# 在網格環境中以工作分支度為基準之動態排程

**邱紹豐　　謝中揚**
**大葉大學資訊工程學系**
**r9406035@mail.dyu.edu.tw**

## 摘要

現今的科學計算與模擬，常必須仰賴大量的儲存空間或龐大的運算能力，若使用高效能的超級電腦或叢集運算，所需花費的成本是非常高的。眾多學者們認為整合分散式異質資源的網格技術，可解決大量運算時資源需求的問題，並且預期該技術將會成為未來的運算型態。在此架構下，使用者所提交的工作會被拆解成有序子工作，並將各子工作分派至適當的處理單元處理。在大量工作需要處理時，傳統使用的先至先服務(First Come First Serve)排程機制並不會分析工作特性，以致效能不佳。而本研究於排程時分析每個工作的特性，以評估工作之優先權。再以處理單元之處理能力，推算工作完成時間，找出最適合該工作的處理單元。藉此提早工作完成時間。

**關鍵字**: 網格、動態排程、異質性處理單元

## 1. 序論

網格是一種透過網際網路來分享運算資源及資料的一種開放標準與技術。在網格環境下，使用者可透過終端設備提出工作要求，工作會被拆解成可分散處理的子工作，再透過網路交由其它適當的處理資源。由於無法事先預測使用者所提交工作模式，因此必須要有一動態排程機制決定工作間的優先權與處理單元的分派。

本文針對使用者在網格環境中所提交工作的不確定性，設計了一動態排程及資源分派的策略，以最高回應比優先(Highest Response Ratio Next, or HRRN)，並修改最多連外分支度優先(Most Outgoing Branch First, or MOBF)來評估子工作之優先權，且於分派前預估子工作於處理單元中執行的完成時間，以推測出可最早完成該工作的處理單元，進而縮短整體的工作完成時間。

本論文架構如下，第二章我們介紹關於網格環境中排程與資源分派的相關研究，並討論其應用於動態排程時所遭遇的困難。第三章為我們依 HRRN 與修正之 MOBF 所設計的動態排程演算法，稱之為 MOBF$^+$。第四章為我們的實驗及與其它方法比較的結果。第五章則為結論與我們未來研究方向的討論。

## 2. 相關研究

網格之目的為整合分散的異質資源，透過整合資源來提供大量的運算能力。在分散式運算中，使用者所提出的工作被拆解成多個可平行處理的子工作，一般以有向非循環圖(Directed Acyclic Graph，或是 DAG)描述之。以圖 1 為例，該圖描述一個編號 $d$ 之 DAG，該 DAG 包含$\{t_{d,0}$、$t_{d,1}$、$t_{d,2}$、$t_{d,3}$、$t_{d,4}$、$t_{d,5}$、$t_{d,6}\}$7 個子工作節點，其中 $t_{d,0}$ 為 DAG 之工作進入點(entry task)，以 $t_{d,\text{entry}}$ 表示，$t_{d,6}$ 為工作結束點(exit task)，以 $t_{d,\text{exit}}$ 表示。每個子工作有其工作負載量 WL(節點左下方數字)及工作類型 WT(節點右下方數字)，例如 $t_{d,4}$ 的工作負載量 WL($d$, 4) = 5、工作類型 WT($d$, 4) = 4，而它的前置工作集合 PRED($t_{d,4}$) = $\{t_{d,1}, t_{d,2}\}$，後繼工作集合 SUCC($t_{d,4}$) = $\{t_{d,6}\}$。當某個子工作的前置工作集合為空集合，則代表該工作為 ready task，可開始執行。

傳統的網格環境，會針對每位使用者所提出的工作，選取若干個處理單元 (Processing Element, 後稱為 PE)以產生一個相對應的虛擬組織(Virtual Organization, 或 VO)，並且將工作分派給該虛擬組織，直到工作完成後虛擬組織才會解散[8, 11]。由於此種運行模式可能發生該工作目前只有一個子工作可處理，導致虛擬組織中大多數的 PE 都處於閒置狀態。相對的，若以子工作為單位分派至 PE，而不將整個使用者所提交的工作分派給虛擬組

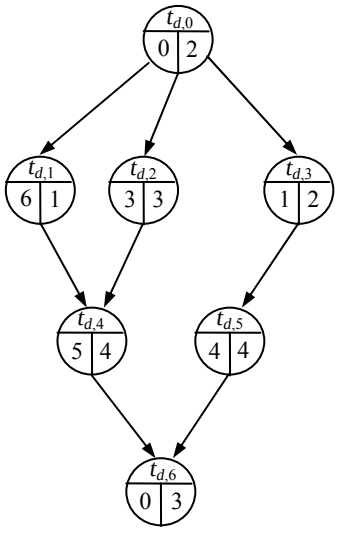


**圖 1　DAG 範例**

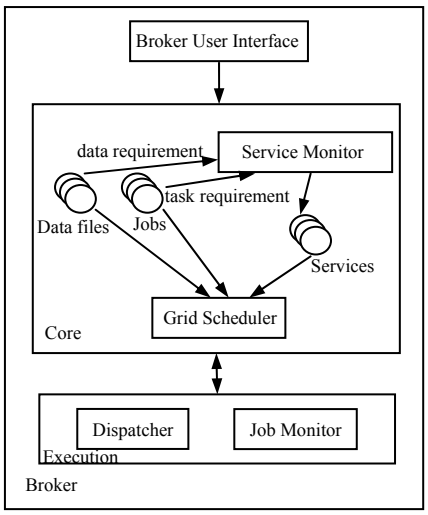


**圖 2 代理人架構示意圖**

織，便可改善此缺點[2]。在網格環境中，使用者與 PE 間是採 client/server 架構來運行，因此當使用者在使用 server 的資源時，無法提供自身的運算能力給其他工作。Michele Amoretti 等學者提出以 P2P 為基礎的網格[7]，將網格上所有的節點都視為同儕(Peer)。透過此架構，每個節點都可同時扮演服務提供者及消費者，以解決網格環境中消費者無法提供資源的問題。

此外，為了簡化使用者對資源的存取作業，Krishna Nadiminti 等人提出了代理人（Broker）技術[6]。代理人會將使用者提出的工作解析成子工作，並且找尋符合這些子工作類型的可用資源。代理人之架構可分為 Interface、Core、及 Execution 三層，其架構如圖 2 所示。Interface 轉換使用者送出的資料成為符合 Core 運作的格式，Core 取得資料後，經由 Service Monitor 尋找可用資源的位置及相關資訊，並以 Grid Scheduler 進行工作排程並將排程的結果由 Execution 送至相對應的資源執行。

在此架構中以 Grid Scheduler 對整體網格效能的影響最為明顯。傳統常用的演算法包含了 First Come First Serve (FCFS)、Shortest Job First (SJF)、及 Highest Response Ratio Next (HRRN)[5]。FCFS 的優點在於其工作分派之公平性，但也造成了即使目前有閒置的 PE，負載小的工作可能因大工作費時的運算而只能被迫延後處理。SJF 則以最小負載量的工作優先處理，可明顯提升 PE 的使用率，但由於優先選擇最小負載量的工作，將可能導致負載量大的工作長時間處於等待狀態，造成飢餓(starvation)現象[1]。作業系統中常用的 HRRN 權重評估(式 3)，會因等待時間越長而提升其優先權，有效的解決了 SJF 所造成的飢餓問題。在 DAG 中當一個子工作之後繼工作數量越高，代表該工作完成後會使較多的子工作成為 ready task。而越多的子工作成為 ready task，PE 的使用率也就相對的提高。因此，Most Outgoing Branch First (MOBF)便依據子工作之後繼工作數量決定其優先權，如式 1 所示。子工作之後繼工作 SUCC($t_{d,i}$)的數量越大，其優先權也相對提高。DSGE (Dynamic Scheduling for Grid Environment)以 HRRN 及 MOBF 之乘積來評估子工作之優先權（如式 2 所示），經實驗後結果顯示有較好的效能[2]。

$$\mathrm{MOBF}(t_{d,i}) = \frac{\mathrm{SUCC}(t_{d,i})}{\sum \mathrm{SUCC}(t_{d,j})}, \text{where } t_{d,j} \text{ is a ready task} \quad (1)$$

$$\mathrm{DSGE}(t_{d,i}) = \mathrm{HRRN}(t_{d,i}) \times \mathrm{MOBF}(t_{d,i}) \quad (2)$$

Srikumar Venugopal 等學者針對代理人架構中的 Grid Scheduler，設計一排程演算法[10]，目的在於解決科學運算中，資料大量且分散儲存的工作排程問題。該演算法分為兩部分，Initialization 及 Scheduling Loop。Initialization 用於確認運算資源的相關資訊，再交由 Scheduling Loop 排程。Scheduling Loop 會計算每個工作其最早處理完成的 PE，並將工作指派給該 PE，一旦有新工作加入，所有未處理的工作便會重新排程[10]。由於該策略只要有新工作提出，就必須重排所有未處理的工作，導致 Scheduler 的運算量大幅提升。

當網格中大多數工作的工作負載量都偏低時，PE 可快速的處理完這些工作負載低的工作，進而要求下一個工作。但頻繁的工作要求也造成了 PE 及網路的額外負擔。因此，Nithiapidary Muthuvelu 等人提出利用 PE 於特定時間內可處理的負載量，將工作組合成符合該負載量之工作群組，再一次送至 PE，來避免頻繁的工作要求[9]。

針對各種不同的排程演算法，Bruno de Azevedo Vianna 等人也提出了一個工具，來幫助使用者設計和評估網格環境中各種排程演算法的優劣[3]。該工具整合了 Static Scheduling 及 Dynamic Scheduling 的技術[4]，以期達到以下三點目的：

1. 針對網格環境設計一個合適的 heuristics。
2. 評估所選的演算法是否達到訂定的標準，例如演算法執行的時間。
3. 最後利用模擬或是實際的執行來證實它在執行上是可行的。

## 3. 排程演算法

在圖 3 中，A、C、E 三個子工作均擁有相同的後繼工作數量，若以 MOBF 評估原則，它們會有相同的優先權。但分析它們後繼工作的特性，可得知 E 完成後便可立即再釋放一個 ready task F，而 task A

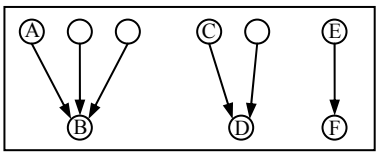


**圖 3 分支權重範例**

與 C 完成並不會馬上影響到後繼工作成為 ready task。因此，若先處理子工作 E 來釋放更多 ready task，便可提高 PE 的使用率。本研究針對 MOBF 在排程過程中可能遇到上述的問題，提供一修正的演算法，以提升整體效能。

### 3.1 HRRN 優先權評估

HRRN 優先權評估如式 3 所示，由於每個子工作的工作負載量 WL($d$,$i$)不會改變，等待時間 WAIT($t_{d,i}$)越長，優先權也會跟著提高。而 WL($d$,$i$)較小的 task 優先權提升的速度會高於 WL($d$,$i$)大的 task，這是由於工作負載量大的 task 通常擁有較大的延遲時間容忍，因此評估優先權時，工作負載量小的 task 應該要優先處理。

$$\mathrm{P_{HRRN}}(d,i) = 1 + \frac{\mathrm{WAIT}(t_{d,i})}{\mathrm{WL}(d,i)} \tag{3}$$

### 3.2 MOBF$^+$分支度評估

當某一個子工作完成，該工作對於後繼工作的影響程度會依照後繼工作的前置工作數量而有所不同。如圖 3 之範例所示，當 A 完成後，對 B 而言它的前置工作完成了 1/3。同樣的，C 完成對 D 之前置工作數量的影響為 1/2。而 E 完成對於 F 之前置工作數量的影響則為 1/1。

利用上述之概念，我們修改了 MOBF 中權重評估的方式，提出 MOBF$^+$ (Most Outgoing Branch First Plus)的評估原則。MOBF$^+$是以子工作分支度的權重和，在總 ready task 分支度中所佔的比例，來評估優先權。子工作分支裡每個分支的權重值，以該分支的前置工作數量為考量，前置工作越多，該分支所得權重越低，當前置工作只剩 1 個時，則該分支的權重為最大值 1。此處|PRED($t_{d,i}$)|表示子工作 $t_{d,i}$ 的前置工作數量，S1($t_{d,i}$)與 Sn($t_{d,i}$)分別表示 SUCC($t_{d,i}$)第一個及最後一個元素，工作 $t_{d,i}$ 的分支的權重和(TotalBranchWeight($d$, $i$), 或稱 TBW($d$, $i$))可表示如式 4。利用式 4 之結果，計算 $t_{d,i}$ 之分支權重和在總分支度中的比例 $\mathrm{P_{MOBF+}}(d,i)$，其中的 $\sum \mathrm{B_{readyTask}}$ 代表目前所有 ready task 分支度的和，如式 5 所示。

$$\mathrm{TBW}(d,i) = \sum_{e=\mathrm{S1}(t_{d,i})}^{\mathrm{Sn}(t_{d,i})} \frac{1}{|\mathrm{PRED}(e)|} \tag{4}$$

$$\mathrm{P_{MOBF+}}(d,i) = \frac{\mathrm{TBW}(d,i)}{\sum \mathrm{B_{readyTask}}} \tag{5}$$

### 3.3 子工作優先權

本研究利用 $\mathrm{P_{HRRN}}(d,i)$及 $\mathrm{P_{MOBF+}}(d,i)$之評估結果來推得子工作 $t_{d,i}$ 的優先權 P($d$,$i$)，如式 6。當 $\mathrm{P_{HRRN}}(d,i)$上升時，$\mathrm{P_{MOBF+}}(d,i)$越大的子工作，優先權上升的幅度也要越大。本文採用 HRRN 與 MOBF$^+$ 之乘積作為子工作優先權的評估。該項評估值會套用至排程演算法中的 ready task queue，用於評估每個 ready task 的優先權，以決定其分派順序。

$$\mathrm{P}(d,i) = \mathrm{P_{HRRN}}(d,i) \times \mathrm{P_{MOBF+}}(d,i) \tag{6}$$

### 3.4 PE 優先權評估

本文工作分派演算法中，由於 ready task 會優先挑選較早釋放的 PE，因此必須建立一優先權佇列 (Priority Queue)，稱之為 *PEQ* 以儲存每個 PE 釋放的時間點，時間點越早則優先權越高。以 $pe_j$ 表示編號 $j$ 之 PE，釋放時間 release time 以 rt 表示，則 PE 之優先權 PRI($pe_j$)如式 7 所示。

$$\mathrm{PRI}(pe_j) = \frac{1}{pe_j.\mathrm{rt}} \tag{7}$$

### 3.5 服務時間

每個 PE 皆有其提供的工作類型與處理能力，以 CP($pe_j$, WT($d$,$i$))表示 $pe_j$ 對 WT($d$,$i$)此種工作類型的處理能力。子工作 $t_{d,i}$ 於 $pe_j$ 執行所需花費的時間成本以 COMP($t_{d,i}$, $pe_j$)表示，則該成本的計算方式可表示如式 8。

$$\mathrm{COMP}(t_{d,i}, pe_j) = \begin{cases} \dfrac{\mathrm{WL}(d,i)}{\mathrm{CP}(pe_j, \mathrm{WT}(d,i))}, & \text{if } \mathrm{CP}(pe_j, \mathrm{WT}(d,i)) > 0 \\ \infty, & \text{otherwise} \end{cases} \tag{8}$$

### 3.6 分派策略

為了不影響網格的效能，本研究設計了一預測分派演算法。該演算法會針對接下來即將釋放的 PE，事先預測要分派給它的工作。一旦 PE 完成工作，便可即時取得後續工作，以減少 PE 閒置的時間。觸發事件及演算法可整理如表 1。

**表 1 觸發事件及其演算法**

| 觸發事件 | 演算法 |
|---|---|
| 使用者提出工作 | issueJob |
| PE 完成工作 | taskCompleted |

issueJob(DAG$_d$, *threshold*)
1. ENQUEUE($t_{d,entry}$)
2. **if** has idle PE **then**
3. assignIdlePE(*threshold*)
4. forecastNextPE(next release PE, *threshold*)

**圖 4 issueJob 演算法**

使用者提出工作後，所觸發的 issueJob 演算法如圖 4 所示。*threshold* 為事先定義的門檻值，目的在於決定工作所會被測試的最大次數，若將 *threshold* 定義為 10，則工作最多只會與 10 個 PE 進行測試，當 *threshold* 越大，越能找出適合該工作的 PE，但測試所耗費的運算成本也越高。首先演算法會將工作之 entry task 加入 ready task queue （後以 *readyTaskQ* 表示）。若目前有閒置的 PE，則執行 assignIdlePE 演算法為閒置的 PE 找尋適合的工作。最後再以 forecastNextPE 演算法為下一個釋放的 PE 進行工作預測。assignIdlePE 及 forecastNextPE 演算法詳見圖 6 及圖 7。

```
taskCompleted(pe_j, t_d,i, threshold)
1.  for each task t ∈ SUCC(t_d,i) do
2.    if PRED(t) = 1 then
3.      ENQUEUE(t)
4.  remove t_d,i form DAG_d
5.  pe_j.status ← "available"
6.  if ft ≠ null then
7.    assign ft to pe_j
8.    DEQUEUE(ft)
9.    pe_j.status ← "busy"
10.   updateWeight(PEQ, pe_j)
11. if has idle PE then
12.   assignIdlePE(threshold)
13. forecastNextPE(next release PE, threshold)
```

**圖 5 taskCompleted 演算法**

taskCompleted 演算法會在 task 完成後被觸發，內容如圖 5 所示。圖中之 *ft* 用於暫存下一個釋放的 PE 會被指派的工作。1 至 3 行將 $t_{d,i}$ 之後繼工作為 ready task 的子工作加入 *readyTaskQ*。接著把 $t_{d,i}$ 從 $DAG_d$ 中移除，然後將 $pe_j$ 的狀態設為 "available"。6 至 10 行檢查 *ft* 中是否有儲存預測工作，有則將工作分派給 $pe_j$，再把 *ft* 由 *readyTaskQ* 中移除，並將 $pe_j$ 狀態設為"busy"，最後更新 $pe_j$ 在 *PEQ* 中的權重。第 11 至 12 行檢查是否有 PE 閒置，若有則觸發 assignIdlePE 演算法。再由第 13 行為下一個釋放的 PE 進行預測。

```
assignIdlePE(threshold)
1.  ect ← ∞
2.  ePE ← null
3.  for each task t ∈ readyTaskQ do
4.    for each of first threshold PE p ∈ PEQ do
5.      if p.rt < ect then
6.        if COMP(t, p) + Max(p.rt, now) < ect than
7.          ect ← COMP(t, p) + Max(p.rt, now)
8.          ePE ← p
9.      else
10.       p ← threshold^th PE of PEQ
11.   if ePE.status = "available" then
12.     assign t to ePE
13.     DEQUEUE(t)
14.     ePE.status ← "busy"
15.     updateWeight(PEQ, ePE)
16.   if status of all PEs are "busy" then
17.     exit
```

**圖 6 assignIdlePE 演算法**

圖 6 演算法用來為閒置的 PE 分派工作。演算法中的 *ect* 用於儲存目前測試之工作可最早被完成的時間，*ePE* 用於儲存最早完成時間之 PE，*now* 則為目前的系統時間。首先 3 至 10 行演算法依序由 *readyTaskQ* 中取出工作，而測試的 PE 個數在不超過 *threshold* 下，依序以 *PEQ* 中各 PE 的處理能力推算處理完成時間，並將最早完成工作的 PE 及完成時間記錄於 *ePE* 與 *ect*。若測試途中發現目前測試的 PE 其釋放時間已大於記錄的 *ect*，代表之後 PE 完成工作的時間不可能再小於目前記錄的 *ect*，所以直接將 *p* 設為 *threshold*'th 的 PE，以結束該工作的測試。第 11 至 15 行檢查 *ePE* 目前是否閒置，若是則將工作分派給它，然後把 *t* 從 *readyTaskQ* 中移除，再將 *ePE* 的狀態設為"busy"，並更新 *PEQ* 中 *ePE* 的權重。接著取出下一個工作繼續測試，直到 *readyTaskQ* 中的工作都測試過或所有的 PE 均忙碌才結束演算法。

```
forecastNextPE(pe_j, threshold)
1.  ect ← ∞
2.  ePE ← null
3.  ft ← null
4.  Build forecastRTQ
5.  for each Task t ∈ forecastRTQ do
6.    for each of first threshold PE p ∈ PEQ do
7.      if Max(p.rt, now) < ect then
8.        if COMP(t, p) + Max(p.rt, now) < ect than
9.          ect ← COMP(t, p) + Max(p.rt, now)
10.         ePE ← p
11.     else
12.       p ← threshold^th PE of PEQ
13.   if ePE = pe_j then
14.     ft ← t
15.     exit
```

**圖 7 forecastNextPE 演算法**

forecastNextPE 用於預測下一個釋放的 PE 後續所會被分派的工作，內容如圖 7。由於預測時所使用之 ready task queue，與原始的 ready task queue 內容會有差異，因此必須另外建立一預測用的 ready task queue。圖 7 以 *forecastRTQ* 表示。第 4 行的 *forecastRTQ* 除了複製原本 *readyTaskQ* 的內容，還必須加入預測的 $pe_j$ 完成工作後將產生的 ready task。且 *forecastRTQ* 在 HRRN 評估時，其等待時間 WAIT($t_{d,i}$)會以該 PE 釋放的時間點來評估，而不是使用目前的系統時間。演算法首先以 5 至 12 行計算出最適合工作 *t* 的 PE，再由 13 至 15 行檢查該 PE 是否為要預測的 $pe_j$，是則將該工作儲存於 *ft*，然後結束預測，否則將由 *readyTaskQ* 中取出下一個 task 繼續測試。

## 4. 實驗結果

本研究之初步測試分別以 25、50、100、200 個 DAG，均使用 Uniform 分佈，PE 數量分別為 10、20、30，PE 的平均處理能力為 1.5。使用者提交工作之 DAG 屬性的詳細資訊如表 2。

圖 8 為本研究之初步實驗結果，由實驗數據可發現，當使用者提交的工作(DAG)數量以倍數成長時，本方法之工作完成時間並不會跟著以倍數成長。例如在實驗中以 30 個 PE 處理 100 與 200 個 DAG，工作完成時間分別為 1222.422 及 2296.187，成長的速度均小於 2 倍。

為了與其它演算法進行比較，本研究也使用了和[2]相同的環境設定來進行測試。圖 9 為引用[2]之實驗結果與本文所提出之方法完成時間的比較圖。其中 FCFS、SJF、HRRN 完成所有工作的時間均在 14000 以上，DSGE 為 7135.031，而本文之演算法為 6203.018，為五種方法中最快完成工作的演算法。

**表 2 DAG 資訊**

| | |
|---|---|
| 平均負載量 | 600 |
| 平均 Task 數量 | 30 |
| 工作類型上限 | 10 |

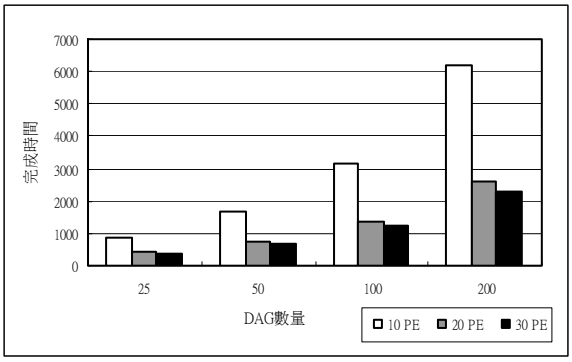


**圖 8 不同 DAG 及 PE 數量下之測試結果**

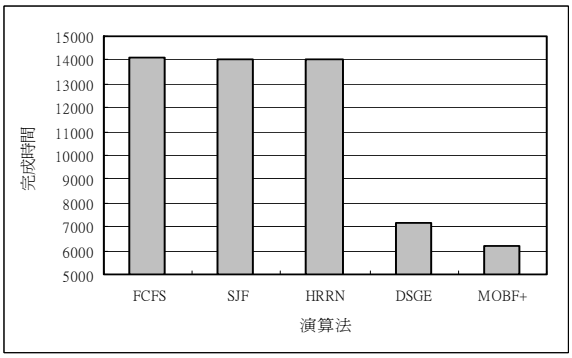


**圖 9 各種演算法下之工作完成時間**

## 5. 結論與未來工作

本研究針對在網格環境下，提出一動態工作排程及分派策略。相較於傳統的靜態排程機制，本研究所提出之方式，透過分析子工作特性，建立最佳的分派順序。而每個子工作在分派前，便事先預估該工作於各 PE 執行的完成時間，以推測出最適合的 PE。且為了不影響網格效能，演算法會在下一個 PE 釋放前，先行預測接下來要分派給它的工作，一旦 PE 釋放，便可立即再取得工作，避免 PE 閒置。對於網格環境中使用者所提交工作之不確定性，在初步的測試中我們所提出的機制確實能減短工作整體完成時間。

往後我們將擴大範例數量進行模擬，以驗證本演算法於網格環境中的效能表現，未來將更進一步導入 PE 與 PE 間資料傳輸的通訊成本(communication cost)。考慮子工作間傳輸資訊所需花費時間成本。當子工作所需傳送的資訊量越大，傳輸的時間成本越高，這也意味著子工作可能因此選擇頻寬較大的 PE，而不是處理能力最高的 PE，來降低傳輸時的時間成本。透過 communication cost 的概念，可更確切的推算出工作完成時間。

## 參考文獻

[1] Avi Silberschatz, Peter Baer Galvin, and Greg Gagne, Operating System Concepts, 7th Edition, John Wiley & Sons, 2004. (ISBN: 0-471-69466-5)

[2] Andy Schiou Chiou and Chen-Kun Tsung, “Dynamic Scheduling for Jobs in the Grid Environment,” the 3rd International Conference on Cybernetics and Information Technologies, Systems and Applications (CITSA 2006), July 2006.

[3] Bruno de Azevedo Vianna, Ariel Alves Fonseca, Nilmax Teones Moura, Luiz Toscano Menezes, Helder de Amorim Mendes, Jacques Alves da Silva, Cristina Boeres and Eugene Francis Vinod Rebello, “A Tool for the Design and Evaluation of Hybrid Scheduling Algorithms for Computational Grids,” Proceedings of the 2nd workshop on Middleware for grid computing, 2004.

[4] Cristina Boeres, Alexandre Lima and Eugene Francis Vinod Rebello, “Hybrid Task Scheduling: Integrating Static and Dynamic Heuristics,” Proceedings of the 15th Symposium on Computer Architecture and High Performance Computing, Nov. 2003.

[5] HARVEY M. DEITEL, Operating Systems Second Edition, Addison-Wesley Publishing Company, 1990. (ISBN 0-201-18038-3)

[6] Krishna Nadiminti, Hussein Gibbins, Xingchen Chu, Srikumar Venugopal and Rajkumar Buyya,

"The Gridbus Grid Service Broker and Scheduler (v.3.0) User Guide," Grid Computing and Distributed Systems (GRIDS) Laboratory, Department of Computer Science and Software Engineering, The University of Melbourne, Australia.

[7] Michele Amoretti, Francesco Zanichelli and Gianni Conte, "SP2A: a Service-oriented Framework for P2P-based Grids," Proceedings of the 3rd international workshop on Middleware for grid computing, 2005.

[8] Marcus Alexander, "Getting to Grips with the Virtual Organization," Long Range Planning, Elsevier, 1997.

[9] Nithiapidary Muthuvelu, Junyang Liu, Nay Lin Soe, Srikumar Venugopal, Anthony Sulistio and Rajkumar Buyya, "A Dynamic Job Grouping-Based Scheduling for Deploying Applications with Fine-Grained Tasks on Global Grids," Proceedings of the 2005 Australasian workshop on Grid computing and e-research, 2005.

[10] Srikumar Venugopal, Rajkumar Buyya and Lyle Winton, "A Grid Service Broker for Scheduling Distributed Data-Oriented Applications on Global Grids," Proceedings of the 2nd International Workshop on Middleware for Grid Computing, 2004.

[11] Yuan Pu Shao, Matthew Lee Kwok-on and Shao Yi Liao, "Virtual Organizations: The Key Dimensions," Academia/Industry Working Conference on Research Challenges, 2000.