软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2013,24(9):2117−2131 [doi: 10.3724/SP.J.1001.2013.04347]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

副本协作预取中文件相关性查询机制*

田 田, 罗军舟, 宋爱波, 东 方

(东南大学 计算机科学与工程学院,江苏 南京 211189)

通讯作者: 田田, E-mail: tian_tian@seu.edu.cn

摘 要: 副本协作预取是一种网格副本复制策略,旨在通过获取隐性高价值文件进一步降低数据访问延迟.副本协作预取的关键在于隐性高价值文件的确定和获取,因此,提高隐性高价值文件获取的速度能够大幅度提升副本协作预取的性能.利用 DHT(distributed hash table)组织网格节点,以快速定位隐性高价值文件查询所必需的文件相关性信息.针对隐性高价值文件的特殊查询模式,提出基于预取规则树的文件相关性信息存储结构及其查询机制,以提高查询效率.基于真实网格作业运行记录的实验,验证了所提出方法的有效性和高效性.

关键词: 网格;副本复制;协作预取;文件相关性;分布式哈希表

中图法分类号: TP393 **文献标识码**: A

中文引用格式: 田田,罗军舟,宋爱波,东方.副本协作预取中文件相关性查询机制.软件学报,2013,24(9):2117−2131. http://www.jos.org.cn/1000-9825/4347.htm

英文引用格式: Tian T, Luo JZ, Song AB, Dong F. Query mechanism for file correlation in cooperative replica prefetching. Ruan Jian Xue Bao/Journal of Software, 2013,24(9):2117−2131 (in Chinese). http://www.jos.org.cn/1000-9825/4347. htm

Query Mechanism for File Correlation in Cooperative Replica Prefetching

TIAN Tian, LUO Jun-Zhou, SONG Ai-Bo, Dong Fang

(School of Computer Science and Engineering, Southeast University, Nanjing 211189, China)

Corresponding author: TIAN Tian, E-mail: tian_tian@seu.edu.cn

Abstract: A cooperative replica prefetching mechanism is a new replication strategy aiming to reduce the data access latency in grid by the means of replicating implicit high-value files (IHVFs). The key of this new mechanism is the exploration and location of IHVFs. Locating the IHVFs quickly can improve the performance of the mechanism. This paper employs the DHT (distributed hash table) technology to organize grid nodes to quickly locate the file correlation information which is essential for the query of IHVFs. Considering the specific query pattern, a PTree-based storage structure and corresponding query mechanism for file correlation information are proposed to improve the IHVF query efficiency. Simulation results based on real grid workloads have confirmed the effectiveness and high efficiency of the proposed methods.

Key words: grid; replication; cooperative prefetching; file correlation; DHT

数据网格聚集大量分布式的计算、存储等资源,以确保网格应用的顺利执行[1,2].网格作业被提交到网格节点上执行时,可能需要一些文件作为输入,特别是对于数据密集型应用的作业,其输入文件的数据量通常较大.若作业请求的文件未存储在本地,就需要从其他节点获取该文件的副本,以确保作业的顺利执行.此时,作业的

* 基金项目: 国家自然科学基金(61070161, 61003257, 61202449, 61272054); 国家重点基础研究发展计划(973)(2010CB328104); 国家科技支撑计划(2010BAI88B03, 2011BAK21B02); 高等学校博士学科点专项科研基金(20110092130002); 国家科技重大专项科研基金(2010ZX01044-001-001); 江苏省产学研前瞻性联合研究项目(BY2012202); 江苏省自然科学基金(BK2008030); 江苏省网络与信息安全重点实验室(BM2003201); 计算机网络和信息集成教育部重点实验室(93K-9); 上海市可扩展计算与系统重点实验室(上海交通大学)(2010DS680095); 浙江师范大学计算机软件与理论省级重中之重学科开放基金

收稿时间: 2011-11-26; 修改时间: 2012-05-29; 定稿时间: 2012-10-26

执行时间很大程度上会受到数据传输时间的影响,数据传输时间过长会极大地影响作业的执行效率.为了降低数据访问延迟,网格中通常利用副本复制技术将文件以多副本的形式存储于多个节点.在我们之前的研究工作中提出了网格虚拟组织副本协作预取机制[3],通过获取隐性高价值文件的副本,进一步降低了网格数据访问延迟.副本价值可以通过不同的效益函数进行反映.我们利用副本在未来一段时间内的访问次数来表示副本价值,访问次数越多,副本价值越高.在此基础上,隐性高价值文件(implicit high-value file,简称 IHVF)是指具有较高价值但还未在本地节点体现访问需求的文件.与之相对,显性高价值文件(explicit high-value file,简称 EHVF)是指具有较高价值且已在本地节点体现访问需求的文件.例如,若在某节点 P 上存在时刻 T,文件 A 在 T 时刻前被访问 20 次;文件 B 在 T 时刻前被访问 0 次,T 时刻后被访问 20 次(文件 A,B 不在节点 P 上),则在 T 时刻,相对于节点 P,A 是显性高价值文件,B 是隐性高价值文件.显性高价值文件是文件访问需求驱动的复制策略通常选择的复制对象[5−9].该类策略的特点即需求驱动,即只有在节点对文件有所需求时才会考虑复制副本.网格副本领域之前的研究成果都忽略了隐性高价值文件的存在.

副本协作预取的关键,在于隐性高价值文件的确定和获取.虚拟组织作为网格中实现分布式资源发现和共享的基本架构,其重要特性是其成员具有共同目标,因此更容易访问相同的或相关性较大的文件[3].在虚拟组织中,隐性高价值文件可以通过获取与显性高价值文件相关性较大的文件而实现[4].文件的相关性可用文件相关性信息(file correlation information,简称 FCI)进行表示.文件相关性信息由各个节点根据自身的文件访问记录独自提取,分布存储于各个节点之上,其形式为(FO_p,FO_s).其中,FO_p 和 FO_s 各代表一个文件对象(file object,简称 FO),其含义是与文件对象 FO_p 相关的文件对象为 FO_s,文件对象可能是单个文件也可能是多个文件的集合.至此,隐性高价值文件的获取可以简单分为如下两步:

(1) 以显性高价值文件作为 FO_p 查询文件相关性信息;

(2) 根据获取的 FO_s,选择合适的副本复制到本地节点.

由于获取隐性高价值文件所必需的文件相关性信息并不存储在本地节点,因此,本地节点需要与其他网格节点进行协作,以实现隐性高价值文件的预取,我们称其为副本的协作预取.

不难看出,文件相关性信息的快速查找和定位对副本协作预取的性能显得尤为重要.若能提供一种快速高效的文件相关性信息的组织和管理机制,对分布在各个节点上的文件相关性信息进行有效存储和管理,进而保证其快速查找和定位,则可以加速获取隐性高价值文件,进一步提高副本协作预取的性能.DHT 技术是一种分布式的查询协议,可扩展性较好,最为关键的是,其能够提供资源的快速定位.而文件相关性信息同样是分布式存储,且由于虚拟组织环境的动态性使得其查询机制也需要较好的可扩展性,因此,DHT 技术十分适合用于文件相关性信息的查询.但是,由于文件相关性信息本身的特性以及隐性高价值文件的特殊查询模式,使得在具体实现时还需要解决以下 3 个问题:

(1) 利用 DHT 实现文件相关性信息查询的关键在于 DHT 所采用的哈希函数中哈希关键字的取值方式,该取值方式需要适用于节点提取出的不同形式的文件相关性信息,例如,FO_p 和 FO_s 中包含的文件个数可能不同;

(2) 隐性高价值文件查询有其特殊的查询模式,为了提高查询效率,需要设计与其对应的文件相关性信息的存储方式;

(3) 由于每个网格节点执行的作业不同,导致形成的文件访问记录也不尽相同,因此,不同的节点提取文件相关性信息时,对于相同的 FO_p 可能会提取出不同的 FO_s,在存储和查询时需要进行有效的处理.

为此,我们利用 DHT 组织虚拟组织中的网格节点,据此设计出基于 DHT 的副本协作预取模型,并提出一种用于协作预取中文件相关性信息的查询机制.本文的贡献主要包括以下几个方面:

(1) 设计了基于 DHT 的副本协作预取模型,以加速隐性高价值文件的获取;

(2) 分析了隐性高价值文件的特殊查询模式,提出基于预取规则树的文件相关性信息存储结构及算法以提高查询效率;

(3) 给出了与基于预取规则树的存储方式相对应的文件相关性信息查询机制.

本文第 1 节介绍网格环境下副本复制和预取的相关工作.第 2 节描述基于 DHT 的副本协作预取模型.第 3 节给出该模型的关键机制.第 4 节模拟实验并分析结果.第 5 节总结当前工作.

1 研究现状

网格副本复制技术已经得到了广泛深入的研究.由于网格节点存储资源有限,尽可能地复制价值较高的副本已经成为共识[5].传统的文件访问需求驱动的副本复制策略通常选择显性高价值文件进行复制[5−9],其实质是利用了文件访问的时间局部性原理.根据时间局部性原理,最近被访问的文件在未来有更大几率获得较多的访问次数,属于显性高价值文件,因此需要优先复制.与上述工作针对的对象不同,副本协作预取研究的是隐性高价值文件副本的复制.另一方面,已有的副本复制多关注于系统全局性能,比如降低数据平均访问延迟[8−10],副本协作预取则关注于本地数据访问延迟.这是因为在网格环境下,个体服务质量(例如本地数据访问延迟)相对于总体服务质量(例如数据平均访问延迟)已经得到了越来越多的重视[11,12].各个节点部署有不同的应用,对数据访问速率具有不同的需求.副本复制策略需要更好地满足这些需求.

在副本复制中充分考虑文件间的相关性,已经被证明是一种有效提高复制效率的方式,也是近年来副本复制研究方面的热点.文献[13]通过分析网格中真实的高能物理应用文件访问记录,提出了文件簇(filecule)的概念,文件簇的实质是相关性较大的文件组成的集合.在随后的研究中[14,15],研究者提出了基于文件簇的副本复制方式,将副本复制的粒度由单个文件扩大到文件簇,并证明了基于文件簇的副本复制方式的可靠性和有效性.文献[16]同样提出了一种基于文件聚类的副本复制策略,作者采用了与文献[13]不同的文件打包方式,将一定时间段内连续被访问的文件打包形成文件集合,并以此文件集合作为最小复制对象进行副本复制.文献[17]则提出了一种基于文件再联合(file reunion)的副本复制策略.通过计算文件与一组作业的组相关度(group correlation degree),将原本可能存放在不同节点的组相关度较大的文件存放到同一存储节点,以此降低数据访问延迟.上述策略都基于文件相关性原理,节点通过对自身已有的文件访问记录进行分析,提取文件相关性,进而对文件存储位置作出适当调整.然而,上述策略顺利实施的前提是本地节点必须存储有提取文件相关性信息所必需的文件访问记录,并且缺乏对各个节点提取的文件相关性信息的有效协作和利用,这使得它们的复制对象仍然只局限于显性高价值文件.

与上述策略不同,副本协作预取对各个节点提取的文件相关性信息采用了一种协作处理的方式,本地节点在没有相关文件访问记录的情况下可以通过查询其他节点的文件相关性信息以获取隐性高价值文件.隐性高价值文件的获取同样基于文件相关性原理,即通过寻找与显性高价值文件具有较大相关性的文件来实现.文件间的相关性信息通过引入文件预取技术进行描述.在文件预取中,通常根据文件访问记录提取出预取规则,以此来反映文件间的相关性,虽然提取方式可能不同[18],但都基于一个基本的原理,即文件访问序列中位置靠近的文件通常相关性较大.之所以没有采用基于文件内容计算文件相关性的方式(典型如 VSM 算法[19]),主要是基于两个原因:一是因为网格系统文件数量巨大,采用该方式会消耗大量存储资源和计算资源;其次是因为网格中很多文件只包含单纯的数值型数据,语义性不强,难以提取实质内容.

为了能够快速定位文件相关性信息,加速获取隐性高价值文件,我们引入 DHT 技术实现文件相关性信息的存储和查询.本文使用 Chord[20]作为典型的 DHT 技术,其余不同的 DHT 技术可以类似地使用.Chord 提供的基于 DHT 的路由协议,可以确保在 $O(\log N)$时间内快速定位资源,这使得 Chord 在众多领域已被广泛应用,比如文件存储、视频播放等.目前,网格环境下预取方面的研究较少,文献[21]研究内存网格的内存预取技术,文献[3]研究了网格虚拟组织中的文件协作预取.在文献[3]中,我们已经通过对虚拟组织文件访问特性的深入分析,给出了隐性高价值文件在虚拟组织中的存在依据.并着重研究了副本协作预取中文件相关性信息的动态提取方式,提出了作业类型为中心的文件预取算法.本文则主要研究文件相关性信息的存储和查询机制,以达到加速获取隐性高价值文件的目标.

2 基于 DHT 的副本协作预取模型

首先介绍副本协作预取的触发场景.若某节点经常接收到文件 f 的请求且请求数达到阈值,则根据文件访问需求驱动的复制策略,文件 f 属于显性高价值文件.此时,若 f 未在本地存储,则应在本地节点创建文件 f 的副本.本地节点在创建 f 的副本之后,即可以启动副本协作预取,试图通过预取文件 f 相关的隐性高价值文件的副本降低未来一段时间的数据访问延迟.副本协作预取触发后,本地节点需要查询 f 的文件相关性信息,并根据返回的结果选择合适的文件启动预取.在此前的工作中,我们已确定采用预取规则(prefetching rule,简称 P-Rule)描述文件相关性信息[3].预取规则定义见定义 1.具体的预取规则提取方式不在本文讨论范围,提取细节可以参考文献[3],本文假设各个节点已经提取出预取规则.

定义 1(文件预取规则 P-Rule). P-Rule=$P_1P_2\ldots P_n\rightarrow S_1S_2\ldots S_m$.

每条预取规则由前缀和后缀两部分组成,分别对应于文件相关性信息中的 FO_p 和 FO_s.其中,预取规则的前缀长度为 n,后缀长度为 m.其含义是:若已经访问文件 $P_1,P_2,\ldots,P_n$,则随后有可能访问的文件依次为 $S_1,S_2,\ldots,S_m$.每条预取规则都有其对应的支持度来反映其在文件访问记录中出现的频繁程度,预取规则的支持度指的是从所有文件访问模式中提取出该预取规则的总次数.

下面给出基于 DHT 的副本协作预取模型的详细描述.模型采用 Chord 环连接虚拟组织中的网格节点.每个网格节点根据自身的文件访问记录独立提取预取规则,每个节点和每条预取规则经过哈希后都拥有全局唯一 ID,每条预取规则由具有与该预取规则相同 ID 或最近 ID 的节点进行维护[20].节点提取的预取规则除了本地存储备份外,还需要存储到 Chord 环上的维护节点以供查询.节点的预取规则发生改变时需要发出更新消息及时通知维护节点进行更新.模型包括两个关键机制:P-Rule 存储机制和 P-Rule 查询机制.P-Rule 存储机制顾名思义,负责存储预取规则以满足文件相关性信息的查询需求,本文提出了基于预取规则树的存储方式以提高 P-Rule 查询性能;P-Rule 查询机制提供了实现文件相关性信息查询的基本算法,并针对可能的查询负载不均设计了负载均衡策略.

3 关键机制

3.1 P-Rule存储机制

当节点生成一条新的预取规则时,除了存储到本地的预取规则库,还需要根据 Chord 提供的一致性哈希函数(consistent hashing)将该预取规则哈希后并存储到 Chord 环上对应的维护节点.P-Rule 存储机制的关键在于一致性哈希函数的哈希关键字取值问题.该取值方式首先需要满足两个要求:

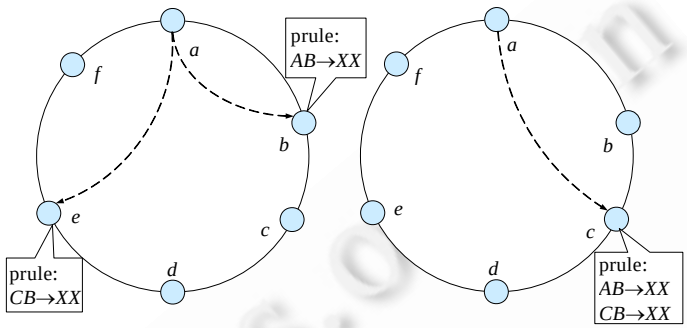
(1) 通用性.由于提取出的预取规则前后缀长度可能不一,该取值方式需要适用于不同前后缀长度的预取规则;

(2) 高效性.在满足通用性的基础上,需要尽可能地提高查询效率.

显然,最简单的方式是取每条预取规则的完整前缀作为关键字进行哈希,并存储到对应维护节点,我们称这种存储方式为 FullPre.FullPre 满足通用性,查询实现也较为简单,可以直接调用 Chord 提供的查询算法完成查询,例如需要查询 AB 为前缀的预取规则,此时取 AB 作为哈希关键字查询即可.但是,FullPre 会带来新的问题,即预取规则的无序存放.

设想如下场景:网格节点 a 试图预取文件 B 相关的隐性高价值文件,因此需要查询 B 的文件相关性信息.为了增加预取准确率,可以在查询中引入前缀:节点查询本地文件访问记录,发现文件 B 之前常伴有文件 A 或 C 的访问,即有文件访问模式 AB 和 CB,因此需要查询分别以 AB 和 CB 为前缀的预取规则.针对上述场景,在采用 FullPre 存储方式时,前缀为 AB 和 CB 的预取规则由于前缀不同,很可能经过哈希后会存储到不同的节点,如图 1(a)所示,预取规则 $AB\rightarrow XX$ 和 $CB\rightarrow XX$(X 表示任意文件 ID)经过哈希后分别存储到了节点 b 和 e 上,我们将这种现象称为预取规则的无序存放.预取规则无序存放导致节点 a 发出的预取规则查询需要在 Chord 环上进行两次路由,查询代价为 $2O(\log N)$,而我们希望只进行一次路由即可完成预取规则查询.

从查询需求来看,节点 *a* 所关心的是文件 *B* 的隐性高价值文件.反映到预取规则上,就是需要查询前缀尾字母为 *B* 的预取规则.因此,若能将前缀尾字母相同的预取规则存放在同一维护节点,则节点 *a* 发出的预取规则查询只需要一次路由即可完成,正如图 1(b)所示,以 *AB* 和 *CB* 为前缀的两条预取规则经过哈希后都存储在节点 *c* 上.这种存储方式避免了预取规则的无序存放带来的查询代价增高的问题.



(a) FullPre (b) LastPre

Fig.1 Illustration of FullPre and LastPre

图 1 FullPre 和 LastPre 存储方式图例

为此,我们提出了一种基于预取规则树(P-Rule tree,简称 PTree)的 P-Rule 存储方式,用 LastPre 表示.LastPre 的基本思想是:取每条预取规则前缀的尾字母作为关键字进行哈希并存储到相应维护节点,在维护节点上形成一种被我们称作预取规则树的存储结构,并且每个维护节点都可能存储多棵预取规则树.图 2 具体描述了 LastPre 存储方式.假设节点 *a*,*b*,*d*,*f* 和 *g* 各自生成了一些预取规则,如图 2(a)所示,这些预取规则的前缀尾字母都是 *B*.以 *B* 作为关键字进行哈希,设哈希后的对应维护节点是 *c*,则将上述这些预取规则都存储到节点 *c*,进而形成图 2(c)所示的一棵关于文件标识符 *B* 的预取规则树,用 *PTree*(*B*)表示.预取规则树的构建借鉴了 Trie 树的思想,存在一些基本性质如下:

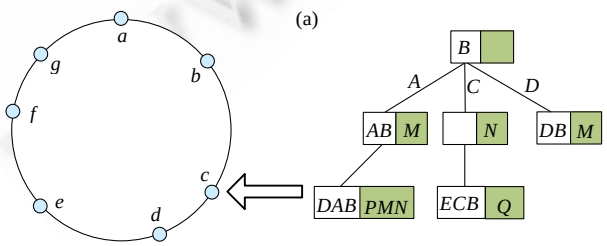
(1) 预取规则树的节点都用 pNode 表示,其中根节点必然对应于预取规则前缀的尾字母;

(2) 树中节点间通过字符相连接.从根节点到某一节点,将路径上经过的字符连接起来所形成的字符串即为该节点对应的标识符;

(3) 每个节点的所有子节点都具有不同的标识符.

AB→*M*	From node *a*
DAB→*PN*	From node *b*
CB→*N*	From node *d*
ECB→*Q*	From node *d*
DAB→*PM*	From node *f*
DB→*M*	From node *g*

(a)



(b) (c)

Fig.2 Detailed description of LastPre

图 2 LastPre 存储方式

在预取规则树中,节点除了自身的标识符外,还附有一个后缀,定义如下:

定义 2(预取规则树节点的后缀($pNode_{suf}$)). 若一条预取规则的前缀和预取规则树中某个节点的标识符相同,则符合上述条件的所有预取规则的后缀按序形成的集合组成了该节点的后缀.

仍以图 2(c)为例,*PTree*(*B*)中存在节点"*DAB*",而图 2(a)中预取规则 *DAB*→*PM* 和 *DAB*→*PN* 的前缀也是 *DAB*,因此,节点"*DAB*"的后缀包含 *P*,*M*,*N*.由此可以看出,节点的后缀可能由多个预取规则的后缀组成,而这些预取规则的后缀可能不同,支持度也可能不同.$pNode_{suf}$ 具体的计算和更新方法将在第 3.2 节进行介绍.

P-Rule 存储算法见算法 1 和算法 2 所示.节点生成一条新的预取规则 *p* 时,执行算法 1,向预取规则 *p* 对应的维护节点发出 P-Rule 存储请求(*prule_store_request*),存储请求包含参数 *p* 和 *q*,其中,*q* 是预取规则 *p* 前缀的尾字母.维护节点接收到 P-Rule 存储请求后,执行算法 2.算法 2 首先定位待插入的预取规则 *p* 对应的预取规则树,然后定位该预取规则树中待更新的节点(第 3 行~第 8 行),该节点必须满足如下条件:节点的标识符等于预取规则 *p* 的前缀.需要注意的是,若预取规则树中找不到满足上述条件的节点,需要创建新的满足上述条件的节点.算法 2 最后调用 *UpdatePNode*(·)函数计算和更新该节点的后缀.

算法 1. *StorePR*(prule *p*).

1. Extract the last letter of the prefix of *p* and assign it to *q*
2. *k*=*Hash*(*q*)
3. send a *prule_store_request*(*p*,*q*) to the node which is an immediate successor of *k*

算法 2. *InsertPR*(·).

Require:

(1) (*p*,*q*), a *prule_store_request*, *p* is a p-rule, *q* is the last letter of the prefix of *p;*

(2) *p*[*n*], the full prefix of *p*;

(3) *T*, identifier of the node that needs to be updated.

1. When receiving a *prule_store_request*(*p*,*q*), extract the full prefix of *p* and assign it to *p*[*n*]
2. Locate the PTree *t* whose root node is *q*
3. *T*=*q*
4. **for** *i*=*n*−2 **to** 0 **do**
5. **if** Locate a pnode in *t* with its identifier identical to the string "*p*[*i*]…*p*[*n*−1]"
6. **then** *T*=*p*[*i*]+*T*; *i*=*i*−1
7. **else** Create a new pnode *n* with identifier "*p*[*i*]…*p*[*n*−1]" and make *n* to be the child of the node whose identifier is identical to the string "*p*[*i*+1]…*p*[*n*−1]" in PTree *t*
8. **end for**
9. *UpdatePNode*(*p*,*T*)

3.2 $pNode_{suf}$的计算和更新

虚拟组织中,每个网格节点都可以处理用户提交的作业.不同的作业具有不同的文件访问请求,因此各个节点形成的文件访问记录也不尽相同.随之带来一种情况,即各个节点可能会提取出前缀相同而后缀不同的预取规则,例如图 2(b)中显示的 *DAB*→*PM* 和 *DAB*→*PN*.在采用 LastPre 方式存储预取规则时,以 *DAB*→*PM* 和 *DAB*→*PN* 为例,若 *DAB*→*PM* 支持度为 s_1,*DAB*→*PN* 支持度为 s_2,则在 *PTree*(*B*)的节点 *DAB* 上,需要一种算法综合处理所有前缀等于该节点的标识符的预取规则,以确定该节点的后缀,即 $pNode_{suf}$ 的值.在此例中,即需要确定在已访问 *DAB* 的情况下,*P*,*M* 和 *N* 被访问的可能性大小排序.

根据定义 1,预取规则后缀中排位越靠前的文件越有可能被访问,预取价值越高.延续这一设定,认定在预取规则树节点的后缀中排位越靠前的文件预取价值越高.为此,创建预取价值系数(prefetching value,简称 pv)来衡量文件的预取价值,系数越高,意味着文件被访问的可能性越大,在预取规则后缀中排位越靠前.计算方式如下:对于一个预取规则树中的节点,设有 *n* 条预取规则其前缀等于该节点的标识符,支持度分别为 $s_i(1\leqslant i\leqslant n)$.设定

后缀长度为 m 的预取规则由前至后每一位对应的权重分别为 $\alpha_1,\alpha_2,\ldots,\alpha_m$,且 $\alpha_1>\alpha_2>\ldots>\alpha_m$.本文中,$\alpha_1$ 取 0.9,其他权重值以 0.1 逐位递减,且后缀长度 m 不大于 4.那么对于一个文件标识符 f,若其在预取规则 i 的后缀中处于位置 k,则 f 在预取规则 i 中获得的预取价值系数 $pv_i=\alpha_k\times s_i$.据此,f 的累积预取价值系数 $pv_{total}(f)$ 为

$$pv_{total}(f)=\sum_{i=1}^{n}pv_i=\sum_{i=1}^{n}\alpha_k\cdot s_i \tag{1}$$

至此,预取规则树节点的 $pNode_{suf}$ 计算方法如下:对于每棵预取规则树的每个节点,将其后缀可能包含的文件标识符集合记为 P_{suffix},计算 P_{suffix} 中每个文件标识符的 pv_{total},最终按照 pv_{total} 的大小降序排列形成该节点的后缀.此外,当网格节点某个预取规则的支持度发生改变时,需要即时更新维护节点上对应节点的后缀.网格节点以(*p*,*s*)的形式发出预取规则更新消息,其中,p 是预取规则,s 是 p 更新后的支持度.更新消息通过 Chord 环路由到维护节点,维护节点根据 p 的前缀确定需要更新的预取规则树中的节点,重新计算 p 所涉及到的所有文件标识符的 pv_{total},最终对该节点 P_{suffix} 中的所有文件标识符重新排序,以形成新的节点后缀.

需要指出的是,为了提高预取准确率,对象的动态流行度分布模型已经在预取中得到了越来越多的重视[18].在我们之前的工作中,提出了一种基于动态支持度的预取规则提取算法,该算法对每条预取规则分别记录其最近支持度和过去支持度,在此基础上,根据公式计算其逻辑支持度,而逻辑支持度更好地反映了文件动态流行度对预取性能的影响.上述这种动态的提取方式会增加预取规则的更新频率,表现在:每个节点每隔一段时间 t_u(t_u 的值事先设定),都需要重新计算发生改变的预取规则的逻辑支持度.这会导致每隔一段时间 t_u,在 Chord 环上都会出现大量的预取规则更新消息需要路由.消息数量的大量增加会增加网络流量,加重网络负载,我们采用打包发送的方式以降低消息数量:每个节点对于本节点需要发往同一维护节点的所有更新消息,每 k 条打包成一个更新消息进行发送,进而路由到相应的维护节点.此时,打包的消息数 k 会影响打包发送方式的性能,我们会在实验部分给出相关实验评价.

3.3 P-Rule查询机制

节点试图获取某个文件相关的隐性高价值文件时,需要执行 P-Rule 查询.P-Rule 查询算法见算法 3、算法 4.查询请求节点执行算法 3,向目标节点为 k 发出 P-Rule 查询消息(*prule_query_request*).查询消息包含参数 f 和 *pre*[*n*].其中:f 表示希望获取文件 f 相关的隐性高价值文件;*pre*[*n*]是查询前缀数组,根据本地节点上的文件访问记录确定.其中,n=1 时称为单前缀查询,n>1 时称为多前缀查询.例如,若文件 f 之前常伴有文件 e 的访问,则将 n 赋值为 1,将 e 赋值给 *pre*[0].此外,*pre*[*n*]也可以为空,将这种情况下的查询当作是单前缀查询的一种特殊情况,其查询前缀仅为 f 本身.维护节点 k 收到 P-Rule 查询消息后,执行算法 4.算法 4 对于每个查询前缀,依次定位符合查询要求的预取规则树和树中的节点,若节点的后缀不为空,则将排名前 K 的后缀存入 *pfile* 数组,*pfile* 数组作为返回值用于存放符合查询要求的文件标识符.如无特殊说明,本文中 K 一般取 2.

算法 3. *FindPR*(·).

Require:

(1) *f*, file identifier;

(2) *pre*[*n*], string array of the prefix of query message.

1. *k*=*Hash*(*f*)

2. send a *prule_query_request*(*f*, *pre*[*n*]) to the node which is an immediate successor of *k*

算法 4. *LocatePR*(·)

Require:

(1) (*f*,*pre*[*n*]), a *prule_query_request*;

(2) *t*, a Ptree.

Output: *pfile*[·].

1. Locate the PTree *t* whose root node is *f*

2. **if** *pre*[*n*]==*NULL*

3. **return** *TopKselect*(*f*)
4. **else**
5. **for** *i*=0 **to** *n*−1 **do**
6. **if** Locate the pnode *v* in *t* whose identifier is identical to string *pre*[*i*]+*f*
7. *pfile*[·]=*pfile*+*TopKselect*(*v*)
8. **return** *pfile*[·]

由算法 3、算法 4 可知,P-Rule 查询的响应时间可以分为两个部分:第 1 部分是寻找预取规则所在维护节点的时间,用 ST_1 表示;第 2 部分是在维护节点上查找符合查询要求的预取规则树及树中节点的时间,用 ST_2 表示.ST_1 即查询消息在 Chord 环上的路由时间,用 $O(\log N)$表示.ST_2 与维护节点上预取规则树的数量 m,查询前缀的个数 n 以及每个查询前缀的长度 l 有关,可以表示为 $O(m)+O(nl)$.在 P-Rule 实际查询中,当网格节点数目 N 较大时,m,n 和 l 相对于 N 都可以忽略不计,导致 ST_2 相对于 ST_1 可以忽略不计,因此,P-Rule 查询响应时间可简单表示为 $O(\log N)$.

节点查询负载较重时会导致查询响应时间增加,甚至不能响应,为此需要均衡节点负载.Chord 采用的一致性哈希提供了较好的负载均衡功能,对于任意 N 个节点和 K 个关键字,有很高的可能性使得每个节点最多只负责 $(1+\varepsilon)K/N$ 个查询关键字,$\varepsilon=O(\log N)$[20],这在每个查询关键字都具有相近的查询负载时会比较有效.然而,协作预取中热点问题同样客观存在,各个预取规则的查询密度不尽相同,导致每棵预取规则树也可能具有不同的查询负载.查询负载主要耗费节点的计算资源,因此,本文用 CPU 负载衡量一个节点的负载情况.设节点处理能力为 C_i,当前负载为 L_i,则节点利用率 u_i 表示如下:

$$u_i=\frac{L_i}{C_i}.$$

当 $u_i>1$ 时,表示节点过载.为了能够及时进行负载转移,一般 u_i 设置有一个小于 1 的阈值,超过该阈值即引发负载转移.首先,设置合理的负载转移粒度.为了避免第 3.2 节中出现的预取规则无序存放导致查询代价变大的问题,我们选择以整棵预取规则树为最小负载转移单位,而不是预取规则树的子树.这是因为若以预取规则树的子树作为转移单位,则一次 P-Rule 查询所需要的预取规则经过转移之后很可能会不在同一维护节点上,这无疑会导致查询响应时间的增加;其次,在负载转移大小的设置上,文献[22]已经指出,负载转移时应尽量避免较大的负载移动,我们通过设置负载转移比率 δ 来控制转移负载 L_p 的大小,$L_p=\delta\times C_i$.本文中,默认 δ 为 0.2.至此,具体的待转移预取规则树(用集合 PT_{shift} 表示)的打包方式可如下所述:每个节点统计本地存储的每棵预取规则树产生的查询负载,当节点过载时,按照查询负载由大到小的顺序,逐个将查询负载小于 L_p 的预取规则树放入到 PT_{shift} 中,直到 PT_{shift} 的累积负载大于等于 L_p 时停止.

节点确定了待转移的负载之后,需要确定负载转移的目标节点.目标节点的选取通过调用网格资源监控服务[23]实现.网格资源监控服务是网格中的一种基础服务,负责监控网格中所有节点资源的状态信息,而节点负载信息就属于状态信息的一种.资源监控服务被调用后,会根据当前收集到的负载信息按照由小到大的顺序确定 top$K$ 个轻载节点,并从中随机返回一个给负载转移请求节点.同时,资源监控服务将返回的轻载节点进行标记,以表明该节点处于接受负载状态.处于接受负载状态的节点则不能再作为轻载节点推荐给其他负载转移请求节点.图 3 是 PTree 负载转移的示意图.其中,节点 c 是 $PTree(B)$的原始维护节点.所谓原始维护节点,是指采用 LastPre 存储方式时预取规则树的初始存储节点,每个预取规则树只有唯一的原始维护节点.预取规则树在迁移时会保留有自身的原始维护节点信息.对于每棵预取规则树,只有其原始维护节点存放有该规则树对应的 PTree 索引表条目.PTree 索引表中存放的则是每棵预取规则树最近一次迁移后所在的维护节点地址.如图 3 所示,整个 PTree 负载转移过程可以分为如下几步:

(1) 节点 c 调用网格资源监控服务确认负载转移目标节点为 e,则将 PT_{shift} 中所有预取规则树的数据副本以端到端的方式传输给节点 e.这里有两点需要说明:首先,在负载转移未结束之前,PT_{shift} 中的所有预取规则树在节点 c 上仍然存在,以用于响应负载转移过程中可能收到的 P-Rule 查询请求;其次,若负

载转移过程中节点 c 收到涉及到 PT_{shift} 中预取规则树的 P-Rule 更新请求,则暂时不进行更新,而是等到负载转移结束后再将更新消息导入到转移后的预取规则树维护节点;

(2) PT_{shift} 中所有预取规则树的数据副本传输到节点 e 之后,节点 e 向节点 c 发出确认消息(ack message).节点 e 核实迁移来的所有预取规则树的原始维护节点信息,对于原始维护节点不是 c 的每棵预取规则树(适用于预取规则树多次迁移的情况),节点 e 需要通知其原始维护节点该预取规则树目前的维护节点地址,以方便原始维护节点更新 PTree 索引表;

(3) 节点 c 收到确认消息后,删除本节点上存储的 PT_{shift} 中的所有预取规则树,并更新 PTree 索引表中属于 PT_{shift} 的所有预取规则树的路由信息.更新方式如下:对于 PT_{shift} 中的每棵预取规则树,若该预取规则树的原始维护节点即为本地节点,则将 PTree 索引表中该预取规则树对应的值更新为新的维护节点地址,本例中以 $PTree(B)$为例,节点 c 将索引表中 $PTree(B)$的值更新为 e;反之,则不做任何更新.

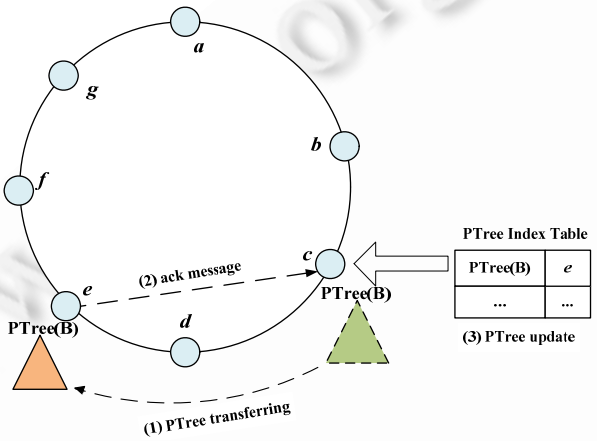


Fig.3 Illustration of PTree shifting process

图 3 预取规则树转移图例

3.4 副本协作预取流程

在采用了基于 DHT 的副本协作预取模型,并利用 P-Rule 存储机制和查询机制实现了文件相关性信息的存储和查询后,副本协作预取的具体流程可以描述如下:

(1) 本地节点 a 经常接收到文件 f 的请求,若文件 f 的请求数符合文件访问需求驱动的复制策略中创建副本的条件,则在节点 a 创建副本 f;

(2) 节点 a 创建副本 f 的同时启动协作预取,试图获取 f 相关的隐性高价值文件的副本;

(3) 节点 a 调用算法 3 发出 P-Rule 查询消息;

(4) 查询消息路由到目标维护节点 k;

(5) 维护节点 k 调用算法 4 并将返回值发回给 P-Rule 查询请求节点;

(6) 本地节点 a 在收到维护节点 k 返回的包含符合预取要求的文件标识符的消息后,选取副本启动副本复制.

从流程的步骤(1)、步骤(2)可以看出,副本协作预取与文件访问需求驱动的复制策略具有紧密联系,主要表现在副本复制的触发时机上.这是因为:需求驱动复制策略的复制对象是显性高价值文件,体现的是已有的文件访问需求;而协作预取的复制对象是隐性高价值文件,体现的是未来的文件访问需求.因此,将这两者相结合可以进一步降低数据访问延迟.

在采用副本协作预取的系统中,所有预取来的文件副本会存储到本地节点上的一个独立存储空间中,我们称为预取池.预取池的空间大小 S_p 与本地节点存储空间 S 之间有如下关系:

$$\lambda = \frac{S_p}{S} \tag{2}$$

S_p的大小会影响协作预取的效果,可通过设置λ的大小调整预取池的空间大小,这点我们在实验中会有详细说明.另外,预取池采用 LRU 算法进行管理.

4 实验分析

4.1 实验设置

首先,评估基于 DHT 的副本协作预取模型在 P-Rule 查询、更新以及负载转移时的性能.编程实现了上述模型,利用 PeerSim[24]模拟 Chord 环境,在此基础上,编写了预取规则树的存储、更新、查询和转移等相关函数,以支持 P-Rule 存储、更新、查询和负载均衡等功能.P-Rule 查询时,采用 FullPre 存储方式作为 LastPre 的比较对象.负载转移时,资源监控服务确定 top$K$ 个轻载节点,K 设置为节点总数的 5%.实验中使用的预取规则提取自真实的网格作业运行记录,后者采集自网格平台 SEUGrid,SEUGrid 是东南大学专为 AMS 实验搭建的一个网格平台[25].其次,应用副本协作预取的最终目标是为了降低数据访问延迟,因此,实验需要评估基于 DHT 的副本协作预取模型对网格数据访问延迟的影响.实验中,通过回放真实的作业运行记录,模拟网格作业从提交到发出文件请求以及文件请求被响应后的执行过程.我们共采集了 3 周的作业运行记录,从第 1 周的记录中提取出预取规则,而后两周记录则用作测试集.每条记录包含作业的运行节点、作业请求的文件、请求文件大小等参数.

4.2 结果与分析

实验中主要采用的性能指标如下:

(1) 跳数(hops):P-Rule 查询成功的平均节点数.这里的查询成功包括返回值为 *NULL* 的情况;

(2) 作业平均响应时间(average response time,简称 ART):作业响应时间指的是网格作业从提出文件请求直到获得所有所需文件的时间.若请求的文件存储在本地,则认为响应时间小到可以忽略不计;反之,则需要从其他节点获取所需文件的副本.这时,文件传输时间占据了响应时间的绝大部分,设定这种情况下的响应时间即为文件传输时间,其值为文件大小和节点间带宽的比值.所有作业的平均响应时间称为作业平均响应时间;

(3) 更新流量(update traffic):P-Rule 更新时在 Chord 环上路由的总的消息数目.更新流量越大,网络负载越大.

实验 1 测量分别采用 LastPre 和 FullPre 两种存储方式时,P-Rule 查询在不同节点数和查询模式下的跳数.模拟生成节点数为 k 的 Chord 网络,k 取值由 100 增大到 1 000,如图 4 所示.针对每个生成的 Chord 网络,将收集的预取规则分别以 LastPre 和 FullPre 两种方式进行存储.P-Rule 查询采用随机查询,查询请求的发生服从泊松随机过程,其平均到达间隔随着模拟节点数的增加由 0.1ms 递减到 0.01ms,其中,节点为 1 000 时取 0.01ms.节点负载用每秒处理的 P-Rule 查询请求进行衡量,实验中,若超过 80q/s 则引发负载转移.负载转移比率δ设为 0.2.由第 3.3 节可知,P-Rule 查询根据查询前缀数目的多少分为单前缀查询和多前缀查询两种情况.为此,我们设置了两种查询模式:一种是单前缀查询模式,其所包含的查询都是单前缀查询;另一种是混合查询模式,其所包含的查询可能是单前缀查询,也可能是多前缀查询,并通过设定混合比例参数θ来调节两者所占比例,θ表示多前缀查询在所有查询中所占的百分比.图 4 显示的是 LastPre 和 FullPre 在单前缀查询模式下的跳数,可以看到,两者相差不大.这是因为在单前缀查询模式下,即使采用 FullPre 存储方式也不会出现本文第 3.1 节所述的预取规则无序存放而导致的查询代价增大的问题.图 5 则显示了 LastPre 和 FullPre 在混合查询模式下的跳数,其中,θ设为 0.5.由图 5 可以看出,在混合查询模式下,LastPre 相对于 FullPre 具有更好的 P-Rule 查询性能,在不同节点数的情况下,跳数都得到了有效降低,最多的时候降低了约 38%,这表明 LastPre 较 FullPre 更适合于多前缀查询.

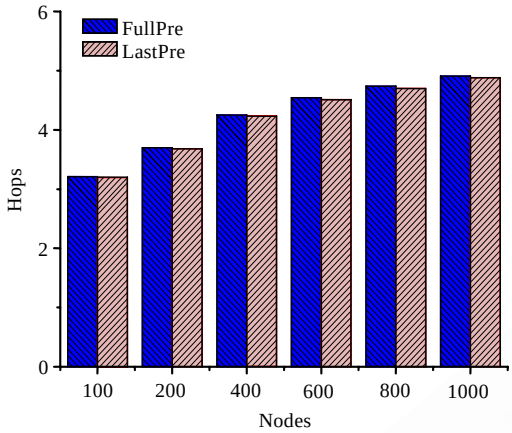


Fig.4 Hops of FullPre and LastPre under the one-prefix query pattern

图 4 单前缀查询模式下 FullPre 和 LastPre 跳数

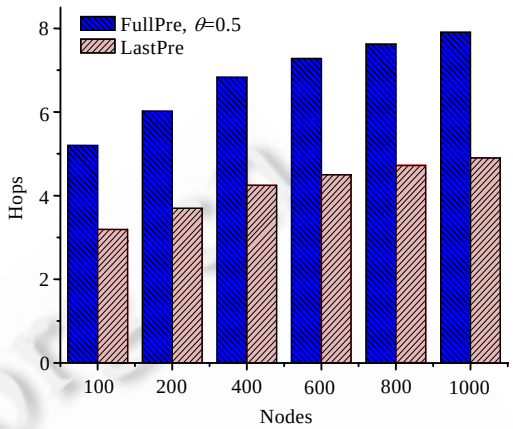


Fig.5 Hops of FullPre and LastPre under the mixed-query pattern, θ=0.5

图 5 混合查询模式下 FullPre 和 LastPre 跳数

实验 2 通过改变混合比例参数θ以更加细致地考察 LastPre 和 FullPre 在混合查询模式下的查询性能.采用跳数差(hop difference)作为性能指标,具体由 FullPre 对应的跳数减去 LastPre 对应的跳数得到.图 6 显示了节点数分别为 100 和 400 时两种存储方式在不同θ取值下的跳数差.实验中,P-Rule 查询请求的发生仍服从泊松随机过程,到达间隔的设置同实验 1.由图 6 可以看出,无论节点数取 100 还是 400,随着θ的增大,即多前缀查询在查询中所占的比重越来越大,LastPre 和 FullPre 的跳数差逐渐增大.这进一步证明了 LastPre 在多前缀查询中所具有的优势.事实上,对于前缀个数为 n 的多前缀查询,若采用 FullPre 存储方式,最坏情况下需要 n 次路由,而 LastPre 存储方式得益于预取规则树的设计,只需要一次路由.需要说明的是,实验中生成的多前缀查询大多只包含两个前缀,可以预见,若是采用前缀个数更多的查询,比如 3 个乃至 4 个前缀的查询,两者的跳数差会进一步拉大.鉴于多前缀查询是 P-Rule 查询中经常遇到的情况,我们认为,基于预取规则树的 LastPre 存储方式可以更好地支持 P-Rule 查询.在随后的实验中若无特殊说明,预取规则都采用 LastPre 存储方式.

实验 3 测试 P-Rule 更新消息打包发送方式的性能,采用更新流量作为性能指标.由第 3.2 节可知,副本协作预取每隔 t_u 时间对预取规则的逻辑支持度计算并更新,因此,每隔 t_u 时间 Chord 环上都会出现大量的 P-Rule 更新消息需要路由.t_u 的时间长短会影响 P-Rule 更新消息的数量,若设置较短,则单次更新的消息数可能较少;反之,单次更新的消息数可能相对较多,短时间内对网络负载影响更大.实验 3 着重考虑的是单次预取规则更新所导致的消息数目的爆发性增长可能对系统网络负载带来的影响,其目标是希望通过打包发送方式减少短时间内出现的过多的消息数,以降低网络负载.在我们之前的工作中,t_u 设定为 8 小时.由于两次更新的间隔较长,可以看作相互独立,因此,实验 3 中我们只模拟单次 P-Rule 更新时的场景以测试打包发送方式的性能.基于上述考虑,生成节点数为 100 的 Chord 网络,每个节点生成 500 条更新消息,每条更新消息随机选取一条预取规则作为更新内容.采用两种更新消息发送方式:一种为非打包方式,即每条更新消息都通过 Chord 环独立路由;另一种为打包发送方式.图 7 显示了打包的消息数 k 不同取值时的更新流量(已作归一化处理),其中,k=0 表示不打包的方式.由图 7 可以看出,打包发送方式极大地降低了更新流量,降幅最大达到了 83%.而 k>10 后,降幅的增大并不明显且趋于稳定,这是因为发往同一个节点的更新消息数是有限的,k 增大到一定程度后,即使再增加 k 的值也不会减少打包后形成的消息数目.

实验 4 测试 PTree 负载转移时引入的性能开销.首先测试带宽消耗.为简单起见,我们假设每棵 PTree 大小相同,在负载转移时消耗相同的带宽.基于上述假设,采用参与负载转移的 PTree 总数来衡量负载转移的带宽消耗.随机选取 10%的预取规则,通过赋予其较大的查询频度生成热点查询,其余的预取规则仍采用随机查询.负载转

移参数设置与实验 1 相同.图 8 显示了节点数为 100、查询请求到达间隔为 0.1ms 情况下,模拟运行 30s 不同查询频度下 PTree 的转移总数.可以看出,随着热点查询频度的增加,PTree 转移总数也在增加.需要说明的是,PTree 存储的只是一些⟨名,值⟩对,且其大小受到预取规则的前后缀长度的限制,其中,前缀的长度决定了 PTree 的层数.受制于预取准确率,前缀长度一般不会太长,在我们提取的预取规则中,前缀长度超过 4 的预取规则数量很少.基于上述原因,PTree 转移时并不会消耗较多带宽.其次测试转发开销对 P-Rule 查询的影响.图 9 显示了 150 个节点、到达间隔 0.1ms 情况下,系统前 20%高负载节点平均每秒处理完成的 P-Rule 查询数.随着查询频度增加,高负载节点能够处理完成的查询数有所降低,这主要是由转发处理的查询消息数量增加所导致.由第 3.3 节可知,节点接收到 P-Rule 查询请求后,会判别待查询的 PTree 是否存储在本地节点.若存储在本地节点,则查询本地的 PTree;否则,将查询消息转发到该 PTree 的当前维护节点.查询消息的转发处理需要占用一部分负载,因此,热点查询频度增大时,即使查询的 PTree 已经迁移,节点仍需要处理较多的查询转发消息,从而影响了本地 PTree 查询的处理效率.对于上述问题,可以考虑在查询路径上引入缓存加以解决,这也是我们下一步需要改进的地方.

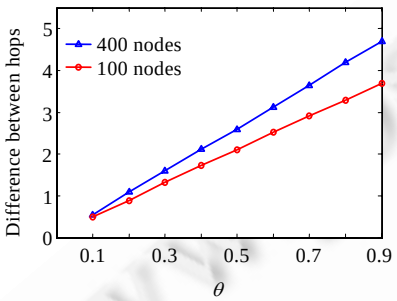


Fig.6 Effect of θ on hop difference

图 6 θ不同取值时的跳数差

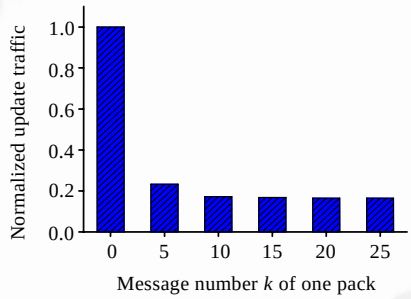


Fig.7 Update traffic of packing method

图 7 打包方式的更新流量

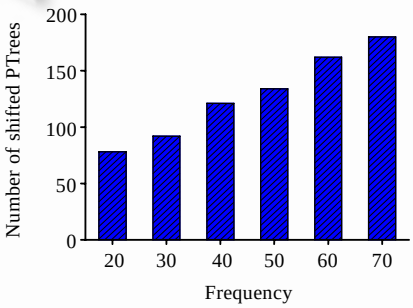


Fig.8 Number of shifted PTrees

图 8 PTree 转移总数

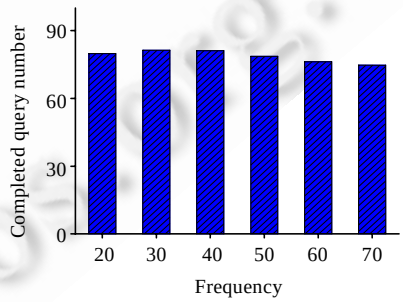


Fig.9 Throughput of P-Rule query

图 9 P-Rule 查询吞吐率

实验 5 通过回放测试集中的作业记录验证基于 DHT 的副本协作预取对网格数据访问延迟的影响.采用作业平均响应时间作为性能评测指标.首先构建 Chord 环,环中节点对应于我们采集作业记录的虚拟组织中的真实节点.作业回放过程中采用两种副本复制方式:一种是仅采用文件访问需求驱动的副本复制策略,另一种是协作预取和文件访问需求驱动相结合的复制策略.回放过程中,若某个节点试图获取某文件相关的隐性高价值文件副本,则通过 Chord 环上的对应节点发出 P-Rule 查询请求,以此启动协作预取.关于文件访问需求驱动的复制策略,我们实现了一种目前普遍应用的策略,该策略实现简单,描述为:当某个网格节点对同一个文件的请求次数达到阈值时,就在该节点上创建该文件的副本.此外,实验中设定每个网格节点提供 20GB 的存储空间,任意两节点连通且带宽默认为 100Mbps.图 10 表示模拟运行 100,200 直到 1 000 个作业后,两种副本复制方式所对应

的作业平均响应时间,其中,公式(2)中的λ分别取 0.1,0.2,0.25,λ=0 则表示仅采用文件访问需求驱动的复制策略.由图 9 可以看出,λ取值为 0.1,0.2 或 0.25 这 3 种情况时,作业平均响应时间相对于λ=0 时都得到了有效降低,这说明采用协作预取和文件访问需求驱动结合的副本复制方式确实可以进一步降低数据访问延迟.再从λ单个取值时对应的情况看,λ=0.2 时的降幅较λ=0.1 时的降幅有明显提高,但λ增大到 0.25 时,降幅提高的幅度很小,这表明通过增大预取池空间来提高协作预取性能是有限度的.事实上,预取池的空间不宜过大.一方面是因为副本预取的命中率受限;另一方面,若预取的副本过多可能反而会影响系统原有的需求访问驱动的副本复制,这里的影响主要是指带宽资源和存储空间的竞争等.这也是实验 5 中λ的取值不大于 0.25 的原因.综合考虑,本实验中λ取 0.2 较为理想.

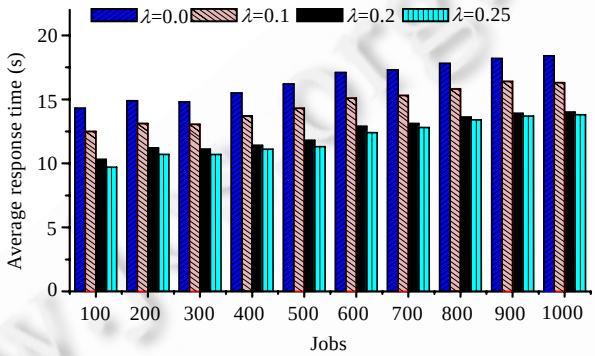


Fig.10 Effect of λ on average response time

图 10 λ不同取值时的作业平均响应时间

实验 5 证明了采用副本协作预取确实可以进一步降低作业平均响应时间,但是这也带来了存储资源和带宽资源的消耗.前者可以通过设置预取池的空间大小进行控制,后者则主要是由预取所引发的数据传输所造成.为了优化带宽资源的利用率,我们提供了两个解决方式:一是选择合适的隐性高价值文件副本传输的时机.隐性高价值文件面向的是未来一段时间文件访问的需求,因此并不一定需要即刻复制到本地节点,可以延迟复制过程直到网络较为空闲的时段(例如夜间时刻)再进行复制,而在网络较忙时暂缓复制;二是采用数据压缩技术,这不仅可以降低存储空间的消耗,也可以在数据传输时消耗更少的带宽资源.特别是一些半结构化的数值型数据,具有很好的压缩效果.目前,大多数网格平台,尤其是针对科学应用的网格平台,都具有较为充裕的带宽资源[26],SEUGRID 同样如此,校园网乃至数据传输专线的使用为带宽资源的使用提供了充分保障.因此,副本协作预取以额外带宽消耗换取作业响应时间降低的方法是可行的.

5 结束语

副本协作预取是一种新型的网格副本复制策略,通过获取隐性高价值文件进一步降低了网格中的数据访问延迟.副本协作预取的关键在于隐性高价值文件的确定和获取.本文利用 DHT 技术组织网格节点,以快速定位隐性高价值文件查询所必需的文件相关性信息,达到加速获取隐性高价值文件的目的.针对隐性高价值文件的特殊查询模式,提出了基于预取规则树的文件相关性信息存储方式 LastPre,提高了查询效率,并给出了与存储方式相对应的文件相关性信息的查询机制.模拟实验结果表明,对于文件相关性信息查询,LastPre 较传统 DHT 存储方式具有更好的查询性能.

在下一步的工作中,我们会研究如何在云计算数据中心环境下运用协作预取技术以降低作业的执行时间.

在数据中心环境下,由于新型网络架构和路由技术的采用,磁盘读取速度取代网络带宽成为了制约作业执行时间长短的重要因素[27],这时,可以通过将数据存储于内存中以加快作业执行,为此,需要研究内存的协作预取技术.

References:

[1] Pacitti E, Valduriez P, Mattoso M. Grid data management: Open problems and new issues. Journal of Grid Computing, 2007,5(3): 273−281. [doi: 10.1007/s10723-007-9081-9]

[2] Venugopal S, Buyya R, Ramamohanarao K. A taxonomy of data grids for distributed data sharing, management, and processing. ACM Computing Surveys, 2006,38:Article 3. [doi: 10.1145/1132952.1132955]

[3] Foster I, Kesselman C, Tuecke S. The anatomy of the grid: Enabling scalable virtual organization. Int'l Journal of High Performance Computing Applications, 2001,15(3):200−222. [doi: 10.1177/109434200101500302]

[4] Tian T, Luo JZ, Song AB, Wu ZA. Cooperative replica prefetching mechanism for virtual organization in grid. Ruan Jian Xue Bao/Journal of Software, 2011,22(10):2370−2382 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/3925.htm [doi: 10.3724/SP.J.1001.2011.03925]

[5] Ranganathan K, Foster I. Identifying dynamic replication strategies for a high performance data grid. In: Lee CA, ed. Proc. of the Int'l Grid Computing Workshop. LNCS 2242, London: Springer-Verlag, 2001. 75−86. http://dl.acm.org/citation.cfm?id=652840

[6] Carman M, Zini F, Serafini L, Stockinger K. Towards an economy-based optimization of file access and replication on a data grid. In: Proc. of the 2nd IEEE/ACM Int'l Symp. on Cluster Computing and the Grid (CCGRID 2002). Washington: IEEE Computer Society, 2002. 340−345. [doi: 10.1109/CCGRID.2002.1017156]

[7] Ranganathan K, Foster I. Decoupling computation and data scheduling in distributed data-intensive applications. In: Proc. of the 11th Int'l Symp. on High Performance Distributed Computing (HPDC 2002). Washington: IEEE Computer Society, 2002. 352−358. [doi: 10.1109/HPDC.2002.1029935]

[8] Sato H, Matsuoka S, Endo T, Maruyama N. Access-Pattern and bandwidth aware file replication algorithm in a grid environment. In: Proc. of the 9th IEEE/ACM Int'l Conf. on Grid Computing (GRID 2008). Washington: IEEE Computer Society, 2008. 250−257. [doi: 10.1109/GRID.2008.4662806]

[9] Rahman RM , Barker K, Alhajj R. Study of different replica placement and maintenance strategies in data grid. In: Proc. of the 7th Int'l Symp. on Cluster Computing and the Grid. Washington: IEEE Computer Society (CCGRID 2007). 2007. 171−178. [doi: 10.1109/CCGRID.2007.111]

[10] Rahman RM, Barker K, Alhajj R. Replica placement design with static optimality and dynamic maintainability. In: Proc. of the 6th IEEE Int'l Symp. on Cluster Computing and the Grid (CCGRID 2006). Washington: IEEE Computer Society, 2006. 434−437. [doi: 10.1109/CCGRID.2006.85]

[11] Xiao N, Fu W, Lu XC. QoS-Awared replica placement techniques in data grid applications. Science in China (Series F), 2009, 39(10):1063−1071 (in Chinese).

[12] Cheng CW, Wu JJ, Liu PF. QoS-Aware, access-efficient and storage-efficient replica placement in grid environments. The Journal of Supercomputing, 2009,49(1):42−63. [doi: 10.1007/s11227-008-0221-1]

[13] Iamnitchi A, Doraimani S, Garzoglio G. Filecules in high-energy physics: Characteristics and impact on resource management. In: Proc. of the 15th ACM Int'l Symp. on High Performance Distributed Computing (HPDC 2006). 2006. 69−80. [doi: 10.1109/HPDC.2006.1652137]

[14] Doraimani S, Iamnitchi A. File grouping for scientific data management: Lessons from experimenting with real traces. In: Proc. of the 17th IEEE Int'l Symp. on High Performance Distributed Computing (HPDC 2008). New York: ACM Press, 2008. 153−164. [doi: 10.1145/1383422.1383429]

[15] Iamnitchi A, Doraimani S, Garzoglio G. Workload characterization in a high-energy data grid and impact on resource management. Journal of Cluster Computing, 2009,12(2):153−173. [doi: 10.1007/s10586-009-0081-3]

[16] Sato H, Matsuoka S, Endo T. File clustering based replication algorithm in a grid environment. In: Proc. of the 9th IEEE/ACM Int'l Symp. on Cluster Computing and the Grid (CCGRID 2009). Washington: IEEE Computer Society, 2009. 204−211. [doi: 10.1109/CCGRID.2009.73]

[17] Abdurrab AR, Xie T. FIRE: A file reunion based data replication strategy for data grids. In: Proc. of the 10th IEEE/ACM Int'l Conf. on Cluster, Cloud and Grid Computing (CCGrid 2010). Washington: IEEE Computer Society, 2010. 215−223. [doi: 10.1109/CCGRID.2010.12]

[18] Ban ZJ, Gu ZM, Jin Y. A survey of Web prefetching. Journal of Computer Research and Development, 2009,46(2):202−210 (in Chinese with English abstract).

[19] Salton G, Wong A, Yang CS. A vector space model for automatic indexing. Communications of the ACM, 1975,18(11):613−620. [doi: 10.1145/361219.361220]

[20] Stoica I, Morris R, Liben-Nowell D, Karger D, Kaashoek M, Dabek F, Balakrishnan H. Chord: A scalable peer-to-peer lookup protocol for Internet applications. IEEE/ACM Trans. on Networking, 2003,11(1):17−32. [doi: 10.1109/INFCOM.2004.1354648]

[21] Chu R, Lu XC, Xiao N. A data prefetching algorithm for RAM grid. Ruan Jian Xue Bao/Journal of Software, 2006,17(11): 2234−2244 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/17/2234.htm [doi: 10.1360/jos172234]

[22] Godfrey B, Lakshminarayanan K, Surana S, Karp R, Stoica I. Load balancing in dynamic structured P2P systems. In: Proc. of the 23rd Annual Joint Conf. of the IEEE Computer and Communications Societies (INFOCOM 2004). 2004. 2253−2262. [doi: 10.1109/INFCOM.2004.1354648]

[23] Chung WC, Chang RS. A new mechanism for resource monitoring in grid computing. Journal of Future Generation Computer Systems, 2009,25(1):1−7. [doi: 10.1016/j.future.2008.04.008]

[24] Montresor A, Jelasity M. PeerSim: A scalable P2P simulator. In: Proc. of the 9th Int'l Conf. on Peer-to-Peer Computing (P2P 2009). 2009. 99−100. [doi: 10.1109/P2P.2009.5284506]

[25] Luo JZ, Song AB, Zhu Y, Wang XP, Ma T, Wu ZA, Xu YB, Ge L. Grid supporting platform for AMS data processing. In: Chen G, Pan Y, Guo M, Lu J, eds. Proc. of the ISPA Workshops 2005. LNCS 3759, Berlin: Springer-Verlag, 2005. 276−285. [doi: 10.1007/11576259_31]

[26] Al-Kiswany S, Ripeanu M, Iamnitchi A, Vazhkudai S. Are P2P data-dissemination techniques viable in today's data-intensive scientific collaborations? In: Kermarrec AM, Bouge L, Priol T, eds. Proc. of the 13th European Int'l Conf. on Parallel Processing. LNCS 4641, Heidelberg: Springer-Verlag, 2007. 404−414. [doi: 10.1007/978-3-540-74466-5_44]

[27] Ananthanarayanan G, Ghodsi A, Shenker S, Stoica I. Disk-Locality in datacenter computing considered irrelevant. In: Proc. of the 13th USENIX Conf. on Hot Topics in Operating Systems (HotOS 2011). USENIX Association Berkeley, 2011. http://dl.acm.org/citation.cfm?id=1991613

附中文参考文献:

[4] 田田,罗军舟,宋爱波,伍之昂.网格虚拟组织副本协作预取机制.软件学报,2011,22(10):2370−2382. http://www.jos.org.cn/1000-9825/3925.htm [doi: 10.3724/SP.J.1001.2011.03925]

[11] 肖侬,付伟,卢锡城.数据网格中服务质量感知的副本放置方法.中国科学(F 辑:信息科学),2009,39(10):1063−1071.

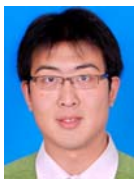
[18] 班志杰,古志民,金瑜.Web 预取技术综述.计算机研究与发展,2009,46(2):202−210.

[21] 褚瑞,卢锡城,肖侬.一种内存网格的数据预取算法.软件学报,2006,17(11):2234−2244. http://www.jos.org.cn/1000-9825/17/2234.htm

田田(1983－),男,江苏淮安人,博士生,主要研究领域为网格与云计算,分布式数据处理.
E-mail: tian_tian@seu.edu.cn

宋爱波(1970－),男,博士,副教授,CCF 会员,主要研究领域为网格与云计算,海量数据处理, Petri 网理论与应用.
E-mail: absong@seu.edu.cn

罗军舟(1960－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为下一代网络体系结构,协议工程,网络安全与管理,网格与云计算,无线局域网.
E-mail: jluo@seu.edu.cn



东方(1982－),男,博士,讲师,主要研究领域为网格与云计算,海量数据处理.
E-mail: fdong@seu.edu.cn