On-demand Resource Allocation in Active Networks

Chien-Wen Chen, Fu-Fan Yang and Cheng-Zen Yang
Department of Computer Engineering and Science
Yuan Ze University
Chungli, Taiwan, R.O.C.
E-mail: {antony,jally,czyang}@syslab.cse.yzu.edu.tw

摘要

主動式網路(active network)中的主動式節點(active node)不僅具有傳遞封包的功能，而且還有執行使用者程式的能力，所以主動式網路能快速佈建新的服務和應用。主動式節點在執行程式時必須消耗節點上的資源，因此在主動式節點上有效的分配資源來防止不正常的資源消耗及提昇節點效能(performance)是一重要的研究課題。本論文提出了依需求資源分配(on-demand resource allocation)演算法來滿足不同 flow 的資源需求特性，使得主動式節點有最大效能，此演算法亦考慮了服務品質(Quality of Service, QoS)課題，除了提供設定優先權的機制外，亦限制每個 flow 被分配資源的比率上限；依三種不同 flow 對資源需求的特性，本論文設計了三種實驗環境來比較均等資源分配、公平資源分配及本論文所提出的依需求資源分配演算法，結果顯示依需求資源分配演算法在三種實驗環境下都有比較好的效能。

一、簡介

隨著網際網路(Internet)的快速發展，使用者不斷的要求網路上能提供各種新的服務來滿足其需求。在現今大部分的網路服務所使用的網路通訊，其通訊協定(protocol)除了根據各式各樣現存的標準外，若使用者想要自訂一套通訊協定來符合新服務的需求，則必需經過各方的冗長討論及繁複的制定過程才有可能完成，如此一來在時效性上則無法符合使用者的需求，因此發展新的下一代網路架構來解決此項問題有其強烈的需求，主動式網路(active network)是目前各研究機構討論最熱門的下一代網路架構。美國國防部高等研究計劃局(Defence Advanced Research Projects Agency, DARPA)的研究社群在 1994 年提出主動式網路(active network)的概念，在主動式網路的路由器(router)或交換器(switch)稱之為主動式節點(active node)，不僅具有原本封包傳遞(packet forwarding)的功能而且還有執行程式計算的能力，因此使用者可將自訂的新服務佈建在網路中的節點上。

主動式網路的主動式節點軟體架構如圖 1 所示[7]，在最底層為節點作業系統(NodeOS)，負責多工處理通過節點的串流(flow)及管理各 flow 所使用的通訊(bandwidth, bw)、記憶體(memory)及計算(CPU)等資源，而上層則分別為執行環境(Execution Environments, EEs)及主動式應用程式(Active Applications, AAs)， EE 負責提供一個執行環境給不同的程式碼，其中針對個別的 AA 定義了特別的程式模型(programming model)。

主動式網路目前遭遇最大的瓶頸為效能(performance)問題[2, 4, 7, 8, 11]，因為在主動式節點上可執行許多的 AAs，當許多的 AAs 在節點上同時執行時，由於節點上的資源不敷使用或無法有效的分配，使得每個 AAs 的效能都被降低，此外進入節點的每個 flow 由不同的來源產生，每個 flow 皆需要節點的 CPU、memory 及 bw 等資源來處理，假如有一不當行為或惡意攻擊的 flow 消耗大量的節點資源，則將會影響其他正常 flow 的效能，因此在主動式節點上有效的資源分配(resources allocation)來防止不正常的資源消耗及提昇節點的效能是一重要的研究課題。在以往相

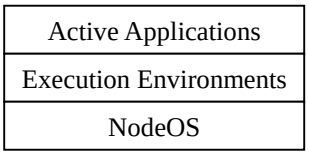

圖 1 主動式網路的主動式節點軟體架構(取自[7])

關的研究當中已指出了主動式網路中資源分配的重要性，也提出了一些資源分配或管理的設計原則，但是對於同一 flow 中對不同資源需求有極大差異的情形或不同 flow 間對資源需求有極大差異的情形並沒有考慮[2, 3, 5, 8, 10, 11, 12]，本論文提出一簡單明確的資源分配演算法來應用於主動式節點上，使得主動式節點的資源使用達到最大的效能，並防止不正常的資源消耗。

在傳統網路上的資源使用大都以 bw 為主，對於 bw 及 memory 的需求大都是與其所傳送封包的大小成比例關係，甚至對於 CPU 的需求幾乎為一常數。在主動式網路上的資源使用則不能僅考慮 bw，因為主動式節點可執行程式的緣故，使得 CPU 及 memory 資源的分配必需妥善處理。此外 flow 對資源需求的特性可以分成四種，第一種為 flow 對 CPU 及 bw 的需求量皆不高，例如：一般資料查詢服務；第二種為 flow 對 CPU 及 bw 的需求量皆很高，例如：影音資料傳輸及壓縮服務；第三種為 flow 對 CPU 的需求量高但是對 bw 的需求量低，例如：數學運算服務；第四種為對 CPU 的需求量低但是對 bw 的需求量高，例如：檔案下載服務；若第一種 flow 與第二種 flow 同時存在於節點中時，稱之為 flow 間資源需求的不平衡(unbalance)，第三種與第四種為 flow 本身對不同資源需求的不平衡，針對這些各式各樣的服務需求，在本論文提出了依需求資源分配(on-demand resource allocation)演算法來滿足不同 flow 的資源需求特性，使得主動式節點有最大的效能，此演算法亦考慮了服務品質(Quality of Service, QoS)的課題，除了提供設定優先權(priority)的機制外，亦限制每個 flow 被分配資源的比率上限；依照四種不同 flow 對資源需求的特性，本論文設計了三種實驗環境來比較均等資源分配(equal resource allocation)、公平資源分配(fair resource allocation)及所提出的依需求資源分配演算法，結果顯示依需求資源分配演算法在三種實驗環境下都有比較好的效能。

在本論文中，第二節為相關研究，在第三節提出本論文研究所根據的主動式節點架構及依需求資源分配演算法，第四節為模擬在三種不同實驗環境下與其它演算法比較的結果與分析，最後為本論文的結論。

二、相關研究

在傳統網路節點上的資源分配從早期以封包先進先出(First-In-First-Out, FIFO)或先到先服務(First-Come-First-Serve, FCFS)的原則到後來演進為公平佇列(Fair Queuing, FQ)[6]及權重式公平佇列(Weighted Fair Queuing, WFQ)[1]的方法，都只考慮單一資源分配，並無法直接套用於主動式網路中多項資源分配的特性。在主動式網路中有關資源分配與管理的研究方面，Janos [11] 為一專為主動式網路設計的 NodeOS 架構，其特性為堅強的資源管理及對不可靠的 Java 應用程式控制，Janos 最底層為 Moab，它以 domain 為資源控制及終止回收的單位，依序對各 domain 分配資源，防止不同 domain 間使用資源的相互干擾，domain 為類似一般作業系統的 process 觀念，Janos 對於資源管理的特性是保護資源。Brunner 與 Stadler [8] 提出 VAN (Virtual Active Network)架構，VAN 是網路使用者與網路提供者的介面，網路使用者透過 VAN 可享有專用的網路資源來安裝並執行其想要的服務，並自行管理被分配的資源而不需要與網路提供者有太多的溝通，網路提供者只要負責分配資源給不同的網路使用者並區隔資源使用干擾即可，其 bw 資源分配是以 deficit round-robin 的方式，而 CPU 及 memory 資源分配是用 FCFS。RCANE (Resource Control Active Network Environment) [10]為一主動式網路資源控制架構，使用 virtual processor 來區隔每個 flow 的資源使用，以共用 heap 方式來解決不同 flow 間 GC

(Garbage Collection)相互干擾的問題，RCANE 在實作上以 Nemesis 為 NodeOS，並以 PLAN(Packet Language for Active Networks)作為 memory 區隔的工具，另外用修正後的 EDF (Earliest Deadline First) 演算法處理 CPU 及 bw 的分配；LARA (Lancaster Active Router Architecture) [12]的特性為一強調高效能的主動式節點架構，以 PAL (Platform Abstract Layer)來監督資源使用，在 CPU 方面分為 NodeOS 與 EE 二層做排程，以公平方式分配資源，並以 timer callback 監督違規使用的程式，其它資源也是以公平方式分配資源，並限制資源的使用以保護每個 flow 可使用的資源。在這些研究當中雖然提出了一些資源分配或管理的設計原則，但是對於同一 flow 中對不同資源需求有極大差異的情形或不同 flow 間對資源需求有極大差異的情形並沒有考慮；在 2000 年時，由 Ramachandran, Pandey 與 Chan [15]根據傳統網路的公平資源分配觀念及主動式網路的特性提出了簡單明確的公平資源分配演算法，指出在 flow 對 CPU 及 bw 需求不平衡的情況下, 若把節點的 CPU 及 bw 仍個別公平且均等的分配給每個 flow, 將會造成主動式節點內部的暫存區擁塞，他們提出將節點的 CPU 及 bw 整合一起考慮，然後以公平的方式分配則可解決此項問題，但他們提出的公平資源分配演算法並沒有考慮在主動式節點內不同 flow 對資源需求的特性及優先權的機制，若某些 flow 對 CPU 及 bw 的需求較其它 flow 高出許多時，則整個主動式節點的效能將受到限制。因此在本論文中提出了依需求資源分配演算法來滿足不同 flow 的資源需求特性並提供設定優先權的機制，使得主動式節點有最大的效能。

三、依需求資源分配演算法

本論文所依據的主動式節點架構是參考[15]，如圖 2 所示，在這個架構下的主動式節點分為二部份：CPU scheduler 及 output scheduler，本論文假設所有經由主動式節點的 capsule 皆需要 CPU 及 bw 資源；主動式節點處理 capsule 的程序如下，節點在網路卡收到 capsule 後，capsule 被放

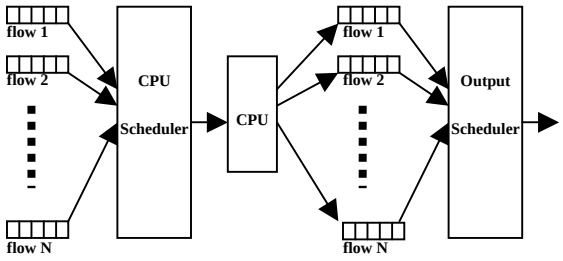

圖 2　主動式節點架構(參考[15])

入個別 flow 相對應的 CPU queue 中，CPU scheduler 從這些 CPU queue 中選擇 capsule 來處理，在處理完 capsule 後，所產生的 output capsule 被放入個別 flow 相對應的 output queue 中，output schedule 從這些 output queue 中選擇 capsule 傳送出去。本論文將依循[15]對於節點的 memory 資源不限制的假設, 即 CPU queue 及 output queue 的容量不設限制，另外在本論文中假設 capsule 需要的資源數量可用 AVNMP (Active Virtual Network Management Prediction)[14]預測，AVNMP 是使用一模擬模式比實際時間提前執行來預測資源的使用量，隨後再根據與實際的誤差值做修正，另外亦可用半隨機方式及狀態轉換建立 application model[13, 17]來預測 application 的資源使用量。

假設在主動式節點有 N 個 flow，均等資源分配採用 FQ 為資源分配的觀念、每個 flow 不論是在 CPU 或 bw 皆分配到相同比率 1/N 的資源，它的優點是實作簡單，缺點為當 flow 對 CPU 及 bw 的需求差異極大或 flow 間的資源需求差異極大時，資源無法適當的調配。公平資源分配由 Ramachandran, Pandey 與 Chan [15]提出，把節點的 CPU 及 bw 整合一起考慮，然後以公平的方式分配，每個 flow 在 CPU 加上 bw 皆分配到相同比率 2/N 的資源，它的優點為當 flow 對 CPU 及 bw 的需求差異極大時, 可在總分配的比率不超過 2/N 的公平原則下予以調配，缺點是在 flow 間的資源需求差異極大時，受到比率 2/N 的影響，資源仍無法適當的調配。本論文提出的依需求資源分配演算法即可解決這些問題，其特性為動態的依據 flow 的需求比率分配資源，為避免受到惡意或不正常的 capsule 攻擊而消耗太多資源，CPU 及 bw

個別最多只能分配 2/N, 若有多餘比例的資源則分配給分配比率小於 1/N 的 flow，如此保證即使在不公平的情況下分配資源，資源需求小的 flow 不會被資源需求大的 flow 排擠，另外一項特性為優先權機制, 優先權只有高和低二種等級, 一般 flow 優先權設為低等級，使用者有需求時可設為高等級，等級設為高時即分配二倍的需求，設為二倍的理由為假設在 queue 中至少有一個 capsule 在等待，如此使得 flow 有較多的資源可加速處理。

依需求資源分配演算法如圖 3 所示，在時間 t 時計算每個 flow 被分配資源的數量，首先分別計算 CPU 及 bw 在時間 t 時全部 flow 的總需求量，檢查每個 flow 的 CPU 或 output queue 是否有 capsule 等待處理, 如果有 capsule 等待處理則檢查其 flow 的優先權是高或低，若是優先權高的 capsule 則加倍計算其需求，優先權低的 capsule 則以一般需求計算；接下來根據需求來計算每個 flow 所能分配到資源的比率，若發現有 flow 分配比率超過 2/N, 則將其限制在 2/N, 如此可避免有些 flow 佔用太多 resource；然後把所有 flow 分配比率超過 2/N 的部分收集起來分配給分配比率未超過 1/N 的 flow，這樣可避免當 flow 間對資源需求數量相差太大時，資源需求少的 flow 不會分配到太少的資源，而能維持一定的服務品質，最後根據這些比率來計算主動式節點分配到每個 flow 的資源數量，假如所分配的資源為足夠處理本次需求的情況下，capsule 在 CPU queue 時，則將此 capsule 移至 output queue 等待輸出，若 capsule 在 output queue 則表示此 capsule 已輸出完成，但如果所分配的資源為不足夠處理本次的需求，剩餘未處理完的部份則等待下次再處理。

圖 3 的符號說明如下， N ：在節點中的 flow 數量。 N_C ：使用 CPU 的比例小於 1/N 的 flow 數目。 N_O ：使用 bw 的比例小於 1/N 的 flow 數目。 $NodeCpu$ ：節點的 CPU 資源數量。 $NodeBw$ ：節點的 bw 資源數量。 $CpuQueue[i]$ ：flow i 在 CPU queue 的 capsule 數量。 $OutQueue[i]$ ：flow i 在 output queue 的 capsule 數量。 $Cpu\,\mathrm{Pr}\,iority[i]$ ：flow i 在 CPU scheduler 的優先權。 $Out\,\mathrm{Pr}\,iority[i]$ ：flow i 在 output scheduler 的優先權。 $CpuDemand[i]$ ：flow i 所需 CPU 數量。 $OutDemand[i]$ ：flow i 所需 bw 數量。 $SumCpuDemand[i]$ ：全部 flow 所需 CPU 數量。 $SumOutDemand[i]$ ：全部 flow 所需 bw 數量。 $CpuPercentage[i]$ ：分配 flow i 的 CPU 比率。 $OutPercentage[i]$ ：分配 flow i 的 bw 比率。 $SpareCpuPercentage[i]$ ：節點所剩餘的 CPU 資源比率。 $SpareOutPercentage[i]$ ：節點所剩餘的 bw 資源比率。 $CpuAllocate[i]$ ：分配 flow i 的 CPU 數量。 $OutAllocate[i]$ ：分配 flow i 的 bw 數量。

四、實驗結果

本論文用模擬程式來比較分析均等資源分配、公平資源分配及依需求資源分配演算法，模擬由 10 個 host 產生 10 個相對應的 flow, 每個 flow 產生 capsule 的時間週期為遵循平均值為 5 個時間單位的指數分佈(exponential distribution)，執行期間為 1000 個時間單位；主動式節點的 CPU 資源為每個時間單位可執行 50 個 cycles, bw 資源為每個時間單位可傳送 5000 個 bits；每個 flow 產生的 capsule 所需的 CPU 及 bw 資源數量要均勻的分佈在上下限範圍內，避免集中在某些值，所以使用均勻分佈(uniform distribution)產生, 均勻分佈的上限及下限值則根據每個 flow 的特性而有不同，flow 對 CPU 及 bw 資源的需求分為高與低二種，對 CPU 需求低的 flow 產生的 capsule 所需的 CPU 數量由 1 到 5cycles 的均勻分配產生，對 CPU 需求高的 flow 產生的 capsule 所需的 CPU 數量由 20 到 35cycles 的均勻分配產生, 對 bw 需求低的 flow 產生的 capsule 所需的 bw 數量由 200 到 600bits 的均勻分配產生, 對 bw 需求高的 flow 產生的 capsule 所需的 bw 數量由 3000 到 4000bits 的均勻分配產生，根據這些 flow 的特性本論文建構下列三種實驗環境 ：

- 實驗環境 1：

每個 flow 產生的 capsule 所需的 CPU 數量由 1 到 5cycles 的均勻分配產生, 所需的 bw 數量由 200 到 600bits 的均勻分配產生。此為建構 flow 對

Algorithm On-demand resource allocation

Active node operation at t time unit

Initial $SumCpuDemand \leftarrow 0$; $SpareCpuPercentage \leftarrow 0$; $N_C \leftarrow N$;

$SumOutDemand \leftarrow 0$; $SpareOutPercentage \leftarrow 0$; $N_o \leftarrow N$;

for (flow $i \leftarrow 1$ to N) do

if $(CpuQueue[i] > 0)$ then //計算在時間 t 時 CPU 需求總和

$SumCpuDemand \leftarrow SumCpuDemand + CpuDemand[i]$;

if ($CpuPriority[i]$ is *high*) then $SumCpuDemand \leftarrow SumCpuDemand + CpuDemand[i]$;

else $CpuDemand[i] \leftarrow 0$; $N_C \leftarrow N_C - 1$;

if $(OutQueue[i] > 0)$ then //計算在時間 t 時 bw 需求總和

$SumOutDemand \leftarrow SumOutDemand + OutDemand[i]$;

if ($OutPriority[i]$ is *high*) then $SumOutDemand \leftarrow SumOutDemand + OutDemand[i]$;

else $OutDemand[i] \leftarrow 0$; $N_O \leftarrow N_O - 1$;

for (flow $i \leftarrow 1$ to N) do

$CpuPercentage[i] \leftarrow \frac{CpuDemand[i]}{SumCpuDemand}$; //計算在時間 t 時 flow i 的 CPU 需求比率

if ($CpuPriority[i]$ is *high*) then $CpuPercentage[i] \leftarrow 2 \times CpuPercentage[i]$;

if ($CpuPercentage[i] > \frac{2}{N}$) then //限制 flow 被分配的 CPU 比率最多 $\frac{2}{N}$

$SpareCpuPercentage \leftarrow CpuPercentage[i] - \frac{2}{N} + SpareCpuPercentage$;

$CpuPercentage[i] \leftarrow \frac{2}{N}$; $N_C \leftarrow N_C - 1$;

$OutPercentage[i] \leftarrow \frac{OutDemand[i]}{SumOutDemand}$; //計算在時間 t 時 flow i 的 bw 需求比率

if ($OutPriority[i]$ is *high*) then $OutPercentage[i] \leftarrow 2 \times OutPercentage[i]$;

if ($OutPercentage[i] > \frac{2}{N}$) then //限制 flow 被分配的 bw 比率最多 $\frac{2}{N}$

$SpareOutPercentage \leftarrow OutPercentage[i] - \frac{2}{N} + SpareOutPercentage$;

$OutPercentage[i] \leftarrow \frac{2}{N}$; $N_O \leftarrow N_O - 1$;

for (flow $i \leftarrow 1$ to N) do

if $(CpuPercentage[i] > 0)$ then //多的資源分配給比率少於 $\frac{1}{N}$ 的 flow

if ($CpuPercentage[i] < \frac{1}{N}$) then $CpuPercentage[i] \leftarrow \frac{SpareCpuPercentage}{N_C} + CpuPercentage[i]$;

$CpuAllocate[i] \leftarrow NodeCpu \times CpuPercentage[i]$;

if ($CpuAllocate[i] \geq CpuDemand[i]$) then

move capsule from CPU queue to output queue; $CpuDemand[i] \leftarrow 0$;

else $CpuDemand[i] \leftarrow CpuDemand[i] - CpuAllocate[i]$;

if $(OutPercentage[i] > 0)$ then

if ($OutPercentage[i] < \frac{1}{N}$) then $OutPercentage[i] \leftarrow \frac{SpareOutPercentage}{N_O} + OutPercentage[i]$;

$OutAllocate[i] \leftarrow NodeBw \times OutPercentage[i]$;

if ($OutAllocate[i] \geq OutDemand[i]$) then capsule finish; $OutDemand[i] \leftarrow 0$;

else $OutDemand[i] \leftarrow OutDemand[i] - OutAllocate[i]$;

圖 3 依需求資源分配演算法

CPU 及 bw 需求平衡及每個 flow 之間對資源需求平衡的環境

- 實驗環境 2：

flow 1 到 4 產生的 capsule 所需的 CPU 數量由 20 到 35cycles 的均勻分配產生，所需的 bw 數量由 200 到 600bits 的均勻分配產生；flow 5 到 10 產生的 capsule 所需的 CPU 數量由 1 到 5cycles 的均勻分配產生，所需的 bw 數量由 3000 到 4000bits 的均勻分配產生。此為建構 flow 對 CPU 及 bw 需求不平衡的環境。

- 實驗環境 3：

flow 1 到 4 產生的 capsule 所需的 CPU 數量由 20 到 35cycles 的均勻分配產生，所需的 bw 數量由 3000 到 4000bits 的均勻分配產生；flow 5 到 10 產生的 capsule 所需的 CPU 數量由 1 到 5cycles 的均勻分配產生，所需的 bw 數量由 200 到 600bits 的均勻分配產生。此為建構每個 flow 之間對資源需求不平衡的環境。

本論文分別模擬均等資源分配、公平資源分配及依需求資源分配演算法在這三種不同特性的實驗環境。在實驗環境 1 之 flow 對資源需求平衡情況的實驗結果顯示三種演算法不論在 CPU 或 bw 的效能幾乎一樣，此外在每個 flow 的 CPU queue 的平均等待時間皆為 0，表示 CPU 皆能即時處理每個 flow 產生的 capsule；圖 4 則顯示三種演算法在每個 flow 的 output queue 的平均等待時間，只有依需求資源分配演算法能即時傳送每個 flow 的 capsule，其他二種都需要等待時間，而公平資源分配演算法等待時間較高的原因為 CPU 資源使用較多的緣故。

在實驗環境 2 之 flow 對 CPU 及 bw 需求不平衡情況的實驗結果如圖 5 至 10 所示，由圖 5 顯示在 CPU 使用效能方面，公平資源分配及依需求資源分配演算法二者相同且較均等資源分配演算法的效能要好，圖 6 顯示在 bw 使用效能方面，依需求資源分配演算法則明顯優於其他二者；圖 7 為 CPU 處理失敗率的比較，在 flow 對 CPU 及 bw 需求不平衡情況下，均等資源分配演算法無法適時的調配資源，使得對 CPU 有高度需求的 flow 1 到 4 沒有足夠的資源處理，造成失敗率高，相對應的在圖 8 的 CPU queue 平均等待時間都遠較其它二種演算法長；圖 9 為 output 輸出失敗率的比較，由於均等資源分配演算法的 flow 1 到 4 在前面 CPU queue 等待的 capsule 太多，以至於到後面 output 輸出的 capsule 相對較少，所以在此忽略，觀察對 bw 有高度需求的 flow 5 到 10，仍與在 CPU 的情形類似，均等資源分配演算法仍無法適時的調配資源，使得對 bw 有高度需求的 flow 5 到 10 沒有足夠的資源處理，在這種實驗環境下，公平資源分配及依需求資源分配演算法二者有較好的資源分配調整能力，但是再比較二者在圖 9 與 10 的表現，發現公平資源分配演算法在 flow 1 到 10 的 output 輸出的失敗率與平均等待時間皆較依需求資源分配演算法高出許多、這是因為公平資源

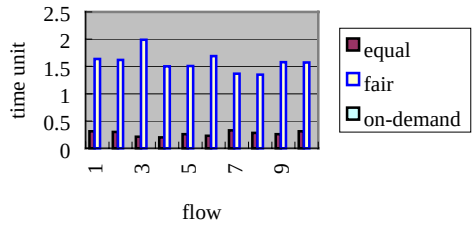

圖 4 實驗環境 1 output queue 平均等待時間比較

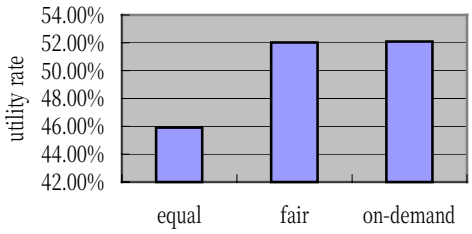

圖 5 實驗環境 2 CPU 的效能比較

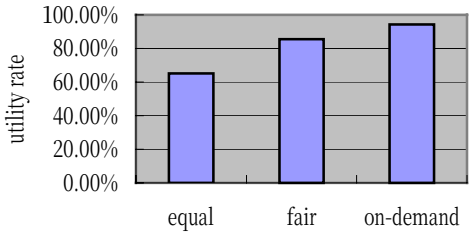

圖 6 實驗環境 2 bw 的效能比較

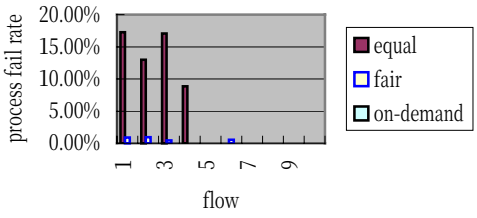

圖 7 實驗環境 2 CPU 處理失敗率比較

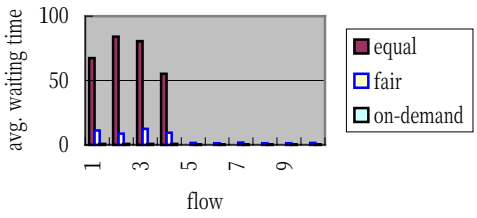

圖 8 實驗環境 2 CPU queue 的平均等待時間比較

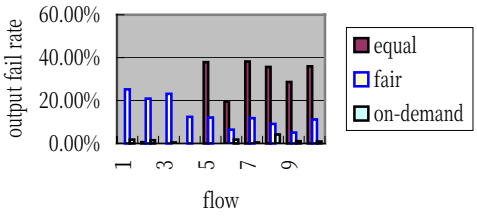

圖 9 實驗環境 2 output 輸出失敗率比較

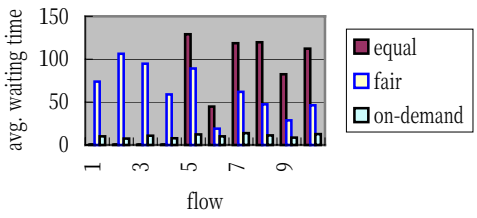

圖 10 實驗環境 2 output queue 平均等待時間比較

分配演算法在 CPU 使用較多的情形下，影響 bw 分配的比率。

在實驗環境 3 之每個 flow 間對資源需求不平衡情況的實驗結果如圖 11 至 16 所示，由圖 11 顯示在 CPU 使用效能方面，公平資源分配及依需求資源分配演算法二者相同且較均等資源分配演算法的效能要好，圖 12 顯示在 bw 使用效能方面，雖然依需求資源分配演算法明顯優於其他二種演算法，但公平資源分配演算法的效能明顯較差，這是因為在處理對 CPU 及 bw 需求都很高的 flow 時，由於在 CPU 使用較多的情形下，使得 bw 分配的比率太少，造成即使其它 flow 有較多閒置的 bw 亦無法分配到不敷使用的 flow，這是公平資源分配演算法的缺點；圖 13 為 CPU 處理失敗率的比較，在每個 flow 間對資源需求不平衡情況下，均等資源分配演算法同樣無法適時的調配資源，使得對 CPU 有高度需求的 flow 1 到 4 沒有足夠的資源處理，造成失敗率偏高，相對應的在圖 14 的 CPU queue 平均等待時間都遠較其它二種演算法長；圖 15 為 output 輸出失敗率的比較，結果與實驗環境 2 類似，由於均等資源分配演算法的 flow 1 到 4 在前面 CPU queue 等待的 capsule 太多，以至於到後面 output 輸出的 capsule 相對較少，所以在此忽略，觀察公平資源分配演算法在圖 15 及 16 的表現，如前所述，由於 flow 1 到 4 的 CPU 使用比率超高的情形下，使得 bw 分配的比率太少，造成 output 輸出失敗率大幅提高，也相對的使在 output queue 的平均等待時間加長；在這種實驗環境下，依需求資源分配演算法仍有很好的表現。由以上實驗結果可知本論文所提出的依需求資源分配演算法除了有較好的 CPU 及 bw 使用效能之外，對於有各種需求特性的 flow，不論在 CPU queue 或 output queue，capsule 的處理完成率及平均等待時間表現皆較公平資源分配及均等資源分配演算法優異。

五、結論

本論文提出了依需求資源分配演算法其優點為：1) 依 flow 需求來滿足不同的資源需求特性，使節點資源可適當調配，2) 每個 flow 被分配資源的比率有上限，可避免資源被不正常消耗及保障分配資源比率較少的 flow，3) 有優先權機制，讓有快速處理需求的 flow 有較多的資源；依照不同 flow 對資源需求的特性，設計了三種實驗環境來比較均等資源分配、公平資源分配及本論文所提出的依需求資源分配演算法，結果顯示依需求資源分配演算法在三種實驗環境下都有比較好的效能。

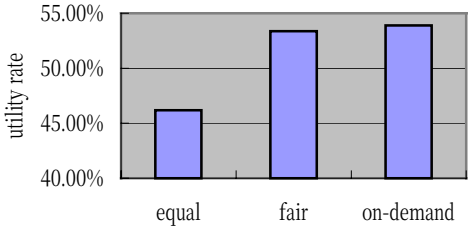

圖 11 實驗環境 3 CPU 的效能比較

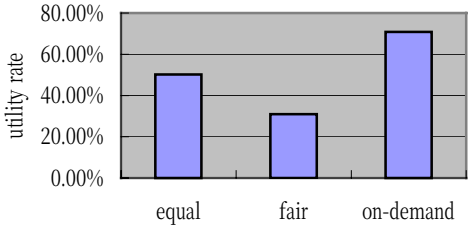

圖 12 實驗環境 3 bw 的效能比較

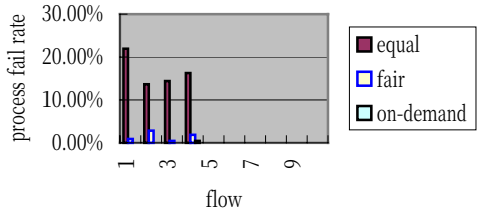

圖 13 實驗環境 3 CPU 處理失敗率比較

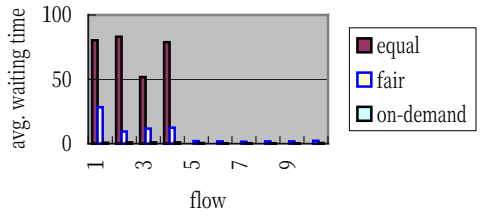

圖 14 實驗環境 3 CPU queue 的平均等待時間比較

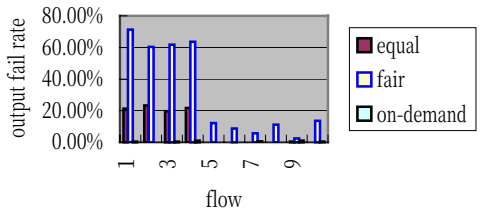

圖 15 實驗環境 3 output 輸出失敗率比較

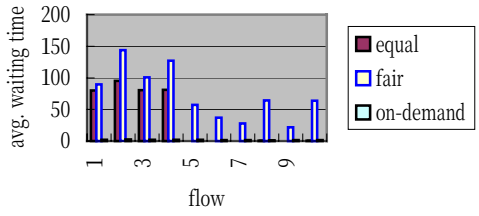

圖 16 實驗環境 3 output queue 平均等待時間比較

參考資料

[1] A. Demers, S. Keshav, and S. Shenker. “Analysis and Simulation of a Fair Queueing Algorithm.” Internetworking: Research and Experience, Vol. 1, No. 1, pp. 3–26, 1990

[2] D. Decasper, B. Plattner, G. Parulkar, S. Choi, J. D. DeHart, and T. Wolf. “A Scalable High-Performance Active Network Node.” IEEE Network, pp. 8-19, January/February 1999.

[3] D. Tennenhouse, J. M. Smith, W. D. Sincoskie, D. J. Wetherall, and G. J. Minden. “A Survey of Active Network Research.” IEEE Communications Magazine, pp. 80-86, January 1997.

[4] D. Wetherall, U. Legedza, and J. Guttag, “Introducing New Internet Services: Why and How.” IEEE Network, pp. 12- 19, May/June 1998.

[5] J. M. Smith, K. L. Calvert, S. L. Murphy, H. K. Orman, and L. L. Peterson. “Activating Networks: A Progress Report.” IEEE Computer, Vol. 32, No. 4, pp. 32-41, April 1999.

[6] J. Nagle. “On Packet Switches with Infinite Storage.” IEEE Transactions on Communications, Vol. 35, pp. 435-438, 1987.

[7] L. Peterson, Y. Gottlieb, M. Hibler, P. Tullman, J. Lepreau, S. Schwab, H. Dandekar, A. Purtell, and J. Hartman. “An OS Interface for Active Routers”, IEEE Journal on Selected Areas of Communications, Vol. 19, No. 3, March 2001.

[8] M. Brunner, and R. Stadler. “Management in

Telecom Environments that are based on Active Networks.” Journal of High Speed Networks, March 2001.

[9] M. Shreedhar, and G. Varghese. “Efficient Fair Queuing using Deficit Round Robin.” IEEE/ACM Transactions on Networking, Vol. 4, No. 3, pp. 375-385, June 1996.

[10] P. Menage. “RCANE: A Resource Controlled Framework for Active Network Services.” In Proc. Of the First International Working Conference on Active Networks (IWAN’99), Berlin, 1999.

[11] P. Tullmann, M. Hibler, and J. Lepreau. “Janos: A Java-Oriented OS for Active Network Nodes.” IEEE Journal on Selected Areas in Communications, Vol. 19, No. 3, pp. 501-510, March 2001.

[12] R. Cardoe, J. Finney, A.C. Scott, and W.D. Shepherd. “LARA: A Prototype System for Supporting High Performance Active Networking.” In Proc. of the First International Working Conference on Active Networks (IWAN '99), Berlin, 1999.

[13] V. Galtier, K. Mills, Y. Carlinet, S. Leigh, and A. Rukhin. “Expressing Meaningful Processing Requirements among Heterogeneous Nodes in an Active Network.” In Proc. of the Second International Workshop on Software and Performance (WOSP 2000), Ottawa, Canada, September 2000.

[14] V. Galtier, K. Mills, Y. Carlinet, S. Bush, and A. Kulkarni. “Predicting Resource Demand in Heterogeneous Active Networks.” In Proc. of the MILCOM 2001, Virginia, October 2001.

[15] V. Ramachandran R. Pandey, and S.-H. G. Chang. “Fair Resource Allocation in Active Networks.” In Proc. of the Ninth International Conference on Computer Communications and Networks, pp.468 -475, 2000.

[16] S. Floyd, and V. Jacobson. “Link-sharing and Resource Management Models for Packet Networks.” IEEE/ACM Transactions on Networking, Vol. 3, No. 4, pp. 365-386, August 1995.

[17] Y. Carlinet, V. Galtier, K. Mills, S. Leigh, and A. Rukhin.” Calibrating an Active Network Node.” In Proc. Of the Second Annual Workshop on Active Middleware Services convened in conjunction with the Ninth IEEE International Symposium on High Performance Distributed Computing, Pennsylvania, August 2000.