Scalable WWW Proxy Cache Server
之建置與分析研究

劉大川、張傑生、楊子翔
國立交通大學計算機與網路中心
新竹市大學路 1001 號
Phone: 03-5731905 Fax: 03-5714031
E-mail: ltc@news.cc.nctu.edu.tw, {jason,robe}@proxy.nctu.edu.tw

Abstract

近年來，隨著網際網路的普及，加上 WWW 多媒體介面的風行，網路頻寬的需求成長遠遠超過實際頻寬的擴充，這種現象的持續發生，後果就是帶來無止境的等等等，也有人稱之爲所謂的 World Wide Wait。

根據台灣學術網路骨幹頻寬使用狀況分析的結果[1][2]，大約 60% - 70% 的流量，都是 WWW 應用的傳輸，爲了有效抒解頻寬壅塞狀況，台灣學術網路技術小組於 1998 年決議將既有之骨幹頻寬分割，劃出所謂的 WWW 專屬頻寬，只允許各大區網中心的 WWW Proxy Cache Server 使用，希望能藉由 Cache 的幫助，降低重複傳輸數量，促成頻寬再利用，進而提升頻寬整體有效使用率。

經過一年多的 WWW Proxy Cache Server 建置經驗，我們發現了許多問題，亦提出相對應之解決方法，藉由本篇文章在此與大家分享。

Keywords: WWW、Proxy、Cache、Scalable

1. Introduction

短短幾年間，WWW 的快速走紅，非但加速了網際網路的普及化，更因此改變了人們的使用習慣，大一統網際網路媒體的存取介面。然而圖文並茂，再加上各式影音多媒體的背後，意味著資料量的暴增，首當其衝的就是早已滿載的網路頻寬，除此之外，WWW 伺服器面對著日益增加的傳輸量，只能望著節節高昇的系統負荷束手無策。

然而，隨著許多 WWW 相關研究如雨後春筍般冒出後，大家突然意識到，原來在網路上傳輸的 WWW 網頁，竟然有著相當程度的同質性，也就是對於某一特定族群，例如某一大學系所，或是某一研究單位，大家無時無刻瀏覽著相同內容的文章，參觀的網站也都集中在某些範圍，這種 locality 的特性，正是符合 Cache 大展身手的條件，於是乎，WWW Proxy Cache Server 的概念與實作，便被人們引入 Internet 的世界。

台灣學術網路於 1998 年起，爲了抒困頻寬日益壅塞的窘境，大力推廣各單位建置 WWW Proxy Cache Server，根據一年多來的觀察，成效斐然，至少於瀏覽網頁等基本服務，已經可以滿足大多數人的需求。另一方面，面對服務需求的高速成長，許多單位陸續加入第二台、甚至第三台 Cache Server，希望能夠降低每台機器的系統負荷，以維持服務品質的水準，接踵而來的問題，便是這些多重伺服器之間，如何協同分工，進而達到有效率的合作。

由於台灣學術網路負有研究的使命，本篇文章將針對由 NLANR 所發展，並且原始碼公開的 Squid [3]爲例，探討在多重伺服器的環境之下，如何能夠有效率的建構一個 WWW Proxy Cache Server 服務群。

本文第二章將由軟硬體以及作業系統的觀點，探討影響 WWW Proxy Cache Server 效能的因素，第三章將比較各種多重處理器協同分工之方法，第四章則介紹目前我們所建置的環境，第五章則做效能評估。

2. WWW Proxy Cache Server 之效能調校分析

在本章中，我們將由軟硬體以及作業系統的觀點，來探討各種影響 WWW Proxy Cache Server 效能的因素。

2.1 硬體

網際網路的頻寬租借費用，往往佔了整個網路營運成本的一大部分，這種情況對於越洋國際線路，表現得更是明顯。也因此，大多數的人都有一個共識，就是願意投資各種節省頻寬的 Cache 機器，包括 DNS、News、FTP、WWW。因爲比較起昂貴的頻寬，這些機器的建置費用，不過是九牛一毛。以下我們將分別對每一個系統組件個別討論：

2.1.1 CPU

在整個系統中，CPU 效能通常都不是最早遇到的瓶頸。然而，由於 CPU 肩負起 context switching 以及 I/O monitor 的重要任務，再加上要能夠快速地從巨大的 object index table 中，查詢 object 的存在與否、fresh 程度，我們依舊不能小覷 CPU 的重要性。尤其是在作業系統支援 SMP 甚至多重處理器的情況下，CPU 的數目可說是多多益善。

2.1.2 Memory

記憶體在整個系統中扮演的角色，重要性可說是數一數二。他的用途大概可分爲：

1. Hot Object Cache

這部分最主要是提升 response time，藉由 memory 高速存取的特性，將熱門物件存放在記憶體中，將可大幅降低 disk I/O，直接將物件送至使用者手上 。這一部份所用的記憶體數量，是可以彈性增減的，也就是降低使用量，雖然會增加反應時間，但是卻不會嚴重影響整體系統效能。

2. In-transmission Object

而這部分所需的數量，常常是比較無法控制的部分。由於 in-transmission object 數量多寡，與同時服務的連線數量有關，所以大體上說來，如果是提供大量服務的機器，必須預先規劃額外的記憶體，以備不時之需。

3. Cache Index

至於這部分，則是最基本的需求，也是往往人們容易忽略的一點。由於 Squid 必須在記憶體中存放一塊 cache object 的 index table，明確指出物件在儲存裝置的哪一層目錄、哪一位置，這意味著，記憶體的需求，是隨著儲存裝置的大小而成長的。往往人們只注意到增加儲存裝置的容量，然而卻忽略了必須相對應增加記憶體的重要性，導致儲存裝置容量越大，系統效能反而下降，而且有很多的情況，是儲存裝置根本無法全數使用，因爲系統根本無法維持夠大的 index table。總歸一句話，想要提升整體效能，記憶體可以說是多多益善。

2.1.3 Storage

最常見的儲存裝置，非硬碟莫屬。事實上，也唯有硬碟，才符合便宜、快速、以及高容量的特性。對於整體系統效能來說，如何才能夠在一定時間內，提供最多的服務數量，對於這點，硬碟佔了關鍵性的地位。 原因很簡單，在大部分的情況之下，系統的瓶頸都會出現在儲存裝置。仔細分析系統運作，儲存裝置的主要工作可分爲：

1. Cache HIT - Object Read From Storage

當系統收到使用者的 requests 後，在記憶體中查表發現，儲存裝置中存有使用者所需求的物件，此時硬碟的工作就是將物件讀出。

2. Cache MISS - Transferred Object Saved To Storage

當系統沒有使用者需要的物件時，還是必須替使用者取回，而後，必須存入儲存裝置，以利往後使用者取用。

3. Expire Old Cache Objects

由於儲存裝置的容量並非無限，當使用率接近滿載時，必須將老舊過時的物件清除，否則新物件無法存入。這一部份的工作其實負荷不輕，因爲牽涉到大量的記憶體查表，以及硬碟 I/O，更嚴重的是，對於硬碟來說，大部分要移除的物件，都是散落各地，也就是考驗硬碟的 non-squential I/O 效能。

由以上三點不難發現，所有的動作都是在考驗硬碟 I/O 的能力，無論是 read、write、seek 等等。對於儲存裝置，我們的建議是，盡量選擇高轉速的硬碟，也就是 seek time 越短越好，理論上，seek time 牽涉到絕大部分的系統效能。

每一次 request 所需時間 =
seek time + rotation latency + read object

假設我們使用 seek time 10ms 的硬碟，而非常理想地，每一個物件都能剛好在一次 seek 就能讀取完成，並且我們將其他的時間省略（磁頭定位、碟盤旋轉），那麼，每一顆硬碟，每秒的服務次數理論值，則是 100 次 request。在使用 SCSI 裝置的情況下，每一顆硬碟可以同時動作，也就是說，可以提供服務的次數，與硬碟數量成正比。我們的第二個建議，則是使用多個低容量硬碟，而非少數高容量硬碟。

最後一點值得注意的是，常常有人會使用 RAID 當作儲存裝置，其中又以 RAID level 5 爲最多。事實上，這是會明顯拖慢系統效能的。理由很簡單，RAID 3、4、5 都會因爲 data integrity 的考量，而加入 parity check 的功能，並且將資料分散存放，平均散佈在每一顆硬碟中。於是，爲了服務任何一個 request，都必須牽涉到一顆以上的硬碟 I/O，明顯的，能夠同時提供服務的數量，當場減半。再者，Cache 內的物件，我們並沒有必要負起「保管」的責任，也就是說，當硬碟損壞時，資料流失並不會給我們帶來任何困擾。是以，盡量避免多餘的 overhead，使用最爲簡單的方式作爲儲存裝置，反而給我們帶來更高的效率。

2.1.4 Network

網路效能常常也是系統會遇到的瓶頸之一。尤其是將 server 置於一個擁擠不堪的網域中，包括 ethernet 先天的特性，switch/router 的處理速度、throughput 能力，通通都會造成整體輸出無法提高的結果。如果能夠增加網路卡數目，讓 server 同時跨於不同的子網路，根據經驗，的確可以大幅提升 throughput。並且，server 的放置，必須考慮到整體網路的 topology，將其中一片網路卡，放於最靠近出口的網域，也能夠減少封包長途跋涉的困擾。

2.2 作業系統

選用一個穩定性夠高，功能強大的作業系統，對於系統的效能，絕對有實質的幫助。新一代的作業系統，都有支援 multi thread 的特性，如果配上多重處理器，更是如虎添翼。至於在 disk I/O 的部分，以 Linux、FreeBSD 來說，都有支援所謂的 asynchronous disk I/O mode，利用非同步更新的特性，可以大幅增加硬碟寫入的速度，以上這兩點都是決定 server 效能的關鍵性因素。

另外，在 format 硬碟的時候，必須先分析以往儲存的物件，大略算出物件的平均大小，依此決定 format 的參數，例如 inode 數量與 block 大小。過多的 inode 會造成儲存空間的浪費，但是 inode 不足的情況，則會無法利用剩下的儲存空間。至於 block 的大小，則會影響硬碟讀寫效能。如果能夠參考物件大小的分佈圖，將 block 大小設定於滿足大部分物件，則將可降低讀取次數，間接提高單位時間的的服務次數。

2.3 軟體

對於 Squid 本身來說，有幾點必須要特別注意。首先就是良好的 DNS server，由於 URL 中的 host 必須轉換成 IP 才得以存取，所以 Squid 會對 DNS server 大量查詢，所以 DNS 回覆的速度，將會間接影響到 Squid 的效能，特別是當 DNS server 出錯時，Squid 常常因此而罷工停擺，因此，系統的效能，與 DNS server 息息相關。

另外一點，則是要妥善設定 expire 相關的參數，在 Squid 中，有兩個 expire 數值需要設定，分別爲高水位以及低水位。每一次儲存裝置使用率到達高水位後，系統將執行 expire，移除過期物件，將使用率調降至低水位。然而，expire 這個動作，由於牽涉到大量的記憶體查詢以及 disk I/O，每次執行都會造成系統效能降低，必須盡量避免。解決之道，爲拉大高低水位的差距，將低水位之值降低，如此，可以減少 expire 的次數。此外，盡量讓系統於深夜、凌晨執行此一動作，減少對使用者的影響。

2.4 與其他伺服器之合作軟體

Squid 的一大特性，就是能夠與其他伺服器溝通，彼此互通有無，分享 Cache 物件。而此一動作，必須透過彼此間的 ICP （Internet Cache Protocol）查詢。然而，根據觀察，大量的 ICP 查詢，往往也是降低系統效能的主要因素。由於 ICP 查詢與一般的 HTTP request 相同，都必須迫使系統做 index table lookup，以檢查是否該物件存在 Cache 中。而此一動作，牽涉到大量 memory I/O，對於系統負荷，可想而知。

我們所提出的建議爲，盡量避免。也就是說，除非必要，否則不要送出 ICP 查詢。在此歸納幾點：

1. 大型伺服器不要對小型伺服器查詢
例如區網中心的伺服器，查詢的對象應該是其他區網中心的伺服器，而不是一般國中小學的迷你伺服器。由於 Cache 容量差距過大，通常送往小伺服器的查詢，很少會得到 HIT 的回答。

2. 只針對國外網站送出查詢
以國內網站來說，在大部分的情況下，自行抓取絕對會比送 ICP 查詢，再決定是否自行抓取來得快。再者，國內頻寬的價格，比起國外頻寬，幾乎可以忽略不計。是以，必須將 ICP 用在最需要的地方，也就是，浪費國內資源，節省國外資源。

3. 不要送出沒有意義的查詢
有許多物件，是屬於無法 cache 的，也就是每次的傳回值都不同，例如搜尋引擎以及聊天室等等。這類含有 CGI/ASP/? 等等 keyword 的 URL，在送出 ICP 查詢前，應該預先過濾，以免浪費頻寬，並且造成其他伺服器的困擾。

3. 多重伺服器協同分工之比較

單一伺服器的最大問題，就是終究會達到其效能的上限，通常也就是硬體的瓶頸無法突破。在 Computer Science 的世界中，Cluster 的觀念早已被提出，然而，對於 Cache Server 來說，要能夠做到 cluster 的境界，恐怕還有一大段路要走。本章將比較目前較爲常見的幾種 Cache Server 分工合作方式。

3.1 DNS Round Robin

好幾年前，就已經有人提出用 DNS round robin 的方式，將多台伺服器連結，對外服務，如圖 1。

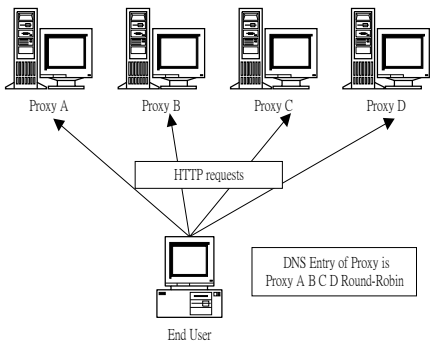


圖 1：DNS Round Robin

這種方式的最大好處，對於使用者來說，他們只要記得某一個代表號的 FQDN，而不必費心強記一串 IP。對於管理者而言，可以很方便的隨時增減伺服器，擴充彈性充裕，而且，這些伺服器還可以分散到不同網域，以達到網路負載平衡的理想。然而，這種方式卻有一個無法避免的缺點，那就是缺乏 fault tolerance，一旦單一伺服器出錯時，DNS Server 不可能主動發現，所以使用者將會不定期遇到煩人的錯誤訊息。

而對於伺服器群來說，彼此之間的溝通合作，仍存在許多困難。舉例來說，如果彼此間獨立運作，那麼 Cache 物件重複的情況將非常嚴重，也就是資源浪費。若是彼此之間利用 ICP 查詢，避免儲存相同的物件，那麼伺服器之間的 ICP 查詢數量，將氾濫至嚴重影響系統效能的境界（一份 HTTP request 將轉成三份 ICP query）。

因此，近來幾乎已經沒有看到有用 DNS round robin 來做 Cache Cluster 的例子了。

3.2 Multi Layer Cache Hierarchy

由於 DNS round robin 有著 user friendly 的好處，然而卻有伺服器難以互相溝通合作的困擾，於是有人提出多層式架構，也就是能夠兼顧兩者好處的解決方式，如圖 2。

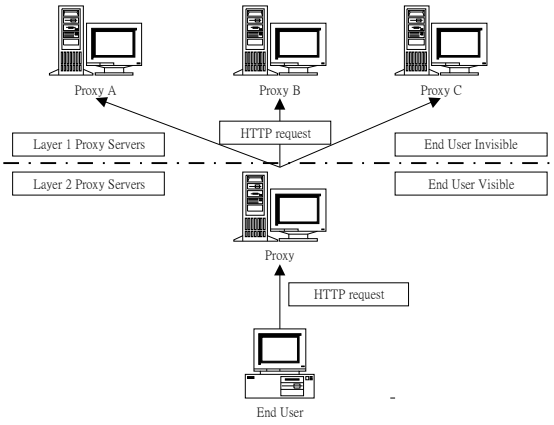


圖 2：Multi Layer Cache Hierarchy

對於使用者來說，面對他們的，仍舊是那部叫做 Proxy 的機器，然而，對於 Proxy 這部機器而言，他所做的事情，大部分都只是轉手的工作，也就是將所接收到的 HTTP requests，分析判斷之後，交給上層真正負責的伺服器，對於上層的伺服器來說，他們可以劃分工作範圍，分別負責不同的 domain（.com、.tw）或者是 protocol（HTTP、FTP），避免造成 Cache 物

件重複存放的浪費。而且，由於上層的伺服器群是使用者看不到的，所以管理者可以任意增減機器，以調整整體效能，符合彈性的原則。

然而，如同之前 DNS round robin 的問題一樣，fault tolerance 的問題仍舊沒有解決，如果面對使用者的伺服器出問題，對於使用者而言，將無法瀏覽網頁。而且，這種兩層，甚至多層式的架構，將面臨嚴重的效能衰減的麻煩。因爲多次轉手的後果，就是造成所有的 HTTP requests 反應遲緩。更令人頭大的是，通常在整體系統中，最先發生頻頸的地方，就是面對使用者的那台機器，試想，這一部機器所要承受的 requests 數目，是上層伺服器群所承受的總和，而他又是整個架構中負責承上啓下的樞紐，一旦機器無法承受負荷，整個架構馬上就垮了。

3.3 Proxy Auto Configuration

由於前面所提到的兩種作法，都無法符合 fault tolerance。於是，就有人由另外的方向思考問題，將腦筋動到 end user 的瀏覽器上，也就是 Netscape 提出的 Proxy Auto Configuration[4]。

Proxy Auto Configuration（以下簡稱 PAC）事實上只是一個 java script，相較於以往手動指定 proxy server，PAC 可以提供更大的彈性。當你每次發出存取某一 URL 的要求時，整個 URL 就會當成參數交給 PAC script 判讀，然而在 script 中，可以指定多部伺服器，也就是說，當某部機器出問題的時候，PAC 會自動將 request 轉到其他機器，以達到 fault tolerance 的效果。除此之外，也可以在 PAC 中判斷 URL，將不同 domain 或者是不同 protocol 的 request，自動導向到不同的伺服器。

最重要的是，對於使用者來說，他們的 proxy 設定檔，只是短短的一行 URL，例如：
http://proxy.nctu.edu.tw/proxy.pac。
對於系統管理者來說，如果他需要對伺服器數目做任何增減或是調整系統設定，他只要更改此一 script 的設定，這個動作，對於使用者來說是完全 transparent，也就是說，PAC 同時滿足了使用者所需要的設定簡單化與管理者所需要的系統彈性化。

觀察目前大部分的 Proxy Server，都已經要求使用者採用 PAC 設定，而不要用以往的指定單一伺服器方式，顯然，這個方式最爲大部分的人所接受。

4. 系統規劃建置

綜合以上兩章的討論，並且爲了提出有效的解決方案，我們在交通大學計算機中心，架設 WWW Proxy Cache Server，藉此觀察結果，並驗證成效。

4.1 需要解決的問題

在提出系統架構之前，我們必須先提出兩點重要的關鍵問題。首先就是要如何在多重伺服器的環境下，協同工作，再來就是如何有效減少不必要的伺服器負擔。

4.1.1 多重伺服器的分工問題

在多重伺服器的環境之下，大家通常直覺地就會想到利用 ICP 的方式，溝通各台伺服器，以分享其 Cache 物件。然而，根據我們的觀察發現，這其中仍存在著許多問題。

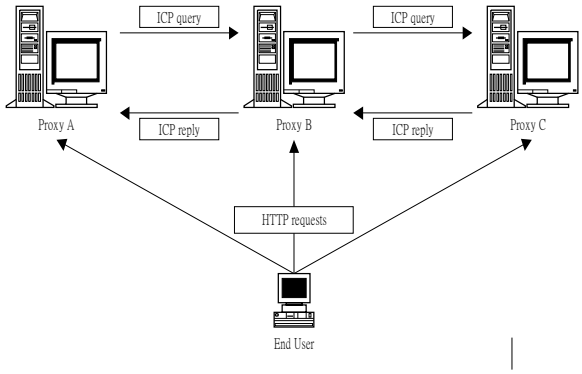


圖 3：多重伺服器環境下的 ICP 溝通

如上圖 3，假設使用者向 Proxy A 發出一個 HTTP request，當 Proxy A 收到的同時，一定會先檢查自己的 index，如果發現物件存在，則繼續檢查其 freshness，如果這個物件還在 fresh 的狀態，則可以直接送給使用者，反之，如果發現此一物件已經進入 stale 的狀態，則必須向原始的 WWW 伺服器發出 IMS 查詢（If Modified Since），檢查是否該物件有被更動過。如果物件確定是不需要更新的，則伺服器才能夠將此一物件，傳送給使用者。

然而，如果當 Proxy A 發現自己 index 內並無此一物件時，它將會向 Proxy B、C 發出 ICP 查詢，看看是否 Proxy B、C 存有此一物件。對於 Proxy B、C 而言，如果他們沒有該

物件，那麼就會回覆 Proxy A ICP MISS，或者是他們發現自己存有此一物件，而且此一物件仍在 fresh 的狀態下，他們將會回覆 ICP HIT。然而，如果此一物件是在 stale 的狀態下呢？問題就出在這裡。

在一般的狀況之下，Proxy B、C 是不會主動替這一個物件送出 IMS 查詢的，也就是他們會直接告訴 Proxy A ICP MISS。然而，一個 IMS 查詢，不過是幾百 byte 的封包，如果 Proxy A 因此要自行抓取該物件，可能是幾百 KB 甚至數 MB 的資料量，頻寬的浪費，可見一般。

因此，在我們的規劃中，我們清楚分配伺服器的工作，並不會讓他們工作範圍重疊，一來避免上述問題，二來彼此間獨立運作，並不需要 ICP 查詢，對於維持系統效能來說，反而是一件好事。

4.1.2 減少不必要的伺服器負擔

根據長時間的觀察，我們發現伺服器常常在處理一些沒有必要的 HTTP requests，平白增加伺服器負荷，以及浪費寶貴的 Cache 空間。

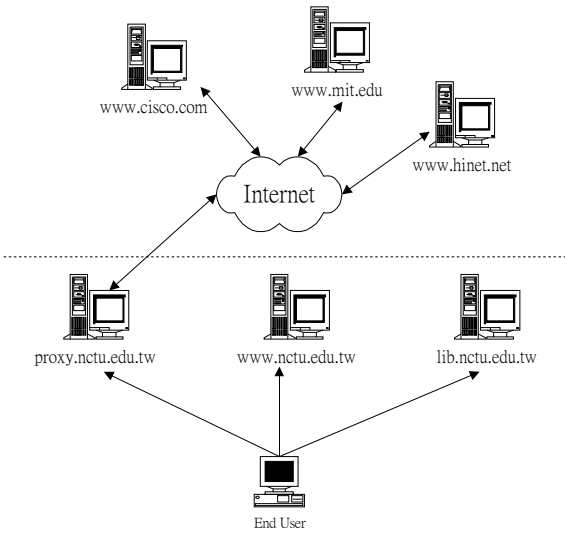


圖 4：伺服器之網路位置示意圖

由圖 4 所示，我們可以很清楚的看出，以最下方的使用者而言，存取同是位於校內的 proxy server 以及 www 與 lib 來說，理論上都應該是相同時間（或者說相差無幾）。直覺地，對於校園內的網站，我們就不需要也不應該透過 proxy server 轉接，除了徒增存取時間之外，更增加 proxy server 的負擔，進而在 proxy server 的 cache 中佔用寶貴空間。事實上，這種情況，對很多 proxy server 是屢見不鮮的。

另外，我們也觀察到，proxy server 每日都有收到非常大量的 HTTP request，連往一些股票網站，而那些股票行情資訊，由於具備有即時的特性，也就是物件 fresh 的時間非常短，導致幾乎 99% 的 requests 對 proxy server 來說，都只是單純的代轉，完全無法做 Cache 的動作，對於這類的 requests，也應該由使用者直接對該 WWW 網站存取，不應該透過 proxy server。

除了以上所提到的兩種比較明顯的 HTTP requests 外，還有幾種也是可以過濾的，例如現在有蠻多廣告網站，他們送出的網頁，具有跟股票網頁相同的特性，也就是將 fresh 時間設定非常短暫，以及隨時更換新的內容。此外就是搜尋引擎，那些查詢結果的網頁，因人而異，根本不可能 Cache。最後還有聊天室，這種 HTTP requests 對於 proxy server 來說，也是只能代轉而已。

以上所列的這幾種 HTTP requests，通通都可以由 Proxy Auto Configuration 的 script 中檢查出，並且過濾之。

4.2 系統規劃

我們所規劃的架構如下圖 5。

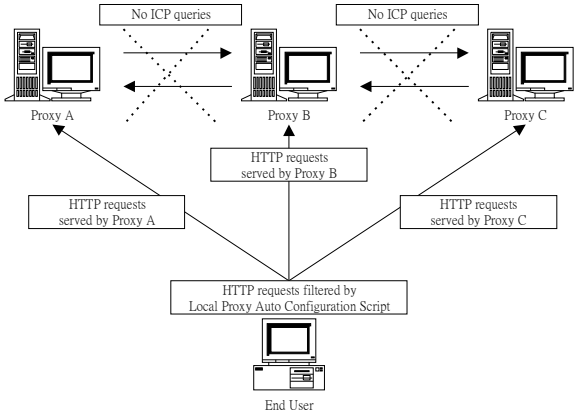


圖 5：規劃的 Proxy 架構

我們可以分成兩個部分來討論，分別爲 client 端與 server 端。

4.2.1 Client 端

對於一般使用者,我們提供唯一的 proxy 設定方式，也就是指定 Proxy Auto Configuration script 的 URL,我們並不鼓勵使用者直接設定使用某一伺服器。

至於這份 PAC script，則幫我們解決了所有前述的問題。在此將其功能大致解釋如下：

過濾錯誤的 HTTP requests
常常有使用者寫錯 URL，例如不小心輸入一些不合法的字元(,_\),如果這些 requests 都必須送至 proxy server 才能作判斷,對伺服器的效能多少會有所影響,所以在 client 端就完成判斷，對大家來說都是一件好事。

Bypass 不需經過 Proxy 的 HTTP requests
首先判斷目的地的 IP 與 domain，如果發現同是在校內，或者是位於高速網路骨幹之上的節點(NCTU、NTHU、NCHC),則要求 browser 直接對該目的地存取，不要透過 Proxy。再來判斷目的地是否在預先列出的名單中，也就是符合即時股票資訊的網站，或是廣告網站等等。最後則是針對 request 的 URL 分析，如果含有特定 keyword 例如：CGI、ASP、?等等字元，也是直接存取，不經過 proxy。

分配 request 至負責的 Proxy Server
完成以上判斷之後，也就是符合必須透過 Proxy 存取的條件後，則對 request 的目的地作分析，根據不同的 domain，將 request 交給不同的 Proxy 處理。

4.2.2 Server 端

如上圖 5 所示,我們共有三部伺服器同時提供服務，其硬體配備完全相同，大致說明如下：

CPU：Pentium II 450MHz
RAM：1GB
Storage：SCSI 7200rpm 9G HD * 7
Network:兩組 100BaseTX 介面,分別接在對校外的 7513 router，以及校內的 Gigabit Ethernet Switch 之上

同時，爲了避免 4.1.1 節提到的問題，以及第三章所列出的多層轉手 overhead,在規劃上，我們必須多所考慮。

1. 避免多層轉手的 overhead
爲了避免 HTTP requests 在多層架構中多次轉手，我們將三台機器直接面對使用者，提供服務。這三部機器對外存取資料，將直接對目的網站發出請求，不透過其他機器代轉。

2. 平均分配國內外網站
在以往的架構，我們曾經嘗試將工作劃分爲，一部負責國內網站，兩部負責國外網站，過沒多久，我們馬上發現一個嚴重的問題。負責國外網站的機器，承受過多的 ICP 查詢，而負責國內網站的機器，則是窮於應付大量快速的連線要求。因此，我們決定將工作平均分配，每一部機器同時負責一部份的國內網站與國外網站，如此一來，各機器大致可以負擔相當的 ICP 查詢的數量，以及快速連結的國內連線，與低速連結的國外連線。

3. 減少不必要的 ICP 衝擊
由於大量的 ICP 查詢將嚴重影響系統的正常運作，在我們的規劃中，三台機器將獨立運作，各自負責不同的網域，也就是彼此間不需要任何溝通，免除 ICP 查詢。同時爲了減少不必要的 ICP 查詢，我們也尋求其他 proxy server 的合作，在送出 ICP query 之前，先行過濾含有 CGI/ASP/? 等關鍵字的 URL。

4. 預先過濾錯誤或是不合法的連線要求
分析 log 可以輕易的發現,伺服器常常會收到錯誤或者是不合法的連線要求，例如由 ISP 用戶所送出的連線要求，或者是其他非骨幹網路伺服器所送出的大量 ICP 查詢。雖然 Squid 可以輕易判斷這些錯誤的連線，並且不予以提供服務。然而這項無意義的工作，卻也會間接降低系統效能。每次 Squid 收到這類請求時，必須先檢查 ACL 設定，發現爲不合法請求時,將發出 DENIED 訊息,同時在 log 中記上一筆，然而，通常這類的請求，尤其是 ICP 查詢，都是成千上萬筆蜂擁而至，對於 Squid 來說，雖然成功阻擋了連線請求，卻也付出沈重的代價。

因此，我們目前的作法，爲每日分析 log，並且列出黑名單,直接將其 IP 由 router 預先過濾，或者是由 proxy 伺服器上的 IP filter 程式過濾,如此一來,Squid 免除了查表、判斷、發出錯誤訊息、記錄等等繁瑣工作，將可專心致力於正常的服務。

5.效能評估

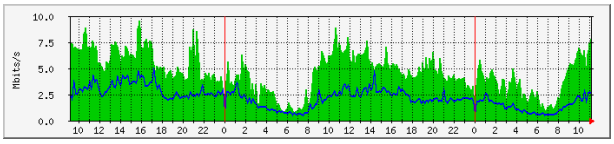
自從 1998 年初，我們積極建置 Proxy 伺服器後,便時時注意 log 統計分析，以及監測網路流量變化。持續一年多的觀察比較，我們將分成以下幾點討論，驗證我們所得到的成果。

5.1 Proxy Cache Log 統計分析

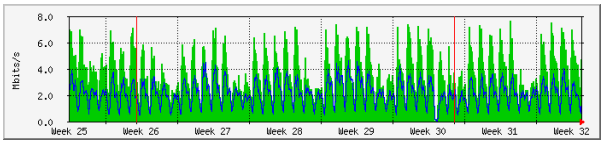
根據長時間的觀察，目前三部機器每日服務的傳輸量，大約在 60G-80G 之間（寒暑假以及例假日會明顯降低），服務的對象包括新竹區網的學校（交大、清華、中華、其他高中國中國小等），以及 TANet 骨幹上的其他 Proxy 伺服器。至於 hit rate 來說，request 部分（網頁、URL）大約在 40%-50% 之間，如果是以傳輸量來看，則高達 45%-60%，詳細分析 log 後可以明顯發現，因爲存在許多熱門的圖片、動畫與 MP3，尤其是後兩者動輒 5-10MB，當多人同時下載相同的物件時，對於傳輸量的 hit rate 來說，可說是大量增加。然而站在節省頻寬的立場而言，這種情況也正是發揮 Cache 效率的最佳機會。

5.2 Layer 2 流量分析

由於我們的伺服器分別跨接在兩個網域[5]，也因此我們可以輕易算出進出雙向的流量。以下我們只針對校外流入（目的網站與其他伺服器提供）以及校內傳輸的部分加以討論。



校外傳入（藍色）	Maximum	5048.0 Kb/s	Average	2632.0 Kb/s	Current	2728.0 Kb/s
傳入校內（綠色）	Maximum	9624.0 Kb/s	Average	4640.0 Kb/s	Current	7880.0 Kb/s



校外傳入（藍色）	Maximum	4440.0 Kb/s	Average	2480.0 Kb/s	Current	2448.0 Kb/s
傳入校內（綠色）	Maximum	9184.0 Kb/s	Average	3832.0 Kb/s	Current	6144.0 Kb/s

以上爲 1999 八月的流量資訊，我們可以經由很簡單的公式算出：
頻寬提升率 =（傳入流量 – 輸出流量）/ 傳入流量

利用八月的平均值帶入，（3832-2380）/2380 = 0.61，也就是大約 61% 的提升，參考上一節的 log 統計，非常符合傳輸量的 hit ratio。

5.3 Layer 3/4 流量分析

根據 Layer 3/4 流量統計[6]，比較 1998 與 1999 六月份的流量內容，WWW 佔整體輸入比例，由 33% 降至 18%，然而當月的總傳輸量，分別爲 4182 GB 與 7894 GB，WWW 傳輸量則爲 1380 GB 與 1421 GB，其中傳輸量大幅上升的主要因素，歸功於 TANet 對國外的專線由兩條 T1 提升至 T3，而國內骨幹網路，也大多提升至 T3。因此，比較起總傳輸量的大幅增加，WWW 傳輸量僅僅小幅上揚，以及 Proxy 伺服器的 throughput 仍舊持續成長，再加上客觀環境顯示，WWW 熱潮在網際網路上絲毫看不出任何衰退的跡象，我們可以肯定 Proxy 的 Cache 功能，有效達成降低頻寬使用的任務。

另外，根據 WWW 對外傳輸狀況的 IP 分析，可以發現 NCTU Proxy 伺服器佔總傳輸量比率的 50% 以上，其它的部分包括了新竹區網的其他 Proxy 伺服器，以及使用者直接連線的部分。對此結果，我們感到相當滿意，顯示大部分的使用者都是透過 Proxy 取用 WWW 網頁資訊。

5.4 使用者習慣分析

今年五月一日，當教育部開始全面限制非 Proxy 伺服器出國存取 WWW 資訊時，交大幾乎沒有什麼人發出質疑，也未在網路上見到一堆文章，詢問如何設定 Proxy。對於長期推動使用 Proxy Server 的交大計中，可說是非常欣慰的一件事。

6. 結論

在未來的網路世界，可以預見得到，頻寬需求的成長速度，依然會大幅領先頻寬升級的速度，WWW 的發燒熱，短期內也絕對不會降溫，如何能夠將有限的資源，做最有效的分配，絕對還是熱門的研究話題。

解決 WWW 的問題的最有效方法，就是建置 WWW Proxy Cache Server。爲了邁向此一目標，我們必須由兩方面著手。首先就是教育使用者，推廣 Cache 的觀念，鼓勵大家使用

Proxy 伺服器，避免重複浪費頻寬。再來則是技術方面，除了持續建置高效能的伺服器群外，另外也應該嘗試利用深夜凌晨間，頻寬閒置的時候，對國外網站進行物件 refresh，以保持 Cache 的新鮮度。此外，各區網中心的 Proxy 伺服器，利用這段時間，互相交換熱門物件，也是值得實驗的方向。

7. 參考文獻

[1] http://netflow.edu.tw

[2] 劉大川，「台灣學術網路出國線路壅塞狀況之分析研究」，TANet 98

[3] http://squid.nlanr.net/Squid/

[4]http://home.netscape.com/eng/mozilla/2.0/relnotes/demo/proxy-live.html

[5] http://mrtg.nctu.edu.tw

[6] http://netflow.nctu.edu.tw