一种改进的分布式搜索引擎模型

钱立兵，季振洲，吴 昊

（哈尔滨工业大学 计算机科学与技术学院，150001 哈尔滨）

摘 要：为了解决传统分布式搜索引擎存在的搜索性能问题，从索引结构、查询算法方面改进了传统模型.提出了一种非集中的高并行化搜索模型，该模型按照文档主题对索引分类，对较长的倒排记录表采用位图结构，利用多线程技术对索引节点实现并行搜索算法（multi max score heap，MMSH）.实验结果表明：改进模型中的索引分类方法与倒排表结构的位图策略，能够增强 Merge 层查询的针对性，降低 Merge 层节点的 CPU 和内存开销；在倒排表不能完全存入内存情况下，MMSH 算法能够实现高度并行化查询，其查询效率高于经典的 term-at-a-time 算法，缩短了平均查找时间，提高了系统吞吐量.索引分类、位图结构以及并行查询算法能够避免查询的盲目性，改善了分布式搜索引擎的性能.

关键词：分布式引擎；索引分类；倒排结构；并行搜索

中图分类号：TP393 **文献标志码**：A **文章编号**：0367-6234(2014)07-0008-06

An improved model of distributed search engine

QIAN Libing, JI Zhenzhou, WU Hao

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

Abstract: To solve the problem of search performance in traditional distributed search engine, a non-centralized high parallelization search model was proposed and the traditional model was improved in the index structure and search algorithm. In the model, the index was classified according to document theme, bitmap structure was employed for longer inverted record list, and parallel search algorithm (multi max score heap, MMSH) was achieved in index node by using multi-threading technology. Experimental results show that the improved search model with index classification and bitmap strategy of the inverted list structure can enhance the search pertinence in Merge layer, reduce CPU and memory cost. In the case that the inverted list can not be completely stored in memory, MMSH algorithm can implement highly parallel search and its query efficiency is higher than the classical term-at-a-time algorithm, which shortens the average search time and improves the system throughput. Index classification, bitmap structure and parallel query algorithm can avoid query blindness and improve the performance of distributed search engines.

Keywords: distributed indexing; index classification; inverted structure; parallel search

大规模文档数据使得分布式引擎的检索和查询都面临性能的压力，为了提高分布式搜索模型的性能，在索引合并和查询算法方面，很多学者开展了研究工作.针对索引合并，Heinz 等[1]讨论了合并的索引构建方法及其相对于排序方法的优势；Büttcher 等[2]研究了 Logarichmic 合并算法并验证该算法对于索引合并的高效性；Hao 等[3]提出对索引压缩和查询预处理来优化文档排序.关于查询算法，document-at-a-time（DAAT）和 term-at-a-time（TAAT）是两种经典的算法[4-5]；Turtle 等[6]根据词项的最大得分策略（max score）优化了查询过程；Strohman 等[7]使用预处理剪裁前 k 个最好结果的得分下界改进查询效率.然而，这些改进均建立在传统模型基础上，并没有改变传统模型自身的一些局限，如宏观上索引文档结构划分的无序性，用户全局性查询的盲目性以及系统扩展性不强等.

收稿日期：2013-09-25.
基金项目：国家自然科学基金资助项目（61173024）；广东省部产学研结合基金资助项目（2011A090200037）.
作者简介：钱立兵（1986—），男，博士研究生；
季振洲（1965—），男，教授，博士生导师.
通信作者：钱立兵，qiannu159@163.com.

本文优化了传统分布式搜索模型的结构，按照文档主题对索引文档进行分类，提出高效、可扩展的并行分布式模型结构；以优化模型为基础，改进索引倒排表结构，并利用 MaxScore 策略，对堆排序进行优化，设计出针对索引节点的并行查询算法.

1　传统分布式搜索引擎

1.1　分布式搜索模型

分布式搜索引擎有沙漏模型、Map/Reduce 模型、P2P 网络模型等，其中，沙漏模型应用最为广泛. 图 1 是经典的沙漏模型[8]，系统包含前端应用展现层（processing of front，PF）、结果汇聚层（merge）和独立引擎层（search engine，SE）.

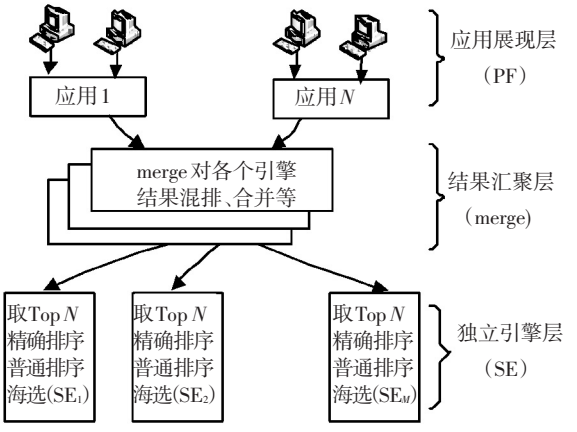

图 1　分布式引擎的沙漏模型结构

PF 层主要处理不同业务需求，每一类应用都会根据该应用特征解析 query，然后访问 merge 层.merge 层起到承上启下作用，接受 PF 层传递的多个请求，向 SE 层多个独立引擎发送；合并各个引擎请求结果后发送给 PF 层.SE 层是分布式引擎系统中的基础层，承担海选、相关性、精选等过程，是系统搜索核心过程.

1.2　分布式查询过程

PF 层接收用户请求（query），经过词干提取、用户行为、查询意图[10-11]分析得到若干个特征词 $\{q_1, q_2, \cdots, q_m\}$，再由 merge 层将这些特征词转发到 SE 层的各个独立引擎.每个独立引擎根据特征词对其倒排表进行海选、普通排序、精排，堆排序得到各自 Top N 结果.merge 层根据 SE 层的各个独立引擎所得到结果集，进行除重、过滤、相关性排序得到最终 Top N 个结果，返回用户.若独立引擎数为 n，特征词数目为 m，则用户一次请求在 merge 层会产生独立引擎查询次数 $m * n$.

这种分布式查询过程使得整个系统处于繁忙状态.高并发用户请求，使得 merge 层的请求任务与结果合并任务成倍增加；同时索引节点的增加，也会导致 merge 层增加负载任务.因此，如何提高系统扩展性，适应高并发查询是搜索引擎研究的关键目标之一.

2　并行分布式搜索

2.1　并行模型提出

为了克服沙漏模型的不足，本文根据文档主题对文档集进行分类，建立分类索引.文档主题的分类方法是根据用户的请求，提取用户查询特征词，以特征词代表文档，按照二元分类方法的 NN 算法[12]，计算各种类别模式下特征词的得分，与最低阀值比较，保留大于最低阀值的类别作为分类结果.特征词的提取按照文献[13]的词项权重为

$$w_{t,d} = \frac{tf_{t,d} \times \lg(N/df_t + 0.01)}{\sqrt{\sum_{t \in d} [tf_{t,d} \times \lg(N/df_t + 0.01)]^2}}.$$

式中：$w_{t,d}$ 为词 t 在文档 d 中权重；$tf_{t,d}$ 为词 t 在文档 d 中的词频；N 为文档总数；df_t 为出现 t 的所有文档数.

按照主题分类使得有些文档可能属于多个类别（跨类文档），对于这种跨类文档，可以采用简单的 URL 近似匹配，前提要对各个类别保存主体的 URL 前缀，具体做法是对跨类文档的 URL 分别匹配所属类别的 URL 集，保留 URL 匹配的最大文档，确定跨类文档最终只属于一个类别，按照单个类别建立索引.

图 2 是基于传统的分布式引擎所设计的可扩展的分布式模型.在 merge 层，按照文档主题建立 1～M 个类别的索引集，每个 merge 节点管理一类索引节点，例如 merge_1 节点管理索引 Index(1,1)～Index(1, c_1) 节点，使得查询请求所访问的索引节点具有针对性.

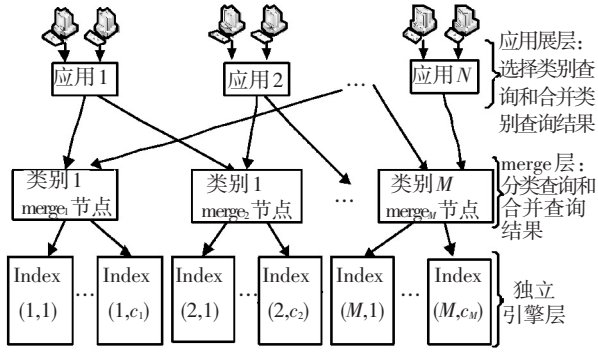

图 2　具有分类索引结构的分布式模型结构

依据图 2 建立的模型，分类查询过程如下：当应用层得到用户查询请求 query 后，经过分词得到查询关键词 Keyword，由 Keyword 确定待查询的 merge 节点，被触发的 merge 节点查询本类索引节点，查找的结果返回到 merge 层，merge 层合并本

类索引节点的结果返回给应用层.

该模型的查询类别识别方法为:首先,应用层需要保存 merge 层各个类别模式的特征词和近义词,匹配用户查询 Keyword;然后,应用层根据识别类别向所属的所有类别的 merge 层发送请求,被触发的 merge 节点再按照本节点的文档类别访问独立引擎层的索引节点.如图 2 所示,merge 层只是被动接收应用层发送的请求,识别用户过程由应用层负责,这样能够提早识别出类别,避免全局查找.

2.2 索引倒排表的改进

上述建立的并行模型,对于倒排记录经常发生变化的文档集合,需要更好的索引结构.索引合并是检索系统的重要任务.一种优化归并的方法是采用跳表(skip list)[14]策略能够获得很好的性能,即在构建索引时构建一个跳表,当跳表指针的目标项小于另一个表的当前比较项时可以采用跳表指针直接跳过.如图 3 所示,对 t_1 = "Information"和 t_2 = "Retrieval"进行合并,假如 t_1 的指针移到 17,t_2 的指针移到 65,较小项为 17.此时并不需要移动 t_1,而是检查跳表指针,此时为 30,仍然比 65 小,t_1 直接跳到 30,即跳过了 21 和 24.

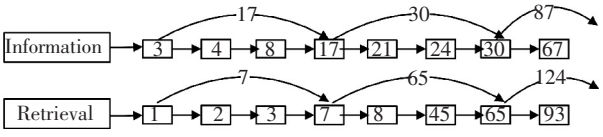

图 3 带有跳表指针的倒排记录表

倒排记录经常发生变化的文档,使得倒排记录表指针频繁发生变化,从而丧失了跳表的性能.为了解决跳表结构的性能问题,对于较长的倒排表的词项设计一个位图(Bitmap)结构(考虑 Bitmap 结构本身的内存开销,较短的记录表无须设计 Bitmap).Bitmap 结构的存储顺序号与文档 ID 对应,若文档存在倒排记录表中,对应位标记为 1,否则为 0. 例如:ID = 1 027 的文档记录在第 1 027 bit存储位,即存储在 Bitmap 结构的第 128 B 的第 3 位(1 027 = 128 * 8+3).

如图 4 所示,对 t_1 = "Information"、t_2 = "Retrieval"和 t_3 = "Organization",对较长倒排列表的词项 t_1 和 t_2 建立 Bitmap,t_3 链表长度较短无需建立 Bitmap,先对 t_1 和 t_2 的 Bitmap 合并保存到中间的 Bitmap merge 中,再遍历 t_3 得到记录表的文档号,添加到 Bitmap merge,即得到合并结果.

本文针对较长记录表的选择依据设计了两种方法:1) 采用固定常数 γ,记录大于 γ 时建立 Bitmap;2) 设计一个可调参数 $\lambda(0 < \lambda < 1)$,设文档集数目 N,$L(t)$ 为词项 t 的倒排表长度,当 $L(t) > \lambda N$ 时,建立 Bitmap.两种方法中 γ 或 λ 需要通过根据内存容量情况去选择.

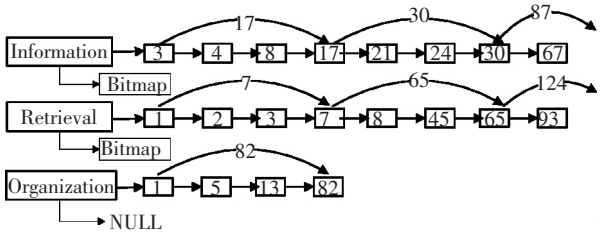

图 4 带有 Bitmap 结构的倒排记录表

对于 k 个长度为 L 的记录表合并,普通算法采用多路合并,时间复杂度为 $O(L * k * \lg k)$. 采用 Bitmap 策略时,满足条件的倒排记录表建立 Bitmap 的时间复杂度为 $O(k * t)$,即 k 个 Bitmap 按位运算时间开销(t 为两个 Bitmap 合并开销,通常为常量),提高了查询速度.空间上需要对每个较长记录表设计 N 位的 Bitmap,即建立 $O(r * N/8)$(r 为满足建立 Bitmap 条件的记录表个数).在查询过程中,对多个查询词的请求,先对词项的 Bitmap 合并(对于不存在 Bitmap 结构的词项,按照正常的遍历方式得到文档号信息,然后加入到 Bitmap 结构中).在动态更新文档时,只需要添加或删除 Bitmap 中对应的文档号位置.较少的内存空间开销所带来倒排表合并的速度优势很重要,对较长链表直接进行位图合并,能够较好地提高查询速度.在实际应用中,根据内存开销允许的程度去选择 γ 或 λ,对倒排记录表选择建立 Bitmap 结构,能够优化系统性能.

2.3 并行查询算法设计

针对文中设计的搜索模型,提出一种优化的并行查询算法.查询算法中的评分函数是依据 Robertson 等[15]使用的 Okapi BM25.

$$\mathrm{score}_{\mathrm{BM25}}(q,d) = \sum_{t \in q} \lg\left(\frac{N}{N_t}\right) \times \mathrm{TF}_{\mathrm{BM25}}(t,d), \quad (1)$$

$$\mathrm{TF}_{\mathrm{BM25}} = \frac{f_{t,d} \times (k_1 + 1)}{f_{t,d} + k_1 \times [b \times (l_d / l_{\mathrm{avg}}) + 1 - b]}. \quad (2)$$

式中:N 为文档集数量;N_t 为包含词项 t 的文档数;$f_{t,d}$ 为词项 t 在文档 d 中出现次数;k_1(默认是 1.2) 用来调节 TF 的饱和度;b(默认是 0.75) 用于归一化文档长度;l_d 为文档 d 词条数;l_{avg} 为文档集中所有文档平均词条数.

算法中设计 3 类数据结构:词项(term)堆用于管理查询词项 $\{t_1, t_2, \cdots, t_m\}$,追踪每个查词项项 t 的下一个文档;Bitmapmerge 结构(位图结构)用于存储倒排表合并后文档 ID 的存在状态;result 堆用于维护当前找到的 Top k 个搜索结果.

采用线程池(thread pool)分配和管理 3 类数据结构,每个线程都有独立的 3 种结构,线程初始化时构建 3 类数据结构,3 种结构都是随着当前用户的查询词项加入,查询结束后清空.

如图 5 所示,对于查询词集合 $query_{terms}$ = {"computer","science","technical"},启动线程池中的线程 1,建立 3 个元素的 term 堆,分配一个 Bitmap merge,同时分配含 Top k 的 result 堆(按得分建立小根堆).term 堆中每个词项连接的文档按照 ID 排序,result 堆在运行中保存当前最好的 k 个文档,根节点是当前第 k 个最好的文档,当找到一个新文档比根节点分数高时,利用新文档替换根节点,进而调整堆结构.

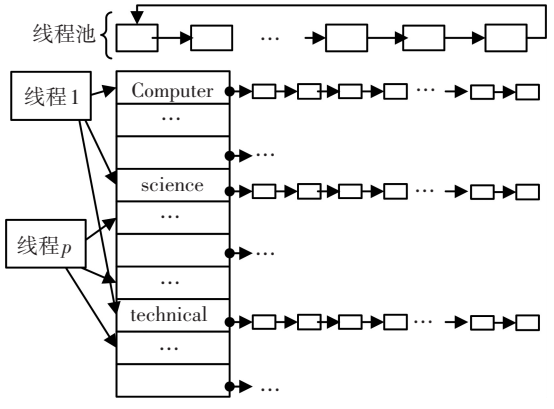

图 5　线程池结构与并行查询方法示例

根据 turtle 和 flood 的 maxscore 策略优化查询过程,式(2)中 TF 得分贡献不超过 $k_1 + 1 = 2.2$(k_1 取默认值 1.2),即词项总得分不超过 $2.2\lg(N/N_t)$.改进的查询算法采用了此种技术优化查询过程.

在支撑多核的硬件平台下,设计 thread pool 结构管理并行执行的线程.倒排记录表作为所有查询线程的公共访问资源,查找过程不会改变公共资源,因此对倒排列表的访问本身存在高度并发访问优点.由于线程本身开启和切换存在开销,尽可能使分配的线程任务饱满,而不是分配过多的任务量轻的线程.

对于搜索引擎业务,设请求任务并发数为 n/s,线程最大处理任务数为 C_{max},为了避免过多的轻量线程开销,要求每个线程至少处理任务数为 C_{min},线程池最多数目为 p(由硬件限制,一般为核数 2 倍最优),实际开启线程数 P_{num}. 当 $n \leqslant C_{min} * p$ 时,$P_{num} = n/C_{min}$,即开启 n/C_{min} 个线程;当 $C_{min} * p < n \leqslant C_{max} * p$ 时,开启的线程数 $P_{num} = 2 * p/(C_{min} + C_{max})$;当 $n > C_{max} * p$ 时,线程处于超负载情况,充分利用硬件资源,开启 $P_{num} = p$ 个线程.通过实验测得 $C_{min} = 2$,$C_{max} = 6$,即任务数小于 2 时,属于轻量线程;任务数大于 6 时,属于超负荷线程.

通过上述计算得到开启线程数 P_{num},得到如图 6 所示的 MMSH 算法,每次处理 n 个请求 queryList[1…n];每个线程平均处理任务数 n/P_{num}. 每个线程调用改进的堆排序裁减 MSH(max score heap)算法处理请求任务,如图 7 所示.

算法 1　MMSH 算法
输入:待处理 query 列表数组 queryList[n]
输出:n 个查询结果 results[n]
1 根据任务数 n,计算开启线程数 P_{num}
2 每个线程分配 n/P_{num} 任务,按照线程 ID 分配任务 queryList[ID * n/P_{num} + 1 ~ (ID + 1) * n/P_{num}]
3 平行执行 P_{num} 个线程,每个线程 n/P_{num} 个任务
4 每个线程 for i = 0 to n/P_{num} do
5　　解析 query[i] 得到特此词集 tokens,感兴趣 k
6　　获得合并倒排表运算 opt,如 AND、OR 操作
7　　调用 MSH 算法,得到 results[i]

图 6　对堆排序裁剪的一种多线程并行查询算法

算法 2　MSH 算法
输入:m 个词项 $< t_1, t_2, \cdots, t_m >$; Top k 为 k; 预先计算的最大得分 maxscore = Σ maxscore(t_i); 倒排表运算符 opt(如 AND 或 OR 运算); Bitmapmerge 保存中间倒排表合并结果
输出:查询结果 rusults
1 初始化 results 所有得分为 0,初始化 Bitmapmerge 结构
2 对 m 个词项建立 term 最小堆,堆的元素包含词项和词项在倒排表中的下一个文档号(nextDoc),在建堆过程中:
3　if 词项 t_i 存在 Bitmap 结构
4　　then t_i 的 Bitmap 与 Bitmapmerge 合并//按照 opt 运算
5　　else 遍历记录表获得文档号,填入 Bitmapmerge
6 按照 term 中的 nextDoc 升序调整 term 堆
7 while term 堆最小元素 nextDoc 存在 do
8　记录文档号 d_{ID} = term 堆最小元素 nextDoc,score = 0
9　while d_{ID} 存在 do
10　　if(result 堆最小元素得分> maxscore || $d_{ID} \notin$ Bitmapmerge)
11 break;
12　　按照式(1)对文档 d 算分,保存到 score
13　　更新 term 堆最小元素 nextDoc
14　　更新 term 堆
15　　if(score> result 堆最小元素得分)//更新堆
16　　result 堆最小元素得分,文档号
17　　调整 result 堆
18　　按照得分对 results 降序排序,并返回 rusults

图 7　MSH 算法

MMSH 算法的核心处理是 MSH 算法,如图 7 所示,通过对堆排序的剪裁,在保证结果一致性情况下,优化了查询性能.MSH 算法能够保证算法的结果与堆排结果一样,但速度要快很多,这是因为一旦发现某个文档不可能为前 k 个结果,立即被忽略.

3　实验与讨论

硬件资源:IBMX360f 服务器,16 核处理器、16 G 内存;通过商业引擎 Ha3+hadoop 技术,分别构建如图 1 的分布式引擎的沙漏模型(Search1)和如图 2 改进的分布式引擎(Search2)架构.通过比较改进模型和传统模型的查询算法效率和系统吞吐量,说明设计的有效性.

1)实验环境搭建. 为了实验条件的一致性,两种结构均使用 8 个节点服务器. Search1 和 Search2 都使用 4 个索引节点,2 个 merge 节点,2 个PF 节点.

2)实验数据. 借助于商业引擎的 Hadoop 集群抓取得到 50 G 文档数据;查询请求时根据查询日志,得到 1 万查询 query,作为 query 集.

3)实验方式. 通过脚本程序对 query 集定时请求系统,模拟用户访问,访问数由实验进行调节.

4)查询语法. 分类索引建立后,需要额外产生类别属性 category,在应用端识别出本次查询的 category,语法如:category = 3&&query = "MP3" &&hit = 10&&…,其含义为查询类别 3 的 merge 节点,查询词为 MP3,用户感兴趣的前 10 条文档信息等.

Search 1 的 merge 层采用集中方式,索引按照文档划分;Search2 进行分类和动态化处理,按照多个节点的非集中方式管理.由于索引结构的不同,导致二者查询方式很大差异,使得查询速度相差较大.同时,实验将要验证并行查询算法对高并发任务的有效性.

3.1 merge 层负载

为了验证改进的模型结构,在不同并发请求情况下,对 Search1 和 Search2 两种结构的 merge 层负载进行试验.图 8 和图 9 分别记录多个并发 query 的 merge 层节点的 CPU 和内存平均开销,实验中设置返回用户 Top 100 个结果,随着请求并发数增多,内存和 CPU 变化明显.CPU 开销与并发数几乎是线性增大,Search2 的 merge 节点 CPU 占用率约为 Search1 的 75%.merge 层节点的内存开销主要对所属类别索引节点的查询中间结果的缓存,并发任务越多,Search1 和 Search2 之间的内存开销相差越大,其原因是未分类 Search1 使得各个索引节点返回结果数以索引节点数的倍数增大,花费更多内存开销.

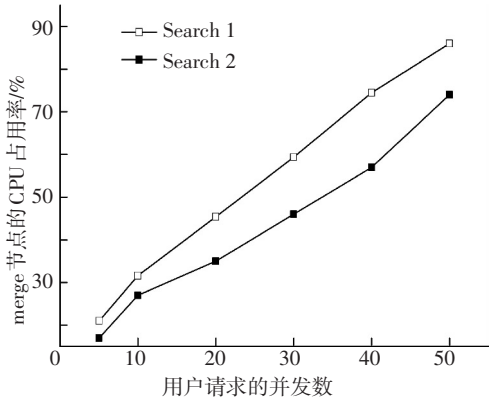

图 8 Search1 和 Search2 在不同并发数 query 的平均 CPU 占用率

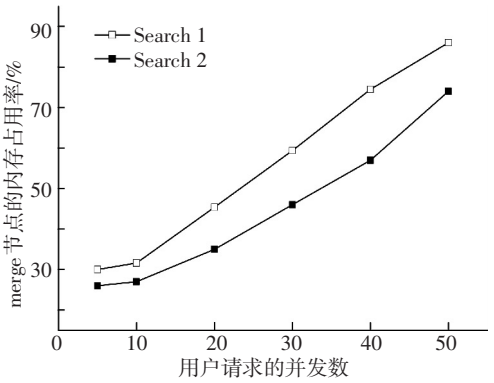

图 9 Search1 和 Search2 在不同并发数 query 的平均内存占用率

由于传统分布式模型对各个索引节点查询结果需要进行多次合并、排序等工作,使得 merge 成为系统的瓶颈,query 的并发数直接影响 merge 节点性能;而改进分布式模型根据索引类别,按照各类查询的 merge 节点处理查询信息,具有很强的针对性,从而减少 merge 节点的负载.

3.2 搜索节点的并行查询性能

为了验证 MMSH 算法优越性,实验中采用 Search2 模型.由于在倒排表索引不能完全存入内存情况下, TAAT 算法不需要磁盘访问,优势高于 DAAT 算法,而文中实验侧重讨论此种情况下 MMSH 算法优势,为此 MMSH 算法只与 TAAT 算法进行比较.在相同的并发用户请求和多线程执行下,从搜索引擎层的平均搜索时间和吞吐量两个方面比较二者性能.实验中两个重要参数是:线程数 p 和参数 Top k (用户选取前 k 个结果, k = 10 是系统默认值).

图 10 中,通过比较 MMSH 算法和 TAAT 算法的平均查找时间可以看出,MMSH 算法具有明显的优势,从图 10 可以看出,多线程体现并行查询的优越性,平均查找时间几乎随着线程数目增大,下降趋势明显.当线程数达到一定数量时,线程本身存在开销,如继续开启更多线程,会降低系统性能,如实验中开启 64 个线程效果没有 32 线程好.这是由于在搜索过程中,改进算法中利用 MaxScore 排除不必要计算过程,从而提高了查询速度.

图 11 的系统吞吐量是指整个搜索引擎层平均每秒钟所能够处理的 merge 层发送的请求数,而不是单个引擎的吞吐量.对于 merge 层发送的每一类请求,需要多个搜索节点并行搜索,然后把各种结果汇总到对应的 merge 节点.从图 9 可以看出,在线程数达到 32 前,多线程技术几乎能够得到线性的加速比;TAAT 算法在吞吐量方面落后于 MMSH 算法,体现了 MMSH 算法优越性.线程数为 32 时,MMSH 算法和 TAAT 算法的性能优

于 64 个线程数情况，这是由于过多的线程数导致线程开启和切换的开销较大.

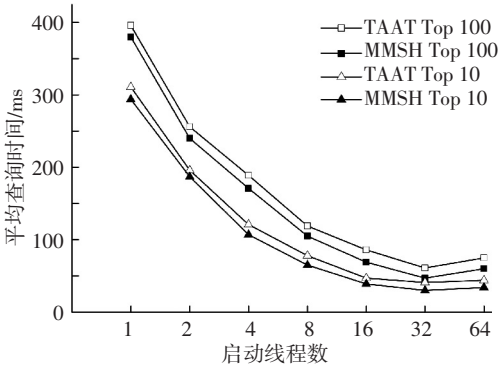

图 10　搜索引擎层使用 TAAT 算法和 MMSH 算法平均查询时间

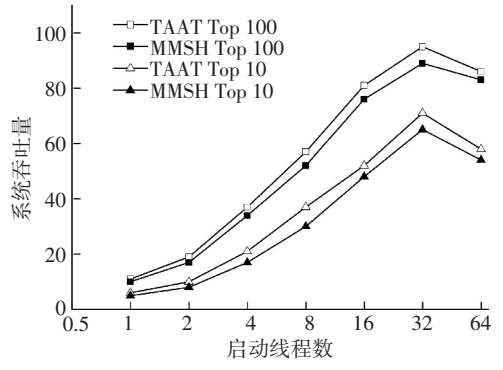

图 11　搜索引擎层使用 TAAT 算法和 MMSH 算法时系统吞吐量

从图 10、11 的 Top k，可以看出启动相同的线程数目，k 越大，查询时间越长，吞吐量越小，并且 MMSH 算法优于 TAAT 算法，即系统开销越大.这是因为用户 Top k 中的 k 选择，直接影响查询过程中待计算的文档得分数量，导致系统性能差异.由于 MMSH 算法使用 maxscore 的提前打分策略，同等条件下，可以提高系统性能.

4　结　论

1）根据文档主题特征，对索引进行分类，设计分类查询方法，解决查询盲目性.

2）对较长倒排记录表设计 Bitmap 结构，有效降低倒排表合并的查询时间复杂度，实现快速的倒排合并工作.

3）利用 maxscore 策略和多线程方法，在查询过程中建立词项堆、结果堆和 Bitmapmerge 结构，能够提前结束查询打分，从而优化了倒排表查询算法.

参考文献

[1] HEINZ S, ZOBEL J. Efficient single-pass index construction for text databases[J]. Journal of the American Society for Information Science and Technology, 2003, 54(8): 713-729.

[2] BÜTTCHER S, CLARKE C L A. Indexing time vs. Query time trade-offs in dynamic information retrieval systems [C]//Proceedings of the 14th ACM International Conference on Information and Knowledge Management. New York, NY: ACM, 2005: 317-318.

[3] HAO Y, SHUAI D, TORSTEN S. Inverted index compression and query processing with optimized document ordering [C]//Proceedings of the 18th International Conference on World Wide Web. New York, NY: ACM, 2009: 401-410.

[4] MANNING C D, RAGHAVAN P, SCHÜTZE H. An introduction to information retrieval [M]. New York, NY: Cambridge University Press, 2009:124-136.

[5] BÜTTCHER S, CLARKE C L A, CORMACK G V. Information retrieval implementing and evaluating search engine[M]. Cambridge, MA: MIT Press, 2010:137-173.

[6] TURTLE H, TOMPA F W. Query-evaluation: strategies and optimization [J]. Information Proceeding & Mangagement, 1995, 31(1):831-850.

[7] STROHMAN T, TURTLE H, CROFT W B. Optimization strategies for complex queries [C]//Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. New York, NY:ACM,2005: 219-225.

[8] BARROSO L A, DEAN J, HOLZLE U. Web search for a planet: the google cluster architecture [J]. IEEE Micro, 2003, 23 (2): 22-28.

[9] 王晓春,李生,杨沐昀,等. 查询会话中的用户行为分析[J]. 哈尔滨工业大学学报, 2011,43(5):76-78.

[10] 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[J]. 哈尔滨工业大学学报, 2013,45(1):124-128.

[11] 黎玲利, 王宏志, 高宏, 等. XML 数据流上 Top-K 关键字查询处理 [J]. 软件学报, 2012, 114(4): 1561-1577.

[12] 李晓明,闫宏飞,王继民.搜索引擎——原理、技术与系统[M]. 2 版. 北京:科学出版社, 2012:253-282.

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

[14] STROHMAN T, CROFT W B. Efficient document retrieval in main memory[C]//Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. New York, NY: ACM, 2007: 175-182.

[15] ROBERSTON S, ZARAGOZA H, TAYLOR M. Simple BM25 extension to multiple weight fields[C]//Proceedings of the Thirteenth ACM International Coference on Information and Knowledge Management. New York, NT: ACM, 2004:42-49.

（编辑　张　红）