行政院國家科學委員會補助專題研究計畫成果報告

※※※※※※※※※※※※※※※※※※※※※※※※※
※　　　　　　　　　　　　　　　　　　　　　　　　※
※　　　Web Switch 系統的設計與製作(1/2)　　　※
※　　　　　　　　　　　　　　　　　　　　　　　　※
※※※※※※※※※※※※※※※※※※※※※※※※※

計畫類別：■個別型計畫　　□整合型計畫

計畫編號：NSC 90－2213－E－009－089－

執行期間：90 年 08 月 01 日至 91 年 07 月 31 日

計畫主持人：李程輝　交通大學電信系 教授

共同主持人：

本成果報告包括以下應繳交之附件：

□赴國外出差或研習心得報告一份

□赴大陸地區出差或研習心得報告一份

□出席國際學術會議心得報告及發表之論文各一份

□國際合作研究計畫國外研究報告書一份

執行單位：國立交通大學電信系

中　華　民　國　91　年　5　月 22 日

行政院國家科學委員會專題研究計畫成果報告

計畫編號：NSC 90-2213-E-009-089-
執行期限：90 年 8 月 1 日至 91 年 7 月 31 日
主持人：李程輝　交通大學電信系 教授
共同主持人：
計畫參與人員：

一、中文摘要

本年度計畫主要研究 Web 交換機系統的核心技術：多欄位高速封包分類與負載平衡機制。多欄位高速封包分類演算法的設計主要是運用在 URL 轉送表(URL Forwarding Table)的搜尋機制，以有效降低 URL 轉送表的搜尋時間，避免形成 Web 交換機的效能瓶頸。為了達到有效快速的封包分類，我們首先發展了一個兩個欄位的封包分類方法，稱之為 Double Binary Search 演算法，其最差情況的搜尋時間複雜度為 2log$W$，其中 W 為欄位最大的位元數；依此方法延伸至 K 個欄位的封包分類，其搜尋時間複雜度為 Klog$W$。經由實際收集到的路由表格來隨機產生所需的兩個欄位之樣本轉送表格，再透過軟體模擬分析比較，結果顯示在 W = 32 位元下，我們所發展的演算法，其最差情況的搜尋時間效能均優於目前文獻上已存在的機制(如 Line Search，Rectangle Search，及 Nested Binary Search 等演算法)三倍至十六倍之間；而所需平均搜尋速度則比其他機制快上 3.2 倍至 23.8 倍之間。

負載平衡機制的研發主要是要將系統負載根據網路訊務流量及伺服器的狀況，平衡的分散，以增加連線要求的處理速度及數量，並降低網路延遲時間及服務要求的回應時間，來達到更高的效率。於此計畫中，我們設計了一個網頁文件分配器來達到負載平衡的效果，經由軟體模擬分析來評估，在輕載、中間負載量、重載的情況下，所設計的機制對要求的反應時間及伺服器的使用率等之效能；並且與其他既存的方法比較。結果顯示，我們所提供的負載平衡機制在這些效能上優於其他既存的方法。

關鍵詞：網站伺服器，負載平衡，多欄位封包分類，URL 轉送表搜尋。

Abstract

In this project, we investigated two core technologies for developing web switching system: multi-field packet classification and load balancing mechanism. The multi-field packet classification algorithm is used to support the search engine in the URL forwarding table which the lookup time is an important performance issue in developing a Web switch. To achieve fast lookup, we proposed a so-called Double Binary Search algorithm for two-field packet classification. The worst-case lookup time of such an algorithm is 2log$W$, where W is the maximum number of bits for fields. Further, it is easy to extend our proposed scheme to support K-field packet classification and the corresponding time complexity will be Klog$W$. Through experiments on two-field routing tables, we found that, for W = 32 bits, our scheme outperforms other existing mechanisms with the worst-case lookup speed from 3 times to 16 times. For the average-case search speed, our scheme is faster than others from 3.2 times to 23.8 times.

Designing a load balancing scheme for a cluster of Web servers aims to equally distribute the load across the servers and to select a best server that can give a satisfactory user-perceived retrieval delays and minimize the network traffic load. In this project, we have developed a so-called adaptive partitioning dispatcher in an attempt to meet these requirements. We further perform an experimental study on response time and server utilization for our proposed scheme and other existing mechanisms under

conditions of light, medium, and heavy workloads. The results showed that our scheme is superior to others in these performance issues.

Keywords: Web Switch, Load balancing, Multi-field packet classification, URL forwarding table lookup.

二、計畫緣由與目的

網際網路流量的指數倍數成長主要是源於全球資訊網（WWW）普及化的蜂擁而至，若此高成長的訊務需求無相對應的解決方案來配合，終將導致過高的網路流量及無法接受的服務反應時間。Web 交換機系統就應蘊而生以有效節省網路頻寬的浪費、平衡伺服器流量負載、降低網路延遲與提供較高的內容可得性（availability）。在設計 Web 交換機系統中，URL 轉送表的搜尋引擎是重要的研發技術之一，主要的功能在於能夠有效的、快速的發送使用者的要求至目的地快速緩衝貯存區系統並快速取出資料回應給使用者。其搜尋引擎牽涉到多欄位封包分類技術，目前存在的演算法包括有 Tree-based，CAM-based，hash-based 等方法。但是這些方法不是最差情形搜尋時間過長就是採用成本過高的 CAM(Content Addressable Memory)來降低搜尋時間而顯得不適用。本研究在避免採用 CAM 的條件下，設計出低時間複雜度的多欄位搜尋演算法，以提升 Web 交換機的執行效能。

負載平衡機制是設計 Web 交換機系統另一個重要的考量技術，主要是運用於一群 Web 伺服器上，以期平均分配流量負載給各個伺服器，且所選擇回應給使用者要求的伺服器能夠滿足使用者意識到的取回延遲與最小化網路流量負載。目前存在的方法有 static partitioning，locality-based，weighted round robin，pick-k 等機制。在不同的 workload 情形下，這些機制突顯出其效能的優缺點，如使用者意識到的取回延遲小或伺服器的使用率高等，經由深入研究評估這些機制的優缺點後，於本計畫我們提出一個調整式的負載平衡機制。經由軟體模擬分析在輕載、中間負載量、重載的情況下，我們所設計的機制對使用者要求的反應時間及伺服器的使用率等效能均優於既存的方法。此外，我們將會根據本年度所探討影響 Web 交換機與網站效能的重要因素，作為往後深入研究與設計的重要參考。

三、研究方法與成果

I. 多欄位封包分類器 — Double Binary Search Algorithm

為了簡化說明，我們以兩個欄位的封包分類演算法為例，依序解說雜湊表格的建立、搜尋程序、演算法效能複雜度分析與模擬分析結果。依此機制可以很容易的延伸至多欄位封包分類器的應用。

（一）雜湊表格的建立

- Conflict check

在轉送表格裡，每個 entry(稱為 filter)之各個欄位均是以 prefix 來表示。而搜尋的結果可能有兩個以上的 filter 同時 match 所需查詢的封包位址，但卻無法分辨哪一個 filter 是最適當的配對 filter 時，這些 filters 稱為 conflicting filters。例如，F_1 = (11011*, 0010*)與 F_2 = (110*, 001011*)是兩個 conflict 的 filter。在建立雜湊表格前，我們先執行 conflict check，找出所有 conflict 的 filter-pair，並加入一些 resolution filter 來解決此 conflict 問題。Resolution filter 的產生是在兩個 conflict filters 中，針對各個欄位，選取 filter 中於此欄位上具有較長的 prefix 來當作此欄位的 prefix。例如，上例中的 F_1 與 F_2，其 resolution filter 為 $F3$ = (F_1[1], F_2[2]) = (11011*, 001011*)。經由此步驟，所形成的就是無衝突的 filter 集合。此外，我們針對任兩個 filters F_a 與 F_b 滿足下列條件也加上 resolution filter 以加速搜尋程序的速度；此條件為：若 F_a 的第一個欄位之 prefix 為 F_b 第一個欄位之 prefix 的 prefix，且 F_a 的第二個欄位之 prefix 與 F_b 第二個欄位之 prefix 為 disjoint 時，增加一個 resolution filter F_{ab} = (F_b[1], F_a[2])。

- filter classification

根據各個 filter 的各個欄位之 prefix 長分類，將相同的 prefix 長度的 filter 分在同一個雜湊表格；例如 F_4 = (1011*, 10*)與 F_5

= (1100*, 00*)屬於雜湊表格 $T(4, 2)$，而 F_6 = (100*, 100*)位於表格 $T(3, 3)$。顯然的，兩個欄位的 filter 集合經由此分類方式將形成兩個維度的雜湊表格空間，即 $(W+1) \times (W+1)$；其中第 i 個維度對應第 i 個欄位。在雜湊表格 $T(i, j)$中進行搜尋的方式是將欲查詢封包第一個欄位之 IP 位址選 i 個位元，而第二個欄位之位址取 j 個位元，然後執行一次的雜湊運算就可以找出與之配對的 filter(如果存在的話)。為了方便以下說明起見，二維雜湊表格空間的第 k 列定義為表格 $T(m, k)$ for $0 \le m \le W$；同理，二維雜湊表格空間的第 l 行定義為表格 $T(l, n)$ for $0 \le n \le W$。

■ Marker and Precomputation

為了在每一列中都能夠執行二元搜尋的效能，我們將位於此列上的雜湊表格內之每個 filter 建立其所屬的 marker 以引導搜尋程序。為了預防在搜尋過程中 back tracking 的發生，我們針對每個所建立的 marker 做最佳配對 filter 的 precomputation。Marker 及 precomputation 機制採用類似於 binary search on prefix lengths [5]所提供的方法。此外，我們將位於每一行(除了第 0 行外)的表格內之 filter 建立新的 projection marker 於第一行中，以利於執行第一個欄位之最長長度配對 prefix 的搜尋；例如，位於雜湊表格 $T(5, 4)$ 內之 filter F_1 = (11011*, 0010*)將產生一個 projection marker (11011*, *)於表格 $T(5, 0)$ 裡。

（二）搜尋程序

1）根據封包之第一個欄位的 IP 位址，以二元搜尋法找尋最長長度配對的 prefix；假設搜尋的結果位於表格 $T(i, 0)$。

2）於第 i 列上，以二元搜尋方式搜尋與此封包兩個欄位 IP 位址配對之 filter。若存在此 filter，則搜尋程序將會回傳其轉送表之目的地位址；反之，回傳一個 “No match”的訊息。

（三）演算法之複雜度分析

由上述搜尋程序得知，我們最多需要 $\log W$ 的雜湊偵測來搜尋第一個欄位最長長度配對的 prefix，而後在此列上最多需要 $\log W$ 的雜湊偵測來找尋最佳配對的 filter。所以在最差情況下總共需要 $2\log W$ 的搜尋時間以決定最佳配對的 filter。

（四）實驗模擬分析

於此，我們從 IPMA 計畫[11]收集了一些實際的 backbone 路由表格，並由此隨機取樣來產生第二個欄位之 prefix，依此方式來產生所需的兩個欄位的 filter 集合。這些表格包括有 AADS、MAE、PAIX 與 MCVAX，如表一所示。為了有效評估最差情況的搜尋時間，我們以每個 filter 之各個欄位 prefix 的左端點為基準，將第一個欄位形成的左端點集合與第二個欄位所形成的左端點集合做所有可能的排列組合，依此製造所需的測試封包的 IP 位址。實驗模擬分析評估所設計的機制外，並比較了 Line Search，Rectangle Search，與 Nested Binary Search 演算法的效能。各個演算法所需最大與平均的雜湊次數顯示於表二中；而各個機制的記憶體需求結果則顯示於表三中。結果顯示，我們所設計的演算法在搜尋時間效能方面遠優於其他的機制，所需的最大雜湊次數遠快於其他機制約三倍到十六倍，而所需的平均雜湊次數則比其他機制快約 3.2 倍到 23.8 倍；再者，我們的演算法僅僅比最少記憶體需求的 Line Search 機制多消耗約兩倍的記憶體，但相對提高了搜尋速度十六倍以上。

II. 負載平衡機制的設計

本計畫主要針對叢集式網站系統 (cluster-based web system) 研究如何有效分配連線要求與處理資料訊務於這些網站上的負載平衡機制，稱之為 Adaptive partitioning algorithm。設計得架構如圖一所示，於系統前端置放一個 layer-7 dispatcher，以負責連線要求的發送分配。此 dispatcher 屬於 layer-7 content-aware 的 dispatcher，其特性是：當決定出 content 的所在位址，dispatcher 將會把 request 發送至擁有此 content 的伺服器。以伺服器的觀點而言，任何一個 content 就代表一個負載量(load)。藉由分配 content (其中 content 的型態可根據靜態/動態與普及程度來區分)，便可將各個伺服器上流通的負載量分配平均，其中我們定義負載平均為平衡化各個伺服器上的存取率(access rate)與各個

伺服器上被存取的 content 檔案大小)。由統計資料得知,90%的連線要求是叢聚在 10%的 content 存取，而此 10%的 content 檔案大小大都不超過 64KB，因此，我們可以依據 content 的普及程度(popularity)來分配，將前 10% popularity 之 content 複製到每一台伺服器上，而剩下的 90% content 再平均分配到所有的伺服器上。關於各個 content 的存取率，我們可經由分析伺服器上的 log files 來獲得，而靜態與動態 request 則分別分配的。此外，我們定義了兩個 threshold 值，稱之為 Thigh 與 Tlow，用來偵測伺服器是否過載(overloaded)或者過低的使用率(underutilized)，來調整連線要求的分配，機動式的調整負載平衡機制。

於此，我們以軟體模擬分析評估所設計的機制並與其他負載平衡演算法比較。實驗模擬的架構如圖二所示，參與比較的演算法包括有 static partitioning、Weighted Round Robin(WRR)、locality-based 等方法；其中 static partitioning 和 locality-based 演算法屬於 content-aware 的 dispatcher，而 WRR 是 content-blind 的機制。分別在三種不同的 workload(light、medium、與 heavy)的情形下測量各個機制的 cumulative frequency of response time，如圖三、四、與五所示。結果顯示我們所提供的方法，對 request 的 response time 均比其他機制好。對於伺服器的使用效能，我們深入探討 static partitioning 與 adaptive partitioning 機制，實驗結果顯示我們所提的演算法可以有效的提升伺服器的使用率，如圖六所示。

四、結論

本年度計畫設計了多欄位高速封包分類演算法(稱之為 Double Binary Search algorithm)來支援 URL 轉送表的搜尋機制，以有效降低 URL 轉送表的搜尋時間，避免形成 Web 交換機的效能瓶頸。Double Binary Search 演算法於兩個欄位之封包分類的最差情況搜尋時間為 $2\log W$，其中 W 為欄位最大的位元數；將此機制延伸至 K 個欄位的封包分類，其搜尋時間複雜度將為 $K\log W$。經由軟體模擬分析其效能並與其他機制(如 Line Search，Rectangle Search，及 Nested Binary Search 等機制)比較，我們得到，在兩個欄位與 $W = 32$ 位元下，我們所發展的 Double Binary Search 演算法所需要的最大雜湊偵測次數遠小於其他機制約三倍至十六倍之間；而所需平均搜尋速度則比其他機制快上 3.2 倍至 23.8 倍之間；且僅僅消耗適中的記憶體量。

此外，我們也設計了一個網頁文件分配器來達到負載平衡的效果，以增加連線要求的處理速度及數量，並降低網路延遲時間及服務要求的回應時間。並經由軟體模擬與效能分析來評估各種不同負載平衡機制於不同的 workload 的情形下，對 request 的 response time 與伺服器的使用率。結果顯示我們所提供的機制優於其他演算法，並且我們發現設計負載平衡機制需要考慮服務的異質性才能因應未來網站訊務的變化。此外，我們將會根據本年度所探討影響 Web 交換機與網站效能的重要因素，作為往後深入研究與設計的重要參考。

五、參考文獻

[1]. D. Rosu, A. Iyengar, and D. Dias, "Hint-based Acceleration of Web Proxy Caches," in *Proc. of Internet Performance, Computing, and Communications Conference, IPCCC'00*, 2000.

[2]. Z. Genova and K.J. Christensen, "Challenges in URL Switching for Implementing Globally Distributed Web Sites," in *Proc. of International Workshops on Parallel Processing*, 2000.

[3]. B. Scott Micheal, K. Nikoloudakis, P. Reiher, L. Zhang, "URL Forwarding and Compression in Adaptive Web Caching," in *IEEE GLOBECOME'00*, 2000.

[4]. S. Nadimpalli and S. Majumdar, "Techniques for Achieving High Performance Web Servers," in *Proc. of Parallel Processing*, 2000.

[5]. M. Waldvogel, G. Varghese, J. Turner, and B. Plattner, "Scalable High Speed IP Routing Table Lookups," in *Proc. of ACM SIGCOMM'97*, Sept. 1997, pp. 25-36.

[6]. M. A. Ruiz-Sanchez, E. W. Biersack, and W. Dabbous, "Survey and Taxonomy of IP Address Lookup Algorithms," in *IEEE Network Magazine*, March/April 2001, pp.

8-23.

[7]. V. Srinivasan, S. Suri, and G. Varghese, "Packet Classification Using Tuple Space Search," in *ACM Computer Communication Review*, 1999.

[8]. M. Waldvogel, "Multi-Dimensional Prefix Matching using Line Search," in *Proc. of IEEE LCN'00*, Nov. 2000, pp. 200-207.

[9]. P. Warkhede, S. Suri, and G. Varghese, "Fast Packet Classification for Two-Dimensional Conflict-Free Filters," in *Proc. of IEEE INFOCOM'01*, 2001.

[10]. P. Gupta and N. McKeown, "Algorithms for Packet Classification," in *IEEE Network Magazine*, March/April 2001, pp. 24-32.

[11]. Merit Networks Inc., "Internet Performance Measurement and Analysis (IPMA) Statistics and Daily Reports," *IMPA Project*, http://www.merit.edu/ipma/routing_table/

[12]. H. Bryhni, E. Klovning, and O. Kure, "A Comparison of Load Balancing Techniques for Scalable Web Servers," in *IEEE Network*, July /August 2000, pp. 58-64.

[13]. V. Cardellini, M. Colajanni, and P.-S. Yu, "Dynamic Load Balancing on Web-Server Systems," in *Proc. Of IEEE Internet Computing*, May/June 1999, pp. 28-39.

[14]. E. Casalicchio and M. Colajanni, "A Client-Aware Dispatching Algorithm for Web Clusters Providing Multiple Services," in *Proc. Of 10th Int'l World Wide Web Conference*, Hong Kong, May 2001, pp. 535-544.

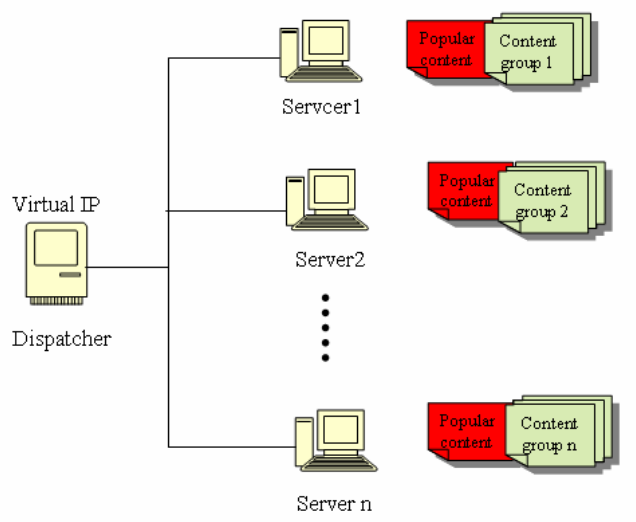
[15]. L. Cherkasova, "FLEX: load balancing and management strategy for scalable Web hosting service," in Proc. Of ISCC 2000, pp. 8 –13.

[16]. A. Paul, Wu-Chi Feng, D.K. Panda and P. Sadayappan, "Balancing Web Server Load for Adaptable Video Distribution," in *Proc. of Parallel Processing*, 2000.

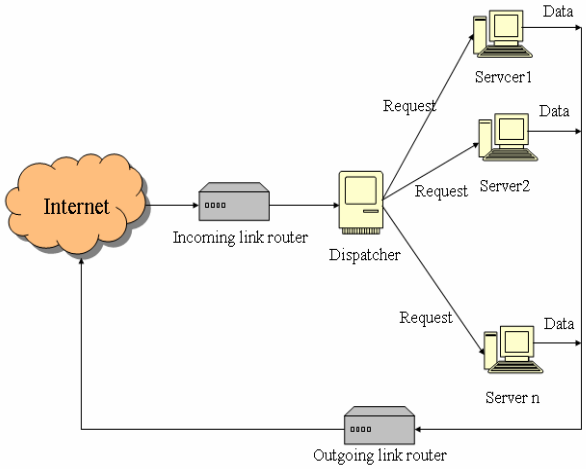
[17]. V. Cardellini, M. Colajanni, and P.S. Yu, "Geographic Load Balancing for Scalable Distributed Web Systems," in *Proc. of Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, 2000.

Table I. The sizes of the random filter sets.

	Sample filter set			
	PAIX	AADS	MAE	MCVAX
Number of filters in conflict free filter set ***F***	14538	28567	30002	41509
Number of filters in extended filter set ***F****	19141	37285	40806	50347



圖一、Cluster-based web system and Dispatcher.



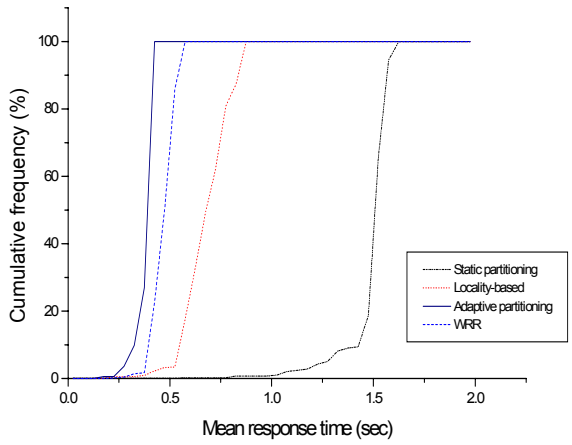
圖二、實驗模擬架構。

Table II. Comparison of average and maximum number of probes required in each algorithm for random filter sets.

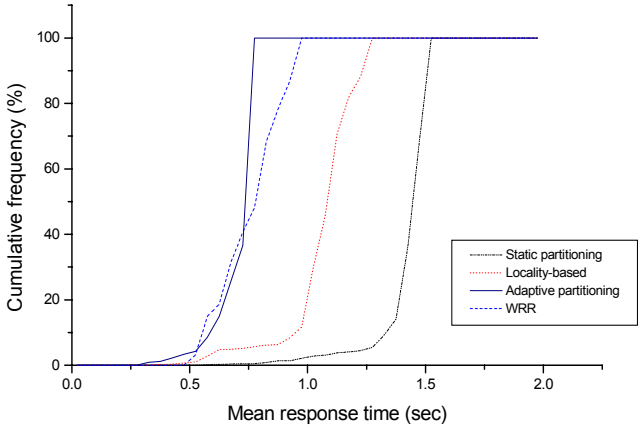
Scheme	PAIX		AADS		MAE		MCVAX	
	Max.	Average	Max.	Average	Max.	Average	Max.	Average
Line Search	192	189.19	192	188.95	191	188.93	192	191.04
Rectangle Search	65	54.19	65	55.16	65	54.25	65	63.46
Nested Binary Search	36	35.97	36	35.98	36	35.98	36	35.99
Double Binary Search	12	8.15	12	8.03	11	8.11	12	8.55

TABLE III. Comparison of memory storage $\tilde{N}$ required in each algorithm.

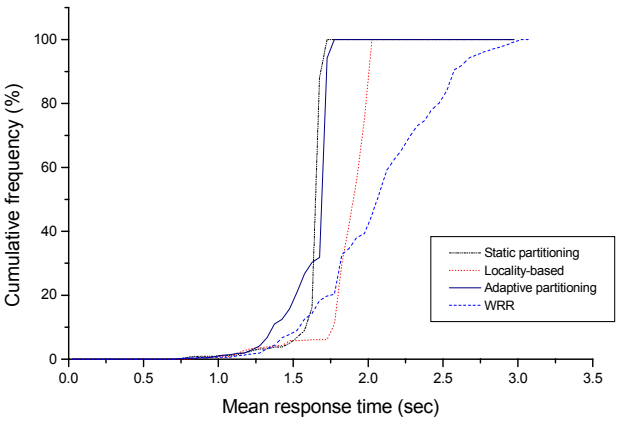
Scheme	PAIX		AADS		MAE		MCVAX	
	$\tilde{N}$	$\tilde{N}/N$	$\tilde{N}$	$\tilde{N}/N$	$\tilde{N}$	$\tilde{N}/N$	$\tilde{N}$	$\tilde{N}/N$
Line Search	35309	2.430	66698	2.335	69081	2.303	175788	4.235
Rectangle Search	333049	22.918	655731	22.961	684403	22.814	1179060	28.408
Nested Binary Search	85199	5.863	155926	5.460	159256	5.309	745694	17.967
Double Binary Search	68389	4.706	127323	4.458	136272	4.542	350988	8.457



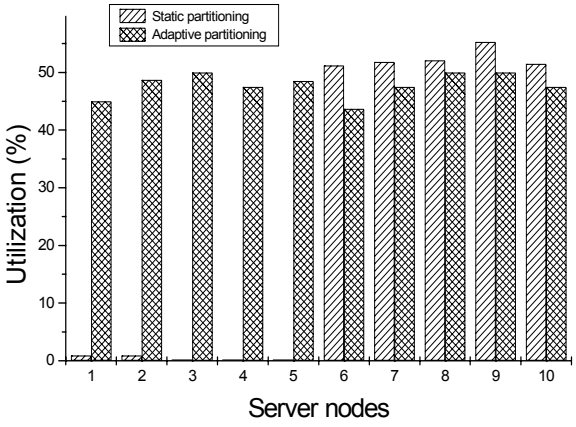
圖三、Cumulative frequency of mean response time under light workload condition.



圖四、Cumulative frequency of mean response time in medium workload condition.



圖五、Cumulative frequency of mean response time in heavy workload condition



圖六、Server utilization of static partitioning and adaptive partitioning under heavy workload condition.