doi:10.3772/j.issn.1002-0470.2014.06.003

Web 搜索引擎的一种检索结构优化方法①

钱立兵② 季振洲

(哈尔滨工业大学计算机科学与技术学院 哈尔滨 150001)

摘 要 为了提升 Web 引擎的检索服务性能和扩展性,提出了一种索引划分方法,依据该方法改进了检索结构,避免了冗余查询,并加速了内部并行化查询过程。按照文档主题和相似 URL 对索引文档分类,而索引节点内则按照词项划分,实现了索引节点查询间和查询内的并行。根据索引分类的结构,改进了系统内部查询逻辑,设计了 Aggregator(简称 Agg)进程队列,实现了异步处理高并发查询。实验结果表明,优化的 Web 引擎结构总体上能够降低查询开销,提高系统吞吐量,相对于传统 Web 搜索模型,其查询速度和吞吐量分别提高了 20% 和 25%。

关键词 Web 搜索引擎, 分布式搜索, 检索结构, 吞吐量, 分类索引

0 引 言

传统 Web 引擎[1,2]适合在小规模数据索引上进行较轻量的查询处理,而在大规模索引数据和高并发的请求任务情况下,系统模型的查询性能不高、扩展性差,很难满足大规模数据和任务的需求,如何改进传统模型结构,设计快速的查询算法是 Web 引擎的关键要素[3]。传统模型通常用文档划分或词项划分方法[4]把文档数据分配到索引节点上。文档划分最大的优点是划分简单[5],每个索引节点管理各自文档,索引节点间独立运行,但是索引节点存在重复和冗余查询[6]以及查询负载不平衡[2,6]等性能问题。词项划分把文档分成词项集, 通常按照流水线方式[7-9]查询,在解决磁盘寻道上比较高效[10],但此种方式在查询内部存在依赖关系,不适合查询内部并行。为了发挥两种索引划分的优点,Marín[11]提出了文档与词项的混合划分,按照后缀数组而非倒排文件的分布式查询处理,然而查询性能并没有明显提高,合并各个索引节点的工作仍然没有减少。

传统模型对索引进行划分后,接收服务在进行查询时需要通过广播向索引节点发送请求,每一个请求都会牵动所有索引节点执行海选、相关性计算、精选[12]等过程,然后再由接收服务搜集各个节点查询结果。对于高并发查询请求和大规模索引数据,索引节点的查询负载和接收服务结果融合的负载将会成倍增加,使得传统模型难以负荷。为了解决高并发查询请求的问题,本文对传统模型进行了优化:改进了 Marín[11]的混合划分策略,首先利用文档主题、相似统一资源定位器(uniform resource locator, URL)对文档进行分类,使得同类主题文档分配到同一索引节点内部,然后在索引节点内部按照词项划分,建立查询内可并行的索引,实现查询内和查询间并行;在此基础上,接收服务按照查询类别查找相应的索引节点,实现针对性查询,提高系统可扩展性。

1 传统的 Web 搜索模型

典型的 Web 搜索模型由三个层次组成[12](见图 1),即应用展现层(processing of front, PF)、结果汇聚层(Merge, 又称接收服务层)和独立引擎层(search engine, SE)。

PF 层负责解析用户提交的查询,包括用户点击分析[13],查询意图分析[14]、查询扩展[15]等,得到完整的 query。

Merge 层向各个引擎广播发送 PF 层的每次请求,接收相应查询结果并对其排序、融合,发送到 PF 层,最终由 PF 层返回到用户。如何从索引节点中快速得到 topk 个结果[16]是 Merge 层的重要性能。

① 国家自然科学基金(61173024)资助项目。

② 男,1986 年生,博士生;研究方向:并行计算,搜索引擎;联系人,E-mail:qiannu159@163.com

(收稿日期:2013-10-26)

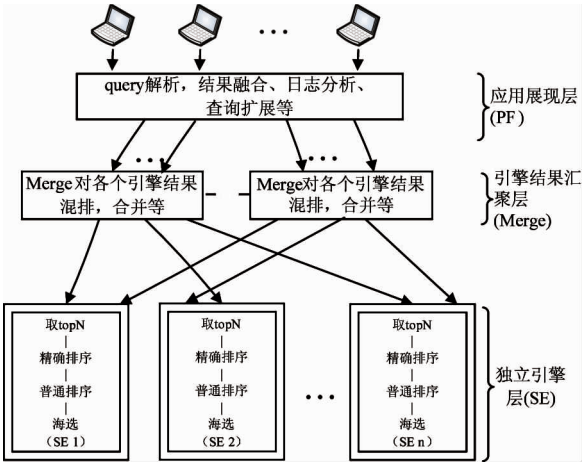

图 1 传统的 Web 搜索引擎模型结构

SE 层按照文档或词项划分索引，每个节点接收到 Merge 层发送的查询请求，查询倒排记录表，按照相关性算法[17-19]、排序算法等得到前 m 个文档结果[16]，然后返回到 Merge 层。

对于规模更大的引擎和高并发查询，Merge 层广播发送和索引节点的结果融合工作使得系统负载成倍增加，索引节点的查询负载与查询并发数同等增长。

2 Web 检索结构的优化

针对传统 Web 搜索模型中 Merge 层和 SE 层存在的广播收发和高负载查询问题，本节提出了一种新的索引分类方法，在此基础上设计了一种异步处理接收和发送的 Agg 进程队列。如图 2 所示，在 SE 层，需要先将索引文档按照主题分类（图中分成 num 个类别（$C_1, C_2, \cdots, C_{num}$），每类索引由同一个 Merge 节点管理，设计 Agg 进程队列，逻辑上构成 Merge 层内部的 Agg 层（为了对问题的表述，设计 Agg 层），用以优化 Merge 查询结构。改进后的检索结构，能够实现索引节点的分类查询。下文将具体阐述这种索引分类思想以及索引节点查询过程。

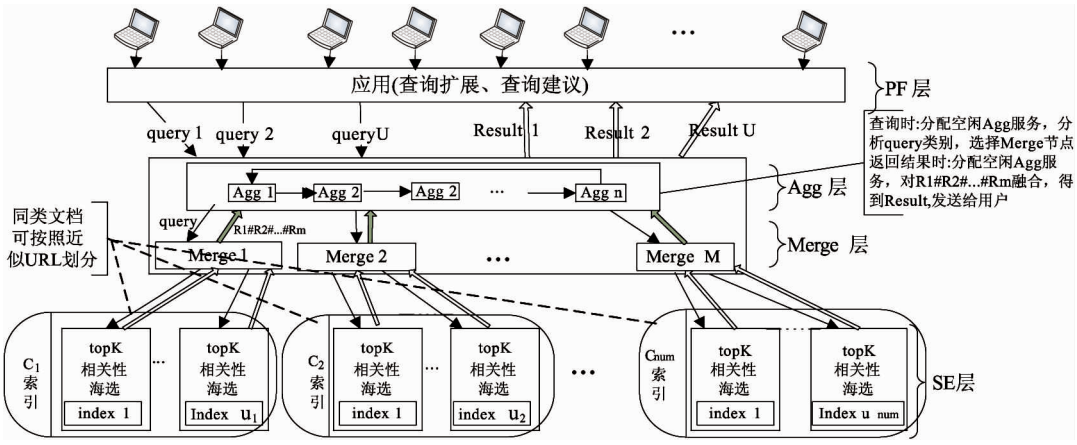

图 2 具有分类索引的改进 Web 搜索引擎结构

2.1 SE 层索引分类的设计

如图 2 所示，SE 层节点按照主题分成 num 个类别的索引，每类索引再按照相似 URL 分类，分配到每个索引节点上。具体实现方法是对 n 个 Web 页面集 Docs 分别提取特征子集，构成文档特征词项矩阵。其中特征子集是指按照每个文档带有权重的特征词组成向量，并将所有特征词按分值大小排序，提取已预订的 m 个最优特征词，即

$$d' = \{(t_1, w_1), (t_2, w_2), \cdots, (t_m, w_m)\} \quad (1)$$

而文档特征词项矩阵可表示为

$$Docs = \begin{bmatrix} d_1 \\ d_2 \\ \cdots \\ d_n \end{bmatrix} \rightarrow \begin{bmatrix} d'_1 \\ d'_2 \\ \cdots \\ d'_n \end{bmatrix} = \begin{bmatrix} (t_{1,1}, w_{1,1}) & (t_{1,2}, w_{1,2}) & \cdots & (t_{1,m}, w_{1,m}) \\ (t_{2,1}, w_{2,1}) & (t_{2,2}, w_{2,2}) & \cdots & (t_{2,m}, w_{2,m}) \\ \cdots & \cdots & \cdots & \cdots \\ (t_{n,1}, w_{n,1}) & (t_{n,2}, w_{n,2}) & \cdots & (t_{n,m}, w_{n,m}) \end{bmatrix} \quad (2)$$

即文档的主题分成 num 个类：

$$Docs = C_1 \cup C_2 \cdots \cup C_i \cdots \cup C_{num} \quad (3)$$

式(1)中的 $w_{i,j}$是指词项的权重。文献[20]提到的词项权重公式未考虑 Web 中标签的权重,本文对此加以改进,假设 Web 页面有 r 个描述的标签,标签的权重计为 $\alpha_i(1 < i < r)$,则

$$w_{i,j}(t_{i,j},d_i) = \frac{TF(t_{i,j},d_i) \times \lg(N/n_k + \varepsilon)}{\sqrt{\sum_{t \in d} [TF(t_{i,j},d_i) \times \lg(N/n_k + \varepsilon)]^2}} \times \left(1 + Bool(t_{i,j}, d_i) \times \frac{\sum_{j \in L} \alpha_j}{\sum_{i=1}^{r} \alpha_i}\right) \quad (4)$$

其中,$w_{i,j}(t_{i,j}, d_i)$为词 $t_{i,j}$在文本 d_i 中的权重,$TF(t_{i,j}, d_i)$为词 $t_{i,j}$在文本 d_i 中词频,N 为训练文本数,n_k 为训练文本集中出现词 $t_{i,j}$的文本数,ε 为调节的修正系数(这里取 0.01),分母是归一化因子,$Bool(t_{i,j}, d_i)$为布尔函数,当 d_i 属于主题标签时为 1,否则为 0,L 为 Web 标签集合,α_i 通过训练文档集获得。

式(2)每一行表示一个 Web 页面,每一列表示所有文档同一级别的特征词权重。依次按照列对文档分类,先按照第一列 $\{t_{1,1}, t_{2,1}, \cdots, t_{n,1}\}$ 的特征词对文档分类(根据文档主题,文档可能属于多类),如果分出的类数没有达到预先定义数目 num,再按照第二列分类,依次类推,直到满足要求。

若式(3)中的 $C_i(1 \leqslant i \leqslant \text{num})$ 文档集仍较大(单个节点索引数据不能承受),还需要对同类文档进行细分。分类方法是提取网页地址的域名,相似 URL 的文档分配到同一节点上(类似字符串相似比较,这里不作讨论),得到:

$$C_i \xrightarrow{\text{按照相似 URL 分类}} \{c_1, c_2, \cdots, c_u\} \quad (5)$$

SE 层的 C_i 类索引再按照 URL 分类,分配到 u 个索引节点上(URL 分类);在索引节点内部按照词项划分。如图 3,设文档集 $Docs = \{D_1, D_2, \cdots, D_{16}\}$,按照式(2)、(3)对文档主题分成 $C_1 = \{D_1, \cdots, D_8\}$,$C_2 = \{D_9, \cdots, D_{16}\}$ 两类文档集,每类文档再按照相似 URL 分类并分配到各个节点。

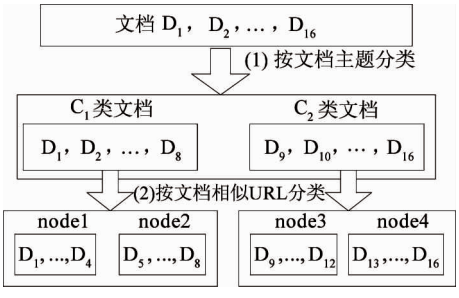

图 3　索引划分:按照主题分类文档,每类再按照 Web 页面 URL 分类

node1 节点和 node2 节点由文档的 URL 分得,分别是 $c_1 = \{D_1, D_2, D_3, D_4\}$、$c_2 = \{D_5, D_6, D_7, D_8\}$ 文档集,如图 4,节点内部利用 4 个核按词项划分倒排表,每个核负责两个词项;节点间为查询间并行、节点内为查询内并行,从而实现查询间和查询内并行化处理。节点内的词项划分按照剪枝方法[18,19]优化索引结构,本文这里不作讨论。在理论情况下,这种混合设计方案能够使得吞吐量增加近 nodes(C_i) $* p$ 倍(p 为节点的核数,nodes(C_i)为本类索引节点数)。

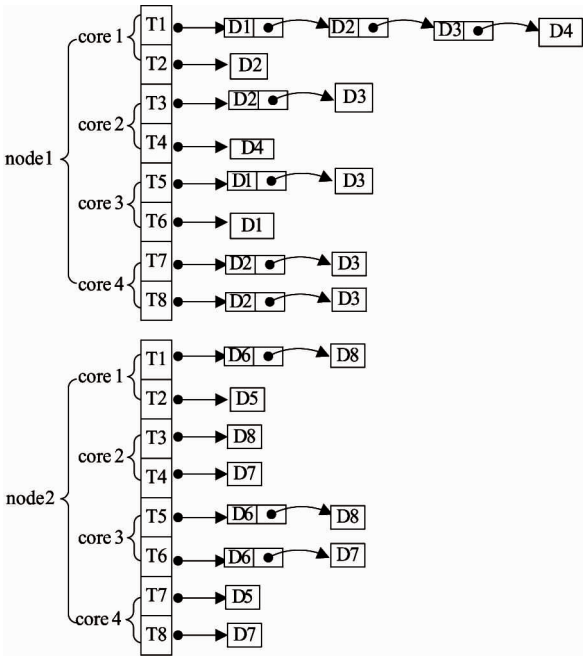

图 4　节点间按照 URL 对文档划分,节点内按照词项划分

2.2　Merge 层并行化查找

在建立分类索引基础上,Merge 层查找过程发生重大变化。同类的索引节点只允许同一 Merge 节点管理(图 2 中 Merge 节点管理一类索引节点),Merge 层节点之间独立并行工作。

利用进程队列 Agg 实现并行处理,使得资源得到最大利用。如图 5,在查询时分配空闲 Agg 进程,分析 query 类别,选择相应类别的 Merge 节点;在 Merge 结果返回时同样分配空闲 Agg 进程,对查询结果 $R_1\#R_2\cdots\#R_i\cdots\#R_t$($R_i$ 搜索结果,#为 R_i 之间拼接符)融合然后返回给用户,最终转化为标准格式(如 XML)返回给用户。

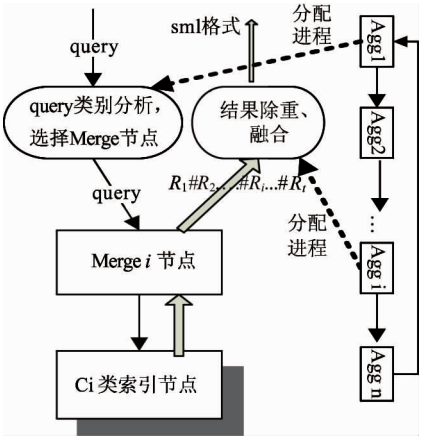

图 5 改进结构的查询流程，动态分配空闲进程处理查询

设 Merge 层有 M 个节点，同个 Merge 节点向相同主题的索引节点集发送请求，图 2 中，Merge i 节点管理 C_i 索引类，在 Merge 层节点较少情况下，可以由每个 Merge 节点管理多类索引。对查询词特征识别，Merge 只需选择相应类别的索引节点进行查询，融合查询结果。如 Merge 得到 query = “NBA”，Merge 层分析该 query 属于体育类（按照文献中分类方法[17]，这里不讨论），只需要查找主题为体育类的索引节点集，然后合并该类索引节点查询结果。

3 查询生命周期及查询工作量对比

对比图 1 和图 2，分析两种结构的查询周期，改进的 Web 引擎查询生命周期：

$$User \xrightarrow[\text{⑧}Html]{\text{①query}} PF \xrightarrow[\text{⑦}XML]{\text{②query}} Agg \xrightarrow[\text{⑥}r_1\#...\#r_m]{\text{③ 选择 Merge 节点}} Merge \xrightarrow[\text{⑤ 返回结果}]{\text{④ 选择索引节点}} SE \tag{6}$$

尽管传统 Web 引擎缺少 Agg 服务，但为了对比，仍按照式(6)访问序列号形式描述传统 Web 引擎查询阶段：

$$User \xrightarrow[\text{⑧}Html]{\text{①query}} PF \xrightarrow[\text{⑥⑦ 多次 xml 返回}]{\text{②③ 多次请求}} Merge \xrightarrow[\text{⑤ 多次结果返回}]{\text{④ 多次广播发送请求}} SE \tag{7}$$

比较式(6)和式(7)，改进 Web 模型增加了 Agg 服务分析查询类别，选择相应的 Merge 节点，实现针对性的查询。查询周期对比见表 1。

改进模型虽然增加了步骤 2 和步骤 6 两步，但下文表 2 数据表明，整体上不影响系统的性能提升。

表 1 两种模型的查询生命周期

步骤	传统模型	改进模型
步骤 1(①→②)	PF 实现查询扩展、推荐等，进行 *query* 重组	
步骤 2(②→③)	无	Agg 进程分析 query 类别，选择 Merge 节点
步骤 3(③→④)	接收 PF 层多次向索引节点广播发送请求	Merge 节点根据查询向本类索引节点发送请求
步骤 4(④→⑤)	所有节点查询倒排表，相关性、堆排	该 query 相关的类别索引节点实现倒排表查询
步骤 5(⑤→⑥)	所有节点的结果合并、topK 排序	所属类节点的结果 topK 排序，发送给 Agg 进程
步骤 6(⑥→⑦)	无	Agg 进程融合多次查询结果
步骤 7(⑦→⑧)	页面组织、融合、排版，配置静态信息、图片链接等最终网页形式返回给用户	

设 M 个 Merge 节点（这里认为每个 Merge 节点管理一类查询），n 个独立引擎 $\{S_1, S_2, \cdots, S_n\}$ 节点（$M << n$），索引类别数为 m。对于传统模型，Merge 接收到 PF 层发送的查询请求，并转发到 SE 层的 n 个节点，从每个节点取得前 k 个结果，得 $n \times k$ 个结果然后进行融合、排序得 top$k$ 个结果返回给用户，即

$$query \rightarrow \sum_{i=1}^{n} \sum_{j=0}^{k} r_{i,j} \tag{8}$$

当 PF 层发生 *Times* 次用户查询，SE 层的每个节点也将产生 *Times* 次查询。对于改进模型，设每类索引分配到 n/m 个索引节点，Merge 的每个节点管理 m/M 个类别索引节点，即

$$query \rightarrow \sum_{i=1}^{(n/m)*(m/M)} \sum_{j=0}^{k} r_{i,j} = \sum_{i=1}^{(n/M)} \sum_{j=0}^{k} r_{i,j} \tag{9}$$

用户随机访问各个主题索引时，SE 的每类索引节点中的每个索引节点产生 *Times*/m 次查询。

分析 Merge 层的结果融合工作和 SE 层每个节点的查询量可得，分类索引和 Merge 层分类查找使得 SE 查找范围降低到原来的 1/m，Merge 融合工作量降低到原来 1/M。改进模型的分类索引使得 Merge 层的节点能够进行针对性查找，大大降低了 Merge 节点和索引节点的查询负载。同时，Agg 服务利用进程队列识别用户查询类别和结果融合工作，实行异步处理，最大化地减小 Agg 层负载。

从以上的分析可得，相对于传统模型，分类的索引结构理论上能够减少 SE 层负载和 Merge 节点的

查询开销,并且在扩展性方面,由于各类索引查询和结果融合相互独立、Agg 队列进程异步执行,使得改进模型具有良好的扩展性。

4 实验与分析

硬件资源:20 台 IBMX360f 服务器,8 核处理器、16G 内存。

软件资源:基于 lucene[21] + hadoop[22] 分布式引擎软件服务器平台,构建图 1 传统 Web 引擎架构(简称 Search A)和图 2 改进引擎架构(简称 Search B)。

(1) 搭建搜索引擎系统:搭建 20 个节点的引擎系统。

① Search A:18 节点的 SE 层引擎服务器,2 个节点作为 Merge 层的服务器。

② Search B: SE 层 16 个节点,Merge 层 2 个节点,Agg 层 2 个节点。

(2) 实验数据是借助 Hadoop 集群抓取的 Web 数据。

(3) 提取日志文件中用户的查询,构建实验中 query,作为搜索引擎的查询数据。

Search A 上没有中间的 Agg 层,索引的建立没有按照分类处理,与 Search B 相比,二者在查询方式上具有较大不同。实验分别从搜索引擎的"平均查询时间"和"系统吞吐量"对比二者的性能。

4.1 平均查找时间

Search A 直接由 PF 层和 Merge 层交互,缺少表 1中的步骤 2 和步骤 6 两个阶段,比较分析时,认为这两个阶段时间为 0。

文档索引量约 32GB,构建 18674 个 query 请求。为保证系统的开销可比性,系统节点数相同,使得两种模型的 SE 节点数不同。Search A 由于采用 18 个节点,按文档划分每个引擎得到数据近似1.78GB;Search B 采用 16 个节点。根据表 1 得知 *query* 查询周期分 7 步,分别对应每个阶段的平均查找时间。系统内部由于查询依赖性,一次用户请求会得到对索引节点多次请求。实验中的多核处理比较是指在索引节点层,Search A 采用多线程的查询间并行查找倒排表;Search B 在查询间和查询内都实现并行处理。图 6 中对比了二者查询过程,给出了每步的开销。

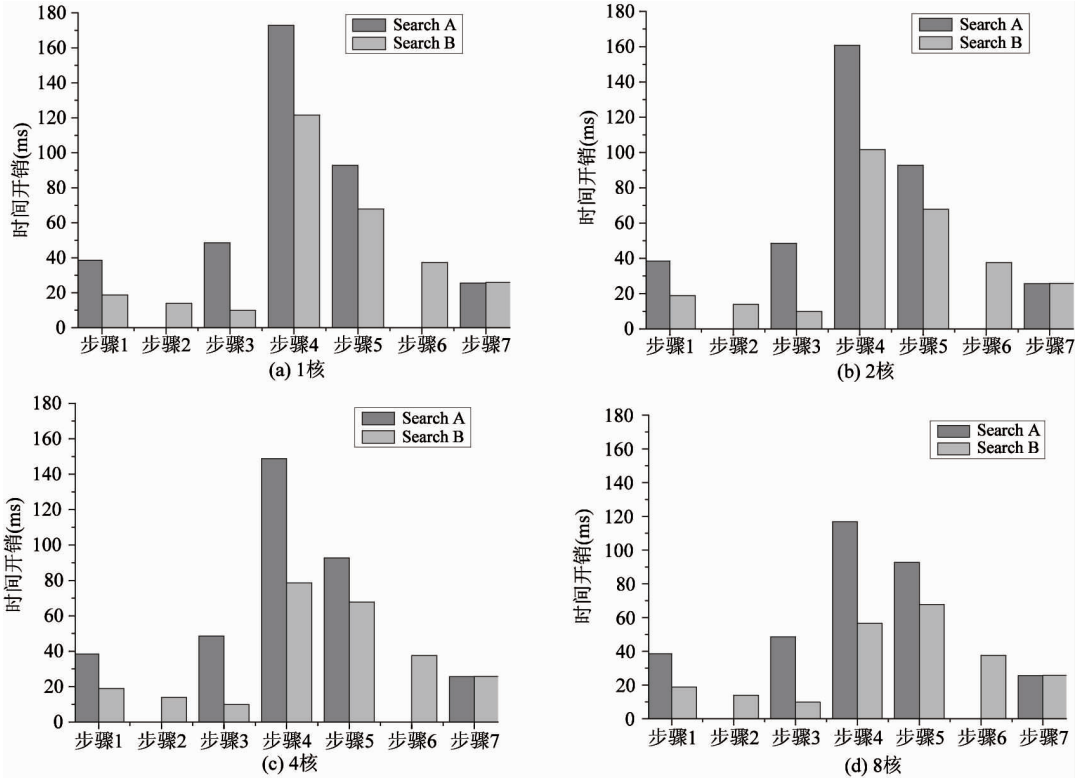

图 6 Search A 和 Search B 的 query 各个阶段平均查询时间对比(SE 层(步骤 4)使用不同数目核处理)

步骤 1:Search A 需要解析用户 query,存在多次收发问题;Search B 移交了这一阶段的解析 query 任务到 Agg 上,从而使时间开销缩减为原来一半。

步骤 2:Search B 特有的步骤,负责对 query 类

别分析,选择相应的 Merge 节点,平均耗时 13ms。

步骤 3:Search A 存在多次发送请求过程,平均延时是改进引擎的近似 5 倍。

步骤 4:(并行查找优化)随着节点核数变化,Search A 在该阶段耗时变化不大,而 Search B 充分利用多核对倒排表各个部分词项查找任务,实现查询内的并行,耗时变化较大。

步骤 5:Merge 层节点收集索引节点的范围不一样,Search A 中汇聚 SE 层的所有节点,收集的数据量多,需搜集完所有索引节点数据再进行堆排,取 topN 结果拼接,系统时间开销长;而 Search B 中 Merge 节点只需要收集本类索引节点的数据(在划分索引时确定),进行堆排、取 topN 结果拼接,系统开销时间小。

步骤 6:Agg 需要对 Merge 发送的结果进行合并,平均延时在 40ms 左右,而 Search A 没有这一步,任务耗时为 0ms。

步骤 7:都经 PF 解析 XML 结构,耗时在约 25ms。

改进的模型中的步骤 2 和步骤 6 都是通过缓冲池进行数据交互(没有通信延时),从整体上看,速度比传统模型提高 20% 以上(如表 2),时间上降低 80ms 以上。

表 2 两种模型平均查找时间对比

整体时间ms 模型 / 步骤4 核数	传统模型	改进模型	时间降低百分比
1	378.4	295.3	21.9%
2	366.3	275.5	24.8%
4	331.3	242.6	26.8%
8	319.6	230.5	27.9%

另外根据表 2 中 Search A 和 Search B 的平均查找时间,在使用单核情况下,Search B 平均查询时间降低 20%,并且随着 SE 层的节点核数增多,Search B 比 Search A 耗时降低更快。从而表明 Search B 运用于多核查询时,其优势更加明显。

4.2 吞吐量 QPS(query per second)

系统在一定时间内处理任务数与系统并发性能有关,尤其在大规模分布式引擎中,要求系统能够同时处理多个请求。通过实验分析 Search A 和 Search B 的吞吐量,采用不同的数据集分别测试它们的吞吐量,如图 7,为了保证二者实验的可对比性,按同样大小的文档数据量建立索引。实验中根据索引节点多核使用情况比较不同的索引量下的系统吞吐量。实验中采用了6组数据,分别是0.5 * 16GB、

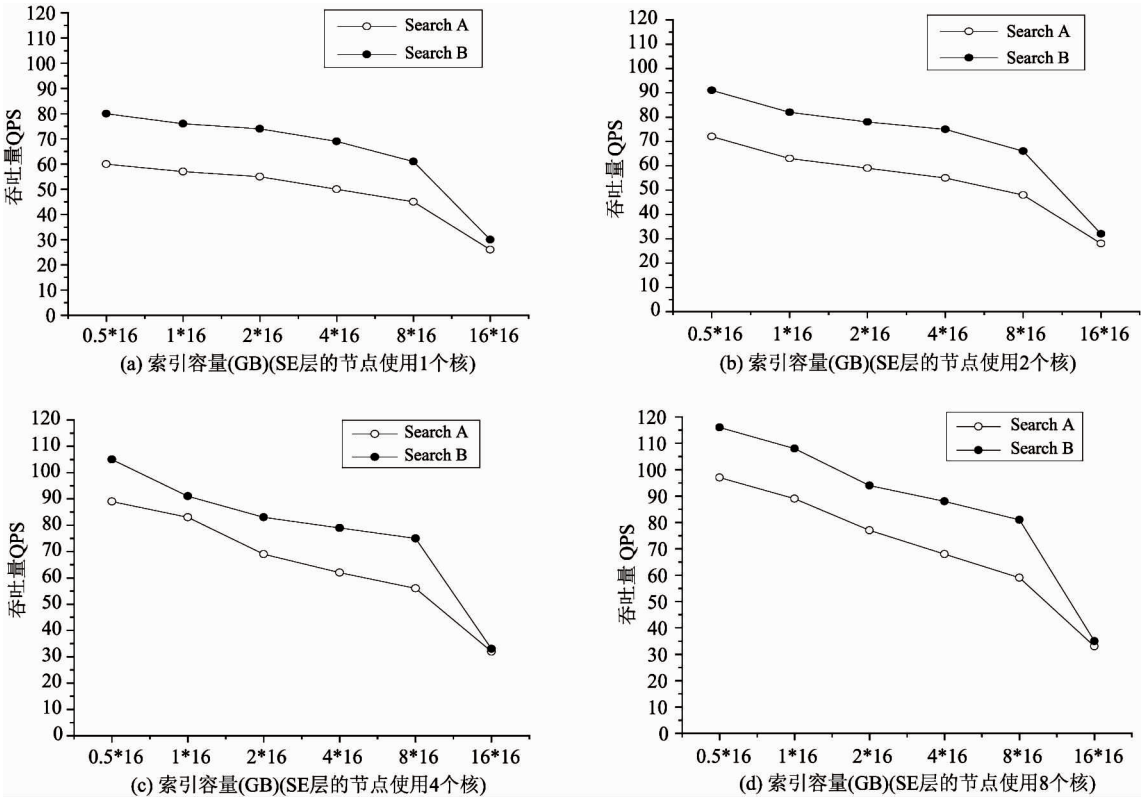

图 7 传统分布式模型和改进的模型吞吐量对比:SE 层,即在步骤 4 中,对各个引擎分别启动(a)单核;(b)双核;(c)四核;(d)八核查找倒排表

1 * 16GB、2 * 16GB、4 * 16GB、8 * 16GB、16 * 16GB,每组数据按主题分为“新闻”、“娱乐”和“体育”三类,为了实验中系统负载均衡,三类数据量基本平衡。实验中的机器物理内存为16GB,显然每个机器可使用的内存小于16GB。

改进的架构由于分类建立索引,每个引擎的通信小于传统架构的每个引擎,从而提高了系统吞吐量。如图7(a),单核情况8GB(0.5 * 16GB)索引量下,系统吞吐量提高近似25%,随着索引量增大,吞吐量会逐渐变少。在多核情况下,两种模型的吞吐量都增大,但随着文档数量变大,Search A 下降趋势要比 Search B 稍快,这是由于 Search B 采用分类索引,Merge 服务并行查找。

从图7还可以看出,随着文档数据量的增大,系统的吞吐量逐渐降低,当索引量接近或大于内存可使用量时,各个子图的吞吐量都几乎达到一个饱和点。这是因为当索引量大于内存使用量时,部分索引会保存在磁盘,存在磁盘访问,也即此时不是CPU 处理瓶颈,而是 I/O 瓶颈问题。I/O 的延迟使得系统吞吐量下降的幅度增大,即使 SE 层采用多核查找,也会存在这样问题。

5 结 论

针对传统 Web 引擎的扩展性问题,从两个方面对传统模型进行优化:(1) 提出索引分类方法,按照 Web 文档主题分类,建立分类索引节点集;对于同类索引节点集,如果单个索引节点空间不够,再按照相似 URL 进行分类,使得同类文档分配到同一索引节点。(2) 为了实现针对性查询,利用 Agg 进程队列实现异步处理查询;Merge 节点按照查询词的类别查找对应节点;充分利用多核技术,实现索引节点在查询间和查询内并行查询。通过实验验证了改进的检索结构能够降低平均查询时间,提高系统吞吐量,极大减少每个查询的检索对象,使得系统具有良好的扩展性。

本文主要从结构上改进检索过程,没有涉及分类索引节点的负载均衡问题、高效查询算法、索引剪枝等方面,下一步工作将要在这些方面展开。

参考文献

[1] Barroso L A, Dean J, Holzle U. Web search for a planet: The Google cluster architectur. *IEEE Micro*, *V.*, 2003, 23(2): 22-28

[2] Puppin D, Silvestri F, Perego R, et al. Load-balancing and caching for collection selection architectures. In: Proceedings of the 2nd international conference on Scalable information systems. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2007. 2

[3] 李晓明,闫宏飞,王继民. 搜索引擎——原理、技术与系统. 第二版. 北京:科学出版社,2012.5 15-17

[4] Anh V N, Moffat A. Structured index organizations for high-throughput text querying. In: String Processing and Information Retrieval, Springer Berlin Heidelberg, 2006. 304-315

[5] Clarke C L A, Terra E L. Approximating the top-m passages in a parallel question answering system. In: Proceedings of the thirteenth ACM international conference on Information and knowledge management. ACM, 2004. 454-462

[6] Puppin D, Silvestri F, Laforenza D. Query-driven document partitioning and collection selection. In: Proceedings of the 1st international conference on Scalable information systems. ACM, 2006. 34-41

[7] Moffat A, Webber W, Zobel J, et al. A pipelined architecture for distributed text query evaluation. *Information Retrieval*, 2007, 10(3): 205-231

[8] Shan D, Ding S, He J, et al. Optimized top-k processing with global page scores on block-max indexes. In: Proceedings of the fifth ACM international conference on Web search and data mining. ACM, 2012. 423-432

[9] Anh V N, Moffat A. Pruned query evaluation using pre-computed impact. In: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval. ACM, 2006. 372-379

[10] Büttcher S B, Clarke C L A, Cormack G V. Information Retrieval: Implementing and Evaluating Search Engines. MIT Press, 2010. 488-505

[11] Marin M, Gil-Costa V. High-performance distributed inverted files. In: Proceedings of the sixteenth ACM conference on Conference on information and knowledge management. ACM, 2007. 935-938

[12] Leibert F, Mannix J, Lin J, et al. Automatic management of partitioned, replicated search services. In: Proceedings of the 2nd ACM Symposium on Cloud Computing. ACM, 2011:27

[13] 何峰,丁晓青. 结合文本聚类和文本检索的语料选取方法. 高技术通讯, 2010,(12):1224-1228

[14] 何靖,李晓明. 搜索引擎效果评测——基于用户点击日志分析的方法与技术. 北京:高等教育出版社, 2012. 93-126

[15] 靳岩钦,张敏,刘奕群等. 搜索引擎用户查询的广告点击意图分析. 哈尔滨工业大学学报, 2013, 45(1): 124-128

[16] Ding S, Suel T. Faster top-k document retrieval using block-max indexes. In: Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval. ACM, 2011. 993-1002.

[17] Grossman, David A. Information retrieval: Algorithms and heuristics. Vol. 15. Springer, (Second Edition) 2010. 280-285

[18] Tsegay Y, Turpin A, Zobel J. Dynamic index pruning for effective caching. In: Proceedings of the sixteenth ACM conference on Conference on information and knowledge management. ACM, 2007. 987-990

[19] Altingovde I S, Ozcan R, Ulusoy Ö. Static index pruning in web search engines: Combining term and document popularities with query views. *ACM Transactions on Information Systems* (*TOIS*), 2012, 30(1): 2

[20] 易明. 基于 Web 挖掘的个性化信息推荐. 北京:科学出版社, 2010. 93-110

[21] http://hadoop. apache. org

[22] http://lucene. apache. org

A method of retrieval structure optimization for Web search engines

Qian Libing, Ji Zhenzhou

(School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001)

Abstract

In order to enhance the retrieval performance and scalability of search services, an index classification partitioning method was proposed. This method was used to improve the index structure so as to avoid the redundancy search and accelerate the parallel search process. By classifying the indexing documents based on the document theme and similar URLs, and categorizing the indexing nodes according to lexical terms, the parallelism of the indexing nodes was implemented in inter-query and intra-query. Based on the structure of the partitioned index, the searching logic was improved in a system, and a process queue structure—Aggregator (Agg) was designed, so the high-concurrence search in an asynchronous processing was realized. The experimental results show that the improved Web engine structure can reduce the inquiry overhead and improve the system throughput. Compared with traditional Web search models, the searching speed and throughput of the proposed model were improved respectively by 20% and 25%.

Key words: Web search engine, distributed search, retrieval structure, throughput, classified index