國立交通大學

資訊科學與工程研究所

碩士論文

雲端運算下的節能負載平衡

Energy Aware Load-Balancing for Cloud Computing

研 究 生：王柏翔

指導教授：陳 健 教授

中 華 民 國 一百 年 七 月

雲端運算下的節能負載平衡

Energy Aware Load-Balancing for Cloud Computing

研 究 生：王柏翔　　Student：Po-Hsiang Wang

指導教授：陳　健　　Advisor：Chien Chen

國 立 交 通 大 學

資 訊 科 學 與 工 程 研 究 所

碩 士 論 文

A Thesis

Submitted to Institute of Computer Science and Engineering

College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science

June 2011

Hsinchu, Taiwan, Republic of China

中華民國一百年六月

雲端運算下的節能負載平衡

研究生：王柏翔　　指導教授：陳　健

國　立　交　通　大　學　資　訊　科　學　與　工　程　研　究　所

中文摘要

雲端運算是近年來熱門的議題，透過虛擬化的機制讓虛擬機器共享同一台實體機器上的硬體資源進而提高實體機器上的資源利用度。然而，雲端資料中心需要負載平衡來避免系統中因為某些實體機器上的負擔過重，導致執行效率降低、硬體錯誤甚至當機的情形發生。我們在這篇論文中提出一個將雲端系統中的負載平衡問題轉換成一個權重雙分圖，再透過 Hungarian 演算法求解其上之最小權重的一組配對，依照得到的配對，平行的將虛擬機器遷移到負擔較低的機器上。一次的配對可以處理系統中多個負擔過重的實體機器來減低系統達到平衡的時間，降低虛擬機器間競爭資源的情形，進而提升系統的產能。我們利用 CloudSim 這個模擬器來測試我們演算法的效能，實驗的結果顯示我們在各種資源負載平衡的狀態之下，不論是系統達到平衡的時間以及各個虛擬機器上應用程式的執行完成時間皆大幅的下降。我們更利用 XCP 雲端平台來搭建我們自己的小型雲端環境來證明我們的演算法在實際的雲端平台上也可以使用。此外，雲端的資料中心為了提供大量運算環境而需要許多伺服器，這些伺服器同時運行造成了大量的能源消耗和碳的排放量。所以我們改良我們的負載平衡演算法，當系統中的工作量降低時，去評估目前需要的伺服器數目，並將多餘的伺服器關閉，在負載平衡的同時達到節能的效果。

關鍵字:雲端運算、負載平衡、虛擬機器、實體機器、Live Migration、雙分圖、CloudSim、XCP

Energy Aware Load-Balancing for Cloud Computing

Student: Po-Hsiang Wang **Advisor: Dr. Chien Chen**

Institute of Computer Science and Engineering
National Chiao-Tung University

Abstract

Recent movement in cloud computing is showing the trend that more and more companies start to deploy their services on the cloud instead of worrying about the troublesome server configuration and administration by themselves. This trend also helps in consolidating hardware resource usage because the same set of machine resources can serve multiple companies. Through a virtualization technique, different companies' virtual machines (VMs) could share hardware resources on a single physical machine to improve the resource utilization. In order to maximize resource utilization, a load-balancing mechanism for cloud computing is needed to avoid performance degradation, hardware error or failure due to some overloading physical machines. Load balancing for cloud computing can be performed by migrating a VM from an overloaded host to an underutilized host. Since VM migration time are proportional to the amount of physical memory allocated to the VM. In a cloud data center with tens of thousands of physical machines and hundreds of thousands of VMs, one by one VM migration may take long time to reach a system load equilibrium state. In this paper, we propose an algorithm that transforms the load-balancing problem into a minimum weighted matching problem of a weighted bipartite graph. According to the minimum weighted matching obtained from Hungarian method, we concurrently migrate VMs from overloaded hosts to underutilized hosts. We use the CloudSim toolkit to test our algorithm's performance and the experimental results show that our algorithm not only obtains a good load balance but also reduces the time to reach system load equilibrium state. We also build a cloud development platform via XCP OS to prove that our algorithm could be used in a realistic cloud environment. Furthermore, a major cause of energy inefficiency in a cloud datacenter is the idle power wasted when servers run at low utilization. Therefore, we modify our load-balancing algorithm to migrate all VMs out of low utilized hosts, then to turn off them to save energy during load-balancing process.

Index Terms- Computing; Load-Balancing; Virtual Machine; Physical Machine; Live Migration Bipartite Graph; CloudSim; XCP

誌謝

感謝這兩年來給予我協助與勉勵的人，才有這篇論文的完成。首先要感謝我的指導教授 陳健博士，在研究的過程中指引我正確且有效率解決問題的路徑，在這段期間遭遇到的挫折，陳老師對我的指導與教誨都能在遇到困難時另尋突破，並且耐心地與我一起解決遇到的難題完成本篇論文，在此表達最誠摯的感謝。同時也感謝我的口試委員，交大的王國禎、易志偉教授以及朱煜煌博士，他們提出了許多寶貴的意見，讓我受益良多。

感謝與我一同努力的陳坤定學長，在研究的過程中互相討論以及提供實驗上的機器佈署方式及實驗方法，使我能突破瓶頸，讓研究更為完善。另外我也要感謝實驗室的學長、同學以及學弟張哲維、陳盈羽、鄭元碩、彭宣翰、張大鈞、蔡世仁以及黃俊憲等人，感謝他們陪我度過這兩年辛苦的研究生活，在我需要協助時總是不吝伸出援手，陪我度過最煩躁與不順遂的日子。

特別感謝我的朋友，林鐘基、孫碩英、李維元及許多大學、以及高中時代的好朋友們，在精神上給我莫大的鼓勵，傾聽我內心的聲音並帶給我無比的溫暖，指導我做出了許多正確的抉擇，而不致迷失了自我，在研究之外帶給我許多的樂趣。一同陪伴我度過最後的學生生涯，並留下許多美好的回憶。

最後，我要感謝家人對我的關懷及支持，他們含辛茹苦的栽培，使我得以無後顧之憂地專心於研究所課業以及研究，我要向他們致上最高的感謝。

目錄

圖目錄


E S A
1896

Chapter 1、簡介

雲端運算(cloud computing)[1]是近年來熱門的研究議題。透過虛擬化(virtualization)的機制讓虛擬機器共享同一台實體機器(physical machine)上的硬體資源而可以同時執行許多虛擬機器(virtual machine)和安裝不同的作業系統。同時，平行執行於實體機器上的虛擬機器也提高了實體機器上的資源利用率。透過雲端運算的服務，使用者不需要自己擁有大量的實體機器來運行應用服務，可以依照自己需求透過租借雲端的服務來降低成本。近年來，包括了 Amazon[2], Google[3]以及 Microsoft[4]都紛紛提供了自己的雲端服務以及負載平衡等機制。

在沒有負載平衡的機制下，虛擬機器可能會集中於某些實體機器上執行。而在同一台實體機器上執行的不同虛擬機器間，會因為共享硬體資源，例如對處理器的競爭以及對網路頻寬之間的競爭等，而造成效能的下降[5]。特別是對於資源量較低的機器所造成的競爭情形越是明顯。

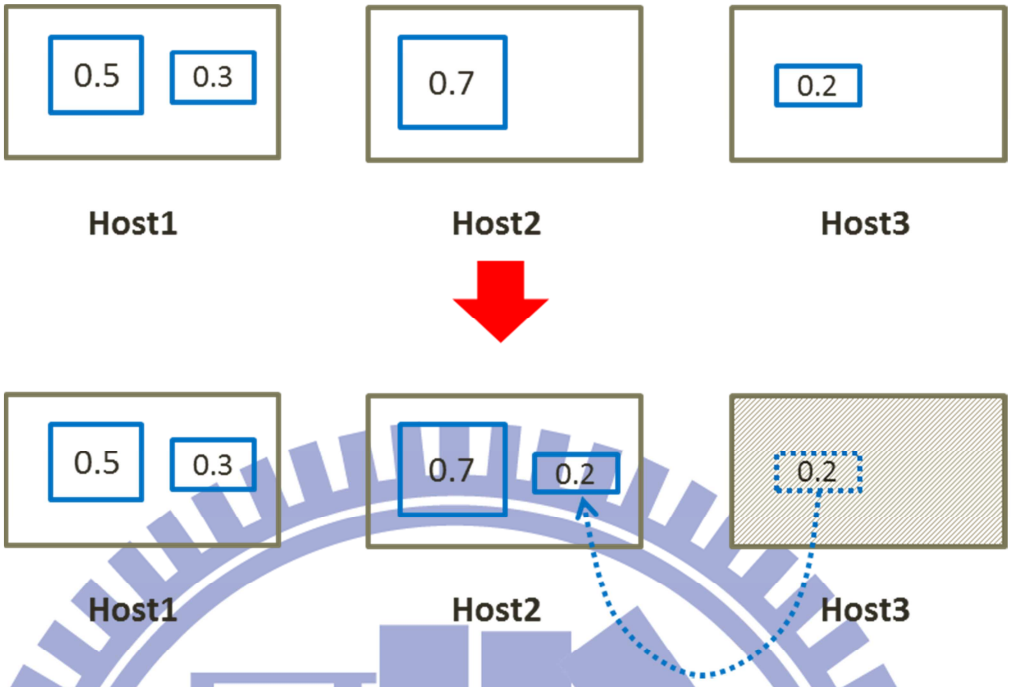
負載平衡(load-balancing)主要是在解決系統中的機器負擔不平衡的現象。考慮系統中現有的工作量，透過分配這些工作給不同的機器運行，來避免某些機器執行過多的工作。藉此加快工作的完成時間，提高整體的工作產能以及系統中的資源使用率。

在雲端的環境中，機器的資源量多寡(capacity)不盡相同，例如處理器速度、記憶體大小等。能力較強的機器可以同時執行較多的虛擬機器甚至負擔需求較高的虛擬機器的運算環境。然而在佈署虛擬機器時，去預測使用者所要求的虛擬機器能力，進而將虛擬機器依照實體機器的能力去佈署是有難度的。當系統中某些實體機器上的負擔過重，導致整體產能下降、硬體錯誤甚至當機的情形發生，要如何有效率的去偵測並且修復，所要付出的代價甚高，這些代價包括了從機器不能運作到被偵測到並且修復的這段時間，以及機器上所有虛擬機器程序皆必須重新執行的時間。然而，透過負載平衡，將系統中不同的虛擬機器，考慮其上的資

源使用率，透過 live migration 的機制來達到實體機器上的負擔皆在管理者設定的一個門檻值(threshold)之下，進而減少虛擬機器之間的資源競爭(resource contention)，提高應用程式執行的效能、以及避免上述的硬體錯誤情形發生。因此，我們提出了一個雲端運算上的負載平衡演算法，自動的監看系統中實體機器的資源利用度，透過虛擬機器的 Live Migration 來讓系統中實體機器的利用度皆低於我們設定的一個門檻值，進而使得實體機器中的不同虛擬機器間的資源競爭情形降低並且避免實體機器因為負擔過重造成硬體錯誤而當機等不容易察覺的情形發生。

在過去的雲端運算研究中，大部分的負載平衡演算法希望找尋虛擬機器可以遷移的最佳實體機器而採用一次只處理一個需要遷移的虛擬機器作法，然而，通常他們並沒有將虛擬機器遷移所需要的時間考慮進去，這將會造成系統達成平衡的時間變長，我們的實驗中顯示了當平衡的時間越長，虛擬機器上執行的應用程式將會因為資源競爭的關係而導致完成時間上升、系統的產能降低。因此，我們提出了一個將雲端系統中的負載平衡問題轉換成一個權重雙分圖，再透過求解其上之最小權重的一組配對，依照得到的配對，平行的將虛擬機器遷移到負擔較低的實體機器上，一次的配對可以處理系統中多個負擔過重的實體機器來讓系統達到平衡的時間縮短，進而提高產能的演算法。

雲端的資料中心通常因為需要提供大量的運算環境而必須同時運行許多的伺服器，這將會消耗大量的能源，除此之外，為了保持伺服器的正常運作，資料中心內使用的散熱系統的運行也需要消耗大量的能源，同時增加了碳的排放量。因此，近年來許多研究如[6][7][8][9][10]皆致力於如何分配雲端資料中心的工作量來讓閒置的伺服器可以暫時關閉，或者對工作量少的伺服器，將其上的虛擬機器遷移到其他伺服器，藉由關閉伺服器來達到節能的效果。圖一呈現一個節能的簡單例子，假設所有實體機器(Host)皆可以容納的工作量為 1，而 Host2 上可容納 Host3 上的工作量，因此，若我們將 Host3 上的虛擬機器遷移到 Host2 上，即可將 Host3 關閉。



圖一 節能示意圖

本篇文章接下來的部分包括:第二章會介紹負載平衡的相關研究；第三章提出我們的負載平衡演算法；第四章將會提出我們針對節能所改進的負載平衡演算法；最後在第五章中，我們經由模擬比較我們的方法以及其他文獻中方法的效能並搭建一個小型的雲端平台來證明我們的演算法可以部屬在實際的雲端環境中；最後第六章則是對於此篇文章的結論以及未來展望。

Chapter 2、相關研究

負載平衡被廣泛的使用在許多領域當中，如分散式系統(distributed system)、網頁伺服器(web server)、以及感測網路(sensor network)等。我們的研究是屬於雲端運算上的負載平衡，在此我們簡單扼要地描述幾個在雲端運算上相關的負載平衡研究。

在[11]的研究中，系統內的每一個虛擬機器上運行一個網頁應用程式，當用戶端的要求進來時先透過 Apache HTTP 負載平衡工具指定該要求給某一個虛擬機器上的網頁應用程式，並同時在虛擬機器上產生一個 session 來提供服務，而網頁應用程式可以產生的 session 即為此演算法所考慮的資源，虛擬機器的負擔計算方式為虛擬機器當前所產生的 session 個數除以系統中存在的所有 sessions 個數。當系統中所有虛擬機器上的 session 比例皆超過演算法設定的門檻值，代表系統中所有的虛擬機器皆判定為負擔過重，則需要產生新的虛擬機器來分擔系統中的要求，並同時規劃下一次蒐集系統資料的時間。而當系統中存在兩個以上的虛擬機器現有的 session 比例皆低於演算法設定的門檻值，此時代表目前系統中所有的工作量是較低的，則關掉一個虛擬機器。透過用戶端要求的重新指派來達到負載平衡的目的並動態的去評斷系統內的工作量。在這個方法中，用戶端要求經由 Apache HTTP 負載平衡工具來分配給不同的網頁應用程式提供服務，而演算法本身只負責偵測目前系統中所有的網頁應用程式是否足以負擔當前的要求，主要的附載平衡仍是透過 Apache HTTP 負載平衡工具來執行。

Zhao and Huang[12]在 EUCALYPTUS[13]雲端運算平台中，考慮到此平台上尚未提供負載平衡的機制而提出名為 COMPARE_AND_BALANCE 的雲端負載平衡演算法。COMPARE_AND_BALANCE 屬於分散式的負載平衡演算法，意即系統中的每一個實體機器都必須同時執行演算法。平衡的主要對象為實體機器上的虛擬機器個數，當實體機器上的虛擬機器個數越多，將被視為負擔越重。演算

法每次執行時，會先統計實體機器上總共有多少個虛擬機器並將該數值儲存在共享儲存器(shared storage)，以供其他實體機器取得，並同時從共享儲存器中取得其他實體機器上的虛擬機器個數，接著隨機選擇系統內的一個實體機器(各實體機器被挑選的機率為其上擁有的虛擬機器與系統中全部的虛擬機器比值)，接著計算並比較挑選的實體機器與當前的實體機器之成本值(cost)，有一定的機率會將當前實體機器中的一個虛擬機器遷移到所選的實體機器上來做負載平衡。然而，在現行的雲端運算環境中，雲端提供者依照價錢來提供不同運算能力的虛擬機器，因此，以實體機器上的虛擬機器數來判定是否為負擔過重並不是那麼的適當，因為有可能該實體機器上的虛擬機器皆恰巧是雲端提供者所提供運算能力最小的機器，而該實體機器各個維度的資源(例，處理器、記憶體等)皆尚可容納更多虛擬機器於其上執行。

在[14]的研究中，透過模擬蜜蜂(honeybee)採集食物的行為來進行負載平衡。系統中的每一個實體機器皆模擬成多個虛擬伺服器(virtual server)，每個虛擬伺服器上皆有一個虛擬的貯列，用來儲存系統中等待服務的要求(request)，而這些虛擬貯列即為蜜蜂採集食物的地點。虛擬伺服器執行完某個虛擬貯列上的要求時會有一個機率將獲益的多寡程度則貼在廣告板(advert board)上告知其他虛擬伺服器。如同蜜蜂覓食可能會自行探索食物來源或跟著同伴一同去食物來源較豐富的地點採集食物的行為，虛擬伺服器在每一回合中依照不同的機率分別隨機選取一個虛擬貯列執行或選擇廣告板上獲益較高的虛擬貯列來執行。

VectorDot 演算法[15]是一個考慮多維度資源(multi-dimension resources)基於雲端平台上的集中式(centralized)負載平衡演算法，透過實體機器中的虛擬機器遷移來達到平衡的效果。作者們建立了一個小型的雲端運算資料中心，並透過IBM **S**torage **V**olume **C**ontroller(SVC)將共享儲存器模擬成虛擬的磁盤(virtual disk)並將這些虛擬磁盤以及實體機器存取虛擬磁盤時中間經過的switch也加入演算法所考慮的資源維度中。演算法在一開始會去偵測系統中存在的觸發點(trigger node)，在這裡，對於任一個實體機器，只要存在任何一個維度的資源利用率大於演算法設定的門檻值時則被視為觸發點，反之則為非觸發點(non-trigger node)。觸發點代表著其上執行的虛擬機器對其造成之工作量超出演算法設立的門檻值，較有可能會產生硬體錯誤、虛擬機器間資源競爭較嚴重等情形而必須減少其工作

量。相反的，非觸發點則代表著實體機器可負擔更多的工作量，同時執行更多的虛擬機器。對於每一個觸發點，演算法必須決定：

1　哪一個觸發點需要優先被處理：對於每一個維度的資源，首先決定該維度的資源利用率超過了演算法設定的門檻值的程度(此處會利用演算法定義的公式求得一個值)，並將在所有資源維度中求得的值加總得到該觸發點之失衡分數(imbalance score)，具有最高的失衡分數之觸發點為優先被處理的對象。

2　虛擬機器需要被遷移到哪一個實體機器中：首先必須決定虛擬機器遷移的適合點(feasible node)，適合點代表著該實體機器必須可以容納虛擬機器之任一維度資源需求量以及虛擬機器存取虛擬磁碟中間經過的switch頻寬。接著選出虛擬機器的最佳適合點，對於一個虛擬機器來說，系統中可能同時存在許多適合點，作者用定義的公式來從這些適合點中挑選最佳的適合點來當作虛擬機器遷移的目標。作者在文章中提出了四個挑選最佳適合點的方法，包括了first-fit、best-fit、worst-fit以及relaxed-best-fit。First-fit、best-fit以及worst-fit皆會線性的搜尋系統中所有的實體機器，不同的是，first-fit將第一個挑選到的適合點作為虛擬機器遷移的目標，best-fit從所有找尋到的適合點中，挑選虛擬機器在原本機器上所造成的負擔以及虛擬機器對遷移目標造成的負擔之差值最大的當作虛擬機器遷移的目標，而worst-fit則是從所有找尋到的適合點中，挑選虛擬機器在原本機器上所造成的負擔以及虛擬機器對遷移目標造成的負擔之差值最小的當作虛擬機器遷移的目標。不同於其他三種方法，relaxed-best-fit考慮到線性搜尋所需付出的代價隨著系統中實體機器個數增加而上升，先隨機的從除了虛擬機器所在的實體機器外所有的實體機器中挑選一個為非觸發點的實體機器並測試他們是否為適合點，直到挑選出一定數量的適合點為止，這些數量遠小於非觸發點的個數，然後在這些適合中決定虛擬機器的遷移目標。這篇論文中同時證明了，使用relaxed-best-fit與best-fit來挑選適合點所得到的結果差不多，但是relaxed-best-fit所花的時間就比best-fit快上許多。

3　對於觸發點上的虛擬機器，哪一個是需要優先被遷移的對象：在觸發點上的虛擬機器，先決定那些虛擬機器被遷移後，可以使觸發點變成非觸發點，若再觸發點中不存在此種虛擬機器則必須考慮所有的虛擬機器。在這些虛擬機器中，考慮虛擬機器對所在的實體機器造成的影響值與欲遷移的實體機器造成的影響值

之差值，差值越大的虛擬機器優先做遷移。

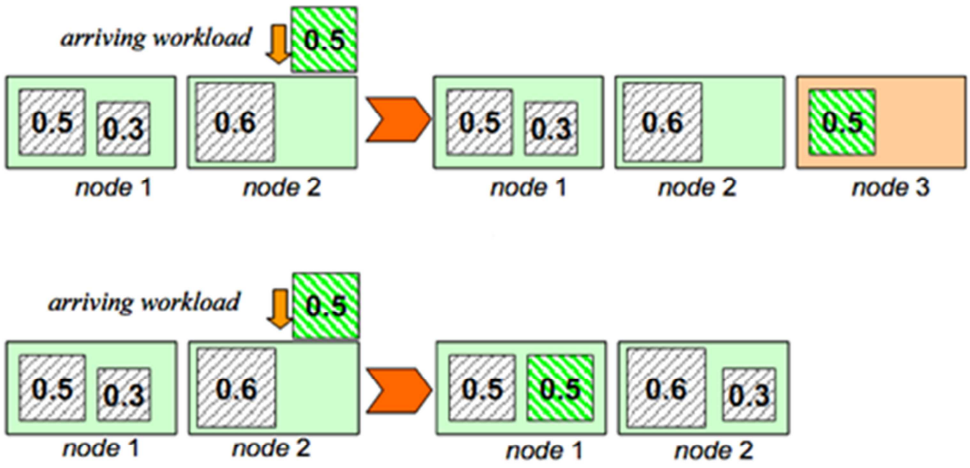
在決定完上述各點後，演算法對挑選的虛擬機器做 Live Migration，演算法執行的每一回合只遷移一個虛擬機器，直到系統中不存在任何觸發點為止。

在[16]的研究中，將 I/O 次數頻繁的應用程式讀取檔案的時間轉換成一個DAG(directed acyclic graph)上的最短路徑問題，利用馬可夫模型(Markov model)來表示檔案存取間的相依關係，提供了一個決定遷移是否對檔案讀取時間有利的決策機制。對於每一個 I/O 次數頻繁的應用程式來說，讀取非本地端的資料庫(database)可能需要很長的時間，這將會導致應用程式執行的完成時間上升，若執行應用程式的虛擬機器遷移到距離較靠近資料庫或者是資料庫所在的實體機器上的這一段時間加上讀取資料庫的時間比待在原本實體機器上讀取資料庫的時間來的短的話，那遷移將會得到好處。

在[7]中，作者將節能的問題模擬成傳統的bin-packing problem，系統中的每一個實體機器皆視為獨立的盒子，而工作量則視為一個各地物品，而節能的問題就變成了如何將物品裝入盒子中而使得現有的盒子數量最少。

每當系統中有新的工作量到達時，演算法就會執行，尋找最適合該工作量放置的盒子(實體機器)，而這並非僅僅在現有的盒子中選擇剩餘容量最多的盒子放置即可。考慮圖二中系統中工作量擺放以及新進的工作量，由於一開始系統中開啟的盒子只有兩個，分別是node1以及node2。node1及node2可以容納多餘的工作量之大小分別為0.2以及0.4，很明顯的，新來的工作量大小為0.5是沒有辦法放到node1或著是node2上面的，那這時候我們就必須去開啟一個新的盒子容納新進來的工作量。然而，若我們先將node1上大小為0.3的工作量搬移到node2上執行，則node1剩餘可以容納的工作大小為0.5與新進來的工作量相同，那我們就可以把新進的工作量指派給node1而不需要去開啟新的盒子。

在這篇文章中，每當系統中有新的工作量抵達，在指派給實體機器之前，演算法會先對當前系統中開啟的盒子全部尋找一遍，尋找是否有盒子是可以藉由先將其上工作量遷移到其他機器後而可以容納新的工作量。而當工作量本身的大小改變，或著是工作量因未完成而離開之時，演算法皆會執行，將系統中剩餘的工作量重新指派，藉此來保證系統中開啟的實體機器越少越好進而達到節能的效果。


arriving workload
0.5
0.5
0.3
0.6
node 1
node 2
0.5
0.3
0.6
0.5
node 1
node 2
node 3
arriving workload
0.5
0.5
0.3
0.6
node 1
node 2
0.5
0.5
0.6
0.3
node 1
node 2

圖二 節能演算法例子[7]

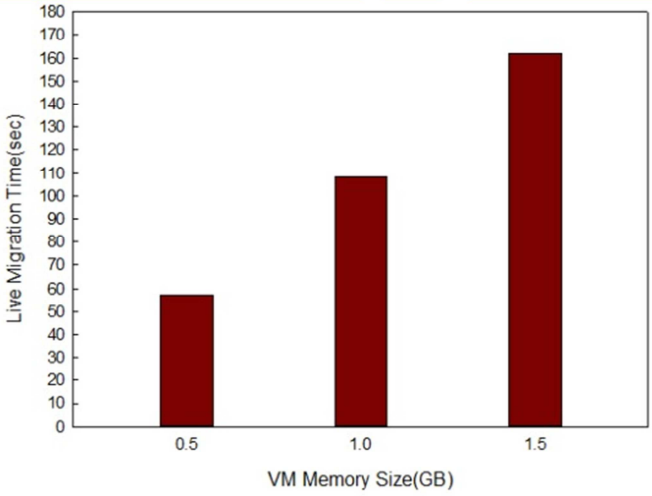
Chapter 3、基於平行遷移的負載平衡演算法

虛擬機器的遷移(migration)對需要在不同時間與地點使用不同實體機器的人提供了重要的行動能力。在雲端運算的環境中，我們使用虛擬機器來執行工作，透過虛擬機器的遷移，可以達到負載平衡，錯誤管理(fault management)的功能。傳統的 migration 必須先將虛擬機器上運行的作業系統關閉，並將虛擬機器的記憶體內容以及作業系統當下的運作狀態儲存並複製一份到遷移目標的實體機器上，然後在該實體機器上重新運行作業系統。然而，在做複製運作狀態以及記憶體內容的這段時間內，整個虛擬機器是沒有辦法提供任何的服務以及運算能力，隨著虛擬機器的記憶體容量越大，或是系統內的網路頻寬越壅塞，造成 migration 的時間變長，不能提供服務的時間也就越長。Live Migration[17]、[18]不同於傳統的 migration，live migration 在欲遷移的實體機器上先建立一個運算能力相同的虛擬機器，在遷移的過程中，原本的服務以及程序的運作皆在原本的虛擬機器上執行而不關閉原本運行中的虛擬機器，讓系統持續的具有服務的能力。直到狀態及記憶體內容複製完畢後，服務以及程序的運作才交由新的虛擬機器執行，大幅的簡短了虛擬機器因為遷移而無法提供服務的時間。

在前一個章節當中，我們看到許多負載平衡演算法並沒有去考慮虛擬機器遷移的時間，導致系統的平衡時間變長。當系統的平衡時間變長，這將使得負載過重實體機器中的虛擬機器間資源競爭情形無法得到舒緩而讓產能降低。我們若將虛擬機器的遷移時間也列入考慮的因素，在[16]的研究中指出，網路頻寬以及虛擬機器的記憶體容量大小為主要影響虛擬機器遷移時間的兩個因素並提出公式(1)來計算虛擬機器的遷移時間。

$$m = {}^{v}/_{h} + C \qquad (1)[16]$$

公式(1)中的v以及h分別代表著虛擬機器的記憶體容量大小以及網路頻寬，由式子中我們可以觀察出遷移時間與網路頻寬大小成反比且與記憶體容量成正比，而 C 為關閉舊有的虛擬機器以及開啟新的虛擬機器所需耗費的一個常數時間。圖三為我們在頻寬為 10Mbytes/sec 的情況之下，在我們搭建的 XCP 雲端平台上對不同記憶體容量大小的虛擬機器做遷移所得的結果。與上述研究中相符合，當虛擬機器的記憶體容量上升，live migration 的時間也跟著變長。當虛擬機器的記憶體容量僅僅為 512MB 時，遷移亦需將近一分鐘的時間，因此，循序的負載平衡演算法如 VectorDot 演算法最快也需要在一分鐘過後才可以再度執行。由於 VectorDot 演算法一回合只會遷移一個虛擬機器，也就是說，一個回合最多只能減少一個觸發點，甚至在某些情況下，原本的觸發點在虛擬機器遷移後仍為觸發點，而虛擬機器的遷移造成另一個本為非觸發點的實體機器變為觸發點，因此系統達到平衡的時間將會很長，這將導致了負擔過重的實體機器上互相競爭資源的虛擬機器無法立即被處理而導致產能降低。



圖三 VM live Migration 時間

3.1 基本想法

當資料中心同時存在許多負擔較重的實體機器，它們就是觸發虛擬機器遷移的實體機器，在這裡我們稱之為觸發點(trigger node)，否則，則為非觸發點(non-trigger node)。考慮虛擬機器遷移的時間，演算法執行一次只處理一個觸發點將會導致其他觸發點內的虛擬機器競爭資源的情形在短時間內無法得到舒緩

而讓系統的產能降低。若一次處理多個觸發點，系統平衡的程度則有可能較差，因此，我們將專注於如何讓系統較快的達到平衡而平衡的程度不至於相差太多。

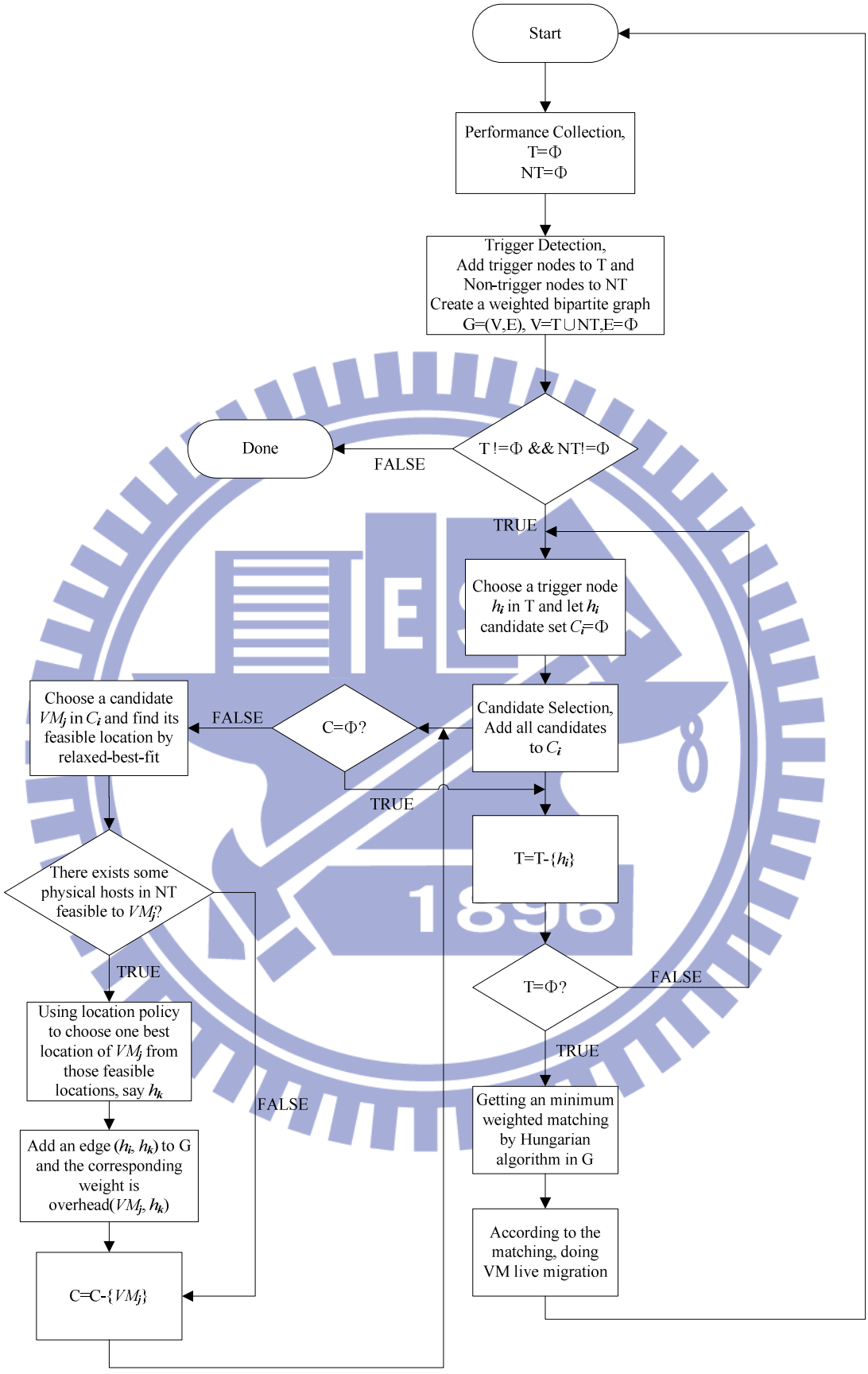
為了解決上述提到的問題，我們提出一個考慮多維度資源、機器間的能力以及一次處理系統中多個觸發點的負載平衡演算法，我們稱為 **B**ipartite **M**atching (BM)的負載平衡演算法，希望可以藉由一次處理多個觸發點，來讓系統達到平衡的時間縮短，盡早降低虛擬機器間的競爭資源情況而提高產能，而仍維持穩定的系統平衡程度。

我們的方法如同 VectorDot[15]中，考慮實體機器的各維度資源利用率以及演算法定義的門檻值來將系統中所有的實體機器分為觸發點以及非觸發點，在所有的觸發點中，用我們的策略挑選出欲遷移的虛擬機器們和其可遷移的非觸發點，然後將上述之資訊轉換成一個權重雙分圖(weighted bipartite graph)，其上之點分別為觸發點以及非觸發點，而圖中的邊以及邊上之權重則分別為欲遷移之虛擬機器所挑選的非觸發點以及該虛擬機器之遷移對非觸發點所產生的負擔。再透過求解最小權重配對問題(minimum weighted matching problem)來得到一組配對，配對中的每一組匹配(pair)代表著一個虛擬機器的遷移，透過這個配對平行的將所有觸發點中的虛擬機器遷移至非觸發點上，持續上述步驟直到系統中不存在任何觸發點為止，來讓系統達到平衡。我們藉由根據求解最小權重配對問題中所得的配對個數來一次處理多個觸發點，使得系統達到平衡的時間縮短，而虛擬機器之間的資源競爭情形也可較快下降。

3.2 實際作法

在[19]的研究中，對負載平衡演算法所需決定的機制以及策略作了以下的分類，分別為 Transfer Mechanism、Metric Mechanism 、Communication Mechanism Participation Policy、Candidate Selection 以及 Location Selection。Transfer Mechanism 主要在決定如何移動工作量來達成負載平衡。Metric Mechanism 用來決定機器的負擔(load)，例如處理器的使用率、實體機器上的虛擬機器個數等。Communication Mechanism 決定了哪些資訊需要被溝通和不同的實體機器如何溝

通這些資訊。Participation Policy 用來決定要執行負載平衡的對象，例如雲端運算中的實體機器們。Candidate Selection 用來選擇負擔過重(over loaded)的機器中，需要移動那些工作量來讓機器的負擔降低。Location Selection 則是用來從負載較低的機器中選定一個理想的實體機器來做為工作量遷移的目標。


Start
Performance Collection,
T=Φ
NT=Φ
Trigger Detection,
Add trigger nodes to T and
Non-trigger nodes to NT
Create a weighted bipartite graph
G=(V,E), V=T∪NT,E=Φ
T !=Φ && NT!=Φ
Done
FALSE
TRUE
Choose a trigger node
h_i in T and let h_i
candidate set C_i=Φ
Candidate Selection,
Add all candidates
to C_i
C=Φ?
FALSE
Choose a candidate
VM_j in C_i and find its
feasible location by
relaxed-best-fit
TRUE
T=T-{h_i}
There exists some
physical hosts in NT
feasible to VM_j?
TRUE
FALSE
Using location policy
to choose one best
location of VM_j from
those feasible
locations, say h_k
Add an edge (h_i, h_k) to G
and the corresponding
weight is
overhead(VM_j, h_k)
C=C-{VM_j}
T=Φ?
FALSE
TRUE
Getting an minimum
weighted matching
by Hungarian
algorithm in G
According to the
matching, doing
VM live migration

圖四 演算法流程圖

圖四為我們演算法 BM 的流程圖，我們的方法基本上可以分為下列步驟：1. 決定機制，包括了 Transfer Mechanism、Metric Mechanism 以及 Communication Mechanism。2. 蒐集機器的資源利用率(resource utilization discovery)。3.偵測觸發點(trigger detection)。4.決定策略，包括了 Participation Policy、Candidate Selection 以及 Location Selection。5.建立權重雙分圖。6. 求解最小權重配對問題並執行虛擬機器之遷移。BM 演算法會持續上述步驟直到系統中不存在任何的觸發點或著是系統中不存在任何的非觸發點，這表示系統中所存在的工作量超過系統的門檻值而無法達成平衡。我們分項細述我們的步驟如下：

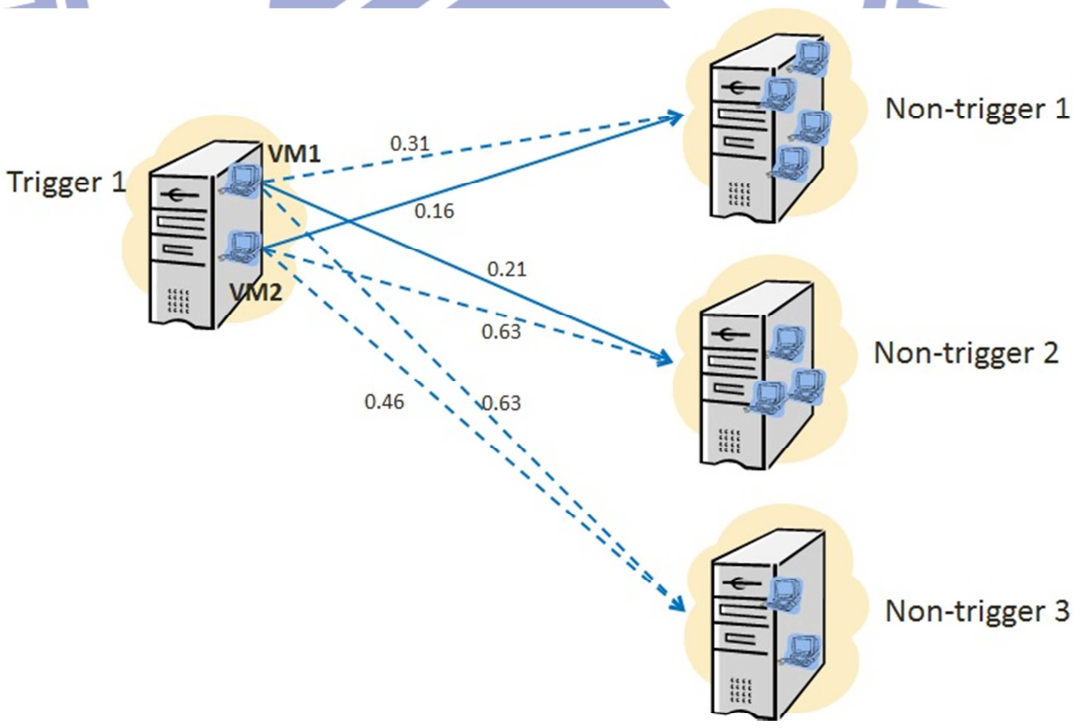
1. 決定機制：如同[19]中提及的，我們首先決定 BM 負載平衡演算法的機制，包括了 transfer mechanism、metric mechanism 以及 Communication Mechanism。在我們的方法中我們的 transfer mechanism 是經由虛擬機器的 live migration 來移動工作量。我們的 metric mechanism 是考慮多維度的資源，包括了處理器、記憶體以及網路頻寬。例如在我們建置的 XCP 雲端平台上，我們的 communication mechanism 是以集中式週期的去發送 HTTP request 的方式取得 XCP 上 rrd 資料庫中紀錄的各實體機器上的資源利用率。
2. 蒐集機器的資源利用率：在這個步驟中，我們透過 communication mechanism 來蒐集所有實體機器的資源利用率，包括處理器利用率、記憶體使用率以及網路頻寬佔用率，並計算實體機器上所剩下的資源量。給定 $H=\{h_1, h_2, \ldots, h_m\}$ 為雲端系統中的實體機器集合，對一個實體機器 $h_i \in H$，它有 n 個維度上的資源，每個維度的資源總量為 $CAP_\alpha=\{cap_{\alpha1}, cap_{\alpha2}, \ldots, cap_{\alpha n}\}$以及資源被使用率為 $R_\alpha=\{r_{\alpha i} \mid r_{\alpha i}\in[0,1], i \in \{1, 2, \ldots, n\}\}$，我們也定義一個 $T=\{t_1, t_2, \ldots, t_n\}$為演算法在各個資源維度上分別設定的門檻值，我們用 $FR_\alpha=\{fr_{\alpha i} \mid fr_{\alpha i}=(1-r_{\alpha i})*cap_{\alpha i}, i \in \{1, 2, \ldots, n\}\}$來代表實體機器 h_i 分別在 n 個資源維度上所剩餘的資源量。
3. 偵測觸發點：對於任一個實體機器 h_α，若存在任一個 $r_{\alpha i} > t_i$，則 h_i 為一個觸發點，否則 h_i 為一非觸發點。我們將這個步驟中決定的觸發點以及非觸發點們分別以集合 T 以及 NT 表示。
4. 決定策略：接著我們決定以下策略，包括了 Participation Policy、Candidate Selection 以及 Location Selection。我們利用實體機器、共享儲存器(shared

storage)以及 switch 來組成一個雲端系統，而這個系統中的所有實體機器即是我們 participation policy 中決定的演算法平衡對象。我們的 candidate policy 如同 VectorDot(VD)[15]，優先考慮觸發點上移除之後可以使觸發點變為非觸發點之虛擬機器，希望遷移最少量的機器後即可減少系統中的觸發點數目。若觸發點上任一虛擬機器的移除皆無法將其改變為非觸發點，則觸發點上的虛擬機器皆是我們可能遷移的 candidates。對於系統中的一個觸發點 h_α，我們定義一個 C_α 集合來表示上述步驟決定的這些 candidate 虛擬機器。對於所有的虛擬機器 $VM_\beta \in C_\alpha$，我們透過我們的 location policy 來決定它們遷移的目標實體機器。首先，我們從非觸發點中挑選虛擬機器遷移的適合點(feasible location)。對於任一個非觸發點，我們希望每一次的虛擬機器遷移不會產生新的觸發點來加快系統達成平衡的時間，所以適合點的挑選條件為虛擬機器之遷移不會使非觸發點變為觸發點，則該非觸發點即為虛擬機器的一個適合點。適合點的找尋如果使用線性搜尋(linear search)的方式可能耗費較多的時間，所以我們使用 VD 中的 relaxed-best-fit 方法隨機的從所有的非觸發點中挑選一個為非觸發點的實體機器並測試他們是否為適合點，直到挑選出一定數量的適合點為止，這些數量遠小於非觸發點的個數，接著利用公式(2)求算虛擬機器 VM_β 對挑選到的適合點 h_γ 所造成的額外負擔值。

$$OH(VM_\beta) = \sum_i \frac{vmrc_{\beta i}}{fr_{\gamma i}}, h_\gamma \in NT, i \in \{1,2,\dots,n\} \qquad (2)$$

$vmrc_{\beta i}$ 代表著虛擬機器 VM_β 在某個實體機器 h_γ 上第 i 維度上的資源需求量。我們將 n 個維度上各個資源需求量除以非觸發點的資源剩餘量加總起來，來表示當 VM_β 移到 h_γ 上時所造成的額外負擔。對於剩餘資源量越少的適合點，虛擬機器對其所造成的負擔越高，公式(2)求得的值亦越高。最後取出擁有最小額外負擔值的適合點為最佳適合點以作為虛擬機器 VM_β 遷移的目標。C_α 集合中虛擬機器們代表著任一個虛擬機器的遷移皆可以使得觸發點 h_α 變成非觸發點，當這些候選的虛擬機器們挑選到同一個最佳適合點 h_γ 為遷移的目標時，我們將會選擇造成 h_γ 最少額外負擔的虛擬機器 v 來做遷移。v 對 h_γ 造成的額外負擔最小這意味著 v 在 h_α 上產生的負擔也最小，因此選擇 v 來做遷移既可

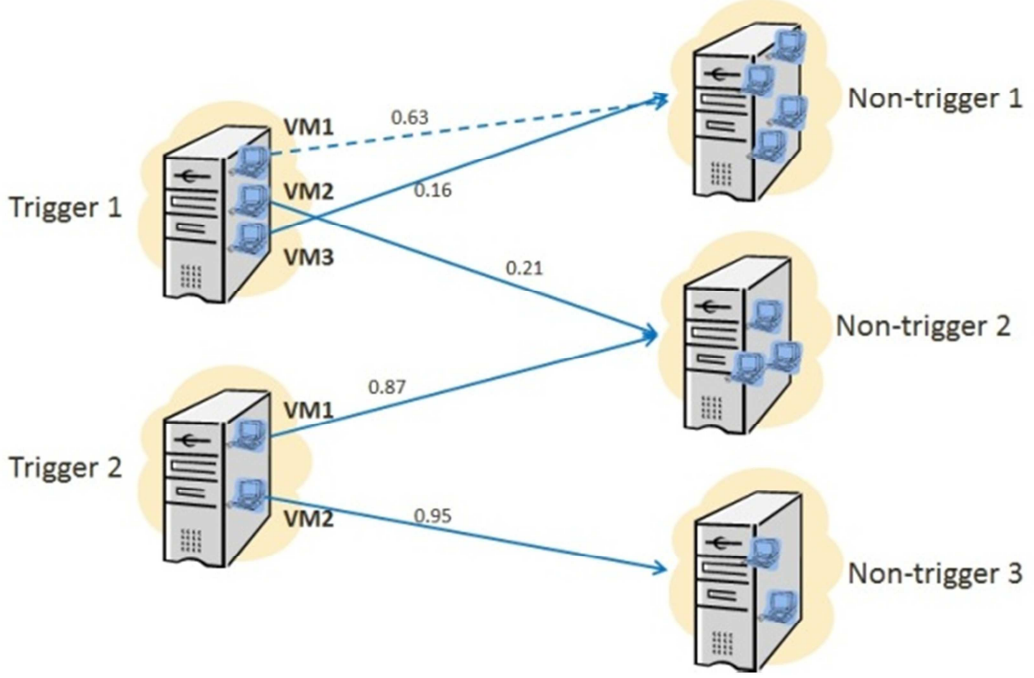
以讓 h_α 變為非觸發點同時又可以讓虛擬機器 v 遷移過後 h_α 的資源使用率較靠近系統的門檻值而得到較好的平衡程度。然而，當觸發點中沒有任何虛擬機器的遷移可以使得觸發點 h_α 變成非觸發點時，C_α 集合就變成觸發點上所有虛擬機器的集合，同樣地，如果這些虛擬機器們皆挑選到同一個最佳適合點 h_γ，我們一樣會選擇造成 h_γ 最少額外負擔的虛擬機器 v 來做遷移，但這可能會產生一個問題，因為遷移任何一個 C_α 中的虛擬機器皆無法將 h_α 變成非觸發點，必須要從 C_α 中遷移多個虛擬機器後才可以讓 h_α 變成非觸發點。然而，我們遷移的虛擬機器 v 在所有的 candidates 中對 h_α 造成的負擔是最小的，而遷移產生負擔較小的虛擬機器將會使 h_α 需要遷移更多的虛擬機器才能變成非觸發點。所以我們藉由 relaxed-best-fit，採用隨機挑選適合點的方式，來減少不同虛擬機器挑選到相同的最佳適合點的這種機會發生。圖五示範了從兩個非觸發節點中如何決定虛擬機器之合適節點。在觸發點 Trigger1 中，VM1 以及 VM2 對所有非負擔節點所造成之負擔值分別為 0.31，0.21，0.63 以及 0.16，0.63，0.46。因此，VM1 與 VM2 所挑選之遷移地點分別為編號為二的非觸發節點以及編號為一的非觸發節點。



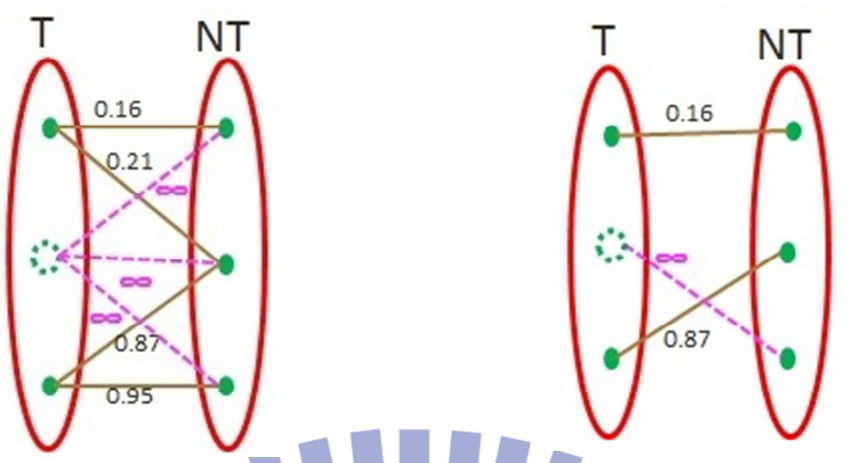
圖五 挑選適合節點範例

5. 建立權重雙分圖：在步驟 3 中，我們將系統中的實體機器分為觸發點以及非觸發點，令 T 以及 NT 分別為系統中所有觸發點以及非觸發點所成的集合。我們建立一個權重雙分圖 $G=(V,E,W)$，其中，$V=T\cup NT$，$E\subseteq T\times NT$，$E=\{(h_\alpha, h_\beta)|\ h_\alpha$ 中有任一虛擬機器選擇 h_β 為遷移目標點$\}$，W 為邊上權重之集合，權重即為上述步驟中挑選適合點時由公式(2)求算的額外負擔值。然而當同一個實體機器上不同的虛擬機器挑選到同一個非觸發點作為遷移的對象時，這會造成兩點間存在超過一個邊，因此我們只選擇對非觸發點造成最小額外負擔值的邊，來保證任兩點之間不會同時存在兩條以上的連線。
6. 求解最小權重配對問題：利用 Hungarian 演算法[20]來求解。Hungarian 演算法主要用來求解權重雙分圖中的最小權重配對問題，藉由分別給予雙分圖中所有節點一個權重值，藉由調整點權重代替調整邊權重來取得最小權重配對。我們在得到配對之後，依照得到的配對將觸發點上的虛擬機器遷移到配對中對應的非觸發點上來進行負載平衡。由於 Hungarian 演算法得到的解為一個完美配對(perfect matching),我們必須保證我們建立出來的權重雙分圖中存在著完美配對。當我們經由上述步驟取得權重雙分圖 G 後，首先我們比較觸發點集合 T 與非觸發點集合 NT 中的元素個數，對於擁有較少元素之集合，我們增加集合中的虛擬點(pseudo-node)直到兩個集合中擁有相同的元素個數。接著，將不同集合中任兩個沒有連線的點以一虛擬邊(pseudo-edge)相連來將 G 變為一個完整權重圖(complete weighted bipartite graph)，而虛擬邊的權重則設定為一個很大的數值,藉著上述的方法來保證建立的雙分圖 G 中存在著完美配對。經過 Hungarian 演算法求得的最小權重配對中，除了虛擬邊之外，每一個邊都代表著一個虛擬機器和其遷移的目標實體機器，我們在這裡稱為有效邊(valid edge)。當得到配對中的有效邊越多，我們的演算法 BM 一個回合可以處理的觸發點也越多，而系統達成平衡的時間也就越快，這也是為什麼我們使用 relaxed-best-fit 來挑選虛擬機器最佳適合點的原因，透過隨機的挑選，我們可以有效的減少不同虛擬機器挑選到同一個最佳適合點的機會來讓權重雙分圖中的邊數目增加，進而讓最小權重配對中的有效邊增加，增加 BM 演算法每次可以處理的觸發點個數來加快平衡的速度。

圖六、圖七(1)和圖七(2)顯示了上述演算法的一個簡單例子。圖六顯示了一個小型雲端系統，系統中分別存在著兩個觸發點以及三個非觸發點。圖中的連線代表著觸發點中虛擬機器挑選到的最佳適合點，邊上的權重則為虛擬機器所造成的額外負擔值。觸發點 Trigger1 中的虛擬機器 VM1 以及 VM3 皆選擇非觸發點 Non-trigger1 為遷移的地點，這使得 Trigger1 與 Non-Trigger1 中存在兩條連線，根據步驟 5 的條件，由於 VM1 以及 VM3 對 Non-trigger1 造成的額外負擔值分別為 0.63 以及 0.16，所以我們只會挑選 VM3 對 Non-trigger1 形成的邊，而連線之權重為 VM3 對 Non-trigger1 造成的額外負擔值 0.16。接著我們將圖六轉換成一個權重雙分圖。圖七(1)為圖七(2)轉換而成的權重雙分圖，其中虛線的點代表虛擬點而虛線的邊代表虛擬邊，利用 Hungarian 演算法我們可以得到圖七(2)中的一組最小權重配對。由配對中我們可得知 BM 演算法將會在這一回合中把 Trigger1 中的 VM3 以及 Trigger2 的 VM1 分別遷移到 Non-trigger1 及 Non-trigger2 上。



圖六 BM 演算法範例



(1) 權重雙分圖　　(2) 最小權重配對

圖七 權重雙分圖和其最小權重配對

Chapter 4、節能負載平衡演算法

在前面的章節中，我們提出了一個基於平行遷移、考慮多個資源維度的負載平衡演算法，我們稱為 Bipartite Matching(BM) Load-Balancing Algorithm。考慮到現今雲端運算資料中心的運行時造成的大量能源消耗，我們希望可以把節能這個議題也加入考慮。在我們閱讀到的文章中，大部分作者的做法是把節能的問題模擬成傳統的 bin-packing 問題，盡可能地減少系統中同時運行的伺服器個數。然而，在考慮到多維度資源的情況下，我們很難去定義一個工作量的大小，因為處理器、記憶體以及網路頻寬的單位皆不相同；而對一個負載平衡的演算法來說，關掉系統中的某些機器勢必會讓系統中其他運行的機器負擔上升，導致虛擬機器競爭資源的情形上升，讓產能降低。因此，要如何決定要讓系統中同時運行多少機器而不讓產能降低太多而同時又能夠達到節能的機制是一個困難的問題。在這個章節中我們提出一個稱為 Energy Aware Bipartite Matching (EABM)的節能負載平衡演算法，去評估使用多少機器可以負擔系統現有的工作量而讓其他的實體機器關閉以達到節能的效果，並綜合前一個章節的演算法來讓我們的演算法可以同時達到負載平衡以及節能的效果。

4.1 能量計算公式

公式(2)為在【12】的研究中，提出了實體機器消耗的能量公式。

$$P(u) = k * P_{max} + (1 - k) * P_{max} * u \qquad (2)[8]$$

式子當中的 P_{max} 是當處理器使用率在 100%時實體機器會消耗的最大能量；k 是當實體機器為空閒時時所會消耗的能源比例；而 u 是實體機器上處理器的使用率。此外，在[12]的研究中也提到，實體機器處於空閒的情況時，所消耗的能源是處理器在使用率為 100%時所消耗能源的 70%，稱為 static power。而剩餘的 30%的

能量則與當下的處理器使用率有關，稱為 dynamic power。考慮到 static power 所佔的比例較重，我們可以將空閒的實體機器關閉來達到節能的效果。

4.2 節能演算法

給定 $H=\{h_1,h_2,\ldots,h_m\}$為雲端系統中的所有實體機器集合、$V=\{v_1,v_2,\ldots v_n\}$為雲端系統中的所有虛擬機器集合以及 $T=\{t_1,t_2,\ldots t_n\}$為各個維度資源所對應的門檻值。首先我們將系統中所有虛擬機器在各個維度所需要的資源量大小加總，我們稱為虛擬機器資源需求總量。

$$\sum vmrc_{ki} \ \forall vm_k \epsilon V, i \in \{1,2,\ldots,n\}$$

再來我們計算系統中各個實體機器各資源維度可使用資源量，在這裡我們稱為可用資源量。

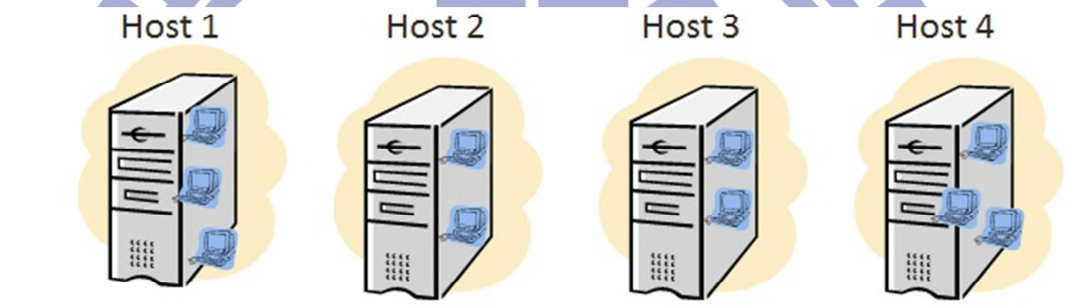
$$t_i * cap_{\alpha i}, h_\alpha \epsilon H, i \in \{1,2,\ldots,n\}$$

由於我們不希望機器的資源使用率超出定義的門檻值，所以在這裡實體機器的可用資源量即是其擁有的資源總量與門檻值的乘積。最後我們將所有實體機器在各個維度上的可用資源量加總，得到系統可用資源總量。

$$\sum t_i * cap_{\alpha i} \ \forall h_\alpha \epsilon H, i \in \{1,2,\ldots,n\}$$

在 4.1 節中我們有提到，我們傾向於關閉系統中處理器能力較弱的機器來減少公式中所占比例較重的 static power。所以我們將實體機器依照在處理器維度上的可用資源量由小至大排序，接著我們將系統的可用資源總量依序減去實體機器在各個維度上的資源可用量，並將那些實體機器記錄起來，直到系統的可用資源總量即將小於虛擬機器資源需求總量為止。這代表著系統中的虛擬機器至少需要這麼多的資源量才可以在不讓實體機器超過門檻值的情形下運行，而依序取出的實體機器則為我們演算法中判定要關閉的機器。我們會將這些實體機器轉變為觸發點，隨著負載平衡的進行對其上所有虛擬機器做遷移，直到實體機器沒有任何虛擬機器時再行關閉。圖 4.1 為節能演算法的一個簡單例子。假設系統中的虛擬機器在處理器及記憶體所需的資源總量分別為 5.7GHz 以及 7GB，而處理器以及記憶體的系統可用資源總量分別為 1.6+1.8+2.6+3.2=9.2GHz 以及

2+4+6+2=14GB。在圖 4.1 中我們已經對實體機器依照處理器維度的可用資源量的大小進行排序。現在考慮關掉 Host1，因為 9.2-1.6=7.6GHz > 5.7GHz 而且 14-2=12GB > 7GB，系統可用資源總量在各個維度上皆大於虛擬機器資源需求總量，所以 Host1 可以關閉，按照一樣的算法，Host2 也可以關閉，關閉後處理器以及記憶體的系統可用資源總量分別為 5.8GHz 以及 3GB。當我們嘗試關掉 Host3 時，因為 5.8GHz-2.6GHz <5.7GHz，所以 Host3 不能關閉。接著我們並不需要再嘗試關掉 Host4，因為 Host4 在處理器維度上的資源可用量大於 Host3 的，由前一個步驟我們知道 Host3 沒辦法關，Host4 當然也不行。這也是我們對各個實體機器依照處理器可用資源量做排序的原因，可以避免搜尋所有的實體機器，加快搜尋的時間。



CPU	Memory
1.60GHz	2GB

CPU	Memory
1.80GHz	4GB

CPU	Memory
2.60GHz	6GB

CPU	Memory
3.20GHz	2GB

圖八 EALB 範例圖

Chapter 5、實驗結果

在這個章節中，我們將會討論在前面章節中提到的 BM 以及 EABM 演算法的效能以及在 XCP testbed 上的實驗。首先，在 5.1 節中我們使用 CloudSim[21]這個專門用來模擬雲端運算環境的模擬工具來實驗 BM 演算法在不同實體機器個數下的效能，並與 VectorDo(VD)[15]的效能做比較。其次，在 5.2 節中利用 **X**en **C**loud **P**latform(XCP)[22]來搭建一個小型的雲端平台來證明我們的方法確實可用於實際的雲端平台上來達到負載平衡的效果。以下我們分別介紹我們的實驗環境和模擬結果，以及如何在 Xen 平台上測試我們的負載平衡機制。

5.1 模擬結果

CloudSim toolkit[21]是一個模擬雲端環境及雲端內部機器行為的開放原始碼工具。藉由簡單的 API 讓使用者可以創建實體機器以及虛擬機器、取得實體機器以及虛擬機器上的資源使用率以及模擬虛擬機器 Live Migration 。提供了正在進行雲端方面的研究者一個大規模且簡便的模擬平台，用來測試自己所設計的雲端運算行為和策略，而不需要直接向雲端提供者租借服務，可以降低研究的成本。在 CloudSim 中，透過 Broker 這個物件當作雲端提供商與使用者的中介者提供雲端租借服務。使用者只須告知 Broker 欲租借的虛擬機器之規格，例如 CPU 能力、記憶體大小等，並將想要執行的應用程式一併提供給 Broker。其後虛擬機器的 allocation 以及應用程式的指派(assign)就交給雲端提供者所制定的策略來執行。Cloudlet 用來模擬應用程序，指定該 Cloudlet 完成所需要執行的總共百萬指令數(MIPS)，即可模擬該應用程式所需執行的時間。我們在這邊的模擬即為一個雲端提供者的角色，對於系統中現有的實體機器與虛擬機器，提供一個負載平衡的機制。

我們分別觀察不同數量的實體機器對 BM 演算法效能的影響，包括了 50、250、450、650、850 以及 1050 台實體機器，虛擬機器數目在 161 台到 3232 台之間，而系統達成平衡後的各個維度資源利用率約為 60%。虛擬機器創建之後隨機指派到實體機器上執行，而一個虛擬機器上執行一個應用程式。此外，我們使

用 2.8GHz(12,000MIPS)四核心的處理器、4GB 的記憶體以及系統網路頻寬在 128MBytes/sec 的實體機器環境當作我們的基準值。將所有實體機器的能力設定在這個基準值的正負 20%來模擬雲端系統中實體機器的異質性(heterogenity)。我們同時假設虛擬機器上執行的應用程式為一個大型的批次檔(batch file)。其他細部的模擬實驗參數列在表格一中。

在實驗數據中，我們為了要測試演算法達到平衡之後的效能，所以我們設定的 cloudlet 長度要足以讓虛擬機器執行完成的時間大於 VD 演算法達成平衡的時間。否則 VD 在尚為達到負載平衡前系統中的應用程式已執行完畢。

接下來我們分別測試演算法達到平衡後系統中的各個資源維度的資源利用率標準差、各個資源維度的平均使用率、使系統達到平衡的時間、達到平衡所需遷移的虛擬機器個數、一開始模擬時負擔過的那些重實體機器上的 cloudlets 完成時間以及不同的系統門檻值下 EALB 消耗的總能量。

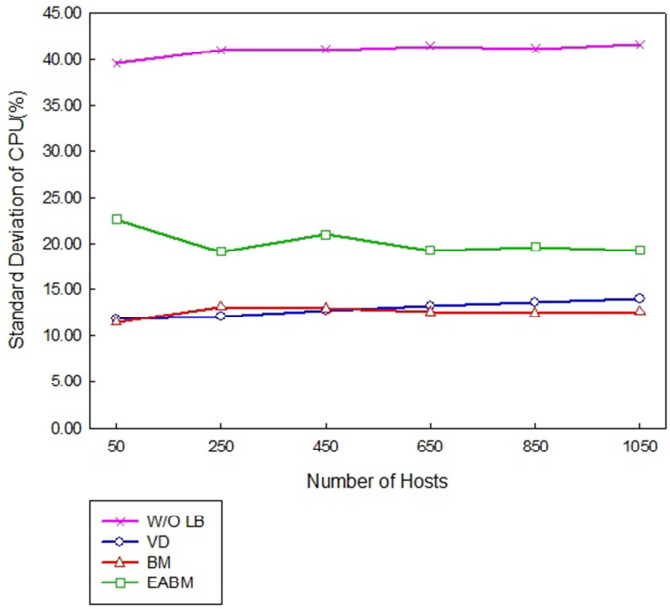
	Parameter	Value
Physical Host	CPU	4±1 core, 2.80GHz
	Memory	4±1GB
	Network Bandwidth	128±25MBytes/sec
VM	CPU	12% ~ 25% of single core 2.80GHz
	Memory	12% ~ 25% of 4GB (492MB ~ 1024 MB)
	Network Bandwidth	12% ~ 25% of 10MBytes (1.2MBytes ~ 2.5 MBytes)
	Cloudlet MIPS Length	660000000

表格一 實驗參數

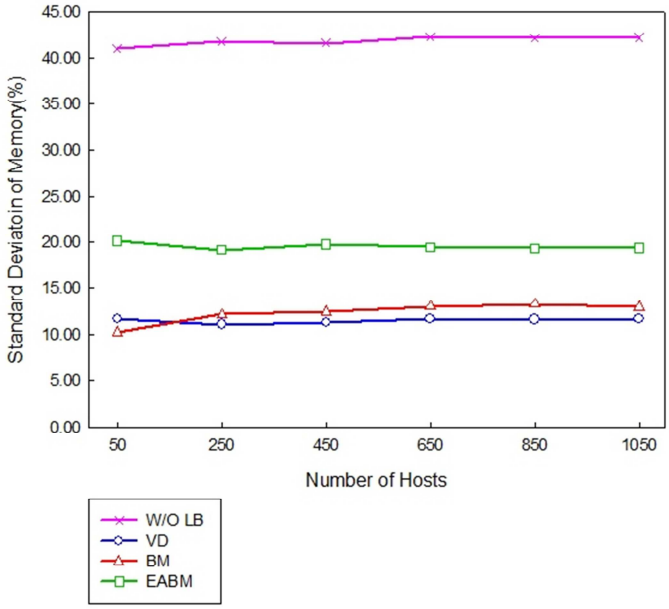
圖九~圖十九 顯示了六個實驗結果，分別為系統達到平衡後各個資源維度的標準差、各個資源維度的平均使用率、系統達成平衡的時間、達成平衡所遷移的虛擬機器數量、每次演算法執行平均的虛擬機器遷移個數、系統中所有 cloudlet 完成時間之總和以及不同的系統門檻值下 EALB 消耗的總能量。

圖十二、圖十三以及圖十四中顯示不同數目的實體機器對不同的維度的資源，包括了處理器、記憶體以及網路頻寬使用率的標準差。資源維度的標準差計算方式為演算法達到收斂後系統內實體機器資源使用率之標準差，在這邊平衡的條件為系統中的所有實體機器皆成為非觸發點。我們的演算法 BM、EABM 以及

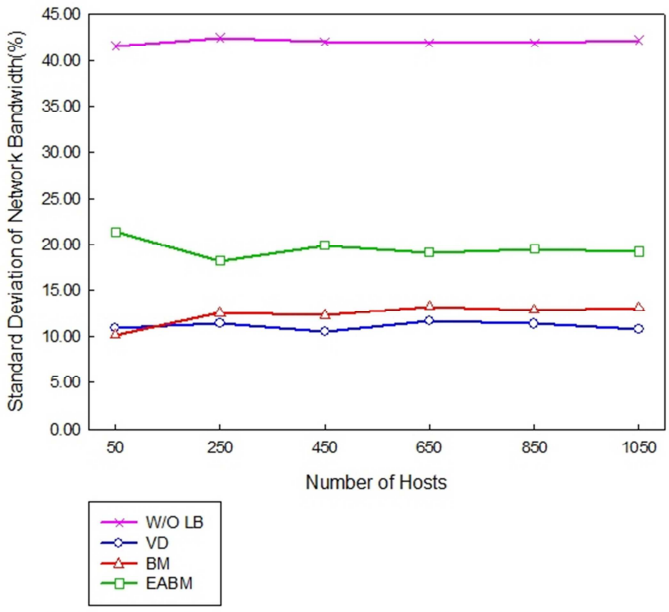
VectorDot(VD)所得到的資源標準差皆會比沒有做負載平衡(W/O LB)的結果來的小。而沒有加入節能時我們的演算法在達到平衡後的標準差與 VD 的差不多，差值皆在 5%之內，顯示了兩個演算法對系統的平衡程度是差不多的，並證明了我們的演算法能達到收斂。至於為什麼加入節能後的 EABM 演算法資源使用率標準差會升高，那是因為為了節能我們關掉了幾台實體機器，強迫將這些實體機器上的虛擬機器遷移到其他的實體機器上執行而造成某些實體機器上的工作量上升的緣故。



圖九 處理器維度標準差

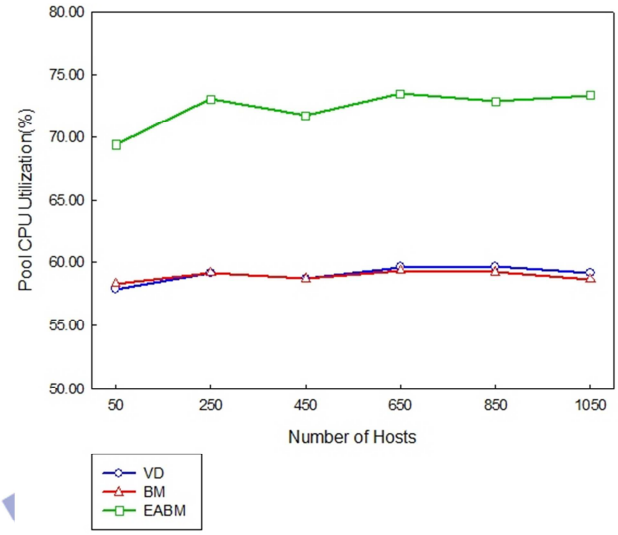


圖十 記憶體維度標準差

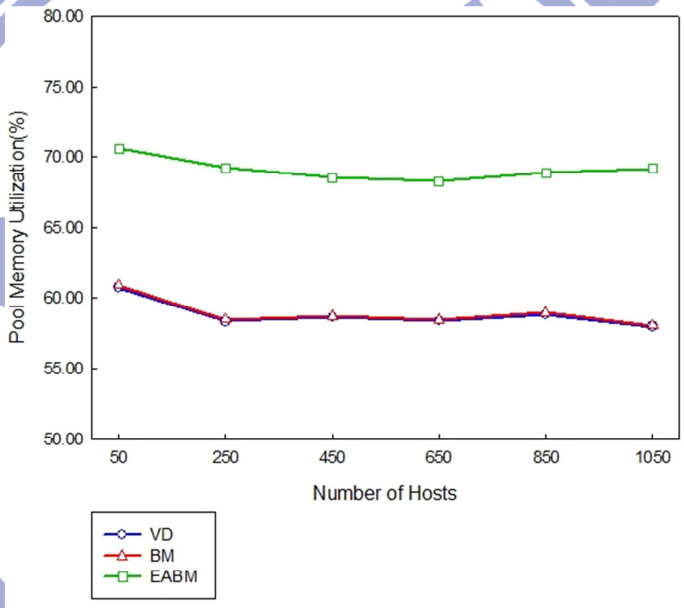


圖十一 網路頻寬維度標準差

圖十二、圖十三以及圖十四為系統在各個維度的平均資源使用率，從圖中我們可以發現，EABM 在各個維度中的平均資源使用率皆上升，這是因為關掉的那些機器上的虛擬機器都遷移到系統中其他機器上執行，工作量的數目不變同時運行的機器變少的緣故，而使得資源利用率之標準差上升。由於 W/O LB 沒有負載平衡之效果，所以在這裡我們就不拿來做比較。



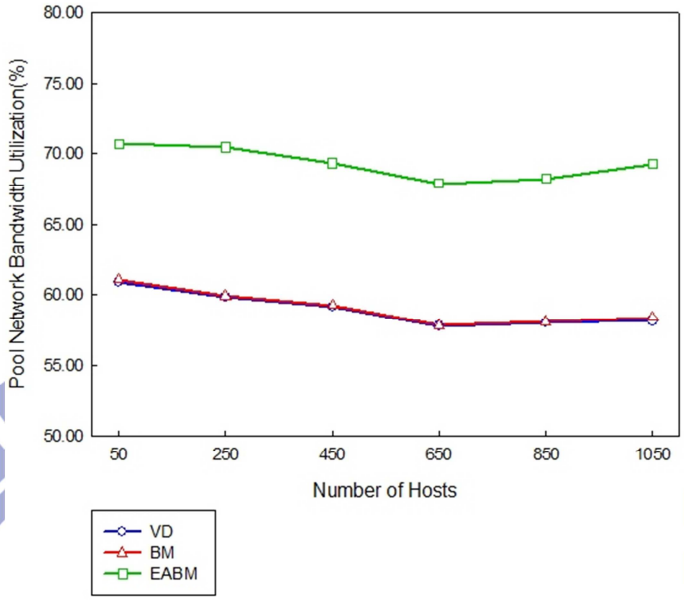
圖十二 系統平均處理器使用率



圖十三 系統平均記憶體使用率

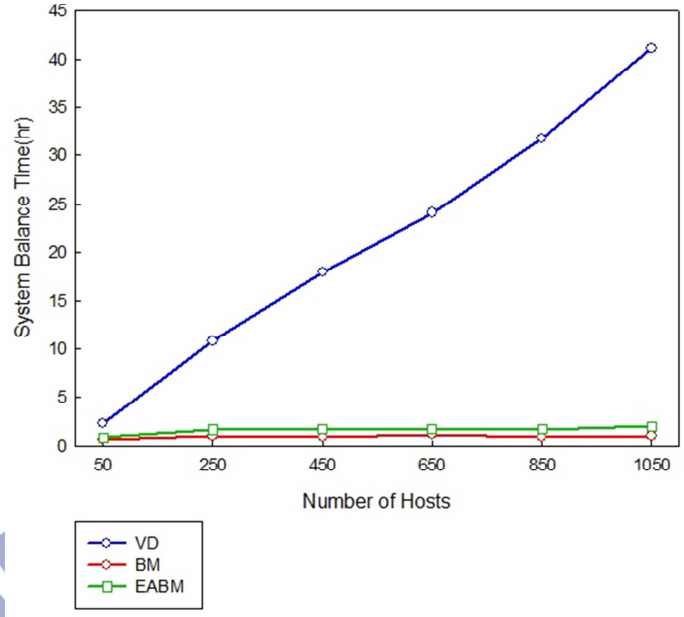
圖十五、圖十六以及圖十七分別是三個演算法讓系統達到平衡的時間比較圖、達到平衡所需遷移的虛擬機器個數以及演算法每次執行平均遷移的虛擬機器個數。平衡的時間的計算方式為負載平衡演算法第一次執行的時間開始到判定系統為平衡時所經過的時間，在這裡我們把虛擬機器遷移的時間也考慮進去，也就是說，必須在這一回合虛擬機器遷移完畢後才可以執行下一回合的負載平衡，並使用公式(1)來求算虛擬機器遷移的時間。在計算達到平衡所需遷移的機器個數中，每一個虛擬機器的遷移皆須考慮，不管虛擬機器之遷移是否為同時執行。舉例來說，在我們的方法可能存在大於一組匹配於最小權重配對中，也就是說，我們可能在同一時間內將一個以上的虛擬機器從它們存在的實體機器遷移到其他實體機器中，那這一次遷移的所有虛擬機器個數皆需要算在達到平衡所需遷移的虛擬機器總數中。演算法每次執行平均遷移的虛擬機器個數計算方式為達到平衡所需

遷移的虛擬機器總數除以演算法的執行次數。演算法的執行次數為第一次執行演算法，直到演算法判定系統達到平衡當作最後一次，這期間總共執行演算法的次數。

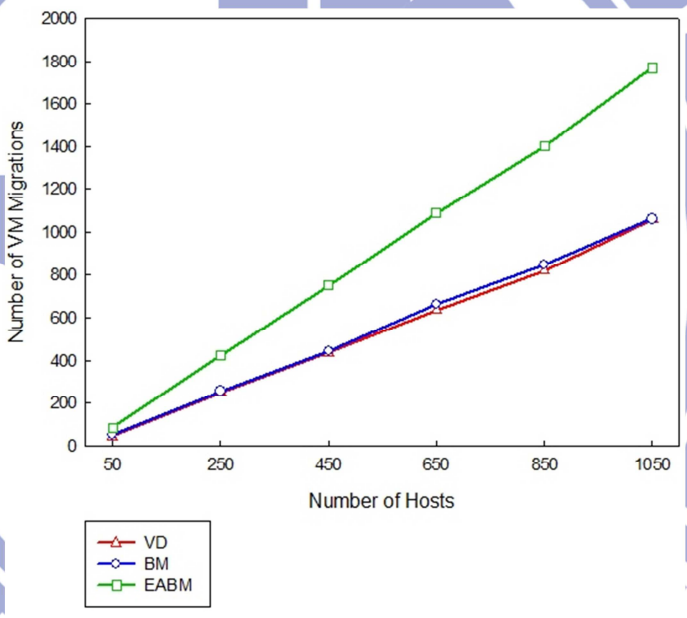


圖十四 系統平均網路頻寬使用率

當實驗的機器數量變多，所產生負載過重的實體機器也變多，從圖十六中我們可以發現，三個演算法所要遷移的虛擬機器數目也因此跟著上升。而當中又以 EABM 遷移的虛擬機器數目最多，那是因為 EABM 演算法決定關閉的機器上原本可能有虛擬機器正在執行，我們必須要先把那些虛擬機器遷移到其他實體機器上才可以將機器關閉，所以多了我們要關閉的那些機器上原本執行的那些虛擬機器遷移。在 VD 中，由於一次最多只能減少系統中一個負載過重的實體機器，因此 VD 演算法所必須達到平衡的時間自然也就跟著上升。反觀我們的演算法 BM 以及 EABM，決定我們演算法達到平衡的時間主要為系統雙分圖中存在之配對，隨著系統中的實體機器數目增加，雖然觸發點的個數增加，但非觸發點的個數也是增加的。因此，我們每次執行演算法所得的最小權重配對的匹配數亦會增加，達到平衡所需執行演算法的次數雖然上升，但不至於如同 VD 般隨著遷移之虛擬機器總數目呈線性的成長。而加入節能條件的演算法 EABM 則因為我們強迫將要關掉的實體機器改變為觸發節點，在同一情況下 EABM 中的觸發節點會比 BM 的來的要多而非觸發節點卻較少，因此所得到的最小權重配對中的匹配數目較少，所以相對地所需的平衡時間也較長。



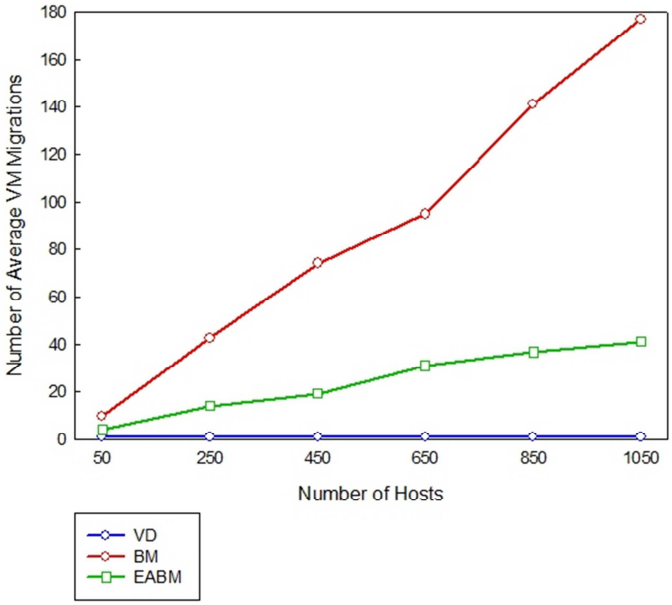
圖十五 系統達成平衡所需時間



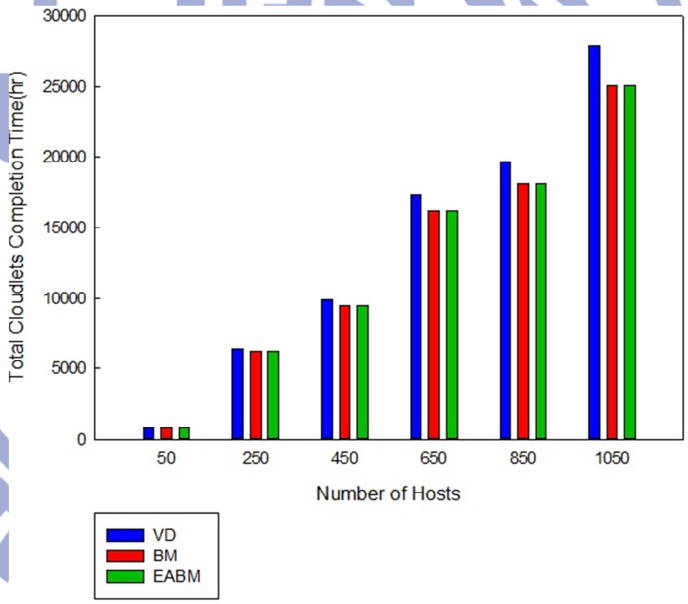
圖十六 總共虛擬機器遷移個數

隨著雙分圖中存在的匹配數越多，BM 以及 EABM 演算法一次可以處理負載過重的實體機器變多，這也是為什麼在圖十七中我們演算法 BM 以及 EABM 每次執行遷移的平均虛擬機器個數都比較多的緣故。

綜合以上所述也就是為什麼我們的演算法在圖十五中隨著機器數目上升而平衡時間幾乎都差不多的原因。對於 250 台實體機器，VD 達到平衡的時間就需要大約 10 個小時的時間，這也意味著，系統中大多數的實體機器上之虛擬機器競爭資源情況需要維持相當長的一段時間才可以得到舒緩，進而導致虛擬機器完成 cloudlet 之執行時間變長。



圖十七 演算法每次執行平均虛擬機器遷移個數



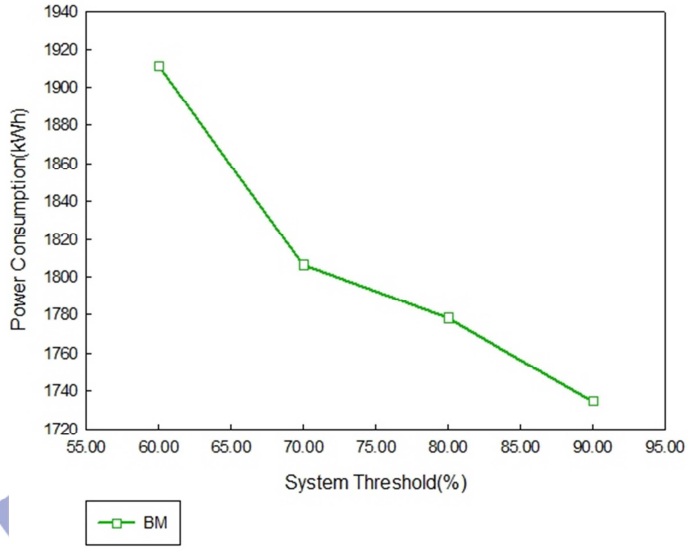
圖十八 Cloudlets 總完成時間

圖十八為系統在不同實體機器數量時負擔過重的實體機器上所有虛擬機器執行的 cloudlets 完成時間之加總。在這裡我們假設每一個虛擬機器執行的 cloudlet 長度皆相同，執行的時間皆超過 VD 演算法中系統達到平衡的時間。我們可以看出我們的演算法 BM 以及 EABM 在不同的實體機器數目中的結果都會比 VD 來的要好，分別好上 1%、3%、5%、7%、9%以及 10%。那是因為在我們

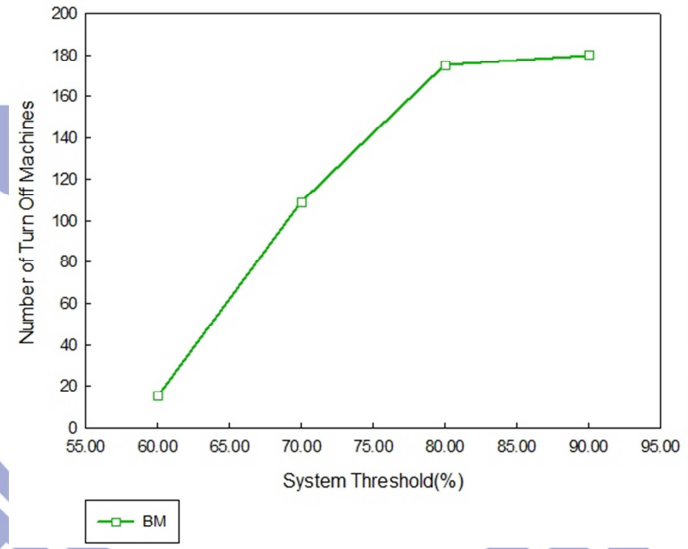
的演算法中，一次可以處理多個負載過重的實體機器，而讓這些的實體機器上的負載減輕，使得虛擬機器競爭資源的情形降低。雖然這可能同時增加了非觸發點上面的虛擬機器執行個數，但是非觸發點在我們的判定中，代表著每個維度的資源利用率皆小於設定的門檻值，也就是說，觸發點尚可以負擔更多的工作量。反觀 VD 演算法中，由於演算法每次的執行只處理一個觸發點，演算法需要執行許多次後才可以有效的減少系統中的觸發點個數，當同時考慮虛擬機器的遷移時間，演算法執行的間隔至少會等於該次虛擬機器的遷移時間，而在這期間中，其他觸發點上虛擬機器競爭資源的情形並不能馬上得到舒緩，因此，cloudlet 完成的時間也就變長了。而 EABM 與 BM 的時間差不多，EABM 的時間又比 BM 來的要長。主要的原因是因為我們為了達到節能的效果而關閉系統中某些實體機器而將其設定為觸發點，這會使得系統中的非觸發點數目減少，所得的最小權重配對改變，而配對中屬於負擔過重的機器的匹配數目下降，進而使負擔過重的實體機器上的虛擬機器的競爭情形較晚才得到舒緩。

我們使用公式(2)來計算我們的功率消耗，假設當機器處理器使用率為 100%時所消耗的能量是固定的，為 P_{max}，從公式中我們可以發現，當實體機器只要是在運行的，即便處理器的使用率為零，仍大約會消耗總能量的 70%也就是 static power，而剩餘的 30%的能量則與當下處理器的使用率有關。在這裡我們的 P_{max} 為 250W。

圖十九及圖二十分別是系統從開始到所有 cloudlets 皆完成的這段時間內所消耗的總能量(千瓦小時)以及我們總共關閉的機器數目。隨著圖二十中我們關閉的機器數目上升，系統消耗的總能量跟著下降，雖然我們強制關掉某一些機器會使得其他機器上同時運行的虛擬機器數目增加而導致處理器的使用率上升，由公式(2)中我們可以觀察出這會增加那些實體機器所消耗的能量。但是 static power 所佔的比例還是比較重，因此，消耗的功率還是會隨著關閉的機器數目上升而下降。



圖十九 系統消耗總能量



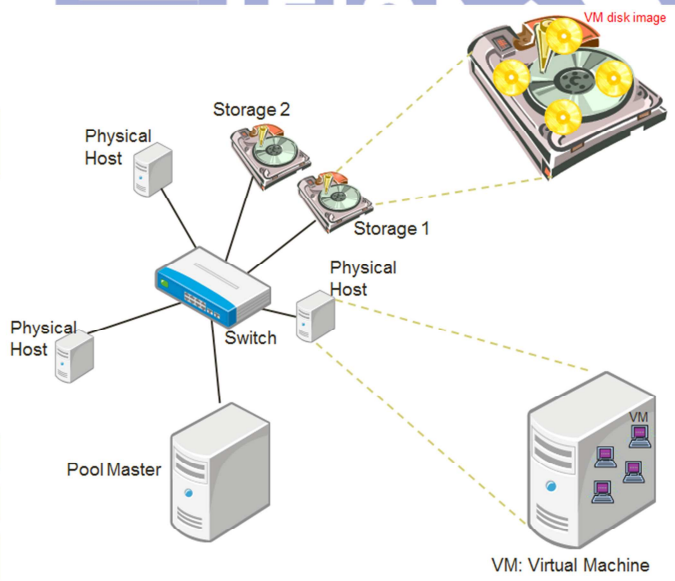
圖二十 關閉的實體機器總數

5.2 XCP 雲端環境

我們使用 Xen Cloud Platform(XCP) ver 0.1.1[22]，一個由 Xen[23]所提供開放原始碼的雲端平台作業系統搭建一個小型的雲端架構並證明我們的 BM 演算法在真實的雲端運算平台上也可以使用。透過 XCP，使用者可以輕易地搭建屬於自己的公有雲或私有雲[1]，並透過 Xen Center[24]、Open Xen Center[25]等用戶端程式來下達指令監控管理雲端內的實體機器以及虛擬機器。XCP 中也提供了

簡單的 API[26]讓使用者可以透過撰寫程式的方式來管理資料中心內的資源及機器。

我們搭建一個雲端運算環境如圖二十一，使用了四台實體機器，兩個共享儲存器以及一個 switch，實體機器的規格可以參考表格二。共享儲存器我們分別使用了一個掛載 NFS 的伺服器以及一個 Network attached storage(NAS)。在 XCP 中，每一個虛擬機器都有一個對應的 VM disk image，主要的用途就像實體機器的存儲器一樣，而虛擬機器的遷移以及將虛擬機器指派到不同的實體機器上執行皆須要先將 VM disk image 存放在共享儲存器(shared storage)中，讓系統中的所有實體機器皆可存取。在本實驗中，我們考慮了處理器以及記憶體兩個維度的資源，而虛擬機器的作業系統則分別使用了 CentOS 5.4 以及 WinXP SP3 兩種，其他細部的實驗參數列在表格二中。

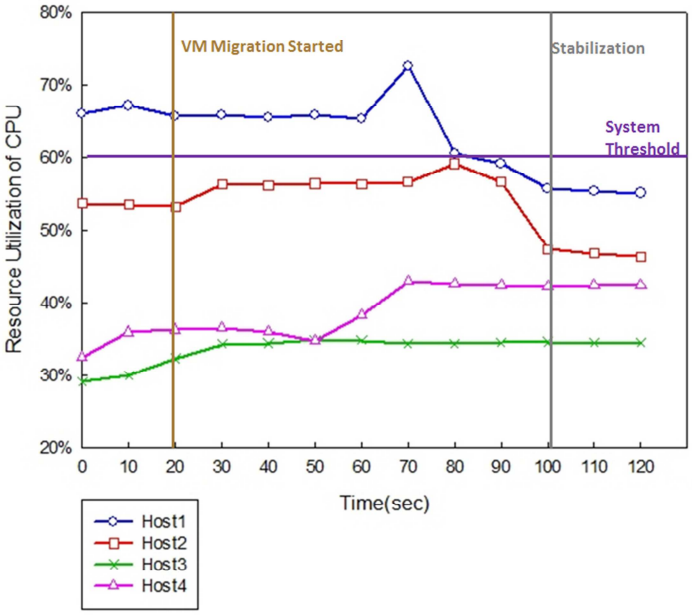


圖二十一 我們搭建的 XCP 雲端平台

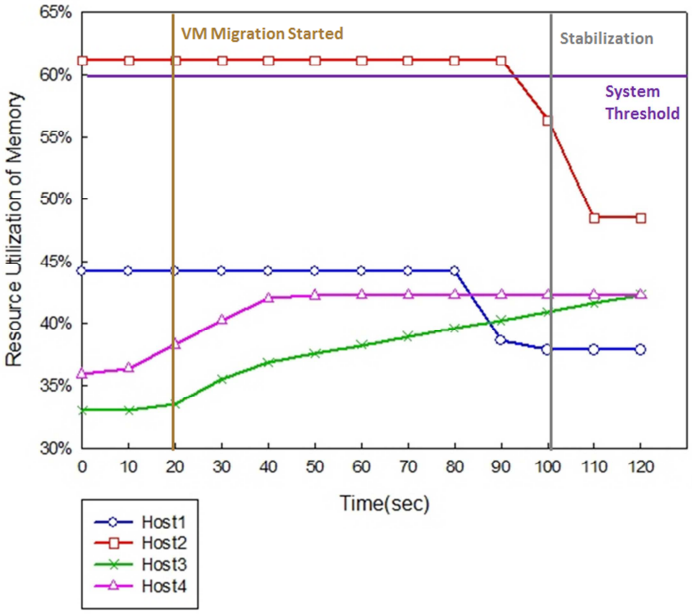
	Parameter	Value
Physical Host	CPU	Intel i5 2.67GHz (4 cores)
	Memory	8GB
Virtual Machine	CPU	8% ~ 21% of single core 2.80GHz
	Memory	8% ~ 17% of 6GB (512MB~ 1024 MB)

表格二 Testbed Evaluation 實驗參數

圖二十二以及圖二十三呈現了在我們的 testbed 上執行 BM 的結果，我們用(處理器使用率，記憶體使用率)來代表實體機器之資源使用率。對於 testbed 中的四台實體機器，Host1、Host2、Host3 以及 Host4，我們讓 Host1(67%，45%)以及 Host2(53%，62%)分別在處理器以及記憶體維度上的資源使用率超出我們定義的門檻值(在此次的實驗中門檻值為 60%)而成為觸發點，Host3(29%，34%)以及 Host4(33%，36%)由於各個維度上的資源使用率皆沒有超過門檻值而成為非觸發點。接著利用演算法求得的配對，將 Host1 與 Host2 上各一個虛擬機器分別遷移到 Host4 以及 Host3 上。在這邊我們可以觀察到，在虛擬機器遷移的過程中，遷移目標的實體機器處理器以及記憶體利利用率會因為遷移過來的虛擬機器而會隨著時間上升，直到遷移的過程完成之後達到一個定值。而虛擬機器所在的實體機器在兩個維度的資源利用率也在遷移的過程完成後才下降到一個定值。上述的情形與[15]中觀察到的虛擬機器之遷移對實體機器的影響是相符合的而在這個實驗也證明了我們的演算法的確可以使用在實際的雲端環境上來達到平衡。



圖二十二 處理器維度資源變化



圖二十三 記憶體維度資源變化

Chapter 6、結論

不同於大部分負載平衡循序處理系統中負擔過重點的方法，我們提出了一個平行處理負擔過重點的負載平衡演算法，可以在短時間內讓系統達到負載平衡，進而減少虛擬機器間的資源競爭的情形，使得雲端系統的產能上升。模擬實驗的結果顯示了對於負擔過重的那些實體機器們其上之虛擬機器所執行的應用程式完成時間，我們的演算法在 1050 台實體機器的情況下比 VectorDot 演算法的應用程式執行時間好上 10%。我們並且利用 XCP 建立由四個實體機器組成的雲端平台來實作我們的雲端負載平衡演算法 BM，同時觀察到虛擬機器在 live migration 過程中對於實體機器在各個資源維度負載造成的影響，並證明我們的 BM 演算法最後的確可以使用在實際的雲端平台上達到負載平衡的目的。我們更在負載平衡的演算法中同時考慮實體機器執行消耗的功率而提出一個節能的負載平衡演算法，在執行負載平衡的同時，將演算法一開始決定要關閉的實體機器上所有運行的虛擬機器遷移到其他的實體機器上直到實體機器上沒有任何的虛擬機器然後關閉。在達到負載平衡的效果而同時可以達到節能。

Chapter 7、參考文獻

[1] M. Armbrust *et al.*, "Above the Clouds: A Berkeley View of Cloud Computing," technical report No.UCB/EECS-2009-28, EECs department, U.C. Berkeley, Feburary 2009.

[2] "Amazon EC2," http://aws.amazon.com/ec2/.

[3] "Google APPs," http://www.google.com/apps/intl/en/business/index.html.

[4] "Microsoft Windows Azure," http://www.microsoft.com/windowsazure/.

[5] G. Wang and T. S. Eugene Ng, "The Impact of Virtualization on Network Performance of Amazon EC2 Data Center," in *IEEE INFOCOM*, March 2010, pp.1-9.

[6] S. K. Garg, C. S. Yeo, A. Anandasivam and R. Buyya, "Environment-conscious scheduling of HPC applications on distributed Cloud-oriented data centers," *Journal of Parallel and Distributed computing*, vol.71, pp.732-749, January 2010.

[7] B. Li, J. Li, J. Huai, T. Wo, Q. Li, and L. Zhong, "EnaCloud: An Energy-saving Application Live Placement Approach for Cloud Computing Environments," in *Proceedings of the IEEE International Conference on Cloud Computing*, September 2009, pp.17-24.

[8] R. Buyya, A. Beloglazov and J. Abawajy, "Energy-Efficient Management of Data Center Resources for Cloud Computing: A Vision, Architectural Elements, and Open Challenges," in *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, July 2010, .

[9] S. Srikantaiah, A. Kansal and F. Zhao, "Energy Aware Consolidation for Cloud Computing," in *Proceedings of the Conference on Power Aware Computing and Systems*, December 2008, p.10-10.

[10] Y. Lee and A. Zomaya, “Energy efficient utilization of resources in cloud computing systems”, *The Journal of Supercomputing*, pp.1-13, March 2010.

[11] T. Chieu, A. Mohindra, A. Karve and A. Segal, “Dynamic Scaling of Web Applications in a Virtualized Cloud Computing Environment,” in *Proceedings of IEEE International Conference on e-Business Engineering*, October 2009, pp.281-286.

[12] Y. Zhao and W. Huang, “Adaptive Distributed Load Balancing Algorithm based on Live Migration of Virtual Machines in Cloud,” in *Proceedings of the Fifth International Joint Conference on INC, IMS and IDC*, August 2009, pp.170-175.

[13] “EUCALYPTUS,” http://open.eucalyptus.com/

[14] S. Nakrani and C. Tovey, “On Honey Bees and Dynamic Server Allocation in Internet Hosting Centers,” *Adaptive Behavior – Animals, Animats, Software Agents, Robots, Adaptive Systems*, vol. 12, September 2004, pp.223-240 .

[15] A. Singh, M. Korupolu and D. Mohapatra, “Server-Storage Virtualization: Integration and Load Balancing in Data Centers,” in *Proceedings of the International Conference for High Perfirmance Computing, Networking, Storage and Analysis*, November 2008, pp.1-12.

[16] K. Sato, H. Sato and S. Matsuoka, “A Model-Based Algorithm for Optimizing I/O Intensive Applications in Clouds Using VM-Based Migration,” in *Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, May 2009, pp.466-471.

[17] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Wareld, “Live Migration of Virtual Machines,” in *Proceedings of the 2nd ACM/USENIX Symposium on Networked Systems Design and Implementation* , May 2005, pp.273-286.

[18] J. Zheng , T. S. Eugene Ng and K. Sripanidkulchai., “Workload-Aware Live Storage Migration for Clouds,” in *Proceedings of the 7th ACM*

SIGPLAN/SIGOPS international conference on Virtual execution environments, March 2011, pp.133-144, .

[19] K. Bubendorfer and J. H. Hine, "A Compositional Classification For Load-Balancing Algorithms," technical report, No.CS-TR-99-9, Victroia University of Wellington, July 1998, .

[20] H. W. Kuhn, "The Hungarian Method for The Assignment Problem," *Naval Research Logistics Quarterly*, vol.2 , pp. 83-97, December 1955.

[21] R. Buyya, R. Ranjan and R. N. Calheiros, "Modeling and Simulation of Scalable Cloud Computing Environments and the CloudSim Toolkit: Challenges and Opportunities," in *Proceedings of the 7th High Performance Computing and Simulation Conference*, June 2009, pp.1-11.

[22] "Xen Cloud Platform," http://www.xen.org/products/cloudxen.html

[23] "Xen," http://www.xen.org/

[24] "XenCenter," http://community.citrix.com/display/xs/XenCenter

[25] "Open XenCenter," http://www.openxenmanager.com/

[26] "Xen API," http://wiki.xensource.com/xenwiki/XenApi