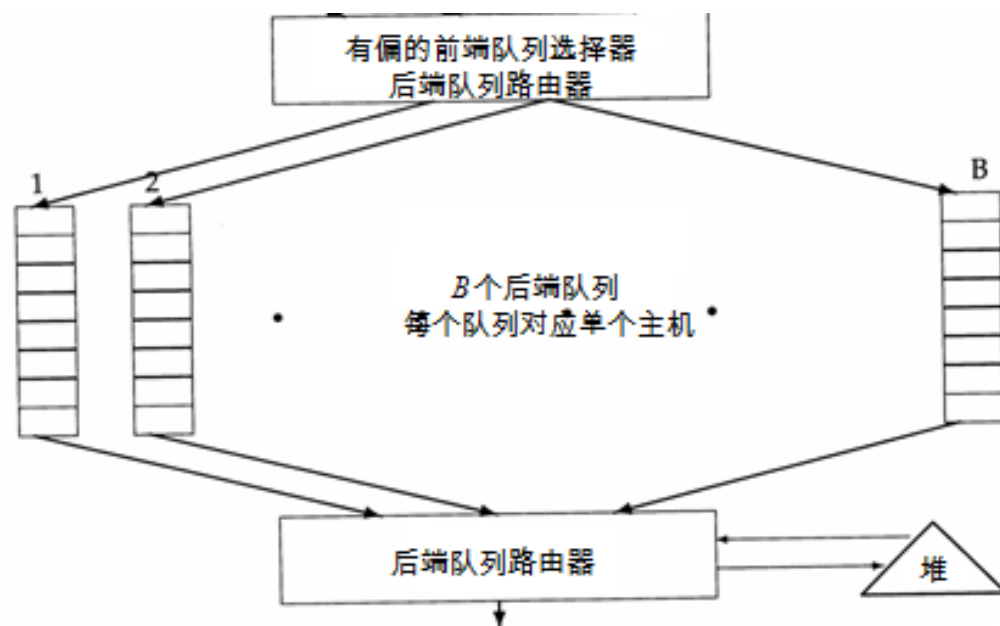
- 在堆中:
- 每个后端队列对应一个元素
- 元素值为该队列对应的主机重新访问的最早时间 t_e
- 该时间 t_e 由下列因素确定 (i) 上次访问该主机的时间 (ii) 时间间隔的启发式方法

Mercator 中的待采集URL缓冲池 ：后端队列(Back queue)



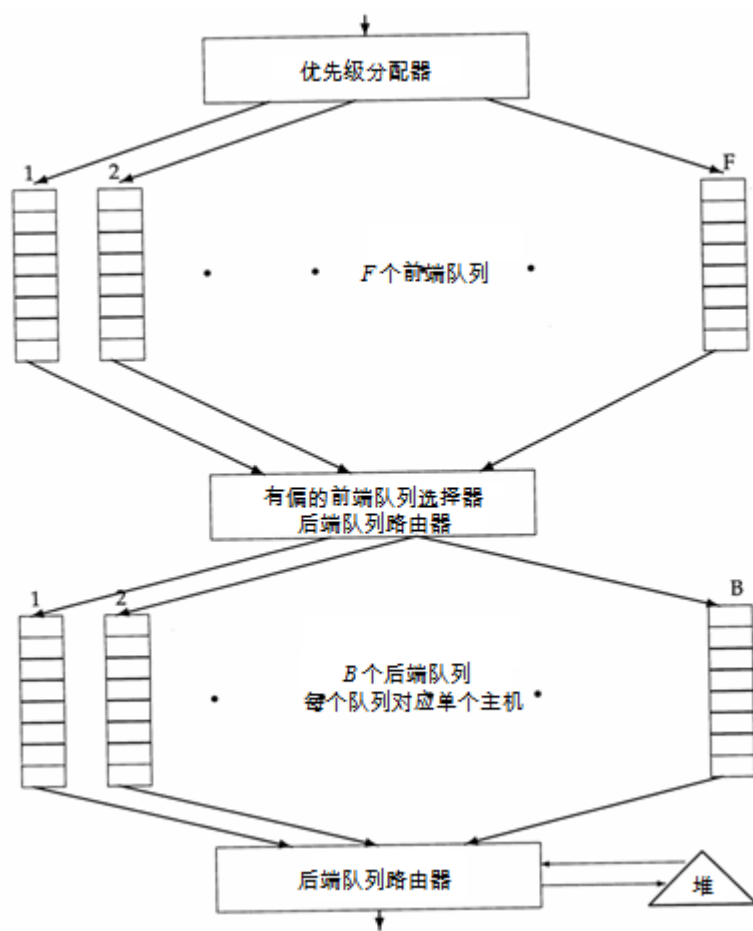
- 抓取器与后端队列交互方法：
- 重复下列操作： (i) 抽取堆中的当前根节点 q (q 是一个后端队列)
- 并且 (ii) 抓取 q 中的头部URL $u \dots$
- \dots 直至 q 为空 \dots
- (也就是说一直抓到 u 为 q 中最后一个URL为止)

Mercator 中的待采集URL缓冲池： 后端队列(Back queue)



- 一旦后端队列 q 为空:
- 重复下列操作 (i) 从前端队列中将一系列URL u 推入并且 (ii) 将 u 加到相应的后端队列中..
- ...
- 直到得到一个 u , u 的主机没有对应的后端队列为止
- 然后将 u 放入 q 并为它建立一个堆元素

Mercator 中的待采集URL缓冲池



- URLs从上流入到缓冲池
- 前端队列管理优先级
- 后端队列保证礼貌性

采集器陷阱(Spider trap)

- 一些恶意的服务器可以产生无穷的链接网页序列
- 一些复杂的采集器陷阱产生的页面不能简单地判断为动态页面

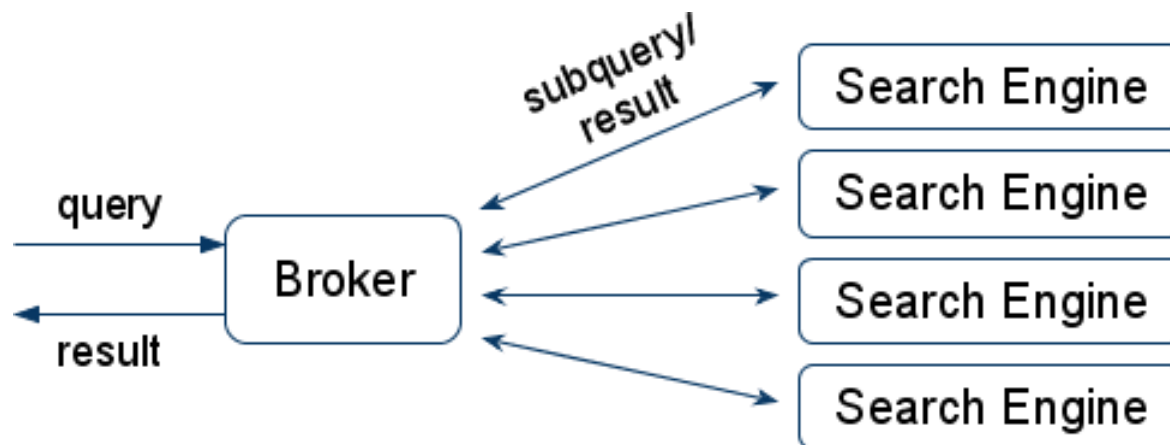
提纲

- ① 上一讲回顾
- ② 一个简单的采集器
- ③ 一个真实的采集器
- ④ 分布式索引

分布式索引

- 随着文档规模不断增长，IR系统难以管理
- 检索和索引的成本随着文档规模增大而增加
- 文档集合越大，检索响应时间越长
- 系统管理的文档数不断增加，性能不断下降，直至系统不可用
- 需要采用分布式架构和算法

提升查询响应速度（1）

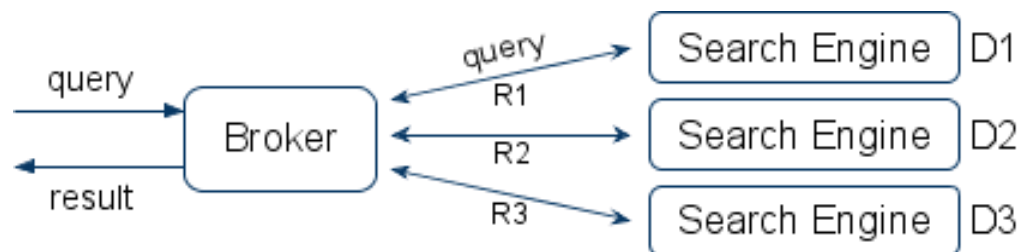


- Broker作用
 - 将查询分解为多个子查询，将子查询发给多个SE程序
 - 聚合查询结果

提升查询响应速度（2）

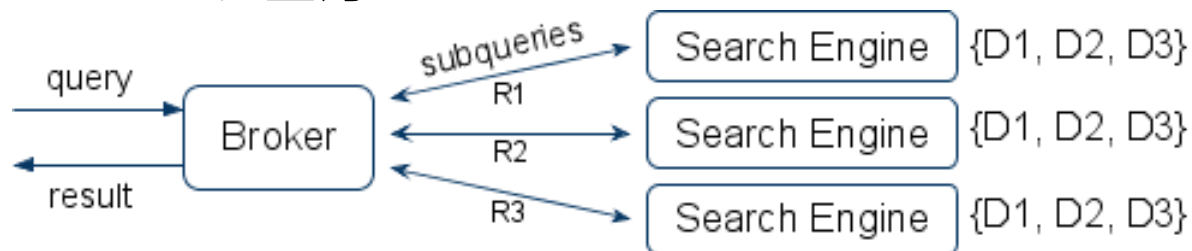
- 提升查询响应速度的两个方法
- 假定文档集为: {D1, D2, D3}

Case 1: 在不同的文档集上执行查询请求



$$\text{result} = R1 \cup R2 \cup R3$$

Case 2: 切分查询

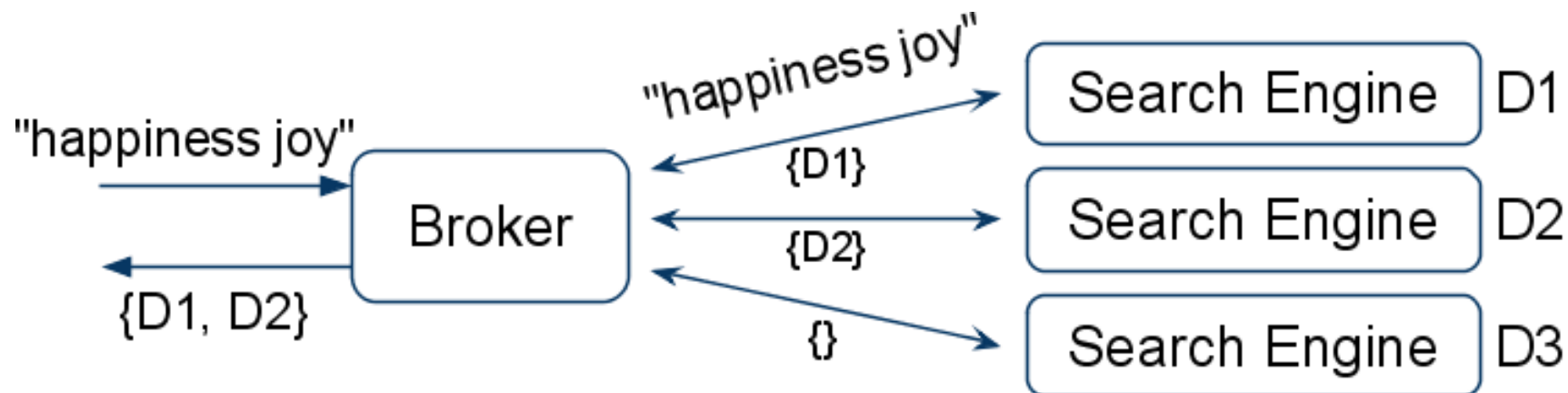


$$\text{query} = Q1 \cup Q2 \cup Q3$$

$$\text{result} = R1 \cup R2 \cup R3$$

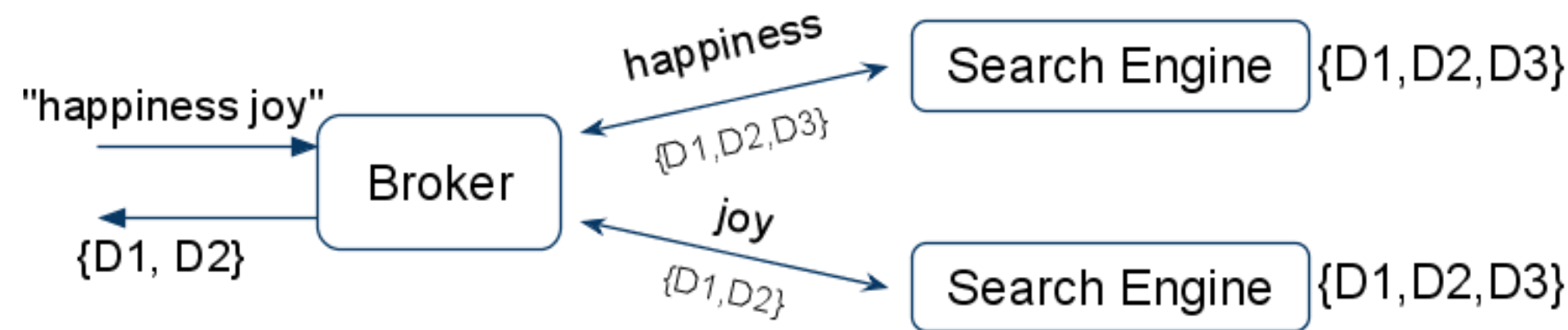
Case 1: 示例

- D1 = *Happiness* is a feeling characterised by stupidity, weird, well being, or *joy*
- D2 = *Happiness* is feeling of *joy*.
- D3 = The pursuit of *Happiness*
- QUERY = "happiness joy"



Case 2: 示例

- D1 = *Happiness* is a feeling characterised by stupidity, weird, well being, or *joy*
- D2 = *Happiness* is feeling of *joy*.
- D3 = The pursuit of *Happiness*
- QUERY = "happiness joy"



$$\text{RESULT} = \{D1, D2, D3\} \cap \{D1, D2\} = \{\mathbf{D1}, \mathbf{D2}\}$$

思考

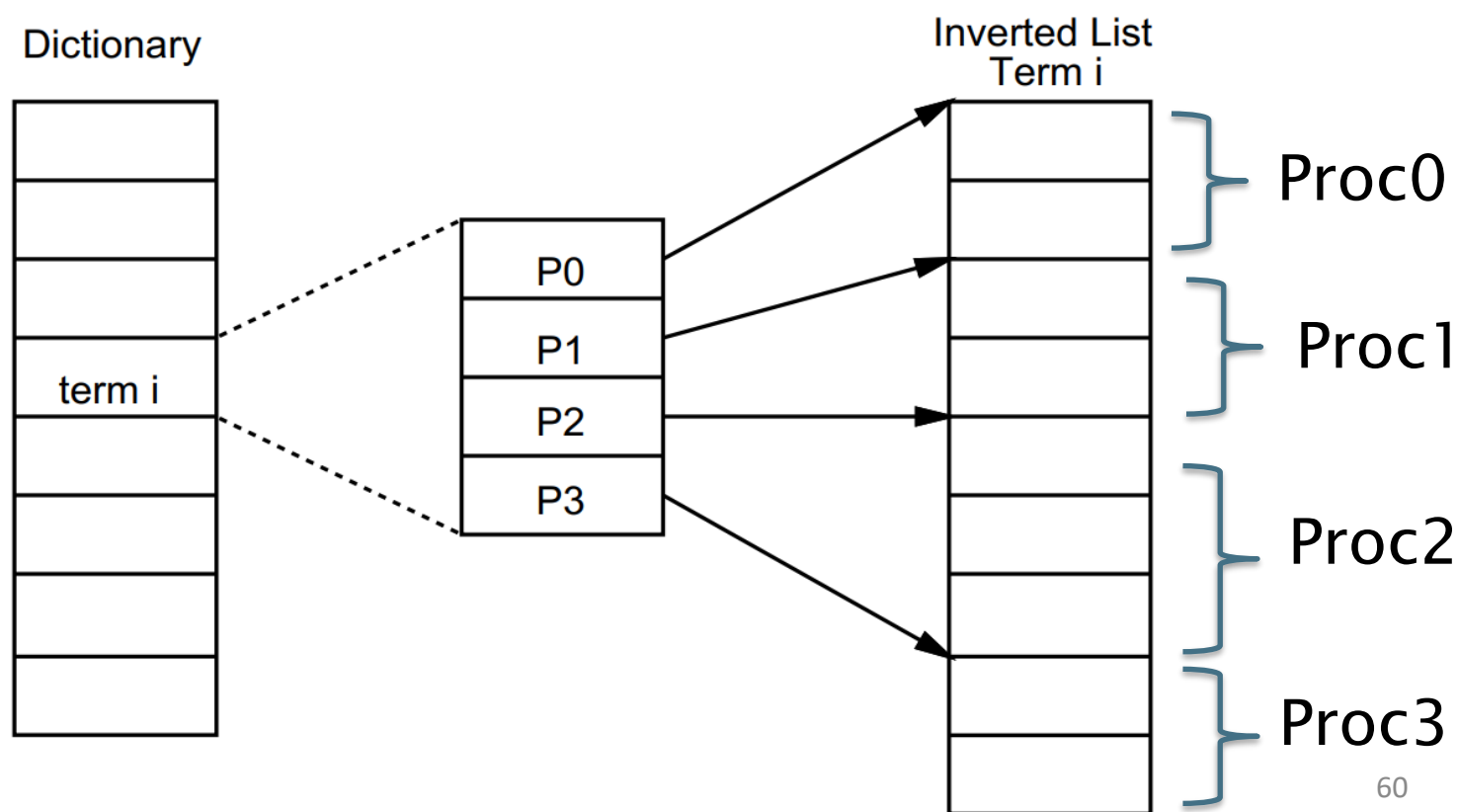
- 如何将文档分布在不同的检索服务器上？

文档集分区方法

- 文档分区法（横向切分法）
 - N个文档在系统的P个处理器/节点间进行分布
 - P个文档子集，每个子集包含文档数 N/P
 - 查询过程中，每个处理器/节点仅处理其文档子集
 - 每个文档子集的结果集汇聚后形成最终结果
- 词项分区法（纵向切分法）
 - 将词项分布到P个处理器/节点上
 - 针对每个文档的检索，将分布到多个处理器/节点上进行

逻辑文档分区法

- 基于原始倒排表进行扩展，每个词项增加P个指针，用于将 posting list切分成P份，每个处理器处理一份文档集



物理文档分区法

- 文档被物理的分成多个子集
- 每个子集拥有独立的倒排索引
- 不同处理器/节点，在查询过程中相互独立
- 系统收到查询请求时，Broker将查询请求分发给所有处理器/节点
- 每个处理器/节点查询各自文档子集，生成中间结果集
- Broker收集中间结果，归并为最终结果集
- P个中间结果集的归并，可采用基于二叉堆的优先级队列（binary heap-based priority queue）

两种文档分区法对比

- 逻辑文档分区法进程之间通信比物理文档分区少
 - 逻辑分区总体性能更高
- 物理文档分区灵活性更高
 - 每个文档分区可独立进行查询
- 将一个简单的IR系统转变为并行/分布式系统，物理文档分区法更容易
- 两种方法都可以使用线程原语实现并发查询

词项分区法

- 纵向切分索引表
- 倒排表在多个处理器/节点之间进行分布
- 每个查询将被分解为单个的查询词项，发给对应的处理器
- 查询请求可以并发处理，每个处理器返回部分查询请求对应的结果
- 查询负载很难平衡，影响该类系统的并发处理能力
- 因此该方法主要目标是查询中涉及的处理器/节点尽量少，负载在处理器/节点中均衡分布
- 一种方法：利用查询日志来指导词项分区

文档分区与词项分区对比

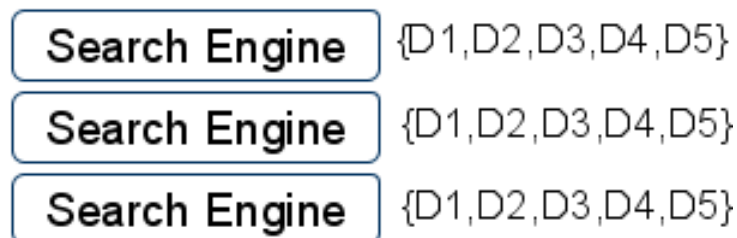
- 文档分区法在倒排索引创建和维护方面更简单
- 每个处理器有独立的I/O通道和磁盘时，文档分区法性能更好
- 如果用户查询请求的词项分布均匀，词项分区法性能更优
- [Webber et al\(2007\)](#)等验证了词项分区法的资源利用率更低，降低了磁盘访问次数和交换的数据量

总体对比

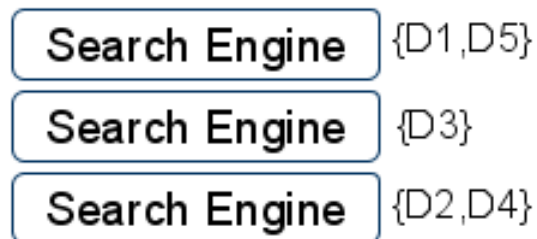
- 文档分区法的主要缺点：
 - 需要向文档子集发起很多不必要的查询操作，这些文档集中可能基本没有相关文档
- 词项分区法的主要缺点：
 - 需要构建和维护全局索引，限制了该方法的可扩展性
- 此外，词项分区法的响应时间的波动范围较大，需要引入复杂的均衡机制以保证响应时间

其他文档集分区方法

- 在所有检索节点上保存全量索引的副本

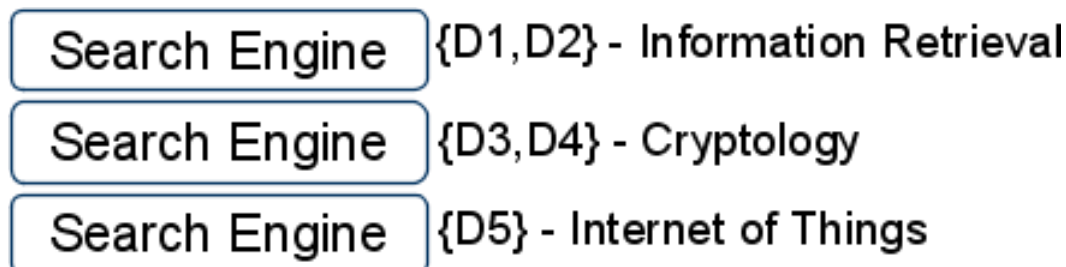


- 随机分布



语义分布

- 按主题分区
- 按词表范围分区
- 按时间范围分区



本讲小结

- 网页采集的概念
- 一个简单的采集器
- 一个真实的采集器Mercator
- 分布式索引

参考资料

- 《信息检索导论》第20章
- Modern Information Retrieval, Chapter 9, Parallel and Distributed IR, book by Ricardo Baeza-Yates and Berthier Ribeiro-Neto
- Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications. Ion Stoica , Robert Morris
- Heydon等人撰写的有关Mercator的论文： Allan Heydon , Marc Najork. Mercator: A scalable, extensible web crawler. WWW 1999. (<http://www.bagualu.net/linux/crawler.pdf>)
- 采集协议标准

课后练习

- 无!