乐德广　郭东辉　吴伯僖　(厦门大学技术物理研究所　361005)

Caching Technology in Network and Its Application

网络缓存技术及应用的研究

摘要：网络缓存技术是目前用来提高互联网的用户响应速度及相关网络性能的关键技术，其中缓存策略及算法对缓存性能的发挥起着重要的作用。本文主要介绍在网络缓存系统中所采用的各种替换算法和更新策略，并说明网络缓存技术的各种应用模式。以目前广泛应用的代理缓存 Squid 为例，分析网路缓存系统兑现的结构和方法。

关键词：文件缓存　替换算法　更新策略　分布式缓存　代理缓存服务器

1 前言

随着 Internet 应用的日益广泛，人们越来越依赖于网络获取知识，了解信息，使 Internet 的用户爆炸式增长，以致出现了以下问题：一方面用户的增多，使服务器的负载过重，不能及时响应用户的请求；另一方面网络带宽的不够以及数据传输链路的延迟，造成了网络拥塞，影响 Internet 的使用效果。如果单纯的依靠网络设备端口的扩容和扩大网络传输带宽来满足用户对上网的需求，需要投入大量的资金。根据网络数据传输的时间和地域相关性，一个用户在某一时刻访问某个数据后，该用户及其周围的用户很有可能再次访问这个数据，如果在相同区域内不同用户每次需要数据时都要到远端服务器获取，则会造成数据的重复传输。数据的重复传输不但浪费了许多网络带宽，使网络速度越来越慢，而且使服务器的负荷加重，因此人们希望通过网络缓存技术来优化网络性能，将未来可能访问到的数据预先放到离用户较近的缓存服务器上，用户对相同数据的访问可以直接从缓存服务器上获取，而不必向远程服务器请求数据。采用网络缓存技术是一种投入少，效率高的提高网络性能的有效方法，研究开发各种网络缓存技术及其应用具有十分重要的意义。本文详细介绍网络缓存系统中的各种替换算法、更新策略等技术，阐述网络缓存的各种应用模式；最后以目前广泛应用的代理缓存 Squid 为例分析网路缓存系统兑现的结构和方法。

2 网络缓存技术

网络缓存技术研究的内容主要是如何选择将特定的数据存储到缓存中，因此在网络缓存技术中，需要解决的问题主要是如何替换和更新缓存数据。

2.1 缓存替换算法

所谓缓存替换即在缓存空间放满后，当有新的数据需要缓存时，如何确定哪些缓存数据可以从缓存中删除，以存储新的数据。常用的算法有 LRU [1]、SIZE [2]、LFU [1] 等。

由于篇幅限制，本文不再对各种算法做进一步描述，有兴趣的读者可参阅相关文献。

2.2 缓存更新技术

网络缓存技术的另一个主要问题是缓存数据的一致性问题，如果缓存数据与实际数据一致，则称该缓存数据有效，否则称为无效缓存数据，因此必须对缓存的数据进行更新，使之和实际数据保持一致。在缓存技术中维护缓存数据一致性的缓存机制目前有 TTL（Time to Live）、Client Pulling 和 Invalidation Protocols 几种。

TTL（Time-to-Live）机制。在 TTL 中，每个缓存数据被赋予一个有效期，缓存在该有效期之内保留该数据，过了有效期之后，则认为缓存数据是无效的，并向原数据服务器发出更新数据请求。如：对于 Web 文档，可以在 HTTP 头上打上“过期（Expires）”来标记数据的有效期。TTL

检测机制使用很广泛，但并不是最有效的，因为缓存中的内容虽然过期，但它们仍可能有效，然而缓存同样会发出更新请求。

Client Polling机制。缓存向服务器周期性的发出查询命令来检查缓存数据的有效性，缓存利用返回信息删除或更新缓存中的数据。在Client Pulling中，缓存查询服务器的时间称为缓存数据的更新时间，缓存记录每个数据最近的更新时间。可以采用不同的方法来确定更新时间，最常见的是在每次向服务器取数据时，向服务器发送"get if modified"请求，如果数据没有被更新，那么缓存中的数据还可以使用。如果发现数据已经更新则新的数据将被存到缓存中替换旧的数据。

TTL和Client Pulling机制只能保证缓存数据与原数据在大部分时间内是一致的，要保证缓存数据在所有时间内都有效可以采用Invalidation protocols机制。与TTL和Client Pulling不同，Invalidation protocols机制是服务器端主动，缓存客户端被动，即服务器跟踪每个被缓存的数据，当该数据改变时，利用Multicast等方式通知每个缓存该数据的缓存客户更新或删除相应的缓存数据。这种机制需要服务器对缓存数据进行跟踪，因此系统实现比较复杂。

3 缓存的应用

3.1 缓存在不同网络节点的应用

在实际应用中，根据缓存在不同网络节点的应用有三种实现方式，即客户端、服务器端以及代理服务器端缓存机制。

3.1.1 客户端的缓存应用

客户端的缓存通常是在浏览器上实现的，如Netscape、Internet Explorer等都具有缓存机制。浏览器把一定时期内用户访问过的文档都存储起来，当用户重新访问同样的文档时，就可以从浏览器的缓存中取出，而不需要重新建立HTTP连接。客户端缓存机制缓存的文档信息大小有限、实现的一致性策略和替换策略都比较简单，它经常提供给用户的是一些失效了的陈旧数据，因此是一种不大可靠的缓存机制。

3.1.2 服务器端的缓存应用

服务器端的缓存机制在服务器上使用内存作为缓存空间。当服务器响应用户对某个数据请求后，在内存缓存空间中保留一个副本，下一次如果有相同的访问请求，就直接将缓存空间中的副本提供给用户，从而减少磁盘读取时间，加快服务器的反应能力和增加服务器的吞吐量。服务器端的缓存机制能够降低用户的访问延迟，但对服务器的要求较高，增加了服务器硬件和软件的复杂度。

3.1.3 代理端的缓存应用

代理服务器缓存机制简称代理缓存。用户对服务器的数据请求到达代理缓存后，如果该数据存储在代理缓存中，代理缓存直接把该数据传给用户作为响应；如果代理缓存没有该数据，则由代理缓存向原数据服务器获取该数据后再传给用户，并在代理缓存上保存该数据。这种缓存机制具有多用户可以共享所有缓存数据的特点，提高了网络的利用率，降低了用户访问延迟和费用。典型的代理缓存有Netscape公司的Netscape Proxy Server、Colorado大学的Harvest Object Cache以及在Harvest基础上NLANR开发的Squid缓存系统等。

3.2 分布式缓存系统

单个缓存节点处理请求的能力和磁盘大小是有限的，一旦缓存过载，用户和外界的连接就会中断；如果单个缓存的磁盘容量太小，缓存中的有些内容来不及再次访问就要被清除出去，导致请求命中率下降。因此可以使用能够相互通信的多个缓存服务器来构成一个分布式缓存系统。多个缓存通过互相协作，在不同缓存之间共享数据，从而克服单个缓存的局限性，更好地提高缓存效率，并进一步优化网络性能。

ICP（Internet Cache Protocol）是最早出现的分布式缓存协议，利用该协议可以方便、快速的设计缓存层次结构，是目前在分布式缓存系统中应用最广泛的协议。ICP可以定义缓存服务器之间的相互关系，如上下级之间定义为父子关系，同级之间定义为同伴关系，并通过ICP报文在缓存服务器间进行通信，共同为用户提供缓存服务。ICP的工作方式如图1所示。图中当客户机请求Internet上的服务器数据时，客户机首先向其所在的本地缓存发送数据请求，本地缓存收到请求后，检查客户请求的数据是否在它的缓存中。如果在，则响应客户的数据请求；如果数据不在缓存中，则向它的同伴缓存节点发送ICP请求，同伴节点回复是否有数据信息，如果有，则从同伴节点拷贝数据，并传给客户。当所有的同伴节点回复没有数据或超过规定时间后，本地缓存向父缓存节点发送ICP请求，如果父节点有对应的数据，则把数据传送给本地缓存，如果父节点也没有数据，则可由父节点向原数据服务器请求数据，传送给本地缓存，或者由本地缓存直接向原数据服务器取数据，再由本地缓存发送给客户机。

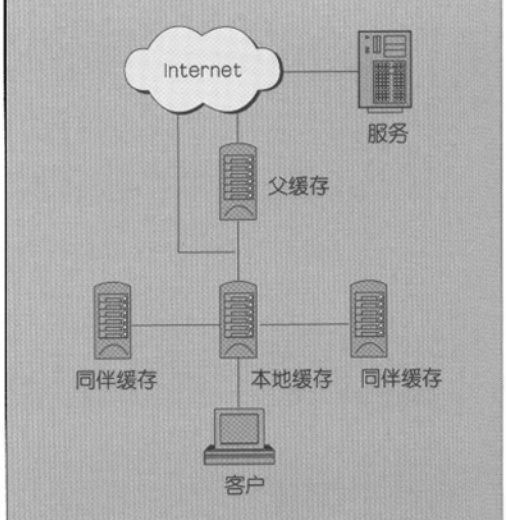

图1 基于ICP的分布式缓存模型

虽然ICP成功的使缓存服务器连在一起，但也有不少的缺点：ICP通信是建立在查询/应答基础上的，当缓存没有用户要求的数据时，它必须向它的同伴有限广播ICP请求，因此同伴数增多时，通信量的开销和处理能力的开销也随之增加，由于必须等待所有的同伴都回答没有相应的缓存数据，才能向服务器发送请求，网络延迟也相应增加，使ICP的效率下降。

Cache Digest。它是ICP的扩展，其主要思想是通过保存其他缓存节点的缓存数据的摘要来减少缓存间的通信量，当本地缓存收到用户的数据请求时，首先检查本地缓存数据和缓存数据摘要，如果本地缓存有数据，则直接响应用户的数据请求；如果缓存摘要里有数据信息，则向相应的缓存节点请求数据，如果没有，则直接向原数据服务器发送请求，而不需要发送ICP请求到所有的其他缓存节点中，从而大大减少了缓存之间的通信量。不同缓存的摘要通过HTTP协议进行周期地更新，保证摘要的可靠性。在Cache Digest中，如果有很多的同伴缓存，那么在内存中存储摘要的代价是很高的，因此这种缓存系统对硬件处理能力要求很高。

CARP（Cache Array Routing Protocol），它是采用Hash路由算法，利用高效的负载平衡来获得缓存最大命中率和最小延迟时间的协议。在CARP中，通过计算Hash函数将数据存在不同的缓存节点上，当缓存节点收到用户请求时，它利用这个函数计算到哪一个缓存节点上去获得所要的数据。为了实现负载平衡，不同的缓存服务器的IP地址不同，但却具有相同的主机名。因此虽然客户是向相同主机名的缓存服务器请求数据，但实际上却是向具有不同IP的缓存服务器请求数据。CARP解决了在ICP中因为缓存节点的增多而引起的缓存之间通信负载增大的问题。

4 缓存系统的兑现

在以上缓存应用中，代理缓存是解决Internet访问速度慢、服务器负载重和网络阻塞等问题的最有效方法。下面以Squid为例来说明缓存系统兑现的结构与方法。

Squid源于ARPA出资开发的缓存服务器Harvest Research Project，它不但实现对Web信息的高效缓存，同样支持FTP、Gopher代理缓存，是一个高性能的代理缓存服务器。Squid在内存中保存访问频繁的数据，并且在磁盘中维护一个高效的缓存数据库。Squid缓存系统结构如图2所示：

图中Storage Manager是Squid代理缓存结构的重要组成部分，每一个缓存数据被存储在缓存文件存储结构中。Storage Manager的Memory Manager用于为经常使用的数据结构分配和管理内存空间；Hash Table建立在内存中，它记录磁盘缓存数据的位置情况，因此通过该Hash Table，Squid可以快速确定缓存数据；在Storage Manager中使用替换、一致性等各种缓存技术来提高缓存性能。在Squid中可以选择采用LRU、LFUDA、GDSF等替换算法，并根据数据响应日期（OBJ_DATE）、数据最后调整日期（OBJ_LASTMOD）、数据缓存时间（OBJ_AGE=NOW－OBJ_DATE）、数据最后一次更新的时间（LM_AGE=OBJ_DATE－OBJ_LASTMOD）、OBJ_AGE与LM_AGE之间的比值（LM_FACTOR＝OBJ_AGE/LM_AGE）、服务器指定的数据最长时间（CLINET_MAX_AGE）、服务器指定过期时间（EXPIRES）等参数与用户设定的数据更新参数：当数据没有指定过期时间时考虑更新的时间（CONF_MIN）、当数据没有指定过期时间时OBJ_AGE的百分比（CONF_PERCENT）、当数据没有指定过期时间时考虑更新的最长时间值CONF_MAX进行比较来确定对缓存数据的更新，算法流程如图3所示：

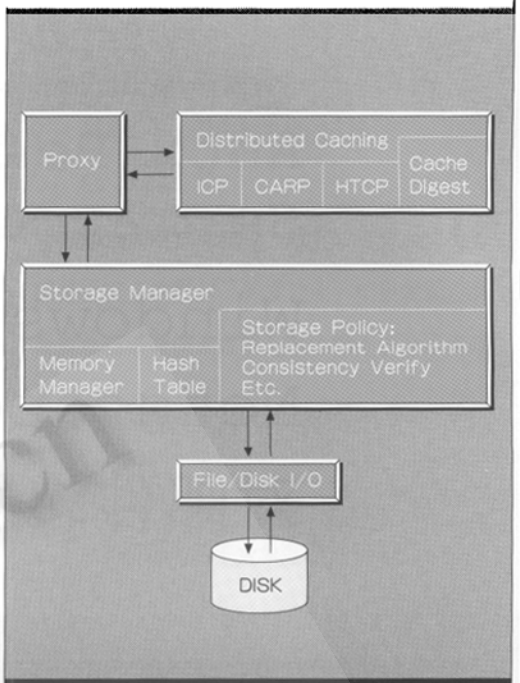

图2 Squid缓存系统结构示意图

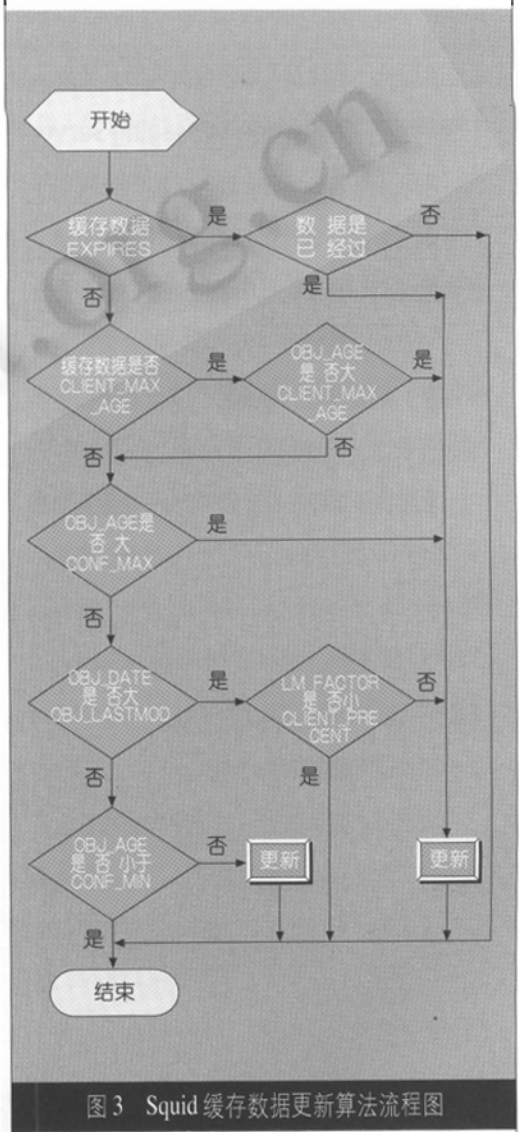

图3 Squid缓存数据更新算法流程图

（文章未完，转第25页）

(以下内容接第35页)

File/Disk I/O用于读写磁盘缓存数据，并设置了I/O缓存。在Squid缓存结构中，网络和磁盘I/O是分开的，这样当缓存出现网络相关错误时，不会影响对磁盘缓存数据的操作。

Squid中的ICP、CARP、HTCP、Cache Digest等模块提供了对分布式缓存系统的支持。利用ICP协议定义Squid代理服务器的各种关系，如父子关系、同伴关系等，并通过ICP报文进行相互通信，构成一个层次式的分布缓存结构。此外利用Cache Digest模块在内存中建立一个摘要表（Digest Table），如果用户端想要的数据自己没有时，可以通过查询Digest Table知道哪个同伴节点上有数据。

5 结束语

网络缓存不是着眼于网络设备性能的提高和扩容，而是在更深一个层次上分析用户数据流的统计特征，利用数据的可复制性和共享性，在边缘网络复制和保存网络数据的一项重要技术。它不但可以减少网络拥塞和服务器负载，提高数据访问速率和服务可靠性，而且大大节省了网络运营费用，从而在ISP、ICP以及跨国公司中得到广泛的应用。因此网络缓存有着非常广阔的发展前景，相信随着网络应用的不断深入，网络缓存将会发挥越来越重要的作用。

参 考 文 献

1 D.L. Willick, D.L. Eager, R.B. Bunt. Disk Cache Replacement Policies for Network Fileservers [C]. Proceeding of the 13th International Conference on Distributed Computing Systems, Pittsburgh, PA, 1993. 2-11.

2 S. Williams, M. Abrams, C. Standridge, G. Abdulla, E. Fox. Removal Policies in Network Caches for World-Wide Web Documents [C]. Proceedings of ACM SIGCOMM, Stanford, CA, 1996. 293-305.

3 R. Karedla, J.S. Love, B.G. Wherry. Caching Strategies to Improve Disk System Performance [J]. IEEE Computer, 1994, 27 (3): 38-46.

4 J. O'Neil, E. O'Neil, G. Weikum. The LRU-K Page Replacement Algorithm for Database Disk Buffering [C]. Proceedings of ACM SIGMOD, Washington, DC, 1993. 297-306.

5 K. Cheng, Y. Kambayashi. LRU-SP: a Size-Adjusted and Popularity-Aware LRU .