東海大學資訊工程研究所

碩士論文

指導教授：朱正忠 教授

具服務品質保證之動態調整雲端虛擬機資源之設計與實作

The Design and Implementation of Dynamic and Adjustment Virtual Machine in the Cloud of Service Assurance

研究生：蔡佩諭

中華民國 一〇四 年 六 月

東海大學碩士學位論文考試審定書

東海大學資訊工程學系 研究所

研究生 蔡 佩 諭 所提之論文

具服務品質保證之動態調整雲端虛擬機資源之設計與實作

經本委員會審查，符合碩士學位論文標準。

學位考試委員會

召 集 人 何信瑩 簽章

委 員 張志宏

楊朝棟

盧士傑

指 導 教 授 朱正忠 簽章

中華民國 104 年 6 月 23 日

致謝

時光飛逝，短暫卻充實的研究所生涯就要告一段落，從剛進入研究領域的懵懂，到現在多了些許知識，這都要感謝我的指導教授 朱正忠 老師孜孜不倦的教誨，以因材施教的教導方式令我在這段學習生涯中，不僅僅是知識的成長，更重要的是培養我解決問題的能力；佩諭也萬分感謝在口試期間給予我論文寶貴建議的張志宏 教授、盧志偉 教授、楊朝棟 教授以及何信瑩 教授。

我也非常感謝實驗室、軟工中心裡的同學與學長，還有系辦公室的姊姊們，在我研究所過程中的協助與幫忙，另外其他實驗室的Fanny Lin同學、Lady Xie學姊、Care Soung學妹與Mandy Tsai同學，時常與我打氣勉勵，使我在撰寫論文的這條路上並不孤單。

感謝我的父母對我的無限付出，讓我在研究所生涯中無後顧之憂的學習，在我低潮時給我鼓勵與打氣，在我懈怠時砥礪、提醒著我，以至讓我最後能夠順利完成本論文，另外非常感謝Ting God在這兩年時間，不厭其煩地給我許多不管是論文撰寫還是資訊相關知識學習上的許多建議與協助，讓我順利的完成研究所學業，與我一起度過這忙碌的時光，最後，我感謝在本篇論文中曾經幫助過我的老師與所有親朋好友，沒有你們的幫助就不會有這篇論文的完成。

蔡佩諭 謹誌於

東海大學資訊工程學系研究所

中華民國一○四年六月

摘要

傳統的視訊服務受限於單台伺服器上的資源，一旦出現較大的連線數量，往往會超過伺服器的負載量，導致封包的傳送出現延遲或遺失並使得影像的播放品質下降。

本論文提出具服務品質之動態調整雲端虛擬機資源的架構，將伺服器虛擬化，利用雲端計算中資源池的概念，在服務過載時，動態增加虛擬機資源改善惡化的服務品質。而當服務負擔較輕的時候，也能動態的減少虛擬機資源以降低成本。我們基於 KVM (Kernel based Virtual Machine)實現了上述的架構，並驗證其可行性。

本論文的實驗方法為 Monitor Manager 模組利用 ssh 協定進入每一台虛擬機查看使用率，設定平均 CPU 使用率 50% 以及平均連線數 50 爲閥值，當超過這兩者其中之一的時候，將通知 Dynamic Executor 調整虛擬機數目，當大量服務連入熱門網站，經由虛擬機器增加來改善服務品質，並以 Nginx 此負載平衡系統以及不同的負載平衡策略來實驗，藉以比較不同的負載平衡演算策略的效能。

關鍵字：雲端計算、服務品質、虛擬化、KVM。

Abstract

The traditional video streaming service is limited to the resource of one single server. If there is a large number of connections, it usually will overload the server's maximum load capability and the QoS of video service will decrease.

A mechanism of monitoring the QoS and dynamically adjusting virtual machines is proposed in this paper. According to the loading of video streaming, the number of virtual machines will change automatically to assure QoS. The adjustment mechanism is implemented based on the KVM as hypervisors and some experiments are conducted to assure this mechanism is effective.

The experimental method of our paper takes the advantage of the Monitor Manager module which checks the average usage rate of cpu and the number of connections as the threshold. When the cpu usage rate exceeds 50% or the number of connections exceed 50, the Monitor Manager module will send a message to the Dynamic Executor for adjustment of the number of virtual machines. Moreover, by deploying Nginx, a load balaner shipped with different algorithms of load balancing, we compare the performance of these different load balancing algorithms.

Keywords: Cloud Computing, QoS, Virtualization, KVM

目錄

圖目錄

表目錄

第一章 緒論

1.1 前言

雲端計算透過網路提供各種服務，使用者可利用簡便的筆記型電腦或行動裝置透過網路存取雲端資源與服務，例如儲存服務: Dropbox 和 Apple 的 iCloud 等，線上閱讀、學習與多媒體服務如:易讀網, NCTU OCW, YouTube, Facebook, Hulu 和、Netfix 等[1][2]。

隨著網路的普及，線上閱讀與學習或 VoD (Video on Demand) 的服務，已成為趨勢並且越來受到使用者歡迎。使用者可以透過網路隨時隨地存取電子書籍與線上學習等服務，且其文字與圖片、影像資料的儲存模式往往採取彈性擴展的雲端儲存資源，並有自動備份的機制確保資料不遺失。傳統設備如硬碟或光碟等作為儲存媒介，儲存空間有一定的限制，實體媒介若沒有好好妥善保存，讓其受到潮濕，發霉及灰塵的影響都將失去儲存的資料[3][4]。

雖然雲端運算能滿足各種形式的儲存需求，但是隨着雲端各種線上學習媒體越來越受到歡迎，連線數的增加使得伺服器的回應出現延遲，其各種學習媒體的服務品質也隨之降低。尤其網路傳播速度快的特性，當有熱門書籍或學習的課程出現的時候，往往瞬間湧入大量的連線，若伺服器無法快速調整，服務品質勢必快速降低，甚至無法服務。而當短暫尖峰一過，爲了因應尖峰而設置的伺服器將閒置，也會是資源的浪費。

近年來，虛擬化技術席捲整個 IT 產業，該技術(Virtualization)起源最早可追朔至 1960 年代，當時由於主機設備相當昂貴，因此主要用於共用分享大型主機硬體以提高硬體使用率，IBM 當時即採用此技術於單一主機上同時運行多個虛擬機(Virtual Machine, VM)，虛擬機共享實體主機資源，如：CPU, RAM, Storage, NIC 等。各個虛擬機可執行多個應用程式且互相獨立不受干擾，有效提升硬體資源使用率。雲端運算的成熟發展與硬體效能大幅提升，虛擬化是其核心技術的最大功臣，提供了像是災難回復、虛擬機搬移(VM migration)、計算與儲存資源的動態管理等特性，便於服務提供者打造客戶專屬環境。也帶給了 IT 管理人員與相關使用者諸多利益如下[5][6][6][8][9][10]: (1)將多個網路服務整合於單一伺服器有效降低硬體設備支出，且針對不同網路服務無需購置特殊硬體設備，能將額外的資金用來擴充現有的架構，(2)在環保議題方面，硬體設備的減少伴隨著電力

以及冷卻需求降低，(3)網路服務的集中化提供更彈性的管理介面，可將多個網路服務集中一個介面管理，亦或是獨自擁有，(4)網路服務分割能確保不同使用群組之間安全性、QoS 等。

利用虛化技術彈性拓展，與便於動態佈署需求資源的優勢，為即時性且需應付大量連線服務的各種服務帶來了更好且便利於提供服務的優勢，使得許多使用服務的相關者都受惠，像是第三方提供閱讀與學習媒體的供應者如易讀網, NCTU OCW 及 Netfix 提供給會員或線上的使用者更多觀看、閱讀與學習的多樣選擇，對於熱門電子書籍、線上學習也要能因應大量的觀看連線數，若消費者付費使用該服務當然不能接受，因該線上媒體服務太熱門而導致觀看服務有延遲，或是久久無法登入該網站享受觀看熱門媒體的服務，時間就是金錢不分消費者與第三方提供線上媒體服務的供應者，都相當依賴雲提供商在於雲資源的運作服務如: 運算、儲存和網路等資源的供應，若是資源的使用負載過高是否能動態調整需求資源，來應付消費者的需求與體驗品質(QoE)。線上媒體資源的需求越來越多，消費者對於該服務提供的服務品質(QoS)也要求越來越嚴格，雲設備在能源的消耗上也應該朝節省能源，降低環境資源的損耗為最終極目標，因此有效的雲資源的供應管理就非常重要，除了針對伺服器上的虛擬機器資源使用率監控，精準設定負載限制當虛擬機器的運作快達到所設定的閥值時，透過一個中介執行模組快速的動態啟動不同虛擬環境的虛擬機資源，佈署與分擔服務過載的虛擬機器。經由此運作模式將能大大的節省人力，且減少手動設定上易出錯與不必要的時間耗費，更是免除了不必要的時間、人力上的浪費，可提升相關設備的管理人員能更有效率與減少工作上的出錯率。若減少各種影響服務品質(QoS)的不良因素，將會大大提升使用者的使用服務的體驗品質(QoE)。

1.2 研究動機與目的

虛擬機之所以擁有高度安全及隔離性，全依賴於虛擬機監督器(Virtual Machine Monitor, VMM)做協調管理，並將其所管理的虛擬機模擬出一套虛擬硬體設備，如：CPU, RAM, Storage 等供使用，就如同真實電腦般執行作業系統及應用程式。

有鑑於服務品質在網頁伺服器服務上的重要性，本論文提出一個虛擬機資源負載與調度的架構，監控虛擬機的使用情況，由一個調度者根據系統負載的程度來調度資源，

並通知前端的 Load Balancer 後端可用的伺服器位置。藉由動態的計算資源負載及資源的調整，以增進服務品質。

1.3 研究架構與實作介紹

本研究進行的方式，首先是蒐集與研讀相關的文獻資料，以及了解實作具服務品質之動態調整虛擬機架構的開放原始碼虛擬化技術的虛擬化環境。並以 Hypervisor 的作業系統 Ubuntu 14.04，在其上裝設 KVM 虛擬系統以及 Libvirt 管理平臺，實作了具服務品質之動態調整虛擬機架構，並設定其應用環境爲網頁服務。此外，我們也將軟體負載平衡器安裝在 Hypervisor 上，我們使用的軟體爲 Nginx, Nginx 做爲負載平衡器具有優異的效能，也支援多種排程演算法，我們在實作上採用的爲 Round-Robin 方式。在虛擬機系統安裝網頁伺服器，我們採用 Node.js 做爲虛擬機內部的網頁伺服器，因爲其輕量及高效的特性，適合在資源較少的虛擬機內部使用。

藉由實作此架構來達到異質性虛擬環境的動態佈署以改善的管理、調配，經由動態調整虛擬機，將負載平衡器與後端的伺服器虛擬化，建構在資料中心的實體機器上。即使伺服器已經虛擬化，虛擬機仍需要手動開啓，且不同的虛擬機環境有不同的設定方法，手工耗費時間也容易易發生錯誤，此架構引入了 Monitor Manager 和 Dynamic Executor 兩個模組。Monitor Manager 監控各個虛擬機器的使用狀況，包含伺服器的收到的請求數目以及各個虛擬機器的 CPU 使用率等。Monitor Manager 根據這些資料判斷是否需要調整虛擬機數目，若需要調整，則通知 Dynamic Executor, Dynamic Executor 可處理不同的虛擬機系統。有別於傳統的網路視訊服務架構，以實體機器來執行串流服務，在服務的運作上較無彈性且在相關資源的設定上較耗費時間與人力，若有大量連線時使用者的體驗品質(QoE)會因為種種原因，像是後端伺服器資源增設的速度不夠快，因而造成網頁伺服器當機故障，相關人員來不及維護等種種問題發生，且當大量連線的尖峰時刻過後又空出一堆閒置機器，基於諸如傳統的網路服務架構的不足之處，故以具服務品質之動態調整虛擬機架構來取代傳統服務，來達到最即時的良好的服務品質(QoS)與體驗品質(QoE)。

1.4 章節導讀

本論文的章節安排如下所述：

第二章　爲相關研究與背景知識:敘述雲端計算、虛擬化技術與中介軟體。

第三章　系統架構介紹:傳統的網路媒體服務架構與具服務品質之動態調整虛擬機架構。

第四章　架構實作:實作了具服務品質之動態調整虛擬機架構，並設定其應用環境爲網頁伺服器服務。

第五章　結論與未來方向。

第二章 背景知識與相關研究

2.1 Cloud Computing

雲端所提供的服務可區分為三種服務類型:軟體即服務(SaaS)、平台即服務(PaaS)、架構即服務(IaaS)，如圖 2-1 所示[4](SaaS)軟體的集合，這些應用架構於基礎架構層提供的資源以及平台層提供的環境之上，並透過網路傳輸給用戶[1][4]。此層提供的應用可讓其使用者透過多元連網裝置(端)取用服務，僅需打開瀏覽器或連網介面即可，不再需要擔心軟體的安裝與升級，也不必一次買下軟體授權，而是根據實際使用情況來付費。而對應用開發者來說，他們可以方便地進行軟體佈署和升級，不需管理或控制底層的雲端架構，例如網路、伺服器、作業系統、儲存等，例如：YouTube, Gmail。(PaaS)雲端應用提供了開發、運行、管理和監控的環境，可說是優化的「雲端中介軟體」，優良的平台層設計可滿足雲端在擴充性(Scalability)、可用性和安全性等方面的要求。此層以用來支援佈署 Video-on-Demand(VoD)的服務，然而此類型的服務有更多服務品質(QoS)的需求[5]。(IaaS) 虛擬化後的硬體資源和相關管理功能的集合，透過虛擬化技術將運算、儲存和網路等資源抽象化，實現內部流程自動化和資源管理優化，進而向外部提供動態、靈活的基礎架構服務。此層的消費者使用處理能力、儲存空間、網路元件或中介軟體等「基礎運算資源」，還能掌控作業系統、儲存空間、已佈署的應用程式及防火牆、負載平衡器等，但並不掌控雲端的底層架構，而是直接享用 IaaS 帶來的便利服務，例如：Amazon [2][3]。

圖 2-1 雲端三種服務模式[4]

2.2 虛擬化技術

虛擬化技術就是利用虛擬化軟體(KVM,Vmware vSphere,Xen)將實體伺服器分割成多個運算資源，這些運算資源可獨立運行不同的作業系統、應用程式，彼此不相干擾，且可同時安全地存取實體機器上的資源，進而提高伺服器資源的有效利用率；而通常這一個個分割區，便是所謂的虛擬機器，它們都各自扮演了猶如硬體伺服器般的角色，當使用者需要大量的運算資源時便可增加虛擬運算資源因應需求，若於資源使用的離峰時段便可動態調整減少虛擬機資源的使用，可防止過多的閒置實體資源，透過虛擬技術來做資源的彈性調度，亦可用於跨伺服器資源的整合調度事宜，即能夠建立電腦運算資源的集合區，確保該集合區之內的所有應用服務，都可持續運作，假使這當中有任一虛擬機器出現故障,則由該虛擬機器所承載的商業應用系統,即可被轉移至另一虛擬化環境，而不會影響其正常運作。因此虛擬化技術本質就是一種資源管理技術,它將硬體、軟體、網路、儲存等硬體設備分離開來，讓使用者能更合理、充分的控制與管理更多的虛擬資源，所以虛擬化技術是雲端運算很重要的基礎建設 [8][9]。

開放原始碼(Open Source)上主流的虛擬化技術分為兩種模式，一種是平台虛擬化技術(Platform Virtualization)又稱硬體虛擬化(Hardware Virtualization)，另一種則是作業系統層級的虛擬化技術(Operating System-level Virtualization) [11]。

2.3 平台虛擬化技術

平台虛擬化技術(Platform Virtualization)又稱硬體虛擬化(Hardware Virtualization)，其架構是用軟體建立一個模擬真實電腦硬體的虛擬機器(Virtual Machine)，而原本執行在實際電腦硬體的作業系統，則運作執行在這個虛擬機器上，虛擬機器內運行的作業系統與實際的電腦硬體之間，會有軟體介面隔離控制這些虛擬機器對硬體的存取。底層實際硬體執行平台虛擬化系統的這一層稱作 Host，而運作在平台虛擬化系統之上的作業系統稱為 Guest，進行平台虛擬化的系統軟體則被稱作 Hypervisor 或虛擬機器管理員(Virtual Machine Manager)，Hypervisor 因為主要是在控制虛擬機器的運作，又被稱為虛擬機器監視器(Virtual Machine Monitor)，縮寫為 VMM [9]。平台虛擬化又可分為以下幾種的演進[12]:

- 完全虛擬化(Full Virtualization)
- 部分虛擬化(Partial Virtualization)
- 作業系統輔助虛擬化(OS Assisted Virtualization)，又稱作半虛擬化(Para Virtualization)
- 硬體輔助虛擬化(Hardware Assisted Virtualization)

2.3.1 完全虛擬化(Full Virtualization)

全部都虛擬化(I/O, Interrupt,指令集等等)，透過軟體來模擬虛擬的硬體環境，不需要硬體或 OS 的協助，而是直接透過 Hypervisor 實現，不用修改 OS 核心，因為 OS 不知自己運行在虛擬化的環境下，且 Guest OS Privilege Operation 需被 Hypervisor Intercept (攔截)，之後經過 Binary Translation 為 Machine Code，故效能較 Para-Virtualization 差。如圖 2-2 所示，其箭頭方向為全虛擬化環境的運作方式，當 Guest OS 需向底層硬體請求資源擴充的需求時，則會透過底層驅動程式轉給虛擬管理程序(Hypervisor)，接著 Hypervisor 會將所有 Guest OS 的需求經二元(Binary Translation)轉譯給底層的作業系統和硬體。目前全虛擬化的產品包括 Vmware Workstation, Virtual Box 等。

圖 2-2 完全虛擬化架構

2.3.2 半虛擬化(Para Virtualization)

半虛擬化又稱作業系統輔助虛擬化(OS Assisted Virtualization)，其作業系統輔助虛擬化並不模擬整個完整的硬體，而是提供一個虛擬化軟體介面供 Guest 作業系統使用，此軟體介面會直接與底層的硬體銜接，因此 Guest 作業系統的指令無須轉譯，因此運作效能會有很顯著的強化，但缺點是 Guest 作業系統須針對這個虛擬化介面進行修改，原因是 Guest OS 必須利用 Hypercall 呼叫 Hypervisor 對缺少的指令進行更換。作業系統輔助虛擬化的架構如圖 2-3 所示，箭頭方向為半虛擬化虛擬環境運作方式，修改後的 Guest OS 與特殊驅動程式配合虛擬化層，將需求直接對硬體轉給底層硬體與作業系統。採用這類技術的產品包括 Vmware ESX Server,Xen，能讓虛擬機器獲得更多的系統資源。

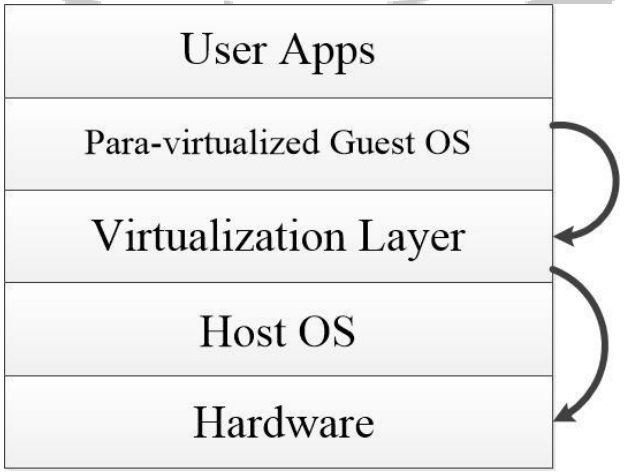

圖 2-3 半虛擬化架構

2.3.3 硬體輔助虛擬化(Hardware Assisted Virtualization)

硬體輔助虛擬化(Hardware Assisted Virtualization)是由中央處理器提供協助虛擬化的指令(Intel VT-x 與 AMD-V)，VMM 會將 Guest 作業系統的指令交由最底層中央處理器硬體處理，因此效能上當然大大勝過半虛擬化與完全虛擬化，Guest 作業系統也完全無需修改。硬體輔助虛擬化架構圖，其箭頭方向為硬體輔助虛擬化環境的運作方式，如圖 2-4 所示，Hypervisor 處於特權模式，可以將 Guest OS 的需求直接轉給底層硬體輔助虛擬化功能處理，不須做轉譯。代表這類型的虛擬化技術有 Vmware ESX Server 與 KVM(Kernel-based Virtual Machine)。

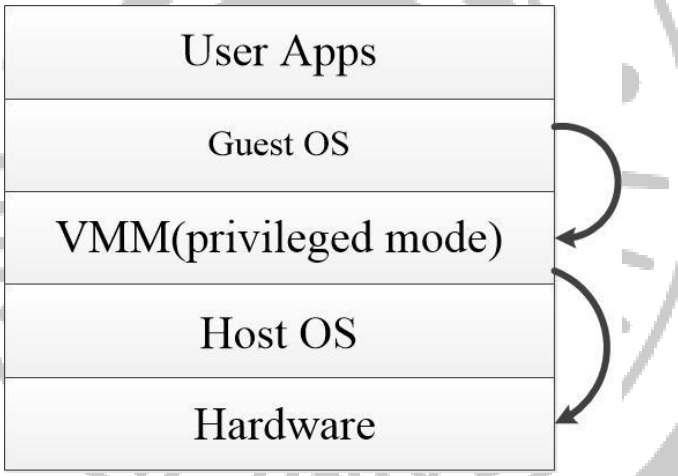

圖 2-4 硬體輔助虛擬化架構圖

2.3.4 VMM

依據 VMM 在系統虛擬化層次可分為三種型態[13]，(1) OS-Hosted VMMs，(2) Stand-alone Hypervisor VMMs，(3) Hybrid VMMs，如下圖 2-5 所示：

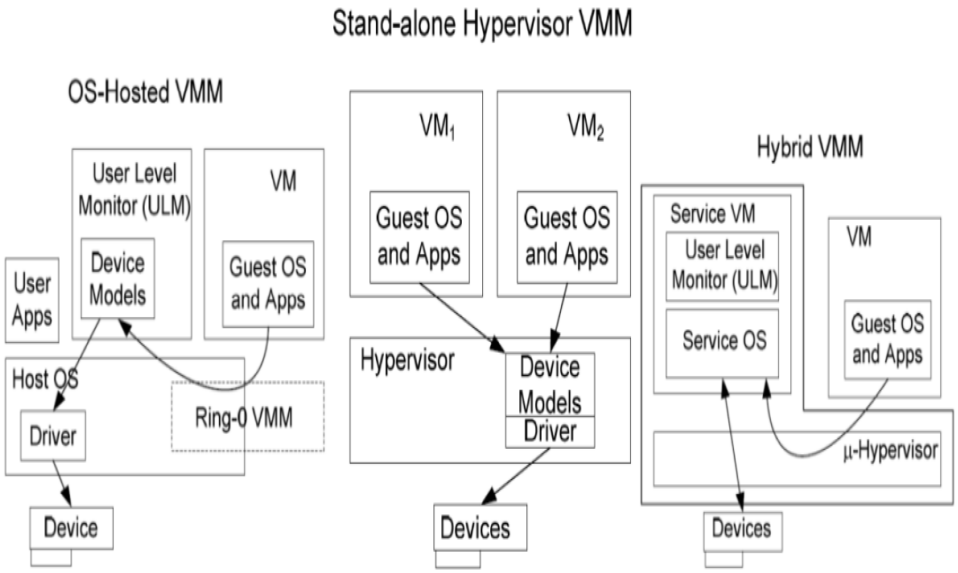

圖 2-5 三種 VMM 軟體架構示意圖[13]

- OS-Hosted VMMs：

  VMM 軟體架構建立於現有 Host OS 基礎架構之上，擁有如 Ring 0 般的特權能控制像 CPU、Memory 等系統資源，替一個或多個 Guest OS 建立執行環境，當 Guest OS 要存取 I/O 裝置會被 VMM 攔截並由 ULM(User-Level Monitor)代理，ULM 為 Host OS 上所運行的程序，包括虛擬 I/O 裝置主要服務來自 Guest OS 的 I/O 需求。

- Stand-alone Hypervisor VMMs：

  不同於 OS-Hosted VMMs, Stand-alone Hypervisor VMMs 不需依附於 Host OS, Stand-alone Hypervisor VMMs 自有 I/O 裝置驅動程式與排程器，能完整控制實體平台資源，替 Guest OS 提供如排程、QoS 等保證。

- Hybrid VMMs：

  結合 OS-Hosted VMMs 與 Stand-alone Hypervisor VMMs 的優點，在 Hybrid VMMs 的架構中主要以一個較小的 Hypervisor 控制 CPU 和記憶體資源，而 I/O 資源存取主要交由較低特權的 Service OS 中所運行的裝置驅動程式，由於 Service OS 能單獨代替 VMM 完成 I/O，可有效促進系統安全與可靠性。如:Xen,Citrix,Hyper VR2 等。

2.4 KVM 和 Libvirt

KVM(Kernel-based Virtual Machine)，此架構完全支援 CPU 的虛擬化技術(Intel VT-x 或 AMD-V)，而週邊裝置(網路卡、顯示卡....等)的虛擬化，則是透過 QEMU 這一套 Open Source 軟體作為虛擬週邊裝置與實體周邊裝置溝通的橋樑。目前的 Linux KVM，再建置與佈署的程序上，更優於其他裸機式的虛擬軟體，且利用 Linux KVM 也可輕易的達到快速且大量佈署虛擬電腦，即使要架構私有雲的雲端服務也可輕易達到，除此之外在備份、快速移機與異地備援也都可以輕易達到，雖比不上市面上的付費軟體但其靈活且彈性的運作，在雲端服務的領域上使用者亦需以企業主的內部需求為選擇優先考量[14]。

Linux KVM 透過 Libvirt 這個管理平台，來管理虛擬化平台所需的 API 函式庫。其最主要的目的，就是要提供一個單一的管理方式，來管理與操控各種不同的虛擬化平台所產生的虛擬電腦，以及不同的虛擬系統的管理程序。Libvirt 這套管理平台，也可以說是一組虛擬化應用程式的集合，包含管理者所需的管理命令，以及與虛擬化有相關的各種功能，像是虛擬化網路裝置(設備)，或是虛擬化儲存裝置，都屬於虛擬化的功能之一，整個 Libvirt 包含了三個部分[15]:

- 一套管理虛擬化平台所需的 API 函式庫
- 一個服務程序(daemon) : Libvirtd
- 一個命令管理工具(virsh)

Linux KVM 提供多種虛擬硬碟的檔案格式，使用者可以依照虛擬電腦的用途，選擇適合的檔案格式，來處理電腦上的資料，除了增加選擇彈性之外，適當的虛擬硬碟格式，則有助於提升虛擬電腦運作的效能。針對於日益減少的硬碟容量也可透過 KVM 的原生命令進行擴增。如圖 2-6 所示:

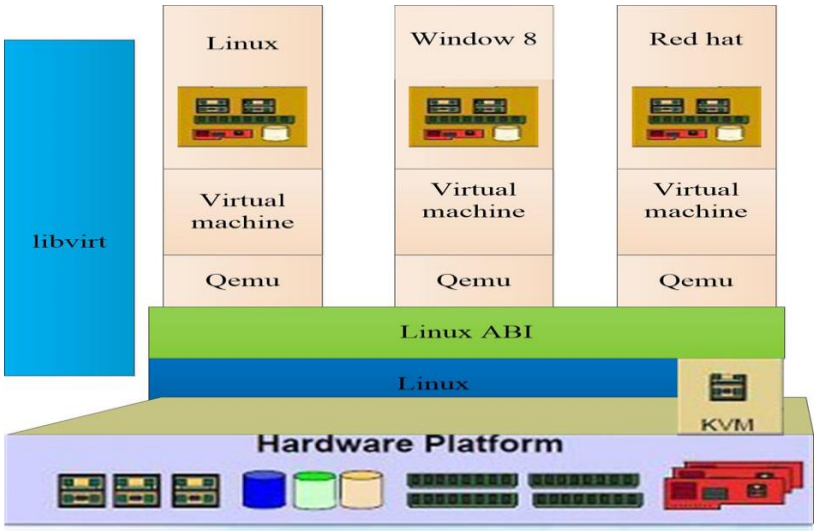

圖 2-6 KVM 架構圖

2.4.1 QEMU

QEMU 是由 Fabrice Bellard 撰寫之開放原始碼模擬器(Emulator)，其特點在於高效率以及可跨平台的動態翻譯(Dynamic Translation)程序，QEMU 支援兩種操作模式，用戶模式(User Mode)：QEMU 行程中提供了在 User Mode 對客戶機進行控制和管理的工具 ，系統模式(System Mode)：QEMU 能模擬整個電腦系統，包括中央處理器及其他週邊裝置，亦能用來在一部主機上虛擬數部不同虛擬電腦。

2.5 作業系統層級虛擬化技術

作業系統層級虛擬化技術的架構，則是藉由讓作業系統核心可以建立多個隔離的使用者空間運作實體(User Space Instance)技術來達成，使用者在這些隔離的使用者空間中運作，感覺就像在實際獨立的環境內運作一樣，而這些隔離的使用者空間運作實體(User Space Instance)又被稱為容器(Containers)、虛擬化引擎(Virtualization Engines,VE)、虛擬私有伺服器(Virtual Private Servers,VPS)。作業系統核心同時也提供管理這些運作實體使用資源的機制，避免單一運作實體使用系統資源過度，而影響到其他運作實體。由於作業系統層級虛擬化技術是由作業系統核心的介面所提供，因此無須像平台化虛擬化技術一樣去模擬整個電腦硬體，所以不需要有 Hypervisor，而沒有 Hypervisor 這中介層，整個虛擬化的負載也會比平台化虛擬化低很多。另外，當然也無需特殊的硬體支援來加速

虛擬化效能。

作業系統層級虛擬化因為與作業系統介面有關，因此虛擬化環境中無法運行異質的作業系統，以 Linux 為例，在 Linux 作業系統層級虛擬化環境中就無法運行 Linux 之外如 Windows、Solaris 等作業系統，只能運行以 Linux 核心為基礎的各式不同 Linux 發行版。代表這類型的虛擬化技術有 OpenVZ，其架構如下圖 2-7 所示:

OpenVZ 是相當早就開始發展的作業系統層級虛擬化技術，目前網路上流行的作業系統層級虛擬化私有伺服器(VPS)大部分都是使用 OpenVZ 技術，因此 Linux 核心中許多作業系統層級虛擬化的 Linux 容器(Linux Container,LXC)技術都是 OpenVZ 發展團隊所貢獻的。

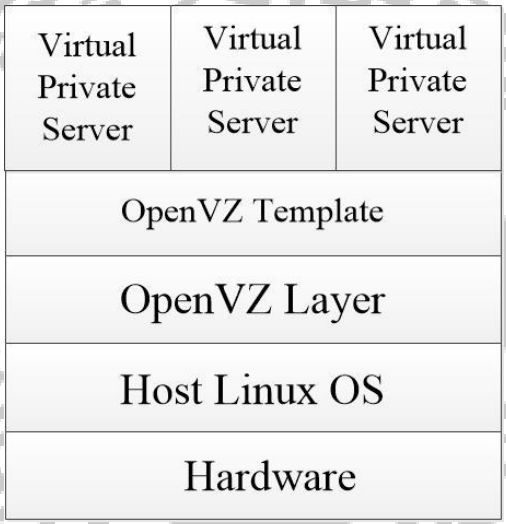

圖 2-7OpenVZ 架構示意圖

2.6 Middleware

在雲端運算技術中，[16][17][18][19][20][21][22] Middleware 位於服務與伺服器叢集之間，提供管理和服務。中介軟體之開發與研究著重在雲端運算環境中的系統整合並提供中介前置作業的服務，為應用提供統一的標準化程式介面和協定，將底層硬體隱藏起來、整合作業系統和網路由於這獨有之特性，軟體中介層能夠整合起具安全性普及計算環境的系統，以達到整合及管理中介體之應用程式之發展。如：

1. 一致的應用程式發展之支援。

2. 使用者管理－使用者身份驗證、許可、定制管理。

3. 資源管理－負載均衡、資源監控、故障檢測等。

4. 安全管理－身分驗證、存取授權、安全審計、安全防護等。

5. 映射管理－映射管理、佈署、管理。

雲端中介軟體相類似的研究其系統架構圖如下圖 2-8 所示:

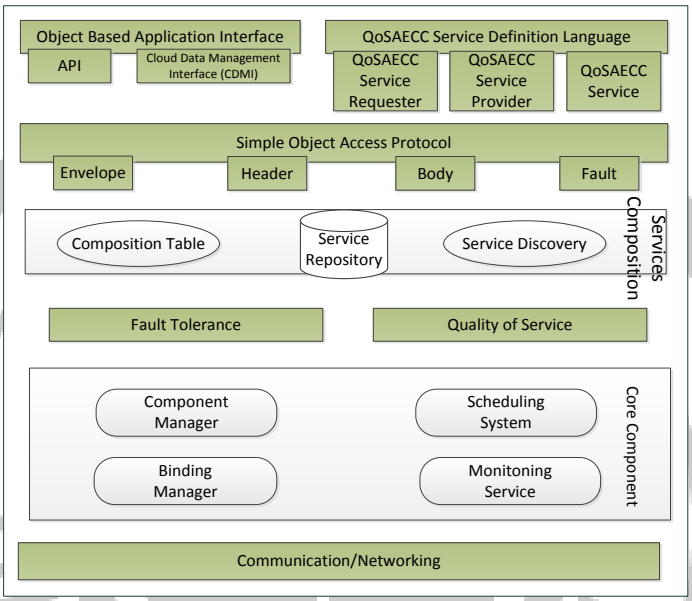

圖 2-8 企業雲端中介軟體架構

此系統架構圖如圖 2-8，結合不同功能元件，1.建立 Service Definition Language，定義包含有 Service Requester, Service Provider, Service 提供雲端運算這個較為複雜的環境以及多重契約與限制的定義描述。2.建立 Services Composition，平台中各服務組合元件化，利用不同元件程序應用開發者可以快速搜尋到開發者所需要的服務構成一個完整的系統。3.確保服務運作狀態，建立 Service Status Collection and Analysis，對應 PaaS 層確認資料存取狀況，並且對 SaaS 確認使用者所需要的服務狀態是否運作。4.Core Component 部份提供 IaaS 基礎架構環境進行資源監控。

此中介架構可結合不同功能元件，建構出需求服務相關說明如下:

1.建立 Service Definition Language，定義包含有 Service Requester, Service Provider, Service 提供雲端運算這個較為複雜的環境以及多重契約與限制的定義描述。

2.建立 Services Composition，平台中各服務組合元件化，利用不同元件程序應用開發者可以快速搜尋到開發者所需要的服務構成一個完整的系統。

3.確保服務運作狀態，建立 Service Status Collection and Analysis，對應 PaaS 層確認資料存取狀況，並且對 SaaS 確認使用者所需要的服務狀態是否運作。

4.Core Component 部份提供 IaaS 基礎架構環境進行資源監控

當使用者要求使用在雲端運算裡面的某些服務時，我們必須需要讓使用者知道系統能提供多高的資源讓使用者使用此服務，故需要系統的資源監控，隨時掌握目前可使用資源，Middleware 利用 Monitoring 的技術對系統進行監控，根據監控目標的差異設計相對應的代理人(Agent)，適時地將各種狀態回傳給中介軟體進行資料分析，針對不同服務提供對象設計 Agent，主要分為：

1. 底層硬體基礎設施 IaaS 服務專用代理人：收集實體機與虛擬機的資源使用量並動態改變其資源分配以達到效能與節能的平衡，並對系統進行異常行為的偵測及紀錄以協助安全監控。

2. 平台 PaaS 服務專用代理人：提供平台服務間的資料交換與同步，達到資料一致與共用的功能。

3. 軟體應用 SaaS 服務專用代理人：應用服務的控管與整合，協助提供終端裝置如智慧型手機應用程式服務請求交涉等功能。

代理人 Agent 監控架構如下圖 2-9 Middleware Monitoring with Agents 所示:

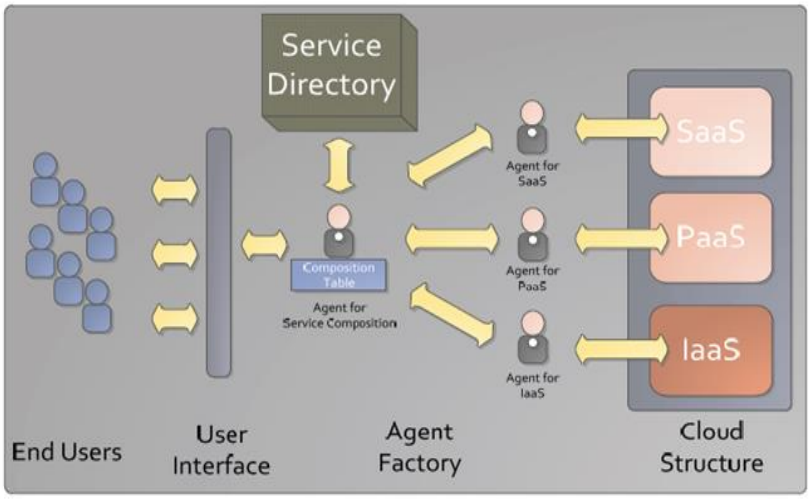

圖 2-9 Middleware Monitoring with Agents

監控類型可再細分為:

- Server Monitoring:

  為了讓系統管理人員可以更方便管理伺服器，當伺服器發生異常情況，為了維持伺服器之正常運作，系統將自動處理異常狀況，做出相對應的應變措施，並將 Log 及實施的應變措施結果回傳給管理者。

- Database Monitoring:

  用來監控資料庫的活動以及分析性能。

- Application Monitoring:

  監控應用程式及進程，它能收集相關資料、運行時間、記憶體使用量和 CPU 時間使用等等，在資源不足或資源過剩時做動態的資源調配處理，並將應用程式類型及資源調配結果寫回資料庫。

- Virtualization Monitoring:

  管理者可以對虛擬主機進行自動化管理，在統一的網管介面中同時對網路、實體主機與虛擬主機進行監控與管理，在資源不足或資源過剩時做動態的資源調配處理，並將記錄當時運算量及其得出的資料最少搬移量且負載最大平衡的解。

- Service Monitoring:

  1.可以指定每次檢查時涵蓋哪些服務是否有執行。

  2.當指定的服務狀態並非[執行中] 可以將其[啟動]

  3.當任何服務有問題時，會發 Email 通知指定的管理者，且通知對象可以多人。

服務的量測通常使用關鍵品質指標(Key Quality Indicators, KQI)和關鍵績效指標。(Key Performance Indicators, KPI)，針對明確的面向量測應用程式或是服務的性能，KQIs 可能來自多個來源，包含服務的性能指標或潛在的支持 KPIs，當服務或應用程式是由多種服務所組成，一些不同的 KPIs 可能需要計算某些特定的 KQI 才能確定，KPI 和 KQI 的映對可能很簡單或很複雜，映對可能是正規的，也可能是根據經驗的[23]。圖 2-10 KPI, KQI, SLA 之間的關係顯示 KPI,KQI 與服務水準協定(SLA)之間的關係，KPIs 可能被定義在 SLAs 中，而 KQIs 從服務層監控的程序中得到，每個 KPI 和 KQI 應包含通知警告與錯誤的門檻值。

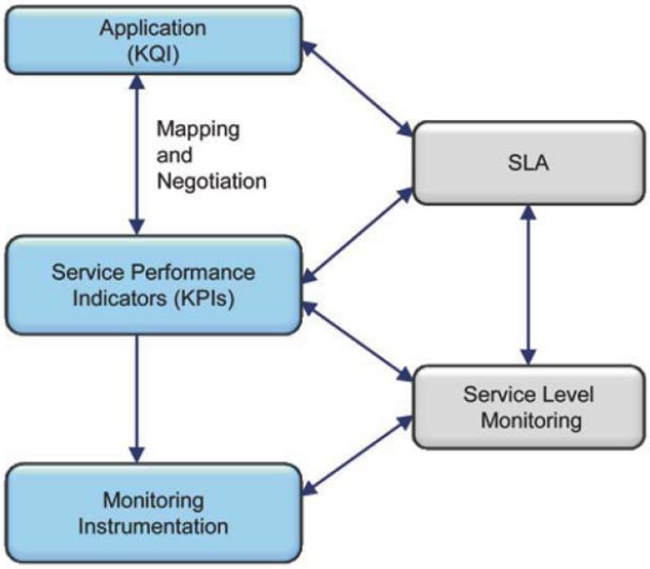

圖 2-10 KPI, KQI, SLA 之間的關係

2.7 Node.js

Node.js 是近來非常熱門的新興技術，它是一個網站後端開發框架，使用 Google Chrome 的 V8 JavaScript Engine 為基礎，也是一個建於 V8 JavaScript 引擎上的一套伺服端平台，採用 JavaScript 程式語言，相當輕量化且速度極快。其事件驅動(Event-Driven)及非阻斷式(Non-Blocking)I/O(輸入/輸出)的特性，用於建構高擴充性(Scalable)網站應用程式，很適合用於即時性的大量資料處理工作，目前許多大型網站(如 LinkedIn、微軟、Yahoo、eBay 等)都已經採用這樣的架構在進行服務[24][25]。由於 Node.js 速度快、系統資源消耗少，一般而言在相同的硬體配置下，使用 Node.js 可以提供更多的負載量；基於 Node.js 帶來的各種益處，它已經是許多 PaaS 平台支援的開發框架，以下是一些比較常見支援 Node.js 的 PaaS，其中包括知名的微軟 Windows Azure 平台。

- Cloud Foundry: Cloud Foundry 是開放源碼的 PaaS 解決方案，支援多種程式語言、開發框架及資料庫等服務，而且更容易開發、測試及佈署。
- Heroku[26]:是一個支持多種程式語言的雲平台即服務。Heroku 作為最開始的雲平台之一，一開始只支援 Ruby，但後來增加了對 Java,Node.js,Scala,Clojure,Python 以及 PHP 等程式語言。
- Joyent[27]:專門提供公有雲的服務，其雲端軟體的科技架構與其他主要雲端服務供

應商相當不同，其軟體特色為硬體利用率高、運算效能快與可擴增性強，因而能使服務營運商、企業與開發者更具競爭優勢。

- Nodester: 是一個開放的 Node.js PaaS(平台即服務)平台。

2.7.1 非阻斷式(Non-Blocking)I/O

非阻斷 I/O 在作業系統的運作方式為不需等待作業系統核心完成所有的操作後，呼叫才能結束。以讀取磁碟上的檔案為例，不需等待系統核心完成尋找磁碟、讀取資料與複製資料到記憶體，不會造成 CPU 等待 I/O，也不會浪費等待時間，因此 CPU 的處理能力也能得到充分的利用也因而可以提高運作性能。作業系統對電腦進行抽象化，將所有的輸入與輸出設備抽象化為檔案。作業系統核心在操作檔案 I/O 時，會透過檔案描述子進行管理，而檔案描述子類似應用程式與系統核心之間的憑證[28][29]。應用程式如果呼叫 I/O，非阻斷式 I/O 是不需要完成整個讀取資料的過程，採取不帶回資料直接返回，若想抓取資料還需要透過檔案描述子再次讀取。為呼叫阻斷 I/O 的過程如圖 2-11 所示。非阻斷 I/O 返回之後，CPU 的時間片段可以用來處理其他事務，明顯提升效率[30]。

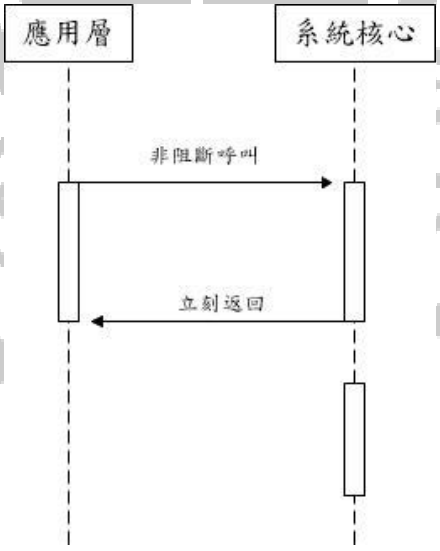

圖 2-11 呼叫阻斷 I/O 的過程[30]

非阻斷 I/O 為了取得完整的資料，應用程式利用輪詢技術重複執行 I/O 作業來確認作業是否完成。輪詢技術可以分為: read, select, poll, epoll 等技術[30]。

2.7.2 事件驅動(Event-Driven)

利用 Node.js 建構 Web 伺服器，透過主迴圈事件觸發的方式來執行程式。對於網路的處理 Node.js 也利用了非同步 I/O,Socket 監聽到的請求都會形成事件交給 I/O 觀察者。事件迴圈會不停處理這些網路 I/O 事件。利用 Node.js 建構 Web Server，正是在這樣的基礎上實現，其流程如圖 2-12 所示。

Node.js 為透過事件驅動的方式處理請求，無須為每一個請求建立額外對應的執行緒，藉以省掉產生和銷毀執行緒的成本，而且作業系統在調度任務時因為執行緒較少，上下文本切換的代價相當低。使得伺服器能夠有條不紊的處理請求，即使在大量連線的情況下，也不受執行緒上下文本切換的影響，此為 Node.js 高效能的一個原因。

事件驅動帶來的高效率已漸漸開始被業界重視。知名伺服器 Nginx 也捨棄了執行多年的執行緒的方式，採用和 Node.js 相同的事件驅動，其特性與 Nginx 相同，不同之處 Nginx 以純 C 撰寫開發，性能較好，但僅適合當成 Web Server，應用於反向代理或負載平衡等服務，在處理具體業務方面較為欠缺。未來 Nginx 大有取代 Apache 之勢。

Node.js 是一套高性能平台，可透過它建構與 Nginx 相同功能，也能處理各種具體業務，且與背後的網路保持非同步暢通 [30][31]。

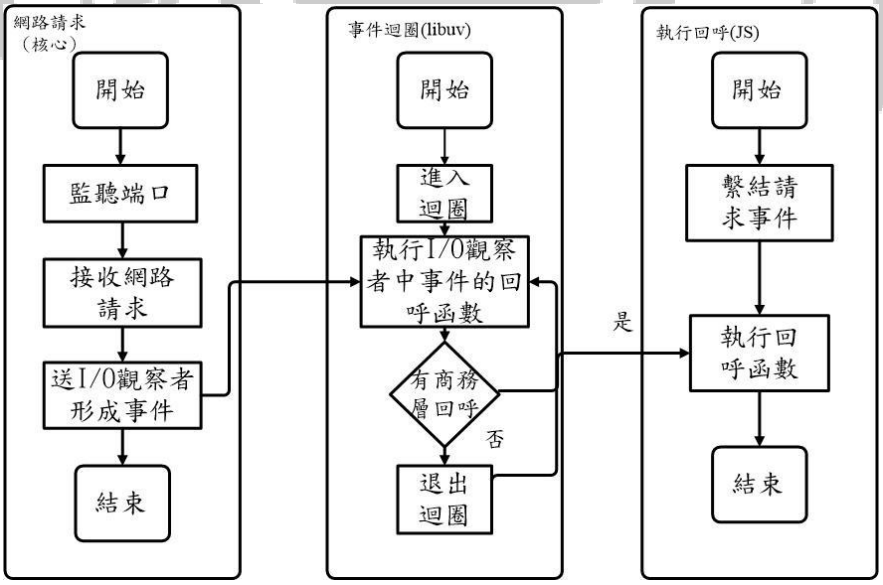

圖 2-12 利用 Node.js 建構 Web Server 的流程圖[30]

2.8 Restful Service

REST (Representational State Transfer-REST)是一種架構風格，基於一套原則描述網路資源如何被定義和定址，REST 是 Roy Fielding 長期在他的博士論文來敘述網路系統的結構風格[32][33]，當 REST 的架構會被拿來作為考慮時，通常具有以下的特性[34]：

a. 狀態和功能被切割在分散式的資源上

b. 每種資源具有唯一的位址，且採用統一且最小化的指令 (通常在網際網路上使用 Http 的指令- GET, POST, PUT, 和 DELETE)

c. 通訊協定是主從式的、無狀態的、階層式的和支援快取 REST 只是一種架構的風格，並不是標準，通常基於使用 HTTP,URI,XML 和 HTML 等，這些現有的廣泛流行的協議和標準。REST 可以用於 IaaS,PaaS,SaaS 三層中。

第三章 系統架構與模組

使用者透過各種裝置經由網路存取視訊服務。傳統的網路視訊服務架構，前端爲一台負載平衡器 (Load Balancer)，負載平衡器會將進來的請求(Request)分配到後方的數台固定數目的伺服器上，伺服器再將處理好的 http 結果透過負載平衡器傳回給使用者。如圖 3-1 負載平衡架構圖所示。

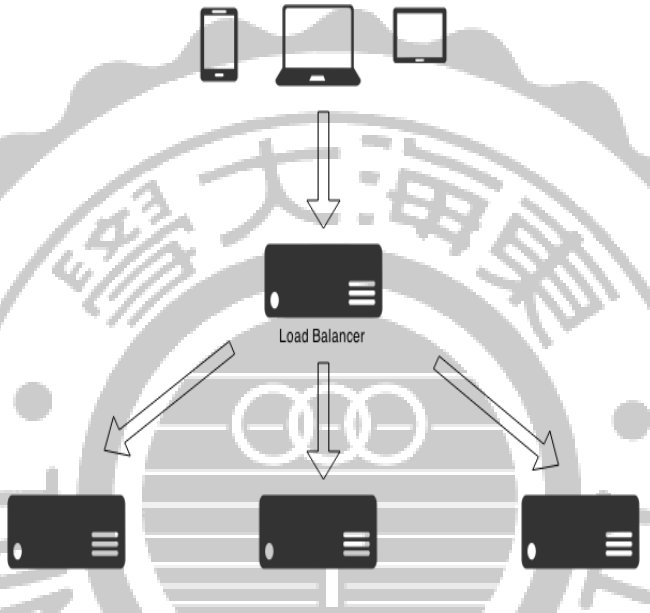

圖 3-1 負載平衡架構圖

此架構後端的伺服器數量是固定的，當請求的數目超過伺服器所能負擔的程度，服務品質便會下降，這時候則需要手動增加後方伺服器的數量並重新設定平衡負載器，往往需耗時數十分鐘到數小時之久，不僅耗費人力，使用者也需忍受長時間低落的服務品質。

爲了解決這個問題，我們提出了具服務品質之動態調整虛擬機的架構，利用虛擬化技術的彈性,再配合前端的負載平衡器做動態的調整,能快速的適應不同數目的需求量，以達到確保服務品質的目的。

3.1 具服務品質之動態調整虛擬機架構

使用者透過各種裝置經由網路存取視訊服務，視訊服務經由負載平衡機制供應給使用者。負載平衡器依照後方伺服器的負載狀況，將使用者的請求分配給後端的伺服器處理。爲了能動態的調整伺服器的數量，我們將負載平衡器與後端的伺服器虛擬化，建構在資料中心的實體機器上。

即使伺服器已經虛擬化，虛擬機仍需要手動開啓，且不同的虛擬機環境有不同的設定方法，手工耗費時間也容易易發生錯誤。因此爲了達到能動態調整虛擬機，此架構引入了 Monitor Manager 和 Dynamic Executor 兩個模組。Monitor Manager 監控各個虛擬機器的使用狀況，包含伺服器的收到的請求數目以及各個虛擬機器的 CPU 使用率等。Monitor Manager 根據這些資料判斷是否需要調整虛擬機數目，若需要調整，則通知 Dynamic Executor,Dynamic Executor 可處理不同的虛擬機系統，如：Ubuntu,Xen 或是 KVM 等，它會根據受到的指令，調整虛擬機的數目，此外，Dynamic Executor 也負責通知負載平衡器虛擬伺服器的增減和 IP 位置。此架構如圖 3-2 虛擬機資源負載與調度架構流程圖所示。

在之後的章節我們會進一步介紹負載平衡器，Monitor Manager 和 Dynamic Executor 等模組與其運作流程。

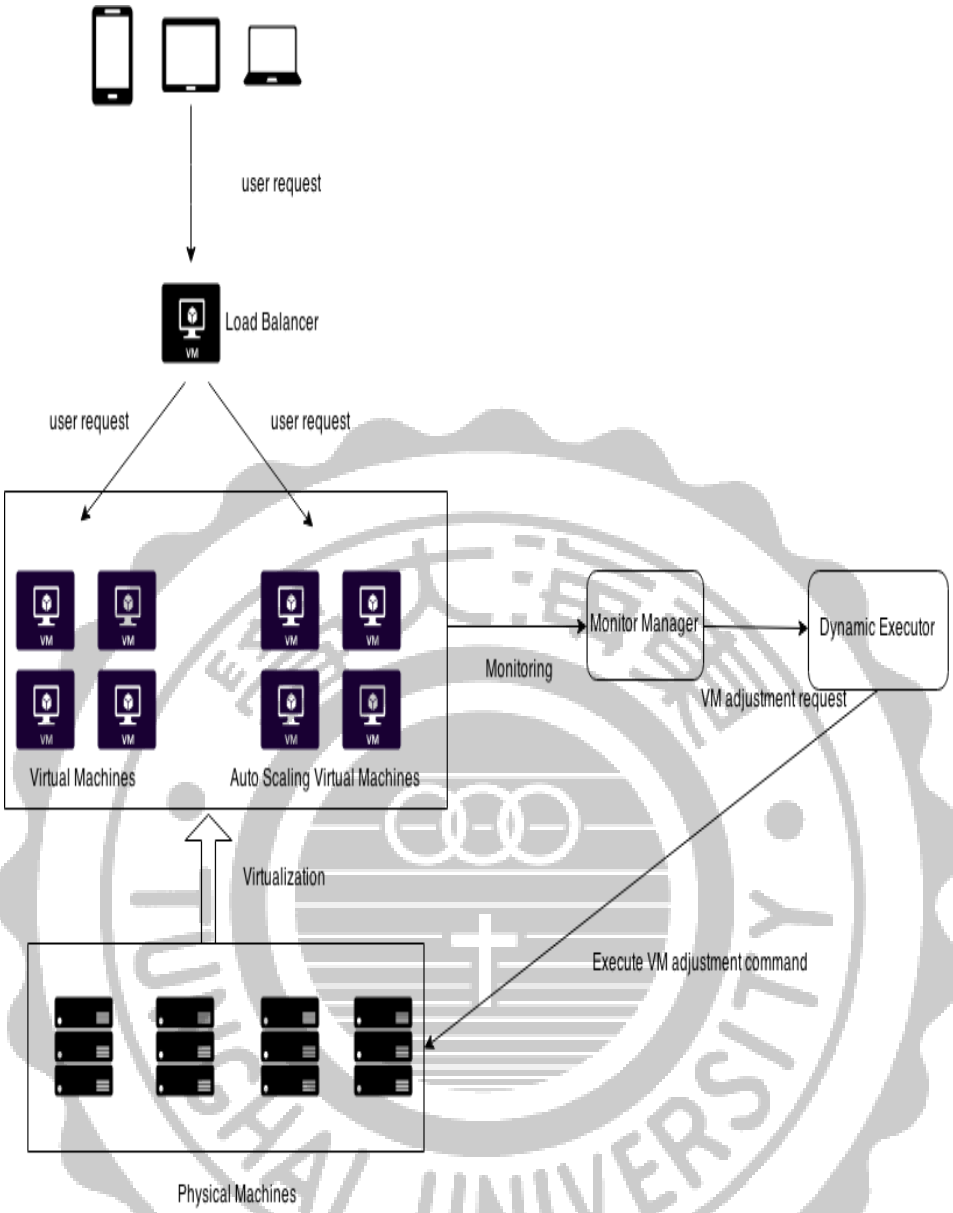

圖 3-2 虛擬機資源負載與調度架構流程圖

3.1.1 負載平衡器

此架構的負載平衡器將平衡網路連接的負載到各個虛擬伺服器上，以達到最佳化資源使用的目的。因此一個負載平衡演算法的好壞，將決定整個負載平衡機制的表現，設計不良的演算法將會影響負載的情況，雖有將工作分散到各台主機，但某些主機的負載可能高於其他的主機，負載伺服器爲了能平均分配請求，[35][36]採取的排程演算法可能有隨機、輪轉 (Round-Robin) 或根據最小連接數等方法。三種排程演算法介紹如下:

- 輪轉法(Round-Robin)：

  此演算法是所有負載平衡演算法最簡單，也是最容易實行的一種演算法，round-robin 會在每個節點中依循輪流選擇，將每一連線要求交由每個運作的虛擬機資源去處理，周而復始將連線分配到每個虛擬機設備之上，但是他並不會去注意後端虛擬伺服器上的連線數目和回應時間長短等情況，因為他把所有後端虛擬伺服器當成是相同效能。

- 隨機法(Random):

  以隨機分配的方式來分配連線的數量。

- 根據最小連接數:

  以伺服器的連線數來做分配，並將新進來的連線分配給負載最輕的伺服器。

三種排程演算法比較表如下表 3-1 三種排程演算法比較表如下所示:

表 3-1 三種排程演算法比較表[35]

三個因素	輪轉法 (Round-Robin)	隨機法 (Random)	根據最小連接數
可用性	1. 架設容易實現 2. 適用各主機處理能力一致時	1. 架設容易實現 2. 連線分配上可能會不均。	1. 架設容易實現 2. 各主機的負載不一定精確
效能	1. 不提供權重值調整 2. 負載程度可能輕者恆輕、重者恆重	1.採取隨機分配的方式，針對資源處理較無章法，後方虛擬資源容易分配不均	1. 依連線的狀況調整分配虛擬資源 2.亦會造成虛擬機資分配不均的狀況

經過如上表的比較與介紹，本論文係採輪轉法(Round-Robin)為我們的負載平衡演算法。

我們的負載平衡器是採取輪轉法來安排連線進來的 http 連線服務，因為輪轉法把所有後方的虛擬伺服器都視為同一效能，且不會去注意後端虛擬伺服器上的連線數目和回應時間長短等情況，負載平衡器僅負責分配連線進來的 http 連線服務。如上圖 3-2 虛擬

機資源負載與調度架構流程圖中我們也將負載平衡器虛擬化了，因此當有大量的連線服務進來時，若原有的負載平衡器無法負荷不斷連線近來的服務時，動態調整、增加負載平衡的虛擬資源以加快消化連線進來的服務分配至後端虛擬伺服器。虛擬化後的負載平衡器更與此架構中所增加的 Dynamic Executor 此模組相配合，因為負載平衡器虛擬化後雖然可以調節大量連線服務分配調度至後方虛擬伺服器，但是若沒有一個類似居中協調告知這些負載平衡器已經動態的調整虛擬機的數目與通知負載平衡器虛擬伺服器的增減和 IP 位置，若負載平衡器一直將服務送往負載過高的虛擬伺服器也無助於提升，大量連線服務時的服務品質。

3.1.2 Monitor Manager

Monitor Manager 爲此架構的重要模組，負責監控各個虛擬伺服器的負載程度，包含 CPU 使用率以及連線數。

監控連線數與 CPU 使用率的目的[35]是為了降低來自客戶端對系統所造成的負載量，在面臨大量用戶端連線收看影音時，導致後端伺服器無法負荷的情形發生，直接影響前端使用者的感受就是系統回應的效能與穩定性需要調整。

CPU 使用率和連線數都定義其所對應的閥值 (Threshold)，當使用率或連線數超過閥值，即代表負載程度上升到將影響使用品質時，Monitor Manager 會立刻發出調整虛擬機數量的命令給 Dynamic Executor。其流程如圖 3-3 所示，Monitor Manager 會查看後端虛擬伺服器的負載程度，當後端伺服器負載超過閥值則通知 Dynamic Exectuor 調整虛擬機資源，且調整完虛擬機資源後，接著修改負載平衡器關於後端虛擬伺服器的數目以及 IP 位置。若後端伺服器負載未超過閥值則不需調整虛擬機資源。

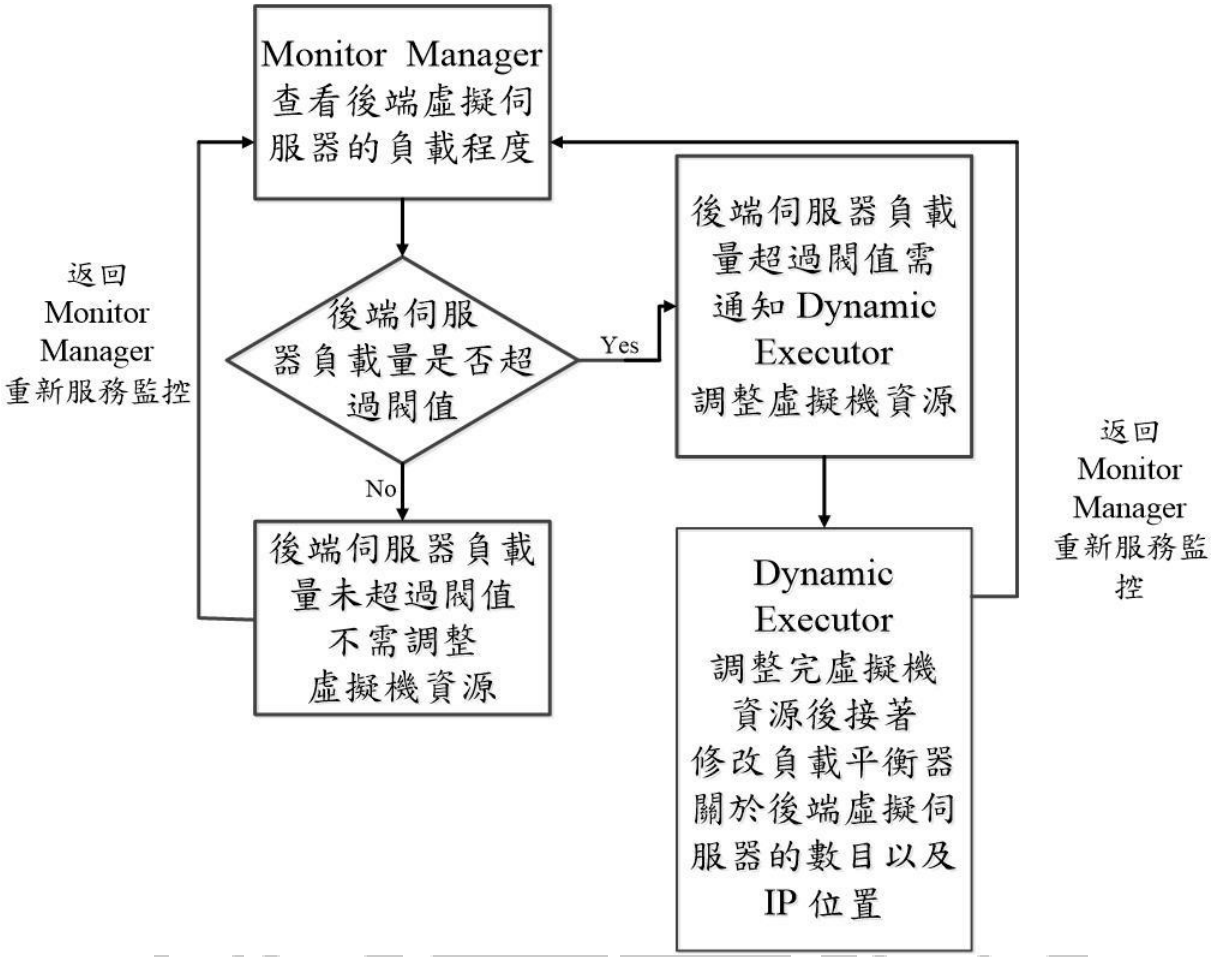

圖 3-3 Monitor Manager 流程圖

3.1.3 Dynamic Executor

Dynamic Executor 可視爲一中介軟體(Middleware)，它可消除異質性，但是目前底層只支援 KVM，並可透過 Dynamic Executor 來調整虛擬機的數目。

在調整虛擬機的數目之後，若負載平衡器不知道現在有哪些伺服器可使用，仍使用原來已負載過高的伺服器，負載程度將依然過高；因此在調整完虛擬機之後，Dynamic Executor 必須接著調整負載平衡器的設定，修改負載平衡器關於後端虛擬伺服器的數目以及 IP 位置。像支援多種虛擬系統一樣，Dynamic Executor 也必須支援常見的軟體負載平衡器，如 Nginx 或 Apache 等。另外若在調整虛擬機的數目或修改負載平衡器設定時發生問題，Dynamic Executor 需立即以電子郵件的方式通知系統管理者。

第四章 實作

本論文實作了具服務品質之動態調整虛擬機架構，並設定其應用環境爲 http 服務。經由實作此架構，負載平衡器是以 Nginx 為架設環境，我們以輪轉法為負載平衡演算，並透過該架構的兩個重要模組來實現當網頁伺服器的連線負載過重時，藉由 Monitor Management 與 Dynamic Executor 兩個模組分工合作，監控管理模組透過遠端連線查看虛擬機器的使用狀況，若連線數與 CPU 使用率超過閥值，就通知動態執行器啟動後端虛擬機器的調度，以提升使用者端在服務使用上的服務品質，並於虛擬機器的增加調度後，修改負載平衡器關於後端可用的虛擬機器的 IP 位置。

4.1 實驗環境

Hypervisor 的作業系統爲 Ubuntu 14.04，在其上裝設 KVM 虛擬系統以及 Libvirt 管理平臺。Hypervisor 環境如表 4-1 Hypervisor 環境所示。

此外，我們也將軟體負載平衡器安裝在 Hypervisor 上，我們使用的軟體爲 Nginx, Nginx 做爲負載平衡器具有優異的效能，也支援多種排程演算法，我們在實作上採用的爲 Round-Robin 方式。

表 4-1 Hypervisor 環境

CPU Cores	2.3 GHz Intel Core i5
Memory	16GB DDR3 1333 MHz
OS	Ubuntu 14.04 LTS
Virtualization	qemu-kvm 2.0.0
VM Management	Libvirt 1.2.2
Disk	75 G

Load Balancer	Nginx

在虛擬機方面，我們安裝的作業系統是 Ubuntu 12.04，因爲網頁服務的關係，需在虛擬機系統安裝視訊串流伺服器，我們採用 Node.js 做爲虛擬機內部的網頁伺服器，因爲其輕量及高效的特性，適合在資源較少的虛擬機內部使用。虛擬機的環境如表 4-2 所示。一開始的實驗環境會先開啟一台虛擬機，之後隨著負擔做動態的調整。

表 4-2 虛擬機的環境

CPU Cores	1
Memory	256 MB
OS	Ubuntu 12.04
http Server	Node.js 0.12.7

4.2 實作網頁伺服器

我們在 Node.js 平台上實作視訊伺服器，Node.js 爲單執行序、事件驅動(Event Driven)的平台，具有高效率的特性，非常適合用於單核的虛擬機環境。

Node.js 本身提供多種豐富的函數庫模組，例如 Crypto,HTTP,OS 或 REPL 等等。我們基於 HTTP 模組實作了網頁伺服器，使用者可以透過瀏覽器連入我們設置的網站。

每個 Node.js 伺服器能服務的連線數有其上限，因此透過負載平衡器以及動態調整虛擬機的機制，確保服務品質。

4.3 實作 Monitor Manager

Monitor Manager 利用 ssh 協定進入每一台虛擬機，查詢其 CPU 使用率，以及 Node.js 伺服器的連線數，之後計算出平均 CPU 使用率以及平均連線數。我們在實作上設定平均 CPU 使用率 50% 以及平均連線數 50 爲閥值，當超過這兩者其中之一的時候，將通知 Dynamic Executor 調整虛擬機數目。表 4-3 爲 Monitor Manager 模組的虛擬碼如下表所示。

表 4-3 Monitor Manager 模組的虛擬碼

```
1    for every minute do
2      total_cpu = 0
3      total_conn = 0
4        for every VM do
5          ssh into the VM
6          total_cpu = total_cpu + vm_cpu
7          total_conn = total_conn + vm_conn
8        end for
9        average_cpu = total_cpu / num_of_vm
10       average_conn = total_conn / num_of_vm
11       if average_cpu > 50 or
12       average_conn > 50 do
13       emit adjustment VM command to
         dynamic executor

     end for
```

4.4 實作 Dynamic Executor

Dynamic Executor 提供 Restful 界面讓 Monitor Manager 呼叫，當收到增加虛擬機的要求之後，使用 virt-clone 指令複製虛擬機映像檔，此虛擬機映像檔爲事先用 vmbuilder 指令設定好的 Ubuntu 12.04 系統，在複製完成之後，使用 virsh start 開啓虛擬機，之後調整 Nginx 負載平衡器的設定。負載平衡器的設定檔如表 4-4 所示。

表 4-4 Nginx 負載平衡設定

```
1    http {
2        upstream vms {
3          round-robin;
4          server 192.168.11.1;
5          server 192.168.11.2;
6          server 192.168.11.3;
7        }
8        server {
9          listen 80;
10         location / {
11           proxy_pass http://vms;
12         }
13       }
14   }
```

第五章　實驗與結果分析

5.1 實驗簡介

本論文在實作上是基於 Nginx 系統來做負載平衡，當有大量連線連入熱門網站造成負載增加，為改善惡化中的服務品質(QoS)，藉由自動開啟虛擬機器，以虛擬機器增加來達到負載平衡，進而改善使用者的體驗品質(QoE)。

本論文的實驗方法為 Monitor Manager 模組利用 ssh 協定進入每一台虛擬機查看使用率，設定平均 CPU 使用率 50% 以及平均連線數 50 爲閾值，當超過這兩者其中之一的時候，將通知 Dynamic Executor 調整虛擬機數目。我們以不同的負載平衡策略來實驗，套入我們的實驗方法藉以比較不同負載平衡策略與機制的效能比較。

5.2 實驗環境

以 Linux 為 Hypervisor 的環境，其作業系統爲 Ubuntu 14.04，在其上裝設 KVM 虛擬系統以及 Libvirt 管理平臺。Hypervisor 的環境如表 4-1 所示。

此外，我們也將軟體負載平衡器安裝在 Hypervisor 上，我們使用的軟體爲 Nginx, 我們在實作上比較了 Nginx 提供的不同的負載平衡演算法, 也就是輪轉法和根據最小連接數法。在虛擬機方面，我們安裝的作業系統是 Ubuntu 12.04，因爲網頁服務的關係，需在虛擬機系統安裝 http 伺服器，我們採用 Node.js 做爲虛擬機內部的 http 伺服器。因此如圖 5- 1 所示，我們設定的網頁服務爲一學術研討會網站。

我們利用 Siege 這套 Benchmark 工具來做壓力測試，使用 Siege 對實驗環境發出大量的 Request，並記錄其平均回應時間(Response Time)和可用率(Availability Ratio)。回應時間指的是 Request 從發出到收到回應的時間。可用率指的是有成功收到回應的 Request 佔全部發出去的 Request 的百分比，

$$availity = \frac{number\ of\ responded\ requests}{numer\ of\ total\ requests} \times 100\%。$$

我們分別對輪轉法和最小連接數法做實驗，並記錄這兩種演算法在不同虛擬機個數下產生的結果。

圖 5- 1 學術研討會網站

5.3 Round-Robin (輪轉法) 的實驗結果

如圖 5-2 所示，在三台機器使用輪轉法的回應時間測試中，此實驗結果請求連線在同時連線數達到 275 個的情況下的平均回應時間是 1.3 秒。一開始分別先以三台伺服器為實驗，後增加為五台伺服器進行實驗，並分別以不同的請求連線數進行回應時間、與可用率的測試。

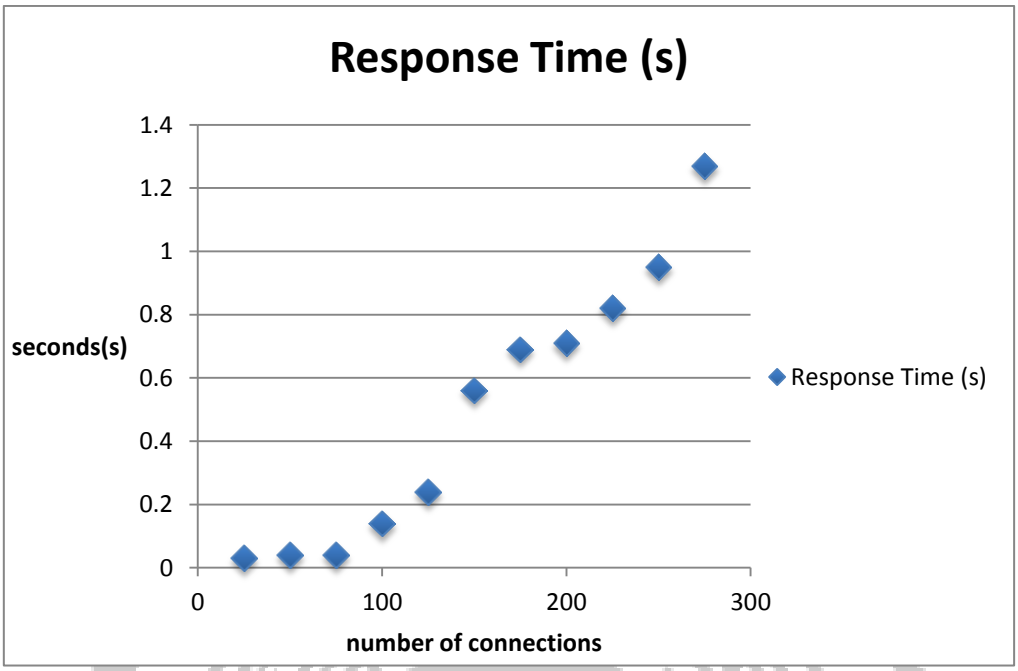

圖 5-2 三台機器使用輪轉法的回應時間

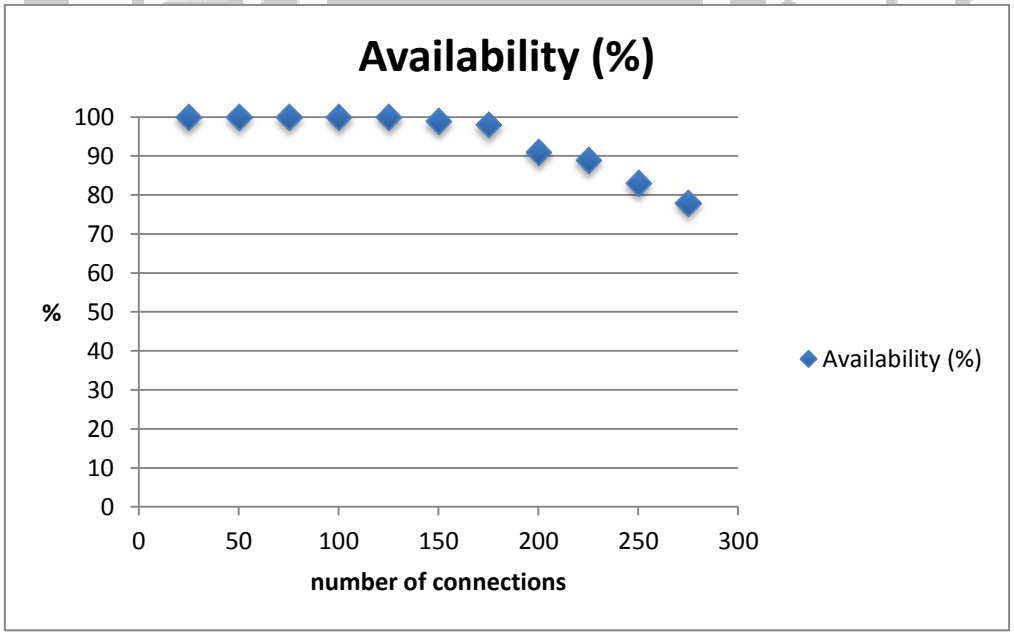

圖 5-3 三台機器使用輪轉法的可用率

如圖 5-3 所示，在三台機器使用輪轉法的可用率測試中，在同時連線數達到 275 個的情況下，其實驗結果是 78%。

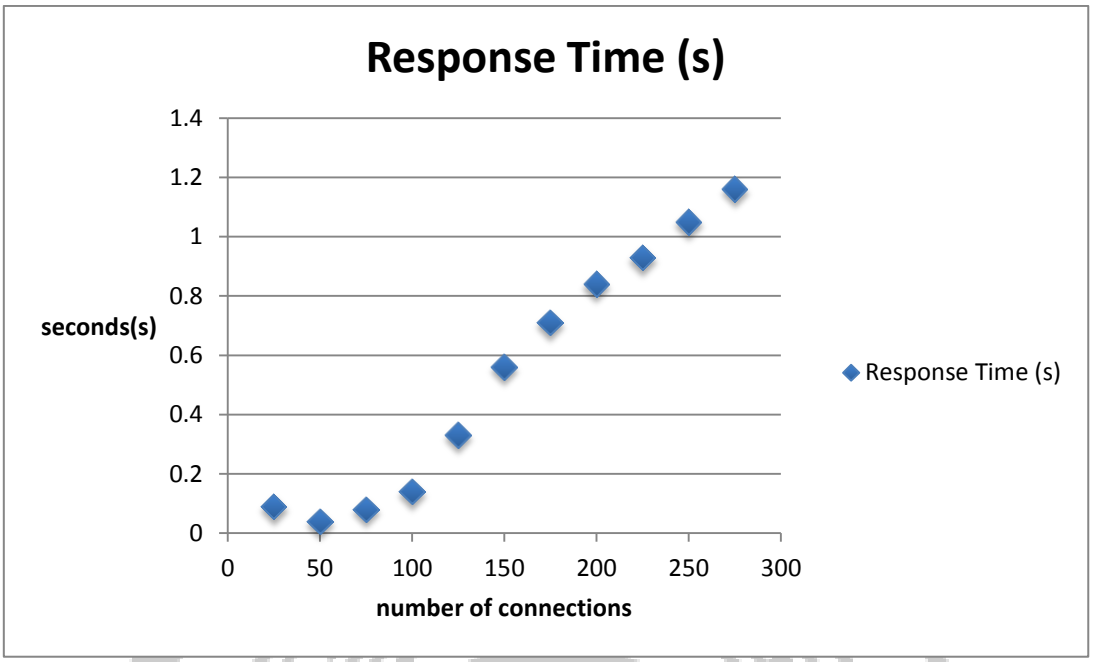

圖 5-4 五台機器使用輪轉法的回應時間

如圖 5-4 所示，在五台機器使用輪轉法的回應時間測試中，在同時連線數達到 275 個的情況下，其平均回應時間的實驗結果是 1.2 秒。

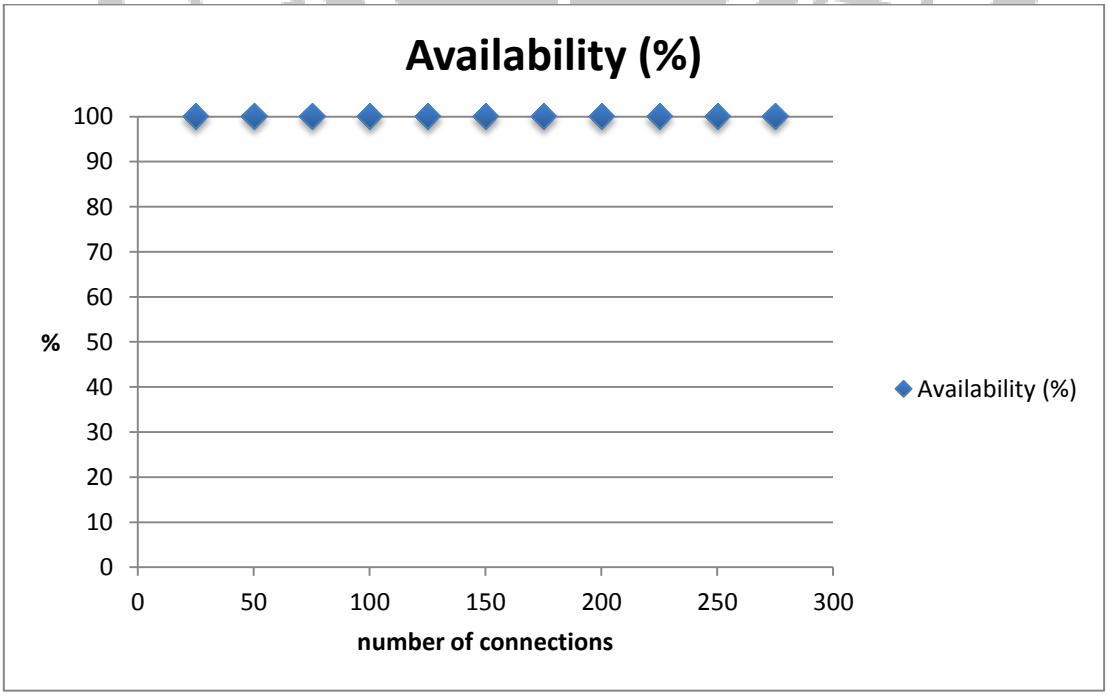

圖 5-5 五台機器使用輪轉法的可用率

如圖 5-5 所示，五台機器使用輪轉法的可用率實驗，其可用率提升到 100%，是因為虛擬機器增加故可用率相對也會提升。

5.4 Least-Conn (最小連接數法) 的實驗結果

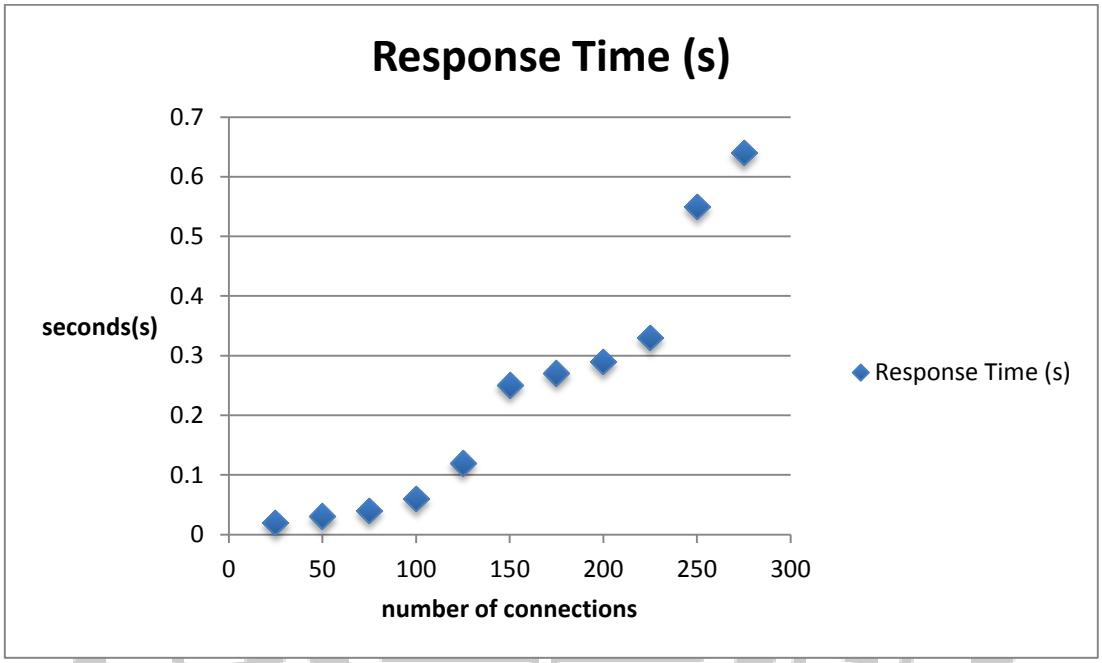

圖 5-6 三台機器使用最小連接數法的回應時間

如圖 5-6 所示，三台機器使用最小連接數法的回應時間，在同時連線數達到 275 個的情況之下其平均回應時間的實驗結果為 0.65 秒。一開始分別先以三台伺服器為實驗，後增加為五台伺服器進行實驗，並且分別以不同的請求連線數進行回應時間的測試並計算出其可用率。

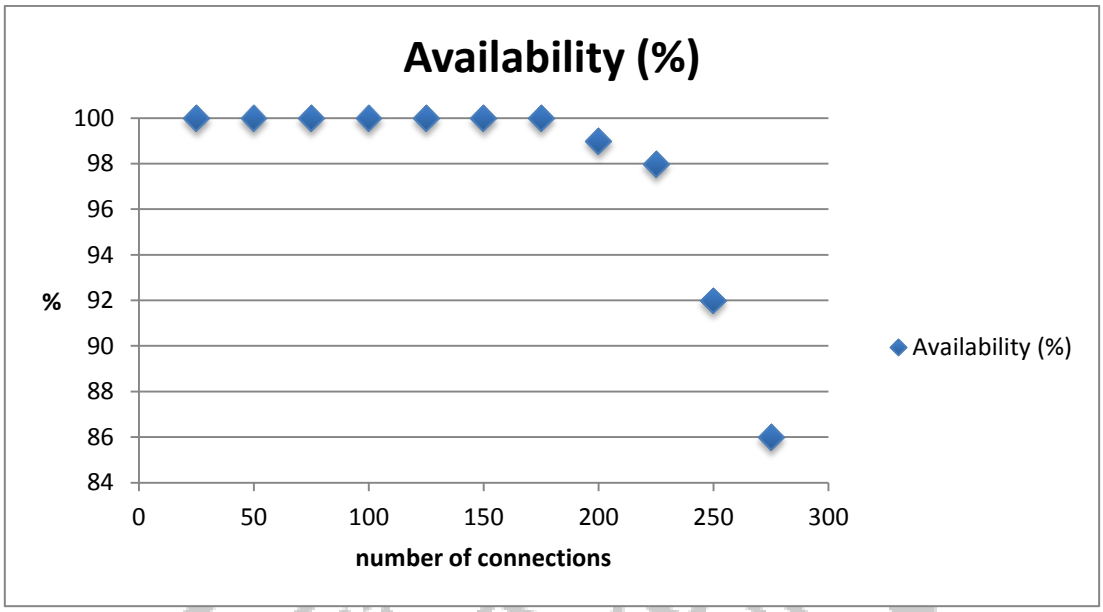

圖 5-7 三台機器使用最小連接數法的可用率

如圖 5-7 所示，在三台機器使用最小連接數法的可用率測試中，在同時連線數達到 275 個的情況下，此實驗結果的可用率為 86%。

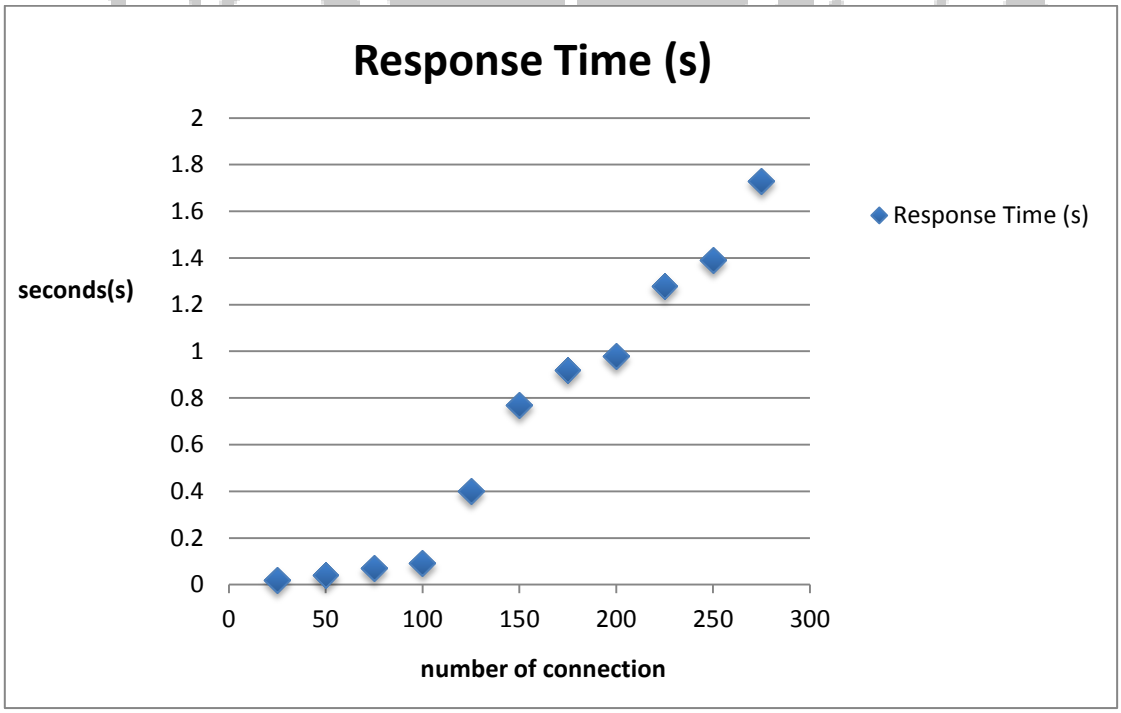

圖 5-8 五台機器使用最小連接數法的回應時間

如圖 5-8 所示，在五台機器使用最小連接數法的回應時間測試中，在同時連線數達到 275 個的情況下，此實驗結果請求連線的平均回應時間為 1.78 秒。

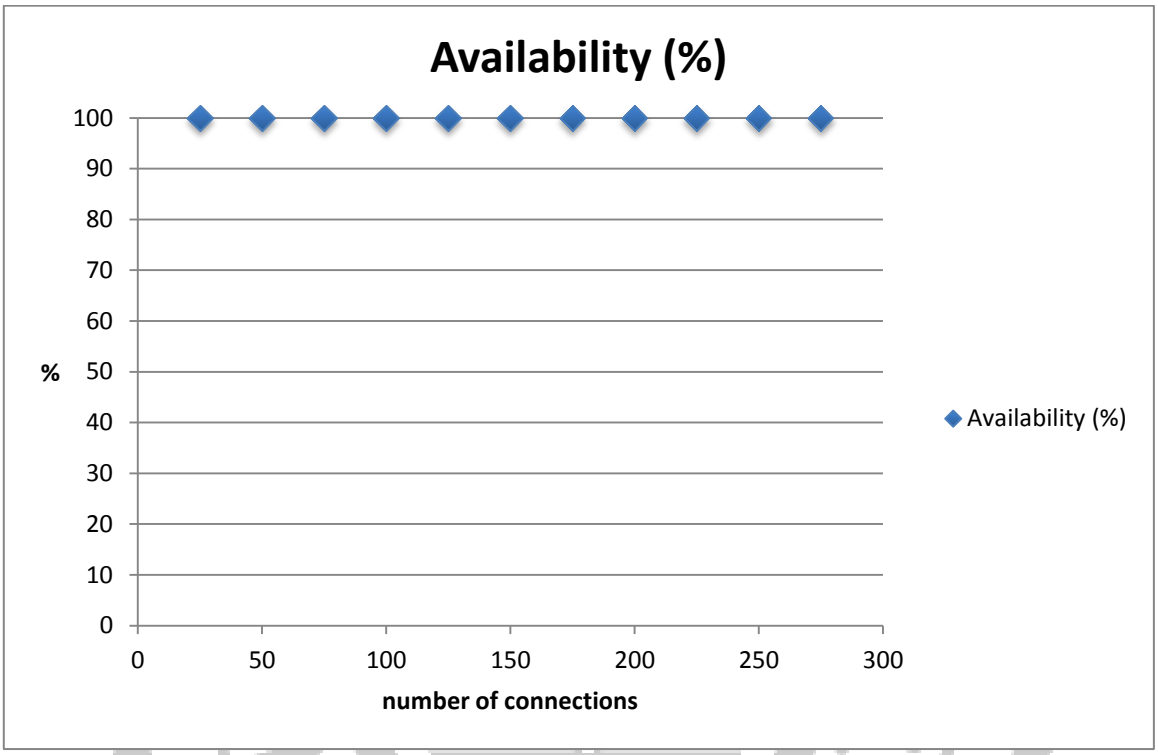

圖 5-9 五台機器使用最小連接數法的可用率

如圖 5-9 所示，在五台機器使用最小連接數法的可用率測試中，因為增加了虛擬機器故使用率從原本的 86%提升到 100%。

5.5 實驗結果分析

根據上述 Round-Robin (輪轉法)與 Least-Conn (最小連接數法)等負載平衡演算法的實驗結果如表 3-1 所述驗證，可知輪轉法會在每個節點中依循輪流選擇，將每一連線要求交由每個運作的虛擬機資源去處理，周而復始將連線分配到每個虛擬機設備之上，但是他並不會去注意後端虛擬伺服器上的連線數目和回應時間長短等情況，因為他把所有後端虛擬伺服器當成是相同效能。反觀最小連接數法以伺服器的連線數來做分配，並將

新進來的連線分配給負載最輕的伺服器。

我們可以從上述的實驗結果所示三台與五台機器，使用輪轉法與最小連接數法的回應時間與可用率，其兩種負載平衡演算法之實驗結果比較分析如表 5-1 所示:

表 5-1 實驗結果比較分析

三台機器 (275 個請求連線數)	輪轉法 (Round-Robin)	最小連接數法 (Least Conn)	備註
回應請求連線的時間(秒)	1.3 秒 (圖 5-2)	0.65 秒 (圖 5-6)	兩者因負載演算特性的不同，故連線請求的回應時間與可用率，以最小連接數法較好
可用率(%)	78% (圖 5-3)	86% (圖 5-7)	
五台機器 (275 個請求連線數)	輪轉法 (Round-Robin)	最小連接數法 (Least Conn)	兩者因負載演算特性的不同，故連線請求的回應時間以輪轉法較好，同時因為增加了伺服器處理連線服務，故可用率都達到了 100%
回應請求連線的時間(秒)	1.2 秒 (圖 5-4)	1.78 秒 (圖 5-8)	
可用率(%)	100% (圖 5-5)	100% (圖 5-9)	

從上表 5-1 可知輪轉法利於較多的伺服器來處理連線服務，並且可以藉由增加機器來提升連線服務的可用率。但相對於最小連接數法來說，增加了伺服器故在分配連線服務時也會較花時間。

本論文以具服務品質之動態調整虛擬機架構中的兩個重要模組 Monitor Manager 模組與 Dynamic Executor 模組相配合，可以與輪轉法與最小連接數法此兩種負載平衡演算法相配合，此實驗達到虛擬機器在越開越多的情況之下，讓可用率越來越高。

第六章　結論與未來方向

使用者透過各種裝置經由網路存取各種電子書籍與線上學習等服務，諸多的媒體服務由負載平衡機制供應給使用者。負載平衡器依照後方伺服器的負載狀況，將使用者的請求分配給後端的伺服器處理，傳統網站服務的架構已無法彈性調度與應付大量的使用者連線。

因此我們提出了具服務品質之動態調整虛擬機架構，該架構透過虛擬化技術可動態的調整伺服器的數量，並將負載平衡器與後端的伺服器虛擬化，建構在資料中心的實體機器上，但是即使伺服器已經虛擬化，虛擬機仍需要手動開啓，且不同的虛擬機環境有不同的設定方法，手工耗費時間也容易易發生錯誤，因此爲了達到能動態調整虛擬機的服務，具服務品質之動態調整虛擬機架構引入了 Monitor Manager 和 Dynamic Executor 兩個重要模組，藉由此兩模組解決異質性虛擬環境的動態佈署，經由動態調整虛擬機以改善管理、調配虛擬機資源。

未來更進一步將 Monitor Manager 修改成觀察者模式，藉由使用者訂閱模式[37]取代 ssh 連線的模式，以提升監控資源使用的效率，透過監控各個虛擬機器的使用狀況，包含伺服器的收到的請求數目以及各個虛擬機器的 CPU 使用率等，Monitor Manager 並根據所設定的閥值判斷是否需要調整虛擬機數目，若需要調整，則通知 Dynamic Executor, Dynamic Executor 扮演中介角色可處理不同的虛擬機系統，如：Ubuntu, Xen 等，它會根據受到的指令，調整虛擬機的數目，此外，Dynamic Executor 也負責通知負載平衡器虛擬伺服器的增減和 IP 位置。

本論文提出具服務品質之動態調整雲端虛擬機資源的架構，並完成此架構的實作，證明其可用於網站服務的可行性，此一動態調整資源的架構，將可降低人工的成本與增進效率，藉由實作具服務品質之動態調整虛擬機架構來讓網站達到最即時的良好的服務品質(QoS)與體驗品質(QoE)。我們將來希望能進一步以實驗方式推算不同環境開啓或關閉虛擬機所需的閥值，並能將此架構推行到其他種類的服務上。

參考文獻

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, “A view of cloud computing,” Communications of the ACM, vol. 53, no. 4, Apr. 3, 2010, pp. 50–58.

[2] G. Dhaker, S. Shiwani, “Auto-scaling, load balancing and monitoring as service in public cloud,” International Journal of Computer Engineering Research (IOSR-JCE), vol. 16, no. 4, Oct. 15, 2014, pp. 39-46.

[3] W. Zhu, C. Luo, J. Wang, and S. Li, “Multimedia cloud computing,” IEEE Signal Processing Magazine, vol. 28, no. 3, Apr. 21 ,2011, pp. 59–69.

[4] F. Sardis, G. E. Mapp, J. Loo, M. Aiash, and A. Vinel, “On the investigation of cloud-based mobile media environments with service-populating and QoS-aware mechanisms,” IEEE transactions on multimedia, vol. 15, no. 4, Mar. 16, 2013, pp. 769-775.

[5] W. Chen, J. Cao, and Y. Wan, “QoS-aware virtual machine scheduling for video streaming services in multi-cloud,” Tsinghua Science and Technology, vol. 18, no. 3, Jun. 3 , 2013, pp. 308–317.

[6] H. Luo, A. Egbert, and T. Stahlhut, “QoS architecture for cloud-based media computing,” Proceedings of the 3rd International Conference on Software Engineering and Service Science (ICSESS), June. 22-24, 2012, pp. 769–772.

[7] D. Armstrong and K. Djemame, “Towards quality of service in the cloud,” Proceedings of the 25th UKPEW Conference on Performance Engineering, Jul. 6-7, 2009, pp. 1-10.

[8] D. Kovachev and R. Klamma, “A cloud multimedia platform,” Proceedings of the 11th International Conference of the Multimedia Metadata Community on Interoperable Social Multimedia Applications, May. 19-20, 2010, pp. 61–64.

[9] L. Rudolph, “A virtualization infrastructure that supports pervasive computing,” IEEE Pervasive Computing, vol. 8, no. 4, Oct. 10, 2009, pp. 8–13.

[10] Cisco, Network Services Virtualization, http://www.cisco.com/c/en/us/products/collateral/switches/catalyst-6500-series-switches/white_paper_c11-531522.pdf, pp. 1-8, 2009, 瀏覽日期: Apr. 1, 2015.

[11] J. E. Smith and R. Nair, “The architecture of virtual machines,” Computer, vol. 38, no. 5, May. 16, 2005, pp. 32–38.

[12] Virtualization,http://www.netadmin.com.tw/article_content.aspx?sn=1412020002&jump

=3, Dec. 15, 2014, 瀏覽日期: Apr. 1, 2015.

[13] Virtualization, http://www.intel.com/content/dam/www/public/us/en/documents/product-specifications/vt-directed-io-spec.pdf, Oct. 3, 2014, 瀏覽日期: Apr. 5, 2015.

[14] M. Fenn, M. A. Murphy, J. Martin, and S. Goasguen, "An evaluation of KVM for use in cloud computing," Proceedings of the 2nd International Conference on the Virtual Computing Initiative, RTP, NC, USA , May. 15-16, 2008, pp. 1-7.

[15] KVM,http://www.linuxinsight.com/finally-user-friendly-virtualization-for-linux.html , Dec. 27, 2006, 瀏覽日期: Apr. 3, 2015.

[16] R. H. Campbell, M. Montanari, and R. Farivar, "A middleware for assured clouds," Journal of Internet Services and Applications, vol. 3, no. 1, Nov. 29, 2011, pp. 87–94.

[17] G. Coulson, P. Grace, G. Blair, L. Mathy, D. Duce, C. Cooper, W. K. Yeung, and W. Cai, "Towards a component-based middleware framework for configurable and reconfigurable grid computing," Proceedings of the 13th IEEE International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE), Jun. 14-16 ,2004, pp. 291–296.

[18] A. H. Ranabahu and E. M. Maximilien, "A best practice model for cloud middleware systems," Proceedings of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications, Oct. 25-29, 2009, pp. 1-10.

[19] S. Tambe, A. Dabholkar, and A. Gokhale, "Fault-tolerance for component-based systems-an automated middleware specialization approach," Proceedings of the 2009 IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC), Mar. 17-20 ,2009 , pp. 47–54.

[20] E. Wohlstadter, S. Tai, T. Mikalsen, J. Diament, and I. Rouvellou, "A service-oriented middleware for runtime web services interoperability," Proceedings of the 2006 International Conference on Web Services (ICWS), Sep. 18-22, 2006, pp. 393–400.

[21] C.-T. Yang, T.-T. Chen, K.-Y. Chou, and W. C. Chu, "Design and implementation of an information service for cross-grid computing environments," Proceedings of the 12th IEEE International Conference on Future Trends of Distributed Computing Systems (FTDCS), Oct. 21-23, 2008, pp. 99–105.

[22] H. P. In, C. Kim, U. Yun, and S. S. Yau, "Q-MAR: a QoS resource conflict identification model for situation-aware middleware," Proceedings of the 9th IEEE Conference on

Distributed Computing Systems (FTDCS), May. 28-30, 2003, pp. 212–218.

[23] The Open Group: SLA Management Handbook, The Open Group, Berkshire, vol. 4, 2004, pp. 1-8.

[24] Node.js, http://www.reader.idv.tw/2014/05/nodejs.html, May. 21,2014, 瀏覽日期: May. 1, 2015.

[25] Node.js,https://github.com/lyhcode/cloudfoundry30days/blob/master/day8.rst, Oct.25,2012, 瀏覽日期: May. 1, 2015.

[26] Heroku, http://www.inside.com.tw/2010/09/20/heroku, Sep. 20, 2010, 瀏覽日期: Apr. 1, 2015.

[27] Joyent,http://micloud.tw/ch/news/company-partners/item/231-joyent,2011, 瀏覽日期: May. 3, 2015.

[28] Node.js, http://dreamerslab.com/blog/tw/tag/node-js/,May.2012, 瀏覽日期: May. 1, 2015.

[29] 非同步 I/O, http://www.infoq.com/cn/articles/nodejs-asynchronous-io, Mar. 22, 2012, 瀏覽日期: May. 5, 2015.

[30] 朴靈,深入淺出 Node.js, 博碩出版社, pp. 25-357, 民國 103 年 7 月

[31] Node.js, http://www.ibm.com/developerworks/cn/linux/l-async, Sep. 28, 2006, 瀏覽日期: May. 3,2015.

[32] C. Yanping, L. Zengzhi, W. Li, and Y. Huaizhou, "Service-cloud model of composed Web services," Proceeding of the 3th International Conference on Information Technology and Applications (ICITA) ,vol. 2, Jul. 4-7, 2005, pp. 97–100.

[33] J. Hu, C. Guo, and P. Zou, "WSCF: a framework for Web service-based application supporting environment," Proceedings of the 2005 IEEE International Conference on Web Services (ICWS), Jul. 11-15, 2005, pp. 1-8.

[34] Restful Service, http://studyhost.blogspot.tw/2008/11/rest.html, Nov. 9, 2008, 瀏覽日期: May. 8, 2015.

[35] 陳信宏,包蒼龍, "網頁伺服器負載平衡架構效能之比較," 自由軟體與教育科技研討會, April. 2009, pp. 63-70.

[36] L. Battestilli, T. Nelms, S. W. Hunter, and G. Shippy, "High-performing scale-out solution for deep packet processing via adaptive load-balancing," Proceedings of the 18th IEEE Conference on Local & Metropolitan Area Networks (LANMAN), Oct. 13-14,

2011, pp. 1–6.

[37] 朱正忠,賴育彬,彭逸群,連文達,陳瑞男, “Design Pattern-Based N-tier Architecture for E-Commerce Systems,” Journal of Internet Technology (JIT), vol. 1, no. 1, Jan. 2001, pp. 81-89.

附錄一 動態調整虛擬機資源

- 建立複製虛擬機的樣本(vm01 為虛擬機複製的樣本)

```
vmbuilder kvm ubuntu \
--domain=kvm \
--dest=vm01 \
--arch=amd64 \
--hostname=vm01 \
--mem=256 \
--cpus=1 \
--user=ubuntu \
--pass=password \
--suite=precise \
--components='main,universe,restricted' \
--addpkg=acpid \
--addpkg=vim \
--addpkg=build-essential \
--addpkg=openssh-server \
--addpkg=avahi-daemon \
--libvirt=qemu:///system
```

virt-clone:複製虛擬機映像檔

virsh start:開啓虛擬機

virt-clone 以 vm01 此複本複製出 vm02，Dynamic Executor 動態的開啟虛擬機(virth start)，如下圖所示。

```
vmcsuser@VMCS-VM-g02350054:~$ sudo virt-clone --original vm01 --name vm02 --file vm03.img --auto-clone
Cloning tmp3G26Vd.qcow2                                                    | 942 MB     00:17

vmcsuser@VMCS-VM-g02350054:~$ sudo virsh start vm02
Domain vm02 started
```