# 行政院國家科學委員會專題研究計畫 成果報告

## 適應於異質性網格計算環境上複合式資源排程技術之設計與研發(第 2 年)
## 研究成果報告(完整版)

計 畫 類 別 ：個別型
計 畫 編 號 ：NSC 95-2221-E-216-011-MY2
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：李開文
碩士班研究生-兼任助理人員：郁家豪
博士班研究生-兼任助理人員：陳世璋
博士班研究生-兼任助理人員：陳泰龍

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

# 適應於異質性網格計算環境上複合式資源排程技術之設計與研發

計畫類別：☑ 個別型計畫　　□ 整合型計畫
計畫編號：NSC95-2221-E-216-011-MY2
執行期間：95 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：　陳世璋、陳泰龍（中華大學工程科學研究所博士生）
李開文、郁家豪（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

## 適應於異質性網格計算環境上複合式資源排程技術之設計與研發

## Design, Analysis and Implementation of Composite Multiple Resource Scheduler for Heterogeneous Grid Computing

計畫編號：NSC95-2221-E-216-011-MY2
執行期限：95 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
陳世璋(博三)、李開文(研二)、陳泰龍(博二)、郁家豪(研二)

### 一、中文摘要

本報告是有關於在異質性網格環境上開發複合式資源排程技術之設計，並且發展具有平台透通性的分析工具。本計畫有三個主要的研究課題：一、發展適應於叢集網格環境之主從式工作排程技術。此項成果可以直接移植到叢集網格的工作排程系統。二、發展複合式資源排程的核心技術。針對異質的計算網格系統與網格拓僕，開發最佳化的評估模組；並以實際的 work load trace tape，分析系統的效能。三、發展具有平台透通性的系統資源排程、調整與學習工具。鑒於網格平台在大量計算與高性能科學應用上漸漸普及，本計畫之研究成果可以直接應用在發展高效能叢集式與網格計算之實驗環境。

**關鍵詞**：異質計算、網格計算、複合式排程、資源排程、工作排程、主從式架構。

### Abstract

This report presents the project to design analysis and implement a composite resource scheduler and a platform mutual analysis tool on heterogeneous grids. There are three major subjects in this research: first, we will develop master-slave task scheduling technologies which can be directly incorporated on cluster grid; second, we will develop the main technique of composite resource scheduler. For heterogeneous grid and its topology, we will devise optimized performance analysis model and analyze system efficiency according to a set of real work load trace tape from SDSC; third, we will develop platform mutual resource scheduling and learning tool. Whereas the grid computing becomes widespread for massive computing and high performance scientific applications, the achievements of this research will facilitate constructing high performance cluster and grid systems.

**Keywords:** Heterogeneous Computing, Grid Computing, Composite Scheduling, Resource Scheduling, Task Scheduling, Master Slave.

### 二、緣由與目的

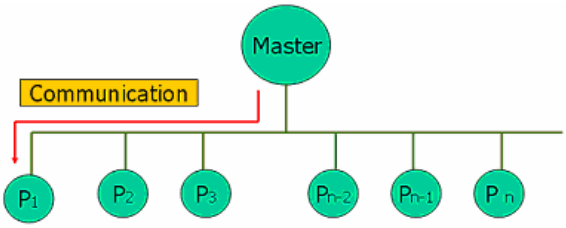
格網計算的技術在近幾年被運用在整合各種類型網路環境下的各種資源，其目標在於讓使用者將來處理大量資料和龐大的計算時能在最短的時間內獲得最有效率的執行成果，所以格網運算技術簡單的說就是利用大規模整合的電腦系統，搭配有效率的網路傳輸，可依照使用者的需求，提供大量的資料處理功能。在異質網路架構下，資源排程的技術是很

重要的，排程的主要目的即是讓參與執行工作的眾多處理器發揮最大的效能，配合網路頻寬做最佳化的傳輸，並且必需讓各處理器閒置時間降到最低，以及降低處理器的閒置時間。這些研究的方向使得格網的高效能運算技術有更多的發展空間。為了結合格網環境與最佳化的資源排程技術，在主從式架構下的網路計算環境所從事的大量工作排程運算技術所延伸的相關問題就是相當值得研究的課題。在格網與主從式架構下，工作排程的良好與否直接影響了程式的完成時間與是否有妥善的運用系統資源。主從式架構下的資源排程可以避免單一個處理器負擔太重，而增加整個工作的結束時間，以達到高效能的計算原則。另一方面，在主從式架構與格網計算的環境下，要有效率的執行主機交付的工作，適當的資料傳輸排程配合執行排班也是很重要的。在異質性的網路計算環境上的研究資源的配置，動態的新增閒置的處理器，以及工作環境的管理，甚至資料安全性的問題都是及待解決與改良的問題。另一方面、過去許多研究根據處理器的異質性來安排工作排程，或根據網路頻寬來執行工作排程。這些研究通常就單一系統因素來考量工作排程，然而，在某些情況下可能無法達到系統的公平性(fairness)與最佳效率與產能(throughput)。在這一個研究計畫中，我們將探討複合式的工作排程技術(composite resource scheduler)，將異質性的 CPU 運算、以及異質性網路頻寬，同時納入考量，作為執行工作分派的主要核心技術。複合式工作排程技術的主要優點可以提升系統整體的公平性，降低反應與延遲時間(delay guarantee)以及提升系統的整體產能。

## 三、研究方法與成果

由於處理器需要收到連續工作資料(Task)，如果處理器數量增加，則某些演算法不容易有效率地執行資料傳輸和執行。為了降低處理器所浪費的閒置時間，發展有效率的工作排程演算法是必需的。主要研究工作包含處理器運算能力配置對整體程式執行的必要性及其在效能上的影響，以及現有資源排程的技術移植與測試。針對 CPU、頻寬的異質性開發複合式資源排程技術。研究的重點有系統產能的提升，資源使用率的提升，系統閒滯、反應、與延遲時間的縮短，此一部分也包含了資料傳輸最佳化與工作排程演算法的最佳化；此外，我們也將探討如何將前述發展的技術應用在異質性網格拓僕架構，其中除了異質性工作分割的探討之外，還包含工作重新分派排程的技術。

執行工作排程時，需考量處理器與異質性網路頻寬以及資料量大小，圖一顯示，叢集式網格系統上工作分派示意圖，不同的網格計算節點的運算能力是不同的。在一個網格系統中，Master 表示資源分派節點，亦可視為 Resource Scheduler。異質的工作佇列將由 Master 分派至各個運算節點。在這一個部分的系統架構之下，我們假設工作的執行是不可以插隊的，也就是說系統資源一旦分配給某個工作，其他的工作不可以同時使用。在通訊的部分，網格系統上 Resource Scheduler 與 Peer Node 之間的 communication 亦視為 Heterogeneous。而其之間的 communication 則假設需要彼此互斥(Exclusive)。

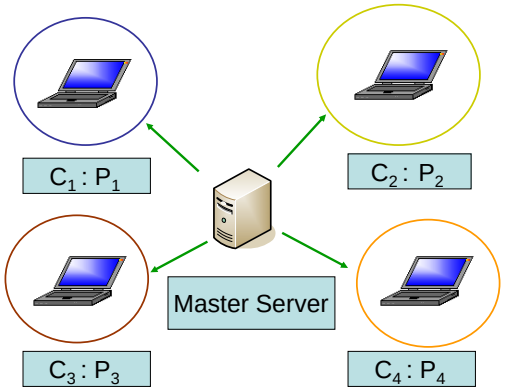


圖一、叢集式網格系統工作分派示意圖

處理器與工作排程的關係主要是由兩個變數 $T_{i\_comm}$ (單元工作傳輸時間)與 $T_i$ (單元工作執行時間) 所組成。如圖二所示，$C_1$ 到 $C_4$ 為四個不同區域中的子處理器連結至 Master-Server，而 $P_1$ 到 $P_4$ 表示子處理器皆有

不同的運算能力，所需的傳送時間為 $T_1$ 到 $T_2$，$T_{1\_comm}$ 到 $T_{4\_comm}$ 即為 Master-Server 傳送一個工作到各區域之處理器所需之單位時間。

本計畫第一年所提出的資源排程演算法為 Shortest Communication Ratio (*SCR*)，主要分為三個部份。第一個部分先對需要參與計算的節點處理器排程，如圖一中的 $P_1$ 到 $P_4$，排序依序為處理器中擁有最快執行效能 $P_1$ 到最慢 $P_4$ 處理器執行效能。




圖二、異質性處理器與異質性網路頻寬示意圖

第二部份使用參與運算的處理器之 ($T_{i\_comm}$ + $T_i$) 計算其最小公倍數的概念將計算出每一個處理器在每一個基本排程週期 (Basic Scheduling Cycle)將接收多少數量的工作以利於系統的排程。

第三部份依照所計算出的基本排程週期內每個處理器所佔的資料傳輸時間比例大小來分配，讓資料傳輸時間比例小(換言之為資料執行時間比例較長)的處理器優先接收工作以利於提早執行，如此一來可以讓每個處理器減少等待時間。

本計畫在第二年提出了三個改良的 *SCR* 資源排程演算法分別為 *SCR-Best-fit*、*SCR-Worst-fit* and Extended *SCR* (*ESCR*)，除了先前的三個部份，第四部份分別利用 *Best-fit*、*Worst-fit* 與二元逼進法(Binary approximation method) 使每個處理器在有限的排程週期 (Scheduling Cycle)、或者在有限的工作結束時間內(Deadline)增加最大的工作接收與處理數量而不會造成資源的閒置與浪費。

*SCR* 演算法與 *Greedy*, *FPF* (Fast Processor First) 演算法 [5,6]最大的不同在於 *Greedy* 演算法為工作單一傳送至處理器做運算，在系統執行期間會因為沒有考慮到工作整批傳送的優點產生許多零碎的系統閒置時間，而 *FPF* 演算法雖然考慮到工作整批傳送，但忽略的異質性網路頻寬所造成的影響，導致有效率的處理器雖然收到比較多工作，但相對的也增加傳輸負載，如此一來其他的處理器必需等待更長的時間才可以接收工作。所以 *SCR* 演算法減少的處理器等待時間與閒置時間，進而提升整體輸出效能。

舉例說明 *SCR* 排程演算法，如圖二之架構，假設 $T_{1\_comm}$=5， $T_{2\_comm}$=2， $T_{3\_comm}$=1，$T_{4\_comm}$=3； $T_1$=3，$T_2$=6， $T_3$=11， $T_4$=13，分述如下：

第一步：在圖中，處理器依照單位工作執行時間( $T_i$)排序。假設有 *n* 個節點，則對處理器排序完後得到集合為<$P_1, P_2, \ldots, P_n$>。

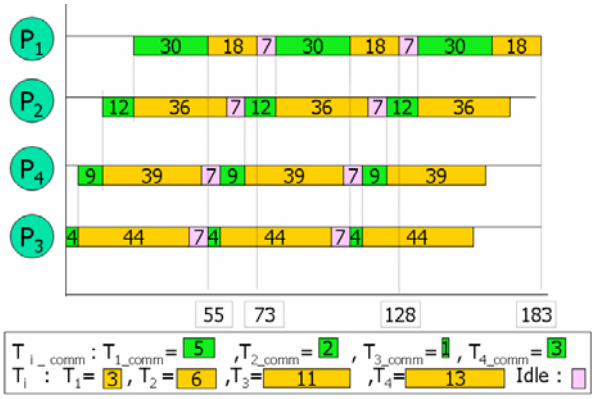
第二步：由步驟一排序節點的結果中所需之 $T_i$ 與各節點之單位工作傳輸時間 $T_{i\_comm}$ 計算其最小公倍數 *LCM*=(5+3, 2+6, 1+11, 3+13)= 48，計算出處理器 $P_1$ 到 $P_4$ 在每一個基本排程週期(*BSC*)接收(6, 6, 4, 3)個單位數量的工作。

第三步：依照所計算出的基本排程週期內每個處理器所佔的資料傳輸時間比例大小來分配，$P_3$ 資料傳輸時間比例小(換言之為資料執行時間比例較長)優先接收工作以利於提早執行，如此一來可以讓每個處理器減少等待時間，執行順序依序為 $P_3, P_4, P_2, P_1$。

如圖三中所顯示，*SCR* 演算法讓資料傳輸時間比例小的節點先接收工作，所以減少其他節點的等待時間，而參與計算的節點皆有接收工作並參與執行進而提升系統效能。

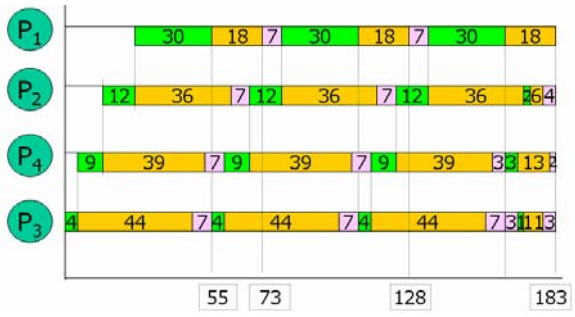
在此範例中工作排程為有限的三個排程週期(*BSC*)而工作結束時間(Deadline)為 183。在 $P_1$ 的最後一個排程週期結束之前，$P_2, P_3, P_4$,

皆有些許的閒置時間在等待 $P_1$ 結束正在執行的工作。為了善用這些可用的處理器資源，必需在 183 單位時間內增加傳送些許工作給 $P_2$ ,$P_3$, $P_4$。改良的 *SCR* 資源排程演算法 *SCR-Best-fit* 與 *SCR-Worst-fit* 將可解決這個問題。




圖三、*SCR* 排程演算法模擬示意圖。

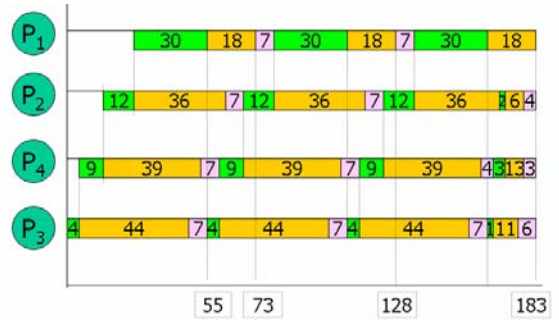
如圖四中所顯示，*SCR-Best-fit* 演算法除了讓資料傳輸時間比例小的節點先接收工作之外，每個運算節點逼近 Deadline = 183 時皆有接收工作並參與執行進而提升系統效能。最後一個週期的閒置時間為 22 單位時間。




圖四、*SCR-Best-fit* 排程演算法模擬示意圖。

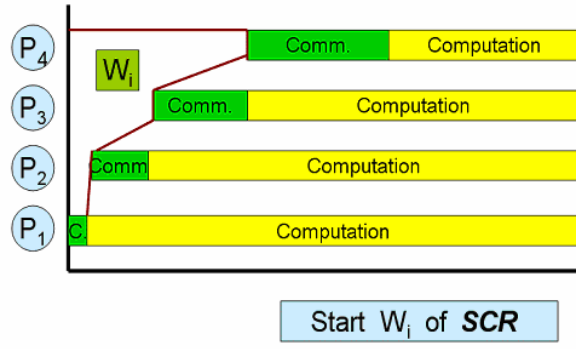
如圖五中所顯示，*SCR-Worst-fit* 演算法中，每個運算節點逼近 Deadline = 183 時皆有接收工作並參與執行進而提升系統效能。與 *SCR-Best-fit* 演算法的不同僅在於最後的工作分配順序不同導致系統閒置時間(idle)有些微差異。最後一個週期的閒置時間為 24。

為了證實本計畫開發的排程演算法為有效率的排程演算法，我們利用同一個架構下的範例之 Largest Communication Ratio (*LCR*)與 *FPF* 演算法排程結果做為比較。可以觀察到在同一個架構下，*LCR* 與 *FPF* 演算法著重於較高的執行節點優先接收工作導致處理器閒置時間與系統初始等待時間皆大於 *SCR* 演算法。

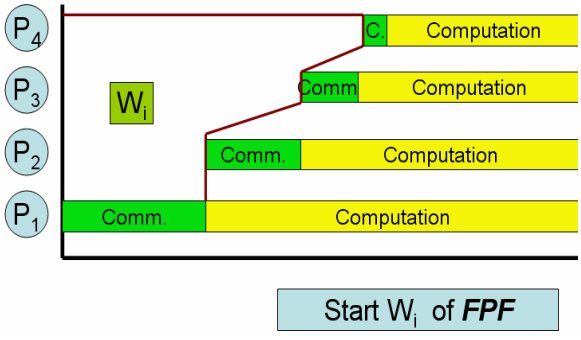



圖五、*SCR-Worst-fit* 排程演算法模擬示意圖。

如圖六(a)與(b)所示，依照處理器效能排序，執行順序依序為 $P_1$, $P_2$, $P_3$,, $P_4$，而 $P_2$, $P_3$,, $P_4$ 的初始等待時間總合 $W_i$ 大於 *SCR* 演算法，某些情況甚至 *FPF* 演算法的閒置時間過長導致 $P_4$ 無法參與計算而造成浪費，降低系統效能。




(a)

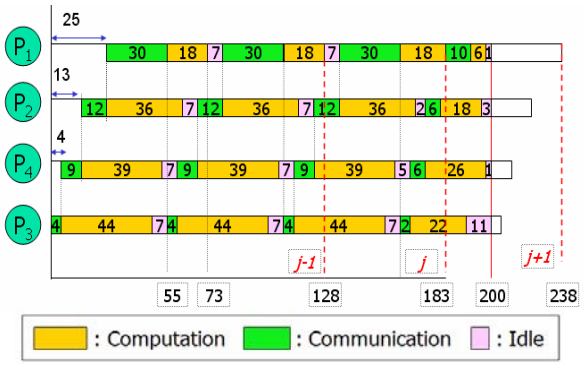



(b)

圖六、(a) *SCR* 排程演算法模擬示意圖、
(b) *FPF* 排程演算法模擬示意圖。

為了讓整體系統效能更加提升與具有彈性，我們設計了 Extended *SCR* (*ESCR*)的演算法，在 *SCR* 排程中的閒置空間，利用二元逼進法使每個處理器在系統所設定的排程週期內與在有限的工作結束時間內增加最大的工作處理數量。

如圖七中所顯示，此範例中，*ESCR* 演算法利用二元逼進法使每個處理器在第 *j*-1 到 *j*+1 的排程週期內指定完成 66 個工作。每個運算節點逼近 Deadline = 199 時皆有接收工作並參與執行進而提升系統效能。*ESCR* 可依照使用者指定固定的工作數量或最後的截止時間，最後一個週期的閒置時間為 12。




圖七、*ESCR* 排程演算法模擬示意圖。

如圖八中 *ESCR* 的二元逼進法，當指定固定的工作數量時，系統會自動辨別工作完成時間落在那一個週期，並且找出 Makespan 將可確保最短的時間內完成指定的工作數量。

*Algorithm_ESCR_Binary_Approcimation* ($T_i$, $T_{i\_comm}$, $Q_{task}$)

// $Q_{task}$ is the amount of tasks to be processed

01. **While ( !(** $Task_{finish}^{ESCR}(x) = Q_{task}$ **) )** {

02. $Left\_t = T_{finish}^{SCR}(BSC_{j-1})$;

03. $Right\_t = T_{finish}^{SCR}(BSC_{j+1})$;

04. $x$=1/2(*Left_t*+ *Right_t*);

05. **if** ( $Task_{finish}^{ESCR}(x) > Q_{task}$)

06. $x$=1/2($x$+ *Right_t*)

07. **else if** ( $Task_{finish}^{ESCR}(x) < Q_{task}$)

*08.* $x$=1/2(*Left_t*+$x$) }

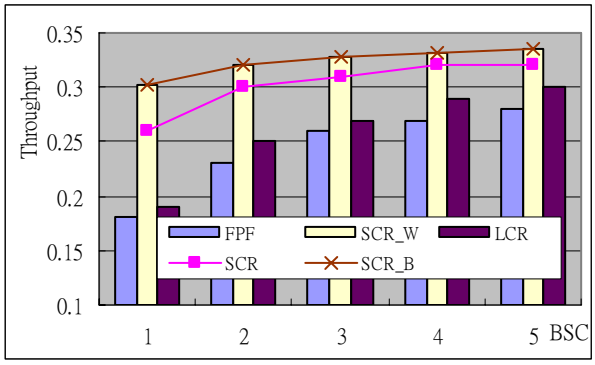
09. *makespan* = $x$;

*End_of_ESCR_Binary_Approcimation*

圖八、*ESCR* 的二元逼進演算法

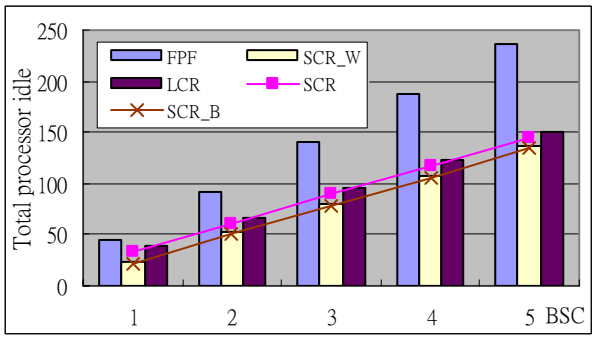
我們針對不同的處理器配置情形與格網異質性網路環境和各個工作排程演算法進行分析，發現排程演算法最佳化或與花費最少排程時間與處理器的異質性有關聯。

圖九的數據模擬為設定每個處理器取得的計算能力為正負十，而網路頻寬差距為正負四，系統執行週期設定為 1 到 5 的基本排程週期，處理器的節點數量為五個節點的平均系統效能輸出比較。




圖九、不同基本排程週期下的 *FPF*，*SCR_W*，*LCR*，*SCR*，*SCR_B* 排程步驟效能輸出數據比較

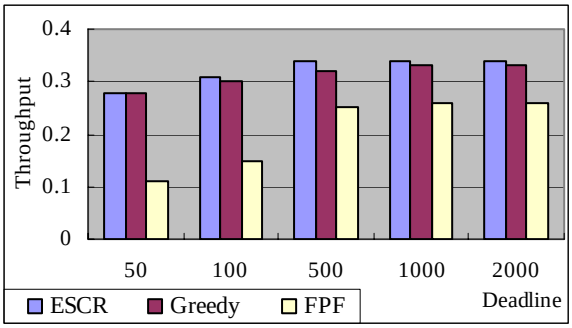
圖十的數據模擬為設定每個處理器取得的計算能力為正負十，而網路頻寬差距為正負四，系統執行週期設定為 1 到 5 的基本排程週期，處理器的節點數量為五個節點的處理器等待時間比較。




圖十、不同基本排程週期下的 *FPF*，*SCR_W*，*LCR*，*SCR*，*SCR_B* 排程步驟處理器等待時間比較

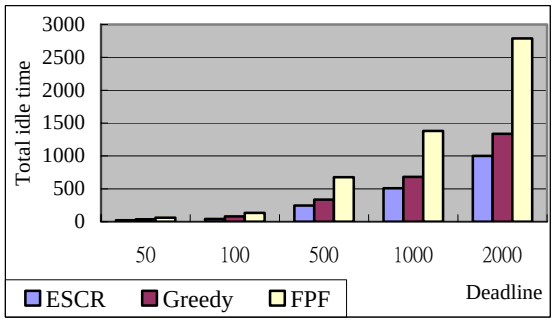
圖十一的數據模擬為設定每個處理器取得的計算能力為正負十，而網路頻寬差距為正負四，系統執行時間設定為 50 到 2000，處理器的節點數量為五個節點的處理器系統效能

輸出比較。



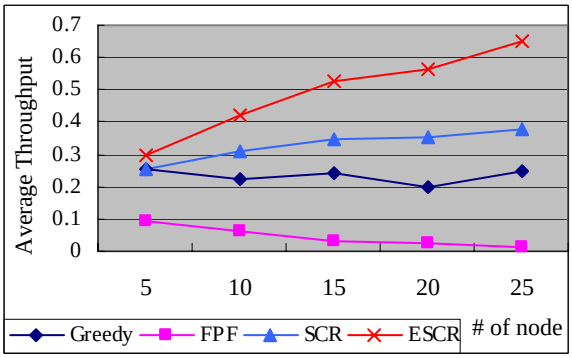

圖十一、不同系統執行時間下的 *ESCR*，*Greedy*，*FPF* 排程步驟效能輸出數據比較

圖十二的數據模擬為設定每個處理器取得的計算能力為正負十，而網路頻寬差距為正負四，系統執行時間設定為 50 到 2000，處理器的節點數量為五個節點的處理器等待時間比較。



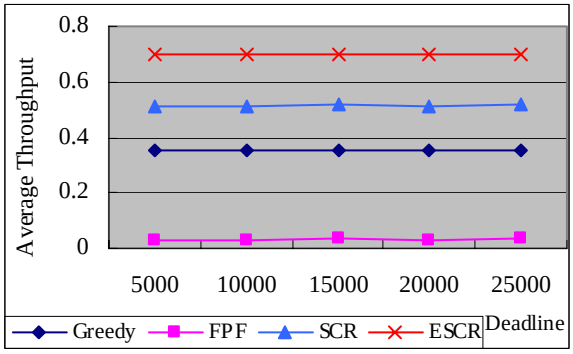

圖十二、不同節點數目下的 *ESCR*，*Greedy*，*FPF* 排程步驟處理器等待時間比較

圖十三的數據模擬為設定每個處理器取得的計算能力為正負五至正負十，而網路頻寬差距為正負五至正負十，系統執行時間設定為 10000，處理器的節點數量為五個節點至二十五個節點的處理器系統平均效能輸出比較。



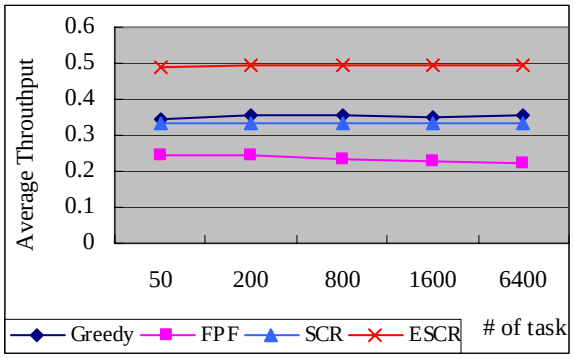

圖十三、不同節點數目下的 *Greedy*、*FPF*，*SCR*，*ESCR* 排程步驟平均效能輸出數據比較

圖十四的數據模擬為設定每個處理器取得的計算能力為正負十，而網路頻寬差距為正負五，系統執行時間設定為 5000 至 25000，處理器的節點數量為二十五個節點的處理器系統平均效能輸出比較。



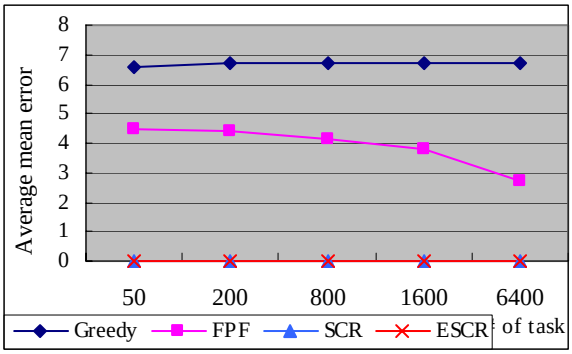

圖十四、不同系統執行時間下的 *Greedy*、*FPF*，*SCR*，*ESCR* 排程步驟平均效能輸出數據比較

圖十五的數據模擬為設定每個處理器取得的計算能力為正負十，而網路頻寬差距為正負五，系統工作數量設定為 50 至 6400，處理器的節點數量為十個節點的處理器系統平均效能輸出比較。




圖十五、不同工作數量下的 *Greedy*、*FPF*，*SCR*，*ESCR* 排程步驟平均效能輸出數據比較

圖十六的數據模擬為設定每個處理器取得的計算能力為正負十，而網路頻寬差距為正負十，系統工作數量設定為 50 至 6400，處理器的節點數量為十個節點的處理器系統平均錯誤比較。平均錯誤的定義為參與運算的所有處理器節點中，沒有收到工作的節點數量，*SCR* 與 *ESCR* 演算法每個節點皆有接受工作並且執行，不會產生系統錯誤。




圖十六、不同工作數量下的 *Greedy*、*FPF*，*SCR*，*ESCR* 排程步驟平均錯誤數據比較

在比較這些結果後，我們發現不管節點數量多或少、網路異質性的高與低、工作數量多寡，*ESCR* 仍然比其他演算法有更高的效能，本計畫中提出 *ESCR* 排程演算法明顯地勝出其他演算法許多，尤其在節點數量多的時候，*ESCR* 演算法與其他演算法有更大的系統效能輸出差距，並且有最少的系統等待時間。

**四、結論與討論**

下面我們歸納本計畫主要的成果:

- 完成發展單一叢集網格系統之主從式工作排程、並且實作應用在 *HPHC* (Heterogeneous Processor with Heterogeneous Communication) 架構下，以實現在異質性叢及網格計算環境中高效率的執行工作排班程式。
- 完成 *SCR*-scheduler 之演算法實作

  我們完成 *SCR* 之演算法實作，可用來判斷資料傳送與執行的最少排程步驟。針對不同數量大小的工作與資料分配性的問題，設計一套處理器優先序列的計算模式、以及基本排程週期計算的公式，研究效能評估機制對整體網路架構的必要性及其在效能上影響。
- 完成 *SCR-Best-fit* 與 *SCR-Worst-fit* 之演算法實作

  完成 *SCR-Best-fit* 與 *SCR-Worst-fit* 之演算法實作，用來增加資料傳送量。與 *SCR* 比較以相同執行環境下，增加了利用 *Best-fit* 與 *Worst-fit* 兩種演算法評估週期計算的最佳化，使排程系統善用整體網路資源，比單純使用*SCR*-scheduler有更好的效能。
- 完成*ESCR*-scheduler 之演算法實作

  完成*ESCR*-scheduler 之演算法實作，用來增加資料傳送與減少執行時間，以彌補 *SCR-Best-fit* 與 *SCR-Worst-fit* 之演算法的不足之處。除了先前設計的處理器優先序列的計算模式、更增加了利用二元逼近演算法評估週期計算的最佳化，使排程系統對整體網路資源有最高的使用率。
- 完成Minimum-Deadline資料分配之排程演算法實作

  針對最短時間排程步驟分析，所設計的排程演算法可以找出最短的系統執行時間(Makespan) 以確保在固定工作量的系統運作下，處理器的運作時間與閒置時間比其他演算法更短。
- 完成Maximum-Job資料分配之排程演算法實作

  針對最大工作量排程步驟分析，所設計的排程演算法可以在指定的系統執行時間(Makespan)內，確保在長系統運作時，處理器可以做出最有效率的系統輸出，亦及在時間內與其他演算法比較起來可以完成最多的工作數量。
- 完成動態網格拓僕模擬。

  針對不同叢集式計算網格架構、我們建立一個可以動態評估外部通訊效能的模式。這一個部分的工作包括，閘道計算、網路基礎頻寬拓樸模擬、即時網路資訊擷取、與權重計算方法。
- 完成*Greedy*, *FPF* (Fast Processor First) algorithm [5,6], 以及 *LCR* (Largest Communication Ratio) [1] 之實作。

  為了證實本計畫開發的排程演算法為有效率的排程演算法，需實做其他排程演算法以便進行實驗。

- 完成用於分析工作排程步驟與網路傳輸頻寬的理論模組
  為了比較排程演算法的優缺點，我們完成了資料傳輸模擬的理論模組，用以判斷排程結果的好壞。
- 完成資料傳送所引起的頻寬競爭之研究
  資料傳送至處理器的過程中會引起處理器互相競爭，增加了系統閒置時間。我們的排程演算法成功地減少了資料傳送時引起的通訊競爭與初始等待時間。
- 完成處理器與網路頻寬配置變數產生器
  為了模擬實際的異質性處理器運算變數，我們實做了一個子處理器與網路模擬產生器，產生高效能運算單元集中於某些處理器與異質性網路頻寬集中或平均分散在各個處理器上的配置變數。

## 五、計畫成果自評

本計畫兩年之研究成果已達到計畫預期之目標。第一年度、在這一個研究主題上共計發表兩篇研討會論文[1, 2]。其中成果 [2] Performance Effective Pre-scheduling Strategy for Heterogeneous Communication Grid Systems 已經被接受於 *Future Generation Computer Science* 期刊（SCI）。 第二年度共計發表兩篇研討會論文[3, 4]。其中成果 [4] An Efficient Job Allocation Method for Master Slave Paradigm with Heterogeneous Networks in Ubiquitous Environments 已經被接受於 *Journal of Supercomputing* 期刊（SCI）。

## 六、參考文獻


[1] Tai-Lung Chen and Ching-Hsien Hsu, "An Efficient Processor Selection Scheme for Master Slave Paradigm on Heterogeneous Networks," *Proceedings of Network and Parallel Computing* (NPC’06), Oct. 2006.

[2] Ching-Hsien Hsu, Tai-Lung Chen and Kuan-Ching Li, "Performance Effective Pre-scheduling Strategy for Heterogeneous Communication Grid Systems," *Future Generation Computer Science, Elsevier*, Vol. 23, Issue 4, pp. 569-579, May 2007. Elsevier, (SCI, EI)

[3] Ching-Hsien Hsu and Tai-Lung Chen, "An Efficient Task Dispatching Method in Heterogeneous Networks," IEEE Proceedings of the 2007 International Conference on Multimedia and Ubiquitous Engineering (MUE’07), pp. 17-22, April 2007. (EI)

[4] 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)

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

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

[7] Oliver Beaumont, Arnaud Legrand and Yves Robert, “Pipelining Broadcasts on Heterogeneous Platforms,” *IEEE Trans. on parallel and distributed systems*, Vol. 16, No.4, pp. 300-313 April 2005.

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

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

[9] 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.

[10] 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.

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

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

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

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

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

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

[17] Ching-Hsien Hsu and Tai-Long Chen, “Grid Enabled Master Slave Task Scheduling for Heterogeneous Processor Paradigm,” *Grid and Cooperative Computing - Lecture Notes in Computer Science*, Vol. 3795, pp. 449-454, Springer-Verlag, Dec. 2005. (GCC’05) (SCI Expanded)

[18] G. Aloisio, M. Cafaro, E. Blasi and I. Epicoco, “The Grid Resource Broker, a Ubiquitous Grid Computing Framework,” Journal of Scientific Programming, Vol. 10, No. 2, pp. 113-119, 2002.

[19] W. E. Allcock, I. Foster, R. Madduri. “Reliable Data Transport: A Critical Service for the Grid.” *Building Service Based Grids Workshop, Global Grid Forum*, June 2004.

[20] B. Allcock, J. Bester, J. Bresnahan, A. L. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnal, S. Tuecke. “Data Management and Transfer in High Performance Computational Grid Environments.” *Parallel Computing Journal*, Vol. 28 (5), May 2002.

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

[22] J. Blythe, E. Deelman, Y. Gil, C. Kesselman, A.Agarwal, G. Mehta and K. Vahi, “The role of planning in grid computing,” Proceedings of ICAPS’03, 2003.

[23] H. Casanova, “Simgrid: A Toolkit for the Simulation of Application Scheduling,”

Proceeding in IEEE Int'l Symp. Cluster Computing and the Grid (CCGrid '01), pp. 430-437, May 2001.

[24] L. J. Chang, H.Y. Chen, H.C. Chang, K.C. Li, and C.T. Yang, "The Visuel Performance Analysis and Monitoring Tool for Cluster Environments", Proceedings of ICS'2004 International Computer Symposium, Taipei, Taiwan, 2004.

[25] M. Cai, A. Chervenak, M. Frank. "A Peer-to-Peer Replica Location Service Based on A Distributed Hash Table." *Proceedings of the SC2004 Conference* (*SC2004*), November 2004.

[26] M. Faerman, A. Birnbaum, H. Casanova and F. Berman, "Resource Allocation for Steerable Parallel Parameter Searches," Proceedings of GRID'02, 2002.

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

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

[29] N. Fujimoto, K. Hagihara, A Comparison among Grid Scheduling Algorithms for Independent Coarse-Grained Tasks, *Applicatios and the Internet Workshops*, 26-30 Jan. 2004, pp.629-35.

[30] J. Nabrzyski, J.M. Schopf, J. Weglarz (Eds), "Grid Resource Management" Kluwer Publishing, Fall 2003.

[31] K. Ranganathan and I. Foster. "Identifying Dynamic Replication Strategies for High Performance Data Grids" Proceedings of International Workshop on Grid Computing, Denver, CO, November 2002.

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

[33] Ming Wu and Xian-He Sun, "A General Self-adaptive Task Scheduling System for Non-dedicated Heterogeneous Computing," Proceeding in IEEE International Conference on Cluster Computing, 2003.

[34] Hui Wang, Minyi Guo, Sushil K. Prasad, Yi Pan, Wenxi Chen," An Efficient Algorithm for Irregular Redistributions in Parallelizing Compilers" *Proceedings of the 2003 International Symposium on Parallel and Distributed Processing and Applications*, pp. 76-87, July 2003.

[35] Chao-Tung Yang, Yu-Lun Kuo, and Chuan-Lin Lai, "Designing Computing Platform for BioGrid," International Journal of Computer Applications in Technology (IJCAT), Inderscience Publishers, ISSN (Paper): 0952-8091, UK, 2004.

[36] X. Zhang and J. Schopf. "Performance Analysis of the Globus Toolkit Monitoring and Discovery Service, MDS2," *Proceedings of the International Workshop on Middleware Performance* (*MP2004*), *part of the 23rd International Performance Computing and Communications Workshop (IPCCC)*, April 2004.

# 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<br>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<br>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.

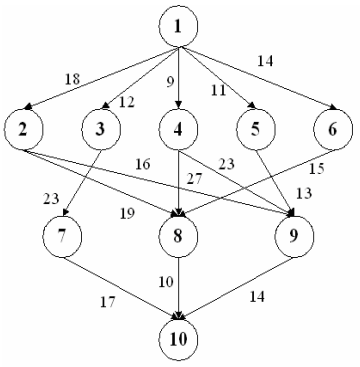
## 1. Introduction

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].

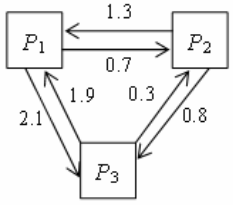
A Direct Acyclic Graph (DAG) [2] is usually used for modeling parallel applications that consists a number of tasks. The nodes of DAG correspond to tasks and the edges of which indicate the precedence constraints between tasks. In addition, the weight of an edge represents communication cost between 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)

| DP / SP | $P_1$ | $P_2$ | $P_3$ |
|---|---|---|---|
| $P_1$ | 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:

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

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,

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

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 $p_y$, $1 \leq x, y \leq P$.

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,

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

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,

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

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,

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

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***
1. Initially, construct an array of Boolean *QV* and a stack *S*.
2. $QV[n_j] = false, \forall\ n_j \in V$.
3. Push $n_{exit}$ on top of S.
4. **While** S is not empty **do**
5. Peek task $n_j$ on the top of S;
6. **If**( all $QV[n_i]$ are *true*, for all $n_i \in pred(n_j)$ or task $n_j$ is $n_{entry}$) {
7. Pop task $n_j$ from top of S and put $n_j$ into scheduling list *L*;
8. $QV[ n_j] = true$; }
9. **Else** /* search the $CT(n_j)$ */
10. **For** each task $n_i$, where $n_i \in pred(n_j)$ **do**
11. **If**($QV[n_i] = false$)
12. Put $CS(n_i)$ into container *C*;
13. **Endif**
14. Push tasks $pred(n_j)$ from *C* into *S* by non-decreasing order according to their critical scores;
15. Reset *C* to empty;

16. /* if there are 2+ tasks with same $CS(n_i)$, task $n_i$ is randomly pushed into $S$.
17. **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 |

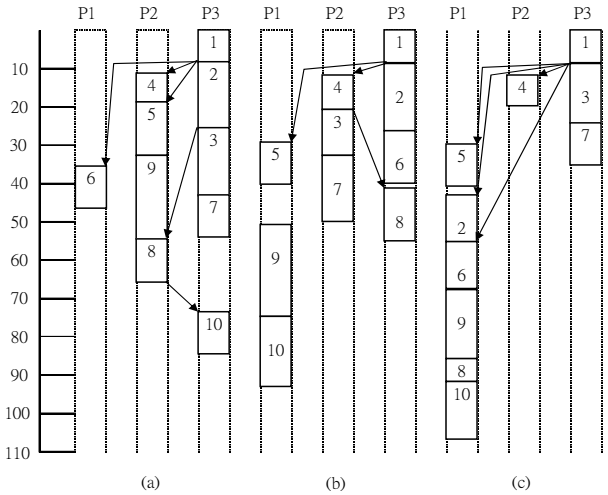



Figure 2: Schedule results of three algorithms on DAG-1 (a) *GCA* (makespan = 84.7) (b) *CA* (makespan = 92.4) (c) *HEFT* (makespan = 108.2).

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:

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

| 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)

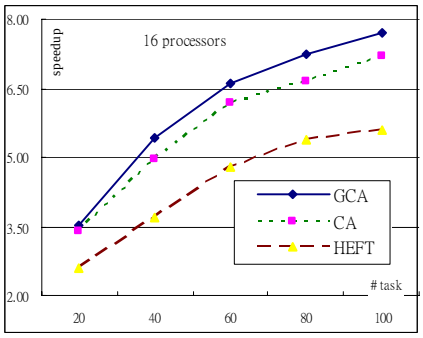



Figure 5: Speedup of *GCA*, *CA* and *HEFT* with different number of tasks (*n*).

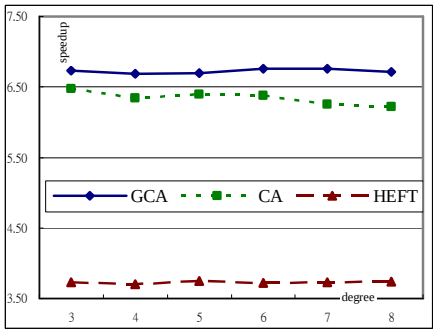
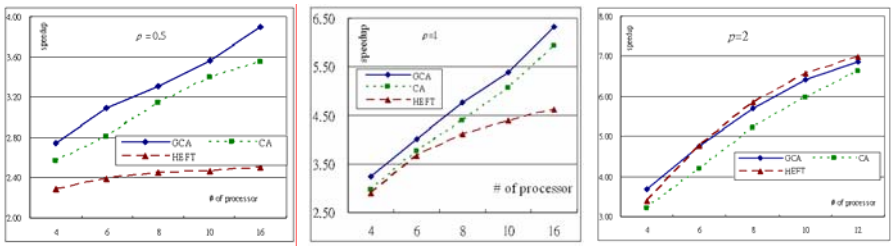



Figure 6: Speedup of *GCA*, *CA* and *HEFT* with different out-degree of tasks (*d*)

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.

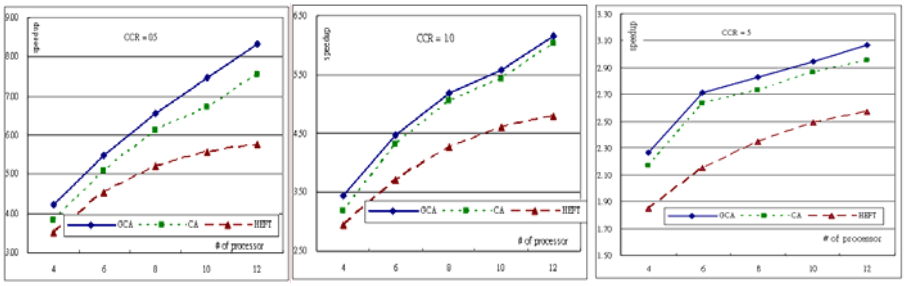
Speedup of the *GCA*, *CA* and *HEFT* algorithms to execute different DAGs with fix processor number (*P*=16) and task number (*n*=60) under different out-degree of tasks (*d*) are shown in Figure 6. The results of Figure 6 demonstrate the speedup influence by task dependence. We observe that speedups 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<br>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<br>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

**(出席 AINA-08 研討會所發表之論文)**

# 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.

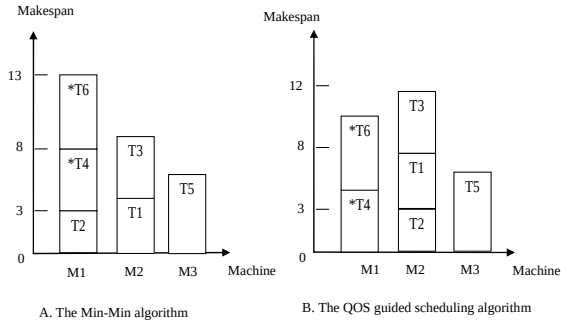
```
Algorithm_QOS-Min-Min()
{
  for all tasks ti in meta-task Mv (in an arbitrary order)
      for all hosts mj (in a fixed arbitrary order)
          CTij = etij + dtj
      end for
  end for
  do until all tasks with QoS request in Mv are mapped
       for each task with high QoS in Mv,
            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 CTli 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 CTli 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 |




A. The Min-Min algorithm

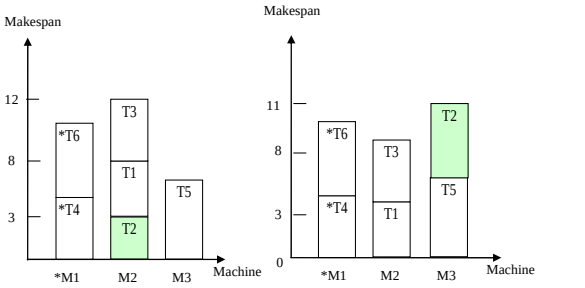
B. The QOS guided scheduling algorithm

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 $i \ne k$, 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*)

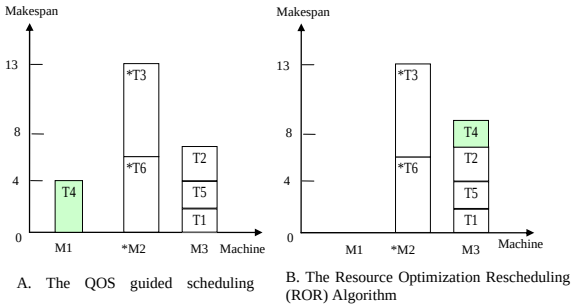
Following the assumptions described in *MOR*, the main idea of the *ROR* scheme is to re-dispatch tasks from the machine with minimum number of tasks to other machines, expecting a decrease of resource need. Consequently, if we can dispatch all tasks from machine $M_x$ to other machines, the total amount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

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

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
                  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.