Web Clusters 的以時間為基礎的需求分配排程法之設計與實作

Design and Implementation of Time-based Request Dispatching Algorithms for Web Clusters

黃銘毅　　姜美玲　　吳孟潮

國立暨南國際大學　　國立暨南國際大學　　國立暨南國際大學

s0213513@ncnu.edu.tw　　joanna@ncnu.edu.tw　　s1213522@ncnu.edu.tw

摘要

近年來，由於個人電腦的價格低廉以及網路的普及化，使得網際網路的流量快速增加，因此，以叢集式系統來取代高速主機成為近年來的趨勢。為了要提昇叢集式系統整體的效能，影響負載平衡的需求分配的排程法是一個重要的研究課題。本論文提出以時間為基礎的排程法來提供有效的負載共享，並實作於 Linux 的 LVS project 內，實驗結果顯示，當後端伺服主機的硬體等級不一致或者相同時，以時間為基礎的排程演算法都能有效地提昇叢集式系統的效能。

關鍵詞：叢集式系統、LVS project、Linux

Abstract

In the recent years, because the price of personal computer is so cheap and people extensively uses network to connect to the web sites, the traffic of World Wide Web increases dramatically. Since cluster systems have the advantage of cost superiority than supercomputers and are able to satisfy the demand of users, using cluster systems to take place of supercomputers in business organizations has become a trend in the recent years. To improve the performance of the whole cluster system, the design of an effective request dispatching algorithm that affects the load balancing of a cluster system is an importance issue. In this paper, we propose a set of time-based request dispatching algorithms to provide effective load sharing and implement them on Linux LVS project. Performance evaluation results show that the proposed time-based request dispatching algorithms can effectively improve the performance of homogeneous and heterogeneous cluster systems.

Keywords: Cluster Systems、LVS project、Linux

一、簡介

WWW 的服務快速成長及使用者日益劇增的情況下，使得 WWW 的流量呈急速增加，為了避免單一主機超載，因此有許多企業改採用叢集式系統來平衡負載，採用叢集式系統可以依照系統的負荷來增加主機的個數來分擔工作量。然而隨著新設備的加入，叢集伺服系統擁有不同等級的新舊伺服主機，如何讓等級不同的伺服主機有效地運用於伺服主機群，於是要藉助排程法來有效地分配並轉送所有使用者的要求（Request）到適當的伺服主機服務，才不會有負載不均所造成某一後端伺服主機的工作量超載或者當機的情況發生。

為了設計出一個有效分配的機制，本論文提出了以時間為機制來反應後端伺服主機目前的負載情形，選擇一台主機來當作分配工作伺服主機，其主要工作負責作有效的分配服務要求及計算服務要求的完成時間等機制來反應後端伺服主機的負荷狀態，以後端伺服主機的負荷來決定工作的排程。

由於 Linux 是開放性原始程式碼，具有高效能及穩定的作業系統，已被廣泛的使用來架設各種的伺服器，所以本論文將所提的以時間為基礎排程法實作於 Linux Virtual Server Project（LVS project）中[3]。然而 LVS Project 中原有八種的排程法皆無法得知後端伺服主機目前的負荷情況，因此無法依照後端伺服主機目前的負荷情況動態地分配工作至適當的伺服主機，而且，在 LVS Project 中，僅有以設定權重（Weight）來反應後端伺服主機的異質性，然而根據 Weight 來分配 request 權重是在初始時即需設定且不會隨著主機負荷情形動態地調整。

本論文提出以時間為基礎的排程法，其原理為，當伺服主機的負載較重時或者當後端伺服主機群的硬體等級差異大時，硬體等級較差

的伺服主機，所要完成前端工作分配主機所給予的工作時間會較久，依此特性，可以反應後端伺服主機目前的負荷情況以及硬體等級之異質性。實驗結果顯示以時間為基礎的排程法可以更有效地改善異質性與同質性的叢集式系統的整體效能。

二、Linux Virtual Server 之分配排程演算法

目前在 Linux 核心 2.4 版的 LVS project[3] 中提供了八種平衡負載的演算法供網站系統管理者來使用，說明如下：

(1) 輪流（Round Robin，RR）排程法

由前端工作分配主機依輪流方式將服務的要求分配給後端伺服主機群來完成。

(2) 權重式輪流（Weighted Round Robin，WRR）排程法

根據後端伺服主機的效能等級來訂定權重，所謂權重的定義可訂定工作分配的比重值，假設有 n 台的後端伺服主機，Wi 為各後端伺服主機的權重值，當總工作量為 $\sum_{n}^{i=1} Wi$ 時，則每一後端伺服主機被分配到的工作量為 Wi。此法的優點可使效能較好的後端伺服主機可服務較多的工作，使得叢集式系統達到最佳的效能。

(3) 最少連線數（Least Connections，LC）排程法

由於前端工作分配主機會記載目前後端伺服主機的連線數，前端工作分配主機會依據目前連線數最少的後端伺服主機優先分配使用者的服務要求。

(4) 權重式最少連線（Weighted Least Connections，WLC）排程法

使用權重來訂定後端伺服主機的服務能力，如同使用權重式輪流排程法一般，假設有 n 台後端伺服主機，每一台後端伺服主機均設定一個權重 Wi，並且目前的連線數為 Ci，前端工作分配主機會根據以下公式來選擇最適當的後端伺服主機：

最適當的後端伺服主機 = min（ $\frac{Ci}{Wi}$ ）

(i=1、、n)

(5) 臨近性最少連線（Locality-Based Least Connections，LBLC）排程法

此排程法的想法在使用者的服務常常會索取相同網頁內容來觀看，因此前端工作分配主機會去維護同一目的地網路位址去對應最近使用的後端伺服主機，若該後端伺服主機目前是可用且系統沒有負載過度時，則由該後端伺服主機來服務，若此目的地網路位址尚未對應到後端伺服主機或者是該後端伺服主機目前在忙碌中，則使用最少連線數法來選擇一台最適的後端伺服主機，將服務要求分配給後端伺服主機。

(6) 臨近性集群最少連線（Locality-Based Least Connections with Replication，LBLCR）排程法

LBLCR 與 LBLC 排程法相異之處在於 LBLC 排程法是從一個目的地網路位址對應到一台後端伺服主機，而 LBLCR 是一個目的地網路位址對應到一群後端伺服主機，其流程為根據服務要求的目的地網路位址去找出對應的後端伺服主機群，使用最少連線數排程法去選擇一台最適的後端伺服主機，若所選的後端伺服主機目前尚未處於忙碌的狀況下，則把此服務要求轉送給此後端伺服主機，若此後端伺服主機目前處於忙碌狀況時，則表示後端伺服主機群目前均是忙碌的，則使用最少連線數排程法去選擇一台後端伺服主機來加入群集中，當這一群後端伺服主機有一段時間沒有修改伺服主機名單時，則把目前連線最多的後端伺服主機移出此群集，以減低群集數量，避免後端伺服主機過多而主機的系統資源沒有達到最佳的利用。

(7) 根據目的地位址雜湊（Destination Hashing）排程法

前端工作分配主機會依服務要求的目的地網路位址，作為雜湊的鍵值（Hash key），由靜態的雜湊表中找出對應的後端伺服主機，當此後端伺服主機目前處於可服務狀態時，則前端工作分配主機將服務轉交給此後端伺服主機。

(8) 根據來源位址雜湊（Source Hashing）排程法

前端工作分配主機會依服務要求的來源網路位址，作為雜湊的鍵值（Hash key），由靜態的雜湊表中找出對應的後端伺服主機，當此後端伺服主機目前處於可服務狀態時，則前端工作分配主機將服務轉交給此後端伺服主機。

三、以時間為基礎的分配排程演算法之設計與實作

現有 LVS project[3]的排程法均為靜態排程法，不會依據叢集式系統中後端伺服主機群各個目前的負載情況來調整，但是要得知後端伺服主機群目前的負載情況時，通常的作法是在前端工作分配主機與後端伺服主機中各實作一個網路程式，每隔一段時間就會收集後端伺服主機群目前系統的負載情況，藉此資訊來平衡後端伺服主機的負載狀況，此法的優點是可使得 IPVS 模組獲得更多後端伺服主機負載情形，能夠更有效地作適當的排程調整，但是缺點在於收集負載情況之網路程式會消耗前後端伺服主機的系統資源及網路的頻寬。另外，尚有要隔多久時間去收集負載資訊的問題，若間隔時間太久時，其負載資訊容易過時，若間隔時間太短時，則會浪費系統資源。

為了要能動態地調整排程，我們提出以時間為基礎的排程演算法，以時間為基礎來反應目前後端伺服主機負載情況。我們將所提出的排程演算法實作於 RedHat 7.0[4]版所附的 LVS project 套件中，我們主要是修改了 IPVS 模組以達到所需的功能。

本研究是設計以時間為基礎的排程演算法，主要是以時間為基礎來反應目前後端伺服主機負載情況，而反應後端伺服主機負載情況的方法可以有不同的方式。以下將介紹本研究所提出之方法，包含快速回應法、累積工作時間最少優先法、平均工作時間最少優先法、及累積未完成工作時間最少優先法。同時，我們亦實作後端伺服主機工作量負荷最少優先法，這個方法的實作，最主要是用來與我們所提出的以時間為基礎的排程演算法作為對照組之用，前 4 種排程法是在前端工作分配主機使用時間資訊來推估各後端伺服主機的負載情形，而此法是由前端工作分配主機需要定期去向後端伺服主機去要負載資訊來作排程的依據。

3.1、快速回應法

一般而言，快速回應（Fast Response）是以前端工作分配主機各發出一個封包給後端伺服主機群來測量回應的速度以反應目前負載情況，前端工作分配主機可依據各後端伺服主機的反應速度來作排程的決策準則。

我們把每一連線記錄中多加入一個時間戳記作為開始的時間 Ts，當連線結束之後，我們以目前時間 Te 減去開始時間 Ts 得到一個回應時間 Tr，並記錄回應時間，如此可以得到每一台後端伺服主機的回應時間，再依回應時間的長短來作為排程的準則。

```
Struct ip_masq {
        struct list_head m_list, s_list, d_list;
        atomic_t refcnt;
        struct timer_list timer;
        __u16                protocol;
        __u16                sport, dport, mport;
        __u32                saddr, daddr, maddr;
        struct ip_masq_seq out_seq, in_seq;
        struct ip_masq_app *app;
        void                *app_data;
        struct ip_masq      *control;
        atomic_t             n_control;
        unsigned             flags;
        unsigned long        timeout;
        unsigned             state;
        struct ip_masq_timeout_table
*timeout_table;
  #ifdef CONFIG_IP_MASQUERADE_VS
        struct ip_vs_dest *dest;
        atomic_t in_pkts;
  #endif
  unsigned long              begin_time;
  struct a_undo              *head,*tail;
};

****************************************

struct ip_vs_dest {
         struct list_head      n_list;
          __u32                 addr;
          __u16                 port;
          unsigned              flags;
          unsigned              masq_flags;
          atomic_t              activeconns;
          atomic_t              inactconns;
          atomic_t              refcnt;
          int                   weight;
         struct list_head      d_list;
         __u16                  protocol;
         __u32                  vaddr;
         __u16                  vport;
         __u32                  vfwmark;
          unsigned long     fast_response_time;
          unsigned long     atime;
          unsigned long     ajob;
          unsigned long     unit;
          unsigned int      threshold;
          atomic_t           current_threshold;
};
```

圖 1. struct ip_masq 及 struct ip_vs_dest

由於需要考量到後端伺服主機軟、硬體的差異性，因此我們設定一個臨界值，這一臨界值表示此伺服主機的最大連線數值，根據臨界值來表示伺服主機最大的負載量，所以在選擇一台後端伺服主機時，判斷該主機是否有超出其臨界值。若此後端伺服主機已超出其臨界值，則選擇其他的伺服主機來提供服務。

在設計以時間為基礎的演算法時，須考量排程演算法收集到時間資訊的時效性，必須每間隔一段時間來重新收集資訊一次，所以間隔時間的長短反應了所收集到的時間資訊是否能夠真正的反應後端伺服主機的負載情況。

以下介紹如何實作快速回應於 IPVS 模組中，包含 struct ip_vs_dest 及 struct ip_masq，如圖 1 所示，反白之處代表由我們外加的變數以供排程演算法之用， ip_masq 這個資料結構主要是用來記載使用者端與叢集式系統的連線記錄，我們在 ip_masq 資料結構中加入 unsigned long begin_time 的變數，它是一個時間戳記，用來記錄建立連線的啟始時間。另一資料結構 ip_vs_dest 主要是記載後端伺服主機的資訊，並且使用雙向串列連接每台後端伺服主機的資料，我們在 ip_vs_dest 資料結構中加入一個 unsigned long fast_response_time 的變數，以作為反應後端伺服主機之目前負載情況，把快速回應實作入 IPVS 模組中。而快速回應演算法之流程圖如圖 2 所示，當使用 IPVSADM 來新增後端伺服主機群時，IPVS 模組會把每台伺服主機的 fast_response_time 設定為零以作為初始化，當有一封包流入叢集式系統且此封包的旗標 syn 等於 1，且目的地埠為 HTTP 80 的服務埠時，IPVS 模組會建立新的連線記錄，並將資料結構 ip_masq 中的 begin_time 變數給予目前的系統時間（jiffies），作為連線起始時間，IPVS 模組會呼叫 ip_vs_schedule() 函式來排程，ip_vs_schedule()會循序讀取每一台後端伺服主機的資料結構 ip_vs_dest 的 fast_response_time 變數，並且從伺服主機群中找出最小 fast_response_time 且尚未超過臨界值的主機，作為選擇後端伺服主機的依據，再將該伺服主機的目前的連線數加 1。

當IPVS模組收到IP層所傳來的封包，若此封包的旗標rst等於 1 且目的地埠為HTTP 80埠時，IPVS模組會找出此封包當初所建立的連線記錄，並且使用目前系統時間jiffies變數減去連線記錄的資料結構ip_masq中的begin_time來得到此連線的回應時間，再將此回應時間更新該後端伺服主機的資料結構ip_vs_dest的fast_response_time變數，並且把該伺服主機的目前連線數減 1。此外，我們在IPVS模組中加入了一個更新資訊機制，因為當後端伺服主機的負載資訊過了一段時間後，負載資訊無法正確反應出真實負載情況，所以每一次間隔時間到了之後就必須重新計算一次。

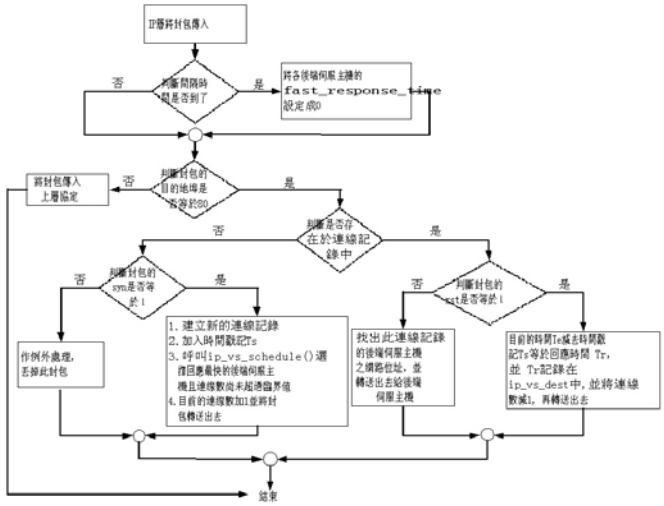

圖 2. 快速回應演算法之流程圖

3.2 累積工作時間最少優先法

累積工作時間最少優先的排程法的設計之觀念在於一段時間內，累積工作時間越多時，表示後端伺服主機的負載越重，反之，其累積工作時間越少時，表示後端伺服主機越有能力負擔新的服務請求，因此我們將累積工作時間最少優先法實作於IPVS模組內，其作法和快速回應相似。

如圖1所示，我們在資料結構ip_masq中，新增一個變數begin_time，用來記錄新增此連線記錄的時間，同時也提供計算要完成此一服務需求所花費的時間。另外在資料結構ip_vs_dest中，新增一個atime的變數，用來累積此後端伺服主機的工作時間，把ip_vs_schedule()修改成找尋累積工作時間最少的後端伺服主機且尚未超過臨界值的伺服主機，優先被挑出來服務。

如圖3所示，累積工作時間最少優先的流程與快速回應的流程相同，同時亦有設定間隔時間來更新時間資訊及設定臨界值的機制。每當建立一連線記錄時就記錄連線開始時間，當連線結束時，就會算出此連線所花費的時間，並累積至該連線記錄的後端伺服主機的累積工作時間atime變數中，當ip_vs_schedule()欲作排程時，會參考後端伺服主機群的atime變數，並找出最小的atime且尚未超過臨界值以作為最適的伺服主機。

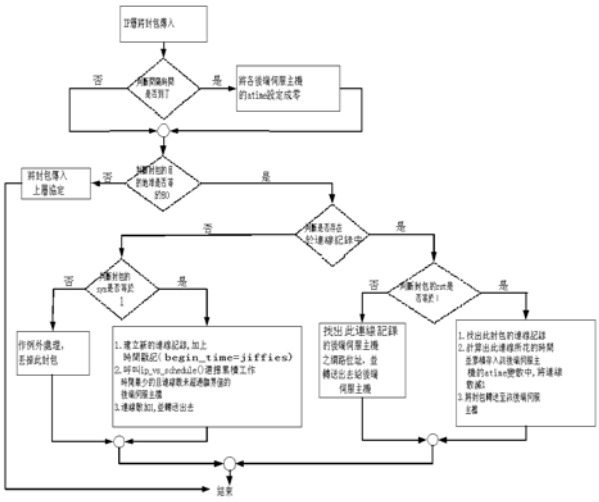

圖3. 累積工作時間最少優先之流程圖

3.3 平均工作時間最少優先法

平均工作時間最少優先是由累積工作時間最少優先演變而來的，此法先計算出累積工作時間及總工作量，其公式:累積工作時間除以總工作量等於平均工作時間，我們在圖1中的ip_vs_dest資料結構中加作了三個變數atime、ajob及unit，分別作為累積工作時間、累積工作量及平均工作時間之用途，在ip_masq中設有begin_time來記錄連線開始的時間，我們修改IPVS模組使其具有累積時間、累積工作量，並計算出平均工作時間之功能，再更新該後端伺服主機的結構變數ip_vs_dest中的unit，如此一來，當ip_vs_schedule()被呼叫時，ip_vs_schedule()可參考各後端伺服主機的平均工作時間來作排程之用。圖4說明其演算法之運作流程與累積工作時間最少優先法相似，不同的是增加了計算平均工作時間的功能。

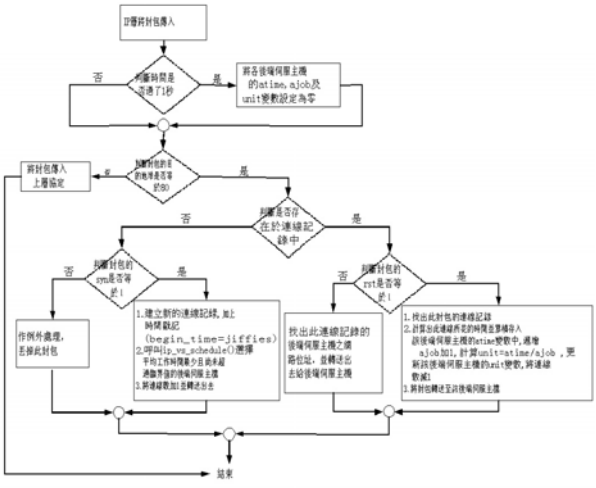

圖 4. 平均工作時間最少優先之流程圖

3.4 累積未完成工作時間最少優先法

累積未完成工作時間最少優先法的想法是基於後端伺服主機累積未完成的工作量越多時，其表示後端伺服主機的負載越重。舉例來說，當後端伺服主機A目前擁有三個HTML檔尚未作完，後端伺服主機B則有兩個圖形檔，因為圖形檔案的大小遠比文字檔大的多，所以我們不能使用以工作量來計算它，改採用計算時間方式來反應後端伺服主機的負載情況，所以計算出屬於該後端伺服主機的每一個尚未結束連線，從開始連線的時間到目前的時間，累積每一個連線時間。

將累積未完成工作時間最少優先法實作於IPVS模組的方式與前面所提的方法有些不同，由於IPVS模組中是以雜湊方式來記錄連線資料，IPVS模組是根據封包的IP標頭的資訊，如：protocol、source port、destination port、source address及destination address來找出連線記錄，所以無法依後端伺服主機的網路位址來找出所有的連線記錄，因此，為了要能收集到後端伺服主機的連線記錄之開始時間，我們增加了struct a_undo資料結構，如圖5所示，其中在資料結構a_undo中的link是用來連接連線記錄，再從連線記錄中新增一連線記錄的開始時間，並使用佇列（Queue）方式來維護此串列，所以如圖1所示，在資料結構ip_vs_dest中，新增加了兩個指標變數，分別為struct a_undo *head,*tail;作為佇列的新增與刪除之用，因為伺服主機在處理使用者的服務需求時都採先到先服務的方式，所以使用兩個指標分別指向佇列的頭尾，以加速佇列處理的速度。

```
struct a_undo
{
        struct ip_massq *link;
        struct a_undo   *head;
        struct a_undo   *tail;
};
```

圖 5 struct a_undo 結構

在圖 6 的流程圖中，當使用者端發出一個請求連線的封包且目的地埠為 HTTP 服務埠時，IPVS 模組會檢查封包的旗標狀態及目的地埠，並新增一連線記錄及配置一個資料結構 a_undo 變數來鏈結連線記錄，並在連線記錄中新增一時間戳記，此時 ip_vs_schedule()會從後端伺服主機群中找出累績未完成工作時間最少的伺服主機且該伺服主機的連線數尚未超過臨界值，當 IPVS 模組收到一個封包，且封包的旗標狀態為 rst=1 時及目的地埠為 80

時，IPVS 模組會根據封包的資料來找出連線記錄，並從連線記錄中找出後端伺服主機的資料，再至後端伺服主機的佇列中移除此一連線記錄。

由於計算累積未完成工作時間都在呼叫排程程式時，即時計算累積未完成工作時間，因此不使用間隔時間來清除時間作重計的動作。

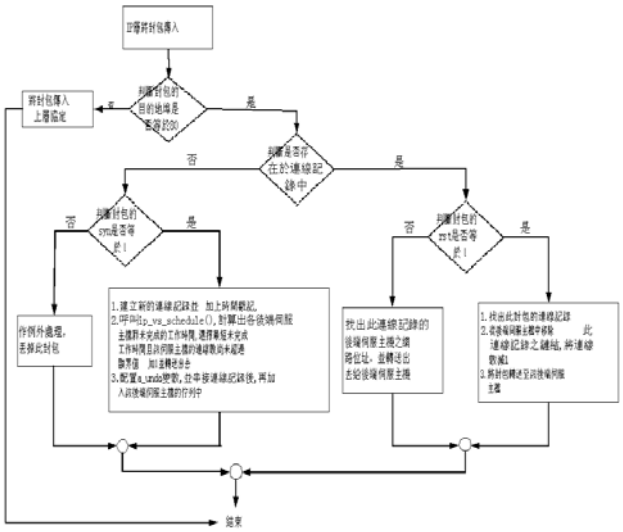

圖 6.累積未完成工作時間最少優先之流程圖

3.5 後端伺服主機工作量負荷最少優先法

後端伺服主機工作量負荷最少優先就是收集後端伺服主機的負載資訊來作為排程的依據。在 Linux 的/proc 目錄中存在許多有關於目前作業系統的狀態的檔案，我們選取二個檔案分別為 loadavg 及 stat[4]來作為瞭解伺服主機目前負載情況，其中自 loadavg 檔案可以得到 CPU 的負載資訊，而自 stat 檔案可以得到磁碟的 I/O 資訊。我們撰寫了一組 Client/Server 的網路程式來收集後端伺服主機的負載情形，並利用以下的公式來表示後端伺服主機目前負載的情況：

後端伺服主機目前負載＝目前 CPU 負載＋α*（磁碟目前讀寫總次數）

其中公式 α 在我們的實驗中所設定為 0.05，是由觀察實驗中後端伺服主機群所得到的數值，再由 ip_vs_schedule()從後端伺服主機群中，找出最小的負載之後端伺服主機來服務。

四、實驗與討論

本實驗叢集式系統之硬體環境考量到同質性及異質性的叢集式系統架構，所以我們建構了兩種不同的硬體環境，如表 1 及表 2 所示，並將整個研究實作於內部私有網路中，以便免不必要的外部因素干擾實驗的公平與正確性。

表 1. 異質性叢集式系統實驗之硬體架構

硬體規格 / 電腦用途	CPU Intel (Hz)	RAM (MB)	DISK IDE(rpm)	Ethernet (100Mbps)
前端工作分配主機	Pentium IV 2.0G	DDR 256MB	7200 40GB	3com-3c905c
後端伺服主機 1	Pentium III 800	SDRAM 128MB	7200 40GB	Realtak 8139
後端伺服主機 2	Pentium III 500	SDRAM 128MB	5400 20GB	SMC1211TX
後端伺服主機 3	Pentium III 300	SDRAM 256MB	5400 20GB	Realtak 8139
後端伺服主機 4	Pentium III 300	SDRAM 256MB	5400 20GB	SMC1211TX
控制測試端電腦	Pentium IV 2.4G	DDR 512MB	7200 80GB	Realtak 8139
測試端電腦 1	Pentium IV 2.4G	DDR 256MB	7200 80GB	Realtak 8139
測試端電腦 2	Pentium IV 2.4G	DDR 256MB	7200 80GB	Realtak 8139

表 2. 同質性叢集式系統實驗硬體架構

硬體規格 / 電腦用途	CPU Intel (MHz)	RAM (MB)	DISK IDE(rpm)	Ethernet (100Mbps)
前端工作分配主機	Pentium IV 2.0G	DDR 256MB	7200 40GB	3com-3c905c
後端伺服主機 1	Pentium IV 2.4G	DDR 512MB	7200 80GB	Realtak 8139
後端伺服主機 2	Pentium IV 2.4G	DDR 256MB	7200 80GB	Realtak 8139
後端伺服主機 3	Pentium IV 2.4G	DDR 256MB	7200 80GB	Realtak 8139
控制測試端電腦	Pentium IV 2.4G	DDR 512MB	7200 40GB	Accton EN120TF
測試端電腦 1	PIV Celeron 1.7G	SDRAM 384MB	7200 40GB	sis 900
測試端電腦 2	PIII Celeron 1.0G	DDR 256MB	7200 40GB	Realtak 8139

在前端工作分配主機中，我們使用了 Redhat 7.0 版所附的 LVS 套件以及使用了 IPVSadm-1.15 版的管理程式來建構前端工作分配主機，並使用 IPVSADM 來設定叢集式系統初始化設定及啟動 IPVS 模組運作，且使用 IP Tunneling 轉送機制來轉送封包給後端伺服主機，在後端伺服主機方面，我們必須設定作業系統能夠支援 IP Tunneling 協定，如此一來，後端伺服主機才能接收來自前端工作分配主機轉送的封包，並且在後端伺服主機中安裝 Apache 伺服軟體，來提供網頁存取的服務。另外在實驗中我們選用公正且免費的測效軟體，分別為 AutoBench[2]及 WebBench[5]。

在 LVS project 所提供的八種排程模組，因為受限於測試端的電腦數量有限及測效軟體只能針對一個網站來作測效，因此無法使用 LBLC、LBLCR、Destination Hashing 及 Source Hashing 等排程法。在 LVS 測效報告[6]中顯示 RR、WRR、LC 及 WLC 排程法在異質性的後端伺服主機中，以 WLC 的效能為最佳，因此以 WLC 排程法來作為我們異質性實驗中的對照組。另一方面，測試 LVS 排程法後得到以 LC 的排程法為最佳，而因此同質性的實驗中 LC 比 WLC 的效能好是因為 WLC 多了處理 Weight 的程序。我們各自以 WLC 及 LC 排程法來作為時間為基礎的排程法作為比較之用。

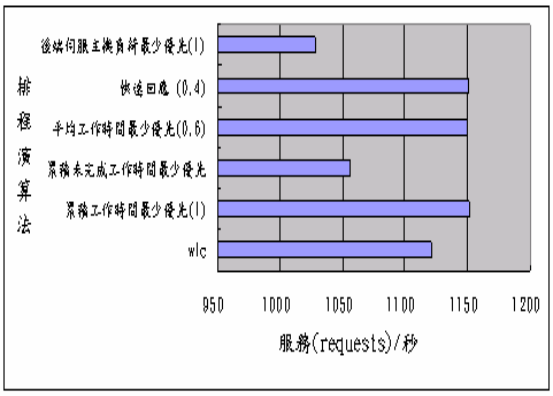

圖 7.　以時間為基礎的排程法與 WLC 排程法之比較圖

4.1 異質性後端伺服主機的實驗

為了能充分反應伺服主機的目前工作負載，我們加入每隔一段時間就要重新計算工作時間的機制，所以我們先個別找出較佳的間隔時間的設定，由實驗中得知累積工作時間最少優先法之間隔設定成 1 秒、平均工作時間最少優先法設定間隔為 0.6 秒較佳，而快速回應法設定間隔為 0.4 秒，及後端伺服主機工作量負荷最少優先法設定間隔為 1 秒為較佳的時間，其中累積未完成工作時間最少優先法中並沒有間隔時間的機制，因為排程程式會即時地計算各台伺服主機未完成的工作時間，所以不需要找間隔時間。實驗結果如圖 7、8 所示，其中以快速回應法與平均工作時間最少優先法為最佳，其次是累積工作時間最少優先，這三種排程法皆可以有效地提高叢集式系統的效率。

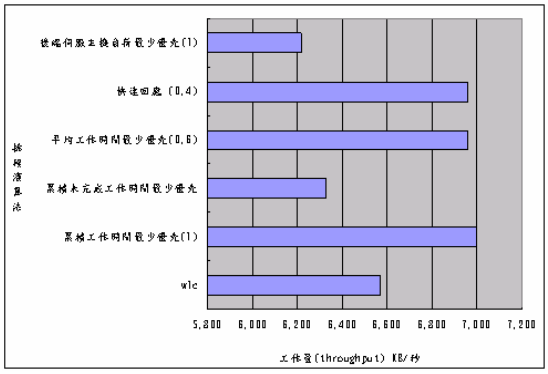

圖 8.　以時間為基礎的排程法與 WLC 排程法之比較圖

4.2 同質性後端伺服主機的實驗

因為後端伺服主機為同質性的實驗環境與異質性的實驗環境不同，所以我們必須再一次重覆地找出間隔時間，由實驗中可得知累積工作時間最少優先法之間隔設定成 1 秒較佳、平均工作時間最少優先法設定間隔為 0.6 秒較佳、快速回應法設定間隔為 0.8 秒，及後端伺服主機工作量負荷最少優先法設定間隔為 1 秒為較佳的時間。實驗結果如圖 9、10 所示，以快速回應法與平均工作時間最少優先法為最佳。

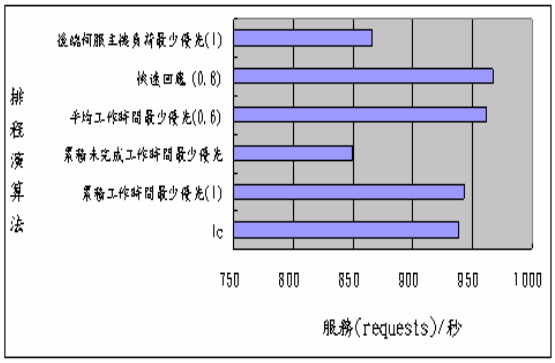

圖 9.　以時間為基礎的排程法與 LC 排程法之比較圖

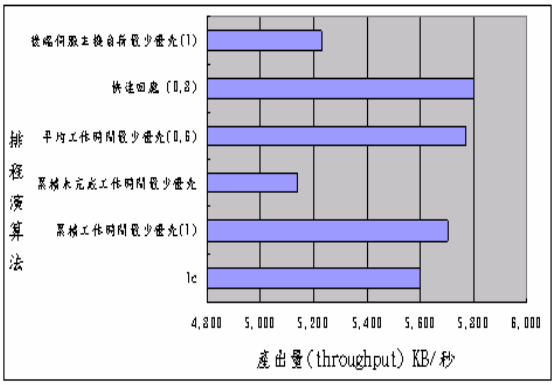

圖 10. 以時間為基礎的排程法與 WLC 排程法之比較圖

五、結論

在以時間為基礎的排程演算中，我們找到了三種有助於平衡負載的排程，實驗的結果顯示，以快速回應法、平均工作時間最少優先法及累積工作時間最少優先法可以提昇叢集式系統的效能。由異質性叢集式系統實驗中，這三種排程法有較顯著的提昇異質性的叢集式系統的效能，能夠將負載依照各後端伺服主機的處理能力來分散。在同質性叢集式系統實驗中，只有快速回應法及平均工作時間最少優先法可以大幅度改善叢集式系統的效能，累積工作時間最少優先法也能提昇其效能，但是都較 LVS 中原有的的 LC 與 WLC 的排程法好。然而，以收集後端伺服主機的負荷法來作排程因為是透過後端伺服主機發送負載資訊的封包，來告知前端工作分配主機目前後端伺服主機的負載情況，但是前端工作分配主機獲取負載資訊必須使得後端伺服主機須耗費 CPU 資源來發送封包告知負載情況，且耗費網路頻寬，因此，使得後端伺服主機的效能降低而影響整個叢集式系統的效能。

在我們的實驗中，我們是採用 WebBench 5.0 及所附的測效檔來作測試，未來將以實際的流量來作測試，且本次的實驗是以一個網站來作測試，其他種類的服務是否也能適用以時間為機制來作排程的演算法，這也是我們要進一步研究的課題。

六、參考文獻

[1] 黃銘毅，叢集式系統以時間為基礎之排程法與以瞭解服務內容之需求分配平台的研究與製作，國立暨南國際大學資訊管理學院研究所碩士論文，中華民國九十二年六月。

[2] Autobench Website, http://www.xenoclast.org/autobench/.

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

[4] RedHat Linux Website, http://www.redhat.com/.

[5] WebBench Benchmark Website, http://www.veritest.com/benchmarks/webbench/home.asp?visitor.

[6] Mike keefe and Patrick O'Rourke, "Performance Evaluation of Linux Virtual Server," April 6, 2001.