叢集式 Web 伺服器分配策略之效能量測

羅孟彥* 曾金山　楊竹星
國立中山大學資訊工程學系　正修科技大學資訊管理系*
{jstseng, csyang}@cse.nsysu.edu.tw,　myluo@csu.edu.tw*

摘要

隨著網際網路快速普及，許多服務逐漸由傳統型態轉變為以網頁服務作為媒介，單一網頁伺服器架構已無法滿足目前網路龐大的使用者需求，叢集式網頁伺服器架構儼然成為另一個合適的解決方案。分配機制在叢集式 Web 伺服器中扮演一個非常重要的角色，因此過去幾年間有許多負載平衡分配策略被發展出來，然而這些研究都只有以模擬系統來加以驗證，因此這些策略在一個真實的系統中運作的效能為何仍屬未知。本文是第一個以實際網站量測並分析比較已被提出之分配策略的研究。

關鍵詞：Web 叢集式伺服器、分配策略、效能量測。

1. 前言

由於網際網路的使用人口快速成長，許多熱門的網路服務如雅虎(Yahoo)、eBay、亞馬遜(Amazon)書店、Google 等，必須隨時應付大量的使用者同時連線進入，因此 Web 伺服器的效能受到嚴峻的挑戰，對於解決 Web 伺服器過度負載的方法除了以更高階的伺服器取而代之外，另一被廣為採用的方法便是叢集式(Cluster-Based)伺服器架構。

在叢集式 Web 伺服器系統架構中，影響整體效能的重要因素之一在於對客戶端服務請求(HTTP Request)的分配，如何有效的將服務請求導向適當的伺服器進行處理，讓伺服器間的負載能均衡，以達到讓使用者能更快速獲得所需資料、減少其等待時間，已成為近年來學術界與工業界熱門的研究議題，目前已有許多負載平衡分配策略被發展出來，然而這些研究普遍存在著兩個問題，首先，這些研究都只有以模擬系統來加以驗證，因此這些策略在一個真實的系統中運作的效能為何仍屬未知。

其次，在這些模擬系統中，過去的研究都假設 Web 的流量遵循一定的統計分布（一般稱之為 Heavy-tailed Distribution[5]），然而經由過去幾年我們實際維運一個 Web cluster 的經驗來看，我們發現這樣的假設已有所改變。就目前的網頁內容來看，Web 檔案的平均大小有變大的趨勢，主要的原因是這幾年網路技術快速進步，頻寬不斷的被增加，使得許多豐富的多媒體內容變為可行，藉由網路作為媒介，一些資訊紛紛上網，如：線上學習系統中的大量的教學影音檔或教材、軟體公司的修補檔(patch)或試用版(trial)軟體、遊戲公司的試玩版遊戲等。我們稱這樣的 Web 流量為 data-intensive 的 workload。在流量特性逐漸改變的情況下，過去所發展的這些分配策略是否仍適用，便成為一個有待研究的問題。

基於上述的問題，本文主要著重於了解以往所被提出的各種分配服務請求策略，在真實環境中運作的情形，我們在一個實際運作中的叢集式 Web 伺服器系統上實做了各種分配機制，並進而設計實驗方法以實際的 data-intensive Web 流量來比較與分析這幾個分配策略的優劣，就我們所知，本文是第一個在實際系統上實作並比較分析各種分配機制的研究。我們的研究結果證實了某些過去研究所假設的觀點，也澄清了一些謬誤。

2. 叢集式 Web 伺服器

現今典型的 Web 伺服器叢集系統，架構上可分為前端的分配器(Dispatcher)或稱為 Web 交換器(Web switch)和後端的伺服器群(Server Farm)，如圖 1所示，前端伺服器對外代表整個伺服器系統，負責將客戶端服務請求導向後端伺服器，而後端伺服器則實際提供客戶端所需的服務。此類架構雖然實際由多部伺服器協同處理使用者服務請求，然而對外界使用者而言仍如同面對單一系統般，前端 Dispatcher 會將大量的網頁服務請求(Web Request)分散交由合適的伺服器去處理，普遍的做法是以分配器的網路位址(IP Address)當作整個 Web 伺服器系統之位址，因此所有的客戶端只知道分配器的網路位址，而不會知道有後端伺服器群的存在，整個系統對客戶端是透明的(Transparent)，客戶端在使用系統上就如使用一台高效能網頁伺服器。

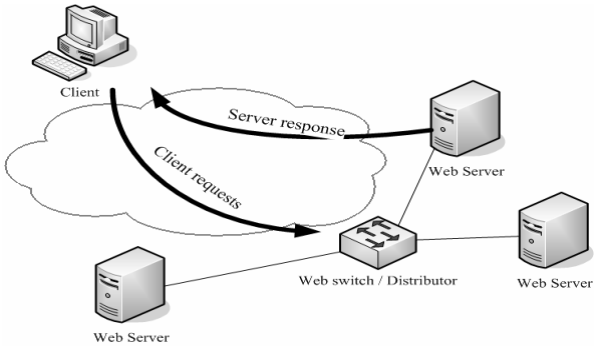

圖 1 Web Server Cluster

叢集式伺服器的架構能同時兼顧高度可擴充性(Scalability)與高度可及性(Availability)，後端伺服器可因需求不斷擴增，前端伺服器可藉由一些偵測機制了解後端伺服器狀況，避免將服務請求導向有問題的節點，因而目前網路上許多中大型網站均採用這樣的架構。

3. 排程策略

在叢集式伺服器架構中，必須有一個機制將客戶端送出的服務請求導到後端伺服器群中適當的伺服器節點，因此服務請求的分配策略便成為決定整個叢集式伺服器系統效能的重要因素之一。

目前所提出來的分配策略大致上可分為 content-blind 與 content-aware 兩大類，content-blind 的策略不考慮客戶端所要求的服務型態，對於每個服務請求皆一視同仁。content-aware 的策略則會依據客戶端所要求的內容性質來作為分配依據，前者的優點是實作簡單，後者有幾項優點：1.可利用服務請求間的 locality 特性，增加記憶體快取擊中率(memory cache hit rate)，進而提高系統整體效能、2.可將網站檔案分散放置在不同的伺服器上(即所謂的 content partition)，伺服器不需採用完全複製(full-replication)，可以增加磁碟空間的可擴充性、3.可針對客戶端服務請求特性特別處置，將較需要處理器效能的請求，如動態網頁(dynamic content)或較需要磁碟存取的請求，如影音檔，各自送往適當的伺服器作處理。然而此方法的缺點在於可能增加系統額外負擔(overhead)。在本章節中我們將介紹目前常用的幾種排程演算法。

3.1 Content-blind approach

常用的 Content-blind 分配策略主要有以下幾種:

(1)隨機(Random):對於客戶端服務請求，隨機選取一台後端伺服器節點處理。

(2)輪巡(Round-Robin)[3]:其運作行為如下，假設有三部伺服器節點(Sever A、B、C)，第一個服務請求往伺服器 A，第二個服務請求往伺服器 B，第三個服務請求往伺服器 C，第四個服務請求再送往伺服器 A，以此類推，不考慮伺服器端與客戶端之資訊，此方法假設所有伺服器節點處理能力均相同，依次將請求分配至不同的伺服器，演算法實做簡單，但不適用於伺服器群中處理性能不一致的情況。

(3)權重式輪巡(Weighted Round-Robin):為解決輪巡演算法無法適用在不同處理能力的伺服器群上，此方法採用加權值表示伺服器的處理能力，將請求數目按加權值的比例分配到各伺服器。加權值高的伺服器優先選擇，加權值高的伺服器比加權值低的伺服器處理更多的服務請求。

(4)最小連線數(Least-Connection):此方法藉由伺服器連線數來評估伺服器負載，記錄各個伺服器已建立 TCP 連線數目，將新的連線請求轉送當前連線數最小的伺服器，但此方法亦會面臨到輪巡演算法所遇到的問題。

(5) 權重式最小連線數(Weighted Least-Connection):當各個伺服器的處理能力不同時，該最小連線數演算法並不理想。權重式最小連線數分配是最小連線數分配的超集合(superset)，各個伺服器用相應的權值表示其處理能力，假設有 n 部伺服器節點，每部伺服器 i 的權值為 Wi (i=1,..,n)，TCP 連接數目為 Ci (i=1,..,n)，將新的連線請求轉送至 Ci/Wi 值最小之伺服器節點。

3.2 Content-aware approach

目前被提出的 content-aware 分配機制中最早的是 Locality-Aware Request Distribution (LARD)[8]，此方法基本精神在於利用 locality 特性，提高主記憶體的 cache hit rate，以獲得更高生產量(high throughput)。[8]採用客戶端連線數做為伺服器負載指標(load index)的依據，運作方式是由分配器先檢查是否有伺服器群組服務過此檔案內容，如果有則將服務請求送往該群組負載最低的節點，反之對於新的服務請求則從所有伺服器中挑選出負載最低的節點，為處理可能有檔案過於熱門導致伺服器過度負載的問題，該方法會檢查是否有節點超過設定的臨界值(threshold)的情形，一但發生超過負載狀況，則增加負載最低的伺服器節點共同服務或移除超過負載的節點。此方法最差情況發生於所有節點皆超過負載，則其運作行為如輪巡(Round-Robin)排程演算法。

EquiLoad[4]其核心概念在於將工作時間長短不同的 job 加以區隔，以避免工作時間短 job 等候在時間長的 job 之後，以達到最短的回應時間(minimize response time)。此方法利用檔案大小來做為衡量 job 工作時間長短的依據，藉由分析過去的歷史紀錄來計算各檔案被存取的頻率，藉此切割整體的工作量，使得各個伺服器節點間的工作量均衡，以達到負載平衡的目的，此方法的優點在於運作簡單且兼顧 locality 與負載平衡，一但區間邊界點(boundary)決定下來，服務請求便依照檔案大小決定送往特定伺服器節點以維持 locality 特性。

AdaptLoad[2][7]則針對 EQUILOAD 所可能面臨的問題提出相關修正，主要是提出模糊邊界(fuzzy boundary)的概念，此外，該方法會不斷的監測客戶端服務請求狀況，當請求數到達設定臨界值則重新計算邊界點，動態調整邊界點以符合未來所需。此方法與 EquiLoad 皆歸類為 Size-Based 排程策略，藉由了解檔案大小來賦予伺服器節點合適的工作量。

4 系統設計與實作

本章節將介紹我們如何設計與實作一個叢集式伺服器系統，也將介紹我們為量測各種排程策略的效能所設計之方法與系統相關模組之實作。

4.1 系統架構

本文所採用之系統架構如圖 2所示，除了一部負責此網域的網域名稱系統伺服器外尚有十部後端 Web 伺服器、一部分配器(Dispatcher)、兩部量測系統效能的模擬客戶端、以及一部負責記錄量測結果的資料庫伺服器。每部伺服器以 100Mbps 的網路介面接至一台 L2 交換器，交換器再用光纖以 1Gbps 速度連接至中山大學校園路由器，所採用的硬體設備為：CPU 為 Intel Petium4 1600MHz、主機板採用華碩 ASUS P4B266、記憶體為 512MB、網路卡採用 Accton 10/100 Mbps、硬碟使用 IDE 系列，分配器為 Intel 1.5GHz、記憶體 128MB、10/100 Mbps 網卡；軟體部份：作業系統為 Gentoo Linux Kernel 2.6.12、Web 伺服器版本，apache-2、檔案系統，ReiserFS、相關模組尚有 php。

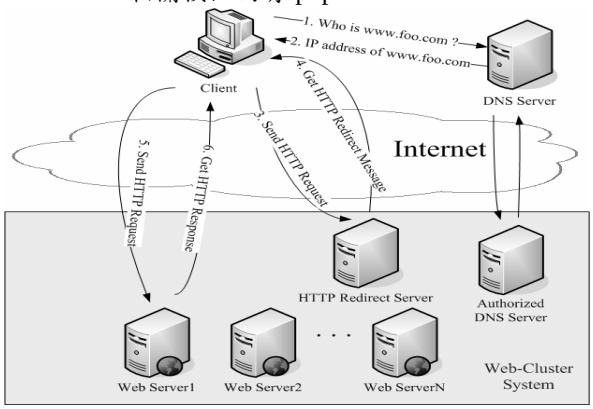

圖 2 System Architecture

在這樣的系統中，我們目前負責維運許多網路服務，使用者欲取得其內容的運作流程如下：
Step1、2.客戶端透過 DNS 取得網站之網路位址(IP address)，此網路位址即分配器之網路位址。
Step3.客戶端將 HTTP 服務請求送往分配器。
Step4.當分配器收到來自客戶端之 HTTP 服務請求，根據分配策略決定出適當之伺服器節點，並送 HTTP 狀態碼(status code) 302 “Found”之重導(redirect)訊息至客戶端，告知實際服務主機位址。
Step5.客戶端收到此訊息後，立即向被告知之伺服器節點建立連線，並要求服務。
Step6.伺服器節點根據本身狀況決定是否接受此要求，並做出回應。

4.2 Workload

為測試各種分配策略的真正效能，本研究以我們所維運的網路服務所產生的真實流量來作為測試依據，因此我們先分析與比較這個測試網站的流量特性。圖 3為這個網站的流量累積分佈(cumulative distribution function)圖，X 軸表示 request 檔案大小(取 log10)，Y 軸表示累積機率。圖中還與中山大學網站(http://www.nsysu.edu.tw)的流量來作為比較，由此我們可以發現，測試網站與學術性網站相比有相當程度的不同，主要是測試網站含有許多大型檔案。

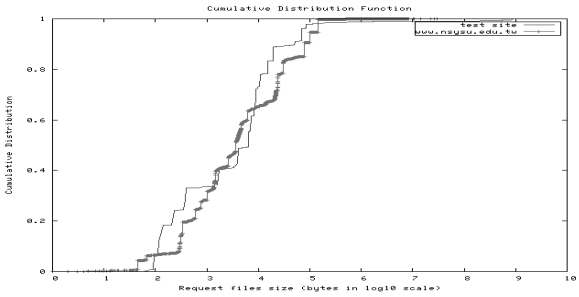

圖 3 Workload

我們再拿 1998 世界足球盃網站(World Soccer Cup Web site)的流量特性進行比較，由表格 1及表格 2我們了解到測試網站檔案最大及平均大小與比較網站差異高達百倍之多。我們稱這種流量特性為 data-intensive Web 流量。

表格 1 Test site Statistical Information Regarding the Unique Files Requested

Number of unique files	Mean (MB)	Median (bytes)	Maximum (GB)
532	85.96	106,779	2.027

表格 2 World Soccer Cup Web site Statistical Information Regarding the Unique Files Requested

Number of unique files	Mean (bytes)	Median (bytes)	Maximum (MB)
17,332	11,786	3,714	3.1

4.3 量測方法設計

我們所設計的量測系統由幾個主要的元件所完成，如圖 4所示，主要模組分成三部分：1.Client performance measure Daemon 、 2.Server node performance monitor Daemon 、3.HTTP Redirect Daemon。

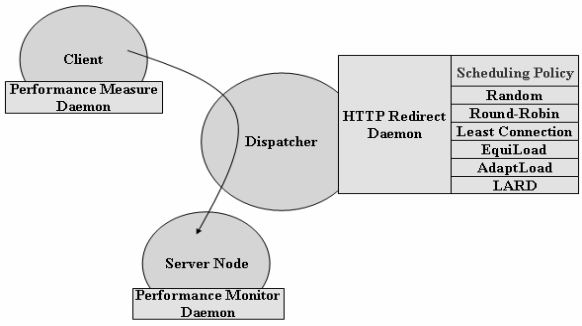

圖 4 System component

Client performance measure Daemon，此模組負責模擬客戶端，藉此測量客戶端取得網頁物件所需花費時間(Round-Trip Time)，來做為評估各種排程策略效能表現的指標。為避免網路壅塞可能造成的量測誤差，我們將量測設備與整個系統連接在同一交換器上，量測方法上，此模組每兩分鐘依序取得50B、100B、200B、400B、800B、1.6KB、3.2KB、6.4KB、12KB、24KB、48KB、96KB、192KB、500KB、1MB 等，大小不同的 15 個檔案當作客戶端取得整個網頁及內嵌物件，並紀錄取得檔案所花費時間 $Time_{total} = Time_{redirect} + Time_{socket} + Time_{response} + Time_{data}$。

$Time_{total}$：客戶端從 Request 送出至檔案全部接收完畢所花費時間。

$Time_{redirect}$：客戶端 Request 發送至分配器，並收到分配器 HTTP Redirection 資訊所花費時間。

$Time_{socket}$：客戶端重新向後端伺服器建立連線，完成 TCP 3-way handshake 所花費時間。

$Time_{response}$：客戶端 TCP 3-way handshake 完成後發送 Request 至收到伺服器端回送資料的封包所花費時間。

$Time_{data}$：客戶端收到資料封包開始至整個資料傳送完畢所花費時間。

此模組源自於一套 OpenSource [1]軟體 http_ping，我們對此軟體作了一小部分的修改使其增加對 HTTP Redirect 支援，以符合我們需求，對於收集得來的資料我們將結果存放進資料庫中以便之後統計。

Server node performance monitor Daemon，在每部伺服器節點安裝此模組，它負責收集紀錄伺服器負載狀況，並提供分配器(dispatcher)必要資訊，我們採用的負載指標(load index)為客戶端連線數(active connection)。此模組每分鐘紀錄系統狀態，我們從 Linux proc 檔取得包含：CPU 使用率、記憶體使用率、Swap 使用率、HTTP processes 數量、磁碟分割區(partition)讀寫次數、磁碟分割區讀寫磁區(sectors)數、網路卡進出 Bytes 數、封包數、loadavg 等資訊，計算前一次紀錄值與最新取得紀錄值的差值除以間隔時間做為此段時間內系統資源使用情況，並將下列結果存放進資料庫中:

(1)CPU 使用狀況：由/proc/stat 檔取得系統在 user mode、user mode with low priority(nice)、system mode 及空閒時段(idle)所花費的 jiffies($1/100^{th}$ 秒)數。

(2)主記憶體與 Swap 使用狀況：由/proc/meminfo 檔取得實體記憶體總量、Swap 空間總量、實體記憶體 Free 空間、Swap Free 空間，計算出記憶體使用率。

(3)硬碟讀寫次數與磁區數：由/proc/diskstats 檔取得系統對硬碟分割區(partition) 發出(issue)讀寫次數及多少磁區數(sectors)對硬碟分割區讀寫，經過換算(1 sector = 512 Bytes)了解硬碟整體輸出量；在 Linux kernel 2.4 中關於硬碟讀取狀況紀錄在/proc/partitions 及/proc/stat 檔。

(4)網路卡進出 Bytes 數、封包數：由/proc/net/dev 檔取得伺服器網路卡進出 Bytes 數、封包數。

(5)Loadavg：根據 Linux 文件說明，/proc/loadavg 檔紀錄系統每 1 分鐘、每 5 分鐘、每 15 分鐘平均在執行(run) queue(state R)或等待磁碟 I/O(state D)的 jobs 數，數值越高代表系統越繁忙。

我們實作一個 HTTP Redirect Daemon 來作為系統的分配機制，其負責將來自客戶端的服務請求導向至適當的後端伺服器，在服務請求排程策略上我們實作了第三章所述的方法。為比較各演算法間的效能表現，圖 5為我們所設計之實驗方法。

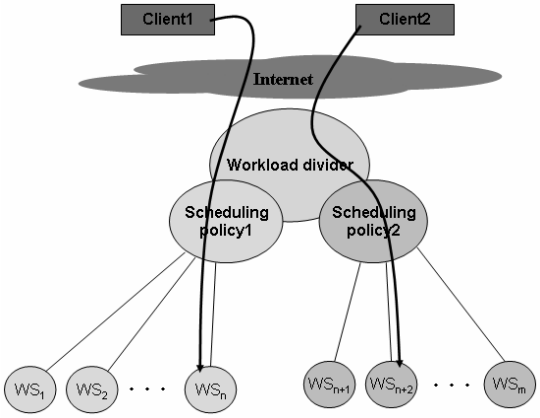

圖 5 Dispatch request

表格 3則為分配器 Pseudo-code，我們同時間一次比較兩種排程策略以避免不同時段因為 Workload 不同可能造成測量結果不公平，我們將十部伺服器節點分成兩組，每組各五部分別跑不同的排程演算法，為避免影響客戶端對 Web 主機的 locality 特性，我們將不同的客戶端以輪巡(Round-Robin)方式選取不同的排程演算法，對於相同的客戶端我們選取該網路位址(IP Address)前一次所選定的排程演算法。

表格 3 Dispatcher Pseudo-code

```
While (true) {
  Fetch next request r; // 新的 request
  Parse_Header(r); // 解析 HTTP request header
  If (dynamic content) {
    serverIP ← special purpose machine;
  } else {
    If (exist hostTable[r.ip]) {
      mechanism ← hostTable[r.ip].mechanism;
      serverIP ← GetServerNode(mechanism);
    } else { // 選擇排程演算法
      mechanism = RoundRobin(mechanismSet);
      hostTable[r.ip].mechanism ← mechanism;
      serverIP ← GetServerNode(mechanism);
    }
  }
  Pack_HTTP_Rsponse_Message(r.uri, serverIP);
  Send HTTP_Rsponse_Message to r.ip;
}
```

目前我們只考慮靜態網頁內容(Static Content)，對於動態網頁內容(Dynamic Content)部份如：PHP、ASP 等，不在本文考量範圍。選定好採用的排程策略後便透過該策略選取適合服務的伺服器節點，並封裝 HTTP 狀態碼(status code) 302 “Found” Redirect 訊息至客戶端，客戶端瀏覽器收到此訊息便重新對該伺服器節點進行服務要求。經過這樣的分配機制，我們統計兩組 Cluster 系統所接受的 Workload 情況，如圖 6所示，我們發現兩組 Workload 相當一致。

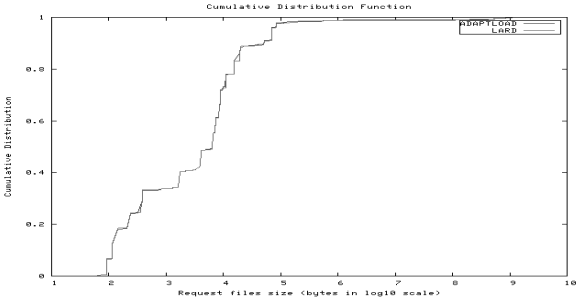

圖 6 Compare workload

5 結果分析

我們將說明幾種排程演算法在實際網站中比較的情況。每次量測皆為長時間累積的結果，且每次量測結果若 Workload 不同則不採用。

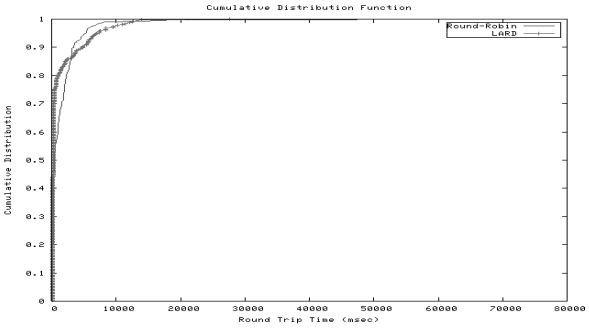

圖 7 Round-Robin v.s. LARD

由圖 7我們發現到 LARD 在真實且 Data-Intensive 網站中表現並非絕對比 Round-Robin 好，影響 RTT 因素我們分成兩種，第一種為檔案在記憶體中快取擊中率(cache hit rate)，另一種為系統本身的負載，進一步將測量時間分為兩個時段，一者系統負載較輕時段如圖 8所示，另一者系統負載較重時段如圖 9所示，我們所採用的負載指標為 httpd process 數。

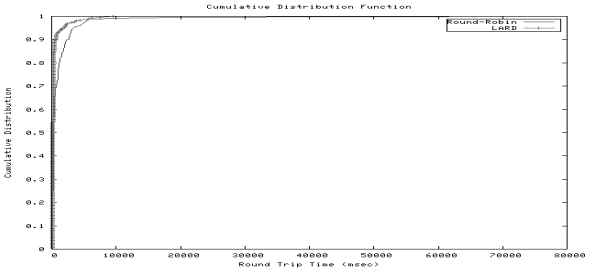

圖 8 Round-Robin v.s. LARD low loading

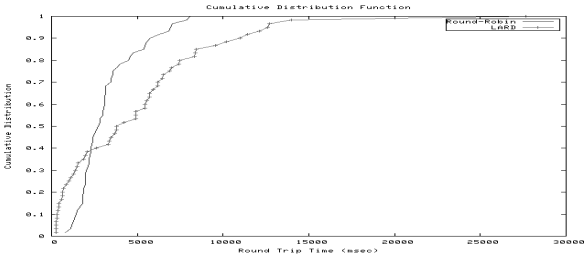

圖 9 Round-Robin v.s. LARD heavy loading

在負載輕時段，LARD 這組系統所測得 RTT 較 Round-Robin 小，我們再從圖 10及圖 11了解其原因為 cache hit rate 較高，在負載重時段，兩組系統 cache hit rate 降至 30~40%，我們進一步從圖 12及圖 13了解到此時段 httpd process 較多，耗用掉系統資源也較多。

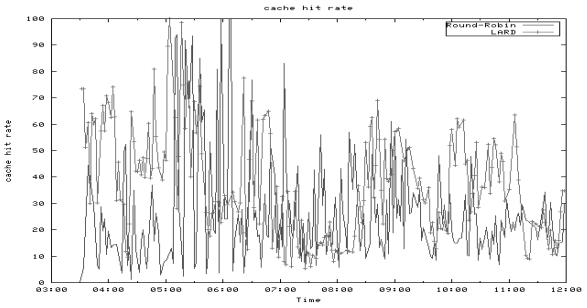

圖 10 cache hit rate - low loading

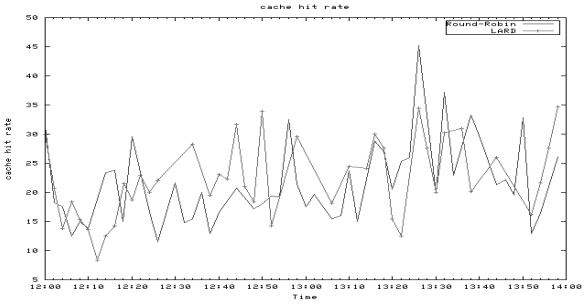

圖 11 cache hit rate - heavy loading

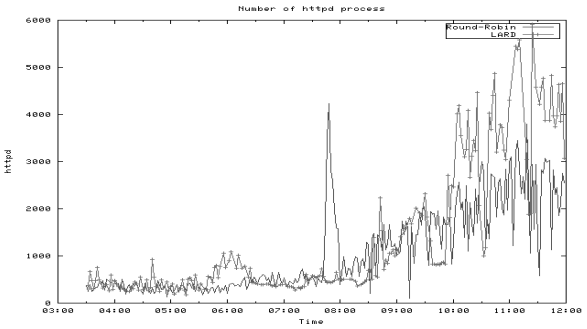

圖 12 compare system loading - low loading

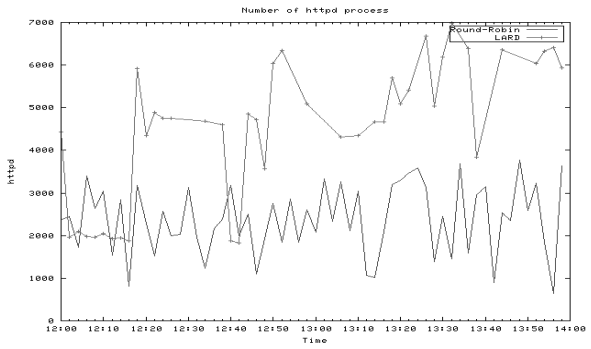

圖 13 compare system loading - heavy loading

由顯示採用 Size-Based 的方式表現不如 LARD，其中我們進一步去了解原因，由表格 4我們發現 EquilLoad、AdaptLoad 在 Data-Intensive Workload 上對於邊界點的決定較一般網站來的大，以第一個 node 的第一次計算出之邊界點來看，檔案大小介於 0~134MB 有一定的機率(P1)會由第一個 node 處理，若再考慮小檔案存取頻率較大檔案多且伺服器節點記憶體有限，大量大小相近之檔案擠入該伺服器節點，反因超過記憶體最大容量而降低快取擊中率。

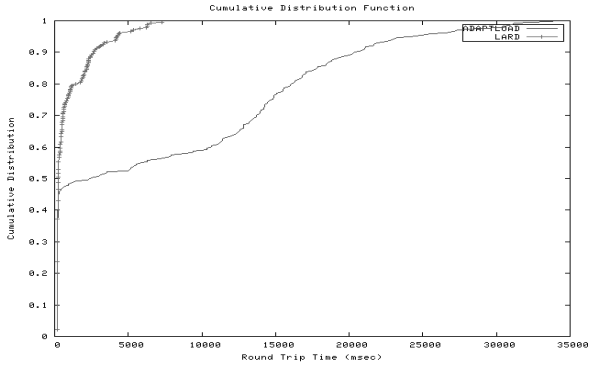

圖 14 AdaptLoad v.s. LARD

表格 4 AdaptLoad Boundary

Our site, S_1 Boundary	Nsysu trace, S_1 Boundary
134MB	131KB
268MB	65KB
134MB	65KB
536MB	65KB
536MB	32KB

6 結論

本文藉由實際運作的網站實作且量測幾種現有已被提出用於 Web Server Cluster 上，服務請求排程策略，確認 Content-aware 排程策略確實比 Content-blind 排程策略能獲致較好的效能，此外以往所提出之排程演算法未考量檔案大到一定程度時，該策略是否適用，我們以一實際且為 Data-Intensive 的網站加以測量，發現在模擬情況下運作不錯的策略在我們實際的環境上運作的並不理想，甚至有較糟的情況。因此我們認為在未來網路內容會日益龐大的情況下，目前所提出來的這些 content-aware 分配策略將不再適用，因此有必要研究更為適合的分配策略。

致謝

本論文承蒙國科會計畫: NSC 93-2213-E-110-038 之補助，特此致謝。

參考文獻

[1] ACME Labs, “http_ping：a measure HTTP latency software”, http://www.acme.com/software/http_ping/

[2] Alma Riska, Wei Sun, Evgenia Smirni, Gianfranco Ciardo, “AdaptLoad: effective balancing in clustered web servers under transient load conditions”, In Proceedings of the 22nd International Conference on Distributed Computin Systems, pages: 104-112, July 2002

[3] Eric Dean Katz, Michelle Butler, Robert McGrath, “A Scalable HTTP server: The NCSA Prototype”, Computer Networks and ISDN Systems, Vol. 27, pages: 155-164, November 1994

[4] Gianfranco Ciardo, Alma Riska, Evgenia Smirni, “EQUILOAD: a load balancing policy for clustered web servers”, Performance Evaluation, Vol. 46, pages: 101-124, September 2001

[5] Martin Arlitt, Tai Jin, “Workload Characterization of the 1998 World Cup Web Site”, technical report HPL-99-35R1, Hewlett-Packard Labs, September 1999

[6] Mauro Andreolini, Michele Colajanni and Ruggero Morselli, “Performance study of dispatching algorithms in multi-tier web architectures”, Performance Evaluation Review, Vol. 30, pages: 10-20, September 2002

[7] Qi Zhang, Alma Riska, Wei Sun, Evgenia Smirni, Gianfranco Ciardo, “Workload-Aware Load Balancing for Clustered Web Servers”, IEEE Transactions on Parallel and Distributed Systems, Vol. 16, No. 3, pages: 219-233, MARCH 2005

[8] Vivek S. Pai, Mohit Aron, Gaurav Banga, Michael Svendsen, Peter Druschel, Willy Zwaenepoel, and Erich Nahum, “Locality-aware request distribution in cluster-based network servers”, In Proceedings of the 8th ACM Conference on Architectural Support for Programming Languages and Operating Systems, pages: 205-216, October 1998