基于时空局部性的层次化查询结果缓存机制*

朱亚东[1]，郭嘉丰[1]，兰艳艳[1]，俞晓明[1]，陈海强[2]，程学旗[1]

[1](中国科学院 计算技术研究所，北京 100190)

[2](中国信息安全评测中心，北京，100085)

A Hierarchical Search Result Caching Based on Temporal and Spatial Locality

Zhu YaDong[1], Guo JiaFeng[1], Lan YanYan[1], Yu XiaoMing[1], Chen Haiqiang[2], Cheng XueQi[1]

[1](Institute of Computing Technology, The Chinese Academy of Sciences, Beijing 100190, China)

[2](China Information Technology Security Evaluation Center，Beijing 100085，China)

+ Corresponding author: E-mail:zhuyadong@software.ict.ac.cn

Abstract: In a result cache, either document identifiers (docID cache) or the actual HTML pages (page cache) can be stored to accelerate the response speed. For a fixed memory size, the docID cache can achieve a higher hit ratio while the page cache can obtain higher response speed. This paper proposes a novel hierarchical result caching scheme based on temporal and spatial locality, in which the result cache is firstly split into two layers: a page cache and a docID cache. In our scheme, page cache will be the first choice for answering some queries, and then the docID cache. In terms of average query response time, the results show that the proposed approach achieves a substantial performance improvement than baseline method by 9% on average, and up to 11% in the best situation. Secondly, the scheme also designs an adaptive prefetching strategy based on docID cache. The experiments show that the proposed scheme combined with the prefetching strategy can lead to an additional performance improvement. And we finally build a complete and effective result caching scheme by capturing the temporal and spatial locality of user search behaviours.

Key words: Page Cache; DocID Cache; Query Response Time

摘　要: **查询**结果缓存可以对查询结果的文档标识符集合或者实际的返回页面进行缓存，以提高用户查询的响应速度，相应的缓存形式可以分别称之为标识符缓存或页面缓存。对于固定大小的内存，标识符缓存可以获得更高的命中率，而页面缓存可以达到更高的响应速度。本文根据用户查询访问的时间局部性和空间局部性，提出了一种新颖的基于时空局部性的层次化结果缓存机制。首先，该机制将固定大小的结果缓存划分为两层：页面缓存和标识符缓存。对于用户提交的查询，该机制会首先使用第一层的页面缓存进行应答，如果未能命中，则继续尝试使用第二层的标识符缓存。实验显示这种层次化的缓存机制较传统的仅依赖于单一缓存形式的机制，在平均查询响应时间上，取得了可观的性能提升：例如相对单纯的页面缓存，平均达到 9%，最好情况下达到 11%。其次，该机制在标识符缓存的基础上，设计了一种启发式的预取策略，对用户查询检索的空间局部性进行挖掘。实验显示，这种预取策略的融合，能进一步的促进检索系统性能的有效提升，从而最终建立起一套时空完备的，有效的结果缓存机制。

基金资助：国家 242 专项：2012F86 和 2013G130；支撑计划：2012BAH39B04；国家 973 项目：2012CB316303 和 2013CB329602；国家自然科学基金：61232010, 61003166 和 61203298；国家重点科技研发项目：2012BAH46B04 和 2012BAH39B02；

关键词: 页面缓存;标识符缓存;启发式预取

1 引言

为了应对繁重和高度动态变化的用户查询流量，各种优化技术已被现代搜索引擎采用。缓存技术是其中一种重要的优化手段，在搜索引擎系统的多个层面上进行了部署和利用，以降低系统负载。在系统的底层，有针对索引结构的倒排链缓存，用以减少与磁盘的 I/O 通信[6,7,8]；在搜索引擎的上层，有针对重复查询的结果缓存，以避免查询的重复执行，从而提高查询处理的能力[2,5,9,3,7,1,8,10]；此外，还可能有其他中间层次的缓存，如针对多个 term 倒排链交集的缓存[4]，定位缓存和 top-k 缓存等[11]。

本文主要针对查询结果的缓存展开研究。文献[5]第一个公开的提出，将结果缓存作为降低查询响应时间的一种手段。文章中，作者研究了查询日志的分布，并比较了几种时下的缓存策略。作者表明，静态缓存的性能普遍较差，而基于 LRU 或 LFU 的动态缓存可以获得较好的性能。文献[9]学习了查询日志局部性的几种形式，提出除了在服务器端进行结果的缓存外，还可以在用户端对查询的结果进行缓存。Lempel 和 Moran 的工作中[3]，提出了一种新的结果缓存策略，称之为 PDC(概率驱动缓存)策略。它是一种改进的预取缓存机制，用于处理其他结果页（第一页之后的结果页）的查询请求。Fagni 等人提出了一种混合结果缓存策略，称之为 SDC[1] (静态-动态缓存)。该策略维持了一个静态缓存和动态缓存。静态缓存存入了在过去较长时期内频繁出现的一些查询，动态缓存则通过一定的缓存策略如 LRU 等进行管理，存入的是近期频繁访问的查询结果。Baeza-Yates 等人也提出了一种与 SDC 较为类似的混合策略，其采用了双动态缓存的结构[10]。Gan 和 Suel 研究了基于权重的结果缓存，将查询的处理代价也作为缓存管理策略考虑的要素之一。此外，作者还提出了一种新的基于特征的缓存策略，文中的实验结果显示该策略取得了较大的性能提升[2]。比较有意思的是，最近 Altingovde 等人讨论了一种混合结果缓存的使用[13]，与本文的思想较为类似，但在很多方面又有着巨大的差异。首先，二者依赖的模型架构基础完全不一样，本文是建立在一种典型的商业分布式架构基础上（具体见第二章节的分析），并认为这种三层的层次性架构（memory-doc server-disk）是层次性缓存机制有效性的理论基础；其次，作者没有从理论上对自己文中的思想进行分析论证，只是简单的实验评估。而本文则从理论建模的角度，对本文提出的层次化缓存机制进行了充分有效的分析；最后，本文还在此基础上提出了一种基于标识符缓存的启发式预取策略，用以挖掘用户查询的空间局部性。这种预取策略的融合，很好地挖掘和拓展了层次化缓存机制的效用，从而进一步地促进整个系统性能的有效提升，最终建立起一套对时间局部性和空间局部性都有效捕捉地，完备的，查询结果缓存机制。

结果缓存主要有两种形式：页面缓存和标识符缓存。给定一个固定大小的内存，标识符缓存可以存储大量的缓存条目，这一数目远大于页面缓存[1]。标识符缓存中的缓存条目是一个有序的 (根据相关度排序) 整数标识符集合，通常只有几十字节大小 (在每页 10 个结果的标准形式下，如 10*8Byte)，而页面缓存条目是可以直接返回给用户的 HTML 页面，一般为几 KB 大小 (如 10*512Byte)。在页面缓存上的命中可以立即返回查询结果，而在标识符缓存上命中后，仍需要读取文档服务器以生成相应的 HTML 页面。所以，对于给定的内存资源，页面缓存机制可以到达较高的响应速度，而标识符缓存可以获取较高的命中率。以往的大多数工作都集中于页面缓存的应用或某一种单独形式的应用。

本文基于用户查询访问的时间局部性和空间局部性，提出了一种新颖的层次化缓存机制，这种缓存机制对页面缓存的高响应速度和标识符缓存的高命中率，进行了总体的均衡。对于有限的内存资源，我们对每一层应占用的内存大小进行了充分的讨论，以获得最优化的性能提升。然后，我们在不同缓存大小的情形下，对本文提出的结果缓存机制进行了讨论和评估。相对于单独的页面缓存和标识符缓存，我们的层次化缓存机制带来了更好的性能提升。同时本文还提出了一种基于标识符缓存的启发式预取机制，挖掘用户查询的空间局部性。实验显示这种预取机制的融合，能进一步的发掘层次化缓存机制的效用，带来更大的性能提升。

文章的剩余部分组织如下:第二章节描述了几种基本的缓存机制以及本文提出的层次化的结果缓存机制，并对每一种策略下的平均查询响应时间进行数学建模，然后比较分析；第三章节根据用户查询的空间局部性，结合标识符缓存的优势，提出了一种启发式预取的策略；第四章节对文章提出的缓存机制和预取策略进行了

实验评估；第五章节对文章进行了总结并简单阐述了将来的工作。

2 层次化的结果缓存机制

在该章节中，我们将详细论述本文提出的层次化的结果缓存机制。首先我们介绍了两种传统的结果缓存机制，然后对每一种策略下的查询的平均响应时间进行了分析和对比评估。

2.1 系统概述

一个典型的大规模网络搜索引擎系统如图 1 所示，它是由一定数目的检索节点组成的分布式架构。在这

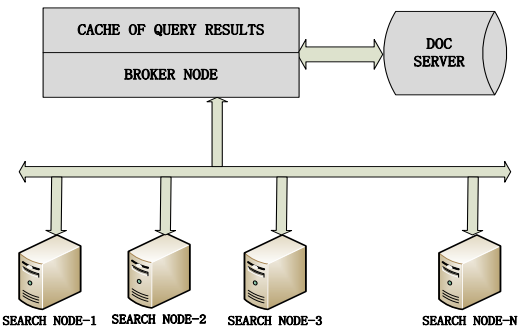

Figure 1: Architecture of a large-scale, distributed WSE with a result cache.

图 1：带有结果缓存的大规模分布式网络搜索引擎构架

些检索节点之上，有一个额外的代理节点。它负责将接受到的用户查询转发到相应的检索节点，同时收集返回的检索结果。然后代理节点根据他们的相关度的得分，对这些结果进行合并排序，从而产生一个有序的文档标识符集合。最后根据这些文档标识符，从文档服务器中获取 URL 和相关的文档摘要信息等，然后生成相应的 HTML 页面返回给用户[1]。检索结果的缓存通常部署在代理节点上面。

根据上面的处理流程，在没有结果缓存的情况下，查询的平均处理时间 t_1，可表述为：

$$t_1 = t_{rank} + t_{summary} + t_{generate} \qquad (0)$$

其中 t_{rank} 指查询的排序处理时间，这一过程包括索引倒排链的读取，计算排序等，最终提供了一个有序的文档标识符集合；t_{summay} 表示根据这些文档标识符从文档服务器中读取相关摘要内容花费的时间；$t_{generate}$ 表示根据摘要内容生成 HTML 页面耗费的时间。事实上，相对于 t_{rank} 和 t_{summay}，将摘要内容生成 HTML 页面耗费的时间非常少，$t_{generate}$ 可忽略不计。所以此时 t_1 可表示为：

$$t_1 = t_{rank} + t_{summary} \qquad (1)$$

2.2 页面缓存机制

大多数以往的工作主要集中在页面缓存的应用上，这里我们也采用该机制作为我们的评估基准(baseline)。这种机制下，查询的基本处理流程如图 2.a 所示。t_2 表示在页面缓存机制下，查询的平均处理时间:

$$t_2 = (1-a)*(t_{rank} + t_{summary} + t_{insert1}) + a * t_{read1} \qquad (2)$$

这里，a 表示页面缓存的命中率；$t_{insert1}$ 表示插入一个条目到页面缓存耗费的时间；t_{read1} 表示从页面缓存读取一个缓存条目耗费的时间；t_{rank} 和 t_{summay} 含义与公式(1)中的相同。

2.3 标识符缓存机制

结果缓存还可以采用标识符缓存的形式，这种机制下，基本的查询处理流程如图 2.b 所示。t_3 表示在标识符缓存机制下，查询的平均处理时间，并表示如下：

$$t_3=(1-b)*(t_{rank}+t_{summary}+t_{insert2}) \\ +b*(t_{read2}+t_{summary}) \quad (3)$$

其中，b 表示标识符缓存的命中率；$t_{insert2}$ 表示插入一个条目到标识符缓存耗费的时间；t_{read2} 表示从标识符缓存中读取一个条目耗费的时间；t_{rank} 和 t_{summay} 含义与公式(1)中的相同。

2.4 层次化的结果缓存机制

这种机制同时融合了页面缓存和标识符缓存，并以一个合适的比例对二者在结果缓存中的大小进行了划分。这种机制下，查询的处理流程如图 3 所示。t_4 表示在该机制下查询的平均响应时间，并表示如下：

$$t_4=(1-a)*((1-b)*(t_{rank}+t_{summary}+t_{insert2}) \\ +b*(t_{read2}+t_{summary}+t_{insert1}))+a*t_{read1} \quad (4)$$

其中，a 表示页面缓存的命中率；b 表示标识符缓存的命中率；其他参数的含义与公式(1),(2),(3)中相同。

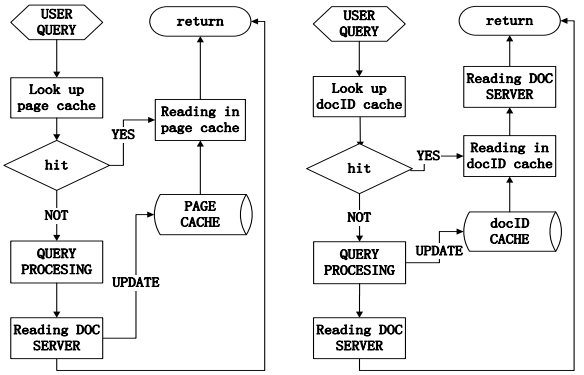

Figure 2: traditional result cache schemes: (a) Processing with page cache, (b) Processing with docID cache

图 2：传统的结果缓存机制: (a)页面缓存处理机制, (b)标识符缓存处理机制

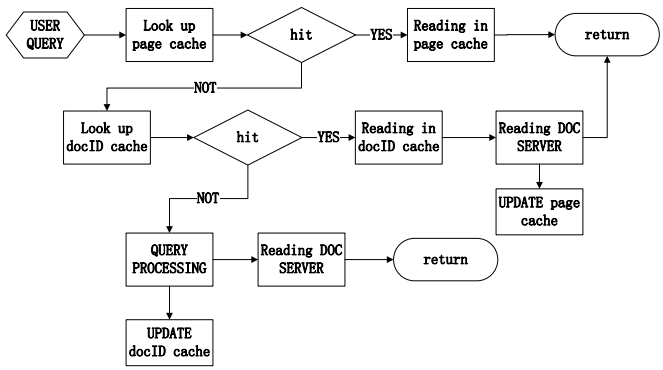

Figure 3: processing with proposed result cache scheme

图 3：层次化的结果缓存处理机制

2.5 时间代价分析

显然，t_1，t_2，t_3 和 t_4 分别表示在，没有结果缓存，页面结果缓存，标识符结果缓存，层次化结果缓存，四种情况下的平均响应时间。对于一个特定的网络搜索引擎系统，一些参数的值如：t_{rank}，t_{summay}，$t_{insert1}$，t_{read1}，$t_{insert2}$ 和 t_{read2} 都是可以直接测试出来，是一个较为稳定的常数值。在本文后续试验章节中，我们提供

了这些参数的估值（参见表格 1）。这里我们以本文的实验环境为例，将这些参数的估值代入公式(2)(3)(4)，

$$t_2 = 30.91(1-a)$$
$$t_3 = 30.91 - 30.5b$$
$$t_4 = (30.91 - 30.5b)(1-a)$$

页面缓存 VS 标识符缓存：我们首先对 t_2 和 t_3 进行对比分析：

$$t_2 - t_3 = 30.91(1-a) - (30.91 - 30.5b)$$
$$= 30.5b - 30.91a$$
$$t_2 - t_3 = \begin{cases} < 0, a > 0.9867b \\ \geq 0, a \leq 0.9867b \end{cases}$$

在本文的实验环境中，每一个页面缓存条目的大小是 5KB(10*512Byte)左右，每一个标识符缓存条目大小是 80B(10*8Byte)。对于固定大小的内存，标识符缓存提供的缓存条目数是页面缓存的 64 倍。所以在相同的缓存管理策略下，标识符缓存的命中率始终大于等于页面缓存的命中率，即 $a \leq b$ 总是成立的。

当 $a > 0.9867b$ 时，也即 $b \geq a > 0.9867b$，有 $t_2 < t_3$，即此时页面缓存机制能提供更低的查询响应时间。这种情况也意味着页面缓存拥有几乎与标识符缓存相同的命中率。显然，可以试想一种极端的情况，如果结果缓存足够的大，那么由标识符缓存机制带来的在缓存条目数量上的优势便会消失。所以对一个相对大容量的结果缓存，页面缓存机制由于其高响应速度，相对于标识符缓存，将会是一个更好的选择。另一方面，对于一个有限的小容量的缓存资源，标识符缓存带来的增益是巨大的，条件 $a \leq 0.9867b$ 很容易成立，此时，标识符缓存机制显然是一个更好的选择。

页面缓存 VS 层次化缓存： 对于 t_2 和 t_4，直接的理论上的比较是困难的。因为两个表达式中的参数 a 的值并不相同。由于页面缓存单个条目的大小是标识符缓存条目的 64 倍，因此，如果我们拿出页面缓存中的很小一部分用于标识符缓存，一方面对页面缓存的命中率几乎没有影响，另一方面由标识符缓存带来的性能增益却是非常可观的。例如，对于有 300K 个缓存条目的页面缓存，如果我们拿出 5K 个条目用于标识符缓存，带来的页面缓存命中率上的变化可忽略不计，即参数 a 没有变化 (实验显示，在相同缓存策略如 LRU 的情况下，页面缓存在 300K 个缓存条目和 295K 个缓存条目下的命中率几乎不变)，而对应的标识符缓存却能额外的提供 320K (5K*64)个缓存条目，显然，这极大提高了结果缓存的命中率。在这种情况下，层次化缓存机制较页面缓存带来的性能提升，可表示如下：

$$\frac{t_2 - t_4}{t_2} \approx \frac{30.5b}{30.91}$$

比如，$b = 10\%$，那么相应的性能提升可达 9.86%。

事实上，我们可以从局部性原理的角度对层次化缓存机制进行评估。如图 4 中所示，这种层次化缓存机制类似于计算机系统中的存储层次模型。最上面的一层拥有最快的速度和最高的成本，最下面的一层拥有最慢的速度和最低的价格。相对于页面缓存机制，层次化缓存机制增加了一个中间层次，这个层次拥有相对高的速度(文档服务器的读速率 $t_{summary}$)，但相对低廉的价格。这样，每一层都作为下面一层的高速缓存，充分利用了局部性原理，很好的融合了速度与成本的目标。

最优划分：对于层次化结果缓存机制，在结果缓存中，对页面缓存和标识符缓存各自层次大小的划分，存在一个均衡。找到一个最优划分使性能最大化是非常重要的。由上面 t_4 的表达式可以发现，t_4 的大小取决于 a 和 b（页面缓存和标识符缓存的命中率），而 a 和 b 除了跟每部分的缓存大小相关外，还跟缓存策略以及实际中具体的用户查询行为有关。所以，我们不能从理论上直接指出，使 t_4 最小化的精确划分。然而，基于上面章节的分析，我们可以获得一些有意义的建议：

- ✧ 对于一个有限的相对较小的结果缓存，我们倾向于将绝大部分的结果缓存空间划分给标识符缓存；
- ✧ 对于一个相对较大的结果缓存，我们倾向于拿出一个较小比例的缓存空间用作标识符缓存，而将大部分的检索结果缓存到页面缓存中；

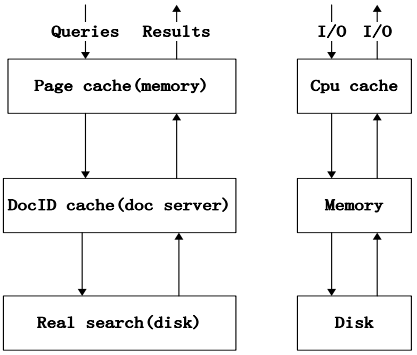

Figure 4: proposed result caching hierarchy comparing with storage hierarchy

图 4：层次化结果缓存与存储层次架构对比

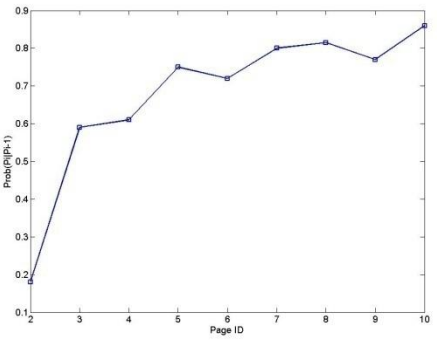

Figure 5: The probability of the occurrence of a request for the next result page

图 5：用户请求下一页的概率

3 基于标识符缓存的启发式预取策略

事实上，用户的查询行为除了具有时间局部性，还具有空间局部性。本章节在层次化缓存机制的基础上，进一步的挖掘用户查询的空间局部性，提出了一种基于标识符缓存的启发式的预取策略。

我们以部分 AOL 检索服务的查询日志为例（2006 年 3 月 1 号到 31 号），对查询日志中用户翻页的可能性进行统计，结果如图 5 所示。它表示的含义为，当给定同一个话题的第(*i-1*)页面的请求后，用户点击第 *i* 个页面的概率。由图中可知，当用户点击了第一页后，其点击第二页的概率只有 0.2 左右，而当用户请求了第 2 个页面后，其则有很大的可能性（概率大于 0.5）继续请求第 3 个页面，其他以此类推。事实上，通常绝大数的用户查询请求都不会超过前 3 个页面[15]。根据这一系列查询日志的统计特征，同时结合标识符缓存自身的优势：即能用较小的缓存空间提供大量的缓存条目，我们提出了一种基于标识符缓存的启发式预取策略，如表 2 表示。其中 query (keywords, m, n)是一个向后端检索平台提交的用户查询形式，查询内容为 keywords，m 为用户请求的结果页面号，n 表示需要在标识符缓存中缓存的，从 m 开始的连续 n 个结果页面。

整个预取策略是非常简单的：当第一个结果页面被请求并且不在任一层次的结果缓存中（首先在页面缓存中查找，其次是标识符缓存），我们会对查询进行扩展，然后向后端的检索平台请求第一和第二两个结果页面，并插入到标识符缓存中；当第二个结果页面被请求，它通常会立即返回给用户，然后向后端检索平台继续请求后续的 3 个页面，并插入到标识符缓存中。通过这种方式，我们仅对那些具有很大访问可能性的有限结果页面进行预取。这样我们在限制由预取带来的额外后端负载的同时，最大化了系统的响应速度。

总体上，我们对那些有较大访问可能性的，有限的结果页面进行了预取，同时利用了标识符缓存的空间优势，有效地压缩了由预取带来的额外空间开销，相对于传统的依赖于页面缓存的方法，显然具有更大的优势。这样在层次化缓存机制的基础上，融合启发式的预取策略，建立了一套完备有效的结果缓存机制。很好的促进 Web 检索性能的提升。

Table 2: An adaptive prefetching strategy based on docID cache

表 2：一种基于标识符缓存的启发式预取策略

n= requested page number	Hit/Miss	Cache Action
n=1	Hit	Return page 1
	Miss	Submit query(keywords, 1, 2) and cached in docID cache
n=2	Hit	Return page 2, submit query(keywords, 3, 3) and then cached in docID cache
	Miss	Submit query(keywords, 2, 3) and cached in docID cache
n>2	Hit	Return page n
	Miss	Submit query(keywords, n, 3) and cached in docID cache

4 实验评估

4.1 数据和实验设置

实验中，我们采用了 AOL 检索服务 2006 年 3 月 1 号到 31 号之间的查询日志。并从中随机选取了一部分的查询子集，包含了 7,172,919 条查询。我们采用了文献[2,11]应用的规则，对查询日志进行预处理，这其中包括去除停用词，移除只包含停用词的查询。在第一小节试验中，我们对那些对后续结果页面访问的查询请求（ItemRank>10）也予以移除 。处理后的日志集合包含 6,587,522 条查询，这里面包含 2,326,676 条不同的查询，其中 1,417,612 条查询仅仅出现过一次，399,657 条查询出现过两次。统计显示这些查询的频率准从 Zipf 分布[12,14,15,16]。实验中，这些查询日志按时间顺序执行，前 60%(3,952,513)用于缓存系统的预热，剩下的 40%(2,635,009)用于测量性能参数。我们在一个单独的机器上，采用一个优化版本的 Lucene 进行索引的建立和查询的处理。该机器采用 8 Intel(R) Xeon(R) CPU E5410@2.33GHz 以及 16GB 的 RAM。真实的应用场景中，一般会存在一个检索节点集群，比如拿 Katta[1]部署的分布式检索框架。这里我们采用的是单个检索节点的实验场景，一方面是出于简单高效的考虑；另一方面是因为，本文关注的是前端代理节点上的查询结果缓存的策略机制，对于后端检索节点，可以看成一个黑盒，而不关心具体的部署细节。另外，我们采用相同的两台机器搭建一个文档服务器，在上面部署一套 voldemort[2]，其中配置的 bdb 缓存是 4GB。我们索引了近 700 万文档，其中包含大约 150 万的 Wikipedia 网页。

表 1 展示了在第二章节中提到的相关参数的估值。由于页面缓存和标识符缓存都在一个内存中，所以 $t_{insert1}$ 和 t_{read1}，与 $t_{insert2}$ 和 t_{read2} 有相同的值，我们统一用 t_{insert} 和 t_{read} 代替。这些参数的估值可以作为参考，代入到第二章节中的时间代价公式中，以简化相关的理论分析。

Table 1: parameter evaluations (ms) in our experiments

表 1：实验中的参数估值(毫秒)

Parameter	t_{rank}	$t_{summary}$	t_{read}	t_{insert}
Evaluation	30.5	0.41	0.00073	0.00144

4.2 层次化缓存机制评估

图 6 展示了在三种缓存机制下的平均响应时间，其中横轴表示：页面缓存条目数（每一个页面缓存条目粒度大小是 5KB），纵轴表示：查询的响应时间（response time），单位毫秒（ms）。对于层次化的结果缓存机制，我们采用了固定的 50K 页面缓存条目用于标识符缓存。结果显示，较传统的页面缓存机制，层次化的缓存机制获得了平均 9%的性能提升。最好的地方甚至超过 11%，如在 300K 条目数的情形。在我们的实验中，每一个页面缓存条目的大小是 5KB(10*512B)，每一个标识符缓存条目的大小是 80B(10*8B)，一个页面缓存条目能提供 64 倍的标识符缓存条目。因此，对于标识符缓存机制，即便仅占据 20K 页面缓存条目数，其依然能提供高达 1280K(20K*64)个标识符缓存条目。显然，当标识符缓存的大小超过 30K 页面缓存条目，所有测试集中的查询都将被缓存，因此，之后的标识符缓存曲线呈水平不变的趋势。另外，我们发现当缓存大小小于 400K 条目时，相对于页面缓存，标识符缓存具有更好的性能。随着缓存大小的增加，由标识符缓存带来的性能增益逐渐减少，页面缓存和层次化缓存之间的间隔也逐渐减小。

图 8，图 9 对页面缓存和标识符缓存的最优划分进行了研究分析。我们分别采用了 200K 和 800K 两种大小的缓存资源，用以模拟有限的小缓存资源和较大的缓存资源两种情形。其中 x 轴表示用于标识符缓存的空间比例大小，y 轴是响应时间，单位毫秒（ms）。在图 7 中，当全部的缓存大小用于标识符缓存时，获得了最低的响应时间。这种情形下(即一个有限的较小的缓存空间)，相对于页面缓存，标识符缓存具有更好的查询性能，所以我们倾向于保留绝大数的缓存空间用于标识符缓存，这也与 2.5 节中的分析一致。在图 8 中，当

1 http://katta.sourceforge.net/

2 http://project-voldemort.com/

10%的缓存空间用于标识符缓存时，获得了最佳的性能。事实上，随着可用的缓存空间的增大，由标识符缓存带来的性能增益逐步减小。进一步的说，只有当标识符缓存占据较小的空间比例的时候，才能一方面将大部分的查询结果缓存在页面缓存中，很好的发挥页面缓存的性能优势；另一方面用也可以很好的利用标识符缓存的优势，即用较小的缓存空间提供大量的缓存条目，对二者的优势进行了有效地均衡。

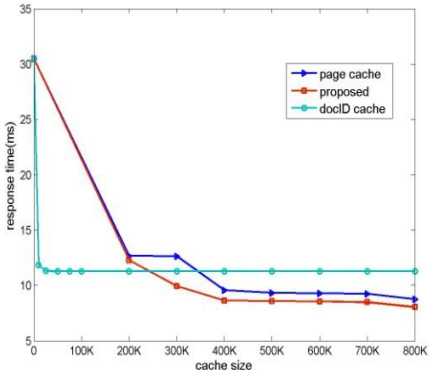

Figure 6: average response time comparison

图 6：平均响应时间对比

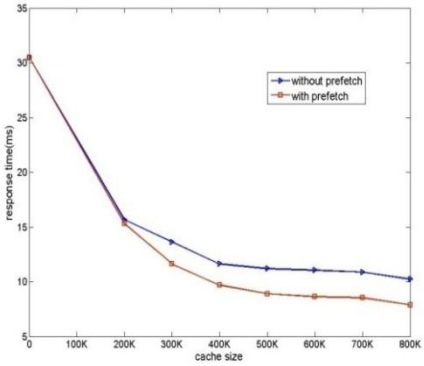

Figure 7: Performance comparison between with prefetching and without prefetching

图 7：是否融合预取机制下的性能对比

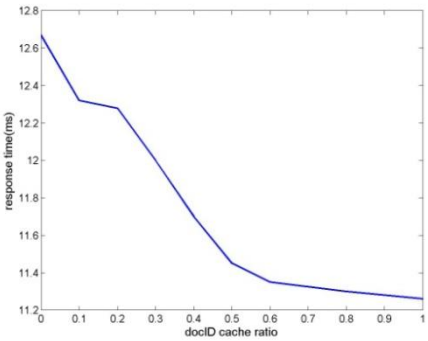

Figure 8: optimal division with 200k items

图 8: 200K 缓存条目数下的最优划分

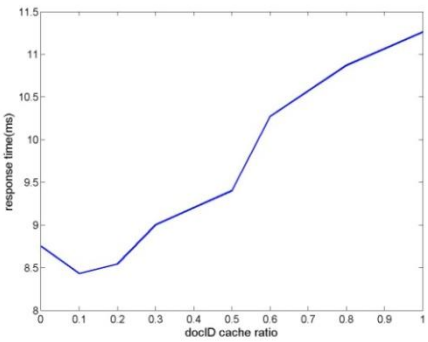

Figure 9: optimal division with 800k items

图 9: 800K 缓存条目数下的最优划分

4.3 基于标识符缓存的启发式预取策略的评估

大量的日志统计显示，通常的用户查询请求仅仅访问了结果页面的第一页[15]。在 4.1 和 4.2 章节的工作中，我们依照文献[2]的方法，将查询日志中对后续结果页面（ItemRank>10）的查询请求去除。这样，我们在不考虑查询预取的情况下，对本文提出的层次化缓存机制进行了有效地评估。事实上，用户的查询除了具有时间局部性，还具有空间局部性。我们对实验的日志数据进行了重新处理，将那些对后续结果页面的查询请求进行了保留。

然后我们对融合了预取策略的层次化缓存机制进行实验评估。实验结果如图 7 所示。显然融合了预取策略的层次化缓存机制，在查询响应时间上，能带来进一步的性能提升，并且随着缓存大小的增大，提升进一步增强并趋于稳定。总体上，我们对那些有较大访问可能性的，有限的结果页面进行了预取，同时利用了标识符缓存的空间优势，有效地压缩了由预取带来的额外空间开销，相对于传统的依赖于页面缓存的方法，显然具有更大的优势。

5 总结和将来的工作

结果缓存是搜索引擎中一种有效的优化手段，用于降低查询响应时间，提高系统吞吐率。对于有限的内存资源，本文提出的层次化缓存机制能获得很好的性能提升。同时在此基础上，我们还提出了一种基于标识符缓存的启发式预取策略，用以挖掘用户查询行为的空间局部性。实验显示，这种预取策略的融合，能进一步的促进整个系统的性能提升。至此，我们建立起了一套完备有效的结果缓存机制，很好的促进了 Web 检索系统的性能提升。在将来的工作中，我们会针对实时搜索引擎中（如新闻搜索，微博搜索）的缓存一致性问题进行进一步的研究[17,18,19,20]，使本文提出的层次化缓存机制具有更广泛的应用场景。

References:

[1] T. Fagni, R. Perego, F. Silvestri, and S. Orlando. Boosting the performance of Web search engines: Caching and prefetching query results by exploiting historical usage data. ACM TOIS, 24(1):51-78, 2006.

[2] Q. Gan, T. Suel. Improved techniques for result caching in Web search engines. In WWW, pp. 431-440, 2009.

[3] R. Lempel, and S. Moran. Predictive caching and prefetching of query results in search engines. In WWW, pp. 19-28, 2003.

[4] X. Long and T. Suel. Three-level caching for efficient query processing in large Web search engines. In WWW, pp. 257-266, 2005.

[5] E. Markatos. On caching search engine query results. Computer Communications, 24(7), 2000.

[6] P. Saraiva, E. de Moura, N. Ziviani, W. Meira, R. Fonseca, and B. Ribeiro-Neto. Rank-preserving two-level caching for scalable search engines. In SIGIR, pp. 51–58, Sept. 2001.

[7] R. Baeza-Yates and F. Saint-Jean. A three-level search-engine index based in query log distribution. In SPIRE, Sept. 2003.

[8] R. Baeza-Yates, A. Gionis, F. Junqueira, V. Murdock, V. Plachouras, and F. Silvestri. The impact of caching on search engines. In SIGIR, July 2007.

[9] Y. Xie and D. O'Hallaron. Locality in search engine queries and its implications for caching. In Proc. IEEE Infocom, 2002.

[10] R. Baeza-Yates, F. Junqueira, V. Plachouras, and H. Witschel. Admission policies for caches of search engine results. In SPIRE, Sept. 2007.

[11] Mauricio Marin, Veronica Gil-Costa, Carlos Gomez-Pantoja. New Caching Techniques for Web Search Engines. In HPDC'10, June 20-25, 2010.

[12] D. Puppin, F. Silvestri, R. Perego, and R. Baeza-Yates. Load-balancing and caching for collection selection architectures. In INFOSCALE, 2007.

[13] Altingovde, I., Ozcan, R., Second Chance: A Hybrid Approach for Dynamic Result Caching in Search Engines. ECIR'11, pp. 510-516, 2011.

[14] J. Wang, S. Shan, M. Lei, Z. Xie, and X. Li, "Web search engine: characteristics of user behaviors and their implication," Science in China, Series F, vol. 44, pp. 351--365, 2001.

[15] David J. Brenes *, Daniel Gayo-Avello. Stratified analysis of AOL query log. Information Sciences 179 (2009) 1844–1858.

[16] C. Silverstein, M. Henzinger, H. Marais, and M. Moricz. Analysis of a Very Large AltaVista Query Log. Technical Report 014, SRC (Digital, Palo Alto), Oct. 1998.

[17] Sadiye Alici, Altingovde, I., Ozcan, R., Time-based Result Cache Invalidation for Web Search Engines. SIGIR'11, July 2011.

[18] B. B. Cambazoglu, F. P. Junqueira, V. Plachouras, S. Banachowski, B. Cui, S. Lim, and B. Bridge. A refreshing perspective of search engine caching. In WWW, pages 181-190, 2010.

[19] R. Blanco, E. Bortnikov, F. Junqueira, R. Lempel, L. Telloli, and H. Zaragoza. Caching search engine results over incremental indices. In SIGIR, pages 82-89, 2010.

[20] S. Alici, I. S. Altingovde, R. Ozcan, B. B. Cambazoglu, and O. Ulusoy. Timestamp-based cache invalidation for search engines. In WWW, pages 3-4, 2011.

附中文参考文献：

[14] 王建勇、单松巍、雷鸣、谢正茂、李晓明，海量 web 搜索引擎系统中用户行为的分布特征及其启示，《中国科学》E 辑，2001 年 8 月，第 31 卷，第四期，372-384 页