并行WWW服务器集群请求分配算法的研究*

邸 烁 郑纬民 王鼎兴 沈美明

(清华大学计算机科学与技术系 北京 100084)

E-mail: dishuo@263.net

摘要 为了有效地提高WWW服务器的吞吐能力、反应速度和可扩展性,国际上许多繁忙站点纷纷转向采用并行WWW服务器集群来替代原有的单一主机服务器.这些站点普遍采用请求分配技术,即集中接收所有到达的HTTP请求,然后"均衡"地分配到集群中的各个服务器进行处理.常用的转轮法、最少连接法和最快连接法等算法在分配请求时,要么对集群中各个服务器的性能不加区分,要么不考虑请求的具体内容,在实际系统中效率较低.文章提出了一种适用于异构集群的局部最优请求分配算法(least time increment,简称LTI),综合考虑服务器性能差异、请求内容和服务器当前负载等因素,作为请求分配的依据.文章还提出了LTI算法的改进版LTI+,能够判别和避免集群进入临界状态.文章给出了算法的理论分析和实验测试结果.在同等条件下,此算法能够达到较小的平均应答延迟和较大的吞吐能力,从而能更好地挖掘集群的并行处理能力,优化集群的整体性能.

关键词 WWW服务器集群,HTTP协议,请求分配,负载均衡,请求分配算法,临界状态.

中图法分类号 TP393

Internet上的许多热门WWW站点,随着访问人数和访问频度的不断增加,开始面临WWW服务器超载的问题[1].为了增大吞吐能力,提高反应速度,人们不得不升级或更换服务器.这里有两种选择,一种是采用昂贵的高性能主机或SMP计算机,另一种则是把多台服务器用局域网络联结成一个集群,通过并行处理来扩展性能[1~3].比较这两种方案,后者的成本较低,并且具有较大的灵活性和较高的可靠性,目前有许多著名的WWW站点已经转向此类计算平台.我们称这种平台为并行WWW服务器集群.

请求分配[2,3]是并行WWW服务器集群中采用的一种典型技术.其主要原理,是由一台特殊的计算机(一般称为请求分配器)集中接收所有的HTTP请求,然后依据一定的原则把它们分配到集群中的各台服务器上去进行处理.分配的主要目的是使各台服务器的负载分布比较均衡,从而获得较高的整体吞吐能力和较快的反应速度.目前常用的请求分配算法主要有转轮法[2]、最少连接法和最快连接法[3]3种.第1种算法实现简单,但从本质上讲没有考虑负载均衡问题.第2种算法对各个服务器的性能不加区分,而且不考虑请求的强度(请求文件的长度).第3种算法虽然考虑了服务器的性能,但没有考虑请求强度.这些局限性使这些算法无法真正做到负载的"均衡"分配,因而无法较好地发挥系统的并行处理能力.

为了有效地提高请求分配算法的效率,并使算法能够适应异构服务器集群,在设计并行WWW服务器集群系统时,应使请求分配器能够知道每一台服务器的处理能力,并且可对集中接收的每一个HTTP请求的内容进行分析,同时应能比较准确地跟踪各个服务器的负载情况.服务器的处理能力可以事先通过实测获得,分析HTTP请求只需分析它的请求头部分,而负载跟踪则可以通过在请求分配器上记录各服务器的应答进度来实

---

* 本文研究得到国家863高科技项目基金资助.作者**邸烁**,1970年生,博士,主要研究领域为计算机网络,并行处理.**郑纬民**,1946年生,教授,博士生导师,主要研究领域为并行机群系统,多机处理技术.**王鼎兴**,1937年生,教授,博士生导师,主要研究领域为并行/分布处理,智能技术与系统.**沈美明**,1938年生,教授,博士生导师,主要研究领域为并行/分布处理,并行机群系统.

本文通讯联系人:邸烁,北京 100084,清华大学计算机科学与技术系

本文1998-03-20收到原稿,1998-08-03收到修改稿

现.基于以上考虑,我们实现了一个异构 WWW 服务器集群,设计了一种综合考虑 HTTP 请求内容和各个服务器性能以及当前的负载情况的请求分配算法 LTI(least time increment),目标是使各个请求的应答时间总和尽可能短.为此,我们把应答总时间的增量作为 LTI 算法的优化对象.为了避免服务器在连接数接近阈值时性能急剧下降,我们还设计了一种请求延迟策略,有效地解决了这个问题.无论是对算法的理论分析,还是对系统的实际测试,LTI 算法的表现都比转轮法和最少连接法和最快连接法要好.

本文第 1 节介绍了我们所提出的集群系统的组成和工作原理.第 2 节简要介绍了性能评价的几个参数,然后分析了影响请求分配性能的若干因素,给出一些定量分析结果.第 3 节给出请求分配算法 LTI 的描述,并对其进行分析和改进.第 4 节给出我们对系统的测试结果,并与最少连接法和最快连接法的结果进行比较.最后,第 5 节给出本文的结论,并对我们未来要做的工作作一介绍.

1 系统组成和工作原理

目前常见的并行 WWW 服务器集群主要有两种组成方式,一种是以 Cisco 的 LocalDirector[3]为代表的隔离式,采用最少连接法或最快连接法进行请求分配;另一种则是以 NCSA 的 Scalable Web Server[2]为代表的非隔离式,采用转轮法进行请求分配.无论哪一种方式,都要求各台服务器上所有 Web 信息的访问路径和内容要完全一样(这一点可以通过网络文件系统或专用的文件服务器来实现).二者的区别在于这些服务器是否直接连接在 Internet 上,对用户可见.隔离式集群采用类似于 Proxy[4]的技术,只有请求分配器具有一个虚拟的 IP 地址,所有的请求都发向此 IP 地址,由请求分配器分配到集群中的各台服务器上去处理,返回的结果也经由请求分配器传回给用户;非隔离式集群中的每一台服务器都有独立的 IP 地址,请求分配是通过动态 DNS[2],HTTP Redirect[3]等技术实现的,服务器对请求的应答不通过请求分配器,而是由各台服务器直接传回给用户.图 1 是两种不同结构集群的组成示意图.

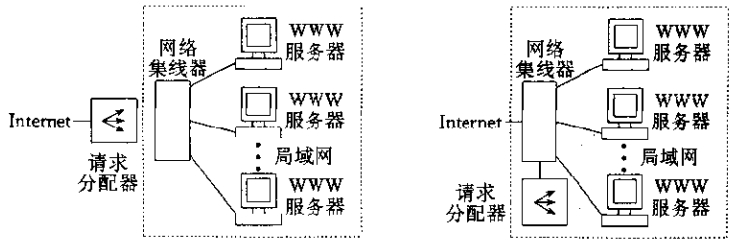

图1 两种不同结构的并行WWW服务器集群的组成结构

我们的集群系统采用的是隔离式方法,集群由 Fast Ethernet 联结两台 Power PC 和两台 Sun Workstation 组成(Power PC 的性能是 Sun Workstation 的两倍),其上运行 UNIX 操作系统和 NCSA HTTPD,请求分配器由一台运行 Windows NT 的 PC 充当.文件系统采用 NFS,请求分配器采用多线程 Proxy 技术.

2 性能评价参数和影响性能的因素

目前,评价 WWW 服务器的性能主要采用吞吐率和平均应答延迟两个参数[5].吞吐率 RPS 定义为平均每秒处理的请求个数,平均应答延迟 MRD 定义为从建立网络连接到应答结束拆除连接之间的时间的平均值.所采用的测试集由若干个大小不一的请求按照一定的分布规律组成.所谓请求的大小,是指请求读取的文件的长度或请求执行的 Script 程序在独占处理机时的执行时间.

HTTP 协议[6]是一种无状态协议,每个请求都由客户程序发起建立一个独立的网络连接,应答结束后由服务器拆除该连接.请求由请求头和请求体两部分组成,请求头是一种具有行结构的文本,请求体是一种称为 MIME 格式的复合文本.请求头指明该请求的目的,如,向服务器请求读取文件、验证文件是否更新或执行服务器上的 Scripts 程序等.不同的 HTTP 请求对服务器产生的负载存在着很大的差异.

RPS 和 MRD 由服务器的处理能力决定. 对于 WWW 服务器集群来说,MRD 还和请求分配策略有密切的联系. 下面我们来分析这些联系.

完成一个应答的时间主要由以下 3 部分组成[7]:

(1) 请求分配器与服务器建立网络连接所需的时间 T_C. 这个时间一般可以忽略.

(2) 读取并传输文件的时间 T_F. 这段时间取决于文件读取和传输的速度 ω. 如果在一段时间内,网络连接数 N 保持不变,则在这段时间内,ω 可以近似地认为与 N 成反比,即 $\omega=\tau/N$. 这里,τ 为只有一个连接时的文件读取和传输速度,对一个特定的服务器,可以由实测获得. 我们称 ω 为应答速度,τ 为机器速度.

(3) 在服务器上执行 Scripts 程序的时间 T_E. 由于情况比较复杂,这段时间很难事先估计. 为了简化问题,在下面的讨论中暂时不考虑 T_E.

这里有一点需要说明:当网络连接数 N 超过某一阈值 N_{Max}(取决于服务器的性能,对一个特定的服务器,可以由实测获得)时,T_C 会突然直线上升,同时 ω 会突然急剧下降[8],如图 2 所示. 我们称这种状态为临界状态,称 N_{Max} 为此服务器的临界连接数. 这种现象可以这样来理解:这些处理要消耗服务器的缓冲区资源,当缓冲区总数接近或超过系统内存限度时,系统开始大量使用"换存",如果此时仍不断有请求到来,则势必产生一种恶性循环,于是系统效率急剧下降. 所以,为了保证系统的 RPS 和 MRD,应设法避免 N 达到 N_{Max}.

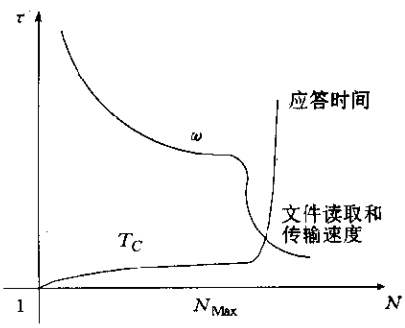

图2　T_C和ω在 N超过 N_{Max}时急剧变化

下面我们来分析有一台服务器时的应答时间.

设在某一时刻该服务器共有 N 个网络连接在同时传输文件,这些文件的剩余长度分别为 $L_1,L_2,\ldots,L_N$. 此时,一个新的 HTTP 请求到达该服务器,请求的文件长度为 L. 不妨设 $L_1\leqslant\ldots\leqslant L_K\leqslant L<L_{K+1}\leqslant\ldots\leqslant L_N$,采用连续模型(传输文件的时间与文件的长度成正比,与机器速度成反比)可以计算(此处省略计算方法)由于 L 的引入而产生的应答时间的增量:

$$\Delta=\frac{2}{\tau}\sum_{i=1}^{K}+\frac{1}{\tau}(2N+1-2K)L. \tag{1}$$

这个结果是下文算法的主要理论依据.

3　请求分配算法

为了实现算法,我们在请求分配器上设置几个表. 第 1 个称为 FLT,其中的内容为服务器上 WWW 空间中所有文件的路径和长度. 第 2 个称为 TST,其中存放的是各个服务器的机器速度. 第 3 个称为 RCT,存放各个服务器正在应答的请求个数. 设集群中共有 M 个服务器,为每个服务器设置一个表 CFT(i),$1\leqslant i\leqslant M$,用来存放此服务器当前正在应答的各个请求所传输文件的长度和剩余长度. 请求分配算法 LTI 采用多线程技术,算法主干部分由两个线程组成,一个用于跟踪各服务器的应答进度,一个用于请求分配.

算法 1. 请求分配算法 LTI.

线程 1:(负载跟踪)

每隔一个固定的小时间段 θ 进行如下操作:

(1) 根据 TST 和 RCT 计算各个服务器的应答速度 ω.

(2) 将所有 CFT(i)中各个文件的剩余长度域减少 $\omega\cdot\theta$,$1\leqslant i\leqslant M$.

如果某文件的剩余长度$\leqslant$0,则删除相应 CFT(i)的对应项,并更新 *RCT* 中此服务器的当前请求个数.

线程 2:(请求分配)

(1) 如果有请求到达,从请求头中的 URL 部分取出请求文件的路径,从 FLT 中查得其请求文件的长度 L.

(2) 对每个服务器 i,根据 TST,RCT 和 CFT(i),采用式(1)的方法计算将 L 引入该服务器所引起的应答

总时间的增量 Δ_i, $1\leqslant i\leqslant M$.

(3) 设 $\Delta_K=\mathrm{Min}(\Delta_i)$, $1\leqslant i\leqslant M$,分配请求 L 到第 K 个服务器,同时更新 RCT 中此服务器的当前请求个数,并在 CFT(i)中增加一项,其中文件长度域和剩余长度域均 1 为 L.

LTI 算法把应答总时间的增量作为优化的对象,关于它的性能,我们有以下定理.

定理. 对于优化 MRD 来说,LTI 算法是一个局部最优算法.

证明:设一个测试集共有 n 个请求,由式(1)和 MRD 的定义,有

$$MRD=\frac{1}{n}\sum_{i=1}^{n}(T_C^{(i)}+\Delta^{(i)})=\frac{1}{n}\sum_{i=1}^{n}T_C^{(i)}+\frac{1}{n}\sum_{i=1}^{n}\Delta^{(i)}.$$

其中 $T_C^{(i)}$ 为各个请求的连接建立时间,按上文讨论为固定值;$\Delta^{(i)}$ 为各个请求在以前所有请求的基础上引起的应答总时间的增量.在 LTI 算法中,分配新到达的每一个请求时,都使相应的 $\Delta^{(i)}$ 取得了当前状态下可能的最小值,所以,从局部意义上讲,该算法是最优的. □

上文曾指出,当连接数 N 接近或超过某一阈值 N_{Max} 时,服务器的性能会急剧下降.上述算法没有考虑这个问题,其结果是在负载很重时,应答时间将大大超过预计值,于是算法的准确性就得不到保证了.解决这个问题可以有几种方法.第 1 种方法是拒绝新连接,直到有服务器退出临界状态.第 2 种方法是将新到达的请求分配到另外一台服务器上去.第 3 种方法是将新到达的请求延迟,等到有服务器退出临界状态时再予以处理.

上述第 1 种方法因为用户很难接受,在实际中是不可行的,故我们不予考虑.我们对第 2 种方法进行了实验研究,发现按照 LTI 算法选出的服务器达到连接阈值时,其他服务器也基本上都接近临界状态,因此,这种方法起不到明显的效果.我们结合第 2 和第 3 种方法,对 LTI 算法进行如下改进.

修改表 RCT,其中除了存放各个服务器正在应答的请求个数 N 以外,还存放它们的临界连接数.增加队列 DRQ,存放因临界状态而延迟的请求及其请求文件的长度.一个服务器接近临界状态是指它的连接数 N 接近但还未达到临界连接数 N_{Max},此时,服务器的性能还没有急剧下降.

算法 2. 请求分配算法 LTI+.

线程 1:(负载跟踪)与 LTI 算法相同.

线程 2:(请求分配)

(1) 如果有请求到达,从请求头中取出请求文件的路径,从 FLT 中查得其请求文件的长度 L.如果此时没有未接近临界状态的服务器,则将这个请求放入延迟队列 DRQ 的队尾,并填入其请求文件的长度;否则:

(2) 对所有未接近临界状态的服务器 i,根据 TST,RCT 和 CFT(i),采用式(1)的方法计算将 L 引入该服务器引起的应答总时间的增量 Δ_i, $1\leqslant i\leqslant M$.

(3) 设 $\Delta_K=\mathrm{Min}(\Delta_i)$, $1\leqslant i\leqslant M$,分配请求 L 到第 K 个服务器,同时更新 RCT 中此服务器的当前请求个数,并在 CFT(i)中增加一项,其中文件长度域和未传长度域均为 L.

线程 3:(处理被延迟的请求)

如果延迟队列 DRQ 非空,每隔时间 θ 检查是否有服务器退出临界状态.如果有,则进行如下操作:

(1) 取出 DRQ 队首的请求,设其请求文件的长度为 L.

(2) 对所有未接近临界状态的服务器 i,根据 TST,RCT 和 CFT(i),采用式(1)的方法计算将 L 引入该服务器所引起的应答总时间的增量 Δ_i, $1\leqslant i\leqslant M$.

(3) 设 $\Delta_K=\mathrm{Min}(\Delta_i)$, $1\leqslant i\leqslant M$,分配请求 L 到第 K 个服务器,同时更新 RCT 中此服务器的当前请求个数,并在 CFT(i)中增加一项,其中文件长度域和未传长度域均为 L;

算法 LTI+增设了一个线程处理被延迟的请求.当服务器集群未接近临界状态时,该算法与算法 LTI 完全相同.当服务器集群接近临界状态时,该算法通过延迟新到达的请求避免集群进入临界状态.对于具有间隙性突发特性的请求序列,该算法实际上是将突发流量均衡到以后的时间中去处理.

4 实验结果的比较和分析

本文第 1 节介绍了我们实现的一个服务器集群.我们在这个平台上对 LTI 和 LTI+算法进行了实验研究,

并与最少连接法和最快连接法进行了比较.

依照文献[9]对请求强度分布的研究，我们设计了一个由不同强度的请求所组成的测试集. 请求强度的分布如表 1 所示.

表 1　请求强度的分布

文件大小(Bytes)	500	5K	50K	500K
所占比例(%)	35	50	14	1

测试用的请求发生器由 4 台 Power PC 充当，上面运行请求发生程序. 每个请求发生程序可以模拟 1～50 个 WWW Client 进程，每个 Client 进程都能按照表 1 的比例不断发出 HTTP 请求，并且能够记录这些请求的应答时间. 实验结果如图 3～6 所示. 图 3 和图 5 给出了 LTI 算法、最快连接算法和最少连接算法在 Client 数从 5 开始逐渐增大时平均应答延迟和吞吐率的变化情况. 图 4 和图 6 给出了 LTI+算法与 LTI 算法在 Client 数从 100 变化到 200 时平均应答延迟和吞吐率的变化情况.

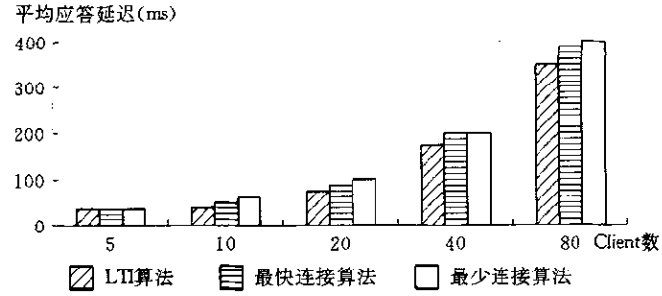

图3　3种算法在不同Client数时的MRD

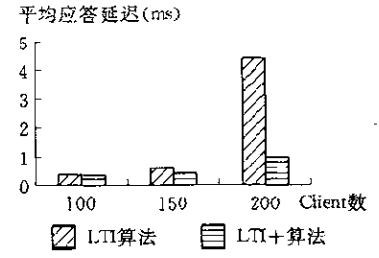

图4　LTI和LTI+算法MRD的比较

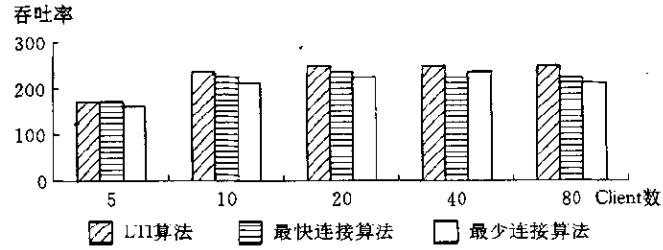

图5　3种算法在不同Client数时的RPS

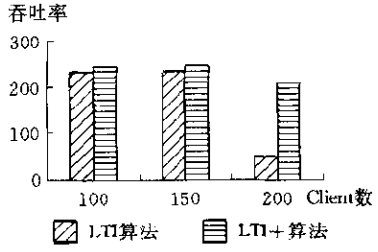

图6　LTI和LTI+算法RPS的比较

图 3 和图 5 的结果证明了如下事实：与最少连接法和最快连接法相比，LTI 算法确实能够有效地减少平均应答延迟和增加吞吐率，当 Client 数较小时，这种性能改善不太明显，当 Client 数增大一些后，LTI 算法的优势才逐渐表现出来. 这种现象说明：在集群的负载较轻的时候，按照 3 种算法分配请求都可以达到均衡负载的目的；当各服务器都达到一定的负荷(连接数达到一定数量)后，LTI 算法对集群负载的跟踪和分析则更为精确，对负载的分配更为均衡.

从图 4 和图 6 可知，在 Client 数进一步增大后，LTI 算法的性能也开始大幅度下降，此时，LTI+的优越性就逐渐体现出来了. 这是因为，随着负载的加重，各服务器均纷纷接近临界状态，LTI+算法判别出了这种状态，采取措施有效地避免了这种趋势的延续，而 LTI 算法没有这种能力.

5　结　论

本文简要介绍了 WWW 服务器集群的组成和工作原理，给出了评价其性能的两个参数，并分析了影响性能的因素，特别是分析了应答总时间的增量与请求强度以及服务器当前状态之间的关系. 在此基础上，我们提出了以应答总时间的增量为优化目标的请求分配算法 LTI，并证明了该算法的局部最优性. 为了解决集群在接近临

界状态时性能急剧下降的问题，我们对LTI算法进行了改进，得到算法LTI+，该算法能够判别和避免集群进入临界状态.最后，我们给出了一些实验结果，通过与其他算法进行比较，证实了LTI和LTI+算法的有效性：它能够更加精确地跟踪和分析集群系统的负载，更加均衡地分配负载，从而达到较好的性能.

在本文的讨论过程中，为了简化问题，暂时没有考虑一些实际中存在的因素，比如，我们在计算读取并传输文件时间时采用了T_F连续模型，没有考虑在服务器上执行Scripts程序的时间T_E等.因而我们的算法还具有一些局限性.目前我们正在对这些问题进行更加深入的研究.此外，我们还在全局文件系统(为集群中各个服务器提供统一的文件访问路径)的构造和对象预取等方面寻找提高性能的途径.

参考文献

1 Kwan Thomas T, McGrath Robert E. NCSA's world wide web server: design and performance. IEEE Computer, 1995, 28(11):68～74

2 Katz Eric Dean, Butler Michelle, McGrath Robert E. A scalable HTTP server: the NCSA prototype. Computer Networks and ISDN Systems, 1994,(27):155～164

3 Cisco Inc.. Scaling the World Wide Web. Technical Report, http: //www.cisco.com

4 Luotonen A, Altis K. World wide web proxies. In: Proceedings of the 1st International Conference on the World-Wide Web. 1994

5 Trent Gene, Sake Mark. WebSTONE: the first generation in HTTP server benchmarking. Whitepaper, Silicon Graphics Inc., February 1995

6 Berners-Lee T, Fielding R T, Frystyk H. Hypertext transfer protocol-HTTP/1.0. Informational RFC 1945, Network Working Group, May 1996

7 Padmanabhan Venkata N, Mogul Jeffrey C. Improving HTTP latency. Computer Networks and ISDN Systems, 1995, (28):25～35

8 Slothouber Louis P. A Model of Web Server Performance. April 1997. http: // louvx.biap.com/webperformance/modelpaper.html

9 Carlton Alexander. An Explanation of the SPECweb96 Benchmark. Whitepaper, Standard Performance Evaluation Corporation, 1996

Research on Request Dispatching Algorithm for Web Server Clusters

DI Shuo ZHENG Wei-min WANG Ding-xing SHEN Mei-ming

(*Department of Computer Science and Technology Tsinghua University Beijing* 100084)

Abstract In order to improve the throughput, response speed and scalability of Web servers efficiently, many famous Web sites have thrown away single servers and turned into Web server clusters. Request-dispatching is a technology used by these sites, which centrally accepts all the coming requests and "evenly" dispatches them to the servers in the cluster. "Round-robin", "Least-connections" and "Fastest Connection" algorithms which Commonly used request-dispatching show low efficiencies, because they either do not consider the difference in performance between servers or do not consider the contents of coming HTTP requests. In this paper, the authors give a novel algorithm designed for heterogeneous Web server clusters, called LTI (least time increment), which considers not only the deference in performance between servers, but also the contents of coming HTTP requests and the current load level of each server in the cluster. They also give the reformed edition of LTI algorithm, namely LTI+, which can judge whether the cluster is closed to the critical state and prevent such trend. Theoretical analysis and real system test show that the algorithm achieves better response latency and throughput than "Least connections" and "Fastest Connection" algorithms. So it makes the Web server clusters more parallel and more scaleable.

Key words Web server clusters, HTTP protocol, request dispatching, load balancing, request dispatching algorithm, critical state.