負載共享的異質性 Web Cluster 之設計與實作

李昀宸[+] 姜美玲[+] 蔡錫鈞[±]
u7213037@ncnu.edu.tw joanna@ncnu.edu.tw sctsai@csie.nctu.edu.tw

國立暨南國際大學資訊管理學系[+]
545 南投縣埔里鎮大學路 1 號

國立交通大學資訊工程學系[±]
300 新竹市大學路 1001 號

摘要

基本的 Web 叢集系統包括了前端的平衡負載主機與後端的真實服務主機群，但是由於電子商務及其他的需求，Web server 處理的不再是簡單的靜態網頁呈現的 requests，而後端的真實服務主機群也可能存在著硬體上的差異性，導致處理不同型態的 requests 能力有所不同。因此，Web 叢集系統也應該將來自 client 端的 requests 分類處理，以達成整個叢集系統有效的負載分享，使叢集系統的資源能更有效地被運用。本篇論文即是以實作一個異質性 Web 叢集系統為目的，來探討更有效的叢集系統架構，與其相對應的排程演算法。

關鍵詞：叢集系統、負載平衡、Linux、Web

一. 簡介

現今的網路時代，大部分的中小企業為了提昇競爭能力，莫不都以網路化以及電腦化來改善企業作業流程及效率。但是，機器設備的使用似乎往往跟不上產品更新的腳步，在一個企業所使用的 server 的折舊年限或許要四年，或者以上，但花大錢的結果卻不能夠得到有效的利用，例如企業擴張後，Web server 無法應付透過網路來接觸該企業的使用者所發出的所有的要求，增添設備必是一定的方法，但舊的設備並未折舊完畢，如此一來，舊設備的淘汰就形成了一種浪費；然而，叢集系統可以聯合數個服務主機來使 Web 的服務量增大，並且可以透過加入新的服務主機的方式，而不需淘汰舊有設備，就可增加 Web 服務量，改善組織的資源運用。而如何運用叢集系統來幫助企業或機關團體，來達到有效的使用機器設備，而甚麼樣的系統架構適合異質性的叢集系統，怎樣的排程演算法能夠有效的將服務量分散到後方的異質性的服務主機群就是本篇論文研究的重點。

而叢集系統的概念其實早已被 IBM 提出，但因為近幾年 Linux 的風行，才有在 Linux 上實作叢集系統；叢集系統為一個鬆散式分散式系統，因此可擴充性（Scalability）就是一個必要的特色；目前，在 RedHat 中有提供了 Linux Virtual Server（LVS）[10,13,14]，來快速及有效的建構一個叢集系統，RedHat 的 LVS 亦強調其高度可用性（High Availability）[11,13,20]，也因此在 LVS 可以有效的建立備援機制，又因為 Linux 有這麼多的好處，並且為一套開放原始碼的作業系統，所以適合中小型企業與學校機關使用，也因此本篇論文也利用 Redhat 的 LVS 來實作，研究異質性 PC 叢

集系統在 Web 上的應用以及相關問題的探討。

以建構在 RedHat Linux 上的 PC 叢集式系統為代表的 LVS Project[10,13,14]為例，它是一個三層式架構的系統，是由平衡負載主機、實際服務主機群、以及備援系統所組成，提供四種不同的靜態的排程演算法，其實作方式有三種，含 LVS/NAT、LVS/TUN、以及 LVS/DR，三者皆實作在 IP 層以上，就可擴充性及高可用性亦有其方案。然而，目前 LVS 只支援靜態的平衡負載策略，且只提供 RRS（Round-Robin Scheduling）、WRRS（Weighted Round-Robin Scheduling）、LCS（Least-Connection Scheduling）、以及 WLCS（Weighted Least-Connection Scheduling）等四種平衡負載排程演算法，無法滿足當實際服務主機群的硬體設備功能有差異時，更有效的平衡負載與負載分享。

本篇論文基於負載分享的觀念，考量在叢集系統中，可能存在的伺服器硬體的差異，我們設計且實作了一個異質性的 Web cluster，並提出一 Client-aware weighted dispatching algorithm，此排程演算法針對於不同類型的 client 的 requests，考量伺服主機群的功能，做有效的負載分享。

二. 相關研究

以 Linux 為基礎的 PC 叢集式系統成為現今熱門探討的主題，在國內外有許多計畫進行相關的研究，如國內即有中研院的 PC Farm 計畫[1]、國科會高速電腦中心的 PC Cluster 計畫[17]、行政院國家太空計畫室的 PC Cluster 計畫[18]、、、等等。國外亦有許多的研究；對於叢集式系統的應用，有應用在平行式計算，希望藉以低成本的個人電腦叢集，達到如大型電腦的高速運算功能，如 Beowulf[3,4]；亦有運用在 Internet 的研究，如 Scalable Web Server、Email Server、Storage Server、DNS Server、Media Server、Proxy Server、、、等等。亦有計畫重點放在建構具有可擴性與高可用性的叢集系統，如 Linux Virtual Server Project[13]、Linux High Availability Project[11]、IBM TCP router[6]、Cisco LocalDirector[5]、、、等等。

在平衡負載演算法有 RRS（Round-Robin Scheduling）、WRRS（Weighted Round-Robin Scheduling）、LCS（Least-Connection Scheduling）、WLCS（Weighted Least-Connection Scheduling）、優先權(Priority)方式、最快連線優先（Fastest Connections）方式、觀察(Observed)方式、以 URL 為基礎（URL-based）方式[2]、Global Server Load Balancing (GSLB)[8]、Content-based Load Distribution[16]、Connection Scheduling[15]、Location-Aware Request Distribution[9]、預估(Predictive)方式、、、等等演算法。

在平衡負載之研究，其中許多卻是採用靜態的策略，即事先設定各伺服主機群的加權植來分配工作量，因此，靜態沒有彈性的策略雖然簡單，但並無法真正達到平衡負載的目標；而動態的策略因需要經常性的收集各伺服主機群的工作負載，因此執行成本較高。

三. 系統設計

3.1 設計簡介

本系統的設計理念是源自於負載分享（Load sharing）與在叢集系統中可能存在的伺服主機群硬體設備的差異，以及對於不同類型的 requests，後端真實服務主機應如何設置來達到有效的負載分享。

在叢集系統中，有可能某些後端真實服務主機的 CPU 處理能力較其他的後端真實服務主機來的強大、快速，但硬碟的讀取速度或傳輸可能不及其他的後端真實服務主機，在這種情形下，我們應該考慮到來自 client 的 requests 是屬於甚麼性質，再對後端真實服務主機群的硬體差異性來做排程演算法的設計，使 requests 能夠更有效的被繞送，且叢集系統變得更有彈性，更善用了不同後端真實服務主機群硬體上的差異，使後端真實服務主機群的資源能更有效的被運用。

就封包類型來分析，一個 Web server 要處理的 requests 類型大致上可以分成一般（normal type）；需要 disk 能力較強的（disk bound），如搜尋引擎；需要更多 CPU 資源的（CPU bound），如 SSL 安全連線。透過對 requests 的分類，再運用對真實服務主機群不同硬體設備設定不同權重的方法，來達成系統建置的目的。

因此，平衡負載主機必須具備分類不同類型 client 的 requests 的能力，並且能依後端的伺服主機群處理各類型的 requests 的能力的不同，設定多份的排程演算法。

3.2 排程演算法

當平衡負載主機收到來自 client 端的 requests 時，平衡負載主機必須將 requests 傳導到後端的後端真實服務主機處理；當然在一個叢集系統裡，後端真實服務主機應有一台以上的電腦所組成；而後端真實服務主機應如何和平衡負載主機互動，來達到叢集系統的設置目標。

3.2.1 動態排程與靜態排程

動態排程與靜態排程各有優劣，動態排程是平衡負載主機依據某些資訊，將來自 client 的 requests 動態地分送到後端的後端真實服務主機群，動態排程又可以分成 client-aware 及 server-aware 的不同，當然也有混合的 client- and server-aware 的方式；以 server-aware 為例，server-aware 是根據後端真實服務主機的資訊來決定 requests 該往哪裡送，譬如：以後端真實服務主機的目前最少連線數為依據，來決定平衡負載主機現正收到的 requests 該由哪台後端真實服務主機處理；而 client-aware 是根據送出 requests 的 client 端的某些資訊，來決定 requests 該往哪個後端真實服務主機送，譬如：以 client 送出的 requests 判斷，檢視 requests 的內容，來決定由哪台後端真實服務主機處理；client- and server- aware 就是綜合了來自 client 的資訊及後端真實服務主機群的資訊，來決定 request 的處理，譬如：偵測後端真實服務主機與 client 間的網路傳輸速度，來決定由哪個後端真實服務主機處理，這對一個設置在廣域網路(WAN)的叢集系統來說是十分有幫助的。

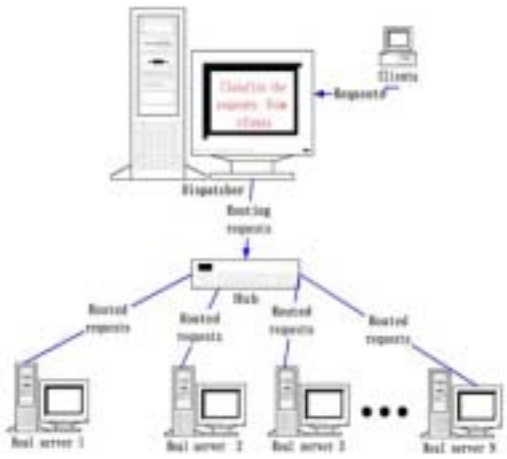
靜態排程是根據某些特定的 requests 分送規則，有序地將 requests 送出，其中最熟為人知的就是 Round-Robin；Round-Robin 是最基本，也是最簡單的的排程演算法，平衡負載主機將來自不同的 client 端的 requests 依序地平均分發給後端的真實服務主機群來處理，依此次序來循環運作，每一部真實服務主機被排到的次數相等，適用於每部真實服務主機的平台設備都一致時的情況。

3.2.2 平衡負載與負載共享

平衡負載的目標在於使叢集系統中的每部真實服務主機的負載能夠平衡，但要能夠達成這樣的目標必須以動態排程方式來分配，以

獲知各個真實服務主機的負載情形，或是使各個真實服務主機的硬體設備完全一樣；而負載分享的目標在於使叢集系統中的每部真實服務主機都能夠分擔來自 client 的 requests，也因此系統設置的彈性較大。

3.3 系統架構

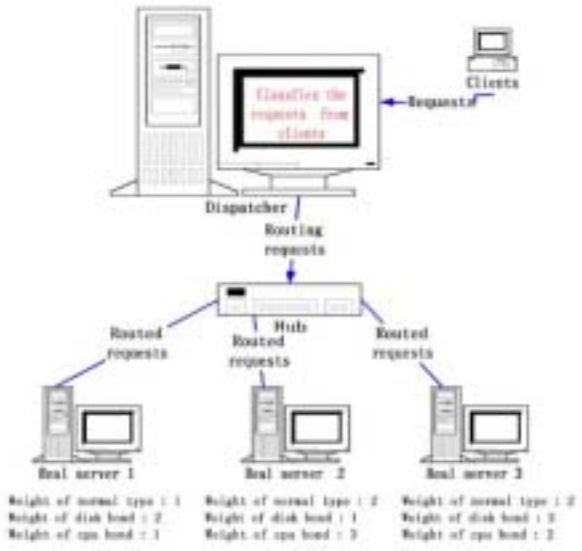


圖一、系統架構圖
顯示 Dispatcher 與後端真實服務主機群的運作與設置

如圖一，有一台位於前端的平衡負載主機，以及多台的後端真實服務主機，前端平衡負載主機為 Dispatcher，負責接收自外來 client 傳送的 requests，一如叢集系統中前端服務主機，而 Dispatcher 負責的工作就是把不同類型的 requests 分類，再根據排程演算法的設定，繞送到後端真實服務主機；後端真實服務主機將分送出來的 requests 接收處理之後，再將相對的 reply 傳回給 client 端。

3.3.1 Client-Aware Weighted Dispatching 演算法

而在這個 weighted dispatching web clusters 所使用的演算法是對 client 的 request 加以分類的方式，再加上 WRRS（Weighted Round-Robin Scheduling），這是一種改良自 RRS 輪循式排程法的一種排程演算法，先對每一部真實服務主機給予一個不同的加權值，而當平衡負載主機收到來自不同 client 的 requests 時，根據 requests 的型態，依照此叢集系統建置時所設定各真實服務主機處理此種 request 型態的加權值的大小，依序分發給真實服務主機群處理，而加權值的大小會影響排程的次數，平衡負載主機會將 requests 多排給加權值較大的真實服務主機，依此循環運作，而一次的循環內，每部真實服務主機的被排程次數會等於加權值，例如：真實服務主機 a、b、c 的加權值分別是 4、3、1 的話，則實際排程則為 abcababa，適用於當真實服務主機群設備不一致的情況下。



圖二、權重與架構範例圖

以圖二為例，當 requests 送進來時，經過 Dispatcher 分類，依照 WRRS 演算法的順序，依序將 requests 繞送到後端真實服務主機，然後再由真實服務主機來做處理，回傳給 client。而權重的設定交由叢集系統建構者或管理者依硬體的差異情形來設定不同的權重；因此，若硬體設備一有不同，真實服務主

機服務某類型的 requests 的權重就有所不同。

四. 系統實作

4.1 實驗平台

實驗的目的在於顯示出當後端真實服務主機群在硬體上有差異時，我們的 Client-aware Weighted Dispatching 演算法是否能將後端的真實服務主機群的各類資源做到有效的負載分享，因此，在實驗中有四台不同的後端真實服務主機及一台前端平衡負載主機，分別為 Cluster1、Cluster2、Cluster3、Cluster4。而前端平衡負載主機要接收所有的到此叢集系統的 requests，而且要傳回所有真實服務主機的回應，因此需要快一點的機器。實驗的硬體設備詳述如表一。

表一.實驗主機主要硬體設備

Cluster1	CPU：Intel Pentium III 500MHz Memory：128MB Disk：18.9GB，SCSI，10000rpm SCSI card：80MB/s
Cluster2	CPU：Intel Pentium III 500MHz Memory：128MB Disk：20GB，IDE，5400rpm
Cluster3	CPU：Intel 233MHz MMX Memory：64MB Disk：10GB，IDE，5400rpm
Cluster4	CPU：Intel Pentium 200MHz Memory：32MB Disk：20GB，IDE，5400rpm
Front-End Dispatcher	CPU：Intel Pentium III 800MHz Memory：256MB Disk：20GB，IDE，5400rpm

在軟體方面，每台主機的作業系統均為 Linux（RedHat 6.2）及 LVS；而後端真實服務主機群的網頁伺服器軟體為 Apache，資料庫軟體為 MySQL，動態網頁由 PHP 呈現。

4.2 Weighted Dispatching Web Clusters

基本上，一個 Web server 所提供的網頁內容可能是一般網頁，或是與資料庫結合的動態網頁，或是 SSL 的安全連線；因此，一個來自 Client 端的 request 要耗掉 Web server 上的資源份量，就會因為 request 的性質而有所不同。

而我們可將封包分為三大類：一般的 requests（Normal Type）、需耗較多磁碟讀取時間的 requests（Disk Type）、需耗較多 CPU 處理時間的 requests（CPU Type）。在此我們只針對一般的 Web service 來處理，也就是網頁的呈現來設計，暫不對一些特殊的多媒體的 service，如 Windows media service 來討論。

一般的來自 client 端的 http requests 都有個 MIME Header，記載著要向 server 讀取的檔案、方法、http 版本、以及其他資訊，以 IE5.5 為例，連到任一站台 request 的 MIME Header 應為以下格式：

```
GET / HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg,
image/pjpeg, application/vnd.ms-powerpoint,
application/vnd.ms-excel, application/msword,
*/*
Accept-Language: zh-tw
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 5.5;
```

Windows 98; Win 9x 4.90)
Host: localhost
Connection: Keep-Alive

以上為連結到 localhost 的 MIME Header 範例，而我們在實作時，檢驗來自 client 端的封包就是靠分析 MIME Header 的訊息。

4.2.1 Client 的 requests 分類方法

Dispatcher 透過參數檢驗、檔案名稱辨識、以及聽取 SSL port 等三種方式來將 client 的 requests 做分類。詳述如下:

Http 用來處理被要求資源的方法有：”OPTIONS”、”GET”、”HEAD”、”POST”、”PUT”、”DELETE”、”TRACE”、”CONNECT”[19]等八種，最常使用的就是”GET”與”POST”；以”GET”的方式來看，在傳輸一個包含有資料庫的動態網頁(如 ASP、PHP)，URL 必須要加上參數，才能將要求的資料從資料庫中顯示出來，而這樣的參數必須使用一些符號，如：%、&，這些符號也會顯示在 MIME Header 的第一行的第二欄裡，因此，我們只需要針對該欄位來作字元檢查，就可以知道是否為 disk type 型的 requests。

但若是該 requests 是用”POST”的方式呢？因為”POST”這個方法或是已有一些 html 網頁裡有 ASP 程式，就直接讀取資料庫裡的資料，這時，我們就對該檔案名稱做辨識，只要是該檔案的 requests 就算是 disk type 的類型，其餘則是 normal type。

那麼 CPU Type 的 requests 在一般的 Web service 就是 SSL 了，因為 SSL 安全連線是一種加密過的資料，因此，我們不能對 SSL 安全連線的 requests 內容直接來做分析；然而，因為 SSL 安全連線有另一個 port 專門接收(預設值是 port 443)，因此就只針對 Web server 聽 SSL 安全連線的 port 將 requests 傳導到後方真實主機，分散負載。

五. 系統效能評估

在效能評估方面，我們將評量 3 個系統的做法：RedHat 的 LVS[13,14], Emiliano Casalicchio 和 Michele Colajanni 的 Web cluster [7], 以及我們的 Weighted Dispatching Web cluster。

RedHat 的 LVS 支援了靜態的 Weighted Round-Robin 排程演算法，可根據後端的真實服務主機群硬體上的差異來設定不同的權值，但並不去分析及考慮 client 的 requests 型態，會有不夠有效地做到負載分享的缺點；而 Emiliano Casalicchio 和 Michele Colajanni 的 Web cluster 提出了一個 Client-Aware Dispatching Algorithm，它會去分析及考慮 client 的 requests 型態，使用的是 Round-Robin 的排程演算法，但它並未考慮到後端的真實服務主機群是異質性的情形；而我們的 Weighted Dispatching Web cluster 採用的是 Client-Aware Weighted Dispatching Algorithm，同樣的它會去分析及考慮 client 的 requests 型態，但針對每種不同的 requests，考慮到後端的真實服務主機群是異質性的情形，使用不同的 Weighted Round-Robin 的排程演算法，來分配 requests 到後端的真實服務主機。

六. 結論與未來工作

我們已實作一個異質性的 Weighted Dispatching Web Cluster，並提出一個 Client-aware 加權排程演算法，目的是希望能有效地達成異質性 Web cluster 的負載共享，

目前我們除了將做嚴謹的系統效能評估之外，未來尚有許多的研究課題，分述如下：

（1）真實服務主機回應 requests 的傳輸方式

目前我們採用的是 LVS/NAT 的叢集架構，在 NAT 的架構上，前端的平衡負載主機要處理所有來自虛擬網路與真實網路之間的網路傳輸，而限制了整個 Weighted Dispatching Web Cluster 擴充效能，未來將研究該如何使用 IP Tunneling[13]或是 Direct Routing[13]的架構來實作，以避免整個系統效能被限制的問題。

（2）研究異質性的 service 的可能性

基本上，Weighted Dispatching Web Clusters 是針對同質性的 Web service 來實驗的，而是否異質性的 service 也能夠透過 requests 的分類，使 Weighted Dispatching Web Clusters 能提供異質性的服務。

（3）研究在廣域網路實作的情形

在廣域網路實作的話，我們更可以對來自 client 的 IP 位址或是真實服務主機群與 clients 之間的網路連線狀況來分析，讓 Dispatcher 能選擇連線速度較快的真實服務主機對 client 的 request 作處理及回應。

而至於記憶體的快取命中率（cache hit ratio）是否會影響系統的反應時間或是提供更多種分類的 requests 型態亦是未來研究的重點。

誌謝

本研究由國科會大專學生參與專題研究計畫編號 NSC90-2815-C-260-005-E-及國科會專題研究計畫編號 NSC90-2213-E-260-022-所支持。

七.參考文獻

[1] Academia Sinica Computing Center, Taiwan, R.O.C., PC Farm Project, http://www.pcf.sinica.edu.tw/.

[2] A. Cohen, S. Rangarajan, and H. Slye, “On the Performance of TCP Splicing for URL-Aware Redirection”, Proceedings of the 2nd USENIX Symposium on Internet Technologies and Systems, Boulder, CO, Oct. 1999.

[3] Beowulf Clustering system, http://cesdis.gsfc.nasa.gov/linux/beowulf/beowulf.html.

[4] Beowulf HOWTO, http://www.linux.org.tw/CLDP/Beowulf-HOWTO.html.

[5] Cisco’s LocalDirector, http://www.cisco.com.

[6] D.M.Dias, W.Kish, R.Mukherjee, and R.Tewari, ”A scalable and highly available Web server”, Proc. of 41st IEEE Comp. Society Int. Conf., Feb 1996.

[7] Emiliano Casalicchio and Michele Colajanni, “A Client-Aware Dispatching algorithm for Web Clusters Providing Multiple Services”, Proceedings of the Tenth International World Wide Web Conference, Hong Kong, May 2001.

[8] Foundry Networks, ServerIron, http://www.foundrynet.com .

[9] John Heidemann, and Dhaval Shah, “Location-Aware Scheduling with Minimal Infrastructure”, USENIX Annual 2000 Technical Conference, June 18-23, 2000, San Diego, California, USA.

[10] Linux Clustering HOWTO, http://yara.ecn.purdue.edu/~pplinux/ppHOWTO/.

[11] Linux-HA Project Web Site, http://www.linux-ha.org/.

[12] Linux Parallel Processing HOWTO, http://yara.ecn.purdue.edu/~pplinux/PPHOWTO/pphowto.html.

[13] Linux Virtual Server Project, http://www.linuxvirtualserver.org.

[14] LVS Cluster Configuration HOWTO, http://www.redhat.com/support/docs/howto/piranha/index.html.

[15] Mark E. Crovella, Robert Frangioso, and Mor Harchol-Balter, “Connection Scheduling in Web Servers”, 2nd USENIX Symposium on Internet Technologies and Systems, October 11-14, 1999, Boulder, Colorado, USA.

[16] Mohit Aron, Darren Sanders, Peter Druschel, and Willy Zwaenepoel, “Scalable Content-aware Request Distribution in Cluster-based Network Servers”, USENIX Annual 2000 Technical Conference, June 18-23, 2000, San Diego, California, USA.

[17] NCHU PC Cluster Project, Taiwan, R.O.C., http://www.nchc.gov.tw/RESEARCH/pccluster.

[18] NSPO PC Cluster Project, Taiwan, R.O.C., http://www.nspo.gov.tw.

[19] RFC2616, Hypertext Transfer Protocol – HTTP/1.1, http://www.w3.org/Protocols/.

[20] The Official RedHat High Availability Server Installation Guide, http://www.redhat.com/support/manuals/RHHAS-1.0-Manual/.