ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.19, No.3, March 2008, pp.744−754
DOI: 10.3724/SP.J.1001.2008.00744

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

一种无“热点”的覆盖网协同缓存策略*

李春洪[1,2+], 冯国富[1,2], 顾铁成[1,2], 陆桑璐[1,2], 陈道蓄[1,2]

[1](南京大学 计算机科学与技术系,江苏 南京 210093)

[2](南京大学 计算机软件新技术国家重点实验室,江苏 南京 210093)

A Hotspots-Free Overlay Cooperative Caching Scheme

LI Chun-Hong[1,2+], FENG Guo-Fu[1,2], GU Tie-Cheng[1,2], LU Sang-Lu[1,2], CHEN Dao-Xu[1,2]

[1](Department of Computer Science and Technology, Nanjing University, Nanjing 210093, China)

[2](State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210093, China)

+ Corresponding author: Phn: +86-25-58821506, Fax: +86-25-83300710, E-mail: lch@dislab.nju.edu.cn, http://cs.nju.edu.cn

Li CH, Feng GF, Gu TC, Lu SL, Chen DX. A hotspots-free overlay cooperative caching scheme. *Journal of Software*, 2008,19(3):744−754. http://www.jos.org.cn/1000-9825/19/744.htm

Abstract: Overlay Web caching (OCC) exploits resources of peers to provide scalable and cost-effective web caching service. In a typical OCC system, which is often characterized by highly heterogeneous node capacities and skewed query distributions, the resources of each node may be utilized in an unbalanced manner, i.e., some nodes are overloaded and become “hotspots”. Unfortunately, there are no effective load balancing mechanisms in existing OCC systems to relief the “hotspots”. This paper proposes a hotspots-free OCC scheme called HFOCC for multimedia content delivery service. Through replicating “hot” objects adaptively to lightly loaded nodes, loads are distributed more evenly across the whole network. Consequently, the hotspots are relieved. In order to utilize cache resource more effectively, HFOCC splits a node’s cache space dynamically into two parts, namely the home cache and the replica cache, and manages them by a uniform policy. With a “soft” lifetime control mechanism, the redundant object replicas are deleted adaptively, and the system performs well under dynamically changing workloads. Experimental results show that HFOCC improves resource utilization and system throughput markedly.

Key words: overlay cooperative caching; replication; load balancing; multimedia content delivery; replica lifetime management

摘 要: 覆盖网协同缓存(overlay cooperative caching,简称 OCC)聚集客户节点的资源来提供可扩展、经济有效的缓存服务.在典型的 OCC 系统中,节点的异构性和工作负载的不对称,容易造成节点资源使用的不平衡,形成一些负载过重的“热点”节点.但对于这一问题,已有的 OCC 系统缺乏有效的负载平衡机制.针对多媒体内容分发服务,设计了一种无热点的 OCC 策略——HFOCC(hotspots free overlay cooperative caching).通过将“热点”对象复制到低负载节点,分散服务请求,达到消除热点的目的.为了提高缓存空间的利用率, HFOCC 将一个节点的缓存空间动态地划分

* Supported by the National Natural Science Foundation of China under Grant Nos.60573106, 60402027, 60573131 (国家自然科学基金); the Natural Science Foundation of Jiangsu Province of China under Grant No.BK2005411 (江苏省自然科学基金); the National Basic Research Program of China under Grant No.2002CB312002 (国家重点基础研究发展计划(973))

Received 2006-01-10; Accepted 2006-12-06

为 home cache 和 replica cache 两部分,并实施统一的缓存管理策略;基于一种“软”副本生命期控制机制,当工作负载发生变化时,冗余副本被及时删除,系统表现出了良好的自适应性.实验证明,HFOCC 有效地提高了系统吞吐率和资源利用率.

关键词: 覆盖网协同缓存;复制;负载平衡;多媒体内容分发;副本生命期管理

中图法分类号: TP316 **文献标识码:** A

近年来,在 P2P 和覆盖网等方面取得的研究进展,为构建经济有效的 Web 缓存系统提供了新的解决方案:各客户节点贡献出一定的存储和带宽资源,并以点到点的方式共享这些资源,形成了一种自组织的、结构灵活的覆盖网协同缓存(overlay cooperative caching,简称 OCC)系统.OCC 聚合闲置的客户端资源来提供缓存服务,无须额外的硬件投入和管理成本,且其服务能力随着用户数量的增加而提高,具备良好的可扩展性.这种新型的缓存系统引起了很多研究者的关注,也出现了若干实验型的 OCC 系统,如 Squirrel[1],BuddyWeb[2],Kache[3]等.研究表明,Web 服务系统的工作负载呈明显的不对称性,系统中各对象的请求率符合典型的类 Zipf 分布,即大部分用户请求集中在少数热门对象上[4].与专用缓存服务器相比,OCC 节点的服务能力较低且存在较大的异构性[5,6].缓存了热门对象的节点由于接收到较多的服务请求而进入过载状态,成为“热点”;其他节点的负载则相对不足,造成了资源的闲置.节点的异构性使得这个问题更为突出.如果系统存在热点,部分缓存命中的请求可能被拒绝(或得不到正常响应),致使系统吞吐量降低.为提高系统的资源利用率和吞吐率,应尽量消除热点.但已有的 OCC 系统并没有引入有效的负载平衡机制来解决这一问题.

本文针对多媒体内容分发服务,提出了一种能实现节点负载平衡的覆盖网协同缓存系统——HFOCC (hotspots free overlay cooperative caching).HFOCC 能够自适应地将热门对象复制到低负载的节点,将负载较为合理地分散到多个节点,达到消除热点的目的.

本文第 1 节介绍相关的研究工作.第 2 节给出系统模型和相关定义,证明基于复制的负载平衡机制的有效性.第 3 节给出 HFOCC 系统的详细设计.第 4 节进行仿真实验和比较分析.第 5 节对全文进行总结.

1 相关工作

传统 Web 缓存系统利用部署于 ISP 或企业网边缘的专用代理服务器来提供服务,存在部署成本高、管理开销大和可扩展性差等缺点[7].OCC 聚合闲置的客户资源来提供缓存服务,基于 P2P 定位服务实现缓存查找,较好地克服了传统缓存系统的缺点.Squirrel[1]是一种全分布的 P2P Web 缓存系统,它以 Pastry[8]作为缓存定位基础设施,在客户主机间实现对普通 Web 对象的共享.BuddyWeb[2]利用定制的 P2P 系统 BestPeer 来组织客户节点,并采用基于内容语义相似度分析的路由和缓存策略.Kache[3]是一种构建在 Kelips 之上的协同缓存系统,具有良好的抗攻击力和网络邻近性.PROP[9]是一种由客户端参与的流媒体代理缓存系统,它利用代理服务器来提供可靠的流服务,而以 P2P 方式组织的客户节点则有效地提高了系统的可扩展性.但以上这些 OCC 系统都忽略了热点问题,也没有引入相应的负载平衡机制.

结构化 P2P 在节点和对象之间建立了一定的映射关系,形成了一个覆盖网络拓扑(如 ring,mesh)作为信息查找和路由的基本结构[10],可在有限跳数内定位一个对象.已有的结构化 P2P 系统如 Chord[11],CAN[12]和 Pastry[8]等,忽略了节点异构性及对象热门度的差异,并假设对象均匀地分布于各节点.但文献[13]表明,简单地借助 hash 函数并不能实现完全的负载平衡.而且在典型 P2P 网络中,节点异构性和工作负载的不对称是不可忽略的两个因素[4−6].为保证这些系统在实际应用中发挥有效的作用,就必须引入负载平衡机制.

一种实现负载平衡的方法是根据节点能力动态地分配赋予节点的 ID 区间.CFS/Chord[11,14]提出了虚拟服务器(virtual server,简称 VS)的概念,并用它来解决负载平衡问题.每个 VS 都拥有一个合法的 Chord ID,并负责一个连续的 ID 区间.根据实际的存储和网络能力,管理服务器为各节点配置相应数量的 VS.当负载发生变化时,节点间通过移动 VS 来交换负载.这种方法的优点是可以最大限度地利用节点的资源,缺点是负载交换可能导致较大网络负载.

复制和缓存也是一种常见的负载平衡机制.PAST[15]和 CFS[14]采用一种端到端的缓存机制来分散“热点”:回传

路径中的各节点都对请求对象进行缓存,后继请求有可能在路由的中间节点得到满足,缓解了源节点的负载压力.文献[16]提出了一种轻量级的自适应复制协议 LAR.节点接收到服务请求时,根据自身的状态和请求节点的负载水平,决定是否在请求节点创建对象副本,并采用“捎带(piggybacking)”复制策略,避免了额外的复制开销.但在 OCC 环境中,节点的缓存空间是一种有限的资源,以上基于复制的负载平衡机制没有对缓存空间的使用进行限制和管理,且缺乏有效的副本生命期管理,系统中可能存在大量的冗余副本,导致缓存使用率较低.因此,它们并不能完全有效地解决 OCC 系统的负载平衡问题.HFOCC 也采用了类似于 LAR 的“捎带”复制策略,但引入了缓存空间管理和副本生命期管理机制,提高了系统吞吐率和资源效率.

2 模型、定义和分析

2.1 系统模型

HFOCC 的目标应用环境为拥有 100~10 000 台客户机的企业网或校园网,节点间有良好的连接,用户具有相近的访问兴趣.假设所有客户节点都具备直接访问 Internet 的能力,它们贡献出一部分存储、带宽和 CPU 等资源并允许其他节点共享,如图 1 所示.当用户请求服务时,首先在覆盖网络内查找所需的对象.如果命中,则用户请求被转发到缓存了该对象的节点;否则,用户从源服务器获得服务,HFOCC 根据缓存算法确定是否缓存该对象. HFOCC 以 Squirrel 的 Home Store 为基本缓存放置模式并进行了扩展.各节点和对象通过一个 hash 函数映射到同一个 ID 空间.每个活动的 HFOCC 节点都分担部分 ID 区域,并负责缓存映射到该区域的对象,成为这些对象的 home 节点.热门对象可以被复制到其他低负载节点以分担 home 节点的负载.为此,HFOCC 对 Home Store 模式进行了扩展:节点的缓存空间被动态地划分为 home cache 和 replica cache 两部分,分别存放 home 对象及其他对象的副本,并维护了一些特殊的数据结构以实现缓存路由和副本调度.

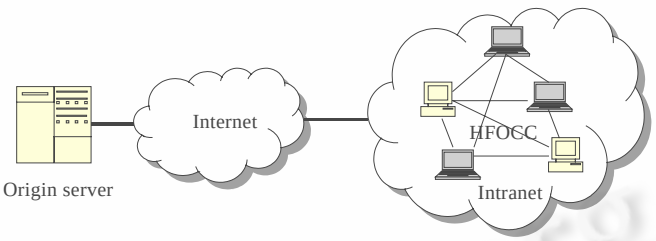


Fig.1 System model of the HFOCC

图 1 HFOCC 的系统模型

2.2 定义和说明

设组成 HFOCC 系统的客户节点集合为 $H=\{n_1,n_2,\ldots,n_N\}$,其中,N 为节点总数.节点 n_j 的资源能力可用一个二元组$\langle S_j,C_j\rangle$刻画,其中,S_j 为用于共享的存储空间大小,C_j 表示节点的服务能力.通常,节点的流媒体服务能力由 CPU 性能、内存大小和出口带宽等因素共同决定.但相对于 CPU 性能和内存资源来说,节点带宽能力提高滞后,而且昂贵得多.在 P2P 应用中,用户通常会对出口带宽设置使用上限.因此,HFOCC 假定节点的出口带宽是唯一的服务瓶颈.源服务器存放了完整的对象集 $O=\{o_1,o_2,\ldots,o_L\}$,L 为对象的总数.对象 o_i 可用一组属性值描述,包括大小、编码率、播放时间和请求率,分别用符号 s_i,b_i,d_i 和 λ_i 表示.设各对象具有相同的大小、编码率和播放时间,即$\forall o_i\in O$,有 $s_i=s$,$b_i=b$,$d_i=d$.

经过一段时间运行后,部分后续请求可以直接在 HFOCC 获得服务.由于请求率分布得不对称,单位时间到达各节点的请求数量存在很大的差异.为了度量节点的负载水平,有以下定义:

定义 1(对象 o_i 在节点 n_j 的负载). 在 t 时刻,如果节点 n_j 缓存了对象 o_i,则 n_j 应提供一定的资源来为对该对象的请求提供服务.我们称该资源量为对象 o_i 在节点 n_j 的负载,并用符号 $l_{i,j}^t$ 表示.

设 t 时刻在节点 n_j 对对象 o_i 的请求率为$\lambda_{i,j}$,则对象 o_i 在节点 n_j 的负载为

$$l_{i,j}^t=\lambda_{i,j}\cdot d_i\cdot b_i=\lambda_{i,j}\cdot d\cdot b \tag{1}$$

定义 2(节点负载). 节点 n_j 的负载 l_j^t 为其缓存的全部对象的负载之和.

设 t 时刻,节点 n_j 的缓存对象集为 Φ_j^t ($\Phi_j^t\subseteq O$),对对象 o_i 的请求到达率为 $\lambda_{i,j}$,则节点 n_j 的负载为

$$l_j^t=\sum_{\forall o_i\in\Phi_j^t}l_{i,j}^t=d\cdot b\cdot\sum_{\forall o_i\in\Phi_j^t}\lambda_{i,j} \tag{2}$$

引入高、低两个负载阈值 T_h 和 T_l($0\le T_l<T_h\le 1$)来刻画节点的负载状态.

定义 3(节点过载). t 时刻,如果节点 n_j 的负载 $l_j^t>T_h\cdot C_j$,则称 n_j 过载.

定义 4(节点欠载). t 时刻,如果节点 n_j 的负载 $l_j^t\le T_l\cdot C_j$,则称 n_j 欠载.

2.3 问题分析

为了便于分析,考察只包含两个节点的 OCC 系统.设时刻 t,节点 n_1 和 n_2 维护的缓存集分别为 $\Phi_1^t=\{o_{11},o_{12},...,o_{1M_1}\}$ 和 $\Phi_2^t=\{o_{21},o_{22},...,o_{2M_2}\}$,其中,$M_1=|\Phi_1^t|$,$M_2=|\Phi_2^t|$,且 $\Phi_1^t\cap\Phi_1^t=\varnothing$.设对象请求率在一段时间内保持稳定,不失一般性,令 $\lambda_{11,1}\ge\lambda_{12,1}\ge...\ge\lambda_{1M_1,1}$,$\lambda_{21,2}\ge\lambda_{22,2}\ge...\ge\lambda_{2M_2,2}$,则 t 时刻节点的负载分别为

$$l_1^t=(\lambda_{11,1}+\lambda_{12,1}+...+\lambda_{1M_1,1})\cdot d\cdot b \tag{3}$$

$$l_2^t=(\lambda_{21,2}+\lambda_{22,2}+...+\lambda_{2M_2,2})\cdot d\cdot b \tag{4}$$

设此时 n_1 过载,n_2 处于欠载状态,即 $l_1^t>T_h\cdot C_1$,$l_1^t<T_l\cdot C_1$.令 $\Delta l_1^t=l_1^t-T_h\cdot C_1$ 为 n_1 的实际负载超出其能力的部分.单位时间内,在节点 n_1 命中的请求数为 $\lambda_{11,1}+\lambda_{12,1}+...+\lambda_{1M_1,1}$,其中,被接纳的请求数为 $\frac{T_h\cdot C_1}{d\cdot b}$,被拒绝的请求数为 $\frac{\Delta l_1^t}{d\cdot b}$.而在 n_2 命中全部请求都将被接纳,单位时间内接纳的请求数为 $\lambda_{21,2}+\lambda_{22,2}+...+\lambda_{2M_2,2}$.令总的请求到达率为 λ,则 t 时刻系统吞吐率 η^t 和请求命中率 HR^t 分别为

$$\eta^t=\frac{T_h\cdot C_1+l_2^t}{d\cdot b} \tag{5}$$

$$HR^t=\frac{1}{\lambda}(\lambda_{11,1}+\lambda_{12,1}+...+\lambda_{1M_1,1}+\lambda_{21,2}+\lambda_{22,2}+...+\lambda_{2M_2,2}) \tag{6}$$

系统的资源利用率为

$$U^t=\frac{T_h\cdot C_1+l_2^t}{C_1+C_2} \tag{7}$$

设造成节点 n_1 过载的原因是因为缓存了热门对象 o_{11},且 $l_{1,1}^t>\Delta l_1^t$.t_1 时刻,在节点 n_2 创建 o_{11} 的副本,单位时间向 n_2“分流” $\frac{\Delta l_1^t}{d\cdot b}$ 个对 o_{11} 的请求,n_1 的负载将下降至 $T_h\cdot C_1$,过载状态得以缓解.为了给副本腾出放置空间,n_2 逐出 o_{2M_2} (设 $\frac{\Delta l_1^t}{d\cdot b}>\lambda_{2M_2,2}$).$n_2$ 的负载变化为

$$l_2^{t_1}=(\lambda_{21,2}+\lambda_{22,2}+...+\lambda_{2(M_2-1),2})\cdot d\cdot b=l_2^t+\left(\frac{\Delta l_1^t}{d\cdot b}-\lambda_{2M_2,2}\right)>l_2^t \tag{8}$$

命中率和吞吐率变化为

$$\eta^{t_1}=\frac{T_h\cdot C_1+l_2^{t_1}}{d\cdot b}>\frac{T_h\cdot C_1+l_2^t}{d\cdot b}>\eta^t \tag{9}$$

$$HR^{t_1}=\frac{1}{\lambda}(\lambda_{11,1}+\lambda_{12,1}+...+\lambda_{1M_1,1}+\lambda_{21,2}+\lambda_{22,2}+...+\lambda_{2(M_2-1),2})=HR^t-\frac{1}{\lambda}\cdot\lambda_{2M_2,2}<HR^t \tag{10}$$

此时,系统资源利用率为

$$U^{t_1}=\frac{T_h\cdot C_1+l_2^{t_1}}{C_1+C_2}>\frac{T_h\cdot C_1+l_2^{t}}{C_1+C_2}=U^{t} \tag{11}$$

以上分析表明,通过在低负载节点创建热门对象的副本,有效地消除了热点,系统吞吐率和资源利用率也得到了提高.由于副本占用了缓存空间,HFOCC 的缓存命中率有一定程度的降低(见式(10)).事实上,当系统存在热点时,部分命中的服务请求被拒绝,命中率指标并不真正反映系统的性能.而且,缓存替换算法可以保证逐出最小请求率的对象来放置副本,因此,损失的命中率被控制在最低限度.损失少量的命中率,换取吞吐率和资源利用率的提高是值得的.本文将吞吐率和资源利用率作为考察 OCC 系统性能的主要指标.

3 HFOCC 的设计

3.1 缓存空间管理

活动的 HFOCC 节点除了成为部分对象的 home 节点以外,还可能成为其他对象的副本持有者.为了便于管理和定位,将节点的缓存空间区分为 home cache 和 replica cache,分别存放 home 对象和副本对象.二者动态地共享缓存空间,并采取统一的缓存管理策略,如图 2 所示.各 home 对象都对应一个 home cache index 项,它包含两个指针,一个用于指示对象的存储地址,另一个指向该对象的 Replica finger.Replica finger 包含一组指向各副本持有节点的地址项.每个 replica cache index 项都指向一个对象副本.在替换策略上,HFOCC 采用 LFU-K[17]缓存替换算法,其优点是可以快速响应系统的负载变化,避免了 LFU 算法存在的“缓存污染”问题.

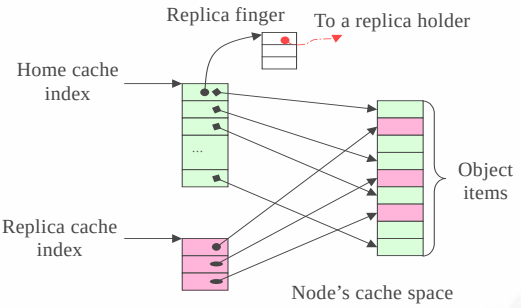


Fig.2 Division and management of cache space in a node

图 2 节点缓存空间的划分和管理

3.2 请求路由

当节点 n^r 请求对象 o 时,HFOCC 的处理过程如下:

(1) n^r 查找本地的 home cache index 和 replica cache index:

(1.1) 若请求在 home cache 命中,请求在本地得到满足;n^r 更新对象 o 的请求记录,处理结束;

(1.2) 若请求在 replica cache 命中,请求在本地得到满足;n^r 更新对象 o 的请求记录,并通知对象 o 的 home 节点 n^{home} 更新请求记录,处理结束;

(1.3) 若请求在本地不命中,则请求通过 DHT 设施被转发到节点 n^{home},转(2);

(2) 节点 n^{home} 更新对象 o 的请求记录,并查找 home cache index:

(2.1) 若请求命中,且 n^{home} 处于非过载状态,则请求被接纳,处理终止;

(2.2) 若请求命中,但 n^{home} 过载,则根据对象 o 的 Replica finger,依次联系各副本持有者:

(2.2.1) 若当前的副本持有者非过载,则该副本持有者更新对象 o 的请求记录,并接纳请求,处理结束;否则,n^{home} 继续探寻下一个副本持有者;

(2.2.2) 若所有的副本都无法接纳请求,请求被转发至源服务器,并执行复制算法,处理结束;

(2.3) 若请求不命中,则请求节点直接从源服务器获取服务.n^{home} 根据 LFU-K 缓存算法确定是否缓存

对象 o,处理结束.

3.3 复制策略

HFOCC 采用一种轻量级的、基于“捎带”的复制策略:请求的发起节点 n^r 在请求报文中附带负载状态信息;如果对象 o 为热门对象,且 home 节点 n^{home} 及所有副本持有节点均过载,则 n^r 从源服务器中获取服务,n^{home} 根据 n^r 的负载水平决定是否在该节点保留对象副本.在动态计算环境中,难以用一个静态标准来衡量对象热门与否,HFOCC 采用了一种简单而有效的判断方法:热点节点中请求率排名前 k(通常 k 取值为 3)的缓存对象为热门对象.

只有当以下 3 个条件同时满足时,HFOCC 才开始执行对象的复制操作.

条件 1:请求在节点 n^{home} 命中,但是,n^{home} 和所有的副本持有者都因为过载而拒绝接纳该请求;

条件 2:节点 n^r 处于欠载状态;

条件 3:在节点 n^{home} 中,对象 o 是请求率最高的 k 个对象之一.

复制过程描述为:

(1) n^{home} 向 n^r 返回一个“MISSING_AND_REPLICATE”消息;

(2) n^r 向源服务器请求对象 o;

(3) n^r 在其共享缓存空间建立对象 o 的副本.如果没有足够的缓存空间,则根据 LFU-K 缓存替换算法逐出一个对象以腾出放置空间;

(4) n^r 向 n^{home} 发送“REPLICA_CREATE_SUCCESS”消息;

(5) n^{home} 在对象 o 的 Replica finger 中添加一个指向 n^r 的项,过程结束.

3.4 副本生命期管理

根据第 3.3 节,热点节点所缓存的“热门”对象将被复制到其他节点,且热门度越高的对象被复制的机率越大.可以推测,当系统达到稳定时,对象的副本数量和它的流行度大致成比例.但由于 Internet 内容服务的工作负载具有动态性,随着时间的推移,对象的流行度可能发生改变.而且,在发生特殊事件(如“911”事件)时,还可能出现“突发的请求流(flash crowds)”,即大部分请求集中在极少数几个对象上.当系统工作负载动态变化或出现“突发流”时,新副本将不断地被创建,冗余副本却长时间得不到替换,造成“缓存污染”.HFOCC 采用了一种“软”副本生命期控制机制,由副本持有节点根据本地缓存策略自主地决定副本的生存期.基本思想是:通过合理的副本调度,保证冗余副本接收到的请求数量趋于 0,从而被替换出缓存空间,达到删除冗余副本的目的.

HFOCC 依据 Replica finger 项的排列次序来调度各副本,因此,通过对 Replica finger 的调整可以实现上述目标.对 Replica finger 的调整发生在副本创建或请求调度时.在 HFOCC 中,Replica finger 用链表结构表示.

副本创建时:n^{home} 收到来自 n^r 的“REPLICA_CREATE_SUCCESS”消息时,创建一个指向 n^r 的 Replica finger 条目,并将它插入到链表的首位.

请求调度时:回顾第 3.2 节的请求路由策略,请求首先被调度到 home 节点.如果 home 节点过载,则请求被依照 Replica finger 项的排列次序调度到各副本持有节点.如果某副本持有节点接纳了请求,则与之对应的 Replica finger 项被移动至链表的首位.在调度过程中,如果发现某个副本已经被替换,则删除该副本所对应的 Replica finger 条目.伪代码描述如下:

```
[1]  Replica_finger_entry=homenode.GetReplicaFingerEntry(o);
[2]  Temp=Replica_finger_entry;
[3]  while Temp!=NULL do
[4]      if Temp→holder.CurrentLoad!=OVERLOAD then
[5]          Temp→pre→next=Temp→next;
[6]          Temp→next→pre=Temp→pre;
[7]          Temp→next=Replica_finger_entry→next;
```

```
[8]          Replica_finger_entry→pre=Temp;
[9]          Replica_finger_entry=Temp;
[10]     else
[11]          Temp1=Temp→pre;
[12]          Temp1→pre→next=Temp1→next;
[13]          Temp1→next→pre=Temp1→pre;
[14]          Temp1→delete();
[15]     end if
[17]     Temp=Temp→next;
[16]  end while
```

3.5 节点动态性的处理

当节点加入或退出系统时,在 ID 空间上与之相邻的节点负责的 ID 区域将发生一定的变化,以它们为 home 的对象集也将发生相应的调整.因此,节点间必须通过交换缓存对象及相关的信息来适应这种变化.但是,节点间过多的对象交换将导致通信流量过大.HFOCC 采取了折衷方案:节点间交换全部相关的访问记录和 Replica finger 信息;而为了降低通信量,只交换热门缓存对象.

节点加入:节点 n^J 加入系统后,自动成为部分对象(用 $M^J(O)$,$M^J(O)\subseteq O$ 表示)的 home 节点.设 n^J 在 ID 空间上的邻居节点为 n^N.首次进入系统的节点没有维护任何缓存对象和相关信息;重新进入的节点,其保留的部分缓存对象可被重新利用,但是它维护的访问记录和 Replica finger 信息不完整并已过时,需进行更新.n^J 应尽快从 n^N 接管各 $M^J(O)$对象的访问记录和 Replica finger 信息,并预取 $M^J(O)$中的热门对象.

节点退出:正常退出的节点 n^D 在离开系统之前,将 $M^D(O)$中各对象的访问记录、Replica finger 信息和 home cache 中 K 个最热门的对象移交给邻居节点 n^J,但并不删除其缓存空间中的对象.

节点失效:如果节点 n^F 失效,其维护的缓存对象及相关信息就没有机会移交给 n^J,将造成缓存性能在一定程度上的下降.HFOCC 没有对节点失效进行专门的处理.因为随着后继请求的到达,$M^F(O)$中的热门对象将在 n^J 被缓存,相关信息可以得到重建.

4 性能评价

4.1 仿真环境设置

系统模型:客户节点总数为 200,节点的平均在线率为 80%.节点的出口带宽满足μ=15b,σ=2b 的正态分布.令源服务器中所有对象的大小总和为 S,相对缓存大小参数用于描述节点缓存空间总大小与 S 的比值.例如,相对缓存大小为 20%表明节点缓存空间的平均大小为 $\frac{20\%\cdot S}{N}$,而节点的实际缓存大小用 $\mu=\frac{20\%\cdot S}{N}$,$\sigma$=0.1·$\mu$的正态分布模型产生.

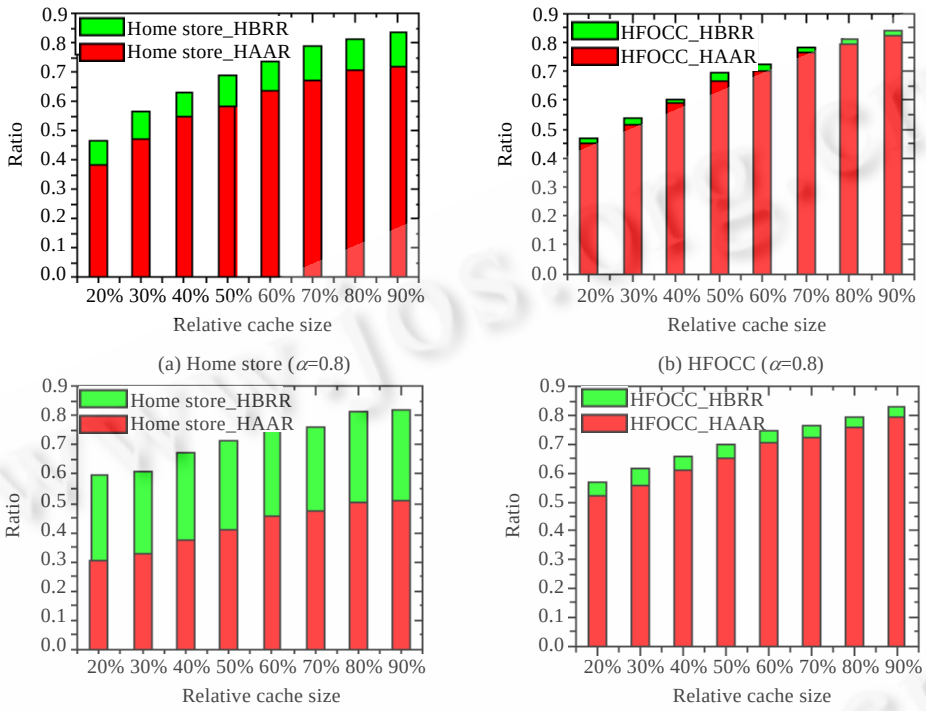
Workload 模型:源服务器中总共存放媒体内容的对象总数为 5 000,各对象均采用编码率为 b 的 CBR 编码,且具有相同的播放时间和存储大小.对象的请求率符合 Zipf 分布,实验中,我们改变α的取值来模拟工作负载的变化.用户请求的到达序列用λ=0.2 的 Poisson 分布模型来产生,并在各节点均匀分布.

实验采用命中且被接纳的比率(hit and accepted ratio,简称 HAAR)、命中但被拒绝的比率(hit but rejected ratio,简称 HBRR)和资源利用率作为主要的性能指标.HAAR 是指在缓存命中且被接纳的请求所占的比例.HBRR 是指虽然缓存命中,但由于节点过载而被拒绝的请求所占的比例.此外,我们还考察了不同负载和资源条件下各对象的副本数量的分布情况.

4.2 HAAR和HBBR的比较

为了反映请求在 OCC 中的处理情况,我们将缓存命中率划分为 HAAR 和 HBRR.通过 HAAR 可以考察 OCC 系统的吞吐率.在请求率一定时,HAAR 越高,系统的吞吐率越高.图 3 展示了不同工作负载分布(α分别取 0.8 和 1.2)

下 HAAR 和 HBRR 的统计情况.在相同工作负载分布下,home store 和 HFOCC 具有非常接近的缓存命中率.由于 home store 缺乏有效的负载平衡机制,系统存在热点,表现出了较高的 HBRR 值和较低的 HAAR 值.如当α=0.8 时,home store 的 HBRR 率约为 10%(如图 3(a)所示),说明约 10%的请求虽然缓存命中但被拒绝.当α=1.2 时,负载分布更加不对称,home store 的 HBRR 值也增加到约为 30%(如图 3(c)所示).而 HFOCC 采取了负载平衡机制,因节点过载被拒绝的请求数大为减少(较低的 HBRR 率),缓存系统可接纳更多的请求(更高的 HAAR 率).



(a) Home store (α=0.8) (b) HFOCC (α=0.8)

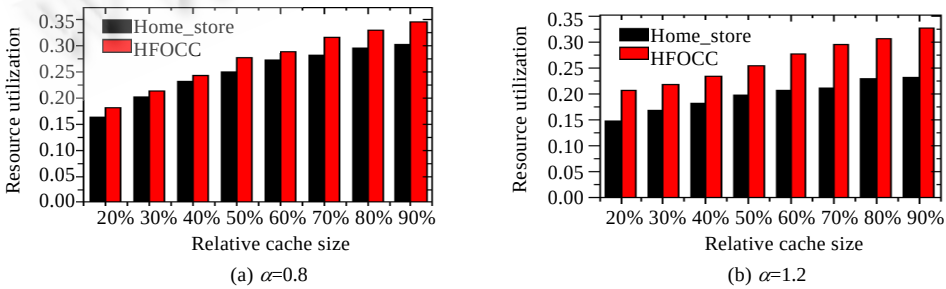
(c) Home store (α=1.2) (d) HFOCC (α=1.2)

Fig.3 Comparison in HAAR and HBRR under various workload and resource conditions

图 3 不同负载和资源条件下的 HAAR 和 HBRR

4.3 资源利用率比较

资源利用率是反映系统效率的重要指标之一.图 4(a)和图 4(b)所示分别为α=0.8 和α=1.2 时系统的资源利用情况.可以看出,在各种工作负载和缓存大小下,HFOCC 的资源利用率均高于 Home store.说明 HFOCC 在消除热点的同时也提高了低负载节点的资源利用率.资源利用率越高,更多的带宽资源被实际用于缓存服务、接纳的用户请求数量也就越大,这与第 4.2 节的结果是一致的.



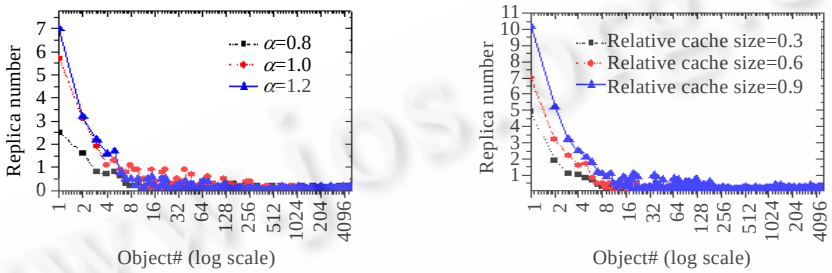
(a) α=0.8 (b) α=1.2

Fig.4 Comparison in system utilization under various workload conditions

图 4 不同负载下系统资源利用率的比较

4.4 副本数量的分布情况

本节考察 HFOCC 中各对象副本数量的分布情况.在不同的α值和缓存大小下,运行 HFOCC 系统,统计运行停止时各节点 replica cache 中的副本对象.我们取运行 10 次的平均结果进行分析.图 5(a)所示为相对缓存大小为 60%时,不同α值下各对象副本数量的分布情况.可以看出,副本主要集中在流行度较高的对象(前 100 个对象)上.对象的流行度越高,副本数量越大.从分布曲线上看,α越大,最大副本数量(即 1#对象的副本数量)越大,但曲线下降也越快.可见,各对象副本数量的分布和流行度分布大致吻合.图 5(b)展示的是当α=1.2 时,不同缓存大小下副本数量的分布情况.总体上,节点缓存空间越大,对象的副本数量越大.这是因为在带宽资源一定的条件下,当缓存空间增加时,各节点的整体负载水平也相应增加,低负载节点可用于"分流"负载的资源相对减少.因此,一个热门对象的负载需要更多的节点来共同分担.



(a) Relative cache size=0.6 (b) α=1.2

Fig.5 The distribution of the replica number for each object

图 5 副本数量的分布情况

4.5 动态工作负载下的性能

本节考察当工作负载动态变化时 HFOCC 的性能.我们动态地改变对象的请求率分布,并统计单位时间内的 HBRR 值和各对象的副本数量.图 6(a)展示了当α值从 0.7 跃变(发生在第 4000″)到 1.2 时,HBRR 值的变化情况.可以看出,home store 的 HBRR 值发生了较大的跳跃,之后围绕在较高的值(约为 0.3)振荡;而在 HFOCC 中,HBBR 值经过短时间的小幅升高后即回落,并围绕 0.04 上下振荡.图 6(b)模拟了出现突发流时(第 4000″)HBRR 值的变化情况.可以看到,home store 的 HBRR 值在第 4000″发生跳跃后即维持在 0.5 的高位振荡;而 HFOCC 的 HBBR 值小幅升高至 0.1 后立即回落到均值约为 0.05 的低位.同样,当α值从 1.2 跳变到 0.8 时(如图 6(c)所示),HFOCC 也表现出较好的适应性.图 6(d)展示的是当α值从 1.2 跳变到 0.8 时,1#对象和 2#对象副本数量的变化情况.可以看出,随着在 t_1 时刻对象的请求率降低,副本的数量开始呈现下降的趋势.经过约 7000″时间,到 t_2 时刻副本的数量达到稳定.可见, HFOCC 可以适应工作负载的大范围变化,且能将性能损失控制在可接受的范围内.

5 结 论

覆盖网协同缓存提供了一种具有吸引力的 Web 缓存方案:聚合闲置的客户资源构建经济有效和可扩展的协同缓存服务.但是,节点的异构性和工作负载的不对称容易造成节点资源使用的不平衡,形成"热点".本文针对多媒体内容分发服务提出了一种可实现负载平衡的 OCC 策略,通过自适应地将热门对象复制到低负载节点,达到分散服务负载和消除热点的目的.HFOCC 具有缓存空间利用率高、开销小、自适应能力强等优点.

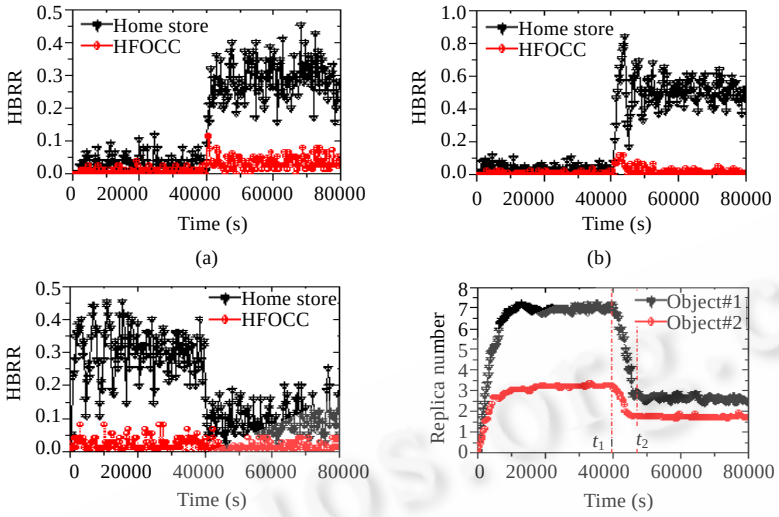


Fig.6 Performance comparison under dynamical workload

图 6 工作负载动态变化时的性能比较

References:

[1] Iyer S, Rowstron A, Druschel P. Squirrel: A decentralized, peer-to-peer Web cache. In: Proc. of the 21st Symp. on Principles of Distributed Computing (PDOC 2002). New York: ACM Press, 2002. 213−222.

[2] Ling B, Wang XY, Zhou AY, Ng WS. A collaboration Web caching system based on peer-to-peer architecture. Chinese Journal of Computers, 2005,28(2):107−178 (in Chinese with English abstract).

[3] Linga P, Gupta I, Birman K. A churn-resistant peer-to-peer Web caching system. In: Liu P, Pal P, eds. Proc. of the ACM Workshop on Survivable and Self-Regenerative Systems. New York: ACM Press, 2003.1−10.

[4] Breslau L, Cao P, Fan L, Phillips G, Shenker S. Web caching and zipf-like distributions: Evidence and implications. In: Doshi B, ed. Proc. of the 18th Annual Joint Conf. of the IEEE Computer and Communications Societies (INFOCOM'99). New York: IEEE Communications Society, 1999. 126−134.

[5] Saroiu S, Gummandi PK, Gribble SD. A measurement study of peer-to-peer file sharing systems. In: Kienzle MG, Shenoy PJ, eds. Proc. of the Multimedia Computing and Networking (MMC). Bellingham: SPIE, 2002. 156−170.

[6] Saroiu S, Gummadi KP, Dunn RJ, Gribble SD, Levy HM. An analysis of Internet content delivery systems. In: Culler D, Druschel P, eds. Proc. of the 5th Symp. on Operating Systems Design and Implementation (OSDI 2002). Boston: USENIX Association, 2002. 315−327.

[7] Wang J. A survey of Web caching schemes for the Internet. ACM Computer Communication Review, 1999,25(9):36−46.

[8] Rowstron A, Druschel P. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In: Guerraoui R, ed. Proc. of the 18th IFIP/ACM Int'l Conf. on Distributed Systems Platforms (Middleware 2001). Berlin: Springer-Verlag, 2001. 329−350.

[9] Guo L, Chen SQ, Ren SS, Chen X, Jiang S. PROP: A scalable and reliable P2P assisted proxy streaming system. In: Tzeng NF, Takizawa M, Hō J, Gakkai S, eds. Proc. of the 24th Int'l Conf. on Distributed Computing Systems (ICDCS 2004). Washington: IEEE Computer Society, 2004. 778−786.

[10] Androutsellis-Theotokis S, Spinellis D. A survey of peer-to-peer content distribution technologies. ACM Computing Surveys, 2004, 36(4):335−371.

[11] Stoica I, Morris R, Karger D, Kaashoek MF, Balakrishnan H. Chord: A scalable peer-to-peer lookup service for Internet applications. ACM SIGCOMM Computer Communication Review, 2001,31(4):149−160.

[12] Ratnasamy S, Francis P, Handley M, Karp R, Shenker S. A scalable content-addressable network. In: Govindan, ed. Proc. of the ACM SIGCOMM 2001. New York: ACM Press, 2001. 161−172.

[13] Rao A, Lakshminarayanan K, Surana S, Karp R, Stoica I. Load balancing in structured P2P systems. In: Kaashoek F, Stoica I, eds. Proc. of the IPTPS 2003. Berlin, Heidelberg: Springer-Verlag, 2003. 256−267.

[14] Dabek F, Kaashoek F, Karger D, Morris R, Stoica I. Wide-Area cooperative storage with CFS. In: Marzullo K, ed. Proc. of the 18th ACM Symp. on Operating Systems Principles. New York: ACM Press, 2001. 202−215.

[15] Druschel P, Rowstron A. PAST: A large-scale, persistent peer-to-peer storage utility. In: IEEE Computer Society, ed. Proc. of the 8th IEEE Workshop on Hot Topics in Operating Systems (HotOS-VIII 2001). Los Alamitos: IEEE Computer Society Press, 2001. 75−80.

[16] Gopalakrishnan V, Silaghi B, Bhattacharjee B, Keleher P. Adaptive replication in peer-to-peer systems. In: IEEE Computer Society ed. Proc. of the 24th Int'l Conf. on Distributed Computing Systems (ICDCS 2004). Washington: IEEE Computer Society, 2004. 360−369.

[17] Sokolinsky LB. LFU-*K*: An effective buffer management replacement algorithm. In: Lee YJ, Li JZ, Whang KY, Lee DH, eds. Proc. of the 9th Int'l Conf. Berlin: Springer-Verlag, 2004. 670−681.

附中文参考文献:

[2] 凌波,王晓宇,周傲英,Ng WS.一种基于 Peer-to-Peer 的 Web 缓存共享系统研究.计算机学报,2005,28(2):107−178.

李春洪(1972－),男,江苏丹阳人,博士,主要研究领域为分布式系统,普适计算.

冯国富(1977－),男,博士,主要研究领域为分布与并行系统,对等计算.

顾铁成(1965－),男,副教授,主要研究领域为分布与并行系统.

陆桑璐(1970－),女,博士,教授,博士生导师,CCF 高级会员,主要研究领域为分布式与并行系统,高性能计算.

陈道蓄(1947－),男,教授,博士生导师,CCF 高级会员,主要研究领域为分布式系统,Internet 计算, CSCW.