P2P 网络搜索技术的研究*

贾杏丹, 张立臣
(广东工业大学 计算机学院, 广东 广州 510090)

摘　要: 分布式存储系统以其分布式控制、自组织性和普遍的适应性而受到越来越多的关注。搜索是所有存储系统的重要组成部分,而对终端用户的反应时间是衡量一个搜索引擎优良的重要指标。讨论了目前几种流行的 P2P 网络搜索技术及特点,并比较其优劣,然后对基于分布式哈希表的搜索技术的几种改进方法进行了分析。

关键词: P2P; 分布式哈希表; Bloom Filter; Cache
中图法分类号: TP301.6　　**文献标识码**: A　　**文章编号**: 1001-3695(2006)01-0071-02

Research on Search Technology of P2P Network

JIA Xing-dan, ZHANG Li-chen
(Faculty of Computer, Guangdong University of Technology, Guangzhou Guangdong 510090, China)

Abstract: Interest in distributed storage system is fueled for its decentralized control, adaptation and self-organization. Search is an important technology for all storage system, and end-user latency is the most important performance metric for a search engine. Discusses several recent popular search technologies of P2P systems and characterizes of this technologies, and compares their advantages and disadvantages, then analyzes several improved ways for DHT-based storage system.

Key words: P2P; DHT; Bloom Filter; Cache

分布式存储系统以其分布式控制、自组织性和普遍的适应性而受到越来越多的关注。但是高级搜索技术仍是一个亟待解决的问题,而在一个搜索引擎中对终端用户的反应时间是最重要的性能指标。在分布式搜索引擎中对最终用户的反应时间多由网络传输时间决定。因此最小化要发送的比特数和发送花费的时间单元数是很重要的。在实际的搜索中,包含有多个关键词需由多台主机协同工作才能完成的查询占大多数,它们决定了网络的负载,因此对它们进行优化对缩短终端用户反应时间是很重要的。

1　P2P 网络中常用的搜索技术的分析

具有集中式的目录服务器的搜索机制(如 Napster),在集中式的目录服务器上存放对等节点的地址信息、元数据和文件的关键词信息。它可以对请求的查询进行快速地查找并返回最合适的目的节点。但是随着网络规模的增大,目录服务器必然成为服务瓶颈,而且会造成单点失败,同时还存在扩展性问题。

采用洪泛查找机制的 P2P 网络,如 Gnutella[2], Freenet 等。可以把这种完全分布式的网络看成是一组对等节点之间的自组网络。节点在进行查找时,首先传播到它的所有相邻节点,然后在传播到相邻节点的所有相邻节点,直至到达预先确定的层次为止。这种查询机制造成网络通信负担较大,也存在扩展性较差的问题。在文献[9]中提出了针对 Gnutella 的利用搜索相近关键词的一组节点构造节点存储的路由表进行组播来减少洪泛查找所造成的网络流量的失控。

基于分布式哈希表的查找机制,如 Chord[3], CAN[4], Pastry[5], Tapestry[6] 等。在 Chord 中每个关键字都保存在它的后继节点上,查找过程就是不断接近它的后继节点最终到达目的节点或查找失败。CAN 基于虚拟的 d 维笛卡儿坐标实现其数据组织和查找功能。Pastry 使用最长共同前缀进行匹配查找。Tapestry 使用邻居映射表进行最长前缀匹配查找,并可把消息传递到最近的存放所要求的对象拷贝的节点。以上介绍的四种基于分布式哈希表的查找机制有很多相似之处。下面对它们进行简单的比较如图 1 所示。

系统	Chord	CAN	Pastry	Tapestry
空间复杂度	$O(n \log n)$	Nr	$O(n \log n)$	$O(n \log n)$
查找复杂度	$O(\log n)$	$Rn^{1/r}$	$O(^{n} \log n)$	$O(\log n)$

图 1　各查找机制比较

由此可知在 Napster 和 Gnutella 中使用的关键词查询的方法,在基于分布式哈希表的 P2P 系统中由于关键词经哈希函数后成为唯一的关键值,就是说基于分布式哈希表查找系统通过一个不透明的关键值来对文件进行查询。关键值选择的方法由构筑在 DHT 之上的应用程序所决定,它缺少有效的关键词查询的功能。然而经改进后,可以不把关键词的查询直接映射在存有相应哈希值的节点上。而是映射在一个哈希表上,节点再映射到此哈希表上来提供高效的关键词查询。在实际的搜索中,包含有多个关键词需由多台主机协同工作才能完成的查询占大多数,它们决定了网络的负载,所以以下的改良方法针对多关键词查询。

收稿日期: 2005-01-01; 修返日期: 2005-04-06
基金项目: 国家自然科学基金(60474072, 60174050); 广东省自然科学基金(04009465, 010059); 广东省高校自然科学研究资助项目(Z03024); 广东省哲学社会科学规划项目(03/04J02)

2 对现有的基于分布式哈希表查找机制的改进方法

2.1 Bloom Filter 算法

Bloom Filter[1] 是一种表示集合的方法,并可简洁地测试一个元素是否在该集合中,它基于哈希函数建立,所存储的比特数远少于它所表示的集合。P2P 网络传输一个基于集合 A 的 Bloom Fiter 而非集合 A 本身,可以减少需要传输的信息量以降低网络流量。但 Bloom Fiter 会导致可预算的错误定位率。Bloom Fiter 的错误定位率随它尺寸的增大而呈指数降低。集合 S 的 Bloom Filter $F(S) = S \cup \varepsilon(S)$, $\varepsilon(S)$ 是错位定位数。P_{fp} 为错误定位率,则

$P_{fp} = (1 - e^{-kn/m})^K$, (k 为哈希函数的个数, m 为 Bloom Filter 的尺寸, n 是集合中元素的个数) 哈希函数选择最优化时,错误定位率为

$$f = 0.6185^{m/n} \quad (1)$$

所以若要保持一定的错误定位率, m 必须与 n 成一定的比例[7]。

下面是关于 Bloom Filter 的集合操作

把集合 S 转换为它相应的 Bloom Filter: $F(S) \leftarrow S$
Bloom Filter 的交集运算: $F(X \cap Y) \leftarrow F(X) \cap F(Y)$
Bloom Filter 的并运算: $F(X \cup Y) \leftarrow F(X) \cup F(Y)$
Bloom Filter 和集合的运算: $(X + \varepsilon(X)) \cap Y \leftarrow F(X) \cap Y$
$F(X) \cap X = X$

优化多关键词搜索重点是降低所用的网络带宽。例如,若服务器 S_A 存放所有含关键词 K_A 的文件集合 A, S_B 存放所有含关键词 K_B 的文件集合 B。$|A|$ 和 $|B|$ 分别表示集合 A 和 B 的大小(即它们包含的文件数目) 。$A \cap B$ 是即包含关键词 K_A 又含关键词 K_B 的文件集合。若一个节点 C 查询搜索既包含关键词 K_A 又含关键词 K_B 的文件即 $A \cap B$。一个直接的方法是: S_A 发送集合 A 给集合 B 所在的节点 S_B。S_B 计算出 $A \cap B$ 然后直接发送 $A \cap B$ 给查询节点 C。若使用 Bloom Filter, S_A 发送集合 A 的 Bloom Filter $F(A)$ 给集合 B 所在的节点 S_B。S_B 计算并发送 $F(A) \cap B$ 给 S_A, S_A 通过计算 $A \cap (F(A) \cap B)$ (与 $A \cap B$ 等价) 去除错误定位的文件,然后发送给节点 C。

S_A 虽可通过计算 $A \cap (F(A) \cap B)$ 在最后去除错误定位的文件,但浪费了带宽。如上,节点 S_A 和节点 S_B 的例子中,共需传递的比特数为 $m + P_{fp}|B| + |A \cap B|$(j 是文件标识符的比特数) 。$|A \cap B|$ 是所要求的交集本身,不能优化。所以可优化的比特数为 $m + P_{fp}|B|$ 与式(1) 联合可得可优化比特数为

$$m + f|B| = m + 0.6185^{m/|A|}|B| \quad (2)$$

当式(2) 取值最小时

$$m = |A| \log_{0.6185}\left(2.081 \frac{|A|}{|B|}\right) \quad (3)$$

由上可知当 A, B 和 j 固定时,优化 m 才能得到最小的传输比特数,所以要合理的选择 Bloom Filter 的尺寸大小。而且当 $|A|$ 和 $|B|$ 不同时优化的性能是非对称的且当 $|A| \leq |B|$ 时传输的比特数更少。

Bloom Filter 交集处理多关键词查询的技术可以推广到任意多个关键词,如下所示:

S_{rq} 是请求查询的节点,求 $A \cap B \cap \cdots \cap Z$
$S_{rq} \rightarrow S_A$: query for $A \cap B \cap \cdots \cap Z$
$S_A \rightarrow S_B$: $F(A)$
$S_B \rightarrow S_C$: $F(F(A) \cap B) = F(A \cap B)$
r
$S_Y \rightarrow S_Z$: $F(F(A \cap \cdots \cap X) \cap Y) = F(A \cap B \cap \cdots \cap Y)$
$S_Z \rightarrow S_Y$: $F(A \cap B \cap \cdots \cap Y) \cap Z$
$S_Y \rightarrow S_X$: $F(A \cap B \cap \cdots \cap Y) \cap Z \cap Y$
r
$S_B \rightarrow S_A$: $F(A \cap B \cap \cdots \cap Y) \cap Z \cap Y \cdots \cap B$
$S_A \rightarrow S_{rq}$: $F(A \cap B \cap \cdots \cap Y) \cap Z \cap Y \cdots \cap B \cap A$

而且当 $|A| \leq |B| \leq |C| \leq \cdots \leq |Z|$ 时可以最优化传输的比特数。

2.2 缓存

使用缓存若 S_B 在本地已经存储有 $F(A)$ 或 A 则可以避免 S_A 继续发送。缓存 $F(A)$ 而非 A 本身的话,相同的空间可以存储更多数据的 Bloom Filter。因为关键词出现的概率呈非对称分布(Zipf 分布) ,所以这就意味着即使很小的 Cache 都可以有很高的命中率。平均地看,一个 Bloom Filter 已存储在另一个节点的概率 p 与该节点 Cache 的命中率相等。此时式(2) 可优化为

$$(1 - r)m + 0.6185^{m/|A|}|B| \quad (4)$$

优化 m 后得

$$m = |A| \log_{0.6185}\left[(1 - r)2.081 \frac{|A|}{|B|}\right]$$

发送比特数的减少和 Cache 命中率的提高近似成线性关系。

2.3 结果的处理

请求查询并不需要返回搜索出的所有结果。如果只传输返回所要求的查询结果,可以在很大程度上减少所要传输的信息量。查询结果的数量与网络中存储的文件的数目成比例,所以用于返回查询结果的带宽和网络规模的增大呈线性增长,因而从系统的可扩展性来说,对结果进行整理是很有必要的。而 Bloom Filter 和 Cache 都不能减少这种线性的增长,所以截去部分结果是唯一的方法。

因为 Bloom Filter 如果被分割就没有任何实际意义,所以 S_A 把本地存储的文件分块,发送一个块的 Bloom Filter 给 S_B, S_B 返回相应块的搜索结果(在此期间 S_A 和 S_B, 保持通信) ,直至达到查询所要求的文件的数目。块的大小依所要返回的文件数目而定。发送一个关键词文件的分块的 Bloom Filter 而不是所有文件的 Bloom Filter 可以减少一个关键词所占用的 Cache 空间,从而提高 Cache 的命中率。

2.4 虚拟主机的技术

P2P 系统的一个显著特点就是异构性。随机的分布关键值可能会导致某个使用频繁的关键词被分配给一个性能不好的节点。或者一个节点可能被分配与它的处理能力不匹配的关键词。虚拟主机[8] 可以降低这种可能性。使用虚拟主机一个节点可能依它的处理能力被划分为若干个逻辑的节点。

2.5 结果的排序

Bloom Filter 和 2.3 节中对结果的处理使得对结果按相关度排序更加复杂。Bloom Filter 只是简单地测试一个元素是否在一个集合中,并不提供元素顺序信息和其他数据在文件中位置的信息,甚至因为使用了 Bloom Filter 而导致这些排序信息的丢失。在 2.3 节中对结果的处理使用了分块的方法,而如果一个块中文档的 ID 比前一个块的小,前一个块文档将优先被发送。

(下转第 97 页)