雲端運算中動態調整虛擬機器運算資源機制

DYNAMIC ADJUSTMENT MECHANISM OF THE VIRTUAL MACHINE COMPUTING RESOURCE IN THE CLOUD COMPUTING

研究生：胡志凱 (Chih-Kai Hu)

指導教授：李良德 (Prof. Liang-The Lee)

大同大學

資訊工程研究所

碩士論文

Dynamic Adjustment Mechanism of the Virtual Machine Computing Resources in the Cloud Computing
Tatung University

中華民國九十九年七月

July 2010

大同大學
資訊工程研究所
碩士學位論文

雲端運算中動態調整虛擬機器運算資源機制

胡志凱

經 考 試 合 格 特 此 證 明

碩士學位論文考試委員

李良德

[illegible]

[illegible]

指導教授

李良德

所長

[illegible]

中 華 民 國 99 年 7 月 9 日

2.1.1 XEN

Xen[2] 是一個開放原始碼，是為 x86 架構的機器而設計的虛擬化軟體；圖 2-3 為 XEN 系統架構。Xen 的主要特色是以半虛擬化的方法實現，這項技術是透過修改虛擬系統的核心，加入一個 Xen 管理程序(Hypervisor)層。它允許安裝在同一硬體設備上的多個虛擬系統可以同時啟動，並且由 Xen 管理程序來進行資源調配。讓虛擬主機中的核心可與外部的 VMM 系統的核心直接透過核心的溝通達到效能的提升，讓系統能被虛擬在 Xen 上面。

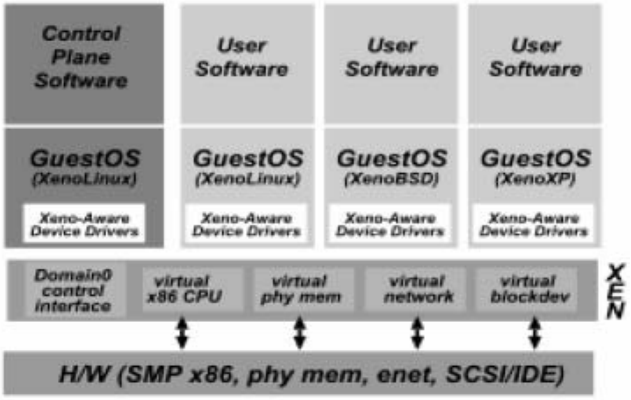

圖 2-3、XEN 架構圖

然而，半虛擬化技術的缺點是，虛擬機器內的操作系統核心必須進行修改，因此目前只支援以 Linux 為基礎的作業系統。在 CPU 大廠 Intel 提供 VT（Virtualization Technology）技術和 AMD-V（AMD Virtualization），使得 XEN 可透過全虛擬化動作來運行許多不同的 x86 作業系統，全虛擬化在不修改 Windows 核心下，也能執行 Windows 系統或 Linux 系統。使用 Intel VT 和 AMD-V 硬體虛擬技術的另一個重點是：以往在沒有硬體虛擬技術時，全虛擬化下的虛擬機器效能低落，藉由硬體虛擬化技術，可使虛擬機器存取硬體效能更加快速與直接，增進虛擬機器內系統效能與可用度。在全虛擬化的狀態下，虛擬機器的硬體裝置都是虛擬出來的，如 CPU、硬

碟，網路介面卡等，因此驅動程式是以虛擬出來的裝置來驅動，CPU 需花費大量的轉譯工作，故在效能上半虛擬的方式較佳。

2.1.2 KVM(Kernel-based virtual machine)

KVM[3]是一個運行在作業系統上的管理程序(Hypervisor)。透過 Linux 核心開發方法把 Linux 核心本身當作一個管理程序並整合管理程序功能，因此可簡化管理和提高虛擬化環境的性能。這種方法的優點是，增加虛擬化效能，即使沒有特別的硬體也能夠達到半虛擬的效能；並可以受益於 Linux 核心本身所有機制，在這種模式下，每個虛擬機器只是一個普通的 Linux 程序(Process)，因此是以標準 Linux 排程(schedule)方式進行運作。一個普通的 Linux 程序有兩種模式執行：核心模式和用戶模式。用戶模式是一個應用程式預設的模式，當應用程序需要從核心作某些動作時會進入核心模式，如讀寫硬碟。KVM 增加了第三種模式，來賓模式(Guest mode)。來賓模式是運行在虛擬機器上，它擁有自己的核心和用戶空間，使虛擬機器內可正常運行與一般作業系統的模式一樣；而在虛擬機器內的 I/O 是由 Qemu 所模擬的，KVM 之架構如圖 2-4 所示。

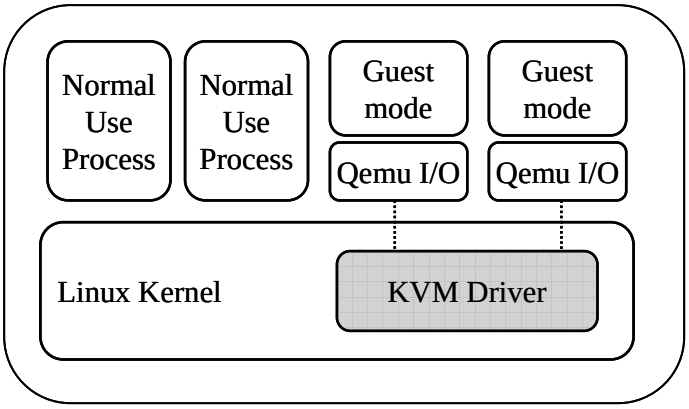

圖 2-4、KVM 架構圖。

如 2-4 顯示，KVM 包含了管理虛擬機器的虛擬硬體裝置、與底層溝通的 KVM Driver、透過修改過的 Qemu 模擬器、以及虛擬硬體裝置給虛擬機器的 Normal Process。

2.1.3 虛擬化資源管理

虛擬化技術的伺服器整合可確保節省能源及硬體支出成本，然而要安全地將應用程式由實體機器轉到虛擬化環境，就必須評估虛擬化本身的資源需求，才不致造成誤判[4]。在資源管理方面，有兩種方法，一種為非限制方式，另一個為限制方式；在非限制方式中，虛擬機器的資源使用是不會被限制，它可以使用不超過這台實體主機的所有資源，這個方法有一個很大的問題，因為這樣會造成其他運行在同一台實體主機的虛擬機器得不到任何資源，而造成資源搶奪的問題，因為它沒有資源使用優先權順序的觀念。第二種方法為限制方式，這個方法的優點是，每一部虛擬機器的資源使用量都被限制，不會無限制的取得實體主機資源，管理者可依據虛擬機器的重要程度不同而有不同的使用量設定，有了設定使用量的上限後，將可很容易地計算出系統資源的使用量及剩下多少可運用的資源，如此將可預測資源的運用[5, 6]。

2.2 雲端運算

雲端運算是一種分散式運算(Distributed Computing)的模式和概念，透過在雲端中的資料中心(Data Center)之動態可擴展性和虛擬化運算資源來提供服務，將龐大而複雜的運算處理程序自動分拆成無數個較小的子程序，交由多部伺服器所組成的龐大電腦叢集系統(Cluster)進行分散和平行運算分析後，將處理結果回傳給使用者[7]。雲端運算也是一種資源共享平台，包括硬體基礎建設、軟體、營運模式(Business

model)都是共享；在這當中其關鍵技術就是虛擬化[8]；透過虛擬化技術將資源彈性的分配，達到資源共享。

2.2.1 雲端運算類型

在雲端運算中主要有三種產業類型[1,7,9]:

一、軟體即服務(Software as a Service) SaaS:

是一種軟體應用的提供模式，應用軟體是由服務供應商所控管，將軟體及應用程式以網路服務形式進行，提供使用者軟體應用服務；例如 Google Docs。

二、平台即服務 (Platform as a Service) PaaS :

用來開發與執行應用程式服務的雲端平台，使用者可依服務供應商所提供的 API(Application Programming Interface)自行開發軟體，並透過該 API 取得資源；例如 Google App Engine。

三、基礎架構即服務(Infrastructure as a Service)IaaS；

提供電腦運算基礎設施，通常是一個虛擬環境的平台，作為一種服務。資源包括：伺服器(Servers)、網路設施(Network equipment)、記憶體(RAM)及儲存硬體(Disk)、CPU。並可動態資源配置、增加或減少依據應用程式運算資源需求；Amazon AWS EC2 (Amazon Web Service Amazon；Elastic Compute Cloud)為此種模式的雲端服務。

2.3 CPU 熱抽換 (CPU hot-plug)

現行的作業系統在開機時，將該作業系統核心載入後，已將 CPU 數量、記憶體大小等資訊載入，此時作業系統已經認得這些裝置設備，因此在作業系統啟動中，

新增 CPU 或是記憶體，理論上作業系統是無法讀取到相關 CPU 或是記憶體的加入。Linux Kernel 在 2.6 版本後，支援 CPU 的熱抽換(CPU hot-plug) [10]，其原理是在作業系統開機時先給予足夠的 CPU 數量，使作業系統在開機後可讀取到 CPU 的資訊，隨後可透過修改作業系統中的虛擬文件系統(Proc)，將 CPU 關閉，使排程器不會將工作排入此關閉之 CPU 內，當需要 CPU 時可在開啟。

第三章 系統設計與架構

本論文將實作一個運行在全虛擬化架構下的動態調整 vCPU 的機制。在本章 3.1 節說明本研究在動態調整 vCPU 資源的挑戰，3.2 節分析整個論文的設計，3.3 節說明本論文所提出之系統架構與流程。

3.1 動態調整vCPU資源之挑戰

本論文所提出之可動態調整虛擬主機中 vCPU 資源機制，在雲端運算中，對於某個時間點上，需處理大量工作之虛擬主機，利用使用者一開始預設的 CPU 最大值，可在高負載事件發生時，在不中斷情況下即時增加 vCPU 數量，讓事件發生對該虛擬機器的效能影響降至最低，並在工作處理完畢負載降低後，回收多餘未使用之 vCPU，以供有需求之使用者使用，讓資源利用度提高，如此也可達到“用多少資源算多少費用”，為雲端運算中虛擬機器使用者，提供一個可靠與穩定並且計費公平的虛擬化環境系統。在監控 vCPU 使用率上，因動態增加或減少 vCPU 數量，而讓虛擬機器中使用的 vCPU 數量不同，使 vCPU 使用率計算也會有所變動。故本研究於雲端運算中建立一套可動態調整虛擬主機之 vCPU 機制時，必須克服一些挑戰：(1) 如何在不中斷下調整 vCPU 資源; (2) 如何設計一個機制使 CPU 的增加是有效益的；(3) 如何詳細記錄每個虛擬機器 vCPU 之使用；(4)如何動態監控 vCPU 使用率。

3.2 系統設計與分析

本論文之 DAVMCR 是基於 KVM(Kernel based Virtual Machine)全虛擬化架構下設計，並且同時適用 XEN 全虛擬化架構下之虛擬機器，其主要之功能在於提供資源

調整的機制及監控虛擬機器 vCPU 負載。在本論文上開啟、關閉、暫停、新增、移除、編輯、儲存並回復虛擬機器等基本管理，皆使用免費軟體 VMM (Virtual Machine Manager)管理工具[11]，然而，VMM 對於動態資源調整並不支援在 KVM 全虛擬化架構下，也無自動化機制，因此本研究於此基礎上提出一套配合 VMM 管理工具之動態調整虛擬機器 vCPU 機制，以提高雲端運算中虛擬機器的負載管理、操作便利性、資源利用度及計費機制之精確度。

在相關研究中，我們發現到 KVM 架構下，使用 VMM 管理工具建立虛擬機器的時候，其 vCPU 資源的設定值中有一選項；vCPU 的最大值，它是屬於固定的資源配置方法，無法在虛擬機器開啟後再作調整；為的要有效的控制 vCPU 資源數量，本論文在 vCPU 的最大值將給予初始值為相對於實體主機的 CPU 數量，例如，實體主機有 4 顆 CPU 則設定為 4，如此有利於往後估算 vCPU 的使用情況與調整，包括詳細記錄每台虛擬機器使用多少 vCPU，並且各使用多久時間，這對於 DAVMCR 機制的運作是很重要的一項依據。因此，DAVMCR 機制需做到：(1)自動化調整虛擬機器之 vCPU 資源; (2) 對虛擬機器進行判斷與分析作為執行虛擬機器動態調整 vCPU 資源之依據; (3) 產生虛擬機器中 vCPU 詳細使用記錄；經上述分析，本論文設計之 DAVMCR 機制中將包含監控模組、調整資源模組、紀錄模組、判斷模組等模組，以達到目的。

3.3 系統架構

如圖 3-1 所示，本系統之四個模組分佈於圖中，各模組之間作協同作業，監控 vCPU 的同時也紀錄 VM(Virtual Machine)的 vCPU 的使用率及正在使用的 vCPU 數量，當判斷 VM 需調整 vCPU 則啟用調整資源並修改 VM vCPU 紀錄，並取得 VM 最新 vCPU 紀錄繼續監控 VM；圖 3-1 中 vCPU.MaxCount 為 vCPU 的最大值 、

vCPU.high.limit 為 vCPU 最高使用率門檻值、vCPU.low.limit 為最低使用率門檻值、vCPU.Time.limit 為持續時間。

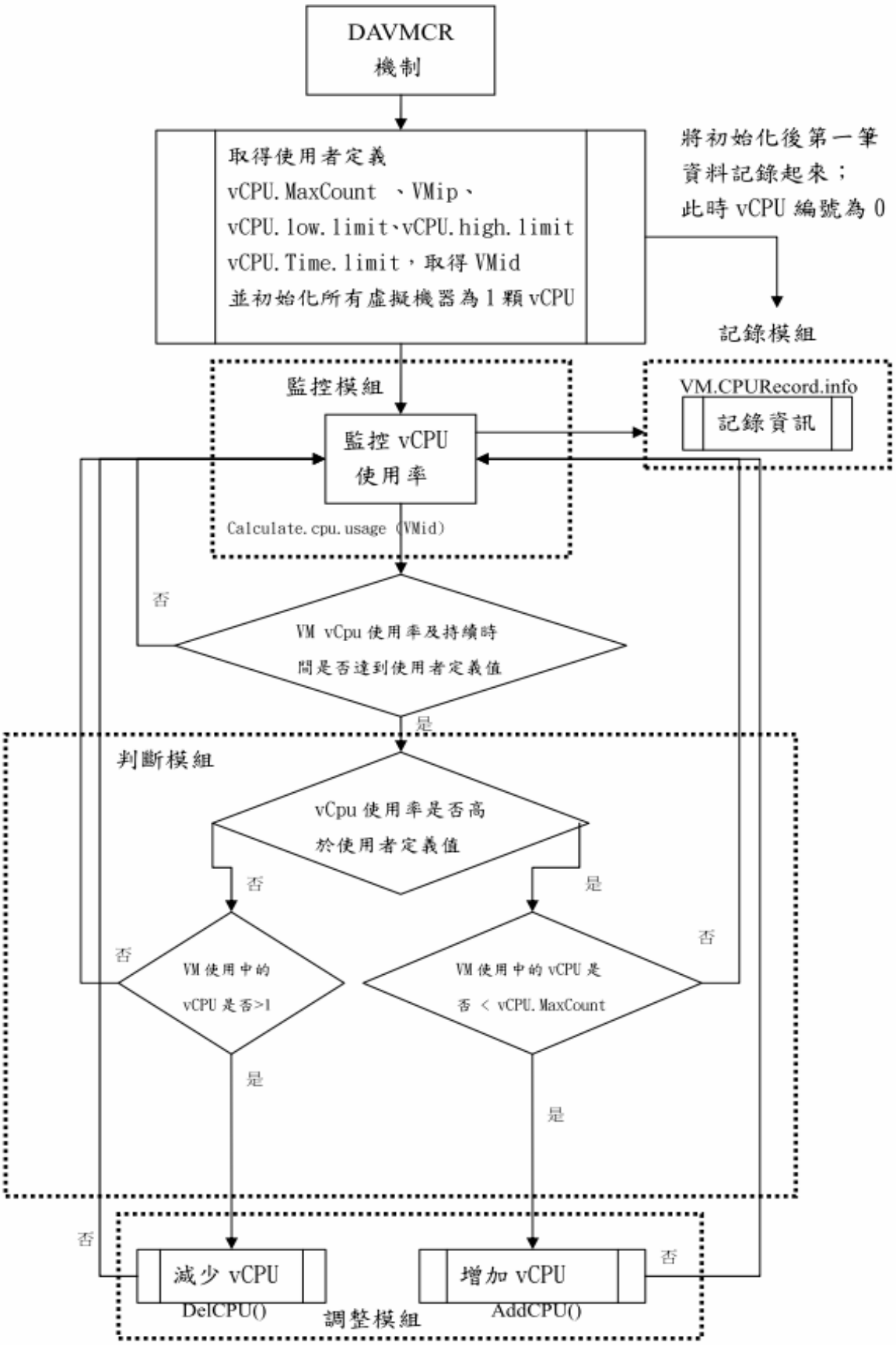

圖 3-1、DAVMCR 機制架構圖

本論文 DAVMCR 機制啟動後，取得使用者所定義的 vCPU.MaxCount、vCPU.high.limit、vCPU.low.limit、vCPU.Time.limit 之值，且初始化所有 VM 中的 vCPU 為 1 顆，該 vCPU 編號設為 0 並放入記錄模組作為第一筆記錄，並開始監控各 VM 中 vCPU 的使用率與數量。當 vCPU 的使用率、持續時間達到使用者預訂的門檻值時，啟動判斷模組；當 vCPU 使用率低於 vCPU.low.limit ，並且 VM 中使用的 vCPU 大於 1，則移除 1 個 vCPU 。如果需增加 vCPU 且 VM 中使用的 vCPU 小於 vCPU.MaxCount，則增加 1 個 vCPU，並啟動調整資源模組開始調整 vCPU 的增加或移除。調整資源模組則是，利用相關研究中所介紹的 CPU 動態調整(CPU hot-plus)之方法；遠端登入該虛擬機器，檢查該 VM 使用中的 vCPU 編號，並記錄回傳至 DAVMCR 機制中記錄模組，爾後調整資源模組將依據此記錄，選擇所要開啟或關閉 VM 中的 vCPU 之編號。

依據上述之運作原理以設計 DAVMCR 機制，其演算法之虛擬碼如下：

BEGIN
Input:
//使用者預先定義條件
vCPU.MaxCount　　//vCPU 最大限制值
vCPU.Usage.high.limit　//最大 vCPU 使用率
vCPU.Usage.low.limit　//最低 vCPU 使用率
vCPU.limit.Time　　//持續時間

VMip　　//VM 內 IP
VMname　　//VM 名稱
Output:
取得 VM 中 vCPU 編號去調整 VM 的 vCPU 數量
Method:
取得啟動的 VMip
取得 vCPU.MaxCount
取得 vCPU.Usage.high.limit
取得 vCPU.Usage.low.limit

```
取得 vCPU.limit.Time
初始化 VM vCPU 數量 = 1
取得啟動中 VM 的 ID , VMid
VM vCPU 編號編號 0 記錄於 VM.CPURecord.info
//記算 vCPU 使用率
VM.CPU.Usage =Calculate.cpu.usage (VMid)
while true :
// 計算 VM CPU 使用率
 for VMid in VMname
//計算啟動中的 VM
   Calculate.cpu.usage (VMid)
     //將每次 VM vCPU 記錄下
   VM.CPURecord.info
//判斷 CPU.Usage 是否符合使用者定義值
  if VM.CPU.Usage >= vCPU.high.limit or VM.CPU.Usage <= vCPU.low.limit
    vCPU.limit.Time – 1
//持續時間達使用者定義值
           if   vCPU.limit.Time = 0
//當持續時間=0 判斷使用率是否高於定義值
            if VM.CPU.Usage => vCPU.high.limit
              if   VMvCPU < vCPU.MaxCount
                   AddCPU //增加 vCPU 到 VM
                   VM.CPURecord.info
                endif
            else VM.CPU.Usage <= vCPU.low.limit
                  if   VMCPU > 1
                   delCPU //從 VM 中減少 vCPU
                   VM.CPURecord.info
                  endif
              endif
           endif
   endif
 endforloop
endwhileloop
END DAVMCR
```

第四章 模擬實驗與結果分析

4.1 實驗環境

本論文所使用平台為 Linux Ubuntu 10.04，CPU 為 AMD Phenom(tm) 9650 Quad-Code Processor @2.30GHz，記憶體為 4GByte。開發程式語言為 Python 2.6。虛擬化函式庫則使用 LinVirt 0.7.1 版。詳細配置規格說明如表 4-1 所示；而虛擬機器內配置，分為 VM-F、VM-D、VM-S 三種型式，VM-F 為一開始即給 VM 4 個 vCPU；VM-D 為套用本研究機制，給予 VM 初始值為 1 個 vCPU 而 Max vCPU 為 4 個；VM-S 為給予 VM 1 個 vCPU；三種型式的 VM 主要差異在於 vCPU 的配置方式，在系統配置方面則一致，詳細規格說明如表 4-2 所示。

表 4-1、實體主機系統規格表。

硬體		規格說明
個人電腦	CPU	AMD Phenom(tm) 9650 Quad-Code Processor @2.30GHz
	RAM	4GB
	網路卡	1Giga/bit
軟體		規格說明
系統	作業系統	Ubuntu 10.04
	虛擬化軟體	KVM84
	虛擬機器管理	Red Hat Virtual Machine Manager 0.6.1[11]
	程式語言	Python2.6
	虛擬化函式庫	LinVirt 0.7.1
	SSH for Python	Paramiko 2.0

表 4-2、虛擬主機系統規格表。

虛擬機器		規格說明
VM-F (KVM 機制)	CPU 數量	4 vCPU
	RAM	1GB
	網路卡	虛擬網卡
VM-D (DAVMCR 機制)	CPU 數量	預設值 1 vCPU 、4 Max vCPU
	RAM	1GB
	網路卡	虛擬網卡
VM-S (KVM 機制)	CPU 數量	1 vCPU
	RAM	1GB
	網路卡	虛擬網卡
軟體		規格說明
系統	作業系統	CentOS 5.4
	網頁程式	Tomcat 5.0
	測試程式語言	Java、Bash

4.2 各型式VM的模擬實驗

本論文之模擬環境如 4-1 節所敘述；在 DAVMCR 之機制中，使用者定義參數部份如下：vCPU 最大值為 4、vCPU 使用率最高為 90%、最低為 30%、持續時間 5 秒。此外安裝 Tomcat5 Web Service 於模擬實驗中的 VM，並利用 Apache 內所附的 ab (Apache Benchmark) 壓力測試工具以產生大量連線，模擬 VM 之工作負載。本實驗利用 Apache ab 程式產生 20 個連線，每次同時 4 個連線，共 5 次；執行指令如下：

ab －n 20 －c 4 http://VMIP/index.htm。

參數 n 為連線數，c 為同時連線數，最後是所要測試的網址；每一個測試連線執行一個從 1 開始找出 700000 個質數的 Java 運算程式；比較虛擬機器在套用 DAVMCR 之機制後，平均反應時間與平均執行時間是否提升及當達到了使用者定義之參數值時，DAVMCR 機制是否確實地增加 vCPU 資源給指定的虛擬機器；本實驗將分別於實體主機上同時執行一台、兩台、三台、四台及五台虛擬機器作為實驗，每次實驗各執行五次測試連線，再取其平均值；圖 4-1 為本實驗架構圖。

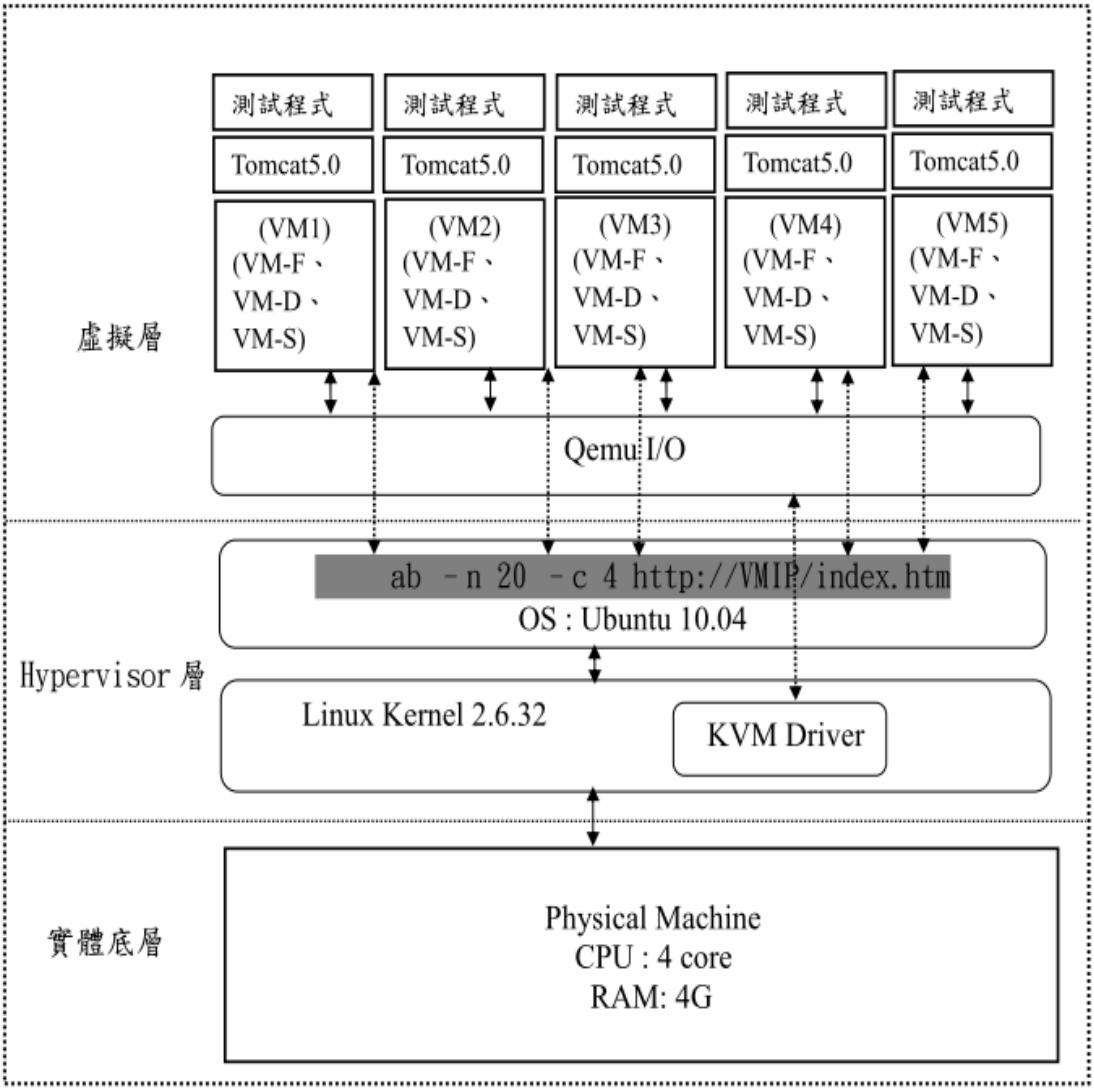
測試程式
測試程式
測試程式
測試程式
測試程式
Tomcat5.0
Tomcat5.0
Tomcat5.0
Tomcat5.0
Tomcat5.0
(VM1)
(VM-F、
VM-D、
VM-S)
(VM2)
(VM-F、
VM-D、
VM-S)
(VM3)
(VM-F、
VM-D、
VM-S)
(VM4)
(VM-F、
VM-D、
VM-S)
(VM5)
(VM-F、
VM-D、
VM-S)
虛擬層
Qemu I/O
ab -n 20 -c 4 http://VMIP/index.htm
OS : Ubuntu 10.04
Hypervisor 層
Linux Kernel 2.6.32
KVM Driver
實體底層
Physical Machine
CPU : 4 core
RAM: 4G

圖 4-1 模擬實驗架構圖

4.2.1 各型式VM的模擬實驗結果

實驗結果其數據顯示如表 4-3、表 4-4、圖 4-2 及圖 4-3 所示，在 VM-D 與 VM-S 的型式下，執行一台虛擬機器時，VM-D 型式下的反應時間與執行時間，有明顯的效能提升，平均約 46%提升，而在 VM-F 的型式下效能最佳，與 VM-D 及 VM-S 比較，約提升 40%和 68%效能；因只有執行一台 VM 下，VM-F 所使用的 4 個 vCPU 等同於對應到實體的 4 個 CPU，使用所有的 CPU 資源，而 VM-D 型式下，因使用者參數中的持續時間設定為 5 秒，故在反應時間與執行時間上與 VM-F 約有 5 秒的差距。

表 4-3 一台 VM 反應時間表。

VM-F vCPU=4 平均時間:5.68 秒					
次數	1	2	3	4	5
	反應時間	反應時間	反應時間	反應時間	反應時間
VM1	6.3	5.69	5.65	5.68	5.1
VM-D MaxvCPU=4 平均時間:9.46 秒					
次數	1	2	3	4	5
	反應時間	反應時間	反應時間	反應時間	反應時間
VM1	9.3	9.92	8.49	10.03	9.54
VM-S vCPU=1 平均時間:17.61 秒					
次數	1	2	3	4	5
	反應時間	反應時間	反應時間	反應時間	反應時間
VM1	17.68	17.64	17.68	17.61	17.43

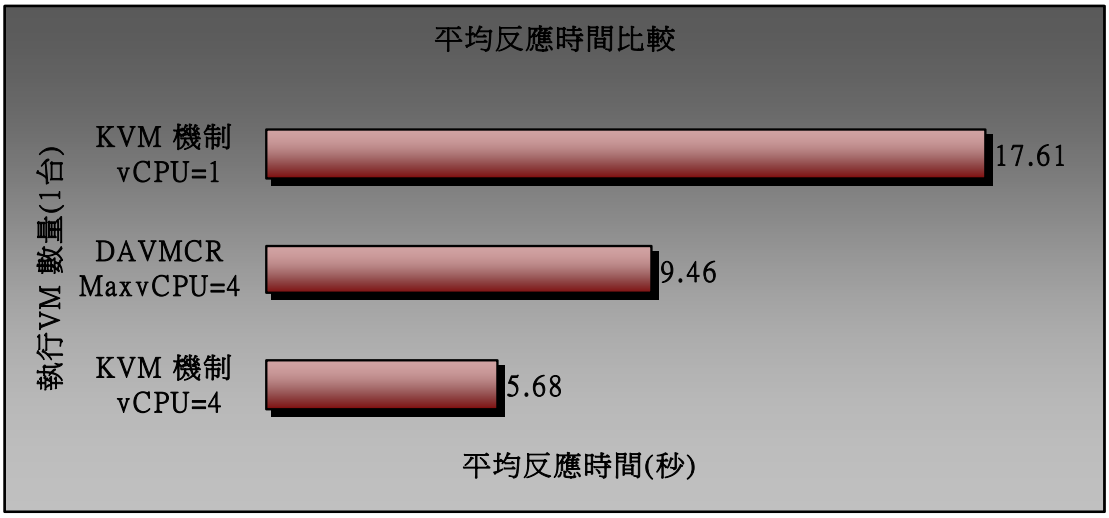

圖 4-2 一台 VM 平均反應時間圖

表 4-4 一台 VM 執行時間表。

VM-F vCPU=4 平均時間:30.05 秒					
次數	1	2	3	4	5
	執行時間	執行時間	執行時間	執行時間	執行時間
VM1	35.15	29.85	29.5	29.79	25.98
VM-D MaxvCPU=4 平均時間:49.38 秒					
次數	1	2	3	4	5
	執行時間	執行時間	執行時間	執行時間	執行時間
VM1	49.3	52.68	44.23	50.75	49.95
VM-S vCPU=1 平均時間:90.95 秒					
次數	1	2	3	4	5
	執行時間	執行時間	執行時間	執行時間	執行時間
VM1	91.66	90.59	90.87	90.76	90.88

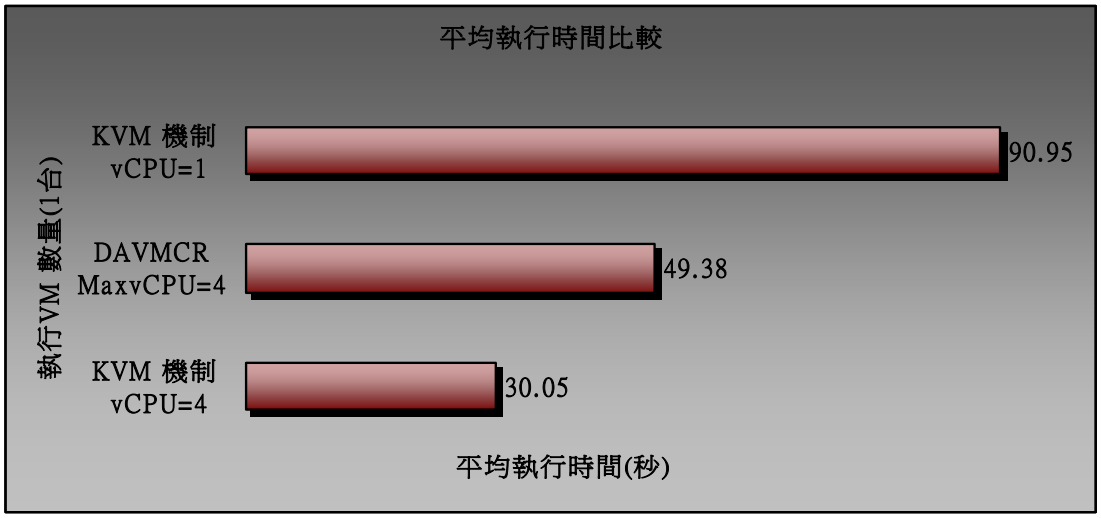

圖 4-3 一台 VM 平均執行時間圖

如表 4-5、表 4-6、圖 4-4 及圖 4-5 所示為同時執行兩台 VM 之實驗結果，在 VM-D 與 VM-S 的型式下，VM-D 的反應時間與執行時間，效能上依舊優於 VM-S 約 33%，但差距已有明顯縮小，而在 VM-F 的型式下效能仍然最佳，與 VM-D 及 VM-S 的差距也明顯縮小至 9%與 40%，因此時在 VM-F 型式下，vCPU 的使用數量是 8 個 vCPU，超過實體主機 4 個 CPU 數量，因而產生 VM 之間的 vCPU 競爭，使每一個 vCPU 只使用到一個實體 CPU 約 50%的使用率；由執行一台 VM 時之數據比較，發現 VM-F 和 VM-D 的平均反應時間與平均執行時間約分別下降 46%與 19%，因此也印證到 VM-F 使用每個實體 CPU 的 50%的使用率，而 VM-D 從下降的數據中可得知，此時 VM-D 的 vCPU 數量為 5 仍然超過實體的 CPU 數量，因此也產生 VM 之間的 vCPU 競爭，故效能上略為下降，VM-S 並無明顯的下降，因並未產生 VM 之間 vCPU 間的競爭。

表 4-5 二台 VM 反應時間表。

VM-F vCPU=4 平均時間:10.56 秒					
次數	1	2	3	4	5
	反應時間	反應時間	反應時間	反應時間	反應時間
VM1	11.0	11.05	11.17	10.63	10.07
VM2	12.4	10.37	10.15	9.46	9.28
VM-D MaxvCPU=4 平均時間:11.65 秒					
次數	1	2	3	4	5
	反應時間	反應時間	反應時間	反應時間	反應時間
VM1	10.77	10.65	12.65	11.49	11.18
VM2	12.61	11.91	11.56	10.75	12.91
VM-S vCPU=1 平均時間:17.46 秒					
次數	1	2	3	4	5
	反應時間	反應時間	反應時間	反應時間	反應時間
VM1	17.59	16.88	17.26	17.25	17.42
VM2	17.66	17.68	17.71	17.58	17.59

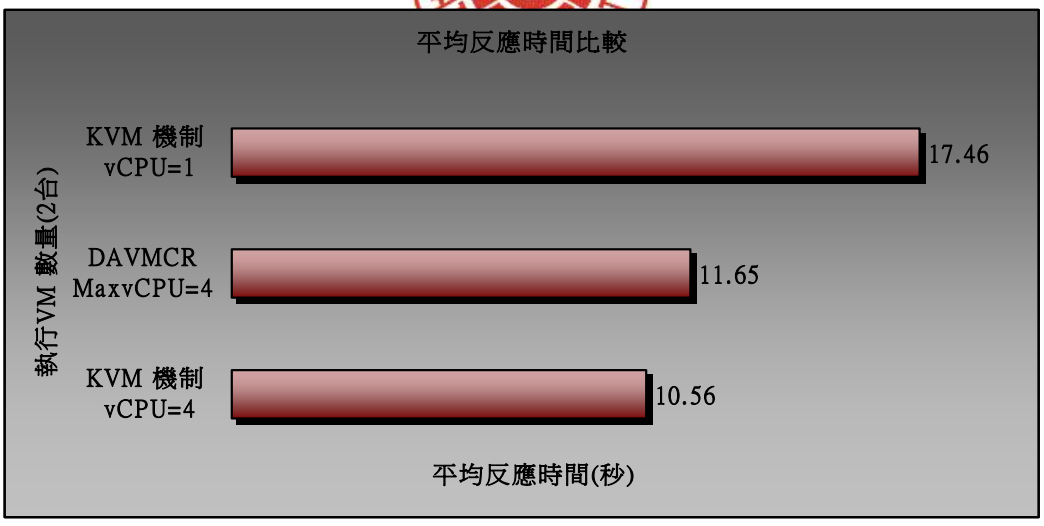

圖 4-4 二台 VM 平均反應時間圖

表 4-6 二台 VM 執行時間表。

VM-F vCPU=4 平均時間:55.34 秒					
次數	1	2	3	4	5
	執行時間	執行時間	執行時間	執行時間	執行時間
VM1	59.19	58.82	58.11	56.37	51.83
VM2	62.33	56.03	53.30	50.34	47.09
VM-D MaxvCPU=4 平均時間:60.33 秒					
次數	1	2	3	4	5
	執行時間	執行時間	執行時間	執行時間	執行時間
VM1	55.53	55.62	61.84	59.67	62.52
VM2	65.35	60.87	59.85	55.63	66.42
VM-S vCPU=1 平均時間:89.94 秒					
次數	1	2	3	4	5
	執行時間	執行時間	執行時間	執行時間	執行時間
VM1	90.39	88.04	88.91	89.15	89.48
VM2	90.72	90.81	91.00	90.46	90.45

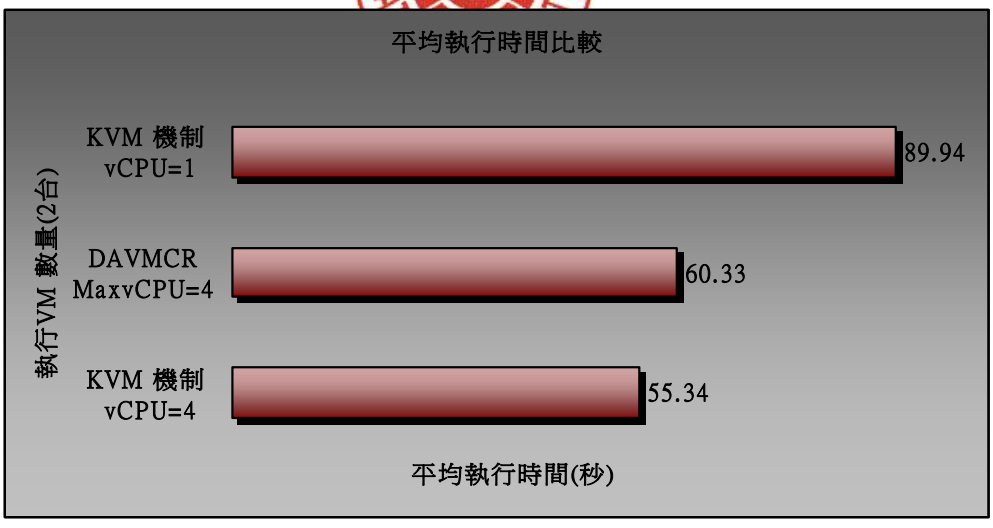

圖 4-5 二台 VM 平均執行時間圖

如表 4-7、表 4-8、圖 4-6 及圖 4-7 所示為同時執行三台 VM 之實驗結果其數據，在 VM-D 與 VM-S 的型式下，VM-D 的反應時間與執行時間，效能上依舊優於 VM-S 效能，但差距已剩 5%，而在 VM-F 的型式下效能仍然最佳，與 VM-D 及 VM-S 的差距約為 9%與 14%，與執行一台 VM 時之數據作比較，VM-F 和 VM-D 的平均反應時間與平均執行時間約下降約 61%與 45%，此時 VM-D 的 vCPU 數量約為 7 個，超過實體的 CPU 數量 3 個，因此也產生 VM 之間的 vCPU 競爭也越來越大，故效能上下降幅度也變大，而 VM-S 仍然未產生 VM 之間 vCPU 間的競爭，因此效能上未有差異。

表 4-7 三台 VM 反應時間表。

VM-F　vCPU=4 平均時間:15.28 秒					
次數	1	2	3	4	5
	反應時間	反應時間	反應時間	反應時間	反應時間
VM1	15.75	16.24	14.10	16.19	16.16
VM2	16.47	14.75	13.89	13.44	15.21
VM3	14.09	15.91	16.44	13.73	16.89
VM-D MaxvCPU=4 平均時間:16.88 秒					
次數	1	2	3	4	5
	反應時間	反應時間	反應時間	反應時間	反應時間
VM1	15.56	18.67	17.14	15.53	17.44
VM2	17.35	17.77	15.05	17.20	17.17
VM3	17.80	17.30	15.76	16.46	16.96
VM-S　vCPU=1 平均時間:17.81 秒					
次數	1	2	3	4	5
	反應時間	反應時間	反應時間	反應時間	反應時間
VM1	18.44	17.42	17.47	18.01	17.65
VM2	17.90	18.19	18.02	18.14	17.51
VM3	17.55	17.73	17.74	17.52	17.89

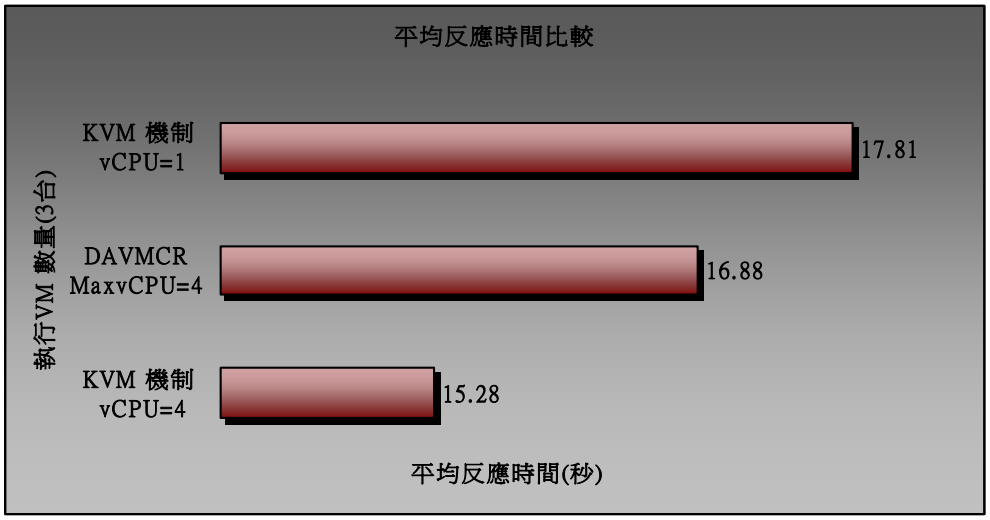

圖 4-6 三台 VM 平均反應時間圖

表 4-8 三台 VM 執行時間表。

VM-F vCPU=4 平均時間:78.27 秒					
次數	1	2	3	4	5
	執行時間	執行時間	執行時間	執行時間	執行時間
VM1	79.00	86.02	70.99	83.32	83.64
VM2	82.82	77.18	70.72	68.14	76.49
VM3	77.62	80.17	83.29	69.79	84.91
VM-D MaxvCPU=4 平均時間:87.24 秒					
次數	1	2	3	4	5
	執行時間	執行時間	執行時間	執行時間	執行時間
VM1	80.03	95.62	87.31	82.63	89.83
VM2	89.23	92.77	77.81	88.24	89.46
VM3	90.70	90.23	81.74	84.31	88.76
VM-S vCPU=1 平均時間:91.67 秒					
次數	1	2	3	4	5
	執行時間	執行時間	執行時間	執行時間	執行時間
VM1	94.25	94.35	89.25	91.99	90.98
VM2	91.27	92.60	92.63	92.80	89.43
VM3	90.22	91.21	92.09	89.90	92.12

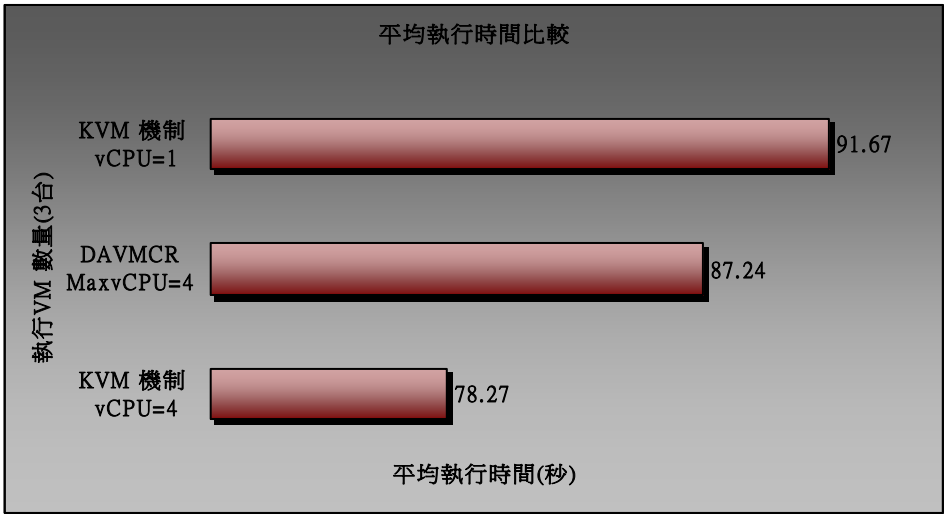

圖 4-7 三台 VM 平均執行時間圖

如表 4-9、表 4-10、圖 4-8 及圖 4-9 所示為同時執行四台 VM 之實驗結果其數據，在同時執行 4 台 VM 的狀態下，VM-S 型式下效能較佳，高於 VM-D 及 VM-F 約為 5%與 11%，與執行一台 VM 時之數據作比較，VM-F 和 VM-D 的平均反應時間與平均執行時間約下降約 72%與 51%，此時 VM-D 的 vCPU 數量約為 8 個，超過實體的 CPU 數量 4 個，因此也產生 VM 之間的 vCPU 競爭也越來越大，故效能上下降幅度也變大，而 VM-S 也略為下降 5%，雖因同時執行 4 台 VM 未造成 VM 之間 vCPU 間的競爭，但因有一個 vCPU 所對應到的實體 CPU 是屬於實體機器所使用的主要 CPU，因此必須與該實體機器共享 1 個實體 CPU 資源故效能略為下降。

表 4-9 四台 VM 反應時間表。

VM-F vCPU=4 平均時間:21.02 秒					
次數	1	2	3	4	5
	反應時間	反應時間	反應時間	反應時間	反應時間
VM1	21.27	21.46	20.83	25.32	19.40
VM2	21.54	20.63	21.50	18.62	22.06
VM3	17.82	20.13	19.26	23.14	22.63
VM4	20.24	20.41	20.05	23.09	20.91
VM-D MaxvCPU=4 平均時間:19.69 秒					
次數	1	2	3	4	5
	反應時間	反應時間	反應時間	反應時間	反應時間
VM1	20.62	19.66	19.34	18.66	23.30
VM2	19.25	19.41	24.10	23.23	18.42
VM3	19.07	20.33	17.77	17.50	18.41
VM4	18.50	20.18	20.12	18.32	17.65
VM-S vCPU=1 平均時間:18.62 秒					
次數	1	2	3	4	5
	反應時間	反應時間	反應時間	反應時間	反應時間
VM1	18.28	18.38	18.35	18.82	18.40
VM2	18.56	18.68	18.91	18.29	18.36
VM3	18.08	18.74	18.76	18.78	19.00
VM4	19.60	18.43	18.71	18.82	18.50

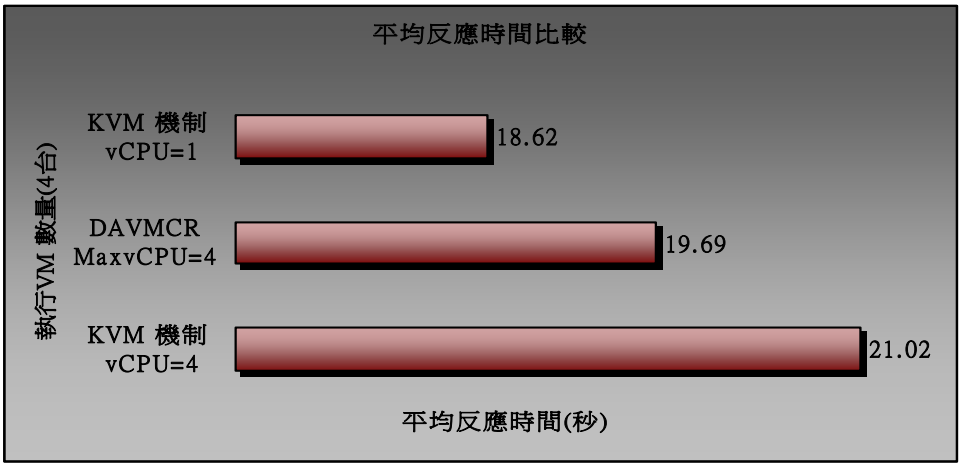

圖 4-8 四台 VM 平均反應時間圖

表 4-10 四台 VM 執行時間表。

VM-F vCPU=4 平均時間:109.41 秒					
次數	1	2	3	4	5
	執行時間	執行時間	執行時間	執行時間	執行時間
VM1	109.82	109.29	109.37	130.16	102.00
VM2	110.41	107.39	111.00	98.89	114.74
VM3	89.99	104.71	100.55	121.91	117.58
VM4	104.79	105.89	107.85	118.25	113.60
VM-D MaxvCPU=4 平均時間:102.45 秒					
次數	1	2	3	4	5
	執行時間	執行時間	執行時間	執行時間	執行時間
VM1	107.57	102.89	102.36	98.93	118.81
VM2	100.17	101.51	122.58	117.29	95.10
VM3	101.99	103.95	95.06	92.22	95.65
VM4	99.02	105.99	101.24	94.69	92.07
VM-S vCPU=1 平均時間:95.71 秒					
次數	1	2	3	4	5
	執行時間	執行時間	執行時間	執行時間	執行時間
VM1	93.30	93.76	93.74	95.78	93.87
VM2	95.39	96.03	96.80	94.06	94.66
VM3	93.21	96.40	96.47	96.73	98.71
VM4	100.77	94.40	96.61	97.55	95.89

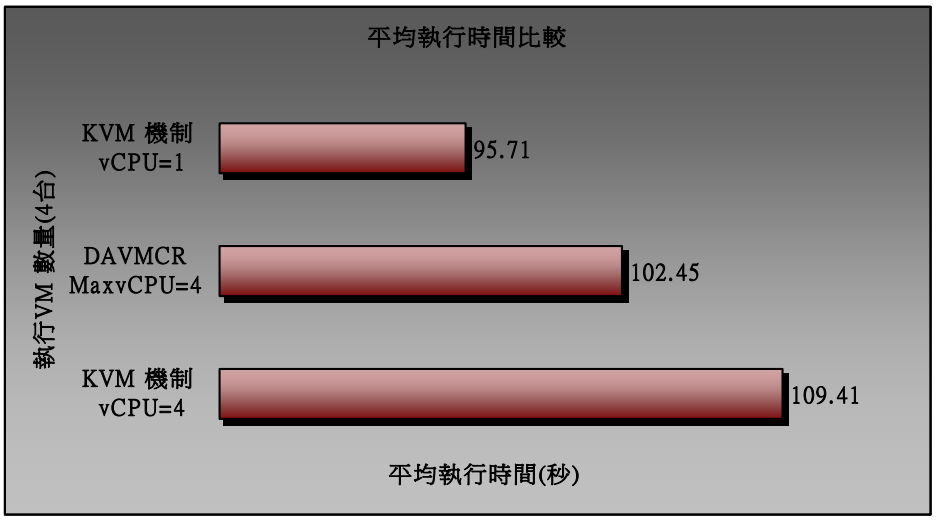

圖 4-9 四台 VM 平均執行時間圖