多計算叢集排程管理系統暨動態負載預測演算法

吳長興
國家高速網路與計算中心
c00csw00@nchc.org.tw

潘怡倫
國家高速網路與計算中心
c00ser00@nchc.org.tw

摘要

近年來個人電腦組成的個人電腦叢集，愈趨於普遍，許多學術研究單位及業界，都可能擁有1座以上的計算叢集。一般使用者都會想要使用比較快的叢集電腦，而導致資源分配不均。要如何充分利用多座計算叢集，並且能讓使用者可以方便使用？這些都是現今高速計算的重要課題之一，因此國網中心分散式計算團隊研發一套多叢集排程管理系統(Multi-Cluster Scheduler)，它是建構在原來的叢集排程軟體之上，亦即不需要更動現有的叢集架構，再加上我們所設計的動態預測排程演算法，其能夠預測叢集的負載程度，來調配工作，使多座叢集可以達到負載平衡，而讓資源更有效率的使用。

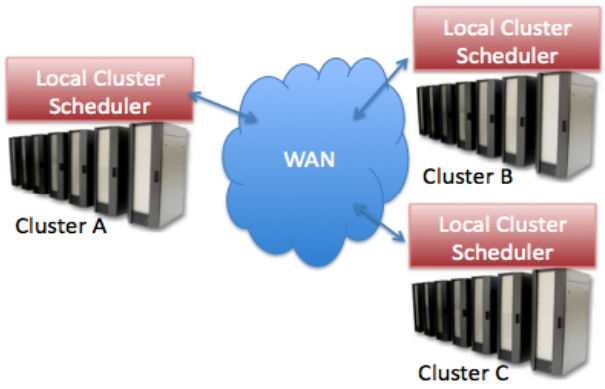
關鍵詞：**高速計算、叢集計算、multi-cluster scheduler、cluster**

一、前言

在計算資源需求劇增的今日，計算資源若僅依靠大型計算主機供應已不敷使用了，因此相形之下，個人電腦叢集(PC cluster)是由價格低廉且容易購得的個人電腦相關零件，來組裝平行電腦，來處理的大量資訊與計算，因為它的強大效能與經濟性，造成個人電腦叢集的歷史雖短，進展卻相當快速[1]。因此吸引了一批又一批的研發人員投入，並陸續地將現有的或是新購的個人電腦串連起來，使得個人電腦叢集在世界各地如火如荼地推展開來。

但因為經費或空間等諸多限制，相同一個單位往往有許多時期建構的計算叢集，因其架構不同以及硬體設備不同難以整併，而新建置且高效能的叢集一定會吸引許多使用者使用，導致許多計算效能略低的叢集資源閒置浪費，造成資源分配不均的狀況發生。而一般業界或研究單位解決的方法，都是希望從格網的架構上的Meta-Scheduler或Resource Broker下手，由於Grid架構較為龐雜，再加上天生架構障礙傳遞資源訊息較為緩慢，往往Grid的Meta-Scheduler或Resource Broker接收到的訊息時，其資訊已經不符合系統狀況，進而造成資源分配不佳的狀況發生[2]、[3]、[4]。

所以為了設法改善此狀況，國網中心分散式計算團隊開發了一套多叢集排程管理系統（Multi-Cluster Scheduler，簡稱為MCS），不需透過任何中介軟體，並追求速度以及排程的精確性，MCS直接建構在多組計算叢集之上，使用者將可以透過MCS來交付工作，並且監控每一座計算叢集的資源， MCS會依照每一座計算叢集的負載程度，自動分配工作及依各叢集的負載狀況，以達資源充分利用負載平衡。



圖一：多叢集(Multi-Cluster)環境

而多叢集就是透過WAN或著一些高速的網

路設備將多組計算叢集給串起來[5]，每一座計算叢集都裝有各自的排程管理系統(Local Cluster Scheduler，簡稱為LCS)，例如: OpenPBS [6]、PBS Pro [7]以及LSF [8]等，如圖一所示。

此外，我們還為了MCS設計了『動態負載預測演算法』，它會根據每一個工作需求，例如需要的CPU個數、需求記憶體大小…等，以及每一座叢集工作的歷史資訊、即時資源狀態…等。來判斷最適合的叢集，並且將工作交付到最適的叢集計算資源上。

本篇文章結構如下：第一部分為前言、第二部分為多叢集架構、第三部分為Multi-Cluster Scheduler核心架構、第四部分則為所提出之核心排程演算法-動態負載預測(DLPS)演算法、第五部分為實驗、第六部分為**應用實例**，與最後第七部分為結論。

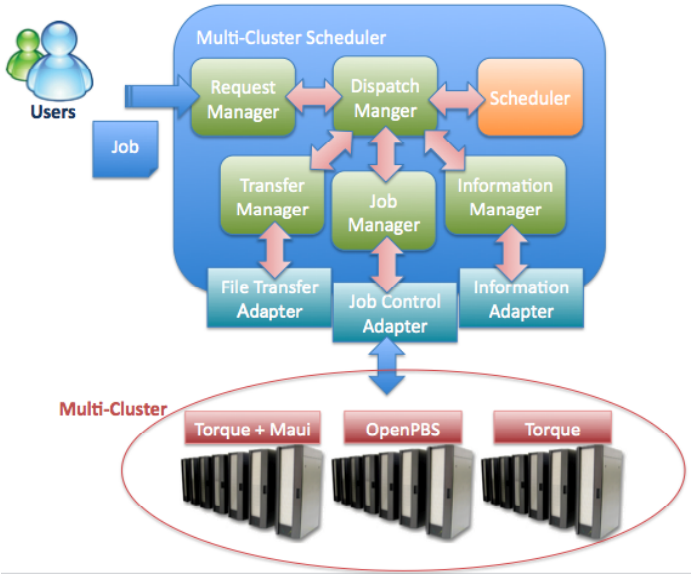
二、多叢集架構

所提出之Multi-Cluster Scheduler（簡稱為MCS），是修改自Gridway [9]、[10]。這套格網架構上的Meta-Scheduler，主要是替換其底層與格網溝通的模組，及依照多叢集的架構修改其模組，並且加上國網中心分散式計算團隊所自行研發的排程演算法，所組合而成。此外，本團隊更有以此為架構基礎，延伸更深入之開發[11]。

MCS整體架構，如圖二所示，主要包含兩層，第一層為MCS，第二層為Multi-Cluster。在每一座叢集之上，都會安裝有各自的排程管理系統(Local Cluster Scheduler，簡稱為LCS)。

MCS其主要透過Information Adapter與每一座叢集上的LCS聯繫，以取得叢集的資源資訊，其中包含：計算節點個數、可用計算節點、工作佇列內工作的個數…等。當使用者透過MCS交付工作時，工作會先進入MCS裡的工作佇列列隊，然後MCS為利用從底層LCS所獲得的資源資訊，自動判斷哪些叢集適合使用者的工作，再加上所有叢集負載考量，挑選出一座最適合的叢集。接著MCS會透過File Transfer Adapter以及Job Control Adapter，將使用者的工作以及執行工作所需的檔案，一併交付至遠端的叢集中。並透過Job Control Adapter監控工作在LCS的執行狀態，當工作執行完畢後，通知File Transfer Adapter將工作執行結果及輸出檔案，取回到MCS所在的機器之上。

第二層為叢集端，每一座Cluster上面會執行著LCS。而MCS會定時透過Information Adapter與LCS來收集與傳遞資源負載資訊並且監控Job執行狀態，接著將所有的資訊轉成XML的格式傳送至MCS，提供給MCS來透過資源使用狀況，來平衡多座計算叢集負載。此外，在每一座MCS控制的叢集上，還能保有其原本的LCS，使用者還是可以直接透過LCS來執行工作。



圖二：多叢集排程管理系統(Multi-Cluster scheduler)架構

三、MCS核心架構

多叢集排程管理系統核心架構，如圖二所示，其核心主要由六個部份所組成，其中包括：Resource Manager、Dispatch Manager、Scheduler、Transfer Manager、Job Manager與

Information Manager。專門負責接收使用者的需求的為Request Manager模組，其主要監聽Socket Port，等候使用者透過指令操作方式來建立連線交付指令及工作。當Request Manager模組接收到指令之後，會先分析所交付的指令內容做相對應處理，假如為交付工作的指令就會依照使用者交付的工作描述檔內容，將工作描述檔轉換成工作描述物件，並把工作描述物件交給Dispatch Manager模組。

Dispatch Manager模組的主要功能就是在各個模組間傳遞工作物件，當Dispatch Manager模組從Request Manager模組中接收到工作描述物件後，首先會將其放入工作佇列中，等到Scheduler模組分配資源給它後，Dispatch Manager模組會將工作描述物件移交給Transfer Manager模組以及Job Manager模組。

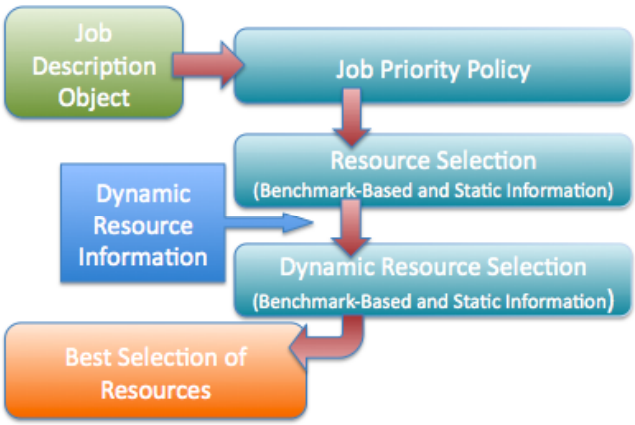
Transfer Manager模組接收到工作描述物件，會先讀取工作描述物件的內容，分析執行這個工作所需要的檔案（執行檔、輸入檔…等），以及分配到哪一座叢集，會先將工作所需要的檔案透過File Transfer Adapter將所需檔案傳送到遠端的叢集上。等到Transfer Manager模組傳輸工作所需的檔案，接著Dispatch Manager會通知Job Manager模組將工作描述檔轉換成LCS所認識的工作描述檔格式傳送到遠端LCS執行。

四、核心排程演算法

分散式計算團隊所設計DLPS（Dynamic Loading Prediction Scheduling Algorithm）動態負載預測演算法，並且安裝在上一段所提到的Scheduler模組之中，Scheduler模組為MCS的核心模組 Scheduler模組會根據透過Information Manager模組收集到的底層計算資源使用的狀況資訊（如：每一座叢集的CPU Cores總數、已使用的CPU Cores數目、節點負載…等），再加上使用者工作需求，進而挑選最適合使用者工作計算的叢集，並將工作移交過去進行計算。

其實作的動態負載預測演算法的Scheduler模組一共分成三個階段，如圖三所示，**第一個階段**為挑選所要執行的工作，亦即為依照使用者優先權、使用的資源多寡來排列工作執行順序。常用的演算法，如 FCFS, Round-Robin, Backfill, Small Job First以及Big Job First等。在原始的Gridway的Scheduler模組中，僅實作了Round-Robin演算法，並且直接選取符合的資源就將工作直接分配。此作法雖然簡便，但是稍嫌不足，會導致大量工作消耗率低。因此在所提出的動態負載預測演算法中，設計了第二與第三階段，繼而可找出最適的分散式計算資源，並提升計算效能。

在經過第一個階段之後，隨即進入我們設計的演算法重點，在動態負載預測演算法的**第二階段**是使用靜態資訊來進行資源的挑選，會依照一些靜態的資訊，如叢集HPL Benchmark的數值大小、網路設備速度以及平行檔案系統等，以及使用者的工作描述檔裡的資源需求，將資源過濾並排列，並存成一個資源串列，接著將工作描述物件及資源串列傳遞到下一個階段。



圖三：核心排程演算法步驟

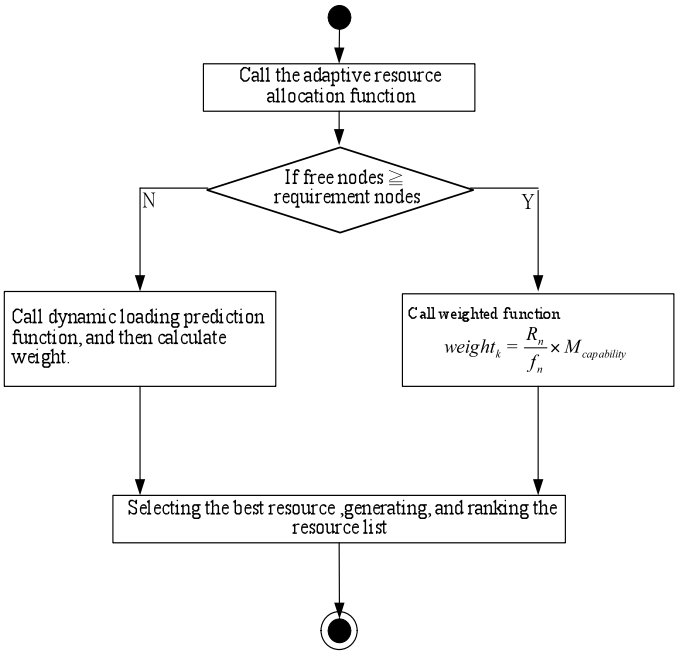
接著，分散式計算團隊實作的Scheduler模組的**第三階段**是動態負載預測演算法(DLPS)，其架構如圖四所示，動態負載預測演算法如下所述：

第一步驟，根據第二階段所傳遞過來的資源串列以及工作描述物件，並依據由Information

Manger傳遞過來的及時資源資訊，將資源串列再進一步篩選濾掉無法提供給使用者的計算叢集。

第二步驟，當資源串列裡的叢集剩餘的CPU及節點個數大於或等於使用者需求，接著需要依照下列的公式(1)，計算出資源串列裡的各個叢集資源的權重。

(1) $weight_k = \frac{R_n}{f_n} \times M_{capability}$ 其中 R_n 為所需的CPU Core數量，f_n 為叢集所剩的CPU Core數量，$M_{capability}$ 為依照每一座計算叢集的計算能力所給予的變數(依照HPL結果，並將某一座計算叢集A為1，其他的叢集依照其算比例)。



圖四：核心排程演算法步驟

當資源串列裡的叢集剩餘的CPU及節點個數小於或等於使用者需求，接著就要依照下列的公式(2)來計算每一座計算叢集的權重。

(2) $Weight_c = \lambda \times \frac{JEF_i}{\sum_{i=1}^{n} JEF_i} + (1-\lambda) \times \frac{EBL_i^*}{\sum_{i=1}^{n} EBL_i^*}$

其中 JEF_i *(job expansion factor)*是每一座計算叢集LCS所統計出來的資料，如公式(3)所示，主要的意義是在那一座計算叢集的每一個工作在工作佇列等待的時間加上執行時間，再除上等待時間限制的平均值。

(3) $JEF_i = \frac{QueuedTime + RunTime}{WallClockLimit}$

而 EBL_i^* (*EstBacklog*)是每一座計算叢集LCS所統計出來的資料，如公式(4)所示，

(4) $EBL_i^* = (\frac{QueuePS \times CPUAccuracy}{TotalJobsCompleted}) \times (\frac{TotalProcHours \times 3600 \times AvailablePorcHours}{DedicatedProcHours})$

*QueuePS*是指在工作佇列裡等待的工作的時間總和，*CPUAccuracy*是指工作實際執行的CPU時間，*TotalJobsCompleted* 是指完成的工作總數，*ToatlPorcHours* 是指完成所有正在計算的工作所需的小時數，*AvailableProcHours* 是指目前可用計算資源的小時數，*DedicatedProcHours*是指所有可用計算資源的小時數。

根據上述(4)公式，將每一座計算出來的 EBL_i^* 以及 JEF_i 數值，算出每一座叢集的權重。

第三步驟，不管剩餘的資源是否大於使用者的需求，都可以依照步驟二找出每一座計算資源的權重，並將資源串列依照權重順序排列，找出可以最快速將使用者工作完成的計算叢集。

在完成動態負載預測演算法的所有流程，會得到出一個最佳的計算資源，接著MCS就會將使用者的工作自動送到那個計算資源去執行。

五、實驗

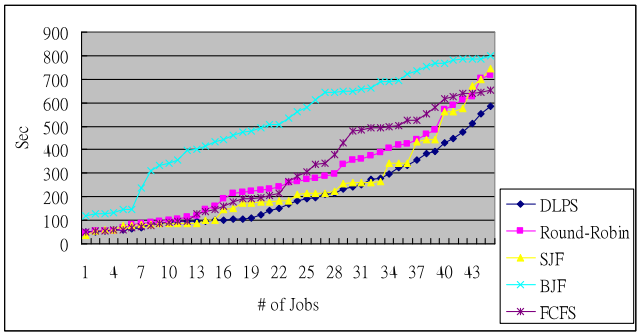
本章節實驗部分，分散式計算團隊使用三座計算叢集做實驗平台，詳細列表如下表一所示，兩座64bit叢集以及一座32bit叢集，分別為

Nacona、Opteron以及Formosa Ext.2。

Resource	CPU Model	Memory	CPU Num	Nodes	Job Manager
Nacona	Intel(R) Xeon(TM) CPU 3.20GHz	4G	16	8	Torque
Opteron	AMD Opteron Processor 248	4G	16	8	Moab
Formosa Ext. 2	Intel(R) Xeon(TM) CPU 3.06GHz	4G	22	11	Maui

表一： Summary Environment Characteristics of NCHC Grid-enabled Resources

於實驗中，將使用幾個演算法當做此次實驗之對照組：Round-Robin、 Short-Job-First (SJF)、Big-Job-First (BJF) 以及First-Come-First-Serve (FCFS)。接著進行隨機化產生多組CPU Cores需求不一的工作群，每一個工作都是使用*Computational Fluid Dynamics* (CFD) *Message Passing Interface* (MPI)計算流體力學的工作[12]，將每一組放到各個演算法去執行，取其平均，實驗結果如圖五所示。其下圖五中縱軸表示工作消耗的時間(單位：秒)，橫軸則表示所消耗之工作數量。由圖五可知，使用動態負載預測演算法(DLPS)演算法確實比其他演算法更有效率，完成工作所消耗時間也較其他演算法少，此一特性，尤其遇到所提交工作數量多的時候更為顯著。



圖五：排程演算法比較

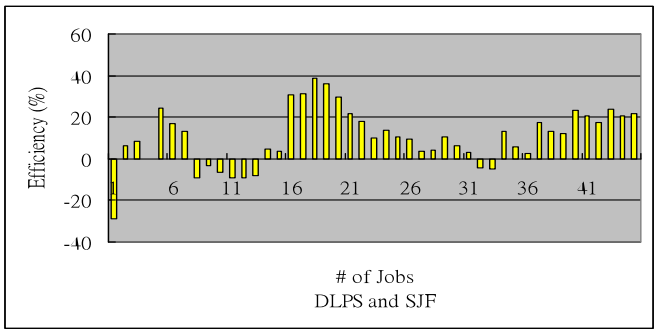
接下來，則將討論到每個演算法效率(*Efficiency*)衡量與比較如圖六、七、八、九所示。

此外，效率衡量指標之公式，如公式(5)所示，

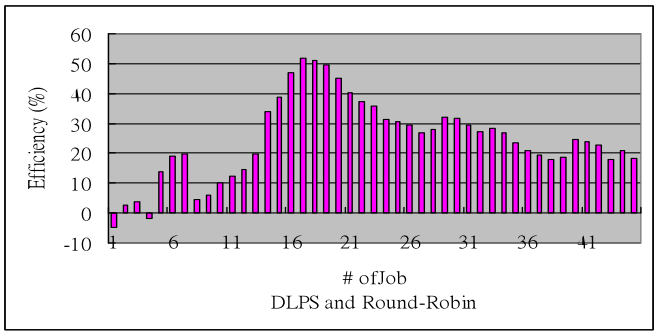
$$(5)\ \frac{TotalTimeConsume_{Algorithm} - TotalTimeConsume_{DLPS}}{TotalTimeConsume_{Algorithm}} \times 100\%$$

其中*TotalTimeConsume*是指所選用之演算法消耗提交工作所需的總時間。故公式(5)則是讓每個演算法都與我們所設計之演算法比較。假設所得之數據為正值，則表示所設計之DLPS演算法比該演算法更為有效率。反之，當所得數據為負值時則表示DLPS較無效率。

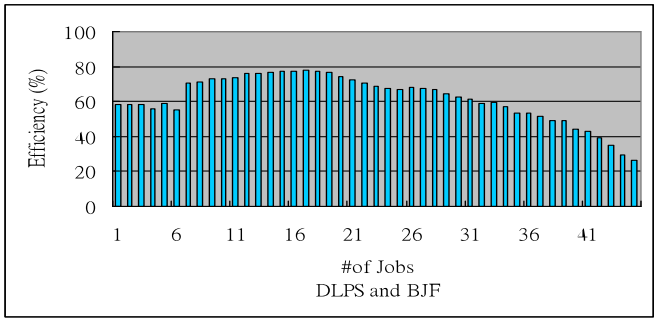
根據下圖六實驗數據所知，當所提交的工作數為小量的狀況之下，如工作個數小於16個工作的時候，所提出DLPS演算法並沒有很突出的效率表現。比SJF (Short-Job-First)所需的時間還要長。此結果是相當合理的，其原因則是因為SJF是將小的Job先計算，導致一開始計算時效率會最好，但是等到計算量與工作數明顯開始增加的時候，根據實驗圖表，所提出之DLPS演算法有較佳成顯著的成效結果。此外，SJF為人所詬病之缺點，就是相對的較容易導致大量工作提交時，需多處理器(Muitl-Core)工作會發生等不到足夠計算資源的狀況，也較容易發生。



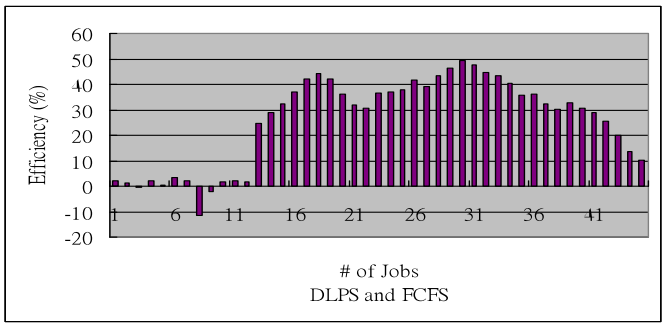
圖六：Compare the Efficiency of DLPS with SJF



圖七：Compare the Efficiency of DLPS with Round-Robin



圖八：Compare the Efficiency of DLPS with BJF



圖九：Compare the Efficiency of DLPS with FCFS

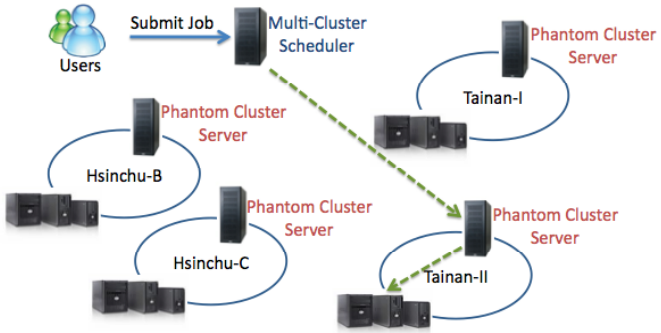
由上述可知，綜合以上的效能圖表六、七、八、九可知，DLPS在叢集計算資源內各機滿載時，更容易達到其最高使用效率。所以，提出之演算法DLPS適合處理大量工作提交。此一特性，則是非常適合應用於多叢集分散式計算以及計算格網環境中。

六、應用實例

此章節中，將討論以多叢集排程管理系統（Multi-Cluster Scheduler，簡稱為MCS）為架構基礎，延伸更深入開發之應用實例。其中，以幻影叢集『Phantom Cluster』[13]軟體為例。

『Phantom Cluster』是以DRBL(Diskless Remote Boot in Linux) [14]為基礎，上面安裝Torque [15]當做排程軟體，全天開啟服務，使用者可以在任何時間將工作交付到安裝有Phantom Cluster軟體的伺服器之上，而白天為電腦教室的個人電腦群，晚上或著假日無人上課的時段，安裝有Phantom Cluster軟體的伺服器會自動喚醒電腦教室的個人電腦，並使用無碟方式開機 (不破壞電腦教室原本的系統環境)，將電腦教室的個人電腦串成一座高速的叢集電腦。

目前在國網中心有三個事業群分別在新竹、台中、台南科學園區，每個事業群各擁有1~2間電腦教室，每間電腦教室都使用Phantom Cluster軟體將其轉變成一座電腦叢集，每間電腦教室雖然可以各自獨立當叢集使用，但無法互相支援。因此我們使用MCS串起北中南三群的電腦教室，組成『Phantom Multi-Cluster』，如圖十所示，使所有的可用資源可以互相支援，讓使用者工作計算更有效率。



圖十： Phantom Multi-Cluster 架構圖

七、結論

國網中心分散式計算團隊開發了一套多叢集排程管理系統（Multi-Cluster Scheduler，簡稱為MCS），其特點包括不需透過任何中介軟體，並追求速度以及排程的精確性，MCS直接建構在多組計算叢集之上，使用者將可以透過MCS來交付工作，並且監控每一座計算叢集的資源，MCS會依照每一座計算叢集的負載程度，自動分配工作及依各叢集的負載狀況，以達資源充分利用負載平衡。此外，並成功解決傳遞資源訊息較為緩慢、訊息過時等，進而造成資源分配不佳的問題發生。

其核心演算法-出動態負載預測演算法(DLPS)，更可輕易的達到資源充分利用與負載平衡。根據實驗結果可知，DLPS在處理大量工作提交時，有相當優異的效率。尤其是，在叢集計算資源內各機滿載或是工作提較數量大時，更容易達到其最高資源使用效率。此特性，是非常適合應用於多叢集分散式計算與計算格網環境中。

八、參考文獻

[1] Hsu, C.-H.; Lo, T.-T. and Yu, K.-M. "Localized communications of data parallel programs Qn multi-cluster grid systems," *European Grid Conference (EGC' 05), Lecture Notes in Computer Science*, vol. 3470, pp. 900-910, June 2005.

[2] R. AI-Khannak, and B. Bitzer, "Load Balancing for Distributed and Integrated Power Systems using Grid Computing," *International Conference on Clean Electrical Power (ICCEP),* 22-26 May, 2007, pp. 123-127.

[3] Javadi, B.; Akbari, M. and Abawajy, J. "Performance analysis of heterogeneous multicluster systems," in Proceedings of the *International Conference on Parallel Processing (ICPP' 05)*, pp. 493-500, 2005.

[4] J. Schopf, "A General Architecture for Scheduling on the Grid, "*Journal of Parallel and Distributed Computing,* special issue, April 2002, p. 17.

[5] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. Freund, "Dynamic Matching and Scheduling of a Class of Independent Tasks onto Heterogeneous Computing Systems," *In the 8th IEEE Heterogeneous Computing Workshop (HCW' 99)*, 1999, pp. 30 – 44.

[6] http://www.mcs.anl.gov/research/projects/openpbs/

[7] http://www.tgc.com/sponsors/veridian/veridian.html

[8] http://www.zdv.uni-mainz.de/cms-extern/lsf/lsf6.0/html/license_scheduler_6.0/lsf_license_scheduler.html

[9] E. Huedo, R. S. Montero and I. M. Llorent, "Evaluating the Reliability of Computational Grids from the End User's Point of View," *Journal of Systems Architecture* 52 (12), pp. 727-736, 2006.

[10] http://www.gridway.org/doku.php

[11] Yi-Lun Pan, Chang-Hsing Wu, and Weicheng Huang, "Grid Widgets: The Lightweight Approach to Acquiring Grid Services with Resource Broker in Grid Environment," the International Conference on Grid Computing Applications (GCA), Las Vegas, NV, July 2009.

[12] W. Huang, "Dynamic Computing Power Balancing for Adaptive Mesh Refinement Applications," *International Parallel Computational Fluid Dynamics Conference' 02*, pp. 411- 418, Nara Japan, April, 2002.

[13] http://phantom.nchc.org.tw/

[14] http://drbl.nchc.org.tw/

[15] http://www.clusterresources.com/products/torque-resource-manager.php