行政院國家科學委員會專題研究計畫　成果報告

異質性平行計算網路下支援 SPMD 程式之資源配置工作管理與資料重組技術之研發(II)

計畫類別：個別型計畫
計畫編號：NSC93-2213-E-216-028-
執行期間：93 年 08 月 01 日至 94 年 07 月 31 日
執行單位：中華大學資訊工程學系

計畫主持人：許慶賢

計畫參與人員：陳世璋、翁銘遠、藍朝陽

報告類型：精簡報告
報告附件：出席國際會議研究心得報告及發表論文
處理方式：本計畫可公開查詢

中　華　民　國 94 年 10 月 28 日

行政院國家科學委員會補助專題研究計畫 ■ **成 果 報 告**
□ **期中進度報告**

異質性平行計算網路下支援 SPMD 程式之資源配置工作管理與資料重組技術之研發(II)

計畫類別：■ 個別型計畫　　□ 整合型計畫
計畫編號：NSC　93－2213－E－216－028－
執行期間：　93 年 08 月 01 日至 94 年 07 月 31 日
執行單位：　中華大學資訊工程學系

計畫主持人：許慶賢 助理教授
共同主持人：
計畫參與人員：陳世璋、翁銘遠、藍朝陽 中華大學資訊工程學系研究生

成果報告類型(依經費核定清單規定繳交)：□精簡報告　■完整報告

本成果報告包括以下應繳交之附件：
□赴國外出差或研習心得報告一份
□赴大陸地區出差或研習心得報告一份
■出席國際學術會議心得報告及發表之論文各一份
□國際合作研究計畫國外研究報告書一份

處理方式：除產學合作研究計畫、提升產業技術及人才培育研究計畫、列管計畫及下列情形者外，得立即公開查詢
□涉及專利或其他智慧財產權，□一年□二年後可公開查詢

中　華　民　國　94 年　10 月　14 日

行政院國家科學委員會專題研究計畫結案報告

異質性平行計算網路下支援 SPMD 程式之資源配置工作管理與資料重組技術之研發

中文摘要

在平行計算系統中所發展出來的 SPMD 計算模式已在許多大量資料運算及高性能科學應用中被廣泛的接受。隨著網路技術的進步與頻寬快速成長，加上經濟成本與效益的考量，異質性網格計算已成為大量資料與科學計算在平行與分散式計算平台以外的另一種選擇。因此，如何可以有效率的移植 SPMD 程式透通於異質性的計算平台上以保有其程式演算的最佳效能成了最值得討論的問題。

本報告是有關於在研發異質性平行計算網路下支援 SPMD 程式之資源配置工作管理與資料重組技術之描述。在這一個計畫中，我們針對 SPMD 平行資料程式在異質性多叢集系統提出有效率的資源配置、工作排程方法、以及通訊局部化技術。在資源配置與工作管理方面，我們分別針對處理器計算能力的異質與網路頻寬的異質，提出 SCTF (Shortest Communication Time First)工作排程演算法，並且開發以網頁為基礎的管理工具，在兩套 PC_Cluster 系統上實作出資源配置、監督與工作排程系統。在通訊局部化技術方面，透過邏輯處理器與資料對應的技術，可以降低不同叢集系統之間處理器的通訊成本。另外，針對異質性平行計算網路下的負載平衡問題，我們也提出了一套以基因演算法為基礎的模糊理論，提升異質性分散式計算平台的排程效率。本計畫的成果可以有助於增加異質性叢集系統的產能以及 SPMD 平行程式在該系統的執行效率。對於工作排程、協調配置與資源管理的問題，我們測試了幾套國外著名的工作排程系統，進行不同平台的測試，實驗結果顯示，SCTF 提升了系統平均產能(throughput)、縮短了工作執行的平均回覆時間(turnaround time)。

關鍵詞：異質性計算、SPMD、分散式記算、平行演算法、資源配置、工作管理、資料重組、網格計算。

Design and Implementation of Resource Allocation and Job Scheduling for Supporting SPMD Programs on Heterogeneous Parallel Computing Networks

Abstract

The SPMD programming model evolved from parallel computing system has become a widely accepted paradigm for massive computing and high performance scientific applications. With the progressing of network technology, the rapid growth of communication bandwidth and the consideration of cost-effective ratio, grid-computing environment has become the other choice for many scientific applications in parallel and distributed computation. Thus, how to execute an SPMD program on heterogeneous computing platforms efficiently is a common challenge.

This report presents the development and implementation of resource allocation and job scheduling for supporting SPMD programs on heterogeneous parallel computing networks. In this project, we have proposed an efficient communication technique for SPMD parallel programs in heterogeneous multi-cluster systems. Utilizing the logical processor to data mapping technique, inter-cluster communications between physical processors can be reduced. Besides, we have also proposed a genetic-fuzzy logic based approach for dynamic load balance on heterogeneous parallel computing systems. The results of our work facilitate increasing the throughput of heterogeneous distributed memory environments and the performance of SPMD parallel programs executing on such systems. For job scheduling, co-allocation and resource management, we have proposed an SCTF (Shortest Communication Time First) task scheduling algorithm. We also developed a web-based resource monitoring tool upon two PC cluster systems. We have tested some major tools that developed by other research teams on different platforms. The experimental results show that SCTF outperforms Beaumont's method in terms of lower average turnaround time, higher average throughput, less processor idle time and higher processors' utilization.

Keywords: Heterogeneous Computing, SPMD, Distributed Computing, Parallel Algorithm, Resource Allocation, Task Management, Data Reconfiguration, Grid Computing.

一、緣由與目的

隨著網路技術的進步與頻寬快速成長，加上經濟成本與效益的考量，將網路上的計算資源結合成為一個具有工作協調能力的計算系統蘊育而生，網格計算(Grid Computing)也因此成為大量資料與科學計算在平行與分散式計算平台以外的另一種選擇。從技術面來說，網格計算環境可以結合平行電腦、工作站叢集、以及網路上任意可用的計算資源，進而加大其運算能力。有鑑於此，在異質性(Heterogeneous)的計算環境上開發輔助運算的軟體工具也成為近幾年來廣為討論的課題。爲了結合既有的平行程式技術與異質性的網格計算平台，在分散式網路計算環境上從事大量平行計算所延伸的相關問題就成了相當值得研究的課題。由於 SPMD 程式模式是在平行計算系統中所發展出來的程式方法，如何可以有效率的移植 SPMD 程式透通於異質性的計算平台上以保有其程式演算的最佳效能自然成了最直接的挑戰。爲了結合既有的平行程式技術與異質性的網格計算平台，在分散式網路計算環境上從事大量平行計算所延伸的相關問題就成了相當值得研究的課題。由於 SPMD 程式模式是在平行計算系統中所發展出來的程式方法，如何可以有效率的移植 SPMD 程式透通於異質性的計算平台上以保有其程式演算的最佳效能自然成了最直接的挑戰。這些問題討論的重點可以從系統與應用程式的管理以及計算平台的架構兩點來研究。

在系統與應用程式的管理方面，工作分配的好壞直接影響了程式的完成時間與系統資源的使用。工作負載平衡(workload balance)則可避免某一系統因工作負擔太重，而拉長整個工作結束的時間，以達到高效能計算的目標。另一方面，在分散式記憶體計算環境下，要有效率的執行一個平行資料的程式，適當的資料配置(Data Distribution)是很重要的。由於資料的區域性(locality)可減少處理器間資料的傳輸，所以在支援 SPMD 程式資料執行於網格計算系統方面，我們將研究在異質性的分散式記憶體群體計算系統中，工作如何有效的分配計算工作至各處理器上，可使得各處理器的工作量是均衡的；我們也將研究有效率的方法來處理的資料分配與資源重組的問題，進而提高資料的區域性；在程式執行期間，減少處理器之間的資料交換，降低通訊成本。

在這個計畫中，我們主要是要研發適應於異質性分散式網路計算環境之工作排程與管理、資源分配以及動態資料重組技術的整合方法，用來提升 SPMD 平行運算程式的效能。主要的工作項目包括研究工作排程、工作負載平衡與工作重新配置對整體程式執行的必要性及其在效能上影響；研究在不同網路領域或網路拓僕環境之間通訊對 SPMD 程式執行動態資料交換所造成的影響及其最佳化；以及發展以網頁為基礎的異

質性網路計算環境工作排程與監督系統。

二、研究方法與成果

對於異質性分散式網路計算環境下工作配置與資源管理的問題，我們首先針對軟體的異質性(包括作業系統、訊息傳遞介面、區域排程策略)，解決系統之間身分確認(authentication)與資源授權(authorization) 的問題。爲了在分散的異質性環境下執行相同的工作，我們在不同的節點上同步配置需要的資源，並且建立一個虛擬的共同執行環境 (MPI 的實作上,也可以利用 MPI_COMM_WORLD 作為程式執行過程中的通訊領域)達到工作協調配置(co-allocation)的目的。在開發工作協調配置的方法上，利用包裝在通訊模組與工作配置模組的工具來達成此目標。主要的好處是可以降低開發資源管理模組的複雜度。我們亦採用 GRAM (Globus Resource Allocation Management) 為工具，透過單一的介面來管理區域資源(包括電腦與網路的服務)，解決資源管理的問題並提高未來系統的擴充性。GRAM 除了可以提供上述安全認證的管理機制，另外還支援複雜的資源協調配置(co-allocation)與錯誤偵測(failure detection)。

由於在異質性的網格計算環境中，工作配置、排程以及計算過程中處理必要的資源重新分配問題都屬於資源協調配置的問題，而這些動態管理的機制其目的在於維持系統的可靠性以及提高程式執行的效能。我們將 SPMD 程式在異質性的計算環境上執行平行計算結合通訊介面技術一併考慮。

資料在不同計算系統平台之間的通訊，我們仍然採用以 MPI 為基礎的通訊介面標準：MPICH-G 的實作(利用 MPI 的通訊介面，將有助於本項計劃所開發的程式，在不同分散式記憶體平台之間的可攜性)，亦有助於我們在不同的分散式平台之間執行程式。相較於其它類似網格計算環境所提供之工具，MPICH-G 提供簡單的單一介面來起始程式的執行。此外，不論在 SMP 或 MPP 之間配置工作的執行，它都使用相同的語法。在不同的網路領域，我們也嘗試使用 Nexus 通訊函式庫所提供之多種不同的通訊機制。針對如何有效的複製 SPMD 程式在異質的分散式系統中選定的電腦，MPICH-G 的實作也克服了硬體與記憶體儲存的困難。我們利用 GASS(Global Access Secondary Storage) 工具，將所要執行的 SPMD 程式複製到每一個遠端的機器上。這裡值得注意的一點是，所有 SPMD 程式必須先由程式人員編譯完成，才能將執行檔散佈到參與執行的節點機器。另外，根據 MPICH-G 的實作，我們可以採用動態需求更新的線上配置方法，送出需求、確認正確的起始時間，進而針對不同的程序提供函式建立虛擬的共同執行環境 (利用 MPI_COMM_WORLD 作為程式執行過程中的通訊領域)。

在多個 MPP 系統上配置計算工作是比較複雜的環節。工作管理上面臨的問題，我們考量的方法討論如下。首先將資源配置給予參與執行的電腦，接著將起始處理程序的執行，最後將所有的處理程序連結為一個大型的計算。由於不同電腦的資源配置與處理程序的建立策略有所差異，因此我們在每一個計算節點之間協調出一個合適的方式。另外，要起始一個處理程序，可能會花上很大的時間和出現不可預期的錯誤。所以我們引用 GRAM 的介面與其函式庫進行錯誤偵測(failure detection)的機制(可以利用 timeout 的方式來決定)，當完成處理程序的起始後隨即實施同步 (synchronizing)。用 GRAM 單一介面來執行區域的排程並且支援工作的協調配置(co-allocation)，可以有效的收集各個系統資源的資訊。同時，我們也利用 LDAP (Lightweight Directory Access Protocol) 的方式，提供資源配置模組最新的系統資訊與狀態。

在異質性平行計算網路下工作分享、負載平衡與重新配置的問題上，我們提出了一套以基因演算法為基礎的模糊理論，提升異質性分散式計算平台的排程效率。另外，在提升 SPMD 程式執行於不同網路領域或網路拓僕環境的效能方面，我們也針對 SPMD 平行資料程式在異質性多叢集系統上提出有效率的通訊技術。透過邏輯處理器與資料對應的技術，可以降低不同叢集系統之間處理器的通訊成本。在資源配置與工作管理方面，我們分別針對處理器計算能力的異質與網路頻寬的異質，提出 SCTF (Shortest Communication Time First)工作排程演算法，並且開發以網頁為基礎的管理工具，在兩套 PC_Cluster 系統上實作出資源配置、監督與工作排程系統。對於工作排程、協調配置與資源管理的問題，我們測試了幾套國外著名的工作排程系統，進行不同平台的測試，實驗結果顯示，SCTF 提升了系統平均產能(throughput)、縮短了工作執行的平均回覆時間(turnaround time)。本計畫的成果有助於增加異質性叢集系統的產能以及 SPMD 平行程式在該系統的執行效率。

三、結果與討論

下面我們歸納本計畫主要的成果:

- 我們在現有的兩套 PC Cluster 架構之下，建置一套異質性的分散式計算平台。根據處理器計算能力與網路傳輸速度的不同(異質)，我們提出一套效能評估模組，用來預測程式的執行效能。
- 本計畫另一個成果是研究計算網格上平行程式通訊最佳化的資料分割技術。利用邏輯處理器與資料對應的方法，可以有效的降低處理器之間資料通訊的時間。我們所提出來的方法可以適應於同質、異質的計算系統。此一技術已於 2005 年歐洲網格

會議中發表，在會場中引起多位學者的興趣與討論。

- 此外，針對異質性網路計算環境下的工作負載平衡，我們提出一套以基因演算法為基礎之模糊理論應用在異質性分散式系統。我們也針對工作排程、協調配置與資源管理的問題，提出 SCTF (Shortest Communication Time First)工作排程演算法、此演算法不論在處理器異質或網路異質的系統都可以很容易的實作出來，實驗的結果也顯示 SCTF 可以有比較好的系統產能與平均工作回覆時間。相關的研究工作亦包括我們將資料重組的技術移植到 SPMD 程式模式下的工作重新(配置)排程。
- 執行本計畫所發表之相關論文列舉如下

1. Ching-Hsien Hsu and Min-Hao Chen, "Communication Free Dynamic Data Redistribution of Symmetrical Matrices on Distributed Memory Machines," Accepted, *IEEE Transactions on Parallel and Distributed Systems* (**SCI**, **EI**, NSC93-2213-E-216-028) //對稱矩陣上動態資料重組技術
2. Ching-Hsien Hsu, Shih-Chang Chen and Chao-Yang Lan, "Scheduling Contention-Free Irregular Redistribution in Parallelizing Compilers," Accepted, The *Journal of Supercomputing*, Kluwer Academic Publisher. (**SCI**, **EI**, NSC93-2213-E-216-028, NCHC-KING-010200) // 異質性系統之通訊排程技術
3. Ching-Hsien Hsu, Shih-Chang Chen and Tzu-Tai Lo, "Locality Preserving Data Partitioning for SPMD Programs on Computational Grid," *Chung Hua Journal of Science and Engineering*, Vol. 3, No. 1, pp. 121-128, January 2005. (NSC92-2213-E-216-028) // SPMD 程式資料區域化技術
4. Ching-Hsien Hsu and Tai-Long Chen, "Grid Enabled Master Slave Task Scheduling for Heterogeneous Processor Paradigm," *Grid and Cooperative Computing - Lecture Notes in Computer Science*, Vol. 3795, pp. 449-454, Springer-Verlag, Dec. 2005. (GCC'05) (SCI Expanded, NSC92-2213-E-216-028) //異質性系統之工作排程技術
5. Ching-Hsien Hsu, Shih-Chang Chen, Chao-Yang Lan, Chao-Tung Yang and Kuan-Ching Li, "Scheduling Convex Bipartite Communications Towards Efficient GEN_BLOCK Transformations," *Parallel and Distributed Processing and Applications - Lecture Notes in Computer Science*, Vol. 3758, pp. 419-424, Springer-Verlag, Nov. 2005. (ISPA'05) (SCI Expanded, NSC92-2213-E-216-028) // 異質性系統之通訊排程技術
6. Ching-Hsien Hsu, Guan-Hao Lin, Kuan-Ching Li and Chao-Tung Yang, "Localization Techniques for Cluster-Based Data Grid," Algorithm and Architecture for Parallel Processing - *Lecture Notes in Computer Science*, Vol. 3719, pp. 83-92, Springer-Verlag, Oct. 2005. (ICA3PP'05) (SCI Expanded, NSC 93-2213-E-216-028) // 叢集式資料網格系統之資料局部化技術
7. Kun-Ming Yu, Ching-Hsien Hsu and Chwani-Lii Sune, "A Genetic-Fuzzy Logic Based Load Balancing Algorithm in Heterogeneous Distributed Systems," Proceedings of the IASTED *International Conference on Neural Network and Computational Intelligence* (NCI 2004), Feb. 2004, Grindelwald, Switzerland. // 異質性系統之工作負載平衡技術

四、計劃成果自評

本計劃之研究成果，達到預期之目標，其中之成果一，以基因演算法為基礎之模糊理論應用在異質性分散式系統工作負載平衡與排程已經發表於 2004 年 *International Conference on Neural Network and Computational Intelligence* 會議。而另一個成果，平行資料程式於多叢集式格網系統中通訊最佳化也在 2005 年歐洲網格會議中發表。最後，我們也在不規則的通訊排程問題的技術上有所突破，日前已被 The Journal of Supercomputing 接受，預計在 2006-2007 年發表。本計畫相關成果的整理與擴充將陸續投稿至國外的會議以及國際期刊。本計畫有堪稱不錯的研究成果，感謝國科會給予機會。下一個計畫年度，我們將更加努力，爭取經費建立更完備的研究環境。另外，對於參與研究計畫執行同學的認真，本人亦表達肯定與感謝。

五、參考文獻

1. D. Angulo, I. Foster, C. Liu and L. Yang, “Design and Evaluation of a Resource Selection Framework for Grid Applications,” *Proceedings of IEEE International Symposium on High Performance Distributed Computing (HPDC-11),* Edinburgh, Scotland, July 2002.
2. K. Czajkowski, I. Foster and C. Kesselman, “Resource Co-Allocation in Computational Grids,” *Proceedings of the Eighth IEEE International Symposium on High Performance Distributed Computing (HPDC-8),* pp. 219-228, 1999.
3. C. Lee, R. Wolski, I. Foster, C. Kesselman and J. Stepanek, “A Network Performance Tool for Grid Computations,” *Supercomputing '99*, 1999.
4. I. Foster, C. Kesselman, C. Lee, R. Lindell, K. Nahrstedt and A. Roy, “A Distributed Resource Management Architecture that Supports Advance Reservations and Co-Allocation,” *Intl Workshop on Quality of Service*, 1999.
5. K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith and S. Tuecke, “A Resource Management Architecture for Metacomputing Systems,” *Proc. IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing*, pg. 62-82, 1998.
6. B. Allcock, J. Bester, J. Bresnahan, A. L. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnal and S. Tuecke, “Data Management and Transfer in High Performance Computational Grid Environments,” *Parallel Computing Journal*, Vol. 28 (5), May 2002, pp. 749-771.

7. H. Stockinger, A. Samar, B. Allcock, I. Foster, K. Holtman and B. Tierney, “File and Object Replication in Data Grids,” *Journal of Cluster Computing*, 5(3)305-314, 2002.
8. S. Vazhkudai and J. Schopf, “Using Disk Throughput Data in Predictions of End-to-End Grid Transfers,” *Proceedings of the 3rd International Workshop on Grid Computing (GRID 2002)*, Baltimore, MD, November 2002.
9. J.M. Schopf and S. Vazhkudai, “Predicting Sporadic Grid Data Transfers,” *11th IEEE International Symposium on High-Performance Distributed Computing (HPDC-11)*, IEEE Press, Edinburg, Scotland, July 2002.
10. M. Colajanni and P.S. Yu, “A performance study of robust load sharing strategies for distributed heterogeneous Web servers,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 14, no. 2, pp. 398-414, 2002.
11. Kun-Ming Yu, Ching-Hsien Hsu and Chwani-Lii Sune, "A Genetic-Fuzzy Logic Based Load Balancing Algorithm in Heterogeneous Distributed Systems," Proceedings of the IASTED *International Conference on Neural Network and Computational Intelligence* (NCI 2004), Feb. 2004, Grindelwald, Switzerland.
12. Ching-Hsien Hsu, Tzu-Tai Lo and Shih-Chang Chen, "Optimizing Communications of Data Parallel Programs on Cluster Grid," Proceedings of the 1st Workshop on Grid Technology and Applications, Dec. 2004, NCHC, Hsinchu, Taiwan.

行政院所屬各機關人員出國報告書提要

撰寫時間： 94 年 3 月 1 日

姓　　名	許慶賢	服務機關名稱	中華大學資工系	連絡電話、電子信箱	03-5186410 chh@chu.edu.tw
出生日期	62 年 2 月 23 日		職　　稱	助理教授	
出席國際會議名稱	European Grid Conference, February 14 -16 2005,				
到達國家及地點	**Science Park Amsterdam, The Netherlands**		出國期間	自 94 年 02 月 12 日 迄 94 年 02 月 18 日	
內容提要	這一次在荷蘭所舉行的國際學術研討會議共計三天。第一天上午由 **Domenico Laforenza** 博士針對 Towards a Next Generation Grid: Learning from the past, Looking into the future 主題發表精闢的演說作為研討會的開始。同時當天也有許多重要的研究成果分為兩個平行的場次進行論文發表。本人選擇了 Architecture and Infrastructure 場次聽取報告。第一晚上本人亦參加酒會，並且與幾位國外學者及中國、香港教授交換意見。第二天本人在上午聽取了 Data and Information Management 相關研究，同時獲悉許多新興起的研究主題，並了解目前國外大多數學者主要的研究方向。當天下午發表我們的論文，本人亦參與大會所舉辦的晚宴。並且與幾位外國學者認識，交流，合影留念。會議最後一天，本人選擇與這一次論文較為相近的 Scheduling, Fault Tolerance and Mapping 以及分散式計算研究聽取論文發表，並且把握最後一天的機會與國外的教授認識，希望能夠讓他們加深對台灣研究的印象。三天下來，本人聽了許多優秀的論文發表。這些研究所涵蓋的主題包含有：網格系統技術、工作排程、網格計算、網格資料庫以及無線網路等等熱門的研究課題。此次的國際學術研討會議有許多知名學者的參與，讓每一位參加這個會議的人士都能夠得到國際上最新的技術與資訊。是一次非常成功的學術研討會。參加本次的國際學術研討會議，感受良多。讓本人見識到許多國際知名的研究學者以及專業人才，得以與之交流。讓本人與其他教授面對面暢談所學領域的種種問題。看了眾多研究成果以及聽了數篇專題演講，最後，本人認為，會議所安排的會場以及邀請的講席等，都相當的不錯，覺得會議舉辦得很成功，值得我們學習。				
出席人所屬機關審核意見					
層轉機關審核意見					
研考會處理意見					

Localized Communications of Data Parallel Programs on Multi-Cluster Grid Systems[1]

Ching-Hsien Hsu, Tzu-Tai Lo and Kun-Ming Yu

Department of Computer Science and Information Engineering

Chung Hua University, Hsinchu, Taiwan 300, R.O.C.

chh@chu.edu.tw

Abstract. The advent of widely interconnected computing resources introduces the technologies of grid computing. A typical grid system, the cluster grid, consists of several clusters located in multiple campuses that distributed globally over the Internet. Because of the Internet infrastructure of cluster grid, the communication overhead becomes as key factor to the performance of applications on cluster grid. In this paper, we present a processor reordering technique for the communication optimizations of data parallel programs on cluster grid. The alignment of data in parallel programs is considered as example to examine the proposed techniques. Effectiveness of the processor reordering technique is to reduce the inter-cluster communication overheads and to speedup the execution of parallel applications in the underlying distributed clusters. Our preliminary analysis and experimental results of the proposed method on mapping data to logical grid nodes show improvement of communication costs and conduce to better performance of parallel programs on different hierarchical grid of cluster systems.

1. Introduction

One of the virtues of high performance computing is to integrate massive computing resources for accomplishing large-scaled computation problems. The common point of these problems has enormous data to be processed. Due to cost-effective, clusters have been employed as a platform for high-performance and high-availability computing platform. In recent years, as the growth of Internet technologies, the grid computing emerging as a widely accepted paradigm for next-generation applications, such as data parallel problems in supercomputing, web-serving, commercial applications and grand challenge problems.

Differing from the traditional parallel computers, a grid system [7] integrates distributed computing resources to establish a virtual and high expandable parallel platform. Figure 1 shows the typical architecture of cluster grid. Each cluster is geographically located in different campuses and connected by software of computational grids through the Internet. In cluster grid, communications occurred when grid nodes exchange data with others via network to run job completion. These communications are usually classified into two types, local and remote. If the two grid nodes belong to different clusters, the messaging should be accomplished through the Internet. We refer this kind of data transmission as *external communication*. If the two grid nodes in the same space domain, the communications take place within a cluster; we refer this kind of data transmission as *interior communication*. Intuitionally, the external communication is usually with higher communication latency than that of the interior communication since the data should be routed through numbers of layer-3 routers or higher-level network devices over the Internet. Therefore, to efficiently execute parallel applications on cluster grid, it is extremely critical to avoid large amount of external communications.

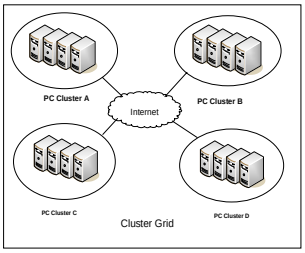


Figure 1: The paradigm of cluster grid.

In this paper, we consider the issue of minimizing external communications of data parallel program on cluster grid. We first employ the example of data alignments and realignments that provided in many data parallel-programming languages to examine the effective of the proposed data to logical processor mapping

[1]The work of this paper was supported NSC, National Science Council of Taiwan, under grant number NSC-93-2213-E-216-028.

scheme. As researches discovered that many parallel applications require different access patterns to meet parallelism and data locality during program execution. This will involve a series of data transfers such as array redistribution. For example, a 2D-FFT pipeline involves communicating images with the same distribution repeatedly from one task to another. Consequently, the computing nodes might decompose local data set into sub-blocks uniformly and remapped these data blocks to designate processor group. From this phenomenon, we propose a processor-reordering scheme to reduce the volume of external communications of data parallel programs in cluster grid. The key idea is that of distributing data to grid/cluster nodes according to a mapping function at data distribution phase initially instead of in numerical-ascending order. We also evaluate the impact of the proposed techniques. The theoretical analysis and experimental results show improvement of volume of interior communications and conduce to better performance of data alignment in different hierarchical cluster grids.

The rest of this paper is organized as follows. Section 2 briefly surveys the related works. In section 3, we formulate the communication model of parallel data partitioning and re-alignment on cluster grid. Section 4 describes the processor-reordering scheme for communication localization. Section 5 reports the performance analysis and experimental results. Finally, we conclude our paper in section 6.

2. Related Work

Clusters have been widely used for solving grand challenge applications due to their good price-performance nature. With the growth of Internet technologies, the computational grids [4] become newly accepted paradigm for solving these applications. As the number of clusters increases within an enterprise and globally, there is the need for a software architecture that can integrate these resources into larger grid of clusters. Therefore, the goal of effectively utilizing the power of geographically distributed computing resources has been the subject of many research projects like Globus [6, 8] and Condor [9]. Frey *et al.* [9] also presented an agent-based resource management system that allowed users to control global resources. The system is combined with Condor and Globus, gave powerful job management capabilities is called Condor-G.

Recent work on computational grid has been broadly discussed on different aspects, such as security, fault tolerance, resource management [9, 2], job scheduling [17, 18, 19], and communication optimizations [20, 5, 16, 3]. For communication optimizations, Dawson *et al.* [5] and Zhu *et al.* [20] addressed the problems of optimizations of user-level communication patterns in local space domain for cluster-based parallel computing. Plaat *et al.* analyzed the behavior of different applications on wide-area multi-clusters [16, 3]. Similar researches were studied in the past years over traditional supercomputing architectures [12, 13]. For example, Guo *et al.* [11] eliminated node contention in communication phases and reduced communication steps with schedule table. Y. W. Lim *et al.* [15] presented an efficient algorithm for block-cyclic data realignments. Kalns and Ni [14] proposed the processor mapping technique to minimize the volume of communication data for runtime data re-alignments. Namely, the mapping technique minimizes the size of data that need to be transmitted between two algorithm phases. Lee *et al.* [10] proposed similar algorithms, the processor reordering, to reduce data communication cost. They also compared their effects upon various conditions of communication patterns.

The above researches give significant improvement of parallel applications on distributed memory multi-computers. However, most techniques only applicable for parallel programs running on local space domain, like single cluster or parallel machine. For a global grid of clusters, these techniques become inapplicable due to various factors of Internet hierarchical and its communication latency. In this paper, our emphasis is on dealing with the optimizations of communications for data parallel programs on cluster grid.

3. Preliminaries

3.1 Problem Formulation

The data parallel programming model has become a widely accepted paradigm for parallel programming on distributed memory multi-computers. To efficiently execute a parallel program, appropriate data distribution is critical for balancing the computational load. A typical function to decompose the data equally can be accomplished via the `BLOCK` distribution directive.

It has been shown that the data reference patterns of some parallel applications might be changed dynamically. As they evolve, a good mapping of data to logical processors must change adaptively in order to ensure good data locality and reduce inter-processor communication. For example, a global array could be equally allocated to a set of processors initially in `BLOCK` distribution manner. As the algorithm goes into another phase that requires to access fine-grain data patterns, each processor might divide its local data into

sub-blocks locally and then distribute these sub-blocks to corresponding destination processors. Figure 2 shows an example of this scenario. In the initial distribution, the global array is evenly decomposed into nine data sets and distributed over processors that are selected from three clusters. In the target distribution, each node divides its local data into three sub-blocks evenly and distributes them to the same processor set in grid as in the initial distribution. Since these data blocks might be needed and located in different processors, consequently, efficient inter-processor communications become major subject to the performance of these applications.

Initial Distribution

Cluster-1			*Cluster-2*			*Cluster-3*		
P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
A	*B*	*C*	*D*	*E*	*F*	*G*	*H*	*I*

Target Distribution

Cluster1			*Cluster2*			*Cluster3*			*Cluster1*			*Cluster2*			*Cluster3*			*Cluster1*			*Cluster2*			*Cluster3*		
P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
a_1	a_2	a_3	b_1	b_2	b_3	c_1	c_2	c_3	d_1	d_2	d_3	e_1	e_2	e_3	f_1	f_2	f_3	g_1	g_2	g_3	h_1	h_2	h_3	i_1	i_2	i_3

Figure 2: Data distributions over cluster grid.

To facilitate the presentation of the proposed approach, we assume that a global array is distributed over processors in `BLOCK` manner at the initiation. Each node is requested to partition its local block into K equally sub-blocks and distribute them over processors in the same way. The second assumption is that each cluster provides the same number of computers involved in the computation.

Definition 1: The above term K is defined as *partition factor*.

For instance, the partition factor of the example in Figure 2 is K=3. (Block A is divided into a_1, a_2, a_3, B is divided into b_1, b_2, b_3, etc.)

Definition 2: Given a cluster grid, C denotes the number of clusters in the grid; n_i is the number of processors selected from cluster i, where $1 \le i \le C$; P is the total number of processors in the cluster grid.

According to definition 2, we have $P = \sum_{i=1}^{C} n_i$. Figure 2 has three clusters, thus C = 3, where $\{P_0, P_1, P_2\} \in$ *Cluster* 1, $\{P_3, P_4, P_5\} \in$ *Cluster* 2 and $\{P_6, P_7, P_8\} \in$ *Cluster* 3, we also have $n_1 = n_2 = n_3 = 3$ and $P = 9$.

3.2 Communication Cost Model

Because the interface of interconnect switching networks in each cluster system might be different; to obtain accurate evaluation, the interior communication costs in clusters should be identified individually. We let T_i represents the time of two processors both reside in *Cluster-i* to transmit per unit data; m_i is the sum of volume of all interior messages in *Cluster-i*; for an external communication between cluster i and cluster j, T_{ij} is used to represent the time of processor p in cluster i and processor q in cluster j to transmit per unit data; similarly, m_{ij} is the sum of volume of all external messages between cluster i and cluster j. According to these declarations, we can have the following cost function,

$$T_{comm} = \sum_{i=1}^{C} T_i \times m_i + \sum_{i,j=1,i\neq j}^{C} (m_{ij} \times T_{ij}) \tag{1}$$

Due to various factors over Internet might cause communication delay; it is difficult to get accurate costs from the above function. As the need of a criterion for performance modeling, integrating the interior and external communications among all clusters into points is an alternative mechanism to get legitimate evaluation. Thus, we totted up the number of these two terms to represent the communication costs through the whole running phase for the following discussions. The volume of interior communications, denoted as $|I|$ and external communications, denoted as $|E|$ are defined as follows,

$$|I| = \sum_{i=1}^{C} I_i \tag{2}$$

$$|E| = \sum_{i,j=1,i\neq j}^{C} E_{ij} \tag{3}$$

Where I_i is the total number of interior communications within cluster i; E_{ij} is the total number of external communications between cluster i and cluster j.

4. Communication Localization

4.1 Motivating Example

Let us consider the example in Figure 2. In the target distribution, processor P_0 divides data block A into a_1, a_2, and a_3. Then, it distributes these three sub-blocks to processors P_0, P_1 and P_2, respectively. Since processors P_0, P_1 and P_2 belong to the same cluster with P_0; therefore, these are three interior communications. Similar situation on processor P_1 will generate three external communications; P_1 divides its local data block B into b_1, b_2, and b_3. It distributes these three sub-blocks to P_3, P_4 and P_5, respectively. However, as processor P_1 belongs to *Cluster* 1 while processors P_3, P_4 and P_5, belong to *Cluster* 2. Thus, this results three external communications. Figure 3 summarizes all messaging patterns of the example into a communication table. The messages $\{a_1, a_2, a_3\}$, $\{e_1, e_2, e_3\}$ and $\{i_1, i_2, i_3\}$ are interior communications (the shadow blocks). All the others are external communications. Therefore, we have $| I | = 9$ and $| E | = 18$.

DP / SP	P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
P_0	a_1	a_2	a_3						
P_1				b_1	b_2	b_3			
P_2							c_1	c_2	c_3
P_3	d_1	d_2	d_3						
P_4				e_1	e_2	e_3			
P_5							f_1	f_2	f_3
P_6	g_1	g_2	g_3						
P_7				h_1	h_2	h_3			
P_8							i_1	i_2	i_3
	Cluster-1			Cluster-2			Cluster-3		

Figure 3: Communication table of data distribution over cluster grid.

Figure 4 illustrates a bipartite representation to show the communications that given in the above table. In this graph, the dashed arrows and solid arrows indicate interior and external communications, respectively. Each arrow contains three communication links.

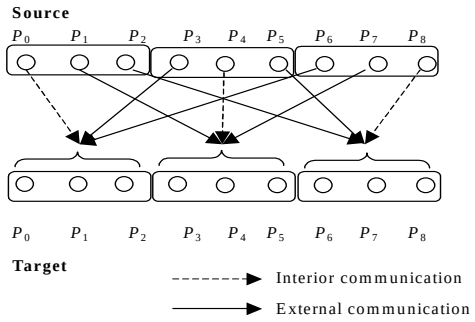


Figure 4: Interior and external communications using bipartite representation.

4.2 Processor Reordering Data Partitioning

The processor mapping techniques were used in several previous researches to minimize data transmission time of runtime array redistribution. In a cluster grid system, the similar concept can be applied. According to assumptions in section 3.1, we proposed the processor reordering technique and its mapping function that is applicable to data realignment on cluster grid. In order to localize the communication, the mapping function produces a reordered sequence of processors for grouping communications into local cluster. A reordering agent is used to accomplish this process. Figure 5 shows the concept of processor reordering technique for parallel data to logical processor mapping. The source data is partitioned and distributed to processors into initial distributions ($ID(P_X)$) according to the processor sequence derived from reordering agent, where X is the processor id and $0 \le X \le P$-1. To accomplish the target distribution ($TD(P_{X'})$), the initial data is divided into K sub-blocks and realign with processors according to the new processors id X' that is also derived from the reordering agent. Given distribution factor K and processor grid (with variables C and n_i), for the case of $K=n_i$, the mapping function used in reordering agent is formulated as follows,

$$F(X) = X' = \lfloor X / C \rfloor + (X \bmod C) * K \qquad (4)$$

We use the same example to demonstrate the above reordering scheme. Figure 6 shows the communication table of messages using new logical processor sequence. The initial distribution of source data is allocated by the sequence of processors' id, $<P_0, P_3, P_6, P_1, P_4, P_7, P_2, P_5, P_8>$ which is derived from equation 4. To accomplish the target distribution, P_0 divides data block A into a_1, a_2, a_3 and distributes them

to P_0, P_1 and P_2, respectively. These communications are interior. For P_3, the division of initial data also generates three interior communications; because P_3 divides its local data B into b_1, b_2, b_3 and distributes these three sub-blocks to P_3, P_4 and P_5, respectively; which are in the same cluster with P_3. Similarly, P_6 sends e_1, e_2 and e_3 to processors P_6, P_7 and P_8 and causes three interior communications. Eventually, there is no external communication incurred in this example in Figure 6.

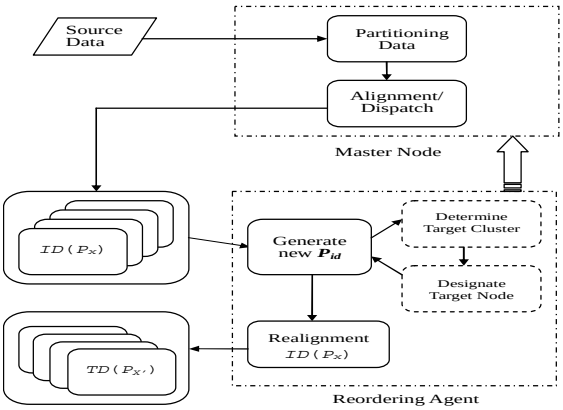


Figure 5: The flow of data to logical processor mapping.

DP / SP	P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
P_0	a_1	a_2	a_3						
P_3				b_1	b_2	b_3			
P_6							c_1	c_2	c_3
P_1	d_1	d_2	d_3						
P_4				e_1	e_2	e_3			
P_7							f_1	f_2	f_3
P_2	g_1	g_2	g_3						
P_5				h_1	h_2	h_3			
P_8							i_1	i_2	i_3
	Cluster-1			Cluster-2			Cluster-3		

Figure 6: Communication table with processor reordering.

The bipartite representation of Figure 6's communication table is shown in Figure 7. All the communication arrows are in dashed lines. We totted up the communications, then have $|I| = 27$ and $|E| = 0$. The external communications are mostly eliminated.

5. Performance Analysis and Experimental Results

5.1 Performance Analysis

The effectiveness of processor reordering technique in different hierarchy of cluster grid can be evaluated in theoretical. This section presents the improvements of volume of interior communications for different number of clusters (C) and partition factors (K).

For the case consists of three clusters (C=3), Figure 8(a) shows that the processor reordering technique provides more interior communications than the method without processor reordering. For the case consists of four clusters (C=4), the values of K vary from 4 to 10. The processor reordering technique also provides more interior communications as shown in Figure 8(b). Note that Figures 8 and 9 report the theoretical results which will not be affected by the Internet traffic. In other words, Figure 8 is our theoretical predictions.

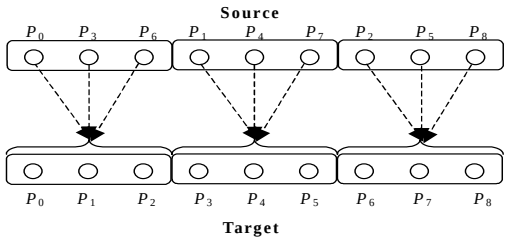


Figure 7: Bipartite representation with processor reordering.

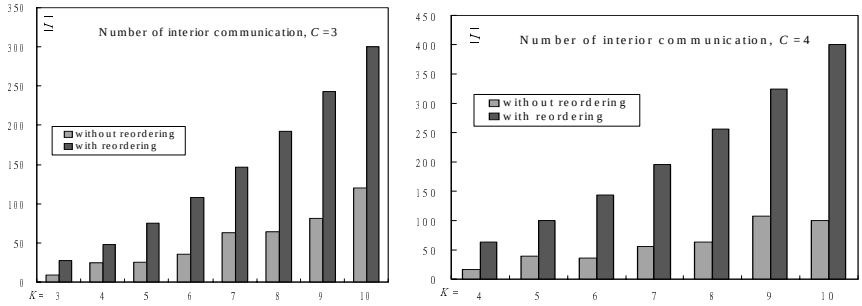


Figure 8: The number of interior communications (a) C=3 (b) C=4.

5.2 Simulation settings and Experimental Results

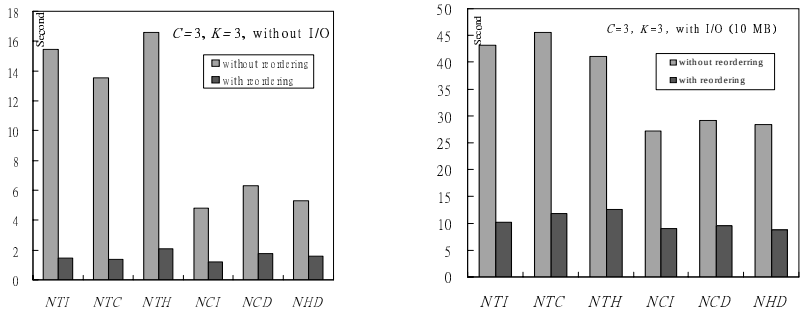
To evaluate the performance of the proposed technique, we have implemented the processor reordering method and tested on *Taiwan UniGrid* in which 8 campus clusters ware interconnected via Internet. Each cluster owns different number of computing nodes. The programs were written in the single program multiple data (*SPMD*) programming paradigm with C+MPI codes.

Figure 9 shows the execution time of the methods with and without processor reordering to perform data realignment when C=3 and K=3. Figure 9(a) gives the result of 1MB test data that without file system access (I/O). The result for 10MB test data that accessed via file system (I/O) is given in Figure 9(b). Different combinations of clusters denoted as *NTI*, *NTC*, *NTD*, etc. were tested. The composition of these labels is summarized in Table 1.

Table 1: Labels of different cluster grid

Label	*Cluster*-1	*Cluster*-2	*Cluster*-3	**Label**	*Cluster*-1	*Cluster*-2	*Cluster*-3
NTI	***N***CHC	***NTH***U	***I***IS	***NCI***	***N***CHC	***C***HU	***I***IS
NTC	***N***CHC	***NTH***U	***C***HU	***NCD***	***N***CHC	***C***HU	***ND***HU
NTH	***N***CHC	***NTH***U	***TH***U	***NHD***	***N***CHC	***TH***U	***ND***HU

In Figure 9(a), we observe that processor reordering technique outperforms the traditional method. In this experiment, our attention is on the presented efficiency of the processor reordering technique instead of on the execution time in different clusters. Compare to the results given in Figure 8, this experiment matches the theoretical predictions. It also satisfying reflects the efficiency of the processor reordering technique. Figure 9(b) presents the results with larger test data (10 MB) under the same cluster grid. Each node is requested to perform the data realignments through access file system (I/O). The improvement rates are lower than that in Figure 9(a). This is because both methods spend part of time to perform I/O; the ratio of communication cost becomes lower. Nonetheless, the reordering technique still presents considerable improvement.



(a) (b)

Figure 9: Execution time of data realignments on cluster grid when C = K = 3.

6. Conclusions and Future Works

In this paper, we have presented a processor reordering technique for localizing the communications of data parallel programs on cluster grid. Our preliminary analysis and experimental results of re-mapping data to

logical grid nodes show improvement of volume of interior communications. The proposed techniques conduce to better performance of data parallel programs on different hierarchical grid of clusters systems. There are numbers of research issues remained in this paper. The current work of our study restricts conditions in solving the realignment problem. In the future, we intend to devote generalized mapping mechanisms for parallel data partitioning. We will also study realistic applications and analyze their performance on the UniGrid. Besides, the issues of larger grid system and analysis of network communication latency are also interesting and will be investigated.

References

[1] Taiwan UniGrid, http://unigrid.nchc.org.tw

[2] O. Beaumont, A. Legrand and Y. Robert, "Optimal algorithms for scheduling divisible workloads on heterogeneous systems," *Proceedings of the 12th IEEE Heterogeneous Computing Workshop,* 2003.

[3] Henri E. Bal, Aske Plaat, Mirjam G. Bakker, Peter Dozy, and Rutger F.H. Hofman, "Optimizing Parallel Applications for Wide-Area Clusters," *Proceedings of the 12th International Parallel Processing Symposium IPPS'98*, pp 784-790, 1998.

[4] J. Blythe, E. Deelman, Y. Gil, C. Kesselman, A. Agarwal, G. Mehta and K. Vahi, "The role of planning in grid computing," *Proceedings of ICAPS'03*, 2003.

[5] J. Dawson and P. Strazdins, "Optimizing User-Level Communication Patterns on the Fujitsu AP3000," *Proceedings of the 1st IEEE International Workshop on Cluster Computing*, pp. 105-111, 1999.

[6] I. Foster, "Building an open Grid," *Proceedings of the second IEEE international symposium on Network Computing and Applications*, 2003.

[7] I. Foster and C. K., "The Grid: Blueprint for a New Computing Infrastructure," Morgan Kaufmann, ISBN 1-55860-475-8, 1999.

[8] I. Foster and C. Kessclman, "Globus: A metacomputing infrastructure toolkit," *Intl. J. Supercomputer Applications*, vol. 11, no. 2, pp. 115-128, 1997.

[9] James Frey, Todd Tannenbaum, M. Livny, I. Foster and S. Tuccke, "Condor-G: A Computation Management Agent for Multi-Institutional Grids," *Journal of Cluster Computing*, vol. 5, pp. 237 – 246, 2002.

[10] Saeri Lee, Hyun-Gyoo Yook, Mi-Soon Koo and Myong-Soon Park, "Processor reordering algorithms toward efficient GEN_BLOCK redistribution," *Proceedings of the 2001 ACM symposium on Applied computing*, 2001.

[11] M. Guo and I. Nakata, "A Framework for Efficient Data Redistribution on Distributed Memory Multicomputers," *The Journal of Supercomputing*, vol.20, no.3, pp. 243-265, 2001.

[12] Florin Isaila and Walter F. Tichy, "Mapping Functions and Data Redistribution for Parallel Files," *Proceedings of IPDPS 2002 Workshop on Parallel and Distributed Scientific and Engineering Computing with Applications, Fort Lauderdale*, April 2002.

[13] Jens Koonp and Eduard Mehofer, "Distribution assignment placement: Effective optimization of redistribution costs," *IEEE TPDS*, vol. 13, no. 6, June 2002.

[14] E. T. Kalns and L. M. Ni, "Processor mapping techniques toward efficient data redistribution," *IEEE TPDS*, vol. 6, no. 12, pp. 1234-1247, 1995.

[15] Y. W. Lim, P. B. Bhat and V. K. Parsanna, "Efficient algorithm for block-cyclic redistribution of arrays," *Algorithmica*, vol. 24, no. 3-4, pp. 298-330, 1999.

[16] Aske Plaat, Henri E. Bal, and Rutger F.H. Hofman, "Sensitivity of Parallel Applications to Large Differences in Bandwidth and Latency in Two-Layer Interconnects," *Proceedings of the 5th IEEE High Performance Computer Architecture HPCA'99*, pp. 244-253, 1999.

[17] Xiao Qin and Hong Jiang, "Dynamic, Reliability-driven Scheduling of Parallel Real-time Jobs in Heterogeneous Systems," *Proceedings of the 30th ICPP*, Valencia, Spain, 2001.

[18] S. Ranaweera and Dharma P. Agrawal, "Scheduling of Periodic Time Critical Applications for Pipelined Execution on Heterogeneous Systems," *Proceedings of the 30th ICPP*, Valencia, Spain, 2001.

[19] D.P. Spooner, S.A. Jarvis, J. Caoy, S. Saini and G.R. Nudd, "Local Grid Scheduling Techniques using Performance Prediction," *IEE Proc. Computers and Digital Techniques*, 150(2): 87-96, 2003.

[20] Ming Zhu, Wentong Cai and Bu-Sung Lee, "Key Message Algorithm: A Communication Optimization Algorithm in Cluster-Based Parallel Computing," *Proceedings of the 1st IEEE International Workshop on Cluster Computing*, 1999.

Grid Enabled Master Slave Task Scheduling for Heterogeneous Processor Paradigm

Ching-Hsien Hsu, Tai-Lung Chen and Guan-Hao Lin

Department of Computer Science and Information Engineering
Chung Hua University, Hsinchu, Taiwan 300, R.O.C.
Email: chh@chu.edu.tw

Abstract: Efficient task scheduling is an important issue on system performance of computational grid. To investigate this problem, the master slave paradigm is a good vehicle for developing tasking technologies of centralized grid system. In this paper, we present an efficient method for dispatching tasks to heterogeneous processors in master slave environment. The main idea of the proposed technique is first to allocate tasks to processors that with lower communication overheads. A significant improvement of this approach is that average turnaround time can be minimized. The second advantage of the proposed approach is that system throughput can be increased by dispersing processor idle time. Our proposed model can also be applied to map tasks to heterogeneous cluster systems in grid environments in which the communication costs are various from clusters. To evaluate performance of the proposed techniques, we have implemented the proposed algorithms along with Beaumont's method. The experimental results show that our techniques outperform Beaumont's method in terms of lower average turnaround time, higher average throughput, less processor idle time and higher processors' utilization.

Keywords: master-slave paradigm, heterogeneous processors, task scheduling, computational grid, Least Job First

1. Introduction

One of the virtues of high performance computing is to integrate massive computing resources for accomplishing large computation problems. Cluster computing is one of the well known high performance paradigms. The use of master slave cluster of computers as a platform for high-performance and high-availability computing is mainly due to their cost-effective nature. As the growth of Internet technologies, computational grids become widely accepted paradigm for solving numerous applications and grand challenge problems.

Computing grid system integrates geographically distributed computing resources to establish a virtual and high expandable parallel machine. In recent years, more and more research work done in scheduling problem in heterogeneous grid systems. A centralized computational grid system can be viewed as the collection of one resource broker (the master processor) and several heterogeneous clusters (slave processors). Therefore, to investigate task scheduling problem, the master slave paradigm is a good vehicle for developing tasking technologies of centralized grid system.

The master slave tasking is a simple and widely used technique. Figure 1 shows an example of the master slave paradigm. One master node connects to *n* slave nodes. A pool of independent tasks are dispatched by master processor and be processed by the *n* slave processors. In a heterogeneous implementation, slave processors may have different computation speeds. Each slave processor executes the tasks after it receives its own part. Communication between master and slave nodes is handled through a shared medium (e.g. bus) that can be accessed only in exclusive mode. Namely, the communications between master and different slave processors can not be overlapped.

In general, the optimization of master slave tasking problem is twofold. One is to minimize total execution time for a given fix amount of tasks, namely minimize average turnaround time. The other one is to maximize total amount of finished tasks in a given time period, namely maximize throughput.

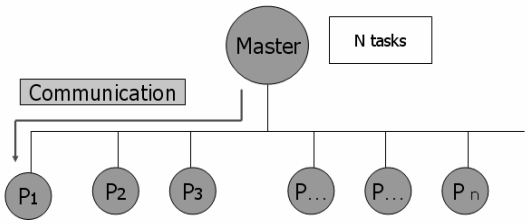


Figure 1: The Master-Slave paradigm.

In this paper, an efficient method for scheduling homogeneous tasks to heterogeneous processors in master slave environment is presented. The main idea of the proposed technique is first to allocate tasks to processors that with lower communication overheads. A significant improvement of this approach is that average turnaround

time can be minimized. The second advantage of the proposed approach is that system throughput can be increased by dispersing processor idle time. Our proposed model can also be applied to map tasks to heterogeneous cluster systems in grid environments in which the communication costs are various from clusters. To evaluate performance of the proposed techniques, we have implemented the proposed algorithms along with Beaumont's method [1, 2]. The experimental results show that our techniques outperform Beaumont's method in terms of lower average turnaround time, higher average throughput, less processor idle time and higher processors' utilization.

The rest of this paper is organized as follows. Section 2 briefly describes the related work. Section 3 introduces the research architecture and defines notation used in this paper. Section 4 presents characteristics of the master-slave model. Section 5 assesses the new scheduling algorithm. The performance comparisons and experimental results are discussed in section 6. Finally, section 7 makes conclusions.

2. Related Work

The task scheduling research about heterogeneous processors can be classified into DAG's model, master-slave paradigm and computational grids. The purpose of task scheduling is to achieve high performance computing and high throughput computing. The former aims at increasing execution efficiency and minimizing the execution time of tasks, whereas the latter aims at decreasing processor idle time and scheduling a set of independent tasks to increase the processing capacity of the systems over a long period of time.

Thanalapati *et al*. [5] bring up an idea about adaptive scheduling scheme based on homogeneous processor platform which used space-sharing and time-sharing to schedule tasks. Han *et al*. [6] presented a scheduling algorithm that enabled software fault tolerant for real-time environment. Recently, researches such as Topcuoglu *et al*. [7], Dogan *et al*. [8] and Hagras *et al*. [9] discussed the task scheduling for heterogeneous computing based on DAGs paradigm. In [7], a scheduling algorithm based on critical path mechanism to prioritize tasks is proposed. Srinivasan *et al*. [15] addressed the scheduling problem with reliability optimization for general heterogeneous computer systems. In [8], more investigations have been done based on incremental cost functions.

With the emergence of Grid and ubiquitous computing, new algorithms are in demand for addressing new concerns arising in the grid environment, such as security, quality of service and high system throughput. Berman *et al*. [11] and Cooper *et al*. [10] addressed the problem of schedule incoming applications to available computation power. Dynamically rescheduling mechanism was introduced for adaptive computing on the grid. Schopf *et al*. [16] present a general architecture with three phases for scheduling on the grid. In [17], an integrated technique for task matching and scheduling onto distributed heterogeneous computing systems is proposed. Based on Priority and Best Fit Mechanism, Min *et al*. [18] developed three novel scheduling Algorithms CO-RSPB, CO-RSBF and CO-RSBFP. In [19, 20], some simple heuristics for dynamic matching and scheduling of a class of independent tasks onto a heterogeneous computing system have been presented. Also an extended suffrage heuristic was presented in [21] for scheduling the parameter sweep applications which were implemented in AppLeS. They also presented a method to predict the computation time for a task/host pair by using the previous host performance.

Chronopoulos *et al*. [3], Charcranoon *et al*. [4] and Beaumont *et al*. [13.14] introduced the research of master-slave paradigm with heterogeneous processors background. Based on this architecture, Olivier Beaumont *et al*. [1, 2] presented a method on master-slave paradigm to forecast the quantity of tasks each processor needs to receive in a period of time. In their implementation, intuitionally, fast processor receives more tasks in the proportional distribution policy. Tasks are also prior allocated to faster slave processors and expected higher system throughput could be obtained.

3. Preliminaries

In this section, we first introduce the basic concept and models of this paper. Then, we define notations and terminologies that will be used in later sections.

3.1 Research Architecture

We revisit several characteristics that were introduced by Beaumont *et al*. [1, 2]. Based on the master slave paradigm demonstrated in figure 1, this paper conforms to the following assumptions.

- Heterogeneous processors: all processors have different computation speed.
- Identical tasks: all tasks are of equal size.
- Non-preemption: tasks are considered to be atomic.
- Exclusive communication: communications from master node to different slave processors can not be overlapped. This assumption can be changed

if non-blocking message passing is applied in grid system.

- Identical communication: all communications between master and slave processors are of same overheads. This assumption can be removed / extended when investigating the scheduling techniques on cluster based computational grid system in which the communication costs between different clusters are various.

To meet the above restrictions, communications between master and slave processors play an important factor to the overall system performance. Therefore, a good permutation of tasking that can avoid link contention and minimize processor waiting time is critical. We will present an efficient scheduling method that improved [1, 2] in the following sections.

3.2 Definitions

To simplify the presentation, we first define notations and terminologies used in this paper.

Definition 1: In a master slave system, master processor is denoted by M and the n slaves are represented by $P_1, P_2, ..., P_n$, where n is the number of slave processors.

Definition 2: Upon the assumption of identical tasks and heterogeneous processors, the time for slave processors to compute one task are different. We use T_i to represent the time of a slave processor P_i to complete one task. In this paper, we assume the computation speed of the n processors is sorted and $T_1 \le T_2 \le \ldots \le T_n$.

Definition 3: T_{comm} is the time of a slave processor to receive one task.

Definition 4: A Basic Scheduling Cycle (*BSC*) is defined as $BSC = lcm(T_1 + T_{comm}, T_2 + T_{comm}, ..., T_n + T_{comm})$, is the total amount of tasks in a scheduling cycle, where n is the number of processor.

Definition 5: The number of tasks a processor P_i must receive in a scheduling cycle is defined as

$$task(P_i) = \frac{BSC}{T_i + T_{comm}}.$$

Definition 6: The communication cost of processor P_i in *BSC* is defined as $comm(P_i) = T_{comm} \times task(P_i)$

Definition 7: The computation time of processor P_i in *BSC* is defined as $comp(P_i) = T_i \times task(P_i)$

Definition 8: The *performance factor* of processor P_i is defined as $\frac{T_{comm}}{T_i + T_{comm}}$. The computation capacity of a master slave system is defined as $\delta = \sum_{i=1}^{n} \frac{T_{comm}}{T_i + T_{comm}}$, where n is the number of slave processors.

We use an example to clarify the above definitions. Figure 2 shows the tasking on four heterogeneous processors. According to definition 2, the time for processors P_1 to P_4 to compute one task are T_1=2, T_2=3, T_3=3 and T_4=5. Communication cost between slave and master node to receive/send one task is define as T_{comm} = 1. According to definitions 4 and 5, *BSC* = *lcm*(3,4,4,6) = 12, *task*(P_1) = 4, *task*(P_2) = 3, *task*(P_3) = 3 and *task*(P_4) = 2. The communication cost and computation time of P_1 in *BCS* are *comm*(P_1) = 4 and *comp*(P_1) = 8, respectively. For other processors, these two values can be determined in a similar way by using the equations illustrated in definitions 6 and 7. Finally, according to definition 8, the performance factor of P_1 to P_4 are $\frac{1}{3}$, $\frac{1}{4}$, $\frac{1}{4}$, and $\frac{1}{6}$, respectively. The computation capacity in this example is $\delta = 1$. In Figure 2, a Greedy allocation method that adapts round robin scheduling policy is illustrated. Tasks are dispatched to faster and available processor first. As shown in Figure 2, the first 3 tasks are sent to P_1, P_2 and P_3. The fourth task is allocated to P_1 again because P_1 is faster than P_4. The fifth task is sent to P_4 which is the only one available processor. In the greedy algorithm, we observe that processors' idle time is scattered unevenly.

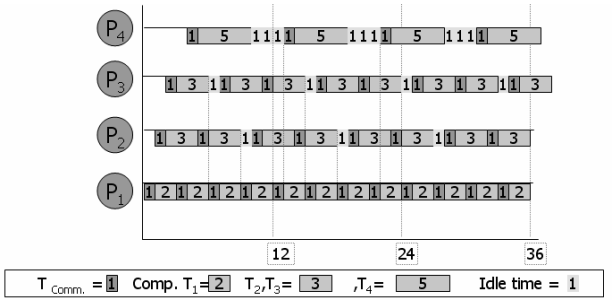


Figure 2: Task scheduling on 4 processors using greedy algorithm.

4. Master Slave Task Scheduling

In this section, we discuss the problem of task scheduling on master slave system in two cases depending on the value of system computation capacity (δ).

4.1 $\delta \le 1$ Scheduling Without Processor Idle

Figure 2 is the case of master-slave system with

$\delta \le 1$. We reuse this example to demonstrate the pre-scheduling algorithm proposed in [1, 2]. As mentioned in section 2, faster processor receives more tasks. Tasks are also prior allocated to them (faster processors). This is so called Most Jobs First (*MJF*). Figure 3 shows the pre-scheduling of tasks of the *MJF* algorithm. As defined in section 3.2, the performance factor of P_1 to P_4 are $\frac{1}{3}$, $\frac{1}{4}$, $\frac{1}{4}$, and $\frac{1}{6}$, respectively. Since *BSC* = 12, therefore, we can have $task(P_1)$=4, $task(P_2)$=3, $task(P_3)$=3 and $task(P_4)$=2 as shown in Figure 3. Furthermore, we observe that the second *BSC* connects to the previous one without any processor idle fragmentation. When the number of tasks is numerous, such scheduling achieves higher system utilization and less processor idle time than the greedy method.

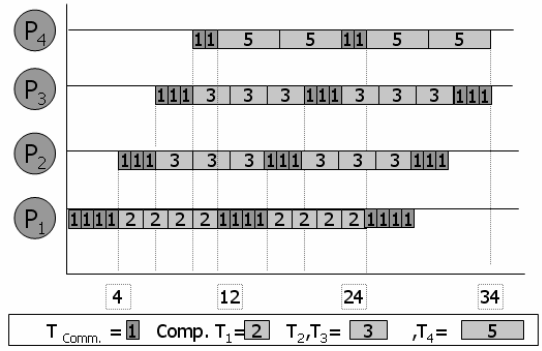


Figure 3: Most Jobs First (*MJF*) Tasking when $\delta \le 1$.

4.2 $\delta > 1$ Scheduling With Processor Idle

Definition 9: Given a master slave system with n heterogeneous processors, $P_{\max}$ is the processor P_k such that $\max\{k \mid \sum_{i=1}^{k} \frac{Tcomm}{T_i + Tcomm} \le 1\}$, where $1 \le k \le n$. i.e. $\sum_{i=1}^{k+1} \frac{Tcomm}{T_i + Tcomm} > 1$. We use $P_{\max+1}$ to represent processor P_{k+1}.

Corollary 1: Given a master slave system with $\delta > 1$, in *MJF* scheduling, $task(P_{\max+1}) = BSC - \sum_{i=1}^{\max} task(P_i)$.

Corollary 2: Given a master slave system with $\delta > 1$, in *MJF* scheduling, T_{idle}^{MJF} is the idle time of $P_{\max+1}$ and $T_{idle}^{MJF} = BSC - comm(P_{\max+1}) - comp(P_{\max+1})$.

Figure 4 shows another example of master slave system with $\delta > 1$. According to the pre-scheduling method described in section 4.1, we have $task(P_1)$=20, $task(P_2)$=15, $task(P_3)$=15. Since $\delta > 1$ in this example, $task(P_{\max+1}=P_4)$ = 10 as illustrated in Corollary 1. From Figure 4, we can see that the first sixty tasks are dispatched to P_1 to P_4 during time period 1 ~ 60 in the first *BSC*. The dispatching is start at time unit 60 in the second *BSC*. We note that P4 completes its tasks and becomes free at time 100. However, the master processor is dispatching tasks to P_3 during time 100 ~ 110 and sends tasks to P_4 start at time 110. This results P_4 stays idle during time period 100 ~ 110. This situation also happens at time 160~170, 220~230, and so on.

Lemma 1: Given a *MJF* scheduling with $\delta > 1$, the completion time of the j^{th} *BSC* can be calculated by the following equation.

$$T(BSC_j) = \sum_{i=1}^{\max+1} comm(P_i) + comp(P_{\max+1}) + (j-1) \times (comm(P_{\max+1}) + comp(P_{\max+1}) + T_{idle}^{MJF}) \quad (1)$$

Proof: Due to page limitation, we omit the proof in this version.

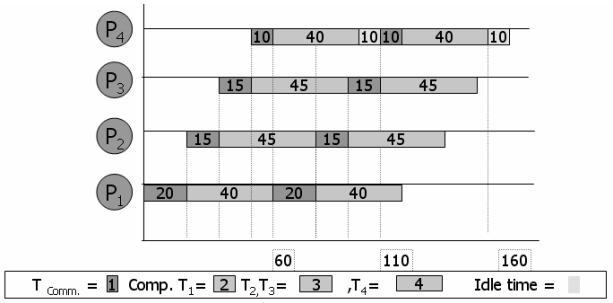


Figure 4: Most Jobs First (*MJF*) Tasking when $\delta > 1$.

5. Shortest Communication Time First (SCTF) Scheduling

The *MJF* scheduling algorithm distributes tasks to slave processor according to processors' speed. Faster processor receives tasks first. This is obviously an efficient approach if the communication contention between master and slave processors is not considered. When communication contention is interacted, the *MJF* algorithm is not optimal in terms of system throughput and average turnaround time. In this section, we present the Shortest Communication Time First (*SCTF*) algorithm. We also discuss the problem of task scheduling on master slave system in two cases depending on the value of system computation capacity (δ).

5.1 $\delta \leq 1$ Scheduling Without Processor Idle

We consider again the example in Figure 2 for examining master-slave scheduling with $\delta \leq 1$. Parameters of this example are recalled, we then have *BSC* = 12, *task*(P_1)=4, *task*(P_2)=3, *task*(P_3)=3 and *task*(P_4)=2. According to definition 6, the communication overheads within *BSC* of each slave processor are *comm*(P_1)=4, *comm*(P_2)=3, *comm*(P_3)=3 and *comm*(P_4)=2. In the *SCTF* implementation, tasks are prior allocated to slave processor that with shortest communication costs. Therefore, P_4 first receives 2 tasks and then P_3 receives 3 tasks, P_2 receives 3 tasks; finally, P_1 receives 4 tasks in the first *BSC*. As shown in Figure 5, the second *BSC* has the same distribution patterns of tasks as that in the first *BSC*. Compare to the example discussed in Figure 3, the completion time of the first *BSC* is accelerated from 22 to 20. Similarly, the second *BSC* is from 34 to 32. Consequently, the *SCTF* minimizes the average turnaround time.

Lemma 2: Given a *SCTF* scheduling with $\delta \leq 1$, the completion time of the j^{th} *BSC* can be calculated by the following equation.

$$T(BSC_j) = BSC + comp(P_1) + (j-1)\times(comm(P_1)+comp(P_1)) \quad (2)$$

Proof: Due to page limitation, we omit the proof in this version.

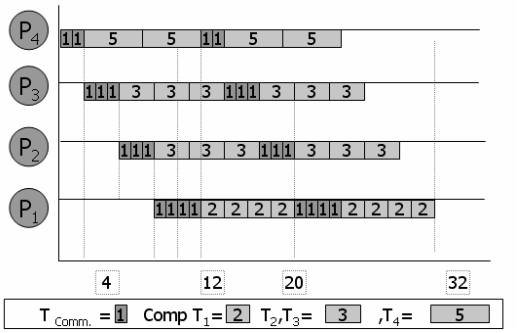


Figure 5: Shortest Communication Time First (*SCTF*) Tasking when $\delta \leq 1$.

5.2 $\delta > 1$ Scheduling With Processor Idle

We use the same example to in section 4.2 to demonstrate the scheduling method with dispersive idle when $\delta > 1$. According to definition 5, we have *task*(P_1)=20, *task*(P_2)=15, *task*(P_3)=15. Applying the *SCTF* concept illustrated in section 5.1, P4 first receives 12 tasks and then P_3 receives 15 tasks, P_2 receives 15 tasks; finally, P_1 receives 20 tasks in the first *BSC* as shown in Figure 6. Furthermore, we observe that P_4 completes its tasks at time 60. It becomes available and can receive more tasks for computing. However, the master processor is sending tasks to P_1. When t=62, master processor sends tasks to P_4 again. Therefore, during t=60~62, P_4 is idle. The same situation happens on P_3 at t=72~74, P_2 is idle at t=87~89 and t=102~104 P_1 remains idle. Compare to the example in Figure 4, P_4 stays 10 units of time idle in *MJF* algorithm while the idle time is reduced and dispersed in *SCTF* algorithm. In *SCTF*, every processor has 2 units of time idle and totally 8 units of time idle. Moreover, we observe that the *MJF* algorithm finishes 60 tasks in 100 units of time. The throughput is 0.6. While in *SCTF*, there are 62 tasks completed during 102 time units. The throughput of *SCTF* is 62/102 (≈0.61) > 0.6. Consequently, the *SCTF* algorithm delivers higher system throughput. On the other hand, the average turnaround time of the *SCTF* algorithm for the first two *BSC*s is 164/124 (≈1.32) which is less than the *MJF*'s average turnaround time 160/120 (≈1.33).

Corollary 3: Given an *SCTF* scheduling with $\delta > 1$, T_{idle}^{SCTF} is the idle time of each slave processor and $T_{idle}^{MJF} = \sum_{i=1}^{\max+1} comm(P_i) - BSC$.

Lemma 3: Given a *SCTF* scheduling with $\delta > 1$, the completion time of the j^{th} *BSC* can be calculated by the following equation.

$$T(BSC_j) = \sum_{i=1}^{\max+1} comm(P_i) + comp(P_1) + (j-1)\times(comm(P_1)+comp(P_1)+T_{idle}^{SCTF}) \quad (3)$$

Proof: Due to page limitation, we omit the proof in this version.

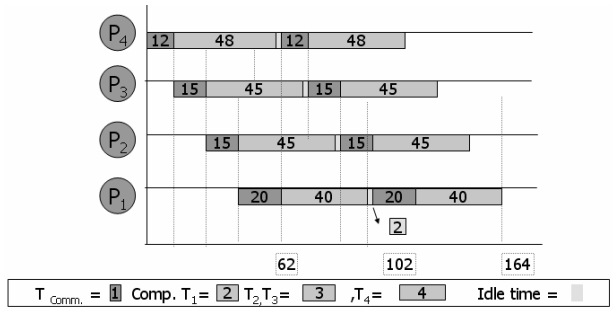


Figure 6: Shortest Communication Time First (*SCTF*) Tasking when $\delta > 1$.

5.3 Master with Computational Ability

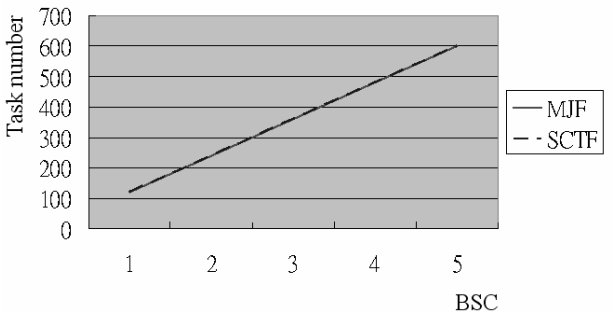
In most grid computing environment, the master processor is responsible to resource co-allocation and management. An alternative approach for master-slave computing is to include the computation power of master processor into the computing. If doing so, the throughput might be increased. Since there is no communication delay for master processor itself, it will be better on enhancing system performance to select the fastest processor as master node.

6. Performance Evaluation

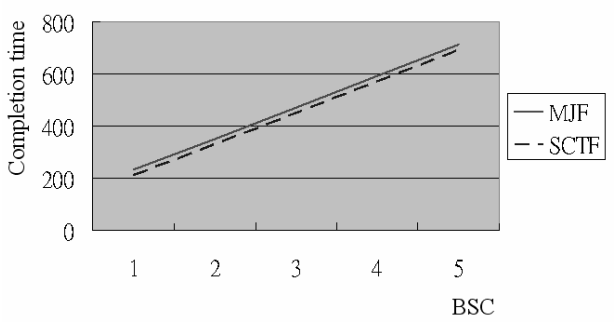
To evaluate the performance of the proposed method, we have implemented the *SCTF* and *MJF* algorithms. We compare some criteria in two scenarios; section 6.1 studies the simulation results for slight heterogeneous processors. The variation of processor speed is ±4; section 6.2 reports the simulation results for larger variation of processor speed ±10. Sections 6.3 and 6.4 present the performance of two algorithms upon different processor number and including or not the master processor for computation.

6.1 The variation of processor speed is small

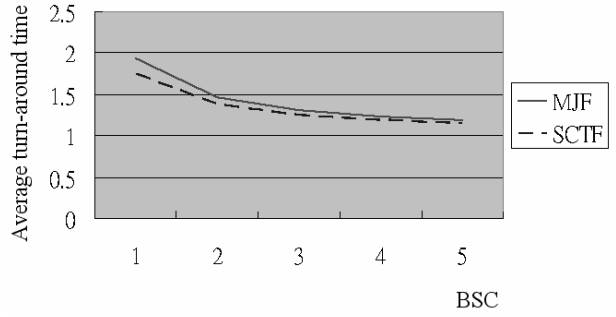
In this experiment, the processor number is 5. The computation speed of slave processors are T_1=3, T_2 =3, T_3 =4, T_4 =5 and T_5 =7 and $\delta \le 1$. We measured the average task turnaround time, *BSC* completion time and number of finished tasks in *BSC*s. Figure 7(a) shows the number of tasks processed in different number of *BSC*s. Both algorithms have the same result on this criterion. Figure 7(b) presents the completion time of each *BSC* in different algorithms. The *SCTF* method has lower completion time. Figure 7(c) gives average turnaround time during different number of *BSC* time period. The *SCTF* algorithm performs better than the *MJF* method. These phenomena also match the discussion in section 5.



(a)



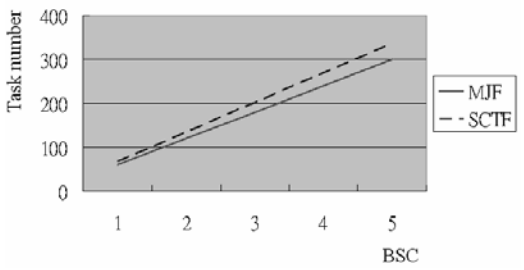
(b)



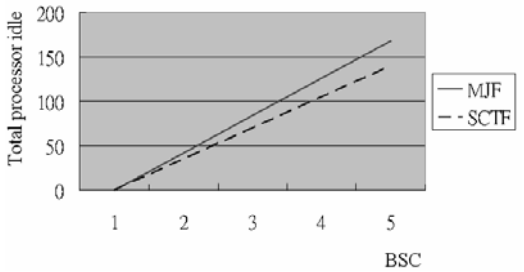
(c)

Figure 7: Simulation results for 5 processors with ±4 speed variation when $\delta \le 1$ (a) number of tasks completion (b) *BSC* completion time (c) Average task turnaround time.

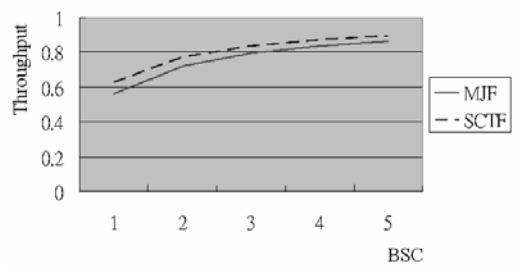
Figure 8 shows the simulation results for the following setting. The computation speed of processors are T_1=2, T_2=3, T_3=3, T_4=4 and T_5=6. Since $\delta > 1$, we add the comparison of processor idle time in this test. Figure 8(a) shows the number of tasks processed in different number of *BSC*s. We can see that the *SCTF* algorithm processes more tasks than the *MJF* method. Figure 8(b) presents processor idle time in different algorithms. The *SCTF* method has lower processor idle time. Figure 8(c) reports system throughput. As the example demonstrated, *SCTF* achieves higher system throughput. Figure 8(d) gives average turnaround time during different number of *BSC* time period. The *SCTF* algorithm performs better than the *MJF* method.



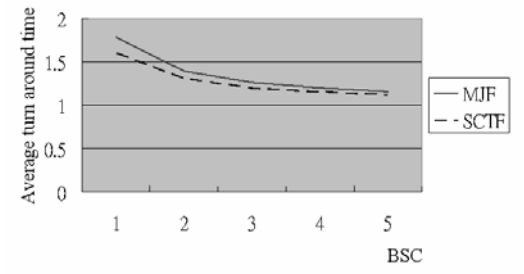
(a)



(b)



(c)

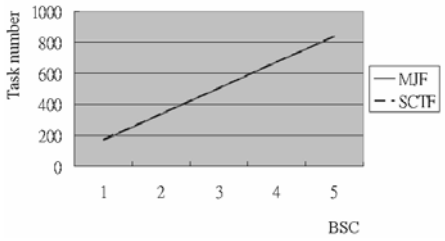


(d)

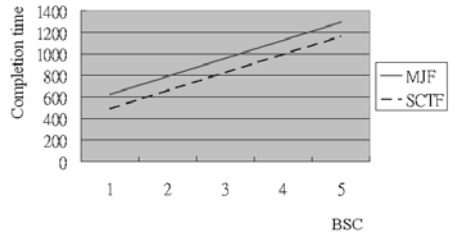
Figure 8: Simulation results for 5 processors with ±4 speed variation when $\delta > 1$ (a) number of tasks completion (b) processor idle time (c) system throughput (d) Average task turnaround time.

6.2 The variation of processor speed is large

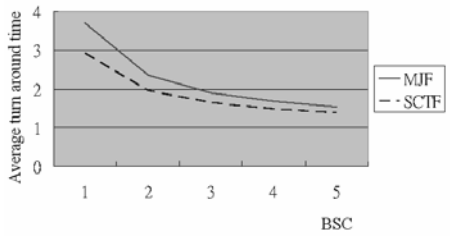
In this subsection, we discussed the variation of processor speed is large than the example of section 4.1. Test in this subsection is based ±10 processor speed variation. The computation speed of slave processors are T_1=3, T_2=3, T_3=5, T_4=7 and T_5=13. Figure 9 shows the simulation results for 5 processors with ±10 speed variation when $\delta \le 1$. We also measured the average task turnaround time, *BSC* completion time and number of finished tasks in *BSC*s. Figure 9(a) shows the number of tasks processed in different number of *BSC*s. Both algorithms have the same result on this criterion. Figure 9(b) presents the completion time of each *BSC* by both algorithms. The *SCTF* method has lower completion time. Figure 9(c) gives average turnaround time during different number of *BSC* time period. The *SCTF* algorithm performs better than the *MJF* method. These phenomena are similar to those obtained from Figure 7 and also match the analysis in section 5.



(a)



(b)



(c)

Figure 9: Simulation results for 5 processors with ±10 speed variation when $\delta \le 1$ (a) number of tasks completion (b) *BSC* completion time (c) Average task turnaround time.

Figure 10 shows the simulation results for 5 processors with ±10 speed variation when $\delta > 1$. The computation speed of processors are T_1=2, T_2=3, T_3 =4, T_4 =4 and T_5 =11. We also calculate processor idle time in this test. Figure 10(a) shows the number of tasks processed in different number of

*BSC*s. We can see that the *SCTF* algorithm processes more tasks than the *MJF* method. Figure 10(b) presents processor idle time by using the two algorithms. Obviously, the *SCTF* method has lower processor idle time. Figure 10(c) reports that the *SCTF* algorithm achieves higher system throughput than the *MJF* algorithm. Figure 10(d) gives average turnaround time during different number of *BSC* time period. The *SCTF* algorithm performs better than the *MJF* method.

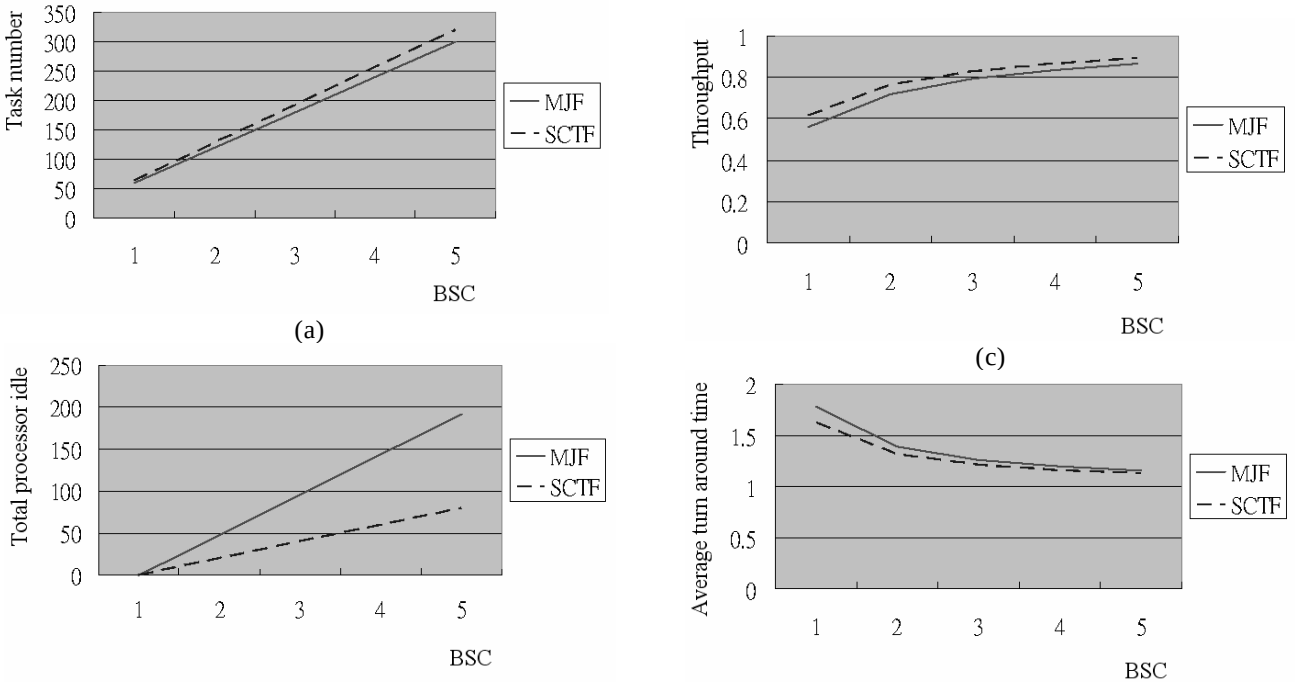


Figure 10: Simulation results for 5 processors with ±10 speed variation when $\delta > 1$ (a) number of tasks completion (b) processor idle time (c) system throughput (d) Average task turnaround time.

Another simulation for evaluating average turnaround time is made upon different number of processors and shown in Figure 11. The computation speed of those slave processors is set as T_1=3, T_2=3, T_3=5, T_4=7, T_5=11, T_6=13, and T_7=15. For the cases when processor number is 1, 2,…, 6, we have $\delta \leq 1$. When processor number increases to 7, we obtain $\delta > 1$. In either case, the *SCTF* algorithm conduces better average turnaround time. From the above results, we conclude that the *SCTF* algorithm outperforms *MJF* for most test samples.

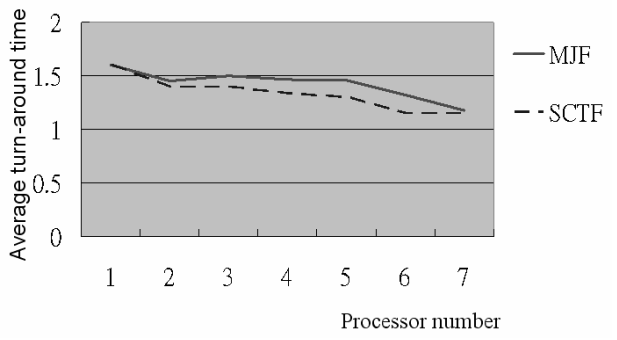


Figure 11: Task average turnaround time on different number of processors

6.3 Relationship of turn-around time and processor numbers when master with computational ability

In this subsection, we tested the system performance

for including the computation power of master processor into the task processing. Three levels of master node are tested. In Figure 12, Comp(*M*)=1, Comp(*M*)=5 and Comp(*M*)=10 represent the master node processes one task by 1, 5 and 10 units of time, respectively. Obviously, Comp(*M*)=1 has the best performance than others. Also, Comp(*M*)=1, Comp(*M*)=5 and Comp(*M*)=10 outperform the *SCTF* method. This result reflects the argument in section 5.3.

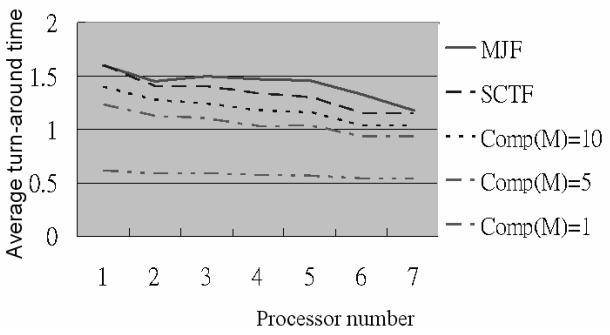


Figure 12: Performance comparison when master processor has different computation ability.

7. Conclusion

The master-slave task scheduling is a typical problem in computational grid system. In this paper, we have presented an efficient algorithm, *SCTF* for heterogeneous processors tasking. One significant improvement of the *SCTF* algorithm is that average turnaround time can be minimized. The second advantage of the proposed approach is that system throughput can be increased by dispersing processor idle time. Our preliminary analysis and simulation results indicate that the *SCTF* algorithm outperforms Beaumont's method in terms of lower average turnaround time, higher average throughput, less processor idle time and higher processors' utilization.

There are numbers of research issues remained in this paper. Our proposed model can also be applied to map tasks to heterogeneous cluster systems in grid environments in which the communication costs are various from clusters. In the future, we intend to devote generalized tasking mechanisms for computational grid. We will also study realistic applications and analyze their performance on grid system. Besides, the issues of heterogeneous communication overheads are also interesting and will be investigated.

Reference

[21] Oliver Beaumont, Arnaud Legrand and Yves Robert, "The Master-Slave Paradigm with Heterogeneous Processors," *IEEE Trans. on parallel and distributed systems*, Vol. 14, No.9, pp. 897-908, September 2003.

[22] Cyril Banino, Olivier Beaumont, Larry Carter, Fellow, Jeanne Ferrante, Senior Member, Arnaud Legrand and Yves Robert, "Scheduling Strategies for Master-Slave Tasking on Heterogeneous Processor Platforms," *IEEE Trans. on parallel and distributed systems*, Vol. 15, No.4, pp.319-330, April 2004.

[23] A.T. Chronopoulos and S. Jagannathan, "A Distributed Discrete-Time Neural Network Architecture for Pattern Allocation and Control," *Proc. IPDPS Workshop Bioinspired Solutions to Parallel Processing Problems*, 2002.

[24] S. Charcranoon, T.G. Robertazzi and S. Luryi, "Optimizing Computing Costs Using Divisible Load Analysis," *IEEE Trans. Computers*, Vol. 49, No. 9, pp. 987-991, Sept. 2000.

[25] Thyagaraj Thanalapati and Sivarama Dandamudi, "An Efficient Adaptive Scheduling Scheme for Distributed Memory Multicomputers," *IEEE Trans. on parallel and distributed systems*, Vol. 12, No. 7, pp.758-767, July 2001.

[26] Ching-Chin Han, Kang G. Shin, Jian Wu, "A Fault-Tolerant Scheduling Algorithm for Real-Time Periodic Tasks with Possible Software Faults," *IEEE Trans. on computers*, Vol. 52, No. 3, pp.362-372, March 2003.

[27] Haluk Topcuoglu, Salim Hariri, Min-You Wu, "Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing," *IEEE Trans. on parallel and distributed systems*, Vol. 13, No. 3, pp. 260-274, March 2002.

[28] Atakan Dogan, Fusun Ozguner, "Matching and Scheduling Algorithms for Failure Probability of Applications in Heterogeneous Computing," *IEEE Trans. on parallel and distributed systems*, Vol. 13, No. 3, pp. 308-323, March 2002.

[29] Tarek Hagras, Jan Janecek, "A High Performance, Low Complexity Algorithm for Compile-Time Task Scheduling in Heterogeneous Systems," *Proceedings of the 18th International Parallel and Distributed Processing Symposium* (IPDPS'04).

[30] Cooper, K. Dasgupta, A. Kennedy, K. Koelbel, C. Mandal, A. Marin, G. Mazina, M. Mellor-Crummey, J. Berman, F. Casanova, H. Chien, A. Dail, H. Liu, X. Olugbile, A. Sievert, O. Xia, H. Johnsson, L. Liu, B. Patel, M. Reed, D. Deng, W. Mendes, C. Shi, Z. YarKhan, A. Dongarra, J., "New Grid Scheduling and Rescheduling Methods in the GrADS Project," *Proceedings of the 18th International Parallel and Distributed Processing Symposium* (IPDPS'04).

[31] Francine Berman, Richard Wolski, Hernri Casanova, Walfredo Cirne, Holly Dail, Marcio Faerman, Silvia Figueira, Jim Hayes, Graziano Obertelli, Jennifer Schopf, Gary Shao, Shava Smallen, Neil Spring, Alan

Su, and Dmitrii Zagorodnov, ”Adaptive Computing on the Grid Using AppLeS,” *IEEE Trans. on parallel and distributed systems*, Vol. 14, No. 4, pp.369-379, April 2003.

[32] S. Bataineh, T.Y. Hsiung and T.G. Robertazzi, “Closed Form Solutions for Bus and Tree Networks of Processors Load Sharing a Divisible Job,” *IEEE Trans. Computers*, Vol. 43, No. 10, pp. 1184-1196, Oct. 1994.

[33] O. Beaumont, V. Boudet, A. Petitet, F. Rastello and Y. Robert, “A Proposal for a Heterogeneous Cluster ScaLAPACK (Dense Linear Solvers),” *IEEE Trans. Computers*, Vol. 50, No. 10, pp. 1052-1070, Oct. 2001.

[34] O. Beaumont, V. Boudet, F. Rastello and Y. Robert, “Matrix-Matrix Multiplication on Heterogeneous Platforms,” *Proc. Int'l Conf. Parallel Processing*, 2000.

[35] Santhanam Srinivasan and Niraj K. Jha, “Safety and reliability driven task scheduling allocation in distributed systems,” *IEEE Trans. on parallel and distributed systems*, Vol. 10, No. 3, pp. 238-251, March 1999.

[36] Jennifer M. Schopf, “A General Architecture for Scheduling on the Grid,” *TR-ANL/MCS-P1000-1002*, special issue of *JPDC on Grid Computing*, April, 2002.

[37] Muhammad K. Dhodhi, Imtiaz Ahmad Anwar Yatama, Anwar Yatama and Ishfaq Ahmad, “An integrated technique for task matching and scheduling onto distributed heterogeneous computing systems,” *Journal of Parallel and DistributedComputing*, Vol. 62, No. 9, pp. 1338–1361, 2002.

[38] Rui Min and Muthucumaru Maheswaran, “Scheduling Co-Reservations with priorities in grid computing systems,” Proceedings of the 2nd *IEEE/ACM International Symposium on Cluster Computing and the Grid* (CCGRID’02), pp. 250-251, May 2002.

[39] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen and R. Freund, ``Dynamic mapping of a class of independent tasks onto heterogeneous computing systems," Proceeding of the *8th IEEE Heterogeneous Computing Workshop* (*HCW '99*), pp. 30-44, Apr. 1999.

[40] T. D. Braun, H. J. Siegel, N. Beck, L. Boloni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys and B. Yao, “A taxonomy for describing matching and scheduling heuristics for mixed-machine heterogeneous computing systems,” Proceedings of the *IEEE Workshop on Advances in Parallel and Distributed Systems*, pp. 330-335, Oct. 1998.

[41] Henri Casanova, Arnaud Legrand, Dmitrii Zagorodnov and Francine Berman, “Heuristics for Scheduling Parameter Sweep applications in Grid environments,” *Proceedings of the 9th Heterogeneous Computing workshop* (*HCW*'2000), pp. 349-363, 2000

A Genetic-Fuzzy Logic Based Load Balancing Algorithm in Heterogeneous Distributed Systems

Kun-Ming Yu, Ching-Hsien Hsu and Chwani-Lii Sune
Department of Computer Science and Information Engineering
Chung-Hua University
Hsin-Chu, 300, Taiwan, R.O.C.

Abstract

Distributed processing is recognized as a practical way to achieve high performance in various computational applications. Many dynamic load-balancing algorithms have been proposed for parallel and discrete simulations. But the actual performances of these algorithms have been far from ideal, especially in the heterogeneous environment. In this paper, a hybrid approach using fuzzy supervised learning and generic algorithm is presented. The fuzzy membership function is dynamically adjusted by the genetic coding. Moreover, the proposed load-balancing algorithm has learning capability. The experimental results show that our proposed algorithm has better performance comparing with other classical load balancing algorithms.

Load-balancing, genetic algorithm, fuzzy logic, heterogeneous environment

1. Introduction

Load balancing in a distributed system is a process of sharing computational resources by transparently distributing system workload. With the advent of high-speed communication links, it has become beneficial to connect stand-alone computers in distributed manner through a high-speed link. The primary advantages of these systems are high performance, availability, and extensibility at low cost. Therefore, distributed computing has gained increasing importance in the recent as a preferred mode of computing over centralized computing. Many researches have proposed different kinds of approaches for the load balancing problem [10, 11, 15, 16].

A load balancing system is composed of three design issues: the information gathering policy, the negotiation policy and the migration policy [1]. Traditional strategies of the load balancing systems usually take advantage of some fix values to distinguish workload (e.g. over-loaded or under-loaded). Many load-balancing approaches based on this conjecture have been introduced in the past [1, 10, 11, 12, 15, 16, 17]. In conventional load balancing systems, resource indexes are necessary to be the input training data, and the output (threshold of workload) can be decided impersonally. But the output values are fixed; it cannot indicate the degree of the workload. Moreover, there exists a sharp distinction between members and non-members; the tasks reallocation action will be made frequently around the threshold. This will result in an unstable system and cause unnecessary overhead. Moreover, the workload estimation of each host is very difficult and time-consuming. To solve this problem, [16] proposed a fuzzy logical theory to estimate the load status of each node and apply fuzzy

information rule to determine the number of tasks shall be migrated on a heavily load node.

The fuzzy logic offers a framework for representing imprecise, uncertain knowledge. Similar to the way, in which human beings make their decisions, fuzzy systems use a mode of approximate reasoning, which allows it to deal with vague and incomplete information. However, fuzzy systems have the problem of determining its parameters. Ones of the most important parameters of fuzzy system are the Membership Functions (MF). The fuzzy inference engine in conjunction with the control rules to determine an appropriate output response then uses the value ranges. In most fuzzy systems, the shape of MF of the antecedent, the consequent and fuzzy rules were determined and tuned through trial and error by human operators. Therefore it takes much iteration to determine and tune them. There are simple methods to turn MF such as Neural Networks [2], genetic algorithms (GA) were used as in [14], and the GA has give faster learning response than the neural networks.

Therefore, we purposed a genetic algorithm approach to construct a *fuzzy logic control* distributed system. This fuzzy logic artificial intelligence setting adjusts controller parameters or membership functions by genetic algorithm, it will not only have the power to improve the efficiency of multitask migration but also have fuzzy parameter learning capability. The fuzzy membership function can be adjustable according to the change of system environment immediately.

The organization of this paper is as follows. Section 2 describes the related work including the load balancing approaches. We then discuss several famous load balancing algorithms and the fuzzy enhanced symmetric algorithm [17] in section 3. We then present the proposed scheme with genetic algorithm embedded fuzzy enhance symmetric algorithm in section 4. Section 5 states the implementation issue as well as the experimental results. The conclusion and future work are provided in the last section.

2. Related Work

Load balancing can be performed either statically or dynamically. Previous researches on static and dynamic load balancing can be found in [4, 7, 13], respectively. In static load balancing, the tasks are assigned to nodes by analyzing their past behaviors or only using some conventional rules, which are independent of the actual current system state. The principal advantage of static policies is their simplicity. There is no need to maintain and process the information about the system state. The results of the previous studies suggest that dynamic policies have greater potential for performance improvement than static policies. There was approach using fuzzy logic control to enhance the dynamic load-balancing algorithm had been published [17]. But the fuzzy membership function was defined by benchmark program offline. After the system running a long period of time, the system status may change a lot. Therefore, the fuzzy membership function of cannot reflect the real situation of the environment.

The workload collection is one of the most important issues in dynamic load balancing approach. The information collection policy denotes not only the amount of workload about the systems but also the information gathering rules used in making the tasks reallocation decision. The goal of this policy is to obtain sufficient

information in order to make a decision weather the host's load is heavy or not. We say that a good information gathering policy [6] should be able to predict the workload in the near future, relatively stable and have a simple (ideally linear) relationship with the resource indexes. But it is difficult in real world, especially in heterogeneous systems.

Many traditional load-balancing schemes used a threshold value as the information policy after load indices generated. If the load index is above (under) the threshold, the host is said to be over-loaded (under-loaded), we can find out that the load status is classified into only two states, heavily or lightly. This binary-state makes the system load state fluctuate between heavily or lightly load wildly when the workload is near the threshold value. It will cause the task reallocation frequently because of little load change. Some researchers added a tolerance range around the threshold value [15, 16].

3. The Dynamic Load Balancing System

The distributed systems can be characterized by distribution of both physical and logical features. The architecture of a distributed system is usually modular and consists of a possibly varying number of processing elements. An arbitrary number of system and user processes may be executed in the system. A process can usually be executed on various machines. There are a number of factors to be considered when selecting a machine for process execution. These factors may include resource availability and utilization of various resources. A dynamic load balancing policy can employ either centralized or distributed control.

3.1 Centralize Load Balancing Model

In this model [3], a processor was appointed to be the centre controller, which collects and updates the information about the state of every other processor in the system. When a node decides that a task is eligible for load balancing, it sends a request to the specified processor to determine the suitable placement of the task. The advantage of this architecture was task reallocation action could be done accurately. But in this scheme has a potential risk, if the centre controller crashed, the system cannot work anymore, and in a large system this information traffic can't ensure deliver the host processor when the network is busy.

3.2 Distributed Load Balancing Model

In distributed model, every host has a local monitor associated. Each monitor collects and updates the information about the state of the local host. The primary advantages of this model are high performance, availability, and extensibility at low cost. Conventional algorithms of this model include *Random*, S*ender-Initiated* [5], *Receiver-Initiated* [5] and *Symmetric Algorithm* [1,17].

3.2.1 Random Algorithm

Among the algorithms, the Random Algorithm is the simplest one [1]. In this algorithm, each node checks the local workload during a fixed time period. When a node becomes over loaded after a time period, it sends the

newly arrived job to a node *randomly* no matter the load of target node is heavily or not. Only the local information is used to make the decision. The Random Algorithm has the lowest overhead because of its simplicity and without negotiation with other hosts. However, it can't reallocate the system load balancing very well.

3.2.2 Sender Algorithm

The Sender algorithm is based on the Sender policy [5]. When a node becomes over-loaded after a period of time, it selects the target node randomly and looking for its load status which is under-loaded or not. If it is under-loaded, an *ACCEPT* message is feedback to original host, otherwise it replies a *REJECT* message. If the requesting node is still over-loaded when the *ACCEPT* reply arrives, the newly arrived task is transferred to the probed node; otherwise the task keeps executing locally. This mechanism seals to push a task from the requesting node to the probed node after a period of time checking.

3.2.3 Receiver Algorithm

The Receiver Algorithm is designed according to the Receiver policy [5]. Once if a host becomes under-loaded, the node will poll the information form any other node to check if it is over-loaded. When an overloaded nodes was found, an *ACCEPT* message is feedback, otherwise it replies a *REJECT* message. The migration of a task from the probed node is still under-loaded.

3.2.4 Symmetric Algorithm

In comparison with the Sender Algorithm and the Receiver Algorithm, the symmetric algorithm shows two-side effects: when a node becomes over-loaded, Sender algorithm enabled; when it is under-loaded, the Receiver algorithm is active. This algorithm is combination version of the Sender and Receiver algorithm [1]. In other words, this model is adjusted based on the current load-level of the node by allowing the algorithm to switch automatically between Sender and Receiver algorithm. When the load status is over-loaded, it plays the role of the Sender algorithm; in contrast, it plays the role of Receiver algorithm.

3.2.5 Fuzzy Enhanced Symmetric Algorithm

Most conventional information gathering policies use load indices with a threshold value to determine the load status of host. The major problem is how to define an appropriate threshold value. The fuzzy theory can improve the information policy would be more objective and flexibility [16] to be the migration policy has improved this shortcoming. Some researchers use fuzzy logic control to solve this problem [17]. In [17], the experimental results show that used the fuzzy inference rules to obtain the migrated task numbers.

The typical architecture of a Fuzzy Logic Control (FLC) is composed of four principal components: a *Fuzzifier*, a

Fuzzy Rule Base, an *Inference Engine*, and a *Defuzzier*. The workload is determined by using FLC in [17]. According to the host's status is over-loaded or under-loaded, negotiation policy would initiate to find the suitable host to make the task migration. If the target node was being found, the migration policy will take advantage of defuzzification to calculate to number of tasks to be migrated.

But this method still not good enough to build a good dynamic load-balancing environment, because of the parameters of FLC be inaccurate and cannot make the best decision of the migrated number of tasks. In order to overcome this shortage, therefore, we proposed a new scheme that embedded online genetic algorithm to tune the fuzzy membership function dynamically. It can adjust the membership function in terms of the feedback values dynamically and react the overall systems status immediately.

4. Genetic based fuzzy logic control system

The genetic algorithm (GA) is an optimization search algorithm. GA is known to be particularly suitable for learning in complex domains and hence can be used for structure and parameter adaptation in fuzzy system, but it takes a considerably long time to converge to a suitable solutions. The basic concepts of GA were developed by Holland [8, 9], and have subsequently been extended in several research studies. Typically the GA starts with little or no knowledge of the correct solution and depends entirely on responses from an interacting environment and its evolution operators to arrive at good solutions.

The GA processes imitate natural evolution, and hence include bio-mimetic operation such as reproduction, crossover, and mutation. A conventional GA has four features: population size, reproduction, crossover, and mutation. GA's maintain a set of candidate solution called a population. Candidate solutions are usually represented as strings of fixed length, called chromosomes, coded with binary character set. The first step of GA is to generate an initial population by random in cycles called generations. The chromosome is represents by a binary string matrix depending upon the system condition. By applying the operators such as selection, crossover and mutation, the chromosome with the highest fitness is chosen to determine the population chromosome

In the paper, the genetic algorithm was designed to adjust the value of membership function of Fuzzy System. An increase in the number of input variables causes an exponential growth in the number of rules generated. We devise an online genetic algorithm (OGA for short) adaptive mechanism for updating of the associated parameters of fuzzy membership function dynamically.

4.1 Online Genetic Algorithm

In OGA processes such as crossover, reproduction and mutation will proceed in the usual manner. The following genetic operation operations are applied to each string:

(1) Coding: The coding of fuzzy membership functions in a chromosome is shown in Figure 1. A triangular membership function is used in the fuzzy set, a and b are representing the center and width of the membership function, respectively. The type of coding used in this research is the concatenated binary string by the position

of membership function center and width.

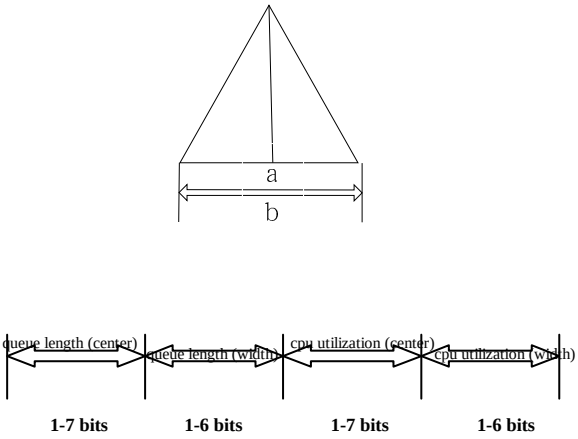


Figure 1: Coding of a fuzzy membership function

(2) Population size: The choice of an appropriate population size is a fundamental decision to be taken in all GA implementations. If the population sizing is too small the GA will usually converge too quickly, and too large a population will take a very long time to evaluate. In the study, population sizes are set to 50 and each chromosome is 26 bits.

(3) Reproduction: Reproduction is the process through selecting two parent genes from the current population. Selection is based probabilistically on a gene's fitness value; the higher the fitness of a gene, the more likely it can reproduce.

(4) Crossover: Crossover operates on two solution strings and results in another two strings. Typical crossover operator exchanges the segments of selected strings across a crossover point with probability. There are two steps produces two offspring by crossover operator. At first, two strings from the reproduced population are mated at random, and a crossover site is randomly selected. Then the strings are crossed and separated at the site. We used two point crossover-site for each parent strings with crossover probability Pc.

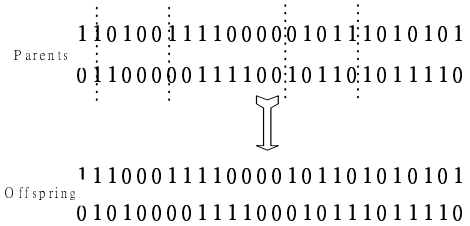


Figure 2: Crossover operation

(5) Mutation: The mutation operator prevents irreversible loss of certain patterns by introducing small random changes into chromosomes. Change each bit value with the probability Pm.

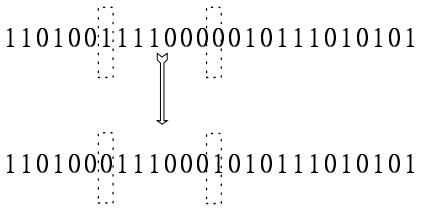


Figure 3: Mutation operation

(6) Fitness Function: The genetic algorithm is able to optimize the characteristics explicit in the fitness function. Here the fitness function using following formula:

$$F = \frac{1}{\alpha * RT^{2} + \beta * TT^{1/2}}$$

where 0 <= α, β <= 1,

RT : the response time,

TT : the turnaround time.

The process of Online Genetic Algorithm for fuzzy control is presented in figure 4.

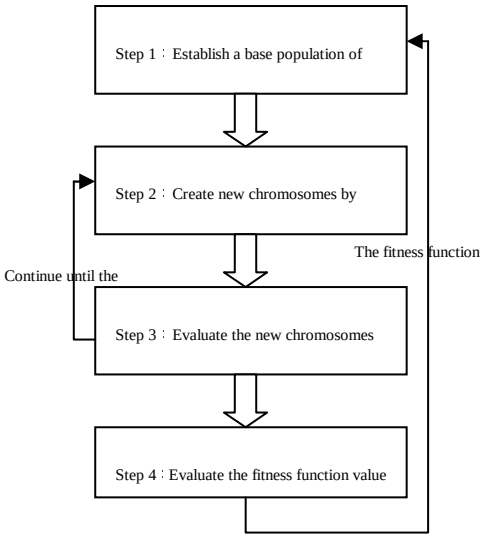


Figure 4: Online Genetic algorithm for fuzzy control

4.2 Structure of Genetic Based Fuzzy Logic Control Systems

There are four components in our proposed genetic based fuzzy logic control load balancing system: information module, negotiation module, migration module, and online genetic algorithm module. The information module defines the workload status of every host. Then the negotiation module will probe the target host to request the task reallocation action. The migration module will make the decision of the migrated number of tasks and move

the tasks to the target host. Finally, the online genetic algorithm will adjust the center value of fuzzy membership function, if the workload was still heavy (or light). Moreover, the online genetic algorithm will evaluate the current fuzzy rules to meet the load index or not. The architecture of genetic based fuzzy logic control model is shown in figure 5.

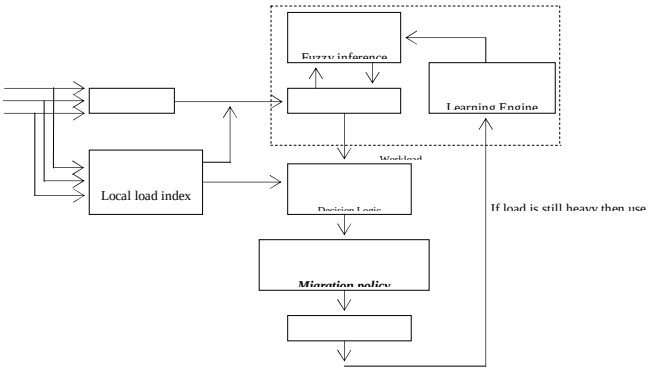


Figure 5: The architecture of our proposed load balancing algorithm

5. Implementation and experimental results

In order to verify the performance of our proposed scheme we implement our algorithm in a distributed environment, called *Java Load Balancing System* [18], which is implemented by using Java language. The system supports heterogeneous, static load balancing and dynamic load balancing. We also implemented three other algorithms (random, receiver initial, and symmetric) for comparsion. In order to verify that our proposed online genetic based fuzzy logic control load balancing algorithm will accomplish a high system performance. Six workstations running different operation systems. The operation systems including the Unix, Win NT and Win 98.

5.1 Online Genetic Algorithm Parameter

The genetic operation should be used in a way that achieves high-fitness individuals in the population rapidly without leading to a total convergence. In the paper, we used partial-random method to achieve high-fitness for a short-time interval. In our experiment, the size of population is set to 50, the total generations is 1000, the probability of crossover =0.8, the probability of mutation is 0.02.

5.2 Experimental results

In the experiment, the task number is adjustable parameters. In the static load balancing part, we used Random dispatch model. Four different algorithms were implemented, including random, receiver initial, symmetric, and our proposed algorithm. We have not implemented the sender-initial algorithm, since its performance usually worse than receiver-initial policy. In the experiment,we compare the performance of average response time, average turnaround time and overall throughput.

5.2.1 Response Time

The response time denotes the time from the submission of a task until the first response is produced. In figure 6, we can see that our algorithm can cut down the average average response time at least 50% when the tasks number is 70.

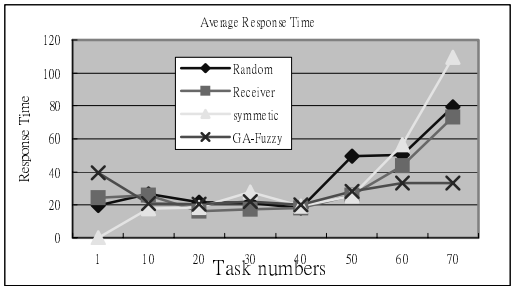


Figure 6 : Response time

5.2.2 Turnaround time

The second index can estimate the efficient of overall system is the turnaround time. The turnaround time is the interval from the time of a task submission to the time of completion. Figure 7 is the average turnaround time of four algorithms. In the figure, we can find out our algorithm has the smallest turnaround time.

5.2.3 Throughput

The overall throughput under different load balancing schemes is also discussed in our experiment. The definition of throughput is the number of processes that are completed per time unit. In figure 8, we also can find that our proposed algorithm always keeps higher throughput when the tasks number is increased.

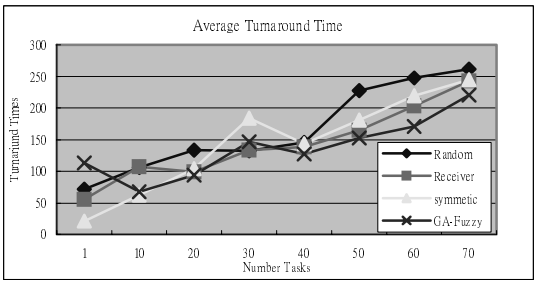


Figure 7: Turnaround time

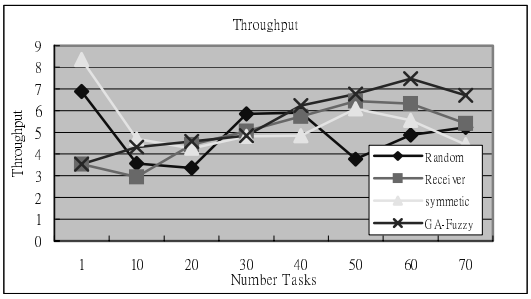


Figure 8: Throughput

6.Conclusion

In this paper, we design and implement an intelligent dynamic load balancing algorithm based on online genetic based fuzzy logic control. In our research, the proposed algorithm can correctly evaluate the workload of each machine in the system and make a decision of the migrate task extactly. In the scheme, OGA can dynamically adjust the fuzzy membership function based on the feedback information.

The experimental results show that our proposed load balancing can indeed significantly reduce the response time and turnaround time as well as increasing overall throughput.

Reference

[1] K. Benmohammed-Mahieddine, P. M. Dew, and M. Krar, " A periodic Symmetrically-Initiated Load Balancing Algorithm for Distributed System", IEEE ICPDS, 1994

[2] F. Cho, C. Ku, "Unsupervised/Supervised Learning for RBF-Fuzzy System", In Herrera, F. Verdegay, J. L. (Eds.) Genetic Algorithms and Soft Computing, 1997

[3] Pallab Dasgupta, A. K. Majumder, and P. Bhattacharya, "V_THR: An Adaptive Load Balancing Algorithm", Journal of Parallel and Distributed Computing 42, 101-108, 1997

[4] Derek L. Eager, Edward D. Lazowska and John Zahorjan, "A Comparison of Receiver-Initiated and Sender-Initiated Adaptive Load Sharing", Journal ACM July 1985, pp.1-3.

[5] D. L. Eager, E. D. Lazowska, and J. Zahorjan, "A Comparison of Receiver-Initiated and Sender-Initated Adaptive Load Sharing", Proc. of the 1985 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, pp. 1-3, Aug. 1985

[6] D. Ferrari and S. Zhou, An Empirical Investigation of Load Indices for Load Balancing Applications, Tech. Rep.UCB/CSD/87/353, Computer Science Division, Univ. of California, Berkeley, CA, 1987.

[7] Anna Ha'c and Theodore J. Johnson, "A Study of Dynamic Load Balancing in a Distributed System", Journal ACM August 1986, pp.348-356.

[8] J.H. Holland, "Adaptation in Natural and Artificial System", Univ. of Michigan Press, Ann Arbor, Mich, 1975.

[9] J.H. Holland, "Genetic algorithms and the optimal allocation of trials", SIAM J. Comput., 1973, 2,(2) , pp.89-104

[10] H. C. Lin and C. S. Raghavendra, "A Dynamic Load Balancing Policy with a Central Job Dispetcher", IEEE Trans. On Parallel and Distributed System, Jul. 1991

[11] K. B. Mahieddine, P. M. Dew, and M. Krar, "A periodic symmetrically-Initiated Load Balancing Algorithm for Distributed Systems", IEEE ICPDS, 1994

[12] Margaret Schaar, Kemal Efe, Lois Delcambre, and Laxmi N. Bhuyan, "Load Balancing with Network Cooperation", IEEE Trans. On Parallel and Distributed System, Jul. 1991, pp. 328-335.

[13] Asser N.Tantawi and Don Towsley, "Optomal Static Load Balancing in Distributed Computer Systems", Journal of the Association for Computing Machinery, Vol. 32, No. 2, Apirl 1985, pp.445-465.

[14] C. Wong, "An Auto-Generating Method in the Fuzzy System Design", Fuzzy-IEEE 97, Braceloana-Spain, pp. 1651-1654, 1997.

[15] Kun-Ming Yu, Siman J-W. Wu, and Tzung-Pei Hong, "A Load Balancing Algorithm Using Prediction", 1997 Workshop on Distributed System Technologies & Application, pp. 496-503, 1997

[16] Kun-Ming Yu and L. K. Wang, "A Dynamic Load Balancing Algorithm Using Artificial Neurnal Network", Proceedings of the IASTED International Conference on Artificial Intelligence and Soft

Computing, pp. 364-367, 1997
[17] Kun-Ming Yu, Yau-Tien Wang and Chih-Hsun Chou, “A Dynamic load balancing Approach using Fuzzy Logic Control”, Proceedings of the IASTED International Conference Artificial Intelligence and Soft Computing (ACS’99), August 9-12,1999
[18] Kun-Ming Yu and Daniel Chen, ”Design and Implementation of a Load Balancing Distributed System”, A thesis submitted to Chung-Hua Polytechinc Institute, taiwan, July, 1998