行政院國家科學委員會專題研究計畫 成果報告

平行資料程式於計算網格上通訊與 I/O 局部化研究與應用工具開發(3/3)

研究成果報告(完整版)

計 畫 類 別 ：整合型
計 畫 編 號 ：NSC 96-2221-E-216-001-
執 行 期 間 ：96 年 08 月 01 日至 97 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：碩士班研究生-兼任助理人員：張智鈞
碩士班研究生-兼任助理人員：郁家豪
碩士班研究生-兼任助理人員：蔡秉儒
博士班研究生-兼任助理人員：陳泰龍

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 ：本計畫涉及專利或其他智慧財產權，2 年後可公開查詢

中 華 民 國 97 年 10 月 30 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

平行資料程式於計算網格上通訊與 I/O 局部化研究與應用工具開發(3/3)

計畫類別：☑ 個別型計畫 □ 整合型計畫
計畫編號：NSC95-2221-E-216-006
執行期間：96 年 8 月 1 日至 97 年 7 月 31 日

計畫主持人：許慶賢 中華大學資訊工程學系副教授
共同主持人：
計畫參與人員： 陳泰龍（中華大學工程科學研究所博士生）
張智鈞、郁家豪、蔡秉儒(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

執行單位：中華大學資訊工程學系

中 華 民 國 97 年 10 月 31 日

行政院國家科學委員會專題研究計畫成果報告

平行資料程式於計算網格上通訊與 I/O 局部化研究與應用工具開發(3/3)

Design and Implementation of Communication and I/O Localization Tools for Parallel Applications on Computational Grids (3/3)

計畫編號：NSC95-2221-E-216-006
執行期限：96 年 8 月 1 日至 97 年 7 月 31 日
主持人：許慶賢　　中華大學資訊工程學系副教授

計畫參與人員：中華大學資訊工程學系研究生
陳泰龍(博二)、張智鈞(研二)、郁家豪(研二)、蔡秉儒(研二)

一、中文摘要

本報告是有關於在異質性計算網格系統和網路拓樸下開發適應型的評估模組與通訊局部化的技術之描述。本計畫執行三年，我們完成自動資料分割工具、平行資料程式效能預測工具、資料局部化選擇器、以及針對特殊平行應用程式的資料局部化學習系統。本項研究所發展的通訊局部化技術與分析工具有助於提升平行資料程式在計算網格上的執行效能。執行本計畫所得到的研究理論、工具開發、與實務經驗亦可作為相關領域學術研究與教學的素材。

關鍵詞：通訊區域化、平行資料程式、計算網格、平行 I/O、資料配置、通訊排程、效能預測、平行編譯器、平行應用、SPMD。

Abstract

This report presents adaptive performance models for optimizing communications of real world parallel applications on heterogeneous grid systems and topologies. This project developed tools for automatic data partitioning, performance prediction of data parallel programs, web-based locality selector and learning systems for scientific applications. The integrated locality preserving techniques and analysis tools developed in this project will facilitate development of efficient data parallel applications on computational grids. The achievements of theorems, tools and experience in this project can be applied in both academic teaching and research. It is the main objective of this project.

Keywords: Localized Communication, Data Parallel Program, Computational Grid, Parallel I/O, Data Distribution, Communication Scheduling, Performance Prediction, Parallelizing Compiler, Parallel Applications, SPMD.

二、緣由與目的

整合計算資源的觀念使得網格計算成為廣泛被接受的虛擬高效能計算平台。網格(Grid Computing)計算系統不同於傳統平行電腦,它連接分散於不同網域的電腦組成一個具有高度擴充性的計算平台。叢集式的網格(Cluster Grid)即是一個典型的系統。對於平行資料程式(Data Parallel Program)而言，程式執行的過程中有可能發生資料的切割、資料的交換，這種情況，在網格系統中，節點之間的通

訊必然發生。計算節點之間的通訊有可能發生於相同叢集之內(Interior Communication)的電腦，也有可能發生於不同叢集系統之間(External Communication)的電腦。為了減少通訊產生的代價，通訊局部化(Localized Communication) 將資料分佈在適當的電腦，使得程式執行過程中節點之間所必須的通訊可以大部分集中在相同的叢集或相同的網域之內。通訊局部化的問題不僅在通訊的層次，其可能應用的範圍包含資料的局部化(Data Localization)、I/O 的局部化(Grid I/O Localization)、處理節點的局部化(Processor Group Localization)。在傳統的平行電腦與分散式記憶體環境之下，有許多類似的研究。這些研究包括通訊的區域化或局部化、通訊排程(Communication Scheduling)、資料分割(Data Partitioning)、資料重新分佈(Data Redistribution)、處理器映對(Logical Processor Mapping)技術等。我們在這一個計畫中，就是要整合過去的這些技術，並且發展適用於網格環境下的方法，同時開發相關輔助的分析與調整工具，建立出一套有效而且簡單的方法與介面，使得平行資料應用程式(Data Parallel Applications)在未來的網格計算系統中可以有更多的應用。

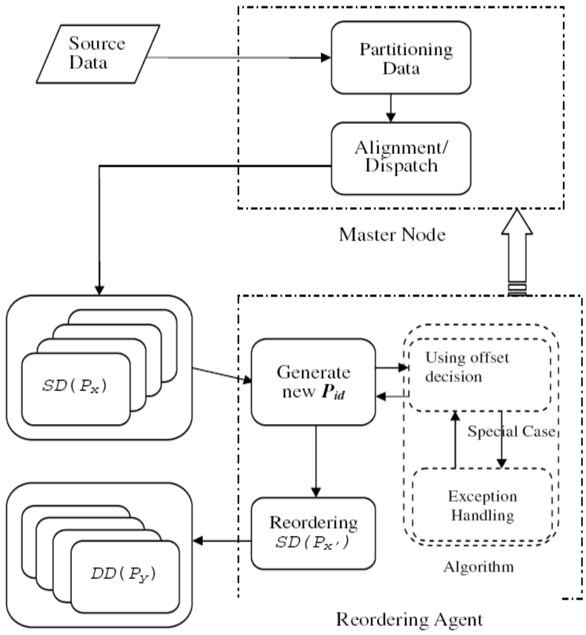
三、研究方法與成果

針對異質性網格計算環境(Non-identical Cluster Grid)內部與外部通訊的問題，我們提出在實際應用程式上進行最佳化的研究。圖一是網格環境中，內、外部通訊的示意圖。在這個網格環境中有三個電腦叢集，總共有十二個處理器。$P_{0\sim2}$ 屬於第一組電腦叢集；$P_{3\sim5}$ 屬於第二組電腦叢集；$P_{6\sim11}$ 屬於第三組電腦叢集。這三組電腦叢集共有六筆資料($a_{1\sim3}$, $f_{1\sim3}$, $g_{1\sim3}$, $h_{1\sim3}$, $k_{1\sim3}$, $l_{1\sim3}$)是傳送給內部處理器，但也有六筆資料傳送給外部處理器($b_{1\sim3}$, $c_{1\sim3}$, $d_{1\sim3}$, $e_{1\sim3}$, $i_{1\sim3}$, $j_{1\sim3}$)。外部通訊量等同於內部通訊量，在未經過通訊最佳化的處理前，大量的外部通訊花費更多通訊成本。

	P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	P_{11}	I	E
P_0	a_1	a_2	a_3										3	0
P_1				b_1	b_2	b_3							0	3
P_2							c_1	c_2	c_3				0	3
P_3										d_1	d_2	d_3	0	3
P_4	e_1	e_2	e_3										0	3
P_5				f_1	f_2	f_3							3	0
P_6							g_1	g_2	g_3				3	0
P_7										h_1	h_2	h_3	3	0
P_8	i_1	i_2	i_3										0	3
P_9				j_1	j_2	j_3							0	3
P_{10}							k_1	k_2	k_3				3	0
P_{11}										l_1	l_2	l_3	3	0
	Cluster-1			**Cluster-2**			**Cluster-3**						18	18

圖一、異質性網格環境中的資料通訊示意圖。

圖二是處理器重新排序的示意圖，利用重新排序的技術，將外部通訊轉換成內部通訊，可有效減少通訊成本。切割 Source Data 以後，由 Master Node 分配給每個 Source Node，而 Reordering Agent 利用重新排序的技術，提供 Source Node 新的 Destination Node。由於屬於內部處理器的 Destination Node 個數提高了，讓內部通訊量增加，使得通訊成本降低，讓內部通訊量增加，使得通訊成本降低且更有效率。



圖二、重新排序處理器的邏輯 ID 之演算法流程。

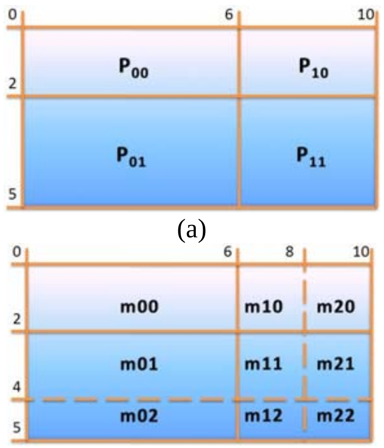
在經過 Reordering Agent 將處理器邏輯 ID 重新排序後，原本屬於外部通訊的 $b_{1\sim3}$, $c_{1\sim3}$, $d_{1\sim3}$, $e_{1\sim3}$, $i_{1\sim3}$, $j_{1\sim3}$ 等六筆資料被轉換成內部通訊，如圖三。使得所有通訊均為內部通訊，有效降低通訊所花費的成本。在實驗中也驗證了此一事實。

	P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	P_{11}	*I*	*E*
P_0	a_1	a_2	a_3										3	0
P_3				b_1	b_2	b_3							3	0
P_6							c_1	c_2	c_3				3	0
P_9										d_1	d_2	d_3	3	0
P_1	e_1	e_2	e_3										3	0
P_4				f_1	f_2	f_3							3	0
P_7							g_1	g_2	g_3				3	0
P_{10}										h_1	h_2	h_3	3	0
P_2	i_1	i_2	i_3										3	0
P_5				j_1	j_2	j_3							3	0
P_8							k_1	k_2	k_3				3	0
P_{11}										l_1	l_2	l_3	3	0
	Cluster-1			Cluster-2			Cluster-3						36	0

圖三、重新排序處理器 ID 後的資料通訊示意圖。

針對每組電腦叢集提供不同數量的處理器之問題，可利用此做法，提高資料傳輸效能。

為了適用於多維度的處理器編排系統，我們也提出多維陣列(Multi-Dimensional Array)資料對應模組，希望可以動態調整通訊的瓶頸，提升程式的執行效益。圖四是處理器跟資料通訊的關係，(a)是 2-D 處理器編排系統，可視為多維系統的表示圖，每個 P 皆視為一個處理器，其所佔面積等同於二維陣列中所分配的資料範圍；(b)為資料重新分配時的需產生資料(m00~m22)示意圖，虛線表示二維陣列新的分配方式。



圖四、處理器與資料通訊的關係。(a)多維處理器示意圖；(b)資料通訊示意圖

為了達到資料配置的要求，處理器經常移動資料，而花費的通訊成本過高時會影響執行效能。為此，我們提出了 Local Message Reduction Optimization，改善資料重新配置之排程演算法，並建立效能對照表。重新計算了每筆通訊的權重，評估並對排程了所有的通訊，除了可以有效降低通訊成本，更能避免資料傳輸所產生的衝突。

四、結論與討論

下面我們歸納本計畫主要的成果:

- 完成自動資料分割模組的開發
- 完成平行資料分割效能預測系統的實作。
- 提出重新排程與資料重新分配的技術
- 實作程式階層的效能預測、與其效能監督工具所提供的資訊，進行程式判別。
- 發表三篇國際期刊與五篇國際研討會論文

Journal Papers:

- Ching-Hsien Hsu, Min-Hao Chen, Chao-Tung Yang and Kuan-Ching Li, “Optimizing Communications of Dynamic Data Redistribution on Symmetrical Matrices in Parallelizing Compilers,” ***IEEE Transactions on Parallel and Distributed Systems***, Vol. 17, No. 11, pp. 1226-1241, Nov. 2006. (SCI, EI)
- Ching-Hsien Hsu, Tai-Lung Chen and Kuan-Ching Li, "Performance Effective Pre-scheduling Strategy for Heterogeneous Communication Grid Systems," ***Future Generation Computer Science***, Vol. 23, Issue 4, pp. 569-579, May 2007. Elsevier (SCI, EI)
- Ching-Hsien Hsu, Shih-Chang Chen and Chao-Yang Lan, "Scheduling Contention-Free Irregular Redistribution in Parallelizing Compilers," The ***Journal of Supercomputing***, Kluwer Academic Publisher, Vol. 40, No. 3, pp. 229-247, June 2007. (SCI, EI)
- Ching-Hsien Hsu, Tai-Lung Chen and Jong-Hyuk Park, “On improving resource utilization and system throughput of master slave jobs scheduling in heterogeneous systems,” ***Journal of Supercomputing***, Springer, Vol. 45, No. 1, pp. 129-150, July 2008. (SCI, EI).

Conference Papers:

- Ching-Hsien Hsu, Justin Zhan, Wai-Chi Fang and Jianhua Ma, “Towards Improving QoS-Guided Scheduling in

Grids," IEEE Proceedings of the third ChinaGrid Annual Conference (ChinaGrid 2008), Dunhunag, Gansu, China.

- Ching-Hsien Hsu, Tai-Lung Chen, Bing-Ru Tsai and Kuan-Ching Li, "Scheduling for Atomic Broadcast Operation in Heterogeneous Networks with One Port Model," Proceedings on the 3rd International Conference on Grid and Pervasive Computing (GPC-08), LNCS 5036, pp. 166-177, May 2008.
- Ching-Hsien Hsu, Yi-Min Chen and Chao-Tung Yang, "A Layered Optimization Approach for Redundant Reader Elimination in Wireless RFID Networks," *Proceedings of 2007 IEEE Asia-Pacific Services Computing Conference* (IEEE APSCC 2007), pp. 138-145, Tsukuba, Japan, December 11-14, 2007.
- Ching-Hsien Hsu, Chih-Wei Hsieh and Chao-Tung Yang, "A Generalized Critical Task Anticipation Technique for DAG Scheduling," Algorithm and Architecture for Parallel Processing - *Lecture Notes in Computer Science*, vol. 4494, pp. 493-505, Springer-Verlag, June 2007. (ICA3PP'07)
- Ching-Hsien Hsu, Ming-Yuan Own and Kuan-Ching Li, "Critical-Task Anticipation Scheduling Algorithm for Heterogeneous and Grid Computing," *Computer Systems Architecture - Lecture Notes in Computer Science*, Vol. 4186, pp. 95-108, Springer-Verlag, Sept. 2006. (ACSAC'06) (SCI Expanded, NSC92-2213-E-216-029)

五、計畫成果自評

本計畫之研究成果已達到計畫預期之目標。第三年年的研究中、針對這一個研究主題上共計發表三篇國際期刊與五篇研討會論文。本計畫有目前研究成果，感謝國科會給予機會。未來，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。

六、參考文獻

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

[2] 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.

[3] 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.

[4] Shih-Chang Chen and Ching-Hsien Hsu, "ISO: Comprehensive Techniques Towards Efficient GEN_BLOCK Redistribution with Multidimensional Arrays", *Parallel Computing Technologies* (*PaCT'07*) - *Lecture Notes in Computer Science*, Vol. 4671, pp. 507-515, Springer-Verlag, Sep. 2007.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] Ching-Hsien Hsu, Min-Hao Chen, Chao-Tung Yang and Kuan-Ching Li, " Optimizing Communications of Dynamic Data Redistribution on Symmetrical Matrices in Parallelizing Compilers," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 17, No. 11, pp. 1226-1241, Nov. 2006. (SCI, EI, NSC93-2213-E-216-029, NCHC-KING-010200)

[10] 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, 2007. (SCI, EI, NSC93-2213-E-216-028, NCHC-KING-010200)

[11] Ching-Hsien Hsu, Tai-Lung Chen and Kuan-Ching Li, "Performance Effective

Pre-scheduling Strategy for Heterogeneous Communication Grid Systems," Accepted, Future Generation Computer Science, Elsevier, 2007. (SCI, EI, NSC93-2213-E-216-029)

[12] Ching-Hsien Hsu, Chih-Wei Hsieh and Chao-Tung Yang, "A Generalized Critical Task Anticipation Technique for DAG Scheduling," Algorithm and Architecture for Parallel Processing - *Lecture Notes in Computer Science*, Springer-Verlag, June 2007. (ICA3PP' 07)

[13] Ching-Hsien Hsu, Chao-Yang Lan and Shih-Chang Chen, "Optimizing Scheduling Stability for Runtime Data Alignment," *Embedded System Optimization - Lecture Notes in Computer Science*, Vol. 4097, pp. 825-835, Springer-Verlag, Aug. 2006. (ESO' 06) (SCI Expanded, NSC92-2213-E-216-029)

[14] 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-216-029)

[15] Ching-Hsien Hsu, Ming-Yuan Own and Kuan-Ching Li, "Critical-Task Anticipation Scheduling Algorithm for Heterogeneous and Grid Computing," *Computer Systems Architecture - Lecture Notes in Computer Science*, Vol. 4186, pp. 97-110, Springer-Verlag, Sept. 2006. (ACSAC'06) (SCI Expanded, NSC92-2213-E-216-029)

[16] D.H. Kim, K.W. Kang, "Design and Implementation of Integrated Information System for Monitoring Resources in Grid Computing," *Computer Supported Cooperative Work in Design*, 10th Conf., pp. 1-6, 2006.

[17] K.C. Li, Ching-Hsien Hsu, H.H. Wang and C.T. Yang, "Towards the Development of Visuel: a Novel Application and System Performance Monitoring Toolkit for Cluster and Grid Environments", Accepted, *International Journal of High Performance Computing and Networking* (*IJHPCN*), *Inderscience Publishers*, 2008.

[18] Emmanuel Jeannot and Frédéric Wagner, "Two Fast and Efficient Message Scheduling Algorithms for Data Redistribution through a Backbone," Proceedings of the 18th International Parallel and Distributed Processing Symposium, April 2004.

[19] C. Lee, R. Wolski, I. Foster, C. Kesselman and J. Stepanek, "A Network Performance Tool for Grid Computations," *Supercomputing '99*, 1999.

[20] K.C. Li, H.H. Wang, C.T. Yang and Ching-Hsien Hsu, "Towards the Development of Visuel: a Novel Application and System Performance Monitoring Toolkit for Cluster and Grid Environments," Accepted, International Journal of High Performance Computing and Networking (IJHPCN), Inderscience Publishers, 2007.

[21] 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.

[22] A. Smyk, M. Tudruj, L. Masko, "Open MP Extension for Multithreaded Computing with Dynamic SMP Processor Clusters with Communication on the Fly," *PAR ELEC*, pp. 83-88, 2006.

[23] 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.

[24] M. Tudruj and L. Masko, "Fast Matrix Multiplication in Dynamic SMP Clusters with Communication on the Fly in Systems on Chip Technology," *PAR ELEC*, pp. 77-82, 2006

[25] 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.

[26] Chun-Ching Wang, Shih-Chang Chen, Ching-Hsien Hsu and Chao-Tung Yang, "Optimizing Communications of Data Parallel Programs in Scalable Cluster Systems," *Proceedings of the 3rd International Conference on Grid and Pervasive Computing* (*GPC-08*), LNCS 5036, pp. 29-37, May 2008

[27] C.T. Yang, I-Hsien Yang, Shih-Yu Wang, Ching-Hsien Hsu and Kuan-Ching Li, "A Recursive-Adjustment Co-Allocation Scheme with Cyber-Transformer in Data Grids," Accepted, *Future Generation Computer Science*, Elsevier, 2008.

[28] 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.

<table>
<tr><td colspan="8" align="center">行政院所屬各機關人員出國報告書提要
撰寫時間： 95 年 9 月 11 日</td></tr>
<tr><td>姓 名</td><td>許慶賢</td><td>服務機關名稱</td><td colspan="2">中華大學
資工系</td><td>連絡電話、
電子信箱</td><td colspan="2">03-5186410
chh@chu.edu.tw</td></tr>
<tr><td>出生日期</td><td colspan="2">62 年 2 月 23 日</td><td colspan="2">職 稱</td><td colspan="3">副教授</td></tr>
<tr><td>出席國際會議名稱</td><td colspan="7">Eleventh Asia-Pacific Computer Systems Architecture Conference (ACSAC-06), Shanghai, China</td></tr>
<tr><td>到達國家及地點</td><td colspan="3">ShangHai, China</td><td colspan="2">出國期間</td><td colspan="2">自 95 年 09 月 06 日
迄 95 年 09 月 08 日</td></tr>
</table>

報告內容應包括下列各項：

一、 參加會議經過

這一次在上海所舉行的國際學術研討會議共計三天。第一天上午由 Guang R. Gao 博士針對 The Era of Multi-Core Chips- A Fresh Look on Software Challenges 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 Languages and Compilers 場次聽取報告。本人也在同一天下午發表這一次被大會接受的論文。

第一晚上本人亦參加酒會，並且與幾位國外學者及中國教授交換意見。第二天本人除了在上午參加 Multi-core，Architecture，Networks 場次，也在下午主持了 Power Management 場次，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。第二天晚上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：ILP, TLP, Processor Architecture, Memory System, Operation System, High Performance I/O Architecture 等等熱門的研究課題。

二、 與會心得

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。

三、 考察參觀活動(無是項活動者省略)

四、 建議

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。

五、 攜回資料名稱及內容

1. Conference Program
2. Proceedings

An Efficient Processor Selection Scheme for Master Slave Paradigm on Heterogeneous Networks

Tai-Lung Chen Ching-Hsien Hsu

Department of Computer Science and Information Engineering
Chung Hua University, Hsinchu, Taiwan

chh@chu.edu.tw

Abstract. It is well known that grid technology has the ability to achieve resources shared and tasks scheduled coordinately. In this paper, we present a performance effective pre-scheduling strategy for dispatching tasks onto heterogeneous processors. The main contribution of this study is the consideration of heterogeneous communication overheads in grid systems. One significant improvement of our approach is that average turnaround time could be minimized by selecting processor has the smallest communication ratio first. The other advantage of the proposed method is that system throughput can be increased via dispersing processor idle time. Our proposed technique can be applied to heterogeneous cluster systems as well as computational grid environments, in which the communication costs vary in different clusters. Experimental results show that our techniques outperform other previous algorithms in terms of lower average turnaround time, higher average throughput, less processor idle time and higher processors' utilization.

1 Introduction

Computational grid system integrates geographically distributed computing resources to establish a virtual and high expandable parallel computing infrastructure. In recent years, there are several research investigations done in scheduling problem for 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 in centralized grid system.

The master slave tasking is a simple and widely used technique [1, 2]. In a master slave tasking paradigm, the master node connects to n slave nodes. A set of independent tasks are dispatched by master processor and be processed on the n heterogeneous slave processors. Slave processors execute the tasks accordingly after they receive their tasks. This will restrict that the computation and communication can't overlap. Moreover, 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.

In this paper, an efficient strategy for scheduling independent 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 present lower communication ratio, which will be defined in section 3.2. Improvements of our approach towards both average turnaround time and system throughput.

The remaining of this paper is organized as follows. Section 2 briefly discusses previous related researches, while in section 3 is introduced the research architecture and definition of notation and terminologies used in this paper, where we also present a motivating example to demonstrate the characteristics of the master-slave pre-scheduling model. Section 4 assesses the new scheduling algorithm, the Smallest Communication Ratio (*SCR*), while the illustration of *SCR* on heterogeneous communication is examined in section 5. The performance comparisons and simulations results are discussed in section 6, and finally in section 7, some conclusions of this paper.

2 Related Work

The task scheduling research on heterogeneous processors can be classified into *DAG*s model, master-slave paradigm and computational grids. The main 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. [13] brought up the idea about adaptive scheduling scheme based on homogeneous processor platform, which applies space-sharing and time-sharing to schedule tasks. With the emergence of Grid and ubiquitous computing, new algorithms are in demand to address new concerns arising to grid environments, such as security, quality of service and high system throughput. Berman et al. [6] and Cooper et al. [11] addressed the problem of scheduling incoming applications to available computation resources. Dynamically rescheduling mechanism was introduced to adaptive computing on the Grid. In [8], some simple heuristics for dynamic matching and scheduling of a class of independent tasks onto a heterogeneous computing system have been presented. Moreover, an extended suffrage heuristic was presented in [12] for scheduling the parameter sweep applications that have been implemented in *AppLeS*. They also presented a method to predict the computation time for a task/host pair by using previous host performance.

Chronopoulos et al. [9], Charcranoon et al. [10] and Beaumont et al. [4, 5] introduced the research of master-slave paradigm with heterogeneous processors background. Based on this architecture, Beaumont et al. [1, 2] presented a method on master-slave paradigm to forecast the amount of tasks each processor needs to receive in a given period of time. Beaumont et al. [3] presented the pipelining broadcast method on master-slave platforms, focusing on message passing disregarding computation time. Intuitionally in their implementation, 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 basic concepts and models of this investigation, where we also define notations and terminologies that will be used in subsequent subsections.

3.1 Research Architecture

We have revised several characteristics that were introduced by Beaumont *et al.* [1, 2]. Based on the master slave paradigm introduced in section 1, this paper follows next assumptions as listed.

- 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.
- Heterogeneous communication: communication costs between master and slave processors are of different overheads.

3.2 Definitions

First, we list definitions, notations and terminologies used in this research paper.

Definition 1: In a master slave system, master processor is denoted by M and the n slave processors 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 execution time of each one of slave processors to compute one task are different. We use T_i to represent the execution time of slave processor P_i to complete one task. In this paper, we assume the computation speed of n slave processors is sorted and $T_1 \leq T_2 \leq \ldots \leq T_n$.

Definition 3: Given a master slave system, the time of slave processor P_i to receive one task from master processor is denoted as T_{i_comm}.

Definition 4: A Basic Scheduling Cycle (BSC) is defined as $BSC = lcm(T_1 + T_{1\_comm}, T_2 + T_{2\_comm}, ..., T_m + T_{m\_comm})$, where m is the number of processors that will join the computation.

Definition 5: Given a master slave system, the number of tasks processor P_i needs to receive in a basic scheduling cycle is defined as $task(P_i) = \frac{BSC}{T_i + T_{i\_comm}}$.

Definition 6: Given a master slave system, the communication cost of processor P_i in BSC is defined as $comm(P_i) = T_{i\_comm} \times task(P_i)$.

Definition 7: Given a master slave system, the computation cost of processor P_i in BSC is defined as $comp(P_i) = T_i \times task(P_i)$.

Definition 8: Given a master slave system, the *Communication Ratio* of processor P_i is defined as $CR_i = \frac{T_{i\_comm}}{T_i + T_{i\_comm}}$.

Definition 9: The computational capacity (δ) of a master slave system is defined as the sum of communication ratio of all processors that joined the computation, i.e., $\delta = \sum_{i=1}^{m} CR_i$, where m is the number of processors that involved in the computation.

Definition 10: Given a master slave system with n heterogeneous slave processors, $P_{\max}$ is the processor P_k such that $\max\{k \mid \sum_{i=1}^{k} \frac{T_{i\_comm}}{T_i + T_{i\_comm}} \leq 1\}$, where $1 \leq k \leq n$. i.e. $\sum_{i=1}^{k+1} \frac{T_{i\_comm}}{T_i + T_{i\_comm}} > 1$. We use $P_{\max+1}$ to represent processor P_{k+1}.

3.3 Master Slave Task Scheduling

Discussions on the problem of task scheduling in master slave paradigm will be addressed in two cases, depending on the value of system computational capacity (δ).

As mentioned in section 2, faster processors receive more tasks is an intuitional approach in which tasks are previously allocated to these faster processors, and this method is called Most Jobs First (*MJF*) scheduling algorithm [1, 2]. Fig. 1 shows the pre-scheduling of the *MJF* algorithm. As defined in definition 8, the communication ratio of P_1 to P_4 are $\frac{1}{3}$, $\frac{1}{4}$, $\frac{1}{4}$, and $\frac{1}{6}$, respectively. Because $BSC = 12$, we have $task(P_1)=4$, $task(P_2)=3$, $task(P_3)=3$ and $task(P_4)=2$. When the number of tasks is numerous, such scheduling achieves higher system utilization and less processor idle time than the greedy method.

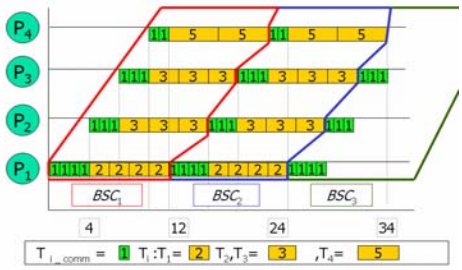


Fig. 1. Most Jobs First (*MJF*) task scheduling when $\delta \leq 1$.

<u>**Lemma 1**</u>: Given a master slave system with $\delta > 1$, in *MJF* scheduling, the amount of tasks being assigned to $P_{\max+1}$ can be calculated by the following equation,

$$task(P_{\max+1}) = (BSC - \sum_{i=1}^{\max} comm(P_i)) \ / \ T_{\max+1\_com} \tag{1}$$

<u>**Lemma 2**</u>: Given a master slave system with $\delta > 1$, in *MJF* scheduling, the period of processor $P_{\max+1}$ stays idle denoted by T_{idle}^{MJF} and can be calculated by the following equation,

$$T_{idle}^{MJF} = BSC - comm(P_{\max+1}) - comp(P_{\max+1}) \tag{2}$$

Another example of master slave task scheduling with identical communication (i.e., T_{i_comm}=1) and $\delta > 1$ is given in Fig. 2. Because $\delta > 1$, according to equation (1), we have $task(P_{\max+1}=P_4) = 10$. We note that P_4 completes its tasks and becomes available at time 100. However, the master processor dispatches tasks to P_3 during time 100 ~ 110 and starts to send tasks to P_4 at time 110. Such kind of idle situation also happens at time 100~110, 160~170, 220~230, and so on.

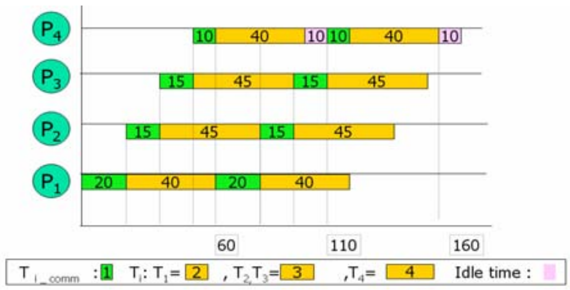


Fig. 2. Most Jobs First (*MJF*) Tasking when $\delta > 1$.

<u>**Lemma 3**</u>: In *MJF* scheduling algorithm with identical communication T_{i_comm}, when $\delta > 1$, the completion time of tasks in the j^{th} *BSC* can be calculated by the following equation.

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

4 Smallest Communication Ratio (*SCR*) Scheduling with Identical Communication

The *MJF* scheduling algorithm distributes tasks to slave processors according to processors' speed, namely, faster processor receives tasks first. In this section, we demonstrate an efficient task scheduling algorithm, Smallest Communication Ratio (*SCR*), focuses on master slave task scheduling with identical communication.

Lemma 4: In *SCR* scheduling algorithm, if $\delta \le 1$ and T_{i_comm} are identical, the task completion time of the j^{th} *BSC* denoted by $T_{finish}^{SCR}(BSC_j)$, can be calculated by the following equation.

$$T_{finish}^{SCR}(BSC_j) = BSC + j \times (comm(P_1) + comp(P_1)) - comm(P_1) \tag{4}$$

Lemma 5: Given a master slave system with $\delta > 1$, in scheduling, the amount of tasks being assigned to $P_{\max+1}$ can be calculated by the following,

$$task(P_{\max+1}) = \frac{BSC}{T_{\max+1} + T_{\max+1\_comm}} \tag{5}$$

Lemma 6: In *SCR* scheduling algorithm, when $\delta > 1$, the idle time of a slave processor is denoted as T_{idle}^{SCR} and can be calculated by the following equation,

$$T_{idle}^{SCR} = \sum_{i=1}^{\max+1} comm(P_i) - BSC \tag{6}$$

The other case in Fig. 3 is to demonstrate the *SCR* scheduling method with dispersive idle when $\delta > 1$. We use the same example in Fig. 2 for the following illustration. Because $\delta > 1$, according to definition 10 and Lemma 5, we have $task(P_{\max+1}=P_4) = 12$. Comparing to the example in Fig. 2, P_4 stays 10 time units idle in *MJF* algorithm while the idle time is reduced and dispersed in *SCR* algorithm. In *SCR*, 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, showing a throughput of 0.6. While in *SCR*, there are 62 tasks completed during 102 time units. The throughput of *SCR* is 62/102 (≈0.61) > 0.6. Consequently, the *SCR* algorithm delivers higher system throughput.

Lemma 7: In *SCR* scheduling algorithm, if T_{i_comm} are identical for all slave processors and $\delta > 1$, the task completion time of the j^{th} *BSC* denoted by $T_{finish}^{SCR}(BSC_j)$, can be calculated by the following equation,

$$T_{finish}^{SCR}(BSC_j) = \sum_{i=1}^{\max+1} comm(P_i) + comp(P_1) + (j-1) \times (comm(P_1) + comp(P_1) + T_{idle}^{SCR}) \tag{7}$$

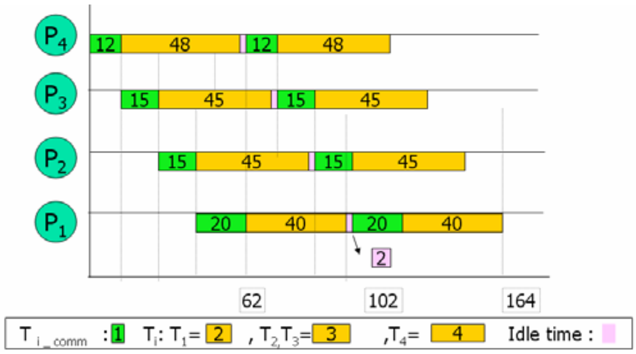


Fig. 3. Smallest Communication Ratio (*SCR*) Tasking when $\delta > 1$.

5 Generalized Smallest Communication Ratio (*SCR*)

As computational grid integrates geographically distributed computing resources, the communication overheads from resource broker / master computer to different computing site are different. Therefore, towards an efficient scheduling algorithm, the heterogeneous communication overheads should be considered. In this section, we present the *SCR* task scheduling techniques work on master slave computing paradigm with heterogeneous communication.

Lemma 8: Given a master slave system with heterogeneous communication and $\delta > 1$, in *MJF* scheduling, we have

$$task(P_{\max+1}) = \left\lfloor \frac{BSC - \sum_{i=1}^{\max} comm(P_i)}{T_{\max+1\_comm}} \right\rfloor \tag{8}$$

Lemma 9: Given an *SCR* scheduling with heterogeneous communication and $\delta > 1$, T_{idle}^{SCR} is the idle time of one slave processor, we have the following equation,

$$T_{idle}^{SCR} = \sum_{i=1}^{\max+1} comm(P_i) - BSC. \tag{9}$$

Lemma 10: Given an *SCR* scheduling with heterogeneous communication and $\delta > 1$, $T_{start}^{SCR}(BSC_j)$ is the start time to dispatch tasks in the j^{th} *BSC*, we have the following equation,

$$T_{start}^{SCR}(BSC_j) = (j-1) \times (BSC + T_{idle}^{SCR}) \tag{10}$$

Lemma 11: Given an *SCR* scheduling with heterogeneous communication and $\delta > 1$, the task completion time of the j^{th} *BSC* denoted by $T_{finish}^{SCR}(BSC_j)$, we have

$$T_{finish}^{SCR}(BSC_j) = \sum_{i=1}^{\max+1} comm(P_i) + comp(P_k) + (j-1) \times (comm(P_k) + comp(P_k) + T_{idle}^{SCR}) \tag{11}$$

where P_k is the slave processor with maximum communication cost.

Another example of heterogeneous of communication with $\delta > 1$ master slave tasking is shown in Fig. 4(a). The communication overheads vary from 1 to 5. The computational speeds vary from 3 to 13. In this example, we have *BSC* = 48.

In *SCR* implementation, according to corollary 3, task distribution is $task(P_1) = 6$, $task(P_2) = 6$, $task(P_3) = 4$ and $task(P_{\max+1}) = task(P_4) = 3$. The communication costs of slave processors are $comm(P_1) = 30$, $comm(P_2) = 12$, $comm(P_3) = 4$ and $comm(P_4) = 9$, respectively. Therefore, the *SCR* method distributes tasks by the order P_3, P_4, P_2, P_1. There are 19 tasks in the first *BSC* dispatched to P_1 to P_4 during time period 1~55. Processor P_3 is the first processor to receive tasks and it finishes at time $t = 48$ and becomes available. In the meanwhile, processor P_1 receives tasks during $t = 48$~55. The second *BSC* starts to dispatch tasks at $t = 55$. Namely, P_3 starts to receive tasks at $t = 55$ in the second scheduling cycle. Therefore, P_3 has 7 unit of time idle. Lemmas 4 and 5 state the above phenomenon. The completion time of tasks in the first *BSC* depends on the finish time of processor P_1. We have $T_{finish}^{SCR}(BSC_1) = 73$.

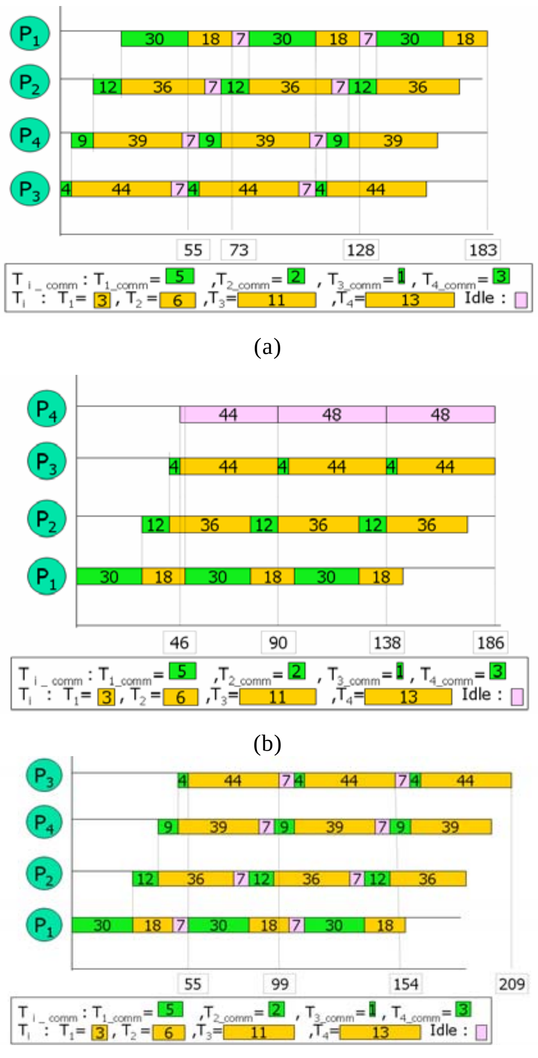


Fig. 4. Task scheduling on heterogeneous communication environment with $\delta > 1$. (a) Smallest Communication Ratio (b) Most Job First (c) Largest communication ratio *(LCR).*

The *MJF* scheduling is depicted in Fig. 4(b). According to corollary 5, $task(P_{max+1}) = task(P_4) = 0$, therefore, P_4 will not be included in the scheduling. MJF has the task distribution order P_1, P_2, P_3. Another scheduling policy is called *Longest Communication Ratio* (*LCR*) which is an opposite approach to the *SCR* method. Fig. 4(c) shows the *LCR* scheduling result which has the dispatch order P_1, P_2, P_4, P_3.

To investigate the performance of *SCR* scheduling technique, we observe that *MJF* algorithm completes 16 tasks in 90 units of time in the first *BSC*. On the other hand, in *SCR* scheduling, there are 19 tasks completed in 73 units of time in the first BSC. In *LCR*, there are 19 tasks completed in 99 units of time. We can see that the system throughput of *SCR* (19/73≈0.260) > *LCR* (19/99≈0.192) > *MJF* (16/90≈0.178). Moreover, the average turnaround time of the *SCR* algorithm in the first three *BSC*s is 183/57 (≈3.2105) which is less than the *LCR's* average turnaround time 209/57 (≈3.6666) and the *MJF*'s average turnaround time 186/48 (≈3.875).

6 Performance Evaluation

To evaluate the performance of the proposed method, we have implemented the *SCR* and the *MJF* algorithms. We compare different criteria, such as average turnaround time, system throughput and processor idle time, in Heterogeneous Processors with Heterogeneous Communications (*HPHC*).

Simulation experiments for evaluating average turnaround time are made upon different number of processors and show in Fig. 7. The computational speed of slave processors is set as T_1=3, T_2=3, T_3=5, T_4=7, T_5=11, and T_6=13. For the cases when processor number is 2, 3… 6, we have $\delta \le 1$. When processor number increases to 7, we have $\delta > 1$. In either case, the *SCR* algorithm conduces better average turnaround time. From the above results, we conclude that the *SCR* algorithm outperforms *MJF* for most test samples.

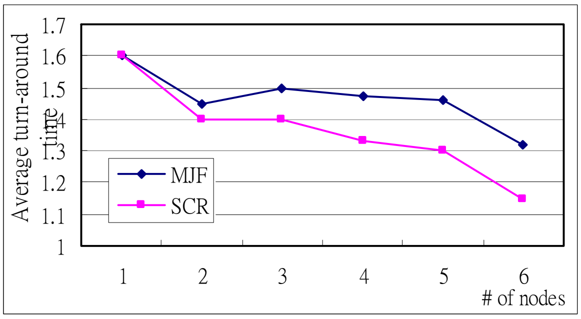
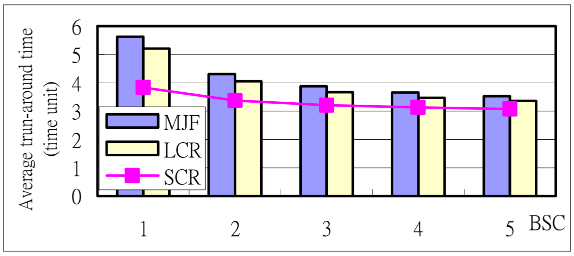
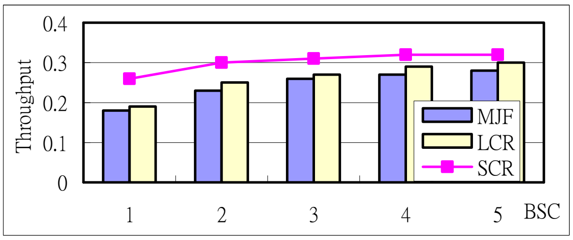


Fig. 5. Average task turn-around time on different numbers of processors.

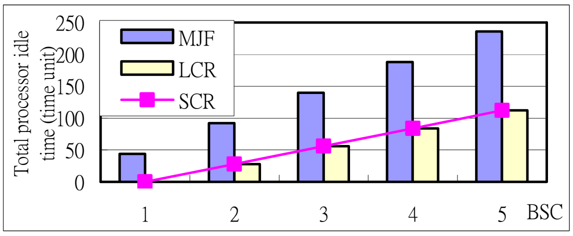
Simulation results present the performance comparison of three task scheduling algorithms, *SCR*, *MJF*, *LCR*, on heterogeneous processors and heterogeneous communication paradigms. Fig. 6 shows the simulation results for the experiment setting that with ±10 processor speed variation and ±4 communication speed variation. The computation speed of slave processors are T_1=3, T_2=6, T_3=11, and T_4=13. The time of a slave processor to receive one task from master processor are T_{1_comm} = 5, T_{2_comm} = 2, T_{3_comm} = 1 and T_{4_comm}=3. The average task turnaround time, system throughput and processor idle time are measured.



(a)



(b)

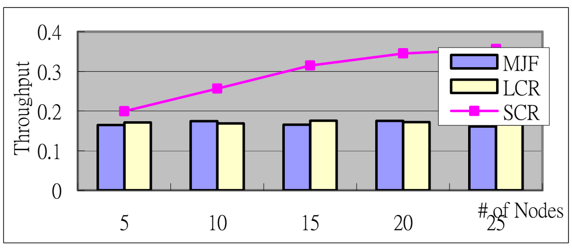


(c)

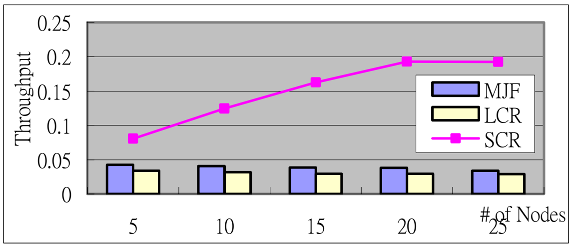
Fig. 6. Simulation results for 5 processors with ±10 computation speed variation and ±4 communication variation when $\delta > 1$ (a) average turnaround time (b) system throughput (c) processor idle time.

Fig. 6(a) is the average turnaround time within different number of *BSC*. The *SCR* algorithm performs better than the *LCR* and *MJF* method. Similarly, the *SCR* method has higher throughput than the other two algorithms as shown in Fig. 6(b). The processor idle time are estimated in Fig. 6(c). The *SCR* and *LCR* algorithms have the same period of processor idle time which is less than the *MJF* scheduling method. These phenomena match the theoretical analysis in section 5.

The miscellaneous comparison in Fig. 7 presents the performance comparison of *SCR*, *MJF* with more cases. The simulation results for the experiment setting that with ±5~±30 processor speed variation and ±5~±30 communication speed variation. The computation speed variation of $T_1 \sim T_n$=±5~±30. The communication speed variation of $T_{1\_comm} \sim T_{n\_comm}$=±5~±30. The system throughput is measured.



(a)



(b)

Fig. 7. Simulation results of throughput for the range of 5~25 processors with ±30 computation speed variation and ±30 communication variation in 100 cases and 100 *BSC* (a) system throughput of the cases when $0<T_i \le 30$ and $0<T_{i\_comm} \le 5$ (b) system throughput of the cases when $0<T_i \le 5$ and $0<T_{i\_comm} \le 30$.

Fig. 7(a) is the case of $0<T_i \le 30$, $0<T_{i\_comm} \le 5$ and the parameter of computation speed and communication speed are to be random and uniformly distributed within different number of nodes and 100 *BSC* for 100 cases. Fig. 7(b) is the case of $0<T_i \le 5$ and $0<T_{i\_comm} \le 30$. The *SCR* algorithm performs better than *MJF* method, and *SCR* method has higher throughput than the *MJF* algorithm as shown in Fig. 7(a) and Fig. 7(b). From the above experimental tests, we have the following remarks. The proposed *SCR* scheduling technique has better task turnaround time and higher system throughput than the *MJF* algorithm.

From the above experimental tests, we have the following remarks.

- The proposed *SCR* scheduling technique has higher system throughput than the *MJF* algorithm.
- The proposed *SCR* scheduling technique has better task turnaround time than the *MJF* algorithm.

The *SCR* scheduling technique has less processor idle time than the *MJF* algorithm.

7 Conclusions

The problem of resource management and scheduling has been one of main challenges in grid computing. In this paper, we have presented an efficient algorithm, *SCR* for heterogeneous processors tasking problem. One significant improvement of our approach is that average turnaround time could be minimized by selecting processor has the smallest communication ratio first. The other advantage of the proposed method is that system throughput can be increased via dispersing processor idle time. Our preliminary analysis and simulation results indicate that the *SCR* 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 that remains in this paper. Our proposed model can be applied to map tasks onto heterogeneous cluster systems in grid environments, in which the communication costs are various from clusters. In future, we intend to devote generalized tasking mechanisms for computational grid. We will study realistic applications and analyze their performance on grid system. Besides, rescheduling of processors / tasks for minimizing processor idle time on heterogeneous systems is also interesting and will be investigated.

References

1. O. Beaumont, A. Legrand and Y. Robert, "The Master-Slave Paradigm with Heterogeneous Processors," *IEEE Trans. on parallel and distributed systems*, Vol. 14, No.9, pp. 897-908, September 2003.
2. C. Banino, O. Beaumont, L. Carter, J. Ferrante, A. Legrand and Y. 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.
3. O. Beaumont, A. Legrand and Y. Robert, "Pipelining Broadcasts on Heterogeneous Platforms," *IEEE Trans. on parallel and distributed systems*, Vol. 16, No.4, pp. 300-313 April 2005.
4. 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.

5. O. Beaumont, V. Boudet, F. Rastello and Y. Robert, “Matrix-Matrix Multiplication on Heterogeneous Platforms,” *Proc. Int'l Conf. Parallel Processing*, Vol. 12, No. 10, pp. 1033-1051, Oct. 2001.
6. F. Berman, R. Wolski, H. Casanova, W. Cirne, H. Dail, M. Faerman, S. Figueira, J. Hayes, G. Obertelli, J. Schopf, G. Shao, S. Smallen, N. Spring, A. Su, and D. Zagorodnov, ”Adaptive Computing on the Grid Using AppLeS,” *IEEE Trans. on parallel and distributed systems*, Vol. 14, No. 4, pp.369-379, April 2003.
7. 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.
8. 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.
9. 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.
10. 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.
11. K. Cooper, A. Dasgupta, K. Kennedy, C. Koelbel, A. Mandal, G. Marin, M. Mazina, J. Mellor-Crummey, F. Berman, H. Casanova, A. Chien, H. Dail, X. Liu, A. Olugbile, O. Sievert, H. Xia, L. Johnsson, B. Liu, M. Patel, D. Reed, W. Deng, C. Mendes, Z. Shi, A. YarKhan, J. Dongarra, ”New Grid Scheduling and Rescheduling Methods in the GrADS Project,” *Proceedings of the 18th International Parallel and Distributed Processing Symposium* (IPDPS’04), pp.209-229, April 2004.
12. H. Casanova, A. Legrand, D. Zagorodnov and F. Berman, “Heuristics for Scheduling Parameter Sweep applications in Grid environments,” *Proceedings of the 9th Heterogeneous Computing workshop* (*HCW*'2000), pp. 349-363, 2000.
13. T. Thanalapati and S. 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.

行政院所屬各機關人員出國報告書提要

撰寫時間： 96 年 6 月 20 日

<table>
<tr><td>姓名</td><td>許慶賢</td><td>服務機關名稱</td><td>中華大學資工系</td><td>連絡電話、電子信箱</td><td colspan="2">03-5186410
chh@chu.edu.tw</td></tr>
<tr><td>出生日期</td><td colspan="3">62 年 2 月 23 日</td><td>職稱</td><td colspan="2">副教授</td></tr>
<tr><td>出席國際會議名稱</td><td colspan="6">2007 International Conference on Algorithms and Architecture for Parallel Processing, June 11 -14 2007.</td></tr>
<tr><td>到達國家及地點</td><td colspan="3">Hangzhou, China</td><td>出國期間</td><td colspan="2">自 96 年 06 月 11 日
迄 96 年 06 月 19 日</td></tr>
<tr><td>內容提要</td><td colspan="6">這一次在杭州所舉行的國際學術研討會議共計四天。第一天下午本人抵達會場辦理報到。第二天各主持一場 invited session 的論文發表。同時，自己也在上午的場次發表了這依次被大會接受的論文。第一天也聽取了 Dr. Byeongho Kang 有關於 Web Information Management 精闢的演說。第二天許多重要的研究成果分為六個平行的場次進行論文發表。本人選擇了 Architecture and Infrastructure、Grid computing、以及 P2P computing 相關場次聽取報告。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，合影留念。第三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，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。</td></tr>
<tr><td>出席人所屬機關審核意見</td><td colspan="6"></td></tr>
<tr><td>層轉機關審核意見</td><td colspan="6"></td></tr>
<tr><td>研考會處理意見</td><td colspan="6"></td></tr>
</table>

(出席 ICA3PP-07 研討會所發表之論文)

A Generalized Critical Task Anticipation Technique for DAG Scheduling

Ching-Hsien Hsu[1], Chih-Wei Hsieh[1] and Chao-Tung Yang[2]

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

[2] High-Performance Computing Laboratory
Department of Computer Science and Information Engineering
Tunghai University, Taichung City, 40704, Taiwan R.O.C.
ctyang@thu.edu.tw

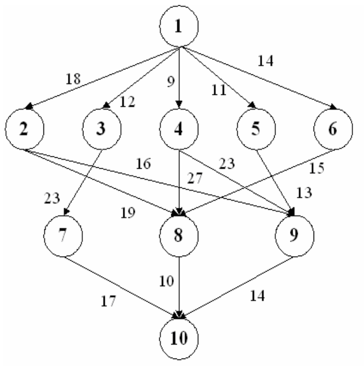
Abstract. The problem of scheduling a weighted directed acyclic graph (DAG) representing an application to a set of heterogeneous processors to minimize the completion time has been recently studied. The NP-completeness of the problem has instigated researchers to propose different heuristic algorithms. In this paper, we present a Generalized Critical-task Anticipation (*GCA*) algorithm for DAG scheduling in heterogeneous computing environment. The *GCA* scheduling algorithm employs task prioritizing technique based on *CA* algorithm and introduces a new processor selection scheme by considering heterogeneous communication costs among processors for adapting grid and scalable computing. To evaluate the performance of the proposed technique, we have developed a simulator that contains a parametric graph generator for generating weighted directed acyclic graphs with various characteristics. We have implemented the *GCA* algorithm along with the *CA* and *HEFT* scheduling algorithms on the simulator. The *GCA* algorithm is shown to be effective in terms of speedup and low scheduling costs.

1. Introduction

The purpose of heterogeneous computing system is to drive processors cooperation to get the application done quickly. Because of diverse quality among processors or some special requirements, like exclusive function, memory access speed, or the customize I/O devices, etc.; tasks might have distinct execution time on different resources. Therefore, efficient task scheduling is important for achieving good performance in heterogeneous systems.

The primary scheduling methods can be classified into three categories, dynamic scheduling, static scheduling and hybrid scheduling according to the time at which the scheduling decision is made. In dynamic approach, the system performs redistribution of tasks between processors during run-time, expect to balance computational load, and reduce processor’s idle time. On the contrary, in static approach, information of applications, such as tasks execution time, message size of communications among tasks, and tasks dependences are known a priori at compile-time; tasks are assigned to processors accordingly in order to minimize the entire application completion time and satisfy the precedence of tasks. Hybrid scheduling techniques are mix of dynamic and static methods, where some preprocessing is done statically to guide the dynamic scheduler [8].

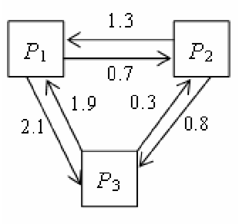
A Direct Acyclic Graph (DAG) [2] is usually used for modeling parallel applications that consists a number of tasks. The nodes of DAG correspond to tasks and the edges of which indicate the precedence constraints between tasks. In addition, the weight of an edge represents communication cost between tasks. Each node is given a computation cost to be performed on a processor and is represented by a computation costs matrix. Figure 1 shows an example of the model of DAG scheduling. In Figure 1(a), it is assumed that task n_j is a successor (predecessor) of task n_i if there exists an edge from n_i to n_j (from n_j to n_i) in the graph. Upon task precedence constraint, only if the predecessor n_i completes its execution and then its successor n_j receives the *messages* from n_i, the successor n_j can start its execution. Figure 1(b) demonstrates different computation costs of task that performed on heterogeneous processors. It is also assumed that tasks can be executed only on single processor with non-preemptable style. A simple fully connected processor network with asymmetrical data transfer rate is shown in Figures 1(c) and 1(d).



(a)

	P_1	P_2	P_3	$\overline{w_i}$
n_1	14	19	9	14
n_2	13	19	18	16.7
n_3	11	17	15	14.3
n_4	13	8	18	13
n_5	12	13	10	11.7
n_6	12	19	13	14.7
n_7	7	16	11	11
n_8	5	11	14	10
n_9	18	12	20	16.7
n_{10}	17	20	11	16

(b)



(c)

SP \ DP	P_1	P_2	P_3
P_1	0	0.7	2.1
P_2	1.3	0	0.8
P_3	1.9	0.3	0

(d)

Figure 1: An example of DAG scheduling problem (a) Directed Acyclic Graph (DAG-1) (b) computation cost matrix (*W*) (c) processor topology (d) communication weight.

The scheduling problem has been widely studied in heterogeneous systems where the computational ability of processors is different and the processors communicate over an underlying network. Many researches have been proposed in the literature. The scheduling problem has been shown to be NP-complete [3] in general cases as well as in several restricted cases; so the desire of optimal scheduling shall lead to higher scheduling overhead. The negative result motivates the requirement for heuristic approaches to solve the scheduling problem. A comprehensive survey about static scheduling algorithms is given in [9]. The authors of have shown that the heuristic-based algorithms can be classified into a variety of categories, such as clustering algorithms, duplication-based algorithms, and list-scheduling algorithms. Due to page limitation, we omit the description for related works.

In this paper, we present a Generalized Critical task Anticipation (*GCA*) algorithm, which is an approach of list scheduling for DAG task scheduling problem. The main contribution of this paper is proposing a novel heuristic for DAG scheduling on heterogeneous machines and networks. A significant improvement is that inter-processor communication costs are considered into processor selection phase such that tasks can be mapped to more suitable processors. The *GCA* heuristic is compared favorable with previous *CA* [5] and *HEFT* heuristics in terms of schedule length and speedup under different parameters.

The rest of this paper is organized as follows: Section 2 provides some background, describes preliminaries regarding heterogeneous scheduling system in DAG model and formalizes the research problem. Section 3 defines notations and terminologies used in this paper. Section 4 forms the main body of the paper, presents the Generalized Critical task Anticipation (*GCA*) scheduling algorithm and illustrating it with an example. Section 5 discusses performance of the proposed heuristic and its simulation results. Finally, Section 6 briefly concludes this paper.

2. DAG Scheduling on Heterogeneous Systems

The DAG scheduling problem studied in this paper is formalized as follows. Given a parallel application represented by a DAG, in which nodes represent tasks and edges represent dependence between these tasks. The target computing architecture of DAG scheduling problem is a set of heterogeneous processors, $M = \{P_k: k = 1: P\}$ and $P = |M|$, communicate over an underlying network which is assumed fully connected. We have the following assumptions:

- Inter-processor communications are performed without network contention between arbitrary processors.
- Computation of tasks is in non-preemptive style. Namely, once a task is assigned to a processor and starts its execution, it will not be interrupted until its completion.
- Computation and communication can be worked simultaneously because of the separated I/0.
- If two tasks are assigned to the same processor, the communication cost between the two tasks can be discarded.
- A processor is assumed to send the computational results of tasks to their immediate successor as soon as it completes the computation.

Given a DAG scheduling system, W is an $n \times P$ matrix in which $w_{i,j}$ indicates estimated computation time of processor P_j to execute task n_i. The mean execution time of task n_i can be calculated by the following equation:

$$\overline{w_i} = \sum_{j=1}^{P} \frac{w_{i,j}}{P} \qquad \textbf{(1)}$$

Example of the mean execution time can be referred to Figure 1(b).

For communication part, a $P \times P$ matrix T is structured to represent different data transfer rate among processors (Figure 1(d) demonstrates the example). The communication cost of transferring data from task n_i (execute on processor p_x) to task n_j (execute on processor p_y) is denoted by $c_{i,j}$ and can be calculated by the following equation,

$$c_{i,j} = V_m + Msg_{i,j} \times t_{x,y}\ , \qquad (2)$$

Where:
V_m is the communication latency of processor P_m,
$Msg_{i,j}$ is the size of message from task n_i to task n_j,
$t_{x,y}$ is data transfer rate from processor p_x to processor p_y, $1 \le x, y \le P$.

In static DAG scheduling problem, it was usually to consider processors' latency together with its data transfer rate. Therefore, equation (2) can be simplified as follows,

$$c_{i,j} = Msg_{i,j} \times t_{x,y}\ , \qquad (3)$$

Given an application represented by Directed Acyclic Graph (DAG), G = (V, E), where $V = \{n_j: j = 1: v\}$ is the set of nodes and $v = |V|$; $E = \{e_{i,j} = \langle n_i, n_j \rangle\}$ is the set of communication edges and $e = |E|$. In this model, each node indicates least indivisible task. Namely, each node must be executed on a processor from the start to its completion. Edge $\langle n_i, n_j \rangle$ denotes precedence of tasks n_i and n_j. In other words, task n_i is the immediate predecessor of task n_j and task n_j is the immediate successor of task n_i. Such precedence represents that task n_j can be start for execution only upon the completion of task n_i. Meanwhile, task n_j should receive essential message from n_i for its execution. Weight of edge $\langle n_i, n_j \rangle$ indicates the average communication cost between n_i and n_j.

Node without any inward edge is called *entry node*, denoted by n_{entry}; while node without any outward edge is called *exit node*, denoted by n_{exit}. In general, it is supposed that the application has only one *entry node* and one *exit node*. If the actual application claims more than one *entry* (*exit*) *node*, we can insert a dummy *entry* (*exit*) *node* with zero-cost edge.

3. Preliminaries

This study concentrates on list scheduling approaches in DAG model. List scheduling was usually distinguished into list phase and processor selection phase. Therefore, priori to discuss the main content, we first define some notations and terminologies used in both phases in this section.

3.1 Parameters for List Phase

Definition 1: Given a DAG scheduling system on $G = (V, E)$, the *Critical Score* of task n_i denoted by $CS(n_i)$ is an accumulative value that are computed recursively traverses along the graph upward, starting from the exit node. $CS(n_i)$ is computed by the following equations,

$$CS(n_i) = \begin{cases} \overline{w_{exit}} & \text{if } n_i \text{ is the exit ndoe (i.e. } n_i = n_{exit}) \\ \overline{w_i} + \underset{n_j \in suc(n_i)}{Max}(\overline{c_{i,j}} + CS(n_j)) & \text{otherwise} \end{cases} \quad (4)$$

where $\overline{w_{exit}}$ is the average computation cost of task n_{exit}, $\overline{w_i}$ is the average computation cost of task n_i, $suc(n_i)$ is the set of immediate successors of task n_i, $\overline{c_{i,j}}$ is the average communication cost of edge $<n_i, n_j>$ which is defined as follows,

$$\overline{c_{i,j}} = \frac{Msg_{i,j} \times \sum_{1 \le x,y \le P} t_{x,y}}{(P^2 - P)}, \quad (5)$$

3.2 Parameters for Processor Selection Phase

Most algorithms in processor selection phase employ a partial schedule scheme to minimize overall schedule length of an application. To achieve the partial optimization, an intuitional method is to evaluate the *finish time* (*FT*) of task n_i executed on different processors. According to the calculated results, one can select the processor who has minimum finish time as target processor to execute the task n_i. In such approach, each processor P_k will maintain a list of tasks, *task-list*(P_k), keeps the latest status of tasks correspond to the $EFT(n_i, P_k)$, the earliest finish time of task n_i that is assigned on processor P_k.

Recall having been mentioned above that the application represented by DAG must satisfy the precedence relationship. Taking into account the precedence of tasks in DAG, a task n_j can start to execute on a processor P_k only if its all immediate predecessors send the essential messages to n_j and n_j successful receives all these messages. Thus, the latest message arrive time of node n_j on processor P_k, denoted by $LMAT(n_j, P_k)$, is calculated by the following equation,

$$LMAT(n_j, P_k) = \underset{n_i \in pred(n_j)}{Max}(EFT(n_i) + c_{u,k}, \text{ for task } n_i \text{ executed on processor } P_u) \quad (6)$$

where $pred(n_j)$ is the set of immediate predecessors of task n_j. Note that if tasks n_i and n_j are assigned to the same processor, $c_{u,k}$ is assumed to be zero because it is negligible.

Because the entry task n_{entry} has no inward edge, thus we have

$$LMAT(n_{entry}, P_k) = 0 \quad (7)$$

for all $k = 1$ to P.

Definition 2: Given a DAG scheduling system on $G = (V, E)$, the *Start Time* of task n_j executed on processor P_k is denoted as $ST(n_j, P_k)$.

Estimating task's start time (for example, task n_j) will facilitate search of available time slot on target processors that is large enough to execute that task (i.e., length of time slot > $w_{j,k}$). Note that the search of available time slot is started from $LMAT(n_j, P_k)$.

Definition 3: Given a DAG scheduling system on $G = (V, E)$, the *finish time* of task n_j denoted by $FT(n_j, P_k)$, represents the completion time of task n_j executed on processor

P_k. $FT(n_j, P_k)$ is defined as follows,

$$FT(n_j, P_k) = ST(n_j, P_k) + w_{j,k} \tag{8}$$

Definition 4: Given a DAG scheduling system on $G = (V, E)$, the *earliest finish time* of task n_j denoted by $EFT(n_j)$, is formulated as follows,

$$EFT(n_j) = \underset{p_k \in P}{Min}\{FT(n_j, P_k)\} \tag{9}$$

Definition 5: Based on the determination of $EFT(n_j)$ in equation (9), if the earliest finish time of task n_j is obtained upon task n_j executed on processor p_t, then the target processor of task n_j is denoted by $TP(n_j)$, and $TP(n_j) = p_t$.

4. The Generalized Critical-task Anticipation Scheduling Algorithm

Our approach takes advantages of list scheduling in lower algorithmic complexity and superior scheduling performance and furthermore came up with a novel heuristic algorithm, the generalized critical task anticipation (*GCA*) scheduling algorithm to improve the schedule length as well as speedup of applications. The proposed scheduling algorithm will be verified beneficial for the readers while we delineate a sequence of the algorithm and show some example scenarios in three phases, prioritizing phase, listing phase and processor selection phase.

In prioritizing phase, the $CS(n_i)$ is known as the maximal summation of scores including the average computation cost and communication cost from task n_i to the exit task. Therefore, the magnitude of the task's critical score is regarded as the decisive factor when determining the priority of a task. In listing phase, an ordered list of tasks should be determined for the subsequent phase of processor selection. The proposed *GCA* scheduling technique arranges tasks into a list *L*, not only according to critical scores but also considers tasks' importance.

Several observations bring the idea of *GCA* scheduling method. Because of processor heterogeneity, there exist variations in execution cost from processor to processor for same task. In such circumstance, tasks with larger computational cost should be assigned higher priority. This observation aids some critical tasks to be executed earlier and enhances probability of tasks reduce its finish time. Furthermore, each task has to receive the essential messages from its immediate predecessors. In other words, a task will be in waiting state when it does not collect complete message yet. For this reason, we emphasize the importance of the last arrival message such that the succeeding task can start its execution earlier. Therefore, it is imperative to give the predecessor who sends the last arrival message higher priority. This can aid the succeeding task to get chance to advance the start time. On the other hand, if a task n_i is inserted into the front of a scheduling list, it occupies vantage position. Namely, n_i has higher probability to accelerate its execution and consequently the start time of $suc(n_i)$ can be advanced as well.

In most list scheduling approaches, it was usually to demonstrate the algorithms in two phases, the list phase and the processor selection phase. The list phase of proposed *GCA* scheduling algorithm consists of two steps, the *CS* (critical score) calculation step and task prioritization step.

Let's take examples for the demonstration of *CS* calculation, which is performed in level order and started from the deepest level, i.e., the level of exit task. For example, according to equation (4), we have $CS(n_{10}) = \overline{w_{10}} = 16$. For the upper level tasks, n_7, n_8 and n_9, $CS(n_7)$ = $\overline{w_7} + (\overline{c_{7,10}} + CS(n_{10}))$ = 47.12, $CS(n_8)$ = $\overline{w_8} + (\overline{c_{8,10}} + CS(n_{10}))$ =37.83, $CS(n_9)$ = $\overline{w_9} + (\overline{c_{9,10}} + CS(n_{10}))$ =49.23. The other tasks can be calculated by the same methods. Table 1 shows complete calculated critical scores of all tasks for DAG-1.

Table 1: Critical Scores of tasks in DAG-1 using *GCA* algorithm

***Critical Scores* of tasks in *GCA* algorithm**									
n_1	n_2	n_3	n_4	n_5	n_6	n_7	n_8	n_9	n_{10}
120.13	84.83	88.67	89.45	76.28	70.25	47.12	37.83	49.23	16.00

Follows the critical score calculation, the *GCA* scheduling method considers both tasks' importance (i.e., critical score) and its relative urgency for prioritizing tasks. Based on the results obtained previously, we use the same example to demonstrate task prioritization in *GCA*. Let's start at the exit task n_{10}, which has the lowest critical score. Assume that tasks will be arranged into an ordered list L, therefore, we have L = {n_{10}} initially. Because task n_{10} has three immediate predecessors, with the order $CS(n_9) > CS(n_7) > CS(n_8)$, the list L will be updated to L={n_9, n_7, n_8, n_{10}}. Applying the same prioritizing method by taking the front element of L, task n_9; because task n_9 has three immediate predecessors, with the order $CS(n_4) > CS(n_2) > CS(n_5)$, we have the updated list L = { n_4, n_2, n_5, n_9, n_7, n_8, n_{10}}. Taking the same operations, insert task n_1 in front of task n_4, insert task n_3 in front of task n_7, insert tasks n_4, n_2, n_6 (because $CS(n_4) > CS(n_2) > CS(n_6)$) in front of task n_8; we have the list L = { n_1, n_4, n_2, n_5, n_9, n_3, n_7, n_6, n_4, n_2, n_6, n_8, n_{10}}. The final list L = {n_1, n_4, n_2, n_5, n_9, n_3, n_7, n_6, n_8, n_{10}} can be derived by removing duplicated tasks.

In listing phases, the *GCA* scheduling algorithm proposes two enhancements from the majority of literatures. First, *GCA* scheduling technique considers various transmission costs of messages among processors into the calculation of critical scores. Second, the *GCA* algorithm prioritizes tasks according to the influence on its successors and devotes to lead an accelerated chain while other techniques simply schedule high critical score tasks with higher priority. In other words, the *GCA* algorithm is not only prioritizing tasks by its importance but also by the urgency among task. The prioritizing scheme of *GCA* scheduling technique can be accomplished by using simple stack operations, push and pop, which are outlined in *GCA_List_Phase* procedure as follows.

```
Begin_GCA_List_Phase
    Initially, construct an array of Boolean QV and a stack S.
    QV[n_j] = false, ∀ n_j ∈ V.
    Push n_exit on top of S.
    While S is not empty do
       Peek task n_j on the top of S;
       If( all QV[n_i] are true, for all n_i ∈ pred(n_j) or task n_j is n_entry)    {
          Pop task n_j from top of S and put n_j into scheduling list L;
          QV[ n_j] = true; }
       Else      /* search the CT(n_j) */
          For each task n_i, where n_i ∈ pred(n_j) do
             If(QV[n_i] = false)
```

```
            Put CS(n_i) into container C;
         Endif
      Push tasks pred(n_j) from C into S by non-decreasing order according to their
   critical scores;
      Reset C to empty;
      /* if there are 2+ tasks with same CS(n_i), task n_i is randomly pushed into S.
   EndWhile
End_GCA_List_Phase
```

In processor-selection phase, tasks will be deployed from list L that obtained in listing phase to suitable processor in FIFO manner. According to the ordered list $L = \{n_1, n_4, n_2, n_5, n_9, n_3, n_7, n_6, n_8, n_{10}\}$, we have the complete calculated *EFTs* of tasks in DAG-1 and the schedule results of *GCA* algorithm are listed in Table 2 and Figure 2(a), respectively.

Table 2: Earliest Finish Time of tasks in DAG-1 using *GCA* algorithm

***Earliest Finish Time* of tasks in *GCA* algorithm**									
n_1	n_2	n_3	n_4	n_5	n_6	n_7	n_8	n_9	n_{10}
9	27	42	19.7	32.7	47.6	53	65.7	54.7	84.7

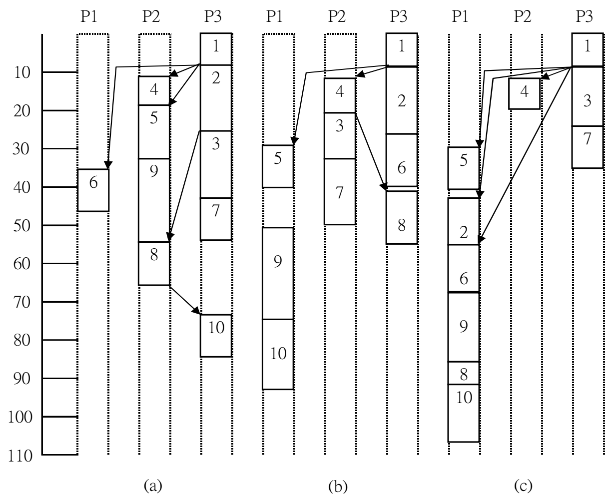


Figure 2: Schedule results of three algorithms on DAG-1 (a) *GCA* (makespan = 84.7) (b) *CA* (makespan = 92.4) (c) *HEFT* (makespan = 108.2).

In order to profile significance of the *GCA* scheduling technique, the schedule results of other algorithms, *CA* and *HEFT* are depicted in Figure 2(b) and 2(c), respectively. The *GCA* scheduling techniques incorporates the consideration of heterogeneous communication costs among processors in processor selection phase. Such enhancement facilitates the selection of best candidate of processors to execute specific tasks.

5. Performance Evaluation

5.1 Random Graph Generator

We implemented a Random Graph Generator (RGG) to simulate application graphs with various characteristics. RGG uses the following input parameters to produce diverse graphs.

- Weight of graph (*weight*), which is a constant = {32, 128, 512, 1024}.
- Number of tasks in the graph (n), where n = {20, 40, 60, 80, 100}.
- Graph parallelism (p), the graph parallelism determines shape of a graph. p is assigned for 0.5, 1.0 and 2.0. The level of graph is defined as $\lfloor \sqrt{v} / p \rfloor$. For example, graph with p = 2.0 has higher parallelism than graph with p = 1.0.
- Out degree of a task (d), where d = {1, 2, 3, 4, 5}. The out degree of a task indicates relationship with other tasks, the larger degree of a task the higher task dependence.
- Heterogeneity (h), determines computational cost of task n_i executed on processor P_k, i.e., $w_{i,k}$, which is randomly generated by the following formula.

$$w_i \times \left(1 - \frac{h}{2}\right) \leq w_{i,k} \leq w_i \times \left(1 + \frac{h}{2}\right). \tag{10}$$

RGG randomizes w_i from the interval [1, *weight*]. Note that larger value of *weight* represents the estimation is with higher precision. In our simulation, h was assigned by 0.1, 0.25, 0.5, 0.75 and 1.0.

- Communication to Computation Ratio (*CCR*), where *CCR* = {0.1, 0.5, 1, 2, 10}.

5.2 Comparison Metrics

As mentioned earlier, the objective of DAG scheduling problem is to minimize the completion time of an application. To verify the performance of a scheduling algorithm, several comparative metrics are given below for comparison:

- *Makespan*, also known as schedule length, which is defined as follows,

$$Makespan = \max(EFT(n_{exit})) \tag{11}$$

- *Speedup*, defined as following equation,

$$Speedup = \frac{\min_{P_j \in M}\{\sum_{n_i \in V} w_{i,j}\}}{makespan}, \text{ where } M \text{ is the set of processors} \tag{12}$$

The numerator is the minimal accumulated sum of computation cost of tasks which are assigned on one processor. Equation (12) represents the ratio of sequential execution time to parallel execution time.

- Percentage of Quality of Schedules (PQS)

The percentage of the *GCA* algorithm produces better, equal and worse quality of schedules compared to other algorithms.

5.3 Simulation Results

The first evaluation aims to demonstrate the merit of the *GCA* algorithm by showing quality of schedules using RGG. Simulation results were obtained upon different parameters with totally 1875 DAGs. Figure 3 reports the comparison by setting different *weight* = {32, 128, 512, 1024}. The term “Better” represents percentage of testing samples the *GCA* algorithm outperforms the *CA* algorithm. The term “Equal” represents both algorithm have same makespan in a given DAG. The tem “Worse” represents opposite results to the “Better” cases. Figure 4 gives the PQS results by setting different number of processors. Overall, the *GCA* scheduling algorithm presents superior performance for 65% test samples.

Speedup of the *GCA*, *CA* and *HEFT* algorithms to execute 1875 DAGs with fix processor number (P=16) under different number of task (n) are shown in Figure 5. The speedup of these algorithms show placid when number of task is small and increased significantly when number of tasks becomes large. In general, the *GCA* algorithm has better speedup than the other two algorithms. Improvement rate of the *GCA* algorithm in terms of average speedup is about 7% to the *CA* algorithm and 34% to the *HEFT* algorithm. The improvement rate (IR_{GCA}) is estimated by the following equation:

$$IR_{GCA} = \frac{\sum Speedup(GCA) - \sum Speedup(HEFT\ or\ CA)}{\sum Speedup(HEFT\ or\ CA)} \qquad (13)$$

weight	32	128	512	1024
Better	65.33%	61.13%	67.07%	67.47%
Equal	34.40%	38.87%	32.93%	32.53%
Worse	0.27%	0%	0%	0%

Figure 3: PQS: *GCA* compared with CA (3 processors)

processor	5	6	7	8
Better	61.13%	72.33%	63.27%	66.60%
Equal	38.87%	27.67%	36.73%	33.40%
Worse	0%	0%	0%	0%

Figure 4: PQS: *GCA* compared with *CA* (*weight* = 128)

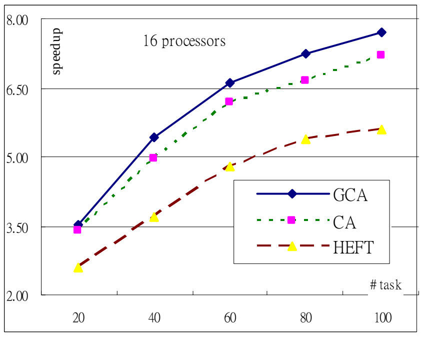


Figure 5: Speedup of *GCA*, *CA* and *HEFT* with different number of tasks (n).

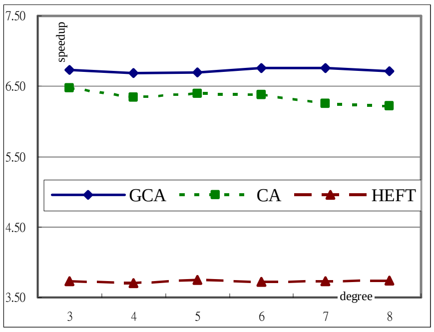
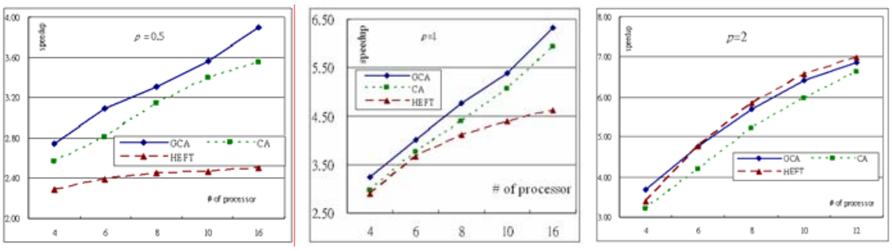


Figure 6: Speedup of *GCA*, *CA* and *HEFT* with different out-degree of tasks (*d*)

Speedup of the *GCA*, *CA* and *HEFT* algorithms to execute different DAGs with fix processor number (*P*=16) and task number (*n*=60) under different out-degree of tasks (*d*) are shown in Figure 6. The results of Figure 6 demonstrate the speedup influence by task dependence. We observe that speedups of scheduling algorithms are less dependent on tasks' dependence. Although the speedups of three algorithms are stable, the *GCA* algorithm outperforms the other two algorithms in most cases. Improvement rate of the *GCA* algorithm in terms of average speedup is about 5% to the *CA* algorithm and 80% to the *HEFT* algorithm.

Figure 7 shows simulation results of three algorithms upon different processor number and degree of parallelization. It is noticed that, graphs with larger value of *p* tends to with higher parallelism. As shown in Figures 7(a) and (b), the *GCA* algorithm performs well in linear graphs (*p*=0.5) and general graphs (*p*=1.0). On the contrary, Figure 7(c) shows that the *HEFT* scheduling algorithm has superior performance when degree of parallelism is high. In general, for graphs with low parallelism (e.g., $p = 0.5$), the *GCA* algorithm has 33% improvement rate in terms of average speedup compare to the *HEFT* algorithm; for graphs with normal parallelism (e.g., $p = 1$), the *GCA* algorithm has 20% improvement rate. For graphs with high parallelism (e.g., $p = 2$), the *GCA* algorithm performs worse than the *HEFT* by 3% performance.

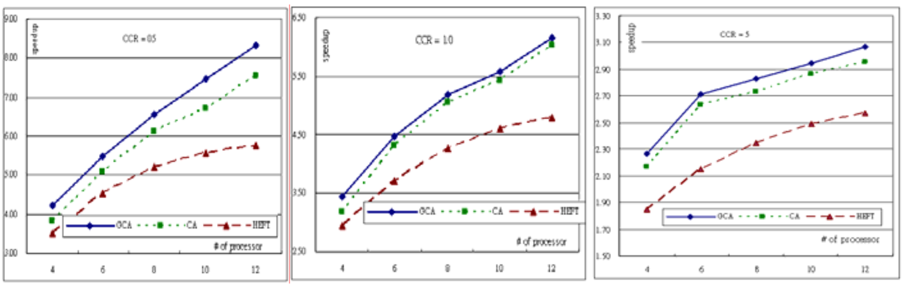
Speedup of the *GCA*, *CA* and *HEFT* algorithms to execute different DAGs with fix processor number (*P*=16) and task number (*n*=60) under different out-degree of tasks (*d*) are shown in Figure 6. The results of Figure 6 demonstrate the speedup influence by task dependence. We observe that speedups of scheduling algorithms are less dependent on tasks' dependence. Although the speedups of three algorithms are stable, the *GCA* algorithm outperforms the other two algorithms in most cases. Improvement rate of the *GCA* algorithm in terms of average speedup is about 5% to the *CA* algorithm and 80% to the *HEFT* algorithm.



(a) (b) (c)

Figure 7: Speedup with different degree of parallelism (*p*) (a) $p = 0.5$ (b) $p = 1$ (c) $p = 2$.

The impact of communication overheads on speedup are plotted in Figure 8 by setting different value of *CCR*. It is noticed that increase of *CCR* will downgrade the speedup we can obtained. For example, speedup offered by *CCR* = 0.1 has maximal value 8.3 in *GCA* with 12 processors; for *CCR* = 1.0, the *GCA* algorithm has maximal speedup 6.1 when processor number is 12; and the same algorithm, *GCA*, has maximal speedup 3.1 for *CCR* = 5 with 12 processors. This is due to the fact that when communication overheads higher than computational overheads, costs for tasks migration will offset the benefit of moving tasks to faster processors.



(a) (b) (c)

Figure 8: Speedup results with different *CCR* (a) *CCR*=0.5 (b) *CCR* = 1 (c) *CCR* = 5.

6. Conclusions

The problem of scheduling a weighted directed acyclic graph (DAG) to a set of heterogeneous processors to minimize the completion time has been recently studied. Several techniques have been presented in the literature to improve performance. This paper presented a general Critical-task Anticipation (*GCA*) algorithm for DAG scheduling system. The *GCA* scheduling algorithm employs task prioritizing technique based on *CA* algorithm and introduces a new processor selection scheme by considering heterogeneous communication costs among processors. *GCA* scheduling algorithm is a list scheduling approach with simple data structure and profitable for grid and scalable computing. Experimental results show that *GCA* has superior performance compare to the well known *HEFT* scheduling heuristic algorithm and our previous proposed *CA* algorithm which did not incorporate the consideration of heterogeneous communication costs into processor selection phase. Experimental results show that *GCA* is equal or superior to *HEFT* and *CA* scheduling algorithms in most cases and it enhances to fit more real grid system.

Acknowledgements

This paper is based upon work supported by National Science Council (NSC), Taiwan, under grants no. NSC95-2213-E-216-006. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSC.

References

[1] R. Bajaj and D. P. Agrawal, "Improving Scheduling of Tasks in a Heterogeneous Environment," *IEEE Trans. on PDS*, vol. 15, no. 2, pp. 107-118, 2004.

[2] S. Behrooz, M. Wang, and G. Pathak, "Analysis and Evaluation of Heuristic Methods for Static Task Scheduling," *Jounal of Parallel and Distributed Computing*, vol. 10, pp. 222-232, 1990.

[3] M.R Gary and D.S. Johnson, "Computers and Interactability: A guide to the Theory of NP-Completeness", W.H. Freeman and Co., 1979.

[4] T. Hagras and J. Janecek," A High Performance, Low Complexity Algorithm for Compile-Time Task Scheduling in Heterogeneous Systems," *Parallel Computing*, vol. 31, Issue 7, pp. 653-670, 2005.

[5] Ching-Hsieh Hsu and Ming-Yuan Weng, "An Improving Critical-Task Anticipation Scheduling Algorithm for Heterogeneous Computing Systems", Proceedings of the Eleventh Asia-Pacific Computer Systems Architecture Conference, LNCS 4186, pp. 97-110, 2006.

[6] E. Ilavarasan P. Thambidurai and R. Mahilmannan, "Performance Effective Task Scheduling Algorithm for Heterogeneous Computing System,**"** *IEEE Proceedings of IPDPS*, pp. 28-38, 2005.

[7] S. Ranaweera and D. P. Agrawal, "A Task Duplication Based Scheduling Algorithm for Heterogeneous Systems," *IEEE Proceedings of IPDPS*, pp. 445-450, 2000.

[8] Rizos Sakellariou and Henan Zhao, "A Hybrid Heuristic for DAG Scheduling on Heterogeneous Systems", Proc. of the *IEEE IPDPS* Workshop 1, pp. 111b, 2004.

[9] H. Topcuoglu, S. Hariri and W. Min-You, "Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing," *IEEE Transactions on PDS*, vol.13, no. 3, pp. 260-274, 2002.

行政院所屬各機關人員出國報告書提要

撰寫時間： 97 年 4 月 20 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	副教授	
出席國際會議名稱	The 22nd International Conference on Advanced Information Networking and Applications (AINA-08), March 25 -28 2008.				
到達國家及地點	**Okinawa, Japan**		出國期間	自 97 年 03 月 25 日 迄 97 年 03 月 28 日	

內容提要

一、主要任務摘要（五十字以內）

AINA-08 是網路相關研究領域一個大型的研討會。這一次參與AINA-08除了發表相關研究成果以外，也在會場上看到許多新的研究成果與方向。此外，也與許多學術界的朋友交換研究心得。

二、對計畫之效益（一百字以內）

這一次參與 AINA-08 除了發表我們在此一計劃最新的研究成果以外，也在會場中，向多位國內外學者解釋我們的研究內容，彼此交換研究心得。除了讓別的團隊知道我們的研究方向與成果，我們也可以學習他人的研究經驗。藉此，加強國際合作，提升我們的研究質量。

三、經過

這一次在 Okinawa 所舉行的國際學術研討會議共計四天。第一天是 Workshop Program。第二天，由 Dr. Michel Raynal 的專題演講， “Synchronization is Coming Back, But is it the Same?” 作為研討會的開始。緊接著是五個平行的場次，分為上下午進行。本人全程參與研討會的議程。晚上在大會的地點舉行歡迎晚宴。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，合影留念。第三天，專題演講是由 Dr. Shigeki Yamada 針對 “Cyber Science Infrastructure (CSI) for Promoting Research Activities of Academia and Industries in Japan”發表演說。本人也參

與的第三天全部的大會議程。晚宴，大會安排交通車到市郊一個花園餐廳舉行。最後一天，本人亦參與了所有的場次，並且發表了這一次的論文。本人主要聽取 GRID 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，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。四天下來，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：無線網路技術、網路安全、GRID、資料庫以及普及運算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。

四、心得

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。

五、建議與結語

出席國際會議，註冊費越來越貴(AINA-08 約兩萬元)，若會議在亞州舉行，補助的經費免強足夠，但是若在歐美，經費往往不足。降低同學參與歐美的會議。

大會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。

六、攜回資料

論文集光碟片

七、出國行程表

3/25 前往 Okinawa　下午研討會報到，參與 AINA-08 Workshop Progra,

3/26 全日參與研討會

3/27 全日參與研討會

3/28 全日參與研討會、晚上飛機返回台灣

Towards Improving QoS-Guided Scheduling in Grids

Ching-Hsien Hsu[1], Justin Zhan[2], Wai-Chi Fang[3] and Jianhua Ma[4]

[1]*Department of Computer Science and Information Engineering, Chung Hua University, Taiwan*
chh@chu.edu.tw

[2]*Heinz School, Carnegie Mellon University, USA*
justinzh@andrew.cmu.edu

[3]*Department of Electronics Engineering, National Chiao Tung University, Taiwan*
wfang@mail.nctu.edu.tw

[4]*Digital Media Department, Hosei University, Japan*
jianhua@hosei.ac.jp

Abstract

With the emergence of grid technologies, the problem of scheduling tasks in heterogeneous systems has been arousing attention. In this paper, we present two optimization schemes, Makespan Optimization Rescheduling (MOR) and Resource Optimization Rescheduling (ROR), which are based on the QoS Min-Min scheduling technique, for reducing the makespan of a schedule and the need of total resource amount. The main idea of the proposed techniques is to reduce overall execution time without increasing resource need; or reduce resource need without increasing overall execution time. To evaluate the effectiveness of the proposed techniques, we have implemented both techniques along with the QoS Min-Min scheduling algorithm. The experimental results show that the MOR and ROR optimization schemes provide noticeable improvements.

1. Introduction

With the emergence of IT technologies, the need of computing and storage are rapidly increased. To invest more and more equipments is not an economic method for an organization to satisfy the even growing computational and storage need. As a result, grid has become a widely accepted paradigm for high performance computing.

To realize the concept virtual organization, in [13], the grid is also defined as "A type of parallel and distributed system that enables the sharing, selection, and aggregation of geographically distributed autonomous and heterogeneous resources dynamically at runtime depending on their availability, capability, performance, cost, and users' quality-of-service requirements". As the grid system aims to satisfy users' requirements with limit resources, scheduling grid resources plays an important factor to improve the overall performance of a grid.

In general, grid scheduling can be classified in two categories: the performance guided schedulers and the economy guided schedulers [16]. Objective of the performance guided scheduling is to minimize turnaround time (or makespan) of grid applications. On the other hand, in economy guided scheduling, to minimize the cost of resource is the main objective. However, both of the scheduling problems are NP-complete, which has also instigated many heuristic solutions [1, 6, 10, 14] to resolve. As mentioned in [23], a complete grid scheduling framework comprises application model, resource model, performance model, and scheduling policy. The scheduling policy can further decomposed into three phases, the resource discovery and selection phase, the job scheduling phase and the job monitoring and migration phase, where the second phase is the focus of this study.

Although many research works have been devoted in scheduling grid applications on heterogeneous system, to deal with QOS scheduling in grid is quite complicated due to more constrain factors in job scheduling, such as the need of large storage, big size memory, specific I/O devices or real-time services, requested by the tasks to be completed. In this paper, we present two QoS based rescheduling schemes aim to improve the makespan of scheduling batch jobs in grid. In addition, based on the QoS guided scheduling scheme, the proposed rescheduling technique can also reduce the amount of resource need without increasing the makespan of grid jobs. The main contribution of this work are twofold, one can shorten the turnaround time of grid applications without increasing the need of grid resources; the other one can minimize the need of grid resources without increasing the turnaround time of grid applications, compared with the traditional QoS guided scheduling method. To evaluate the performance of the proposed techniques, we have implemented our rescheduling approaches along with the QoS Min-Min scheduling algorithm [9] and the non-QoS based Min-Min scheduling algorithm. The experimental results show that the proposed techniques are effective in heterogeneous systems under different circumstances. The improvement is also significant in economic grid model [3].

The rest of this paper is organized as follows. Section 2 briefly describes related research in grid computing and job scheduling. Section 3 clarifies our research model by illustrating the traditional Min-min model and the QoS guided Min-min model. In Section 4, two optimization schemes for reducing the total execution time of an application and reducing resource need are presented, where two rescheduling approaches are illustrated in detail. We conduct performance evaluation and discuss experiment results in Section 5. Finally, concluding remarks and future work are given in Section 6.

2. Related Work

Grid scheduling can be classified into traditional grid scheduling and QoS guided scheduling or economic based grid scheduling. The former emphasizes the performance of systems of applications, such as system throughput, jobs' completion time or response time. Swany *et al*. provides an approach to improving throughput for grid applications with network logistics by building a tree of "best" paths through the graph and has running time of O(*N*log*N*) for implementations that keep the edges sorted [15]. Such approach is referred as the Minimax Path (MMP) and employs a greedy, tree-building algorithm that produces optimal results [20]. Besides data-parallel applications requiring high performance in grid systems, there is a Dynamic Service Architecture (DSA) based on static compositions and optimizations, but also allows for high performance and flexibility, by use of a lookahead scheduling mechanism [4]. To minimizing the processing time of extensive processing loads originating from various sources, the approaches divisible load model [5] and single level tree network with two root processors with divisible load are proposed [12]. In addition to the job matching algorithm, the resource selection algorithm is at the core of the job scheduling decision module and must have the ability to integrate multi-site computation power. The CGRS algorithm based on the distributed computing grid model and the grid scheduling model integrates a new density-based internet clustering algorithm into the decoupled scheduling approach of the GrADS and decreases its time complexity [24]. The scheduling of parallel jobs in a heterogeneous multi-site environment, where each site has a homogeneous cluster of processors, but processors at different sites has different speeds, is presented in [18]. Scheduling strategy is not only in batch but also can be in real-time. The SAREG approach paves the way to the design of security-aware real-time scheduling algorithms for Grid computing environments [21].

For QoS guided grid scheduling, apparently, applications in grids need various resources to run its completion. In [17], an architecture named public computing utility (PCU) is proposed uses virtual machine (VMs) to implement "time-sharing" over the resources and augments finite number of private resources to public resources to obtain higher level of quality of services. However, the QoS demands maybe include various packet-type and class in executing job. As a result, a scheduling algorithm that can support multiple QoS classes is needed. Based on this demand, a multi-QoS scheduling algorithm is proposed to improve the scheduling fairness and users' demand [11]. He *et al.* [7] also presented a hybrid approach for scheduling moldable jobs with QoS demands. In [9], a novel framework for policy based scheduling in resource allocation of grid computing is also presented. The scheduling strategy can control the request assignment to grid resources by adjusting usage accounts or request priorities. Resource management is achieved by assigning usage quotas to intended users. The scheduling method also supports reservation based grid resource allocation and quality of service feature. Sometimes the scheduler is not only to match the job to which resource, but also needs to find the optimized transfer path based on the cost in network. In [19], a distributed QoS network scheduler (DQNS) is presented to adapt to the ever-changing network conditions and aims to serve the path requests based on a cost function.

3. Research Architecture

Our research model considers the static scheduling of batch jobs in grids. As this work is an extension and optimization of the QoS guided scheduling that is based on Min-Min scheduling algorithm [9], we briefly describe the Min-Min scheduling model and the QoS guided Min-Min algorithm. To simplify the presentation, we first clarify the following terminologies and assumptions.

- *QoS Machine* (M_Q) – machines can provide special services.
- *QoS Task* (T_Q) – tasks can be run completion only on QoS machine.
- *Normal Machine* (M_N) – machines can only run normal tasks.
- *Normal Task* (T_N) – tasks can be run completion on both QoS machine and normal machine.
- A chunk of tasks will be scheduled to run completion based on all available machines in a batch system.
- A task will be executed from the beginning to completion without interrupt.
- The completion time of task t_i to be executed on machine m_j is defined as

$$CT_{ij} = dt_{ij} + et_{ij} \quad (1)$$

Where et_{ij} denotes the estimated execution time of task t_i executed on machine m_j; dt_{ij} is the delay time of task t_i on machine m_j.

The Min-Min algorithm is shown in Figure 1.

```
Algorithm_Min-Min()
{
    while there are jobs to schedule
        for all job i to schedule
            for all machine j
                Compute CT_{i,j} = CT(job i, machine j)
            end for
            Compute minimum CT_{i,j}
        end for
        Select best metric match m
        Compute minimum CT_{m,n}
        Schedule job m on machine n
    end while
} End_of_ Min-Min
```

Figure 1. The Min-Min Algorithm

Analysis: If there are m jobs to be scheduled in n machines, the time complexity of Min-Min algorithm is $O(m^2n)$. The Min-Min algorithm does not take into account the QoS issue in the scheduling. In some situation, it is possible that normal tasks occupied machine that has special services (referred as QoS machine). This may increase the delay of QoS tasks or result idle of normal machines.

The QoS guided scheduling is proposed to resolve the above defect in the Min-Min algorithm. In QoS guided model, the scheduling is divided into two classes, the QoS class and the non-QoS class. In each class, the Min-Min algorithm is employed. As the QoS tasks have higher priority than normal tasks in QoS guided scheduling, the QoS tasks are prior to be allocated on QoS machines. The normal tasks are then scheduled to all machines in Min-Min manner. Figure 2 outlines the method of QoS guided scheduling model with the Min-Min scheme.

Analysis: If there are m jobs to be scheduled in n machines, the time complexity of QoS guided scheduling algorithm is $O(m^2n)$.

Figure 3 shows an example demonstrating the Min-Min and QoS Min-Min scheduling schemes. The asterisk * means that tasks/machines with QoS demand/ability, and the X means that QoS tasks couldn't be executed on that machine. Obviously, the QoS guided scheduling algorithm gets the better performance than the Min-Min algorithm in term of makespan. Nevertheless, the QoS guided model is not optimal in both makespan and resource cost. We will describe the rescheduling optimization in next section.

```
Algorithm_QOS-Min-Min()
{
  for all tasks ti in meta-task Mv (in an arbitrary order)
      for all hosts m_j (in a fixed arbitrary order)
          CT_ij = et_ij + dt_j
      end for
  end for
  do until all tasks with QoS request in Mv are mapped
       for each task with high QoS in Mv,
            find a host in the QoS qualified host set that obtains
            the earliest completion time
       end for
       find task t_k with the minimum earliest completion time
       assign task t_k to host m_l that gives the earliest completion
       time
       delete task t_k from Mv
       update d_tl
       update CT_il for all i
  end do
  do until all tasks with non-QoS request in Mv are mapped
       for each task in Mv
           find the earliest completion time and the
           corresponding host
       end for
       find the task t_k with the minimum earliest completion time
       assign task t_k to host m_l that gives the earliest completion
       time
       delete task t_k from Mv
       update d_tl
       update CT_il for all i
  end do
} End_of_ QOS-Min-Min
```

Figure 2. The QoS Guided Algorithm

4. Rescheduling Optimization

Grid scheduling works as the mapping of individual tasks to computer resources, with respecting service level agreements (SLAs) [2]. In order to achieve the optimized performance, how to mapping heterogeneous tasks to the best fit resource is an important factor. The Min-Min algorithm and the QoS guided method aims at scheduling jobs to achieve better makespan. However, there are still having rooms to make improvements. In this section, we present two optimization schemes based on the QoS guided Min-Min approach.

	*M1	M2	M3
T1	7	4	7
T2	3	3	5
T3	9	5	7
*T4	5	X	X
T5	9	8	6
*T6	5	X	X

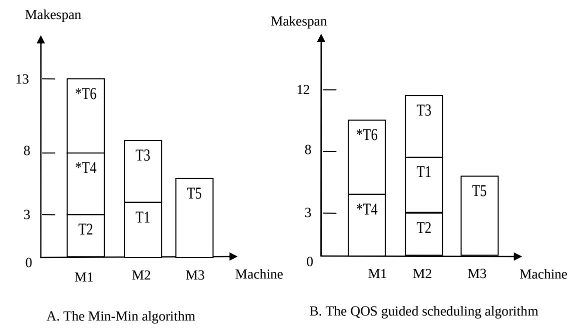
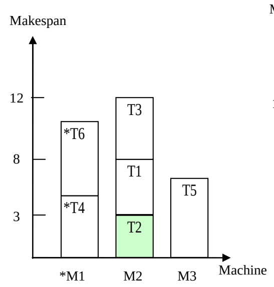


Figure 3. Min-Min and QoS Guided Min-Min

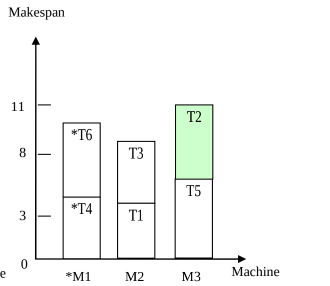
4.1 Makespan Optimization Rescheduling (*MOR*)

The first one is *Makespan Optimization Rescheduling* (*MOR*), which focuses on improving the makespan to achieve better performance than the QoS guided scheduling algorithm. Assume the makespan achieved by the QoS guided approach in different machines are CT_1, CT_2, …, CT_m, with CT_k = max { CT_1, CT_2, …, CT_m }, where m is the number of machines and $1 \le k \le m$. By subtracting $CT_k - CT_i$, where $1 \le i \le m$ and $i \neq k$, we can have m-1 available time fragments. According to the size of these available time fragments and the size of tasks in machine M_k, the *MOR* dispatches suitable tasks from machine M_k to any other machine that has available and large enough time fragments. Such optimization is repeated until there is no task can be moved.

	*M1	M2	M3
T1	7	4	7
T2	3	3	5
T3	9	5	7
*T4	5	X	X
T5	9	8	6
*T6	5	X	X



A. The QOS guided scheduling algorithm



B. The Makespan Optimization Rescheduling (MOR) algorithm

Figure 4. Example of *MOR*

Recall the example given in Figure 3, Figure 4 shows the optimization of the *MOR* approach. The left side of Figure 4 demonstrates that the QoS guided scheme gives a schedule with makespan = 12, wheremachine M2 presents maximum *CT* (completion time), which is assembled by tasks T2, T1 and T3. Since the *CT* of machine 'M3' is 6, so 'M3' has an available time fragment (6). Checking all tasks in machine M2, only T2 is small enough to be allocated in the available time fragment in M3. Therefore, task M2 is moved to M3, resulting machine 'M3' has completion time *CT*=11, which is better than the QoS guided scheme.

As mentioned above, the *MOR* is based on the QoS guided scheduling algorithm. If there are *m* tasks to be scheduled in *n* machines, the time complexity of *MOR* is $O(m^2n)$. Figure 5 outlines a pseudo of the *MOR* scheme.

```
Algorithm_MOR()
{
   for CT_j in all machines
        find out the machine with maximum makespan CT_max and
        set it to be the standard
   end for
   do until no job can be rescheduled
        for job i in the found machine with CT_max
             for all machine j
                  according to the job's QOS demand, find the
                  adaptive machine j
                  if (the execute time of job i in machine j + the
                  CT_j < makespan)
                     rescheduling the job i to machine j
                     update the CT_j and CT_max
                     exit for
                  end if
             next for
             if the job i can be reschedule
                  find out the new machine with maximum CT_max
                  exit for
             end if
        next for
   end do
} End_of_ MOR
```

Figure 5. The *MOR* Algorithm

4.2 Resource Optimization Rescheduling (*ROR*)

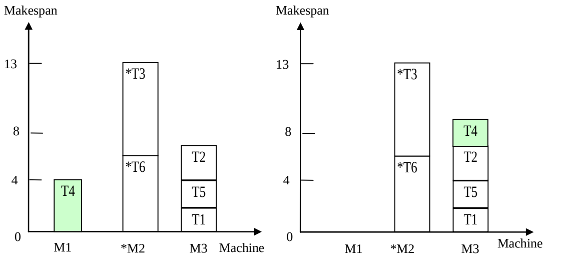
Following the assumptions described in *MOR*, the main idea of the *ROR* scheme is to re-dispatch tasks from the machine with minimum number of tasks to other machines, expecting a decrease of resource need. Consequently, if we can dispatch all tasks from machine M_x to other machines, the total amount of resource need will be decreased.

Figure 6 gives another example of QoS scheduling, where the QoS guided scheduling presents makespan = 13. According to the clarification of *ROR*, machine 'M1' has the fewest amount of tasks. We can dispatch the task 'T4' to machine 'M3' with the following constraint

$$CT_{ij} + CT_j <= CT_{max} \qquad (2)$$

The above constraint means that the rescheduling can be performed only if the movement of tasks does not increase the overall makespan. In this example, $CT_{43} = 2$, CT_3=7 and CT_{max}=CT_2=13. Because the makespan of M3 (CT_3) will be increased from 7 to 9, which is smaller than the CT_{max}, therefore, the task migration can be performed. As the only task in M1 is moved to M3, the amount of resource need is also decreased comparing with the QoS guided scheduling.

	M1	*M2	M3
T1	3	4	2
T2	6	6	3
*T3	X	7	X
T4	4	6	2
T5	5	7	2
*T6	X	6	X



A. The QOS guided scheduling

B. The Resource Optimization Rescheduling (ROR) Algorithm

Figure 6. Example of *ROR*

The *ROR* is an optimization scheme which aims to minimize resource cost. If there are m tasks to be scheduled in n machines, the time complexity of *ROR* is also $O(m^2n)$. Figure 7 depicts a high level description of the *ROR* optimization scheme.

```
Algorithm_MOR()
{
   for m in all machines
         find out the machine m with minimum count of jobs
   end for
   do until no job can be rescheduled
         for job i in the found machine with minimum count of jobs
               for all machine j
                   according to the job's QOS demand, find the
                   adaptive machine j
                   if (the execute time of job i in machine j + the
                     CT_j <= makespan CT_max)
                        rescheduling the job i to machine j
                        update the CT_j
                        update the count of jobs in machine m and
                        machine j
                        exit for
                    end if
               next for
         next for
   end do
 } End_of_ MOR
```

Figure 7. The *ROR* Algorithm

5. Performance Evaluation

5.1 Parameters and Metrics

To evaluate the performance of the proposed techniques, we have implemented the Min-Min scheduling algorithm and the QoS guided Min-Min scheme. The experiment model consists of heterogeneous machines and tasks. Both of the Machines and tasks are classified into QoS type and non-QoS type. Table 1 summarizes six parameters and two comparison metrics used in the experiments. The number of tasks is ranged from 200 to 600. The number of machines is ranged from 50 to 130. The percentage of QoS machines and tasks are set between 15% and 75%. Heterogeneity of tasks are defined as H_t (for non-QoS task) and H_Q (for QoS task), which is used in generating random tasks. For example, the execution time of a non-QoS task is randomly generated from the interval [10, $H_t \times 10^2$] and execution time of a QoS task is randomly generated from the interval [10^2, $H_Q \times 10^3$] to reflect the real application world. All of the parameters used in the experiments are generated randomly with a uniform distribution. The results demonstrated in this section are the average values of running 100 random test samples.

Table 1: Parameters and Comparison Metrics

Task number (N_T)	{200, 300, 400, 500, 600}
Resource number (N_R)	{50, 70, 90, 110, 130}
Percentage of QOS resources (Q_R%)	{15%, 30%, 45%, 60%, 75%}
Percentage of QOS tasks (Q_T%)	{15%, 30%, 45%, 60%, 75%}
Heterogeneity of non-QOS tasks (H_T)	{1, 3, 5, 7, 9}
Heterogeneity of QOS tasks (H_Q)	{3, 5, 7, 9, 11}
Makespan	The completion time of a set of tasks
Resource Used (R_U)	Number of machines used for executing a set of tasks

5.2 Experimental Results of *MOR*

Table 2 compares the performance of the *MOR*, Min-Min algorithm and the QoS guided Min-Min scheme in term of makespan. There are six tests that are conducted with different parameters. In each test, the configurations are outlined beside the table caption from (a) to (f). Table (a) changes the number of tasks to analyze the performance results. Increasing the number of tasks, improvement of *MOR* is limited. An average improvement ratio is from 6% to 14%. Table (b) changes the number of machines. It is obvious that the *MOR* has significant improvement in larger grid systems, i.e., large amount of machines. The average improvement rate is 7% to 15%. Table (c) discusses the influence of changing percentages of QoS machines. Intuitionally, the *MOR* performs best with 45% QoS machines. However, this observation is not always true. By analyzing the four best ones in (a) to (d), we observe that the four tests (a) N_T=200 (N_R=50, Q_R=30%, Q_T=20%) (b) N_R=130 (N_T=500, Q_R=30%, Q_T=20%) (c)

Q_R=45% (N_T=300, N_R=50, Q_T=20%) and (d) Q_T=15% (N_T=300, N_R=50, Q_R=40%) have best improvements. All of the four configurations conform to the following relation,

$$0.4 \times (N_T \times Q_T) = N_R \times Q_R \quad (3)$$

This observation indicates that the improvement of *MOR* is significant when the number of QoS tasks is 2.5 times to the number of QoS machines. Tables (e) and (f) change heterogeneity of tasks. We observed that heterogeneity of tasks is not critical to the improvement rate of the *MOR* technique, which achieves 7% improvements under different heterogeneity of tasks.

Table 2: Comparison of Makespan

(a) (N_R=50, Q_R=30%, Q_T=20%, H_T=1, H_Q=1)

Task Number (N_T)	200	300	400	500	600
Min-Min	978.2	1299.7	1631.8	1954.6	2287.8
QOS Guided Min-Min	694.6	917.8	1119.4	1359.9	1560.1
MOR	597.3	815.5	1017.7	1254.8	1458.3
Improved Ratio	14.01%	11.15%	9.08%	7.73%	6.53%

(b) (N_T=500, Q_R=30%, Q_T=20%, H_T=1, H_Q=1)

Resource Number (N_R)	50	70	90	110	130
Min-Min	1931.5	1432.2	1102.1	985.3	874.2
QOS Guided Min-Min	1355.7	938.6	724.4	590.6	508.7
MOR	1252.6	840.8	633.7	506.2	429.4
Improved Ratio	7.60%	10.42%	12.52%	14.30%	15.58%

(c) (N_T=300, N_R=50, Q_T=20%, H_T=1, H_Q=1)

Q_R%	15%	30%	45%	60%	75%
Min-Min	2470.8	1319.4	888.2	777.6	650.1
QOS Guided Min-Min	1875.9	913.6	596.1	463.8	376.4
MOR	1767.3	810.4	503.5	394.3	339.0
Improved Ratio	5.79%	11.30%	15.54%	14.99%	9.94%

(d) (N_T=300, N_R=50, Q_R=40%, H_T=1, H_Q=1)

Q_T%	15%	30%	45%	60%	75%
Min-Min	879.9	1380.2	1801.8	2217.0	2610.1
QOS Guided Min-Min	558.4	915.9	1245.2	1580.3	1900.6
MOR	474.2	817.1	1145.1	1478.5	1800.1
Improved Ratio	15.07%	10.79%	8.04%	6.44%	5.29%

(e) (N_T=500, N_R=50, Q_R=30%, Q_T=20%, H_Q=1)

H_T	1	3	5	7	9
Min-Min	1891.9	1945.1	1944.6	1926.1	1940.1
QOS Guided Min-Min	1356.0	1346.4	1346.4	1354.9	1357.3
MOR	1251.7	1241.4	1244.3	1252.0	1254.2
Improved Ratio	7.69%	7.80%	7.58%	7.59%	7.59%

(f) (N_T=500, N_R=50, Q_R=30%, Q_T=20%, H_T=1)

H_Q	3	5	7	9	11
Min-Min	1392.4	1553.9	1724.9	1871.7	2037.8
QOS Guided Min-Min	867.5	1007.8	1148.2	1273.2	1423.1
MOR	822.4	936.2	1056.7	1174.3	1316.7
Improved Ratio	5.20%	7.11%	7.97%	7.77%	7.48%

5.3 Experimental Results of *ROR*

Table 3 analyzes the effectiveness of the *ROR* technique under different circumstances.

Table 3: Comparison of Resource Used

(a) (N_R=100, Q_R=30%, Q_T=20%, H_T=1, H_Q=1)

Task Number (N_T)	200	300	400	500	600
QOS Guided Min-Min	100	100	100	100	100
ROR	39.81	44.18	46.97	49.59	51.17
Improved Ratio	60.19%	55.82%	53.03%	50.41%	48.83%

(b) (N_T=500, Q_R=30%, Q_T=20%, H_T=1, H_Q=1)

Resource Number (N_R)	50	70	90	110	130
QOS Guided Min-Min	50	70	90	110	130
ROR	26.04	35.21	43.65	50.79	58.15
Improved Ratio	47.92%	49.70%	51.50%	53.83%	55.27%

(c) (N_T=500, N_R=50, Q_T=20%, H_T=1, H_Q=1)

Q_R%	15%	30%	45%	60%	75%
QOS Guided Min-Min	50	50	50	50	50
ROR	14.61	25.94	35.12	40.18	46.5
Improved Ratio	70.78%	48.12%	29.76%	19.64%	7.00%

(d) (N_T=500, N_R=100, Q_R=40%, H_T=1, H_Q=1)

Q_T%	15%	30%	45%	60%	75%
QOS Guided Min-Min	100	100	100	100	100
ROR	57.74	52.9	48.54	44.71	41.49
Improved Ratio	42.26%	47.10%	51.46%	55.29%	58.51%

(e) (N_T=500, N_R=100, Q_R=30%, Q_T=20%, H_Q=1)

H_T	1	3	5	7	9
QOS Guided Min-Min	100	100	100	100	100
ROR	47.86	47.51	47.62	47.61	47.28
Improved Ratio	52.14%	52.49%	52.38%	52.39%	52.72%

(f) (N_T=500, N_R=100, Q_R=30%, Q_T=20%, H_T=1)

H_Q	3	5	7	9	11
QOS Guided Min-Min	100	100	100	100	100
ROR	54.61	52.01	50.64	48.18	46.53
Improved Ratio	45.39%	47.99%	49.36%	51.82%	53.47%

Similar to those of Table 2, Table (a) changes the number of tasks to verify the reduction of resource that needs to be achieved by the *ROR* technique. We noticed that the *ROR* has significant improvement in minimizing grid resources. Comparing with the QoS guided Min-Min scheduling algorithm, the *ROR* achieves 50% ~ 60% improvements without increasing overall makespan of a chunk of grid tasks. Table (b) changes the number of machines. The *ROR* retains 50% improvement ratio. Table (c) adjusts percentages of QoS machine. Because this test has 20% QoS tasks, the *ROR* performs best at 15% QoS machines. This observation implies that the *ROR* has significant improvement when QoS tasks and QoS machines are with the same percentage. Table (d) sets 40% QoS machine and changes the percentages of QoS tasks. Following the above analysis, the *ROR* technique achieves more than 50% improvements when QoS tasks are with 45%, 60% and 75%. Tables (e) and (f) change the heterogeneity of tasks. Similar to the results of section 5.2, the heterogeneity of tasks is not critical to the improvement rate of the *ROR* technique. Overall speaking, the *ROR* technique presents 50% improvements in minimizing total resource need compare with the QoS guided Min-Min scheduling algorithm.

6. Conclusions

In this paper we have presented two optimization schemes aiming to reduce the overall completion time (makespan) of a chunk of grid tasks and minimize the total resource cost. The proposed techniques are based on the QoS guided Min-Min scheduling algorithm. The optimization achieved by this work is twofold; firstly, without increasing resource costs, the overall task execution time could be reduced by the *MOR* scheme with 7%~15% improvements. Second, without increasing task completion time, the overall resource cost could be reduced by the *ROR* scheme with 50% reduction on average, which is a significant improvement to the state of the art scheduling technique. The proposed *MOR* and *ROR* techniques have characteristics of low complexity, high effectiveness in large-scale grid systems with QoS services.

References

[1] A. Abraham, R. Buyya, and B. Nath, "Nature's Heuristics for Scheduling Jobs on Computational Grids", Proc. 8th IEEE International Conference on Advanced Computing and Communications (ADCOM-2000), pp.45-52, 2000.

[2] A. Andrieux, D. Berry, J. Garibaldi, S. Jarvis, J. MacLaren, D. Ouelhadj, D. Snelling, "Open Issues in Grid Scheduling", National e-Science Centre and the Inter-disciplinary Scheduling Network Technical Paper, UKeS-2004-03.

[3] R. Buyya, D. Abramson, Jonathan Giddy, Heinz Stockinger, "Economic Models for Resource Management and Scheduling in Grid Computing", Journal of Concurrency: Practice and Experience, vol. 14, pp. 13-15, 2002.

[4] Jesper Andersson, Morgan Ericsson, Welf Löwe, and Wolf Zimmermann, "Lookahead Scheduling for Reconfigurable GRID Systems", 10th International Europar'04: Parallel Processing, vol. 3149, pp. 263-270, 2004.

[5] D Yu, Th G Robertazzi, "Divisible Load Scheduling for Grid Computing", 15th IASTED Int'l. Conference on Parallel and Distributed Computing and Systems, Vol. 1, pp. 1-6, 2003

[6] Fangpeng Dong and Selim G. Akl, "Scheduling Algorithms for Grid Computing: State of the Art and Open Problems", Technical Report No. 2006-504, 2006.

[7] Ligang He, Stephen A. Jarvis, Daniel P. Spooner, Xinuo Chen, Graham R. Nudd, "Hybrid Performance-oriented Scheduling of Moldable Jobs with QoS Demands in Multiclusters and Grids", Grid and Cooperative Computing (GCC 2004), vol. 3251, pp. 217–224, 2004.

[8] Xiaoshan He, Xian-He Sun, Gregor Von Laszewski, "A QoS Guided Scheduling Algorithm for Grid Computing", Journal of Computer Science and Technology, vol.18, pp.442-451, 2003.

[9] Jang-uk In, Paul Avery, Richard Cavanaugh, Sanjay Ranka, "Policy Based Scheduling for Simple Quality of Service in Grid Computing", IPDPS 2004, pp. 23, 2004.

[10] J. Schopf. "Ten Actions when Superscheduling: A Grid Scheduling Architecture", Scheduling Architecture Workshop, 7th Global Grid Forum, 2003.

[11] Junsu Kim, Sung Ho Moon, and Dan Keun Sung, "Multi-QoS Scheduling Algorithm for Class Fairness in High Speed Downlink Packet Access", Proceedings of IEEE Personal, Indoor and Mobile Radio Communications Conference (PIMRC 2005), vol. 3, pp. 1813-1817, 2005

[12] M.A. Moges and T.G. Robertazzi, "Grid Scheduling Divisible Loads from Multiple Sources via Linear Programming", 16th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS), pp. 423-428, 2004.

[13] M. Baker, R. Buyya and D. Laforenza, "Grids and Grid Technologies for Wide-area Distributed Computing", in Journal of Software-Practice & Experience, Vol. 32, No.15, pp. 1437-1466, 2002.

[14] Jennifer M. Schopf, "A General Architecture for Scheduling on the Grid", Technical Report ANL/MCS, pp. 1000-1002, 2002.

[15] M. Swany, "Improving Throughput for Grid Applications with Network Logistics", *Proc.* IEEE/ACM Conference on High Performance Computing and Networking, 2004.

[16] R. Moreno and A.B. Alonso, "Job Scheduling and Resource Management Techniques in Economic Grid Environments", LNCS 2970, pp. 25-32, 2004.

[17] Shah Asaduzzaman and Muthucumaru Maheswaran, "Heuristics for Scheduling Virtual Machines for Improving QoS in Public Computing Utilities", *Proc.* 9th International Conference on Computer and Information Technology (ICCIT'06), 2006.

[18] Gerald Sabin, Rajkumar Kettimuthu, Arun Rajan and P Sadayappan, "Scheduling of Parallel Jobs in a Heterogeneous Multi-Site Environment", in the Proc. of the 9th International Workshop on Job Scheduling Strategies for Parallel Processing, LNCS 2862, pp. 87-104 , June 2003.

[19] Sriram Ramanujam, Mitchell D. Theys, "Adaptive Scheduling based on Quality of Service in Distributed Environments", *PDPTA'05*, pp. 671-677, 2005.

[20] T. H. Cormen, C. L. Leiserson, and R. L. Rivest, "Introduction to Algorithms", First edition, MIT Press and McGraw-Hill, ISBN 0-262-03141-8, 1990.

[21] Tao Xie and Xiao Qin, "Enhancing Security of Real-Time Applications on Grids through Dynamic Scheduling", *Proc.* the 11th Workshop on Job Scheduling Strategies for Parallel

Processing (JSSPP'05), pp. 146-158, 2005.
[22] Haobo Yu, Andreas Gerstlauer, Daniel Gajski, "RTOS Scheduling in Transaction Level Models", in Proc. of the 1st IEEE/ACM/IFIP international conference on Hardware/software Codesign & System Synpaper, pp. 31-36, 2003.
[23] Y. Zhu, "A Survey on Grid Scheduling Systems", LNCS 4505, pp. 419-427, 2007.
[24] Weizhe Zhang, Hongli Zhang, Hui He, Mingzeng Hu, "Multisite Task Scheduling on Distributed Computing Grid", LNCS 3033, pp. 57–64, 2004.

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

平行資料程式於計算網格上通訊與I/O局部化研究與應用工具開發(3/3)

計畫類別：☑ 個別型計畫　□ 整合型計畫
計畫編號：NSC95-2221-E-216-006
執行期間：96 年 8 月 1 日至 97 年 7 月 31 日

計畫主持人：許慶賢　中華大學資訊工程學系副教授
共同主持人：
計畫參與人員：　陳泰龍（中華大學工程科學研究所博士生）
張智鈞、郁家豪、蔡秉儒(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

執行單位：中華大學資訊工程學系

中　華　民　國　97　年　10　月　31　日

行政院所屬各機關人員出國報告書提要

撰寫時間： 95 年 9 月 11 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	副教授	
出席國際會議名稱	Eleventh Asia-Pacific Computer Systems Architecture Conference (ACSAC-06), Shanghai, China				
到達國家及地點	ShangHai, China		出國期間	自 95 年 09 月 06 日 迄 95 年 09 月 08 日	

報告內容應包括下列各項：

一、 參加會議經過

這一次在上海所舉行的國際學術研討會議共計三天。第一天上午由 Guang R. Gao 博士針對 The Era of Multi-Core Chips- A Fresh Look on Software Challenges 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 Languages and Compilers 場次聽取報告。本人也在同一天下午發表這一次被大會接受的論文。

第一晚上本人亦參加酒會，並且與幾位國外學者及中國教授交換意見。第二天本人除了在上午參加 Multi-core，Architecture，Networks 場次，也在下午主持了 Power Management 場次，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。第二天晚上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：ILP, TLP, Processor Architecture, Memory System, Operation System, High Performance I/O Architecture 等等熱門的研究課題。

二、 與會心得

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。

三、 考察參觀活動(無是項活動者省略)

四、 建議

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。

五、 攜回資料名稱及內容

1. Conference Program
2. Proceedings

An Efficient Processor Selection Scheme for Master Slave Paradigm on Heterogeneous Networks

Tai-Lung Chen Ching-Hsien Hsu

Department of Computer Science and Information Engineering
Chung Hua University, Hsinchu, Taiwan

chh@chu.edu.tw

Abstract. It is well known that grid technology has the ability to achieve resources shared and tasks scheduled coordinately. In this paper, we present a performance effective pre-scheduling strategy for dispatching tasks onto heterogeneous processors. The main contribution of this study is the consideration of heterogeneous communication overheads in grid systems. One significant improvement of our approach is that average turnaround time could be minimized by selecting processor has the smallest communication ratio first. The other advantage of the proposed method is that system throughput can be increased via dispersing processor idle time. Our proposed technique can be applied to heterogeneous cluster systems as well as computational grid environments, in which the communication costs vary in different clusters. Experimental results show that our techniques outperform other previous algorithms in terms of lower average turnaround time, higher average throughput, less processor idle time and higher processors' utilization.

1 Introduction

Computational grid system integrates geographically distributed computing resources to establish a virtual and high expandable parallel computing infrastructure. In recent years, there are several research investigations done in scheduling problem for 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 in centralized grid system.

The master slave tasking is a simple and widely used technique [1, 2]. In a master slave tasking paradigm, the master node connects to n slave nodes. A set of independent tasks are dispatched by master processor and be processed on the n heterogeneous slave processors. Slave processors execute the tasks accordingly after they receive their tasks. This will restrict that the computation and communication can't overlap. Moreover, 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.

In this paper, an efficient strategy for scheduling independent 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 present lower communication ratio, which will be defined in section 3.2. Improvements of our approach towards both average turnaround time and system throughput.

The remaining of this paper is organized as follows. Section 2 briefly discusses previous related researches, while in section 3 is introduced the research architecture and definition of notation and terminologies used in this paper, where we also present a motivating example to demonstrate the characteristics of the master-slave pre-scheduling model. Section 4 assesses the new scheduling algorithm, the Smallest Communication Ratio (*SCR*), while the illustration of *SCR* on heterogeneous communication is examined in section 5. The performance comparisons and simulations results are discussed in section 6, and finally in section 7, some conclusions of this paper.

2 Related Work

The task scheduling research on heterogeneous processors can be classified into *DAG*s model, master-slave paradigm and computational grids. The main 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. [13] brought up the idea about adaptive scheduling scheme based on homogeneous processor platform, which applies space-sharing and time-sharing to schedule tasks. With the emergence of Grid and ubiquitous computing, new algorithms are in demand to address new concerns arising to grid environments, such as security, quality of service and high system throughput. Berman et al. [6] and Cooper et al. [11] addressed the problem of scheduling incoming applications to available computation resources. Dynamically rescheduling mechanism was introduced to adaptive computing on the Grid. In [8], some simple heuristics for dynamic matching and scheduling of a class of independent tasks onto a heterogeneous computing system have been presented. Moreover, an extended suffrage heuristic was presented in [12] for scheduling the parameter sweep applications that have been implemented in *AppLeS*. They also presented a method to predict the computation time for a task/host pair by using previous host performance.

Chronopoulos et al. [9], Charcranoon et al. [10] and Beaumont et al. [4, 5] introduced the research of master-slave paradigm with heterogeneous processors background. Based on this architecture, Beaumont et al. [1, 2] presented a method on master-slave paradigm to forecast the amount of tasks each processor needs to receive in a given period of time. Beaumont et al. [3] presented the pipelining broadcast method on master-slave platforms, focusing on message passing disregarding computation time. Intuitionally in their implementation, 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 basic concepts and models of this investigation, where we also define notations and terminologies that will be used in subsequent subsections.

3.1 Research Architecture

We have revised several characteristics that were introduced by Beaumont *et al.* [1, 2]. Based on the master slave paradigm introduced in section 1, this paper follows next assumptions as listed.

- 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.
- Heterogeneous communication: communication costs between master and slave processors are of different overheads.

3.2 Definitions

First, we list definitions, notations and terminologies used in this research paper.

Definition 1: In a master slave system, master processor is denoted by M and the n slave processors 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 execution time of each one of slave processors to compute one task are different. We use T_i to represent the execution time of slave processor P_i to complete one task. In this paper, we assume the computation speed of n slave processors is sorted and $T_1 \leq T_2 \leq \ldots \leq T_n$.

Definition 3: Given a master slave system, the time of slave processor P_i to receive one task from master processor is denoted as T_{i_comm}.

Definition 4: A Basic Scheduling Cycle (*BSC*) is defined as $BSC = lcm(T_1 + T_{1\_comm}, T_2 + T_{2\_comm}, ..., T_m + T_{m\_comm})$, where m is the number of processors that will join the computation.

Definition 5: Given a master slave system, the number of tasks processor P_i needs to receive in a basic scheduling cycle is defined as $task(P_i) = \frac{BSC}{T_i + T_{i\_comm}}$.

Definition 6: Given a master slave system, the communication cost of processor P_i in *BSC* is defined as $comm(P_i) = T_{i\_comm} \times task(P_i)$.

Definition 7: Given a master slave system, the computation cost of processor P_i in *BSC* is defined as $comp(P_i) = T_i \times task(P_i)$.

Definition 8: Given a master slave system, the *Communication Ratio* of processor P_i is defined as $CR_i = \frac{T_{i\_comm}}{T_i + T_{i\_comm}}$.

Definition 9: The computational capacity (δ) of a master slave system is defined as the sum of communication ratio of all processors that joined the computation, i.e., $\delta = \sum_{i=1}^{m} CR_i$, where m is the number of processors that involved in the computation.

Definition 10: Given a master slave system with n heterogeneous slave processors, $P_{\max}$ is the processor P_k such that $\max\{k \mid \sum_{i=1}^{k} \frac{T_{i\_comm}}{T_i + T_{i\_comm}} \leq 1\}$, where $1 \leq k \leq n$. i.e. $\sum_{i=1}^{k+1} \frac{T_{i\_comm}}{T_i + T_{i\_comm}} > 1$. We use $P_{\max+1}$ to represent processor P_{k+1}.

3.3 Master Slave Task Scheduling

Discussions on the problem of task scheduling in master slave paradigm will be addressed in two cases, depending on the value of system computational capacity (δ).

As mentioned in section 2, faster processors receive more tasks is an intuitional approach in which tasks are previously allocated to these faster processors, and this method is called Most Jobs First (*MJF*) scheduling algorithm [1, 2]. Fig. 1 shows the pre-scheduling of the *MJF* algorithm. As defined in definition 8, the communication ratio of P_1 to P_4 are $\frac{1}{3}$, $\frac{1}{4}$, $\frac{1}{4}$, and $\frac{1}{6}$, respectively. Because *BSC* = 12, we have $task(P_1)$=4, $task(P_2)$=3, $task(P_3)$=3 and $task(P_4)$=2. When the number of tasks is numerous, such scheduling achieves higher system utilization and less processor idle time than the greedy method.

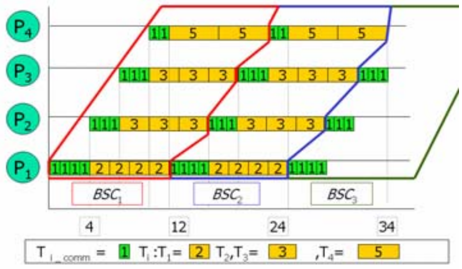


Fig. 1. Most Jobs First (*MJF*) task scheduling when $\delta \leq 1$.

<u>Lemma 1</u>: Given a master slave system with $\delta > 1$, in *MJF* scheduling, the amount of tasks being assigned to $P_{\max+1}$ can be calculated by the following equation,

$$task(P_{\max+1}) = (BSC - \sum_{i=1}^{\max} comm(P_i)) \;/\; T_{\max+1\_com} \tag{1}$$

<u>Lemma 2</u>: Given a master slave system with $\delta > 1$, in *MJF* scheduling, the period of processor $P_{\max+1}$ stays idle denoted by T_{idle}^{MJF} and can be calculated by the following equation,

$$T_{idle}^{MJF} = BSC - comm(P_{\max+1}) - comp(P_{\max+1}) \tag{2}$$

Another example of master slave task scheduling with identical communication (i.e., T_{i_comm}=1) and $\delta > 1$ is given in Fig. 2. Because $\delta > 1$, according to equation (1), we have $task(P_{\max+1}=P_4) = 10$. We note that P_4 completes its tasks and becomes available at time 100. However, the master processor dispatches tasks to P_3 during time 100 ~ 110 and starts to send tasks to P_4 at time 110. Such kind of idle situation also happens at time 100~110, 160~170, 220~230, and so on.

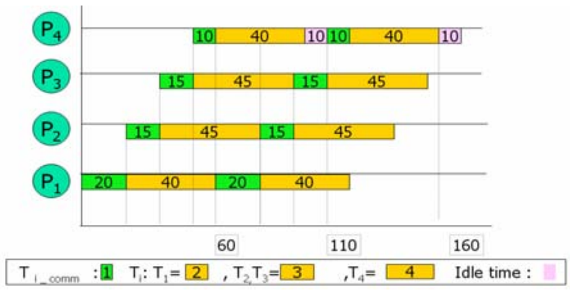


Fig. 2. Most Jobs First (*MJF*) Tasking when $\delta > 1$.

<u>Lemma 3</u>: In *MJF* scheduling algorithm with identical communication T_{i_comm}, when $\delta > 1$, the completion time of tasks in the j^{th} *BSC* can be calculated by the following equation.

$$T(BSC_j) = \sum_{i=1}^{\max} comm(P_i) + j \times (comm(P_{\max+1}) + comp(P_{\max+1}) + T_{idle}^{MJF}) \;\; - T_{idle}^{MJF} \tag{3}$$

4 Smallest Communication Ratio (*SCR*) Scheduling with Identical Communication

The *MJF* scheduling algorithm distributes tasks to slave processors according to processors' speed, namely, faster processor receives tasks first. In this section, we demonstrate an efficient task scheduling algorithm, Smallest Communication Ratio (*SCR*), focuses on master slave task scheduling with identical communication.

Lemma 4: In *SCR* scheduling algorithm, if $\delta \leq 1$ and T_{i_comm} are identical, the task completion time of the j^{th} *BSC* denoted by $T_{finish}^{SCR}(BSC_j)$, can be calculated by the following equation.

$$T_{finish}^{SCR}(BSC_j) = BSC + j \times (comm(P_1) + comp(P_1)) - comm(P_1) \tag{4}$$

Lemma 5: Given a master slave system with $\delta > 1$, in scheduling, the amount of tasks being assigned to $P_{\max+1}$ can be calculated by the following,

$$task(P_{\max+1}) = \frac{BSC}{T_{\max+1} + T_{\max+1\_comm}} \tag{5}$$

Lemma 6: In *SCR* scheduling algorithm, when $\delta > 1$, the idle time of a slave processor is denoted as T_{idle}^{SCR} and can be calculated by the following equation,

$$T_{idle}^{SCR} = \sum_{i=1}^{\max+1} comm(P_i) - BSC \tag{6}$$

The other case in Fig. 3 is to demonstrate the *SCR* scheduling method with dispersive idle when $\delta > 1$. We use the same example in Fig. 2 for the following illustration. Because $\delta > 1$, according to definition 10 and Lemma 5, we have $task(P_{\max+1}=P_4) = 12$. Comparing to the example in Fig. 2, P_4 stays 10 time units idle in *MJF* algorithm while the idle time is reduced and dispersed in *SCR* algorithm. In *SCR*, 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, showing a throughput of 0.6. While in *SCR*, there are 62 tasks completed during 102 time units. The throughput of *SCR* is 62/102 (≈0.61) > 0.6. Consequently, the *SCR* algorithm delivers higher system throughput.

Lemma 7: In *SCR* scheduling algorithm, if T_{i_comm} are identical for all slave processors and $\delta > 1$, the task completion time of the j^{th} *BSC* denoted by $T_{finish}^{SCR}(BSC_j)$, can be calculated by the following equation,

$$T_{finish}^{SCR}(BSC_j) = \sum_{i=1}^{\max+1} comm(P_i) + comp(P_1) +$$

$$(j-1) \times (comm(P_1) + comp(P_1) + T_{idle}^{SCR}) \tag{7}$$

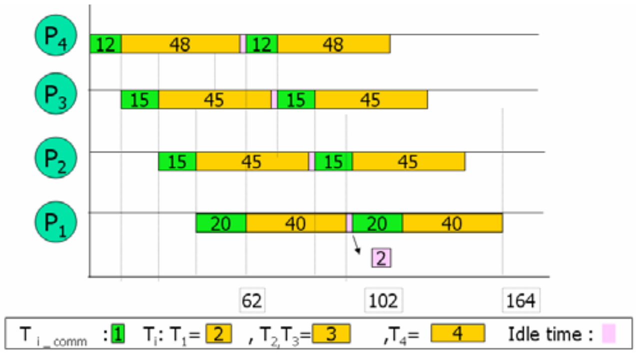


Fig. 3. Smallest Communication Ratio (*SCR*) Tasking when $\delta > 1$.

5 Generalized Smallest Communication Ratio (*SCR*)

As computational grid integrates geographically distributed computing resources, the communication overheads from resource broker / master computer to different computing site are different. Therefore, towards an efficient scheduling algorithm, the heterogeneous communication overheads should be considered. In this section, we present the *SCR* task scheduling techniques work on master slave computing paradigm with heterogeneous communication.

Lemma 8: Given a master slave system with heterogeneous communication and $\delta > 1$, in *MJF* scheduling, we have

$$task(P_{\max+1}) = \left\lfloor \frac{BSC - \sum_{i=1}^{\max} comm(P_i)}{T_{\max+1\_comm}} \right\rfloor \tag{8}$$

Lemma 9: Given an *SCR* scheduling with heterogeneous communication and $\delta > 1$, T_{idle}^{SCR} is the idle time of one slave processor, we have the following equation,

$$T_{idle}^{SCR} = \sum_{i=1}^{\max+1} comm(P_i) - BSC. \tag{9}$$

Lemma 10: Given an *SCR* scheduling with heterogeneous communication and $\delta > 1$, $T_{start}^{SCR}(BSC_j)$ is the start time to dispatch tasks in the j^{th} *BSC*, we have the following equation,

$$T_{start}^{SCR}(BSC_j) = (j-1) \times (BSC + T_{idle}^{SCR}) \tag{10}$$

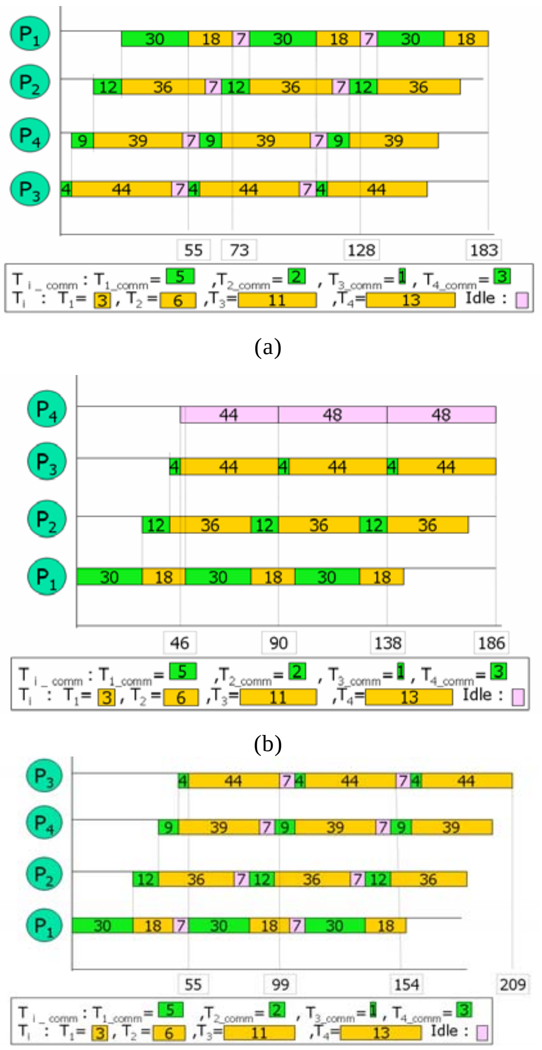
Lemma 11: Given an *SCR* scheduling with heterogeneous communication and $\delta > 1$, the task completion time of the j^{th} *BSC* denoted by $T_{finish}^{SCR}(BSC_j)$, we have

$$T_{finish}^{SCR}(BSC_j) = \sum_{i=1}^{\max+1} comm(P_i) + comp(P_k) + (j-1) \times (comm(P_k) + comp(P_k) + T_{idle}^{SCR}) \tag{11}$$

where P_k is the slave processor with maximum communication cost.

Another example of heterogeneous of communication with $\delta > 1$ master slave tasking is shown in Fig. 4(a). The communication overheads vary from 1 to 5. The computational speeds vary from 3 to 13. In this example, we have *BSC* = 48.

In *SCR* implementation, according to corollary 3, task distribution is $task(P_1) = 6$, $task(P_2) = 6$, $task(P_3) = 4$ and $task(P_{\max+1}) = task(P_4) = 3$. The communication costs of slave processors are $comm(P_1) = 30$, $comm(P_2) = 12$, $comm(P_3) = 4$ and $comm(P_4) = 9$, respectively. Therefore, the *SCR* method distributes tasks by the order P_3, P_4, P_2, P_1. There are 19 tasks in the first *BSC* dispatched to P_1 to P_4 during time period 1~55. Processor P_3 is the first processor to receive tasks and it finishes at time $t = 48$ and becomes available. In the meanwhile, processor P_1 receives tasks during $t = 48$~55. The second *BSC* starts to dispatch tasks at $t = 55$. Namely, P_3 starts to receive tasks at $t = 55$ in the second scheduling cycle. Therefore, P_3 has 7 unit of time idle. Lemmas 4 and 5 state the above phenomenon. The completion time of tasks in the first *BSC* depends on the finish time of processor P_1. We have $T_{finish}^{SCR}(BSC_1) = 73$.



(a)

(b)

(c)

Fig. 4. Task scheduling on heterogeneous communication environment with $\delta > 1$. (a) Smallest Communication Ratio (b) Most Job First (c) Largest communication ratio *(LCR).*

The *MJF* scheduling is depicted in Fig. 4(b). According to corollary 5, $task(P_{\max+1}) = task(P_4) = 0$, therefore, P_4 will not be included in the scheduling. MJF has the task distribution order P_1, P_2, P_3. Another scheduling policy is called *Longest Communication Ratio* (*LCR*) which is an opposite approach to the *SCR* method. Fig. 4(c) shows the *LCR* scheduling result which has the dispatch order P_1, P_2, P_4, P_3.

To investigate the performance of *SCR* scheduling technique, we observe that *MJF* algorithm completes 16 tasks in 90 units of time in the first *BSC*. On the other hand, in *SCR* scheduling, there are 19 tasks completed in 73 units of time in the first BSC. In *LCR*, there are 19 tasks completed in 99 units of time. We can see that the system throughput of *SCR* (19/73≈0.260) > *LCR* (19/99≈0.192) > *MJF* (16/90≈0.178). Moreover, the average turnaround time of the *SCR* algorithm in the first three *BSC*s is 183/57 (≈3.2105) which is less than the *LCR's* average turnaround time 209/57 (≈3.6666) and the *MJF*'s average turnaround time 186/48 (≈3.875).

6 Performance Evaluation

To evaluate the performance of the proposed method, we have implemented the *SCR* and the *MJF* algorithms. We compare different criteria, such as average turnaround time, system throughput and processor idle time, in Heterogeneous Processors with Heterogeneous Communications (*HPHC*).

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

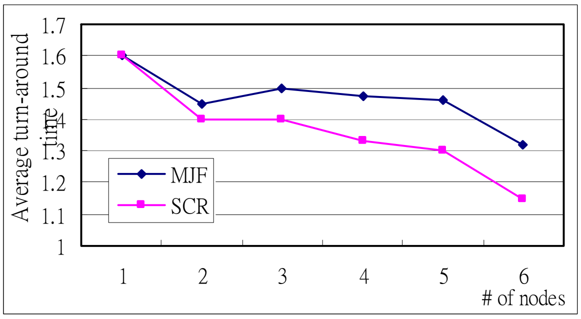
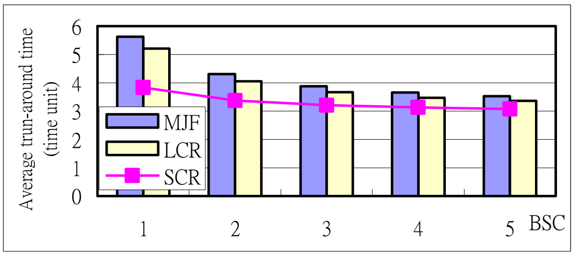
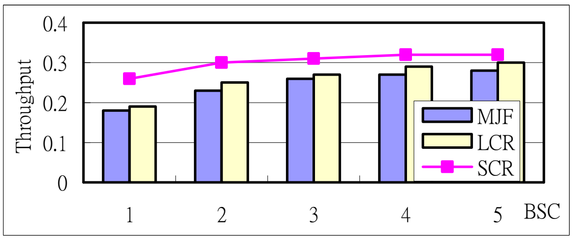


Fig. 5. Average task turn-around time on different numbers of processors.

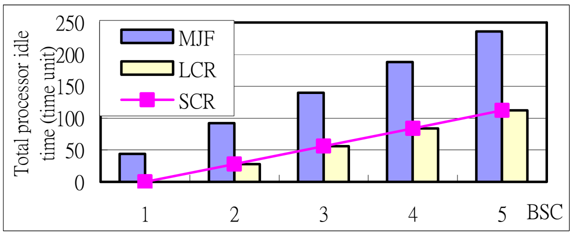
Simulation results present the performance comparison of three task scheduling algorithms, *SCR*, *MJF*, *LCR*, on heterogeneous processors and heterogeneous communication paradigms. Fig. 6 shows the simulation results for the experiment setting that with ±10 processor speed variation and ±4 communication speed variation. The computation speed of slave processors are T_1=3, T_2=6, T_3=11, and T_4=13. The time of a slave processor to receive one task from master processor are T_{1_comm} = 5, T_{2_comm} = 2, T_{3_comm} = 1 and T_{4_comm}=3. The average task turnaround time, system throughput and processor idle time are measured.



(a)



(b)

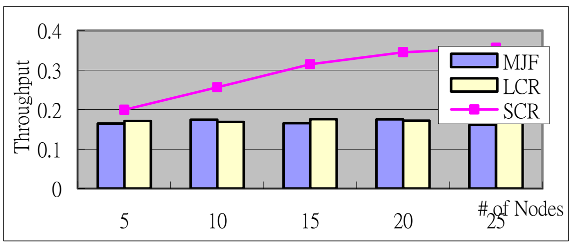


(c)

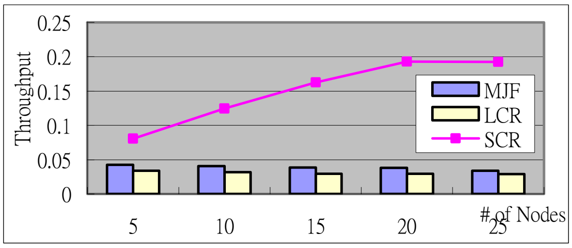
Fig. 6. Simulation results for 5 processors with ±10 computation speed variation and ±4 communication variation when $\delta > 1$ (a) average turnaround time (b) system throughput (c) processor idle time.

Fig. 6(a) is the average turnaround time within different number of *BSC*. The *SCR* algorithm performs better than the *LCR* and *MJF* method. Similarly, the *SCR* method has higher throughput than the other two algorithms as shown in Fig. 6(b). The processor idle time are estimated in Fig. 6(c). The *SCR a*nd *LCR* algorithms have the same period of processor idle time which is less than the *MJF* scheduling method. These phenomena match the theoretical analysis in section 5.

The miscellaneous comparison in Fig. 7 presents the performance comparison of *SCR*, *MJF* with more cases. The simulation results for the experiment setting that with ±5~±30 processor speed variation and ±5~±30 communication speed variation. The computation speed variation of $T_1 \sim T_n$=±5~±30. The communication speed variation of $T_{1\_comm} \sim T_{n\_comm}$=±5~±30. The system throughput is measured.



(a)



(b)

Fig. 7. Simulation results of throughput for the range of 5~25 processors with ±30 computation speed variation and ±30 communication variation in 100 cases and 100 *BSC* (a) system throughput of the cases when 0< $T_i \leq 30$ and 0< $T_{i\_comm} \leq 5$ (b) system throughput of the cases when 0< $T_i \leq 5$ and 0< $T_{i\_comm} \leq 30$.

Fig. 7(a) is the case of 0< $T_i \leq 30$, 0< $T_{i\_comm} \leq 5$ and the parameter of computation speed and communication speed are to be random and uniformly distributed within different number of nodes and 100 *BSC* for 100 cases. Fig. 7(b) is the case of 0< $T_i \leq 5$ and 0< $T_{i\_comm} \leq 30$. The *SCR* algorithm performs better than *MJF* method, and *SCR* method has higher throughput than the *MJF* algorithm as shown in Fig. 7(a) and Fig. 7(b). From the above experimental tests, we have the following remarks. The proposed *SCR* scheduling technique has better task turnaround time and higher system throughput than the *MJF* algorithm.

From the above experimental tests, we have the following remarks.

- The proposed *SCR* scheduling technique has higher system throughput than the *MJF* algorithm.
- The proposed *SCR* scheduling technique has better task turnaround time than the *MJF* algorithm.

The *SCR* scheduling technique has less processor idle time than the *MJF* algorithm.

7 Conclusions

The problem of resource management and scheduling has been one of main challenges in grid computing. In this paper, we have presented an efficient algorithm, *SCR* for heterogeneous processors tasking problem. One significant improvement of our approach is that average turnaround time could be minimized by selecting processor has the smallest communication ratio first. The other advantage of the proposed method is that system throughput can be increased via dispersing processor idle time. Our preliminary analysis and simulation results indicate that the *SCR* 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 that remains in this paper. Our proposed model can be applied to map tasks onto heterogeneous cluster systems in grid environments, in which the communication costs are various from clusters. In future, we intend to devote generalized tasking mechanisms for computational grid. We will study realistic applications and analyze their performance on grid system. Besides, rescheduling of processors / tasks for minimizing processor idle time on heterogeneous systems is also interesting and will be investigated.

References

1. O. Beaumont, A. Legrand and Y. Robert, "The Master-Slave Paradigm with Heterogeneous Processors," *IEEE Trans. on parallel and distributed systems*, Vol. 14, No.9, pp. 897-908, September 2003.
2. C. Banino, O. Beaumont, L. Carter, J. Ferrante, A. Legrand and Y. 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.
3. O. Beaumont, A. Legrand and Y. Robert, "Pipelining Broadcasts on Heterogeneous Platforms," *IEEE Trans. on parallel and distributed systems*, Vol. 16, No.4, pp. 300-313 April 2005.
4. 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.

5. O. Beaumont, V. Boudet, F. Rastello and Y. Robert, "Matrix-Matrix Multiplication on Heterogeneous Platforms," *Proc. Int'l Conf. Parallel Processing,* Vol. 12, No. 10, pp. 1033-1051, Oct. 2001.
6. F. Berman, R. Wolski, H. Casanova, W. Cirne, H. Dail, M. Faerman, S. Figueira, J. Hayes, G. Obertelli, J. Schopf, G. Shao, S. Smallen, N. Spring, A. Su, and D. Zagorodnov, "Adaptive Computing on the Grid Using AppLeS," *IEEE Trans. on parallel and distributed systems*, Vol. 14, No. 4, pp.369-379, April 2003.
7. 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.
8. 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.
9. 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.
10. 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.
11. K. Cooper, A. Dasgupta, K. Kennedy, C. Koelbel, A. Mandal, G. Marin, M. Mazina, J. Mellor-Crummey, F. Berman, H. Casanova, A. Chien, H. Dail, X. Liu, A. Olugbile, O. Sievert, H. Xia, L. Johnsson, B. Liu, M. Patel, D. Reed, W. Deng, C. Mendes, Z. Shi, A. YarKhan, J. Dongarra, "New Grid Scheduling and Rescheduling Methods in the GrADS Project," *Proceedings of the 18th International Parallel and Distributed Processing Symposium* (IPDPS'04), pp.209-229, April 2004.
12. H. Casanova, A. Legrand, D. Zagorodnov and F. Berman, "Heuristics for Scheduling Parameter Sweep applications in Grid environments," *Proceedings of the 9th Heterogeneous Computing workshop* (*HCW*'2000), pp. 349-363, 2000.
13. T. Thanalapati and S. 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.

行政院所屬各機關人員出國報告書提要

撰寫時間： 96 年 6 月 20 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	副教授	
出席國際會議名稱	2007 International Conference on Algorithms and Architecture for Parallel Processing, June 11 -14 2007.				
到達國家及地點	**Hangzhou, China**		出國期間	自 96 年 06 月 11 日 迄 96 年 06 月 19 日	
內容提要	這一次在杭州所舉行的國際學術研討會議共計四天。第一天下午本人抵達會場辦理報到。第二天各主持一場 invited session 的論文發表。同時，自己也在上午的場次發表了這依次被大會接受的論文。第一天也聽取了 Dr. Byeongho Kang 有關於 Web Information Management 精闢的演說。第二天許多重要的研究成果分為六個平行的場次進行論文發表。本人選擇了 Architecture and Infrastructure、Grid computing、以及 P2P computing 相關場次聽取報告。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，合影留念。第三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，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					

(出席 ICA3PP-07 研討會所發表之論文)

A Generalized Critical Task Anticipation Technique for DAG Scheduling

Ching-Hsien Hsu[1], Chih-Wei Hsieh[1] and Chao-Tung Yang[2]

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

[2] High-Performance Computing Laboratory
Department of Computer Science and Information Engineering
Tunghai University, Taichung City, 40704, Taiwan R.O.C.
ctyang@thu.edu.tw

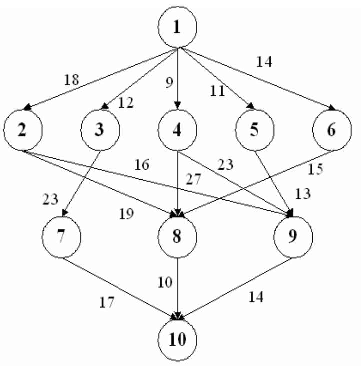
Abstract. The problem of scheduling a weighted directed acyclic graph (DAG) representing an application to a set of heterogeneous processors to minimize the completion time has been recently studied. The NP-completeness of the problem has instigated researchers to propose different heuristic algorithms. In this paper, we present a Generalized Critical-task Anticipation (*GCA*) algorithm for DAG scheduling in heterogeneous computing environment. The *GCA* scheduling algorithm employs task prioritizing technique based on *CA* algorithm and introduces a new processor selection scheme by considering heterogeneous communication costs among processors for adapting grid and scalable computing. To evaluate the performance of the proposed technique, we have developed a simulator that contains a parametric graph generator for generating weighted directed acyclic graphs with various characteristics. We have implemented the *GCA* algorithm along with the *CA* and *HEFT* scheduling algorithms on the simulator. The *GCA* algorithm is shown to be effective in terms of speedup and low scheduling costs.

1. Introduction

The purpose of heterogeneous computing system is to drive processors cooperation to get the application done quickly. Because of diverse quality among processors or some special requirements, like exclusive function, memory access speed, or the customize I/O devices, etc.; tasks might have distinct execution time on different resources. Therefore, efficient task scheduling is important for achieving good performance in heterogeneous systems.

The primary scheduling methods can be classified into three categories, dynamic scheduling, static scheduling and hybrid scheduling according to the time at which the scheduling decision is made. In dynamic approach, the system performs redistribution of tasks between processors during run-time, expect to balance computational load, and reduce processor's idle time. On the contrary, in static approach, information of applications, such as tasks execution time, message size of communications among tasks, and tasks dependences are known a priori at compile-time; tasks are assigned to processors accordingly in order to minimize the entire application completion time and satisfy the precedence of tasks. Hybrid scheduling techniques are mix of dynamic and static methods, where some preprocessing is done statically to guide the dynamic scheduler [8].

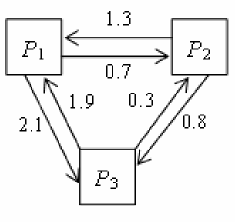
A Direct Acyclic Graph (DAG) [2] is usually used for modeling parallel applications that consists a number of tasks. The nodes of DAG correspond to tasks and the edges of which indicate the precedence constraints between tasks. In addition, the weight of an edge represents communication cost between tasks. Each node is given a computation cost to be performed on a processor and is represented by a computation costs matrix. Figure 1 shows an example of the model of DAG scheduling. In Figure 1(a), it is assumed that task n_j is a successor (predecessor) of task n_i if there exists an edge from n_i to n_j (from n_j to n_i) in the graph. Upon task precedence constraint, only if the predecessor n_i completes its execution and then its successor n_j receives the *messages* from n_i, the successor n_j can start its execution. Figure 1(b) demonstrates different computation costs of task that performed on heterogeneous processors. It is also assumed that tasks can be executed only on single processor with non-preemptable style. A simple fully connected processor network with asymmetrical data transfer rate is shown in Figures 1(c) and 1(d).



(a)

	P_1	P_2	P_3	$\overline{w_i}$
n_1	14	19	9	14
n_2	13	19	18	16.7
n_3	11	17	15	14.3
n_4	13	8	18	13
n_5	12	13	10	11.7
n_6	12	19	13	14.7
n_7	7	16	11	11
n_8	5	11	14	10
n_9	18	12	20	16.7
n_{10}	17	20	11	16

(b)



(c)

SP \ DP	P_1	P_2	P_3
P_1	0	0.7	2.1
P_2	1.3	0	0.8
P_3	1.9	0.3	0

(d)

Figure 1: An example of DAG scheduling problem (a) Directed Acyclic Graph (DAG-1) (b) computation cost matrix (*W*) (c) processor topology (d) communication weight.

The scheduling problem has been widely studied in heterogeneous systems where the computational ability of processors is different and the processors communicate over an underlying network. Many researches have been proposed in the literature. The scheduling problem has been shown to be NP-complete [3] in general cases as well as in several restricted cases; so the desire of optimal scheduling shall lead to higher scheduling overhead. The negative result motivates the requirement for heuristic approaches to solve the scheduling problem. A comprehensive survey about static scheduling algorithms is given in [9]. The authors of have shown that the heuristic-based algorithms can be classified into a variety of categories, such as clustering algorithms, duplication-based algorithms, and list-scheduling algorithms. Due to page limitation, we omit the description for related works.

In this paper, we present a Generalized Critical task Anticipation (*GCA*) algorithm, which is an approach of list scheduling for DAG task scheduling problem. The main contribution of this paper is proposing a novel heuristic for DAG scheduling on heterogeneous machines and networks. A significant improvement is that inter-processor communication costs are considered into processor selection phase such that tasks can be mapped to more suitable processors. The *GCA* heuristic is compared favorable with previous *CA* [5] and *HEFT* heuristics in terms of schedule length and speedup under different parameters.

The rest of this paper is organized as follows: Section 2 provides some background, describes preliminaries regarding heterogeneous scheduling system in DAG model and formalizes the research problem. Section 3 defines notations and terminologies used in this paper. Section 4 forms the main body of the paper, presents the Generalized Critical task Anticipation (*GCA*) scheduling algorithm and illustrating it with an example. Section 5 discusses performance of the proposed heuristic and its simulation results. Finally, Section 6 briefly concludes this paper.

2. DAG Scheduling on Heterogeneous Systems

The DAG scheduling problem studied in this paper is formalized as follows. Given a parallel application represented by a DAG, in which nodes represent tasks and edges represent dependence between these tasks. The target computing architecture of DAG scheduling problem is a set of heterogeneous processors, $M = \{P_k: k = 1: P\}$ and $P = |M|$, communicate over an underlying network which is assumed fully connected. We have the following assumptions:

- Inter-processor communications are performed without network contention between arbitrary processors.
- Computation of tasks is in non-preemptive style. Namely, once a task is assigned to a processor and starts its execution, it will not be interrupted until its completion.
- Computation and communication can be worked simultaneously because of the separated I/0.
- If two tasks are assigned to the same processor, the communication cost between the two tasks can be discarded.
- A processor is assumed to send the computational results of tasks to their immediate successor as soon as it completes the computation.

Given a DAG scheduling system, W is an $n \times P$ matrix in which $w_{i,j}$ indicates estimated computation time of processor P_j to execute task n_i. The mean execution time of task n_i can be calculated by the following equation:

$$\overline{w_i} = \sum_{j=1}^{P} \frac{w_{i,j}}{P} \qquad \textbf{(1)}$$

Example of the mean execution time can be referred to Figure 1(b).

For communication part, a $P \times P$ matrix T is structured to represent different data transfer rate among processors (Figure 1(d) demonstrates the example). The communication cost of transferring data from task n_i (execute on processor p_x) to task n_j (execute on processor p_y) is denoted by $c_{i,j}$ and can be calculated by the following equation,

$$c_{i,j} = V_m + Msg_{i,j} \times t_{x,y}\ , \qquad (2)$$

Where:
V_m is the communication latency of processor P_m,
$Msg_{i,j}$ is the size of message from task n_i to task n_j,
$t_{x,y}$ is data transfer rate from processor p_x to processor p_y, $1 \le x, y \le P$.

In static DAG scheduling problem, it was usually to consider processors' latency together with its data transfer rate. Therefore, equation (2) can be simplified as follows,

$$c_{i,j} = Msg_{i,j} \times t_{x,y}\ , \qquad (3)$$

Given an application represented by Directed Acyclic Graph (DAG), G = (V, E), where $V = \{n_j: j = 1: v\}$ is the set of nodes and $v = |V|$; $E = \{e_{i,j} = <n_i, n_j>\}$ is the set of communication edges and $e = |E|$. In this model, each node indicates least indivisible task. Namely, each node must be executed on a processor from the start to its completion. Edge $<n_i, n_j>$ denotes precedence of tasks n_i and n_j. In other words, task n_i is the immediate predecessor of task n_j and task n_j is the immediate successor of task n_i. Such precedence represents that task n_j can be start for execution only upon the completion of task n_i. Meanwhile, task n_j should receive essential message from n_i for its execution. Weight of edge $<n_i, n_j>$ indicates the average communication cost between n_i and n_j.

Node without any inward edge is called *entry node*, denoted by n_{entry}; while node without any outward edge is called *exit node*, denoted by n_{exit}. In general, it is supposed that the application has only one *entry node* and one *exit node*. If the actual application claims more than one *entry* (*exit*) *node*, we can insert a dummy *entry* (*exit*) *node* with zero-cost edge.

3. Preliminaries

This study concentrates on list scheduling approaches in DAG model. List scheduling was usually distinguished into list phase and processor selection phase. Therefore, priori to discuss the main content, we first define some notations and terminologies used in both phases in this section.

3.1 Parameters for List Phase

Definition 1: Given a DAG scheduling system on $G = (V, E)$, the *Critical Score* of task n_i denoted by $CS(n_i)$ is an accumulative value that are computed recursively traverses along the graph upward, starting from the exit node. $CS(n_i)$ is computed by the following equations,

$$CS(n_i) = \begin{cases} \overline{w_{exit}} & \text{if } n_i \text{ is the exit ndoe (i.e. } n_i = n_{exit}) \\ \overline{w_i} + \underset{n_j \in suc(n_i)}{Max}(\overline{c_{i,j}} + CS(n_j)) & \text{otherwise} \end{cases} \quad (4)$$

where $\overline{w_{exit}}$ is the average computation cost of task n_{exit}, $\overline{w_i}$ is the average computation cost of task n_i, $suc(n_i)$ is the set of immediate successors of task n_i,
$\overline{c_{i,j}}$ is the average communication cost of edge <n_i, n_j> which is defined as follows,

$$\overline{c_{i,j}} = \frac{Msg_{i,j} \times \sum_{1 \le x,y \le P} t_{x,y}}{(P^2 - P)}, \quad (5)$$

3.2 Parameters for Processor Selection Phase

Most algorithms in processor selection phase employ a partial schedule scheme to minimize overall schedule length of an application. To achieve the partial optimization, an intuitional method is to evaluate the *finish time* (*FT*) of task n_i executed on different processors. According to the calculated results, one can select the processor who has minimum finish time as target processor to execute the task n_i. In such approach, each processor P_k will maintain a list of tasks, *task-list*(P_k), keeps the latest status of tasks correspond to the $EFT(n_i, P_k)$, the earliest finish time of task n_i that is assigned on processor P_k.

Recall having been mentioned above that the application represented by DAG must satisfy the precedence relationship. Taking into account the precedence of tasks in DAG, a task n_j can start to execute on a processor P_k only if its all immediate predecessors send the essential messages to n_j and n_j successful receives all these messages. Thus, the latest message arrive time of node n_j on processor P_k, denoted by $LMAT(n_j, P_k)$, is calculated by the following equation,

$$LMAT(n_j, P_k) = \underset{n_i \in pred(n_j)}{Max}(EFT(n_i) + c_{u,k}, \text{ for task } n_i \text{ executed on processor } P_u) \quad (6)$$

where $pred(n_j)$ is the set of immediate predecessors of task n_j. Note that if tasks n_i and n_j are assigned to the same processor, $c_{u,k}$ is assumed to be zero because it is negligible.

Because the entry task n_{entry} has no inward edge, thus we have

$$LMAT(n_{entry}, P_k) = 0 \quad (7)$$

for all $k = 1$ to P.

Definition 2: Given a DAG scheduling system on $G = (V, E)$, the *Start Time* of task n_j executed on processor P_k is denoted as $ST(n_j, P_k)$.

Estimating task's start time (for example, task n_j) will facilitate search of available time slot on target processors that is large enough to execute that task (i.e., length of time slot > $w_{j,k}$). Note that the search of available time slot is started from $LMAT(n_j, P_k)$.

Definition 3: Given a DAG scheduling system on $G = (V, E)$, the *finish time* of task n_j denoted by $FT(n_j, P_k)$, represents the completion time of task n_j executed on processor

P_k. $FT(n_j, P_k)$ is defined as follows,

$$FT(n_j, P_k) = ST(n_j, P_k) + w_{j,k} \tag{8}$$

Definition 4: Given a DAG scheduling system on $G = (V, E)$, the *earliest finish time* of task n_j denoted by $EFT(n_j)$, is formulated as follows,

$$EFT(n_j) = \underset{p_k \in P}{Min}\{FT(n_j, P_k)\} \tag{9}$$

Definition 5: Based on the determination of $EFT(n_j)$ in equation (9), if the earliest finish time of task n_j is obtained upon task n_j executed on processor p_t, then the target processor of task n_j is denoted by $TP(n_j)$, and $TP(n_j) = p_t$.

4. The Generalized Critical-task Anticipation Scheduling Algorithm

Our approach takes advantages of list scheduling in lower algorithmic complexity and superior scheduling performance and furthermore came up with a novel heuristic algorithm, the generalized critical task anticipation (*GCA*) scheduling algorithm to improve the schedule length as well as speedup of applications. The proposed scheduling algorithm will be verified beneficial for the readers while we delineate a sequence of the algorithm and show some example scenarios in three phases, prioritizing phase, listing phase and processor selection phase.

In prioritizing phase, the $CS(n_i)$ is known as the maximal summation of scores including the average computation cost and communication cost from task n_i to the exit task. Therefore, the magnitude of the task's critical score is regarded as the decisive factor when determining the priority of a task. In listing phase, an ordered list of tasks should be determined for the subsequent phase of processor selection. The proposed *GCA* scheduling technique arranges tasks into a list *L*, not only according to critical scores but also considers tasks' importance.

Several observations bring the idea of *GCA* scheduling method. Because of processor heterogeneity, there exist variations in execution cost from processor to processor for same task. In such circumstance, tasks with larger computational cost should be assigned higher priority. This observation aids some critical tasks to be executed earlier and enhances probability of tasks reduce its finish time. Furthermore, each task has to receive the essential messages from its immediate predecessors. In other words, a task will be in waiting state when it does not collect complete message yet. For this reason, we emphasize the importance of the last arrival message such that the succeeding task can start its execution earlier. Therefore, it is imperative to give the predecessor who sends the last arrival message higher priority. This can aid the succeeding task to get chance to advance the start time. On the other hand, if a task n_i is inserted into the front of a scheduling list, it occupies vantage position. Namely, n_i has higher probability to accelerate its execution and consequently the start time of $suc(n_i)$ can be advanced as well.

In most list scheduling approaches, it was usually to demonstrate the algorithms in two phases, the list phase and the processor selection phase. The list phase of proposed *GCA* scheduling algorithm consists of two steps, the *CS* (critical score) calculation step and task prioritization step.

Let's take examples for the demonstration of *CS* calculation, which is performed in level order and started from the deepest level, i.e., the level of exit task. For example, according to equation (4), we have $CS(n_{10}) = \overline{w_{10}} = 16$. For the upper level tasks, n_7, n_8 and n_9, $CS(n_7)$ = $\overline{w_7} + (\overline{c_{7,10}} + CS(n_{10}))$ = 47.12, $CS(n_8)$ = $\overline{w_8} + (\overline{c_{8,10}} + CS(n_{10}))$ =37.83, $CS(n_9)$ = $\overline{w_9} + (\overline{c_{9,10}} + CS(n_{10}))$ =49.23. The other tasks can be calculated by the same methods. Table 1 shows complete calculated critical scores of all tasks for DAG-1.

Table 1: Critical Scores of tasks in DAG-1 using *GCA* algorithm

***Critical Scores* of tasks in *GCA* algorithm**									
n_1	n_2	n_3	n_4	n_5	n_6	n_7	n_8	n_9	n_{10}
120.13	84.83	88.67	89.45	76.28	70.25	47.12	37.83	49.23	16.00

Follows the critical score calculation, the *GCA* scheduling method considers both tasks' importance (i.e., critical score) and its relative urgency for prioritizing tasks. Based on the results obtained previously, we use the same example to demonstrate task prioritization in *GCA*. Let's start at the exit task n_{10}, which has the lowest critical score. Assume that tasks will be arranged into an ordered list L, therefore, we have $L = \{n_{10}\}$ initially. Because task n_{10} has three immediate predecessors, with the order $CS(n_9) > CS(n_7) > CS(n_8)$, the list L will be updated to $L=\{n_9, n_7, n_8, n_{10}\}$. Applying the same prioritizing method by taking the front element of L, task n_9; because task n_9 has three immediate predecessors, with the order $CS(n_4) > CS(n_2) > CS(n_5)$, we have the updated list $L = \{ n_4, n_2, n_5, n_9, n_7, n_8, n_{10}\}$. Taking the same operations, insert task n_1 in front of task n_4, insert task n_3 in front of task n_7, insert tasks n_4, n_2, n_6 (because $CS(n_4) > CS(n_2) > CS(n_6)$) in front of task n_8; we have the list L = $\{ n_1, n_4, n_2, n_5, n_9, n_3, n_7, n_6, n_4, n_2, n_6, n_8, n_{10}\}$. The final list $L = \{n_1, n_4, n_2, n_5, n_9, n_3, n_7, n_6, n_8, n_{10}\}$ can be derived by removing duplicated tasks.

In listing phases, the *GCA* scheduling algorithm proposes two enhancements from the majority of literatures. First, *GCA* scheduling technique considers various transmission costs of messages among processors into the calculation of critical scores. Second, the *GCA* algorithm prioritizes tasks according to the influence on its successors and devotes to lead an accelerated chain while other techniques simply schedule high critical score tasks with higher priority. In other words, the *GCA* algorithm is not only prioritizing tasks by its importance but also by the urgency among task. The prioritizing scheme of *GCA* scheduling technique can be accomplished by using simple stack operations, push and pop, which are outlined in *GCA_List_Phase* procedure as follows.

```
Begin_GCA_List_Phase
    Initially, construct an array of Boolean QV and a stack S.
    QV[n_j] = false, ∀ n_j ∈ V.
    Push n_exit on top of S.
    While S is not empty do
        Peek task n_j on the top of S;
        If( all QV[n_i] are true, for all n_i ∈ pred(n_j) or task n_j is n_entry)    {
            Pop task n_j from top of S and put n_j into scheduling list L;
            QV[ n_j] = true; }
        Else      /* search the CT(n_j) */
            For each task n_i, where n_i ∈ pred(n_j) do
                If(QV[n_i] = false)
```

```
            Put CS(n_i) into container C;
         Endif
      Push tasks pred(n_j) from C into S by non-decreasing order according to their
   critical scores;
      Reset C to empty;
      /* if there are 2+ tasks with same CS(n_i), task n_i is randomly pushed into S.
   EndWhile
End_GCA_List_Phase
```

In processor-selection phase, tasks will be deployed from list L that obtained in listing phase to suitable processor in FIFO manner. According to the ordered list $L = \{n_1, n_4, n_2, n_5, n_9, n_3, n_7, n_6, n_8, n_{10}\}$, we have the complete calculated *EFTs* of tasks in DAG-1 and the schedule results of *GCA* algorithm are listed in Table 2 and Figure 2(a), respectively.

Table 2: Earliest Finish Time of tasks in DAG-1 using *GCA* algorithm

***Earliest Finish Time* of tasks in *GCA* algorithm**									
n_1	n_2	n_3	n_4	n_5	n_6	n_7	n_8	n_9	n_{10}
9	27	42	19.7	32.7	47.6	53	65.7	54.7	84.7

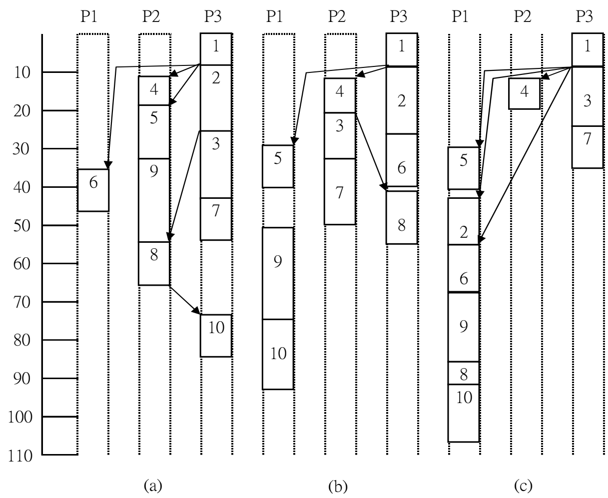


Figure 2: Schedule results of three algorithms on DAG-1 (a) *GCA* (makespan = 84.7) (b) *CA* (makespan = 92.4) (c) *HEFT* (makespan = 108.2).

In order to profile significance of the *GCA* scheduling technique, the schedule results of other algorithms, *CA* and *HEFT* are depicted in Figure 2(b) and 2(c), respectively. The *GCA* scheduling techniques incorporates the consideration of heterogeneous communication costs among processors in processor selection phase. Such enhancement facilitates the selection of best candidate of processors to execute specific tasks.

5. Performance Evaluation

5.1 Random Graph Generator

We implemented a Random Graph Generator (RGG) to simulate application graphs with various characteristics. RGG uses the following input parameters to produce diverse graphs.

- Weight of graph (*weight*), which is a constant = {32, 128, 512, 1024}.
- Number of tasks in the graph (n), where n = {20, 40, 60, 80, 100}.
- Graph parallelism (p), the graph parallelism determines shape of a graph. p is assigned for 0.5, 1.0 and 2.0. The level of graph is defined as $\lfloor \sqrt{v} / p \rfloor$. For example, graph with p = 2.0 has higher parallelism than graph with p = 1.0.
- Out degree of a task (d), where d = {1, 2, 3, 4, 5}. The out degree of a task indicates relationship with other tasks, the larger degree of a task the higher task dependence.
- Heterogeneity (h), determines computational cost of task n_i executed on processor P_k, i.e., $w_{i,k}$, which is randomly generated by the following formula.

$$w_i \times \left(1 - \frac{h}{2}\right) \le w_{i,k} \le w_i \times \left(1 + \frac{h}{2}\right). \quad (10)$$

RGG randomizes w_i from the interval [1, *weight*]. Note that larger value of *weight* represents the estimation is with higher precision. In our simulation, h was assigned by 0.1, 0.25, 0.5, 0.75 and 1.0.

- Communication to Computation Ratio (*CCR*), where *CCR* = {0.1, 0.5, 1, 2, 10}.

5.2 Comparison Metrics

As mentioned earlier, the objective of DAG scheduling problem is to minimize the completion time of an application. To verify the performance of a scheduling algorithm, several comparative metrics are given below for comparison:

- *Makespan*, also known as schedule length, which is defined as follows,

$$Makespan = \max(EFT(n_{exit})) \quad (11)$$

- *Speedup*, defined as following equation,

$$Speedup = \frac{\min_{P_j \in M}\{\sum_{n_i \in V} w_{i,j}\}}{makespan}, \text{ where } M \text{ is the set of processors} \quad (12)$$

The numerator is the minimal accumulated sum of computation cost of tasks which are assigned on one processor. Equation (12) represents the ratio of sequential execution time to parallel execution time.

- Percentage of Quality of Schedules (PQS)

The percentage of the *GCA* algorithm produces better, equal and worse quality of schedules compared to other algorithms.

5.3 Simulation Results

The first evaluation aims to demonstrate the merit of the *GCA* algorithm by showing quality of schedules using RGG. Simulation results were obtained upon different parameters with totally 1875 DAGs. Figure 3 reports the comparison by setting different *weight* = {32, 128, 512, 1024}. The term “Better” represents percentage of testing samples the *GCA* algorithm outperforms the *CA* algorithm. The term “Equal” represents both algorithm have same makespan in a given DAG. The tem “Worse” represents opposite results to the “Better” cases. Figure 4 gives the PQS results by setting different number of processors. Overall, the *GCA* scheduling algorithm presents superior performance for 65% test samples.

Speedup of the *GCA*, *CA* and *HEFT* algorithms to execute 1875 DAGs with fix processor number (P=16) under different number of task (n) are shown in Figure 5. The speedup of these algorithms show placid when number of task is small and increased significantly when number of tasks becomes large. In general, the *GCA* algorithm has better speedup than the other two algorithms. Improvement rate of the *GCA* algorithm in terms of average speedup is about 7% to the *CA* algorithm and 34% to the *HEFT* algorithm. The improvement rate (IR_{GCA}) is estimated by the following equation:

$$IR_{GCA} = \frac{\sum Speedup(GCA) - \sum Speedup(HEFT\ or\ CA)}{\sum Speedup(HEFT\ or\ CA)} \tag{13}$$

weight	32	128	512	1024
Better	65.33%	61.13%	67.07%	67.47%
Equal	34.40%	38.87%	32.93%	32.53%
Worse	0.27%	0%	0%	0%

Figure 3: PQS: *GCA* compared with CA (3 processors)

processor	5	6	7	8
Better	61.13%	72.33%	63.27%	66.60%
Equal	38.87%	27.67%	36.73%	33.40%
Worse	0%	0%	0%	0%

Figure 4: PQS: *GCA* compared with *CA* (*weight* = 128)

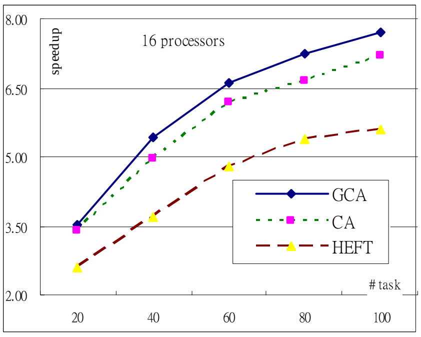


Figure 5: Speedup of *GCA*, *CA* and *HEFT* with different number of tasks (n).

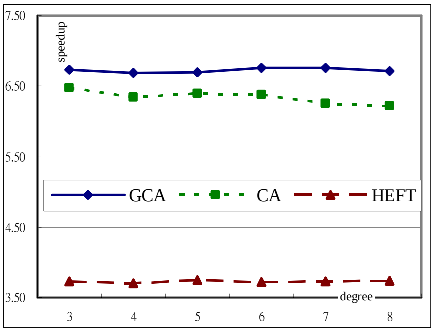
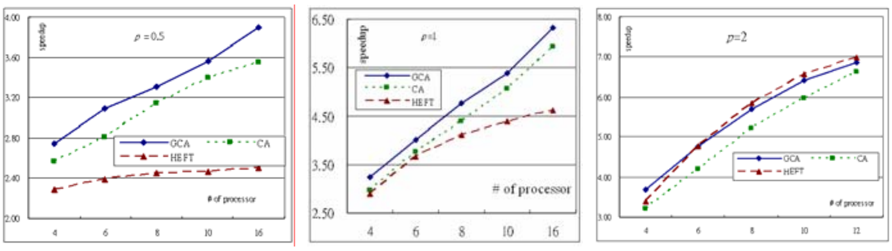


Figure 6: Speedup of *GCA*, *CA* and *HEFT* with different out-degree of tasks (*d*)

Speedup of the *GCA*, *CA* and *HEFT* algorithms to execute different DAGs with fix processor number (*P*=16) and task number (*n*=60) under different out-degree of tasks (*d*) are shown in Figure 6. The results of Figure 6 demonstrate the speedup influence by task dependence. We observe that speedups of scheduling algorithms are less dependent on tasks' dependence. Although the speedups of three algorithms are stable, the *GCA* algorithm outperforms the other two algorithms in most cases. Improvement rate of the *GCA* algorithm in terms of average speedup is about 5% to the *CA* algorithm and 80% to the *HEFT* algorithm.

Figure 7 shows simulation results of three algorithms upon different processor number and degree of parallelization. It is noticed that, graphs with larger value of *p* tends to with higher parallelism. As shown in Figures 7(a) and (b), the *GCA* algorithm performs well in linear graphs (*p*=0.5) and general graphs (*p*=1.0). On the contrary, Figure 7(c) shows that the *HEFT* scheduling algorithm has superior performance when degree of parallelism is high. In general, for graphs with low parallelism (e.g., *p* = 0.5), the *GCA* algorithm has 33% improvement rate in terms of average speedup compare to the *HEFT* algorithm; for graphs with normal parallelism (e.g., *p* = 1), the *GCA* algorithm has 20% improvement rate. For graphs with high parallelism (e.g., *p* = 2), the *GCA* algorithm performs worse than the *HEFT* by 3% performance.

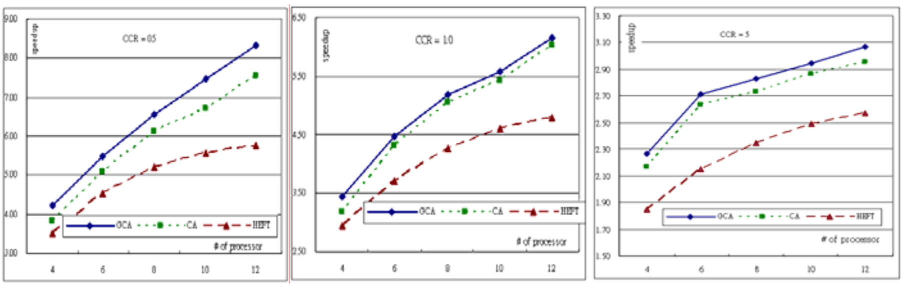
Speedup of the *GCA*, *CA* and *HEFT* algorithms to execute different DAGs with fix processor number (*P*=16) and task number (*n*=60) under different out-degree of tasks (*d*) are shown in Figure 6. The results of Figure 6 demonstrate the speedup influence by task dependence. We observe that speedups of scheduling algorithms are less dependent on tasks' dependence. Although the speedups of three algorithms are stable, the *GCA* algorithm outperforms the other two algorithms in most cases. Improvement rate of the *GCA* algorithm in terms of average speedup is about 5% to the *CA* algorithm and 80% to the *HEFT* algorithm.



(a) (b) (c)

Figure 7: Speedup with different degree of parallelism (*p*) (a) *p* = 0.5 (b) *p* = 1 (c) *p* = 2.

The impact of communication overheads on speedup are plotted in Figure 8 by setting different value of *CCR*. It is noticed that increase of *CCR* will downgrade the speedup we can obtained. For example, speedup offered by *CCR* = 0.1 has maximal value 8.3 in *GCA* with 12 processors; for *CCR* = 1.0, the *GCA* algorithm has maximal speedup 6.1 when processor number is 12; and the same algorithm, *GCA*, has maximal speedup 3.1 for *CCR* = 5 with 12 processors. This is due to the fact that when communication overheads higher than computational overheads, costs for tasks migration will offset the benefit of moving tasks to faster processors.



(a) (b) (c)

Figure 8: Speedup results with different *CCR* (a) *CCR*=0.5 (b) *CCR* = 1 (c) *CCR* = 5.

6. Conclusions

The problem of scheduling a weighted directed acyclic graph (DAG) to a set of heterogeneous processors to minimize the completion time has been recently studied. Several techniques have been presented in the literature to improve performance. This paper presented a general Critical-task Anticipation (*GCA*) algorithm for DAG scheduling system. The *GCA* scheduling algorithm employs task prioritizing technique based on *CA* algorithm and introduces a new processor selection scheme by considering heterogeneous communication costs among processors. *GCA* scheduling algorithm is a list scheduling approach with simple data structure and profitable for grid and scalable computing. Experimental results show that *GCA* has superior performance compare to the well known *HEFT* scheduling heuristic algorithm and our previous proposed *CA* algorithm which did not incorporate the consideration of heterogeneous communication costs into processor selection phase. Experimental results show that *GCA* is equal or superior to *HEFT* and *CA* scheduling algorithms in most cases and it enhances to fit more real grid system.

Acknowledgements

This paper is based upon work supported by National Science Council (NSC), Taiwan, under grants no. NSC95-2213-E-216-006. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSC.

References

[1] R. Bajaj and D. P. Agrawal, "Improving Scheduling of Tasks in a Heterogeneous Environment," *IEEE Trans. on PDS*, vol. 15, no. 2, pp. 107-118, 2004.

[2] S. Behrooz, M. Wang, and G. Pathak, "Analysis and Evaluation of Heuristic Methods for Static Task Scheduling," *Jounal of Parallel and Distributed Computing*, vol. 10, pp. 222-232, 1990.

[3] M.R Gary and D.S. Johnson, "Computers and Interactability: A guide to the Theory of NP-Completeness", W.H. Freeman and Co., 1979.

[4] T. Hagras and J. Janecek," A High Performance, Low Complexity Algorithm for Compile-Time Task Scheduling in Heterogeneous Systems," *Parallel Computing*, vol. 31, Issue 7, pp. 653-670, 2005.

[5] Ching-Hsieh Hsu and Ming-Yuan Weng, "An Improving Critical-Task Anticipation Scheduling Algorithm for Heterogeneous Computing Systems", Proceedings of the Eleventh Asia-Pacific Computer Systems Architecture Conference, LNCS 4186, pp. 97-110, 2006.

[6] E. Ilavarasan P. Thambidurai and R. Mahilmannan, "Performance Effective Task Scheduling Algorithm for Heterogeneous Computing System,**"** *IEEE Proceedings of IPDPS*, pp. 28-38, 2005.

[7] S. Ranaweera and D. P. Agrawal, "A Task Duplication Based Scheduling Algorithm for Heterogeneous Systems," *IEEE Proceedings of IPDPS*, pp. 445-450, 2000.

[8] Rizos Sakellariou and Henan Zhao, "A Hybrid Heuristic for DAG Scheduling on Heterogeneous Systems", Proc. of the *IEEE IPDPS* Workshop 1, pp. 111b, 2004.

[9] H. Topcuoglu, S. Hariri and W. Min-You, "Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing," *IEEE Transactions on PDS*, vol.13, no. 3, pp. 260-274, 2002.

行政院所屬各機關人員出國報告書提要

撰寫時間： 97 年 4 月 20 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		副教授	
出席國際會議名稱	The 22nd International Conference on Advanced Information Networking and Applications (AINA-08), March 25 -28 2008.				
到達國家及地點	**Okinawa, Japan**	出國期間	自 97 年 03 月 25 日 迄 97 年 03 月 28 日		

內容提要

一、主要任務摘要（五十字以內）

AINA-08 是網路相關研究領域一個大型的研討會。這一次參與AINA-08除了發表相關研究成果以外，也在會場上看到許多新的研究成果與方向。此外，也與許多學術界的朋友交換研究心得。

二、對計畫之效益（一百字以內）

這一次參與 AINA-08 除了發表我們在此一計劃最新的研究成果以外，也在會場中，向多位國內外學者解釋我們的研究內容，彼此交換研究心得。除了讓別的團隊知道我們的研究方向與成果，我們也可以學習他人的研究經驗。藉此，加強國際合作，提升我們的研究質量。

三、經過

這一次在 Okinawa 所舉行的國際學術研討會議共計四天。第一天是 Workshop Program。第二天，由 Dr. Michel Raynal 的專題演講， “Synchronization is Coming Back, But is it the Same?” 作為研討會的開始。緊接著是五個平行的場次，分為上下午進行。本人全程參與研討會的議程。晚上在大會的地點舉行歡迎晚宴。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，合影留念。第三天，專題演講是由 Dr. Shigeki Yamada 針對 “Cyber Science Infrastructure (CSI) for Promoting Research Activities of Academia and Industries in Japan”發表演說。本人也參

與的第三天全部的大會議程。晚宴，大會安排交通車到市郊一個花園餐廳舉行。最後一天，本人亦參與了所有的場次，並且發表了這一次的論文。本人主要聽取 GRID 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，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。四天下來，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：無線網路技術、網路安全、GRID、資料庫以及普及運算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。

四、心得

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。

五、建議與結語

出席國際會議，註冊費越來越貴(AINA-08 約兩萬元)，若會議在亞州舉行，補助的經費免強足夠，但是若在歐美，經費往往不足。降低同學參與歐美的會議。

大會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。

六、攜回資料

論文集光碟片

七、出國行程表

3/25 前往 Okinawa　下午研討會報到，參與 AINA-08 Workshop Progra,

3/26 全日參與研討會

3/27 全日參與研討會

3/28 全日參與研討會、晚上飛機返回台灣

Towards Improving QoS-Guided Scheduling in Grids

Ching-Hsien Hsu[1], Justin Zhan[2], Wai-Chi Fang[3] and Jianhua Ma[4]

[1]*Department of Computer Science and Information Engineering, Chung Hua University, Taiwan*
chh@chu.edu.tw

[2]*Heinz School, Carnegie Mellon University, USA*
justinzh@andrew.cmu.edu

[3]*Department of Electronics Engineering, National Chiao Tung University, Taiwan*
wfang@mail.nctu.edu.tw

[4]*Digital Media Department, Hosei University, Japan*
jianhua@hosei.ac.jp

Abstract

With the emergence of grid technologies, the problem of scheduling tasks in heterogeneous systems has been arousing attention. In this paper, we present two optimization schemes, Makespan Optimization Rescheduling (MOR) and Resource Optimization Rescheduling (ROR), which are based on the QoS Min-Min scheduling technique, for reducing the makespan of a schedule and the need of total resource amount. The main idea of the proposed techniques is to reduce overall execution time without increasing resource need; or reduce resource need without increasing overall execution time. To evaluate the effectiveness of the proposed techniques, we have implemented both techniques along with the QoS Min-Min scheduling algorithm. The experimental results show that the MOR and ROR optimization schemes provide noticeable improvements.

1. Introduction

With the emergence of IT technologies, the need of computing and storage are rapidly increased. To invest more and more equipments is not an economic method for an organization to satisfy the even growing computational and storage need. As a result, grid has become a widely accepted paradigm for high performance computing.

To realize the concept virtual organization, in [13], the grid is also defined as "A type of parallel and distributed system that enables the sharing, selection, and aggregation of geographically distributed autonomous and heterogeneous resources dynamically at runtime depending on their availability, capability, performance, cost, and users' quality-of-service requirements". As the grid system aims to satisfy users' requirements with limit resources, scheduling grid resources plays an important factor to improve the overall performance of a grid.

In general, grid scheduling can be classified in two categories: the performance guided schedulers and the economy guided schedulers [16]. Objective of the performance guided scheduling is to minimize turnaround time (or makespan) of grid applications. On the other hand, in economy guided scheduling, to minimize the cost of resource is the main objective. However, both of the scheduling problems are NP-complete, which has also instigated many heuristic solutions [1, 6, 10, 14] to resolve. As mentioned in [23], a complete grid scheduling framework comprises application model, resource model, performance model, and scheduling policy. The scheduling policy can further decomposed into three phases, the resource discovery and selection phase, the job scheduling phase and the job monitoring and migration phase, where the second phase is the focus of this study.

Although many research works have been devoted in scheduling grid applications on heterogeneous system, to deal with QOS scheduling in grid is quite complicated due to more constrain factors in job scheduling, such as the need of large storage, big size memory, specific I/O devices or real-time services, requested by the tasks to be completed. In this paper, we present two QoS based rescheduling schemes aim to improve the makespan of scheduling batch jobs in grid. In addition, based on the QoS guided scheduling scheme, the proposed rescheduling technique can also reduce the amount of resource need without increasing the makespan of grid jobs. The main contribution of this work are twofold, one can shorten the turnaround time of grid applications without increasing the need of grid resources; the other one can minimize the need of grid resources without increasing the turnaround time of grid applications, compared with the traditional QoS guided scheduling method. To evaluate the performance of the proposed techniques, we have implemented our rescheduling approaches along with the QoS Min-Min scheduling algorithm [9] and the non-QoS based Min-Min scheduling algorithm. The experimental results show that the proposed techniques are effective in heterogeneous systems under different circumstances. The improvement is also significant in economic grid model [3].

The rest of this paper is organized as follows. Section 2 briefly describes related research in grid computing and job scheduling. Section 3 clarifies our research model by illustrating the traditional Min-min model and the QoS guided Min-min model. In Section 4, two optimization schemes for reducing the total execution time of an application and reducing resource need are presented, where two rescheduling approaches are illustrated in detail. We conduct performance evaluation and discuss experiment results in Section 5. Finally, concluding remarks and future work are given in Section 6.

2. Related Work

Grid scheduling can be classified into traditional grid scheduling and QoS guided scheduling or economic based grid scheduling. The former emphasizes the performance of systems of applications, such as system throughput, jobs' completion time or response time. Swany *et al*. provides an approach to improving throughput for grid applications with network logistics by building a tree of "best" paths through the graph and has running time of O(*N*log*N*) for implementations that keep the edges sorted [15]. Such approach is referred as the Minimax Path (MMP) and employs a greedy, tree-building algorithm that produces optimal results [20]. Besides data-parallel applications requiring high performance in grid systems, there is a Dynamic Service Architecture (DSA) based on static compositions and optimizations, but also allows for high performance and flexibility, by use of a lookahead scheduling mechanism [4]. To minimizing the processing time of extensive processing loads originating from various sources, the approaches divisible load model [5] and single level tree network with two root processors with divisible load are proposed [12]. In addition to the job matching algorithm, the resource selection algorithm is at the core of the job scheduling decision module and must have the ability to integrate multi-site computation power. The CGRS algorithm based on the distributed computing grid model and the grid scheduling model integrates a new density-based internet clustering algorithm into the decoupled scheduling approach of the GrADS and decreases its time complexity [24]. The scheduling of parallel jobs in a heterogeneous multi-site environment, where each site has a homogeneous cluster of processors, but processors at different sites has different speeds, is presented in [18]. Scheduling strategy is not only in batch but also can be in real-time. The SAREG approach paves the way to the design of security-aware real-time scheduling algorithms for Grid computing environments [21].

For QoS guided grid scheduling, apparently, applications in grids need various resources to run its completion. In [17], an architecture named public computing utility (PCU) is proposed uses virtual machine (VMs) to implement "time-sharing" over the resources and augments finite number of private resources to public resources to obtain higher level of quality of services. However, the QoS demands maybe include various packet-type and class in executing job. As a result, a scheduling algorithm that can support multiple QoS classes is needed. Based on this demand, a multi-QoS scheduling algorithm is proposed to improve the scheduling fairness and users' demand [11]. He *et al.* [7] also presented a hybrid approach for scheduling moldable jobs with QoS demands. In [9], a novel framework for policy based scheduling in resource allocation of grid computing is also presented. The scheduling strategy can control the request assignment to grid resources by adjusting usage accounts or request priorities. Resource management is achieved by assigning usage quotas to intended users. The scheduling method also supports reservation based grid resource allocation and quality of service feature. Sometimes the scheduler is not only to match the job to which resource, but also needs to find the optimized transfer path based on the cost in network. In [19], a distributed QoS network scheduler (DQNS) is presented to adapt to the ever-changing network conditions and aims to serve the path requests based on a cost function.

3. Research Architecture

Our research model considers the static scheduling of batch jobs in grids. As this work is an extension and optimization of the QoS guided scheduling that is based on Min-Min scheduling algorithm [9], we briefly describe the Min-Min scheduling model and the QoS guided Min-Min algorithm. To simplify the presentation, we first clarify the following terminologies and assumptions.

- *QoS Machine* (M_Q) – machines can provide special services.
- *QoS Task* (T_Q) – tasks can be run completion only on QoS machine.
- *Normal Machine* (M_N) – machines can only run normal tasks.
- *Normal Task* (T_N) – tasks can be run completion on both QoS machine and normal machine.
- A chunk of tasks will be scheduled to run completion based on all available machines in a batch system.
- A task will be executed from the beginning to completion without interrupt.
- The completion time of task t_i to be executed on machine m_j is defined as

$$CT_{ij} = dt_{ij} + et_{ij} \quad (1)$$

Where et_{ij} denotes the estimated execution time of task t_i executed on machine m_j; dt_{ij} is the delay time of task t_i on machine m_j.

The Min-Min algorithm is shown in Figure 1.

```
Algorithm_Min-Min()
{
  while there are jobs to schedule
     for all job i to schedule
        for all machine j
           Compute CT_{i,j} = CT(job i, machine j)
        end for
        Compute minimum CT_{i,j}
     end for
     Select best metric match m
     Compute minimum CT_{m,n}
     Schedule job m on machine n
  end while
} End_of_ Min-Min
```

Figure 1. The Min-Min Algorithm

Analysis: If there are *m* jobs to be scheduled in *n* machines, the time complexity of Min-Min algorithm is $O(m^2n)$. The Min-Min algorithm does not take into account the QoS issue in the scheduling. In some situation, it is possible that normal tasks occupied machine that has special services (referred as QoS machine). This may increase the delay of QoS tasks or result idle of normal machines.

The QoS guided scheduling is proposed to resolve the above defect in the Min-Min algorithm. In QoS guided model, the scheduling is divided into two classes, the QoS class and the non-QoS class. In each class, the Min-Min algorithm is employed. As the QoS tasks have higher priority than normal tasks in QoS guided scheduling, the QoS tasks are prior to be allocated on QoS machines. The normal tasks are then scheduled to all machines in Min-Min manner. Figure 2 outlines the method of QoS guided scheduling model with the Min-Min scheme.

Analysis: If there are *m* jobs to be scheduled in *n* machines, the time complexity of QoS guided scheduling algorithm is $O(m^2n)$.

Figure 3 shows an example demonstrating the Min-Min and QoS Min-Min scheduling schemes. The asterisk * means that tasks/machines with QoS demand/ability, and the X means that QoS tasks couldn't be executed on that machine. Obviously, the QoS guided scheduling algorithm gets the better performance than the Min-Min algorithm in term of makespan. Nevertheless, the QoS guided model is not optimal in both makespan and resource cost. We will describe the rescheduling optimization in next section.

```
Algorithm_QOS-Min-Min()
{
  for all tasks ti in meta-task Mv (in an arbitrary order)
      for all hosts mj (in a fixed arbitrary order)
          CTij = etij + dtj
      end for
  end for
  do until all tasks with QoS request in Mv are mapped
      for each task with high QoS in Mv,
          find a host in the QoS qualified host set that obtains
          the earliest completion time
      end for
      find task tk with the minimum earliest completion time
      assign task tk to host ml that gives the earliest completion
      time
      delete task tk from Mv
      update dtl
      update CTil for all i
  end do
  do until all tasks with non-QoS request in Mv are mapped
      for each task in Mv
          find the earliest completion time and the
          corresponding host
      end for
      find the task tk with the minimum earliest completion time
      assign task tk to host ml that gives the earliest completion
      time
      delete task tk from Mv
      update dtl
      update CTil for all i
  end do
} End_of_ QOS-Min-Min
```

Figure 2. The QoS Guided Algorithm

4. Rescheduling Optimization

Grid scheduling works as the mapping of individual tasks to computer resources, with respecting service level agreements (SLAs) [2]. In order to achieve the optimized performance, how to mapping heterogeneous tasks to the best fit resource is an important factor. The Min-Min algorithm and the QoS guided method aims at scheduling jobs to achieve better makespan. However, there are still having rooms to make improvements. In this section, we present two optimization schemes based on the QoS guided Min-Min approach.

	*M1	M2	M3
T1	7	4	7
T2	3	3	5
T3	9	5	7
*T4	5	X	X
T5	9	8	6
*T6	5	X	X

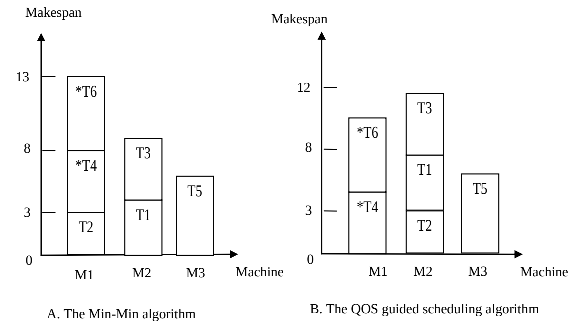
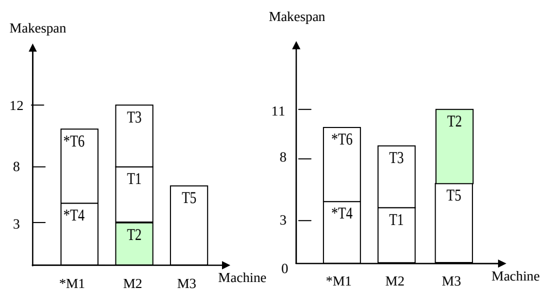


Figure 3. Min-Min and QoS Guided Min-Min

4.1 Makespan Optimization Rescheduling (*MOR*)

The first one is *Makespan Optimization Rescheduling* (*MOR*), which focuses on improving the makespan to achieve better performance than the QoS guided scheduling algorithm. Assume the makespan achieved by the QoS guided approach in different machines are CT_1, CT_2, …, CT_m, with CT_k = max { CT_1, CT_2, …, CT_m }, where m is the number of machines and $1 \le k \le m$. By subtracting $CT_k - CT_i$, where $1 \le i \le m$ and $i \neq k$, we can have m-1 available time fragments. According to the size of these available time fragments and the size of tasks in machine M_k, the *MOR* dispatches suitable tasks from machine M_k to any other machine that has available and large enough time fragments. Such optimization is repeated until there is no task can be moved.

	*M1	M2	M3
T1	7	4	7
T2	3	3	5
T3	9	5	7
*T4	5	X	X
T5	9	8	6
*T6	5	X	X



A. The QOS guided scheduling algorithm

B. The Makespan Optimization Rescheduling (MOR) algorithm

Figure 4. Example of *MOR*

Recall the example given in Figure 3, Figure 4 shows the optimization of the *MOR* approach. The left side of Figure 4 demonstrates that the QoS guided scheme gives a schedule with makespan = 12, wheremachine M2 presents maximum *CT* (completion time), which is assembled by tasks T2, T1 and T3. Since the *CT* of machine ‘M3’ is 6, so ‘M3’ has an available time fragment (6). Checking all tasks in machine M2, only T2 is small enough to be allocated in the available time fragment in M3. Therefore, task M2 is moved to M3, resulting machine ‘M3’ has completion time *CT*=11, which is better than the QoS guided scheme.

As mentioned above, the *MOR* is based on the QoS guided scheduling algorithm. If there are *m* tasks to be scheduled in *n* machines, the time complexity of *MOR* is $O(m^2n)$. Figure 5 outlines a pseudo of the *MOR* scheme.

```
Algorithm_MOR()
{
    for CT_j in all machines
        find out the machine with maximum makespan CT_max and
        set it to be the standard
    end for
    do until no job can be rescheduled
        for job i in the found machine with CT_max
            for all machine j
                according to the job's QOS demand, find the
                adaptive machine j
                if (the execute time of job i in machine j + the
                CT_j < makespan)
                    rescheduling the job i to machine j
                    update the CT_j and CT_max
                    exit for
                end if
            next for
            if the job i can be reschedule
                find out the new machine with maximum CT_max
                exit for
            end if
        next for
    end do
} End_of_ MOR
```

Figure 5. The *MOR* Algorithm

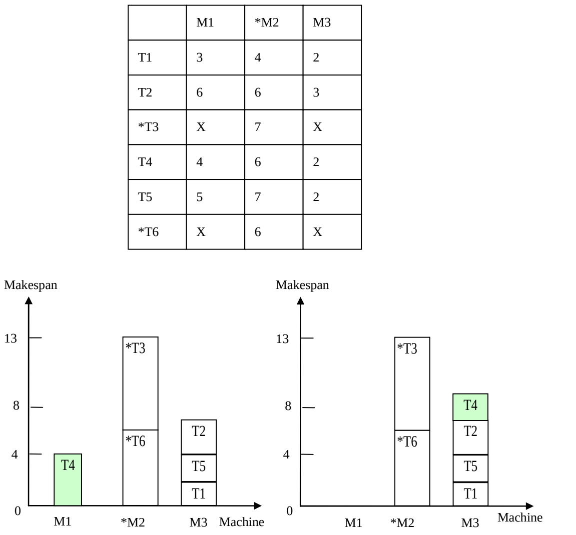
4.2 Resource Optimization Rescheduling (*ROR*)

Following the assumptions described in *MOR*, the main idea of the *ROR* scheme is to re-dispatch tasks from the machine with minimum number of tasks to other machines, expecting a decrease of resource need. Consequently, if we can dispatch all tasks from machine M_x to other machines, the total amount of resource need will be decreased.

Figure 6 gives another example of QoS scheduling, where the QoS guided scheduling presents makespan = 13. According to the clarification of *ROR*, machine ‘M1’ has the fewest amount of tasks. We can dispatch the task ‘T4’ to machine ‘M3’ with the following constraint

$$CT_{ij} + CT_j <= CT_{max} \qquad (2)$$

The above constraint means that the rescheduling can be performed only if the movement of tasks does not increase the overall makespan. In this example, $CT_{43} = 2$, CT_3=7 and $CT_{\max}$=CT_2=13. Because the makespan of M3 (CT_3) will be increased from 7 to 9, which is smaller than the $CT_{\max}$, therefore, the task migration can be performed. As the only task in M1 is moved to M3, the amount of resource need is also decreased comparing with the QoS guided scheduling.



A. The QOS guided scheduling

B. The Resource Optimization Rescheduling (ROR) Algorithm

Figure 6. Example of *ROR*

The *ROR* is an optimization scheme which aims to minimize resource cost. If there are *m* tasks to be scheduled in *n* machines, the time complexity of *ROR* is also $O(m^2n)$. Figure 7 depicts a high level description of the *ROR* optimization scheme.

```
Algorithm_MOR()
{
   for m in all machines
         find out the machine m with minimum count of jobs
   end for
   do until no job can be rescheduled
         for job i in the found machine with minimum count of jobs
               for all machine j
                  according to the job's QOS demand, find the
                  adaptive machine j
                  if (the execute time of job i in machine j + the
                   CTj <= makespan CTmax)
                       rescheduling the job i to machine j
                       update the CTj
                       update the count of jobs in machine m and
                       machine j
                       exit for
                   end if
               next for
         next for
   end do
 } End_of_ MOR
```

Figure 7. The *ROR* Algorithm

5. Performance Evaluation

5.1 Parameters and Metrics

To evaluate the performance of the proposed techniques, we have implemented the Min-Min scheduling algorithm and the QoS guided Min-Min scheme. The experiment model consists of heterogeneous machines and tasks. Both of the Machines and tasks are classified into QoS type and non-QoS type. Table 1 summarizes six parameters and two comparison metrics used in the experiments. The number of tasks is ranged from 200 to 600. The number of machines is ranged from 50 to 130. The percentage of QoS machines and tasks are set between 15% and 75%. Heterogeneity of tasks are defined as H_t (for non-QoS task) and H_Q (for QoS task), which is used in generating random tasks. For example, the execution time of a non-QoS task is randomly generated from the interval [10, $H_t \times 10^2$] and execution time of a QoS task is randomly generated from the interval [10^2, $H_Q \times 10^3$] to reflect the real application world. All of the parameters used in the experiments are generated randomly with a uniform distribution. The results demonstrated in this section are the average values of running 100 random test samples.

Table 1: Parameters and Comparison Metrics

Task number (N_T)	{200, 300, 400, 500, 600}
Resource number (N_R)	{50, 70, 90, 110, 130}
Percentage of QOS resources (Q_R%)	{15%, 30%, 45%, 60%, 75%}
Percentage of QOS tasks (Q_T%)	{15%, 30%, 45%, 60%, 75%}
Heterogeneity of non-QOS tasks (H_T)	{1, 3, 5, 7, 9}
Heterogeneity of QOS tasks (H_Q)	{3, 5, 7, 9, 11}
Makespan	The completion time of a set of tasks
Resource Used (R_U)	Number of machines used for executing a set of tasks

5.2 Experimental Results of *MOR*

Table 2 compares the performance of the *MOR*, Min-Min algorithm and the QoS guided Min-Min scheme in term of makespan. There are six tests that are conducted with different parameters. In each test, the configurations are outlined beside the table caption from (a) to (f). Table (a) changes the number of tasks to analyze the performance results. Increasing the number of tasks, improvement of *MOR* is limited. An average improvement ratio is from 6% to 14%. Table (b) changes the number of machines. It is obvious that the *MOR* has significant improvement in larger grid systems, i.e., large amount of machines. The average improvement rate is 7% to 15%. Table (c) discusses the influence of changing percentages of QoS machines. Intuitionally, the *MOR* performs best with 45% QoS machines. However, this observation is not always true. By analyzing the four best ones in (a) to (d), we observe that the four tests (a) N_T=200 (N_R=50, Q_R=30%, Q_T=20%) (b) N_R=130 (N_T=500, Q_R=30%, Q_T=20%) (c)

Q_R=45% (N_T=300, N_R=50, Q_T=20%) and (d) Q_T=15% (N_T=300, N_R=50, Q_R=40%) have best improvements. All of the four configurations conform to the following relation,

$$0.4 \times (N_T \times Q_T) = N_R \times Q_R \quad (3)$$

This observation indicates that the improvement of *MOR* is significant when the number of QoS tasks is 2.5 times to the number of QoS machines. Tables (e) and (f) change heterogeneity of tasks. We observed that heterogeneity of tasks is not critical to the improvement rate of the *MOR* technique, which achieves 7% improvements under different heterogeneity of tasks.

Table 2: Comparison of Makespan

(a) (N_R=50, Q_R=30%, Q_T=20%, H_T=1, H_Q=1)

Task Number (N_T)	200	300	400	500	600
Min-Min	978.2	1299.7	1631.8	1954.6	2287.8
QOS Guided Min-Min	694.6	917.8	1119.4	1359.9	1560.1
MOR	597.3	815.5	1017.7	1254.8	1458.3
Improved Ratio	14.01%	11.15%	9.08%	7.73%	6.53%

(b) (N_T=500, Q_R=30%, Q_T=20%, H_T=1, H_Q=1)

Resource Number (N_R)	50	70	90	110	130
Min-Min	1931.5	1432.2	1102.1	985.3	874.2
QOS Guided Min-Min	1355.7	938.6	724.4	590.6	508.7
MOR	1252.6	840.8	633.7	506.2	429.4
Improved Ratio	7.60%	10.42%	12.52%	14.30%	15.58%

(c) (N_T=300, N_R=50, Q_T=20%, H_T=1, H_Q=1)

Q_R%	15%	30%	45%	60%	75%
Min-Min	2470.8	1319.4	888.2	777.6	650.1
QOS Guided Min-Min	1875.9	913.6	596.1	463.8	376.4
MOR	1767.3	810.4	503.5	394.3	339.0
Improved Ratio	5.79%	11.30%	15.54%	14.99%	9.94%

(d) (N_T=300, N_R=50, Q_R=40%, H_T=1, H_Q=1)

Q_T%	15%	30%	45%	60%	75%
Min-Min	879.9	1380.2	1801.8	2217.0	2610.1
QOS Guided Min-Min	558.4	915.9	1245.2	1580.3	1900.6
MOR	474.2	817.1	1145.1	1478.5	1800.1
Improved Ratio	15.07%	10.79%	8.04%	6.44%	5.29%

(e) (N_T=500, N_R=50, Q_R=30%, Q_T=20%, H_Q=1)

H_T	1	3	5	7	9
Min-Min	1891.9	1945.1	1944.6	1926.1	1940.1
QOS Guided Min-Min	1356.0	1346.4	1346.4	1354.9	1357.3
MOR	1251.7	1241.4	1244.3	1252.0	1254.2
Improved Ratio	7.69%	7.80%	7.58%	7.59%	7.59%

(f) (N_T=500, N_R=50, Q_R=30%, Q_T=20%, H_T=1)

H_Q	3	5	7	9	11
Min-Min	1392.4	1553.9	1724.9	1871.7	2037.8
QOS Guided Min-Min	867.5	1007.8	1148.2	1273.2	1423.1
MOR	822.4	936.2	1056.7	1174.3	1316.7
Improved Ratio	5.20%	7.11%	7.97%	7.77%	7.48%

5.3 Experimental Results of *ROR*

Table 3 analyzes the effectiveness of the *ROR* technique under different circumstances.

Table 3: Comparison of Resource Used

(a) (N_R=100, Q_R=30%, Q_T=20%, H_T=1, H_Q=1)

Task Number (N_T)	200	300	400	500	600
QOS Guided Min-Min	100	100	100	100	100
ROR	39.81	44.18	46.97	49.59	51.17
Improved Ratio	60.19%	55.82%	53.03%	50.41%	48.83%

(b) (N_T=500, Q_R=30%, Q_T=20%, H_T=1, H_Q=1)

Resource Number (N_R)	50	70	90	110	130
QOS Guided Min-Min	50	70	90	110	130
ROR	26.04	35.21	43.65	50.79	58.15
Improved Ratio	47.92%	49.70%	51.50%	53.83%	55.27%

(c) (N_T=500, N_R=50, Q_T=20%, H_T=1, H_Q=1)

Q_R%	15%	30%	45%	60%	75%
QOS Guided Min-Min	50	50	50	50	50
ROR	14.61	25.94	35.12	40.18	46.5
Improved Ratio	70.78%	48.12%	29.76%	19.64%	7.00%

(d) (N_T=500, N_R=100, Q_R=40%, H_T=1, H_Q=1)

Q_T%	15%	30%	45%	60%	75%
QOS Guided Min-Min	100	100	100	100	100
ROR	57.74	52.9	48.54	44.71	41.49
Improved Ratio	42.26%	47.10%	51.46%	55.29%	58.51%

(e) (N_T=500, N_R=100, Q_R=30%, Q_T=20%, H_Q=1)

H_T	1	3	5	7	9
QOS Guided Min-Min	100	100	100	100	100
ROR	47.86	47.51	47.62	47.61	47.28
Improved Ratio	52.14%	52.49%	52.38%	52.39%	52.72%

(f) (N_T=500, N_R=100, Q_R=30%, Q_T=20%, H_T=1)

H_Q	3	5	7	9	11
QOS Guided Min-Min	100	100	100	100	100
ROR	54.61	52.01	50.64	48.18	46.53
Improved Ratio	45.39%	47.99%	49.36%	51.82%	53.47%

Similar to those of Table 2, Table (a) changes the number of tasks to verify the reduction of resource that needs to be achieved by the *ROR* technique. We noticed that the *ROR* has significant improvement in minimizing grid resources. Comparing with the QoS guided Min-Min scheduling algorithm, the *ROR* achieves 50% ~ 60% improvements without increasing overall makespan of a chunk of grid tasks. Table (b) changes the number of machines. The *ROR* retains 50% improvement ratio. Table (c) adjusts percentages of QoS machine. Because this test has 20% QoS tasks, the *ROR* performs best at 15% QoS machines. This observation implies that the *ROR* has significant improvement when QoS tasks and QoS machines are with the same percentage. Table (d) sets 40% QoS machine and changes the percentages of QoS tasks. Following the above analysis, the *ROR* technique achieves more than 50% improvements when QoS tasks are with 45%, 60% and 75%. Tables (e) and (f) change the heterogeneity of tasks. Similar to the results of section 5.2, the heterogeneity of tasks is not critical to the improvement rate of the *ROR* technique. Overall speaking, the *ROR* technique presents 50% improvements in minimizing total resource need compare with the QoS guided Min-Min scheduling algorithm.

6. Conclusions

In this paper we have presented two optimization schemes aiming to reduce the overall completion time (makespan) of a chunk of grid tasks and minimize the total resource cost. The proposed techniques are based on the QoS guided Min-Min scheduling algorithm. The optimization achieved by this work is twofold; firstly, without increasing resource costs, the overall task execution time could be reduced by the *MOR* scheme with 7%~15% improvements. Second, without increasing task completion time, the overall resource cost could be reduced by the *ROR* scheme with 50% reduction on average, which is a significant improvement to the state of the art scheduling technique. The proposed *MOR* and *ROR* techniques have characteristics of low complexity, high effectiveness in large-scale grid systems with QoS services.

References

[1] A. Abraham, R. Buyya, and B. Nath, "Nature's Heuristics for Scheduling Jobs on Computational Grids", Proc. 8th IEEE International Conference on Advanced Computing and Communications (ADCOM-2000), pp.45-52, 2000.

[2] A. Andrieux, D. Berry, J. Garibaldi, S. Jarvis, J. MacLaren, D. Ouelhadj, D. Snelling, "Open Issues in Grid Scheduling", National e-Science Centre and the Inter-disciplinary Scheduling Network Technical Paper, UKeS-2004-03.

[3] R. Buyya, D. Abramson, Jonathan Giddy, Heinz Stockinger, "Economic Models for Resource Management and Scheduling in Grid Computing", Journal of Concurrency: Practice and Experience, vol. 14, pp. 13-15, 2002.

[4] Jesper Andersson, Morgan Ericsson, Welf Löwe, and Wolf Zimmermann, "Lookahead Scheduling for Reconfigurable GRID Systems", 10th International Europar'04: Parallel Processing, vol. 3149, pp. 263-270, 2004.

[5] D Yu, Th G Robertazzi, "Divisible Load Scheduling for Grid Computing", 15th IASTED Int'l. Conference on Parallel and Distributed Computing and Systems, Vol. 1, pp. 1-6, 2003

[6] Fangpeng Dong and Selim G. Akl, "Scheduling Algorithms for Grid Computing: State of the Art and Open Problems", Technical Report No. 2006-504, 2006.

[7] Ligang He, Stephen A. Jarvis, Daniel P. Spooner, Xinuo Chen, Graham R. Nudd, "Hybrid Performance-oriented Scheduling of Moldable Jobs with QoS Demands in Multiclusters and Grids", Grid and Cooperative Computing (GCC 2004), vol. 3251, pp. 217–224, 2004.

[8] Xiaoshan He, Xian-He Sun, Gregor Von Laszewski, "A QoS Guided Scheduling Algorithm for Grid Computing", Journal of Computer Science and Technology, vol.18, pp.442-451, 2003.

[9] Jang-uk In, Paul Avery, Richard Cavanaugh, Sanjay Ranka, "Policy Based Scheduling for Simple Quality of Service in Grid Computing", IPDPS 2004, pp. 23, 2004.

[10] J. Schopf. "Ten Actions when Superscheduling: A Grid Scheduling Architecture", Scheduling Architecture Workshop, 7th Global Grid Forum, 2003.

[11] Junsu Kim, Sung Ho Moon, and Dan Keun Sung, "Multi-QoS Scheduling Algorithm for Class Fairness in High Speed Downlink Packet Access", Proceedings of IEEE Personal, Indoor and Mobile Radio Communications Conference (PIMRC 2005), vol. 3, pp. 1813-1817, 2005

[12] M.A. Moges and T.G. Robertazzi, "Grid Scheduling Divisible Loads from Multiple Sources via Linear Programming", 16th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS), pp. 423-428, 2004.

[13] M. Baker, R. Buyya and D. Laforenza, "Grids and Grid Technologies for Wide-area Distributed Computing", in Journal of Software-Practice & Experience, Vol. 32, No.15, pp. 1437-1466, 2002.

[14] Jennifer M. Schopf, "A General Architecture for Scheduling on the Grid", Technical Report ANL/MCS, pp. 1000-1002, 2002.

[15] M. Swany, "Improving Throughput for Grid Applications with Network Logistics", *Proc.* IEEE/ACM Conference on High Performance Computing and Networking, 2004.

[16] R. Moreno and A.B. Alonso, "Job Scheduling and Resource Management Techniques in Economic Grid Environments", LNCS 2970, pp. 25-32, 2004.

[17] Shah Asaduzzaman and Muthucumaru Maheswaran, "Heuristics for Scheduling Virtual Machines for Improving QoS in Public Computing Utilities", *Proc.* 9th International Conference on Computer and Information Technology (ICCIT'06), 2006.

[18] Gerald Sabin, Rajkumar Kettimuthu, Arun Rajan and P Sadayappan, "Scheduling of Parallel Jobs in a Heterogeneous Multi-Site Environment", in the Proc. of the 9th International Workshop on Job Scheduling Strategies for Parallel Processing, LNCS 2862, pp. 87-104 , June 2003.

[19] Sriram Ramanujam, Mitchell D. Theys, "Adaptive Scheduling based on Quality of Service in Distributed Environments", *PDPTA'05*, pp. 671-677, 2005.

[20] T. H. Cormen, C. L. Leiserson, and R. L. Rivest, "Introduction to Algorithms", First edition, MIT Press and McGraw-Hill, ISBN 0-262-03141-8, 1990.

[21] Tao Xie and Xiao Qin, "Enhancing Security of Real-Time Applications on Grids through Dynamic Scheduling", *Proc.* the 11th Workshop on Job Scheduling Strategies for Parallel

Processing (JSSPP'05), pp. 146-158, 2005.
[22] Haobo Yu, Andreas Gerstlauer, Daniel Gajski, "RTOS Scheduling in Transaction Level Models", in Proc. of the 1st IEEE/ACM/IFIP international conference on Hardware/software Codesign & System Synpaper, pp. 31-36, 2003.
[23] Y. Zhu, "A Survey on Grid Scheduling Systems", LNCS 4505, pp. 419-427, 2007.
[24] Weizhe Zhang, Hongli Zhang, Hui He, Mingzeng Hu, "Multisite Task Scheduling on Distributed Computing Grid", LNCS 3033, pp. 57–64, 2004.