基于价格和信息预测的 Web Servers 设计

A Price-based Web Servers Desing with Load and Cache Prediction

周 刚 李天宁 陈道蓄 谢 立

(南京大学计算机软件新技术国家重点实验室 南京 210093)

Abstract Increasing WWW visits require efficient Web Servers architecture. In our design, Web Servers are a cluster consisting of replicated Servers. Server predicates its load, which is used to decide the cost to visit itself. Client makes the load balancing decision, and caches the predicated load of Cluster. Based on price theory, Client chooses the server of the lowest cost to serve itself; at the same time, Server predicates the information updating speed of Cluster, and uses it to decide the updating speed of Cache. Finally, the system topology is given and characters are analyzed.

Keywords Information prediction, Price, Load balancing, Intelligent client, Scalability

1. 引言

随着 World Wide Web 的快速增长，热门站点的访问率成指数级增长，这就使许多 Server 严重过载，服务的响应时间大大加长，极大降低了服务质量，将原来的单个 Server 进行硬件升级，既不经济也不现实，因为硬件的升级是有限的。Web Servers 组织方式较好地解决了这一问题，Server 可以在局域网内形成 Cluster[1,2]，也可以采用分布式结构将机器分散在不同的地点[3]，我们采用的是局域网内的 Cluster 结构，该结构保证了 Server 间的通信速度。Server 的负载平衡决策可以做在 Server 端，但是单点故障和瓶颈现象非常明显；也可以做在 DNS 中，但极大地受到了 Internet 中 Cache 的影响；做在 Client 端虽然解决了单点故障和瓶颈现象，但有信息滞后和网络传输大的缺点。本文在负载平衡决策时利用了智能 Client 的灵活性，同时用负载预测克服了信息的滞后性，采用 Server 控制 Client 端 Cache 的更新速度又有效减少了不必要的网络传输，减轻了网络负担。

本文如下组织：2 介绍了 Web Servers 的现有解决方案，并分析其主要缺点；3 提出了负载预测技术、Server 性能评价方法和负载平衡方案；4 分析了传统 Cache 更新算法的缺点，提出了由 Server 端控制的方法；5 给出了系统结构并分析其特点；最后给出结论。

2. 相关工作

Web Servers 中目前常用的负载平衡方法主要包括 HTTP 重定向，基于 DNS 的负载平衡修改 IP 包和智能 Client 等。

HTTP 重定向[4] 用 HTTP 返回 URL Redirection[5]码来实现负载平衡。当一个 Server 负载非常重时，就不对接收到的请求进行服务，而是将请求发给另一个负载较轻的 Server。这种方法有如下缺点：加重了网络负载；容易出现单点故障；并导致系统瓶颈。

基于 DNS 的负载平衡[6~8] 将负载平衡做在域名服务器中，当 Server 端的域名服务器收到请求时，就采用循环的方法或其它方法将它映射到不同的 IP，由不同的 Server 分担负载。它有如下缺点：Internet 上的大量域名服务器对通过它的域名到 IP 的映射做了 Cache，影响了 DNS 的负载平衡效果。

修改 IP 包[9] 在 Server 方设计一个模块，它截获所有发向 Server Cluster 的 IP 包，根据一定的算法，修改 IP 包，再发向 Server Cluster 中的某个 Server。有如下缺点：单点故障，出现瓶颈。

智能 Client[10] 由 Client 来完成域名到具体 Server 地址的映射，由 Client 来进行负载平衡。但 Client 在负载平衡决策时所用的 Server 负载信息不能及时反映 Server 当前的负载情况，有滞后性；Client 因为缓存 Server 的负载信息又带来额外的网络传输，加重了网络负担。

本文在负载平衡决策时利用了智能 Client 的灵活性，同时用负载预测克服了信息的滞后性，采用 Server 控制 Client 端 Cache 的更新速度又极大减少了不必要的网络传输，减轻了网络负担。

3. 负载平衡

3.1 负载预测

在 Web Servers 中，每一个 Server 不断地检测自

身的负载，统计最近的请求到达情况和任务完成情况，预测将来的一段时间内自己的负载，并将它发送给 Server Manager。Server Manager 自始至终都有一个端口用于监听，收集各个对等 Server 的负载信息。

对 Cluster 中的某一个具体的 Server，定义如下变量：

在 T_{n-1} 到 T_n 时间段内被 Server 接受的任务请求为 R_n。

在 T_{n-1} 到 T_n 时间段内 Server 完成的任务请求为 Ro_n。

在 T_{n-1} 到 T_n 时间段内 Server 的负载为 F_n。

预测 Server 在 T_n 到 T_{n+1} 时间段内负载为 F'_{n+1}。

我们有 $F'_{n+1}=F_n e^{\theta(R_n-Ro_n)}$，$\theta>0$，当 T_{n-1} 到 T_n 时间段内接受的任务多，而完成的任务少时，可以认为在将来的 T_n 到 T_{n+1} 时间段内 Server 的负载较 T_{n+1} 到 T_n 时间段内 Server 的负载 F_n 有所增加，即 $e^{\theta(R_n-Ro_n)}>1$，$F'_{n+1}>F_n$；同样当 T_{n+1} 到 T_n 时间段内接受的任务少，而完成的任务多时，可以认为在将来的 T_n 到 T_{n+1} 时间段内 Server 的负载较 T_{n+1} 到 T_n 时间段内 Server 的负载 F_n 有所减少，即 $e(R_{1n}-Ro_n)<1$，$F'_{n+1}<F_n$，这就很好地预测了 T_n 到 T_{n+1} 时间段内 Server 的负载。此外，Server Manager 和各 Server 位于同一局域网内，网络传输非常快，让 Server Manager 进行预测负载的收集不会对 Web Servers 系统的运行有太大影响。

3.2 Server 性能评价

在 Cluster 中的各个 Server 是对等的，是指每一个 Server 提供的具体服务是相同的，对来自 Client 的每一个具体的请求，既可以由 Server A 服务，又可以由 Server B 服务。但是 Server A 和 Server B 的硬件配置可能不同，而这又是不可避免的。在构建一个 Web Server 提供的具体服务时，Server A 可以是最初建立服务时就有的机器，Server B 就可能是为了满足日益增长的服务需求而后加入的硬件性能更高的服务器，显然，它们的服务能力有区别。对 Server 的各个硬件参数打分：

C_1 表示 CPU 的性能，C_2 表示内存大小，…，等等，

则 Server i 的性能表示为 $M_i=\omega_1C_1+\omega_2C_2+\cdots+\omega_nC_n$，$\forall\ 0\leqslant i\leqslant n$，$\omega_i>0$，$\omega_i$ 用来表示 C_i 影响 M_i 的相对力度。

3.3 负载平衡决策

系统中，我们将 Couster 中的每个 Server 定一个价格，用来反映其负载与性能的综合考虑。价格随 Server 负载的加重而变高，随 Server 负载的减轻而变低；负载平衡做在 Client 端，认为 Client 就是市场中的买方，Client 有请求需要服务时，就比较一下当前各 Server 的价格，选择价格低的 Server 为其服务。这样，负载重的 Server 就会少接收到一些服务请求，缓解了负载，负载本来就轻的 Server 就会多承担一些服务，从而达到负载平衡。

Server 中，负载是影响价格的一个重要因素，性能评价指数是影响价格的另一个重要因素。Client 做负载平衡决策时，除了 Server 的价格外，另一个考虑因素就是 Client 对 Server 运行时的价格的敏感度，我们可以有下面的公式：

$$P_i=\frac{1/(F'_{n+1}*Tr_i)^k}{\sum_i 1/(F'_{n+1}*Tr_i)^k},\quad Tr_i=ce^{-w_i},\quad k>0,c>0$$

$F'_{n+1}*Tr_i$ 就是 Server i 的价格，参数 C 调节 M_i 和 F'_{n+1} 影响价格的相对力度；k 在此处是为了调节 Client 对价格的敏感性：当 k＝0 时，价格对 Client 负载平衡决策没有影响，Client 随机分配负载；随 k 增大，Server 的价格对 P_i 的影响加大[11]，因为 Client 总是选择 P_i 最小的 Server 进行服务请求连接[12]，越来越多的请求就会发向性能高、负载轻的 Server。

值得说明的是，此外的 F'_i 不是 Server 当前（F_{n-1} 到 T_n 时间段）的真实负载，而是 Server 在 T_{n-2} 到 T_n 时间段的预测负载，是 Server 在 T_{n-2} 到 T_{n-1} 时间段时对 Server 在 T_{n-1} 到 T_n 时间段的负载的一个预测。不用 T_{n-1} 到 T_n 时间段的真实负载 F_n 的原因在于：若 Server Manager 将 Server 的真实负载 F_nPush 给 Client，当 Client 运用 Cache（Client 用于缓存 Server 的负载信息）中的 F_n 进行决策时，此时 Server 的真实负载已经是 F_{n+1} 了，信息有滞后性。所以 Client 的 Cache 中缓存的是 Server 的预测负载，而不是真实负载[13]。

4. Cache 更新

4.1 Applet 带来的问题

Client 用 WWW 浏览器获得 Web Server 提供的服务，在 Client 端我们做了一个 Cache 模块（Applet），用于收集预测负载信息，使得 Cache 及时反映 Server 的信息变化，最理想的设计策略，是让 Client 方开一个 Server 端口来监听 Web Server 的负载变动信息。Client 初起时，就将监听端口告诉 Server Manager，当 Web Server 中负载信息变动，如有机器加入或退出服务时，由 Server Manager 发一个消息给 Client 端监听端口。然而，这种设计方案不是可行的，因为 WWW 浏览器中的 Java Applet 不可以有 Servers 端口的存在。当然，若每个 Client 与 Servers Manager 至始自终保持一个 Socket 连接，就可以实现负载信息的实时更新，但带来的额外网络负载大得无法接受。

4.2 传统的 Cache 更新方法

传统的方法是采用轮询或 Lazy Update。轮询就是 Client 方开一个进程，它自始至终不断地向 Server Manager 发请求，而 Server Manager 不断将 Cluster 的最新预测负载、机器加入和退出信息回发给 Client，这无疑增加了不必要的、有时是非常大的网络传输开销。当 Client 的请求间隔越小时，网络传输就越大，而其中很大一部分是不必要的；当 Client 的请求间隔很大时，网络传输虽然变小了，但不能及时反映 Server 的信息变更，严重影响了 Client 负载平衡决策。Lazy Update 方法，是在 Client 初起时，Server Manager 一次性将预测负载传给 Client，而后 Client 选择具体的 Server 为其服务。当 Server 没有负载变动时，Server 就正常响应 Client 的请求；当 Server 发生负载变动时，Server 就将变动后的信息附在正常的对 Client 请求的回答上，节省了网络开销，该方法有一个明显的缺点，那就是只有被 Client 请求的 Server 的信息会得到更新，其它 Server 的信息得不到更新，因为 Client 是根据 Cache 中的所有 Server 信息做负载平衡决策，所以这严重影响了负载平衡效果。

4.3 信息更新速度预测

当 Cluster 中预测负载变更速度加快时，我们希望 Client 以更高的频率向 Server Manager 发出信息更新请求；当 Cluster 中预测负载变更速度减慢时，Client 以更低的频率向 Server Manager 发出信息更新请求。在 T_{n-1} 到 T_n 时间段内，Cluster 的负载变更大小为：$\sum_i |F_{i_n} - F_{i_{n-1}}|$；在 T_n 到 T_{n+1} 时间段内，Cluster 的预测负载变更大小为 $\sum_i |F^*_{i_{n+1}} - F_{i_n}|$。设 Client 的最近的一次信息更新请求时间为 Rt_i，Client 在 Rt_i 时间向 Server Mavager 发出信息更新请求，Server Manager 在发回更新后的信息的同时，还告诉 Client 下一次信息更新请求到这一次信息更新请求的时间间隔 Rd_i，Rd 的计算公式如下：

$$Rd_{i+1} = Rd_i e^{-\theta(\sum_i |F^*_{i_{n+1}} - F_{i_n}| - \sum_i |F^*_{i_n} - F_{i_{n-1}}|)}, \theta > 0$$

当 Cluster 中的负载变更速度变大时，$e^{-\theta_1(\sum_i |F^*_{i_{n+1}} - F_{i_n}| - \sum_i |F^*_{i_n} - F_{i_{n-1}}|)} < 1$，$Rd$ 变小，信息更新速度加快，反之减慢，这样就实现了用预测的负载变更速度（$\sum_i |F^*_{i_{n+1}} - F_{i_n}| - \sum_i |F^*_{i_n} - F_{i_{n-1}}|$）来预测信息的更新速度（$Rd$），从而控制了信息的变更速度（$Rd$）。

此外，Cluster 中的机器加入或退出服务，也表示 Cluster 的原有负载平衡被打破，应该加快 Cache 信息的更新速度。若只考虑机器的加入和退出对 Cache 更新速度的影响，有：$Rd_{i+1} = Rd_i e^{-\theta_2(I_n + O_n)}, \theta_2 > 0$，其中 I_n 为在 T_{n-1} 到 T_n 时间段内加入 Cluster 的新机器数，O_n 为在 T_{n-1} 到 T_n 时间段内退出 Cluster 的机器数。综合以上两个方面的考虑，最后得出负载变更速度、机器加入和机器退出信息共同控制 Cache 更新速度的计算公式：

$$Rd_{i+1} = Rd_i e^{-\theta_1(\sum_i |F^*_{i_{n+1}} - F_{i_n}| - \sum_i |F^*_{i_n} - F_{i_{n-1}}|) - \theta_2(I_n + O_n)},$$

$$\theta_2 > 0, \theta_2 > 0$$

θ_1 用来调节 $\sum_i |F^*_{i_{n+1}} - F_{i_n}| - \sum_i |F^*_{i_n} - F_{i_{n-1}}|$ 和 $I_n + O_n$ 影响 Rd_{i+1} 的相对力度。有一点要说明的是，Rd 的下标（i）与 F^*、F、I、O 的下标（n）不一样。Rd 是由 Server Manager 计算后传送给 Cache，Client 有一次信息更新请求，Server Manager 就传送一次，i 就用来对此计数，i 值的变化快慢直接反映了 Cache 信息的更新频率；F^*、F、I、O 的值在 Web Server 中计算，由各个 Server 收集 F、I、O，再由 Server Manager 计算 F^*，n 值的变更速度表示的是 Cluster 中 Sercer 的信息向 Server Manager 报告本地信息的及时程度，由于 Server Manager 和 Cluster 在同一局域网中，所以 n 值变化得很快。

5. 系统结构及特点

5.1 系统结构

如图 1，整个 Web Servers 系统构架于 Internet 上，Client 用浏览器提交和接收服务，Cache 和 Load Balancing 是用 Java 语言写的 Applet，集成到 Web Serves 的访问页面上。

Client 在发出请求前，Load Balancing 模块根据 Cache 中的信息计算出最合适的 Server（P 最低），然后在 Cache 中找出对应的地址，将服务请求发送出去。Client 同时不断向 Server Manager 取得 Cluster 的预测信息，更新 Cache 中的内容，并得到下次向 Server Manager 发出 Cache 更新请求的时间延迟 Td。Client 第一次请求服务时，向 Server Manager 索取 Web Servers 的负载信息，以后请求服务时就不从 Serer Manager 经过，直接将请求发给选定的 Server。

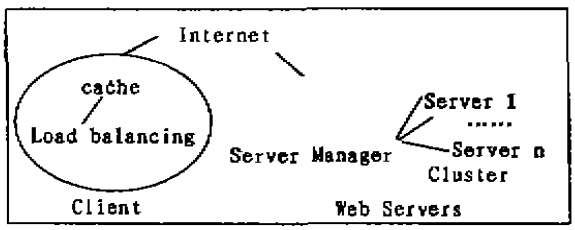

图 1 系统拓扑结构

Server Manager 自始至终都在收集 Server 的负载变更信息，同时关注是否有 Server 的加入和退出，

用这些信息计算出 Web Servers 的信息变更频度，以此来决定 Client 端 Cache 的更新频度，减少不必要的网络开销。Cluster 中的 Server 不断检测本机的性能，及时向 Server Manager 报告负载变化、任务到达和任务完成的情况，供 Server Manager 计算 Rd。表 1 列出了一些重要参数在系统各个模块中的分布情况：

表 1　各重要参数在系统模块中的分布情况表

	F_{n+1}	M	Rt_n	RD_n	F_n	Rd	I_n	O_n	P	对应 Server 的地址
Cache	√	√				√				√
Load Balancing									√	√
Server Manager	√	√			√	√	√	√		√
Server		√	√	√	√					√

5.2　系统特点

· Web Servers 组织简单高效，易于 Server 的增加和减少，便于硬件升级；

· 利用了智能 Client 的灵活性，负载平衡性好；

· 用负载预测克服了信息的滞后性；

· 采用 Server 控制 Client 端 Cache 的更新速度，极大减少了不必要的网络传输，减轻了网络负担；

· 容错性好，当被 Client 选中的 Server 因为各种原因不能服务时，Client 可以在 Cache 中重新选择 P 最小者为其服务；

· Web Servers 的 Cluster 结构利用了局域网快速的网络传输速度，保证了 Server Manager 的工作效率。

结论　本文提出了一种高效的 Web Servers 组织方式，将提供相同服务的 Server 组成 Cluster，在负载平衡决策时利用了智能 Client 的灵活性；同时用负载预测克服了信息的滞后性；采用 Server 控制 Client 端 Cache 的更新速度又极大减少了不必要的网络传输，减轻了网络负担。本文较好地解决了服务器过载的负载动态平衡问题，减少了服务的响应时间。但是，要全面提高 WWW 的服务质量，进一步减少服务响应时间，还需提高网络的传输带宽，彻底改变网络的拥塞，这可以结合硬件，做进一步研究。

参考文献

1　Available at·http·//www. ncsa. uiuc. edu.

2　Available at· http: //www netscape. com.

3　Karaul M, Korilis, Y A, Orda A. A market-based architecture for management of geographically dispersed, replicated Web Servers. Decision Support Systems, 2000, 28: 191～204

4　Andresen D, et al. SWEB: toward a scalable World Wide Web Server on multicomputers. In: Proc. of the 10th Intl. Parallel Processing Symposium (IPPS' 96). IEEE Computer Society Press

5　Berners-Lee T, Fielding R, Nielsen H. RFC1945: Hypertext Transfer Protocol-HTTP/1.0, May 1996

6　Brisco T RFC 1794·DNS Support for Load Balancing, April 1995

7　IBM, Interactive Network Dispatcher User, s Guide, 1997. Available at. HTTP: //www. ics. raleigh. ibm. com/netdispatch/nd2mst. HTM.

8　Katz E D, Butler M, McGrath R. A scalable HTTP Server·the NCSA prototype. Computer Networks and ISDN Systems, 1994, 27(2): 155～164

9　Dias D, et al A scalable and highly available server. Digest of Papers. In: COMCON' 96. Technologies for the Information Superhighway, Santa Clara, CA, February, IEEE Computer Society Press, 1996. 68～74

10　Yoshikawa C, et al. Using smart Clients to build scalable services. In. Proc. of the 1997 USENIX Annual Technical Conference, Anaheim, CA, January, USENIX, 1997. 105～117

11　Claffy K C, et al. Tracking long-term growth of the NSFNET. Communications of the ACM, 1994, 37(8): 34～45

12　Korilis Y A, et al Incentive-compatible network pricing. In: Proc. of the IEEE INFOCOMM. March-April, 1998