基于内容的分布式 Web 服务器调度算法*

杜增凯+, 郑名扬, 鞠九滨

(吉林大学 计算机科学与技术学院,吉林 长春 130012)

A Distributed Algorithm for Content-Aware Web Server Clusters

DU Zeng-Kai+, ZHENG Ming-Yang, JU Jiu-Bin

(College of Computer Science and Technology, Jilin University, Changchun 130012, China)

+ Corresponding author: Phn: 86-431-5169350, E-mail: zkdu@sina.com.cn

http://www.jlu.edu.cn

Received 2002-08-20; Accepted 2003-03-04

Du ZK, Zheng MY, Ju JB. A distributed algorithm for content-aware Web server clusters. *Journal of Software*, 2003,14(12):2068~2073.

http://www.jos.org.cn/1000-9825/14/2068.htm

Abstract: While content-aware distribution policies are getting more popular in cluster-based web systems, they make the dispatching node a bottleneck. To address the scalability and fault-tolerance problem, issues about designing distributed dispatching policies are discussed. For the policies aiming at improving the cache hit rate, a distributed dispatching policy named DWARD (distributed workload-aware request distribution) that takes into account both the load balance and the locality enhancement is presented. Finally, a testbed is implemented on the basis of a Linux kernel to benchmark various dispatching algorithms. The performance results show that DWARD can achieve a favorable throughput compared with the state-of-the-art dispatching policies.

Key words: distributed Web server (DWS); content-based; scalability; fault tolerance

摘 要: 在分布式 Web 服务系统的研究中,基于内容的调度策略日益受到关注.但是,基于内容的请求调度带来的额外开销使得调度节点成为系统的瓶颈,限制了系统规模.为了实现系统的容错和扩展,集中讨论了分布式调度策略的设计问题,并针对难于分布的面向缓存调度策略设计了相应的分布式调度算法 DWARD(distributed workload-aware request distribution).基于 LINUX IP 协议栈的系统测试表明,DWARD 算法可以在适当调整的情况下获得良好的性能.

关键词: 分布式 Web 服务器;基于内容;可扩展性;容错

中图法分类号: TP393 **文献标识码**: A

随着 Internet 的飞速发展,人们对全球网络数据的访问需求也急剧增加.越来越多的站点和 Internet 服务提供商采用机群体系结构(分布式 Web 服务器(distributed Web server),简称 DWS)提供 Web 服务.传统 DWS 系统的研究大都根据数据包的 IP 地址和 TCP 端口号实现请求的调度.基于内容的 DWS 请求调度将客户请求的类

* Supported by the National Natural Science Foundation of China under Grant No.60073040 (国家自然科学基金)

第一作者简介: 杜增凯(1974－),男,河北沧州人,博士生,主要研究领域为分布式系统,分布式 Web 服务器.

型甚至具体内容都纳入了调度的范畴.丰富的调度信息增强了 DWS 的灵活性和智能性,扩大了 DWS 系统的应用范围.

目前,基于内容的 DWS 研究大都使用集中式的调度算法[1~5].这些算法只能在一个集中的调度节点上运行,不但为系统增加了单一失效节点,而且在很大程度上限制了系统的扩展.研究表明[6],基于内容的 DWS 调度所能支持的系统规模与传统的 Web 调度相比下降了近一个数量级.本文试图通过设计分布式的调度策略从根本上解决DWS 系统的扩展问题.首先,我们将通过分析调度算法对调度信息的需求讨论现存调度策略的分布化问题.对于难于分布的面向缓存调度策略,我们设计了相应的分布式调度算法 DWARD(distributed workload-aware request distribution).此算法不但可以让所有的服务节点参与请求的调度,而且可以同时兼顾系统的负载平衡和缓存命中率.另外,DWARD 算法的运行无须节点间频繁的信息交互.基于 LINUX IP 协议栈的系统测试表明,DWARD 算法可以在适当调整的情况下获得良好的性能.

1 基于内容的分布式调度策略

在过去的几年中,研究人员根据各自的需要设计了大量的调度算法,典型调度算法的分类比较可参见文献[7,8].这些算法虽然可以使用各种调度信息实现请求的调度,却只能运行在集中的调度节点上.我们的目标是要设计分布式的调度策略.在采用分布式调度策略的系统中,每个服务节点负责调度自己收到的客户请求.这将使系统中不再存在性能瓶颈和单一失效节点.以下我们将通过分析调度算法对调度信息的需求讨论现存调度算法的分布化问题.

通过分析现存的调度算法,我们把调度信息分为两种:客户请求信息和服务器状态信息.客户请求信息是指可以从客户请求中获取的信息,包括请求的 URL,Cookie,SSL Session ID 等.服务器状态信息有:(1) 静态调度信息:包括系统中各节点的硬件配置和软件配置.硬件配置信息可能包括节点功能强弱、专用服务器类别等.软件配置信息则包括负载集合在各节点的分布以及特定调度算法所需的统计数据.(2) 动态调度信息:包括各服务节点的负载情况(CPU 利用率、I/O 利用率、未处理的连接数)以及它们对文件的缓存情况.表 1 列出了典型的调度算法和调度信息之间的关系.

Table 1 State information used by popular dispatching policies

表 1 典型调度算法使用的调度信息

Algorithms	Client information	Static server information	Dynamic server information
HASH	Requested URL		
CAP	Requested URL[9]		
SITA-E	Requested URL[10]	Size of target file	
LARD	Requested URL[11]		Active connections, cache state
Client affinity	Cookie, SSL Session ID[12,13].		
Service differentiation	Cookie[13], custom HTTP header[14]		
Content placement	Requested URL[15]	Working set of each server node	
Specialized server	Requested URL	Type of specialized server	

由表 1 可以看出,在基于内容的请求调度中,来自客户请求的信息占有重要的地位.CAP,HASH 等调度算法单纯依靠来自客户请求的信息就可以实现完整的调度策略.由于请求的初始接收节点可以方便地获取来自客户请求的信息,这一类算法很容易分布化.

显然,也有不少调度算法需要来自服务器的状态信息.其中一些只需要来自服务器的静态调度信息.利用这些信息,不但可以有效地使用某些专用服务器,而且可以在各服务节点之间实现站点内容的分布.在调度算法 SITA-E 中,一种来自服务器的重要调度信息是 URL 与文件大小的对应关系.由于这些服务器端的静态调度信息可以在系统配置过程中一次性获得,因此相关的调度算法也不难实现分布化.

在表 1 中,只有少数调度算法考虑来自服务器的动态信息.但是由于精确的负载信息较难获取且难以利用,需要负载信息的调度策略往往使用当前连接数来衡量不同节点的负载[11].为了提高静态网页的缓存命中率,面向缓存的调度策略需要获取全局的缓存信息.集中式的调度策略往往假定它们运行在一个集中的调度节点上.在这一特殊的调度节点上,可以很方便地获取全局的缓存信息,进而实现优化的调度决策.然而在一个分布式的调度算法中,各服务节点地位平等,这使得它们无法自然地获得静态网页的全局缓存信息.通过广播的方式固然

可以实现缓存信息的共享,但是让服务节点每响应一个用户的请求都进行一次广播无疑会带来很大的开销.

2 面向缓存的分布式调度算法 DWARD

目前,面向缓存的调度算法主要有 HASH,LARD 和 WARD.HASH 算法是一种静态的调度算法,它可以有效地提高缓存的命中率,但无法保证系统的负载平衡.HASH 算法的另外一个特点就是容易分布化,在第 3 节的系统测试中我们将使用分布式的 HASH 算法 DHASH.LARD[11]算法对系统工作集的划分是动态的,这使得它可以在提高缓存命中率的同时,兼顾服务节点间的负载平衡.然而,在使用 LARD 算法的系统中,大部分的客户请求将被转接[11]到其他服务节点.WARD 算法[16]将访问最为频繁的一小部分文件定义为 core,通过 core 集合的本地服务可以有效地减少昂贵的 TCP 转接操作.然而,WARD 仍然是一个集中式的调度算法.

对比 LARD 算法和 WARD 算法的调度流程可以发现,WARD 算法隐含着一个重要的特征:WARD 算法不考虑负载平衡问题.在 LARD 算法中,负载平衡问题决定了工作集划分的动态变化,使 LARD 算法的每一次请求调度都需要全局缓存信息的支持,也正是这一点使得 LARD 算法难于分布化.WARD 算法对负载平衡问题的忽略让我们认识到,负载平衡功能可以从基于内容的调度算法中分离出来.我们将充分利用这一特征,实现一个分布式的 WARD 算法,称为 DWARD.

与 WARD 算法一样,DWARD 把系统的工作集分为 3 部分:core,partition 和 disk.Core 集合和 partition 集合缓存在各节点的内存中,disk 集合则包含无法缓存在内存中的部分.Core 集合相对较小,然而由于访问频繁,它会覆盖绝大部分的客户请求.partition 和 disk 集合只覆盖少数的客户访问,但它可能包含大部分目标文件.

与 WARD 算法类似,DWARD 算法在本地处理客户对 core 集合的访问.由于 core 集合覆盖了绝大部分的客户请求,core 请求的本地处理避免了大量费时的 TCP 转接操作.然而对于 DWARD 算法来说,core 请求的本地处理还有一个更重要的特征:它避免了节点之间的交互,使得系统对 core 请求的处理可以分布化.

WARD 算法对 partition 集合的处理在某种程度上继承了 LARD 算法的处理方式,其主要目的在于减少内存空间的消耗.在 DWARD 算法中,我们采用 HASH 算法来处理对 partition 集合的访问.这种处理方法不但可以节省内存空间的使用,而且使得系统对这部分文件的处理可以分布化.另外,partition 集合的特点(覆盖请求少、包含文件多)使得客户对这部分文件的访问相对均匀,使用 HASH 算法处理对这部分文件的访问不会导致负载的不均衡.

以下是 DWARD 算法对请求的调度流程:

(1) 请求属于 core,本地服务;

(2) 请求属于 partition,使用 HASH 算法选定目标节点;

(3) 请求属于 disk,本地服务.

由于 core 集合的存在,DWARD 算法可以在前端交换机的支持下实现负载平衡.从这一点可以看出,DWARD 并不是静态的调度算法,这与(分布式)HASH 算法是有区别的.与 LARD/WARD 算法相比,DWARD 算法不再需要集中的调度节点.集中调度节点的消除不但减少了系统的配置,而且为系统除掉了单一的失效节点,在一定程度上提高了系统的容错性能.在 LARD/WARD 算法中,每个客户请求的调度都需要服务节点和调度节点间的一次交互.而在 DWARD 算法中,所有的节点都可以根据自身的局部信息实现请求的调度,这从根本上消除了各服务节点间的频繁交互.

在 DWARD 算法中,core 文件集合的选择直接影响着系统的性能.core 集合的增大会使大量的请求在本地得到服务,显著地降低系统的转接开销.但是 core 集合的增大也会将部分 partition 集合的文件兑换到磁盘上,增加磁盘访问的开销.对于指定的访问模式和系统指标,计算一个较优的 core 集合是可能的.文献[16]提出了一种计算 core 集合大小的算法.该算法考虑了网站的访问模式和大量的系统参数(如节点个数、节点内存、TCP 转接开销、读盘开销等).该算法的缺点在于,它必须事先从访问日志中获取各目标文件的访问频度以及许多精确的系统参数.另外,它也没有考虑到磁盘系统和 CPU 的并行问题.采用适当的工具对系统进行详细的建模分析是很有必要的.在第 3 节,我们将采用实测的方法确定 core 集合的大小.

3 性能评价

为了定量地测试第 2 节提到的调度算法,我们基于 Linux IP 协议栈和 TCP 转接技术实现了一个测试平台.配合不同的调度算法,测试平台可以配置成不同的调度框架.对于集中式的调度算法,我们将测试平台配置为 ScalaServer 调度框架[6].据我们所知,ScalaServer 调度框架是运行集中式调度算法扩展性最好的调度框架.对于分布式的调度算法,我们将使用一种分布式的调度框架.于是所有的待测组合有:

(1) S-LARD:ScalaServer 调度框架上运行 LARD 算法;

(2) S-WARD:ScalaServer 调度框架上运行 WARD 算法;

(3) D-DHASH:分布式调度框架上运行 DHASH 算法;

(4) D-DWARD:分布式调度框架上运行 DWARD 算法.

我们的测试平台由客户节点、前端交换机和服务器群 3 部分组成,如图 1 所示.服务器群由 4 个配置相同的节点组成,其配置为:Pentium133,32 兆内存,IDE 硬盘,100 兆网卡.对于 ScalaServer 调度框架,增加一个调度节点(PIII450,64 兆内存,IDE 硬盘,100 网卡).为了减少客户端的数量,使用较高配置的客户机:PIII933,128 兆内存,IDE 硬盘,100 兆网卡.各服务节点运行 Linux 操作系统(内核版本 2.4.10),并通过一个可装载模块实现 TCP 转接功能.在分布式调度框架中,调度策略的实现分布在各节点的 TCP 转接模块中,而与之对比的 ScalaServer 调度框架则在调度节点上集中实现.服务节点使用 Apache-1.3.20 作为 Web 服务器.客户端使用的软件源自 S-Clients[17].我们在 S-Clients 的基础上增加了日志回放和多机共同测试的功能.我们选用 S-Clients 的原因在于,它可以按照指定的频率产生连接请求,这使得它生成的负载不受被测系统的限制.

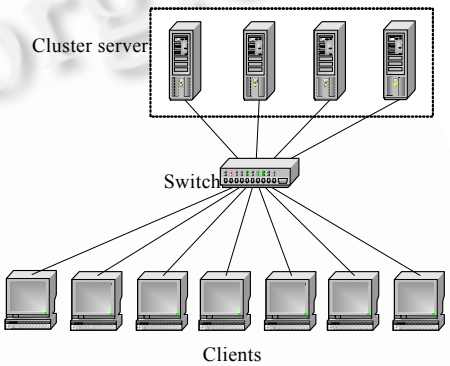

Fig.1 The testbed

图 1 测试平台

在如图 1 所示的测试平台中,前端交换机具有两项功能:(1) 负责客户机与服务节点之间以及各服务节点之间的通信;(2) 在四层实现简单的负载平衡策略.要实现这两项功能可以简单地使用商用的四层交换机.由于没有四层交换设备,我们使用常用的交换设备实现必需的通信功能,而将交换设备的负载平衡功能在测试软件中实现.由于在两种调度框架中前端交换机的功能是相同的,因此这种配置的变化不会影响结果的可比性.

鉴于被测的 4 种算法都是面向缓存的,我们采用日志回放的方法来生成接近现实的 Web 负载.在测试过程中,我们使用来自美国宇航局肯尼迪空间中心 Web 服务器的访问日志(简称 NASA 日志).此日志时间跨度一个月,成功请求为 1 647 832 个,其文件-访问分布图为图 2.

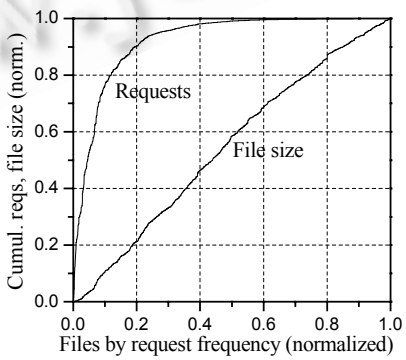

Fig.2 The NASA trace

图 2 NASA 日志

图 2 中 X 轴代表目标文件数(按访问频率降序排列),Y 轴表示这些文件所占的空间和覆盖的请求数.为了便于显示,我们将空间占用量和覆盖请求数进行了标准化处理.NASA 日志包含 4 584 个目标文件,共占空间 200M,要覆盖 97/98/99%的客户请求需要 77.6/90.5/112.6M 内存空间.

首先,我们要为 WARD 和 DWARD 算法确定一个较优的 core 文件集合.图 3 显示了 WARD 和 DWARD 算法在不同大小 core 集合下的性能参数(NASA 日志).图中 X 轴表示 core 集合相对于节点最大缓存空间的百分比,Y 轴表示单位时间内系统完成的请求个数.每个数据点的测试时间为 10 分钟.从图 3 的性能曲线我们可以得出以下几点结论:

(1) 静态调度算法性能较差.在性能曲线的左端点,WARD(DWARD)算法的 core 集合为空.实际上,这是一种静态划分负载集的调度算法.大量的转接操作以及对负载情况的忽略导致了静态调度算法的性能较差.

(2) 适当增加 core 集合大小,可以有效地提高系统的性能.这是因为 core 集合的增加使得访问最为频繁的一小部分文件可以在本地服务,有效地减少了 TCP 转接操作.

(3) 在性能曲线的右端点,所有的客户请求都在本地进行服务.这是一种内容无关的调度算法(最小连接调度).性能曲线的中间部分高于右端点说明基于内容的调度算法可以提高缓存的命中率,优化系统的性能[11].

(4) 性能曲线的右端点仍然具有较高的性能指标,这说明目标文件对系统内存的征用没有给系统的性能带来太大的影响.事实上,10 分钟测试的负载集合约为 126MB,而单个节点所分到的负载集只有 79MB 左右.在测试过程中我们发现,单个节点的最大文件缓存空间在 15MB 左右,右端点较高的性能指标应该源自 Apache 的多进程机制和 Linux 操作系统的虚存管理策略.

图 4 为 4 种组合在不同负载下的性能指标.可以看出,并不是所有的分布式调度算法都具有较好的性能.作为一种静态的调度算法,DHASH 屏蔽了客户端实现的负载平衡功能,D-DHASH 组合在每秒 300 个请求的负载下就达到了饱和.LARD 算法在提高后端节点缓存命中率的同时,可以兼顾服务节点间的负载平衡,这使得 S-LARD 组合的性能指标高出 D-DHASH 15%.在 LARD 算法的基础上,WARD 算法通过减少 TCP 转接操作,将系统的性能优势提高到 33%.很明显,D-DWARD 组合具有最高的性能指标,其吞吐能力高出 D-DHASH 40%.D-DWARD 组合的性能优势在于它不但避免了大量的 TCP 转接操作,而且无须服务节点和调度节点之间的频繁交互.

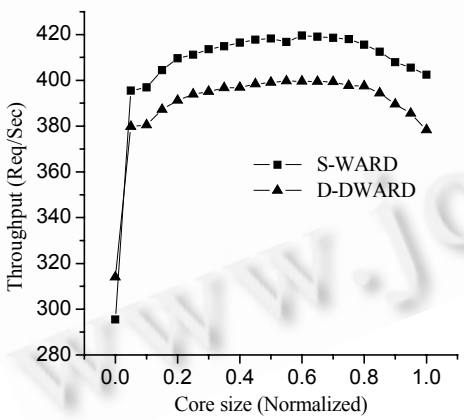

Fig.3 Core selection for the NASA trace

图 3 NASA 日志的 Core 集合选取

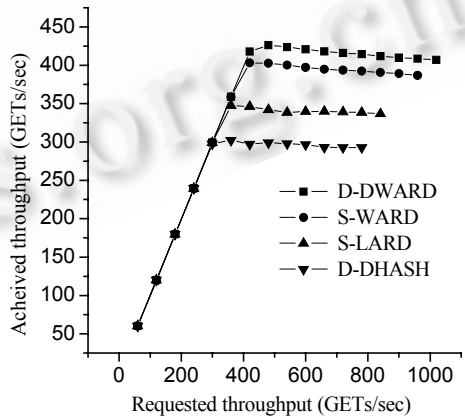

Fig.4 Throughput on the NASA trace

图 4 NASA 日志的性能曲线

值得注意的是,与 S-WARD 组合相比,D-DWARD 的性能指标仅高出 6.5%.在 ScalaServer 调度框架的实现中,我们采用了 URL 压缩和批量请求技术[6]来削减服务节点与调度节点之间的通信开销.NASA 日志的文件集包含大量的航空航天图片,平均文件大小在 45K 左右.目标文件相对较大,在一定程度上减轻了调度节点的负担.另外,在对 S-WARD 组合的性能测试中,调度节点的 CPU 利用率保持在 10%左右.这一部分源自上文提到的优化技术,另一部分则来自调度节点较高的配置(PIII450).如果使用与服务节点配置相同的调度节点,相信 D-DWARD 组合会显示出更大的性能优势.

在图 4 中,各性能曲线的长度并不相同,这是由测试工具 S-Clients 的特点决定的.由于不受被测系统的限制,S-Clients 可以产生超过被测系统服务能力的负载.然而当 S-Clients 产生的请求频率过高时,客户机建立的连接会快速累积,致使所用的文件描述符超过操作系统的限制(1 024),无法生成有效的性能指标.

4 结 语

近年来,基于内容的 Web 调度得到了广泛的研究.然而流行的调度策略往往只能运行在一个集中的调度节点上.这不但为系统增加了单一失效节点,而且在很大程度上限制了系统的扩展能力.本文系统地讨论了现存调度策略的分布化问题,并针对难于分布的面向缓存调度策略设计了相应的分布式调度算法 DWARD.该算法不但从根本上解决了系统的扩展问题,而且为系统容错功能的实现提供了广阔的空间.基于 LINUX IP 协议栈的系统测试表明,DWARD 算法可以在适当调整的情况下获得良好的性能.

References:

[1] Zhang XL, Barrientos M, Chen JB, Seltzer M. HACC: An architecture for cluster-based Web servers. In: Proceedings of the 3rd USENIX Windows NT Symposium. 1999. 155~164.

[2] Cohen A, Rangarajan S, Slye H. On the performance of TCP splicing for URL-aware redirection. In: Proceedings of the 2nd USENIX Symposium on Internet Technologies and System. 1999. 117~125.

[3] Yang CS, Luo MY. Efficient support for content-based routing in web server clusters. In: Proceeding of the 2nd USENIX/IEEE Symposium on Internet Technologies and Systems (USITS'99). 1999.

[4] Carrera EV, Bianchini R. Efficiency vs. portability in cluster-based network servers. In: Proceedings of the 8th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. 2001. 113~122.

[5] Song J, Levy-Abegnoli E, Iyengar A, Dias D. Design alternatives for scalable web server accelerators. In: Proceedings of the 2000 IEEE International Symposium on Performance Analysis of Systems and Software. IEEE Computer Society Press, 2000. 184~192.

[6] Aron M, Sanders D, Druschel P, Zwaenepoel W. Scalable content-aware request distribution in cluster-based network servers. In: Proceedings of the USENIX 2000 Annual Technical Conference. Berkeley: USENIX Association, 2000. 323~336.

[7] Cardellini V, Casalicchio E, Colajanni M, Yu PS. The state of the art in locally distributed Web-server systems. ACM Computing Surveys, 2002,34(2):263~311.

[8] Casalicchio E, Cardellini V, Colajanni M. Content-Aware dispatching algorithms for cluster-based web servers. Cluster Computing, Kluwer Academic Publish, 2002,5(1):67~76.

[9] Casalicchio E, Colajanni M. A client-aware dispatching algorithm for Web clusters providing multiple services. In: Proceedings of the 10th International World Wide Web Conference. Hong Kong: ACM Press, 2001. 535~544.

[10] Harchol-Balter M, Crovella ME, Murta CD. On choosing a task assignment policy for a distributed server system. Journal of Parallel and Distributed Computing, 1999,59(2):204~228.

[11] Pai VS, Aron M, Banga G, Svendsen M, Druschel P, Zwaenepoel W, Nahum E. Locality-Aware request distribution in cluster-based network servers. In: Proceedings of the 8th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VIII). New York: ACM Press, 1998. 205~216.

[12] F5 Networks. 2003. http://www.f5labs.com/.

[13] HydraWeb Technologies. 2003. http://www.hydraweb.com/.

[14] Chandra S, Schlatter Ellis C, Vahdat A. Differentiated multimedia web services using quality aware transcoding. In: Proceedings of IEEE Infocom 2000. Tel Aviv: IEEE, 2000. 961~969.

[15] Resonate Inc. 2003. http://www.resonate.com.

[16] Cherkasova L, Karlsson M. Scalable web server cluster design with workload-aware request distribution strategy with WARD. In: Proceedings of the 3rd International Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems. Los Alamitos: IEEE Computer Society Press, 2001. 212~221.

[17] Banga G, Druschel P. Measuring the capacity of a web server. In: Proceedings of the USENIX Symposium on Internet Technologies and Systems (USITS). Berkeley: USENIX Association, 1997. 61~72.