計算格網中工作排程及處理器配置方法之研究

Job Scheduling and Processor Allocation in Computing Grid

賴冠伯
國立台中教育大學資訊科學學系
gkken77@gmail.com

黃國展
國立台中教育大學資訊科學學系
kchuang@mail.ntcu.edu.tw

李東穎
國立台中教育大學資訊科學學系
donyingle@gmail.com

摘要

格網計算的主要目的是將分散各地的電腦透過網路媒介，讓它們彼此分享資源、計算能力、協同完成一件大型計算，以達成工作負載平衡以及提升資源使用效率等目的。要達成此一目的，工作負載管理方法的好壞是關鍵因素。工作負載管理可分成兩個部分：工作排程及資源配置。本論文使用自行開發模擬環境來研究格網平台上的資源碎裂問題，我們嘗試分析不同的工作排程與處理器配置方法，以期許能找出能降低資源碎裂發生的機率。在研究過程中，本文提出一個新的most-fit處理器配置方法，實驗模擬結果顯示在某些環境的工作負載下達成最佳的系統效能。本論文的實驗結果顯示出SJF (Smallest Job First)的工作排程方法搭配Most-Fit的資源配置方法在大部分的工作負載模式下可達到最佳的效果。

關鍵詞： 工作排程、資源配置、資源碎裂

Abstract

Grid computing aims at integrating computing resources located at different places to share computing power and workload as well as to solve a large computational problem collaboratively. It can achieve load sharing and improve resource utilization. Workload management is a key factor to achieve the goal. Workload management can be viewed as two collaborative parts: job scheduling and resource allocation. We developed a simulation environment to study the resource fragmentation issue on grid platforms. We tried to compare and analyze different scheduling and allocation methods for finding out the method which can reduce resource fragmentation occurrences most. We developed a new resource allocation method called Most-Fit, which was shown superior to other methods under several workload conditions. The simulation studies in this thesis indicate SJF (Smallest Job First) scheduling method together with the Most-Fit allocation method outperformed other combinations of methods under most workload conditions.

Keywords: job scheduling, resource allocation, resource fragmentation

一、前言

格網計算（Grid computing）的主要目的是將分散各地的電腦透過網路媒介，讓它們彼此分享資源，分享計算能力，或一同完成一件大型計算的問題。亦可以將屬於不同管轄範圍的計算資源整合起來，做妥善的統一安排與運用，以達成

工作負載平衡及提升資源使用效率等目的。雖然格網計算有此潛在優點，但格網平台上的系統管理人員在處理工作負載管理問題時，常面臨不小的困擾。因進行工作負載管理時須面對許多決策關卡，例如要用哪一種工作排程方法、和處理器配置方法等等。要在處於不同的環境下，從不同方法中決定出最適合的方法，本身就不是易事，何況演算法的表現優劣常深受系統當中工作負載模式影響，更加深決策中的困難程度。不同的格網系統依據工作負載模式不同，必需選用適合的工作排程與處理器配置，才能達成系統最佳效益。實際上並無一種標準的方法可以一體適用於所有的格網系統。

因此，系統管理人員除了憑藉個人經驗外，需透過實驗的方式來挑選最適合其系統之工作負載模式的工作排程與處理器配置方法。這些實驗，會將之前一段時間範圍內系統上所處理過的工作，重新用別的工作排程與處理器配置方法來處理過一遍，以比較不同方法間的效能優劣。但透過實驗在實際上對外提供服務的格網平台上並不合適，因為會干擾使用者工作的效能。因此，在一個開放的系統上，系統管理者多半受限於許多實際因素，而無法進行相關的實驗研究。而此問題的解決辦法就是開發一個格網模擬平台，也可藉由模擬來比較工作排程和資源分配方法的好壞並有效的縮短時間。

好的工作排程方法可使工作完成時間有效縮短，而好的資源分配方法可使資源使用率更加提昇。也就是在同一時間能分配的工作越多，也就越能減少資源碎裂(Resource-Fragmentation)問題。本文的研究主要在開發能模擬格網平台上工作排程與處理器配置方法的操作環境。藉由此模擬環境，格網平台管理者可在佔用極少系統資源的情形下，僅須極短的時間就能模擬分析過去長時間內的工作負載。以及比較不同排程和處理器配置方法時所造成的效能差異。如，僅需數秒鐘的時間就能模擬完成原本兩年間的實際工作負載。本文即利用此一模擬環境來研究格網平台上的資源碎裂(resource fragmentation)問題，我們嘗試分析不同工作排程與處理器配置方法，以期許能找出降低發生資源碎裂的機率，即代表提升了格網平台的整體資源使用率。反映在使用者感受的，就是可降低執行工作所需的TurnaRound Time。在研究中，也提出新的most-fit處理器配置方法，結果顯示能在某些情況的工作負載下達成最佳系統效能。

本論文其餘章節內容安排如下：第二節描述模擬平台的功能架構及實例和相關研究。第三節探討資源碎裂問題和工作負載管理，將會提到格網內工作排程與處理器配置的討論。第四節為實驗環境描述、參數設定及實驗數據結果討論。第五節整理出結論及未來研究方向。

二、模擬平台

本論文模擬平台具有的功能與特點：

1. 可以支援不同數量的處理器個數。
2. 由小”PCs or Desktop”到大”Clusters 與格網”皆可模擬。
3. 支援同質或異質性的處理器速度。也可以模擬不同格網的工作負載。
4. 內部提供4個工作排程和6個資源配置方法機制。外部提供使用者加入工作排程或資源配置方法。
5. 內建4種機率分佈方法，供使用者在設計模擬工作時使用。
6. 可讓使用者依需求產生不同的機率分佈記錄檔。

本模擬平台提供三大功能。

模擬：模擬等待執行的工作。透過輸入的參數來執行不同負載模式下的工作，最後可顯示工作執行時間、佇列平均長度、平均完成時間、資源碎裂個數…等。**分析**：可進行工作局部分析，例如第幾個工作執行時不同格網的資源分佈情況，也可以使用”單步執行”來觀看詳細的工作內容

。**產生**：提供機率分佈的方法供使用者使用，為了使機率分佈的功能更加貼近使用者。也就是透過選擇產生何種記錄檔，和自訂不同參數，使用者就可以輕易產生記錄檔。

此處介紹四個相關模擬平台來和我們的平台做比較，分別是Bricks[1]、MicroGrid[2]、SimGrid[3]和GridSim[4]。

Bricks：由日本東京大學開發，主要針對高性能廣域計算環境中的各調度方案進行分析和比較。只允許使用集中式排程方法。廣域計算環境主要由三部份組成：客戶端代表客戶提交請求，服務器部份表可獲得的資源，網路部份代表客戶端和服務器之間的網路行為，調度單元的部份用來對各種模擬進行協調。

MicroGrid: 是由美國加州大學聖地牙哥分校開發，直接利用現有實體資源來摸擬一個虛擬的網格環境以執行真實的網格應用，等於是用一個實體案例的運行，來達到更真實評估網格系統的目的。

SimGrid: 也是由美國加州大學聖地牙哥分校的網格研究和創新實驗室(Grid Research And Innovation Laboratory)開發，主要目標是在網格環境中進行運行調度來提供一個合適的模型並產生準確的模擬結果。SimGrid主要使用C語言撰寫，因此在格網平台上的支援度不夠。

GridSim: 由澳大利亞墨爾本大學開發，主要目標是透過模擬來研究基於計算經濟模型的有效資源分配方法。此平台提供了分時和分工的支援，可應用到不同的平行或分散式系統中，像是叢集、格網等等，也支援利用消息事件來進行通信、最後也提供GridStatistics來收集各種模擬的統計數據。底下為平台間的比較分析。

應用真實性和應用類型比較如下:

應用真實性：多數的模擬器使用虛擬的應用程序，少數使用真實的網格應用程序。真實的應用可使試驗結果更趨精準，卻需花費較多的時間完成，因整個程序是真實的。而虛擬的應用是透過理論的基礎來定義時間的花費，所以真實性可供驗證，卻更適合大量的試驗模驗。

應用的類型：應用的類型可分「計算」和「數據」型。若模擬器所支援的應用主要和CPU資源相關為計算型，而和數據存取相關者為數據型

表2.1應用真實性和應用類型比較表

模擬器 / 應用模擬		Bricks	Micro Grid	Sim Grid	Grid Sim	本平台
真實性		虛擬	真實	虛擬	虛擬	虛擬
類型	計算相依性	√	√	√	√	√
	資料相依性	√	X	X	X	X

資源管理和使用特性分類的比較如下:

資源管理部份：主要是資源的分配策略，可分為「分時」(Time-Sharing) 和「分區」(Space-Sharing)。分時的部份指的是依時間為分界，可多工同時利用分時片段來進行工作，以達到一般多工同時執行的方式；而分區的方法，為只有當上一個使用資源的工作完成時才可以調度下一個工作來執行。

表 2.2 資源管理和使用特性分類比較表

模擬器	資源管理		圖形化使用者介面	是否包含資料庫	開發語言
	分時 Time-Sharing	分區 Space-Sharing			
Bricks	√	X	X	X	Java
Micro Grid	√	X	X	Massf [5]	C
Sim Grid	X	√	X	X	C
Grid Sim	√	√	√	Sim Java[6]	Java
本平台	X	√	√	X	Java

三、資源碎裂問題與工作負載管理

此章探討資源碎裂問題是如何產生，以及造成的影響，和工作排程與處理器配置方法間的關連。以及先前文獻上關於工作排程與處理器配置方法的相關研究和本文所提出的方法。

(一)資源碎裂問題?

何謂資源碎裂問題？資源碎裂是指在工作分配中，各個Site的處理器配置都無法分配給下一個工作使用，但全部Site的剩餘處理器加總值卻是足以分配給下一個工作使用，稱此為資源碎裂問題。如圖.3.1所示，使用First-Fit的處理器分配方法搭配FCFS的工作排程方法。當分配到第四個工作時，因沒有一個Site足夠分配給此工作使用而不能分配，但是所有Site的剩餘處理器個數總和是67，大於此工作所要的分配值45，這類的問題定義為資源碎裂問題。

資源碎裂問題和每個Site的處理器分配方法有關，因為使用不同的工作排程方法和處理器分配方法，就會造成不同程度的資源碎裂問題。理想的工作排程方法和處理器分配方法組合可使資源碎裂的問題變少，代表將工作分配得良好，讓資源有效的利用，這是發展一個工作排程方法和處理器分配方法的最大目的。

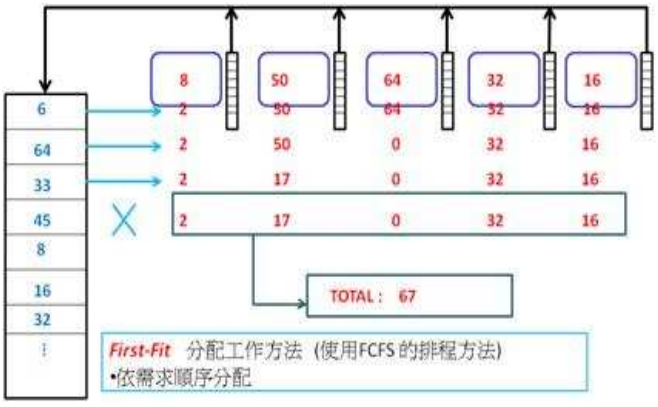


圖.3.1 一個簡易資源碎裂示意圖

(二)格網工作排程與處理器配置的相關研究

關於工作排程的議題已在傳統平行處理領域中研究多年。直到計算格網平台興起，陸續有關如何在格網平台上進行有效工作排程的研究成果發表。在這之前發表的研究成果當中，部分是修改傳統的序列排程(list scheduling)策略，使之適用於格網上[7,8]，另有些以經濟模式為基礎的方法也曾被討論過[9]。本文著重非經濟模式的排程和資源配置方法。

England 和 Weissman[10]中分析計算格網平台上進行平行工作的負載所須付出的時間代價及潛在效益，在同質和異質性格網中都進行相關的實驗研究。針對負載平衡，文獻中[11]曾指出Best-Fit資源配置方法可達最好的效果。在這個方法中，Site的選擇是以工作分配到這個Site後，能剩下愈少的處理器數目越好為原則。

先前研究工作也曾探討計算格網平台上支援multi-site平行工作執行的可能性及潛在效益[12,13]。在跨格網通訊時花費時間成本不高的情況下，multi-site平行工作執行被證實能改善工作的平均反應時間[7,8,11]。multi-site平行工作執行的主要額外時間成本來自不同格網間網路通訊時間大於同格網內部機器間網路通訊時間。此時額外網路通訊時間成本，以原本工作總執行時間的部分百分比來表示[7,11]。

在[14]中作者提出在異質性計算格網中支援multi-site平行工作執行的方法。然而，其所提出的方法需要事先預測工作的執行時間，因此在實務應用上有其限制。在文獻中[15,16]探討可塑性工作(moldable job)的排程與資源配置方法。這些研究前提假設工作執行時間可由所使用的處理器數量直接推估得知，或是需要使用者提供工作執行時間的預估值。

平行系統中，平行批次工作的排程主要利用變動分割法(variable partitioning)[17]，指每個工作使用平行電腦內部所有處理器的部份來進行

計算，通常不同工作間的處理器分配採用先到先服務(FCFS)的策略。然而，FCFS策略易形成資源碎裂並可能導致系統資源利用率的低落。

在FCFS的策略中，一旦系統目前所剩餘的處理器數量不能滿足佇列中的第一個工作時，其後所有等待工作亦不會被分配處理器，必須直到有足夠處理器數目滿足第一個工作，接續的工作才可陸續被分配到處理器執行。此策略會導致資源碎裂問題，雖然可用處理器數目比第一個工作需要的少，卻可滿足下一個工作的需求。但因採用FCFS的策略，導致接下來工作也必須跟著第一個工作一同延遲，原本開始執行的時間。因為FCFS策略不允許佇列中的工作不照順序執行。資源碎裂的形成對系統資源使用率和效能有很大的影響[18]。

（三）本論文提出的方法

工作排程指當有工作進入等待的佇列中時，取得佇列中第幾個工作來分配資源的方法。本模擬平台在工作排程方法中分別提供了FCFS (First Come First Served)、SJF (Shortest Job First)、LJF (Largest Job First)、SRF (Shortest RunTime First)4種。

FCFS (First Come First Served): 先進入佇列的工作先服務。是最簡單、最公平、而且容易了解的。做法是每次工作的取得都取最先進來的工作而不去選擇其他的，對於工作的使用者，FCFS是最公平的，但是效能不一定是最好。

SJF (Smallest Job First): 最小工作優先。指每次工作的取得都是由需求處理器數目最小的工作先執行，此排程策略是不公平的，會發生需要較多處理器數目的工作永遠無法執行。

LJF (Largest Job First): 最大工作優先。LJF和SJF相反。每次工作的取得都是由需求最大處理器數目的工作先執行，此排程策略是不公平的，有可能有需要小處理器數目的工作永遠執行不到。

SRF (Shortest RunTime First): 最短工作執行時間優先。此排程方法下，每一個工作都要預估執行時間。也就是在佇列等待工作中選出預估執行時間最少的先執行。

處理器配置方法指的是，當從上述排程方法中取得佇列中工作時，該如何分配工作到那一個Site去執行。本模擬平台在處理器配置方法方面提供了First-Fit、Best-Fit、Most-Fit 、Worst-Fit、Median-Fit、Random-Fit等六種方法。

First-Fit: 為開始配置工作時，從第一個Site開始配置，當一個Site不能配置時再依序往第二個Site配置下去，直到無法配置為止，例子可參考 圖.3.1。第一個工作到達需要6顆處理器，此時分配到第一個Site。第二個工作到達時需要64顆處理器，因第一個Site處理器不足，而分配到第二個Site，以此類推。

Best-Fit: 為在所有滿足此工作處理器需求Site中，具有最少處理器數目者，則將當下工作分配到此Site。假設當最少處理器數目有數個Site時，則以先分配的Site為主，依序配置至無法配置為止，可參考圖.3.2為例。第一個工作到達需6顆處理器，每個Site都可以分配，但第一個Site扣掉目前分配的處理器後剩餘處理器數目為2顆，是所有Site中剩餘最少的，所以工作將分配到第一個Site。第二個工作到達時需64顆處理器，因第一個和第五個Site處理器不足，而剩下的三個Site扣掉分配處理器數目後的剩餘處理器數目一樣，在一樣的情形下的決策是以排在較前面可分配Site為主，所以分配到第二個Site，以此類推。

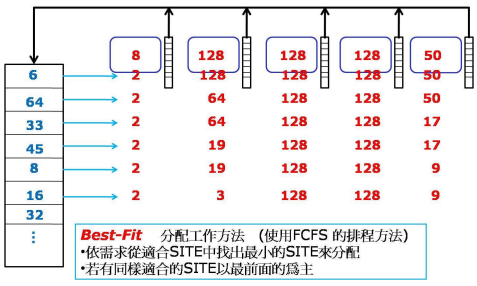


圖.3.2 Best-Fit 的分配方法例子

Most-Fit: 配置方法是以Best-Fit為基礎而改進。為統計所有Site可配置數，即可用處理器數目減去工作需求數目後，再往後可分配幾個工作執行。統計完後分配工作至可配置數最多的Site。假設當可配置數相同時，以Site中擁有最小處理器數目者為配置，而若是處理器數都相同，則與Best-Fit處理方式一樣，例子可參考圖.3.3、圖.3.4、圖.3.5。圖.3.3代表為預先假設，將工作分配到第一個Site時，接下來可配置數的統計。圖.3.4代表為預先假設，將工作分配到第二個Site時的可配置數統計。圖.3.5表示統計完所有Site可配置數時，則將工作分配至符合的Site。此範例中為Site 1。因此，導致Most-Fit較需時間來判斷，但這影響整體時間不大，不會影響到整體工作的執行效率。

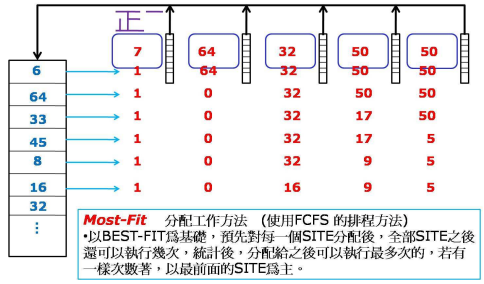


圖.3.3 Most-Fit　Site 1 可配置數統計

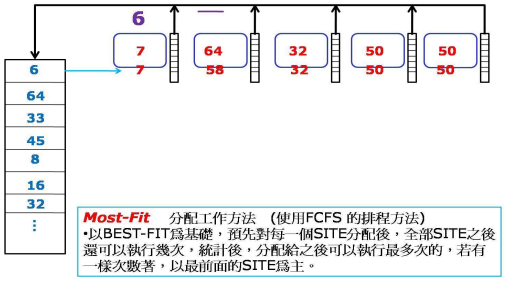


圖.3.4 Most-Fit　Site 2 可配置數統計

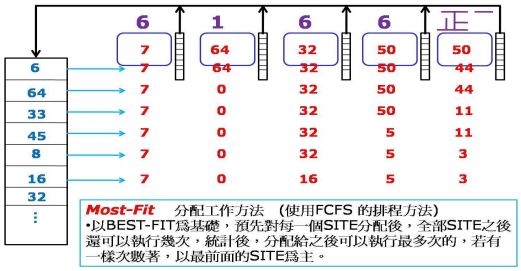


圖.3.5 Most-Fit 可配置數統計完成

Worst-Fit: 配置方法和Best-Fit配置方法相反。分配工作方法是在所有Site中，滿足此工作需求處理器數目，而具有最多處理器數的Site去配置，若有一樣大小處理器個數的Site時，則以較前方的Site為主，依序配置到無法配置為止，例子可參考圖.3.6。第一個工作到達需要6顆處理器，每個Site都可以分配，但第二到第四個Site扣掉目前分配的處理器後剩餘處理器數目為122顆，是所有Site中剩餘最多的，在一樣的情形下的決策是以前面可分配的Site為主，所以將工作分配到第二個Site。第二個工作到達時需要64顆處理器，因第三和第四個Site扣掉分配處理器數目後的剩餘處理器數目一樣剩64顆，在相同情形下的決策是以前面可分配的Site為主，所以分配到第三個Site，以此類推。

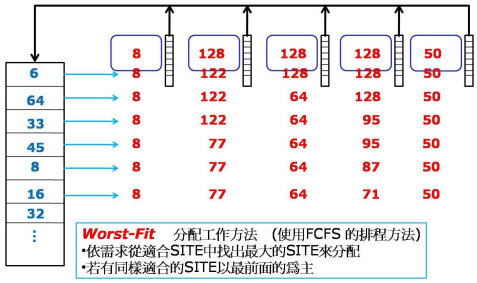


圖.3.6 Worst-Fit 的分配方法例子

Median-Fit: 配置方法是在所有符合工作需求的 Site 中，扣除工作所需處理器數目，選擇每個 Site 剩餘處理器數目為中間值的 Site。例子可參考圖.3.7。第一個工作到達需要 6 顆處理器，每個 Site 都可以配置，但第五個 Site 扣掉目前分配的處理器後剩餘處理器數目為 44 顆，是所有 Site 中剩餘數為中間值者，其他為 2 顆和 122 顆，所以工作將分配到第五個 Site。第二個工作到達時需要 64 顆處理器，只有第二到第四個 Site 扣掉分配處理器數目後的剩餘處理器數目一樣剩 64 顆，在一樣的情形下的決策是以前面可分配的 Site 為主，所以分配到第二個 Site，以此類推。

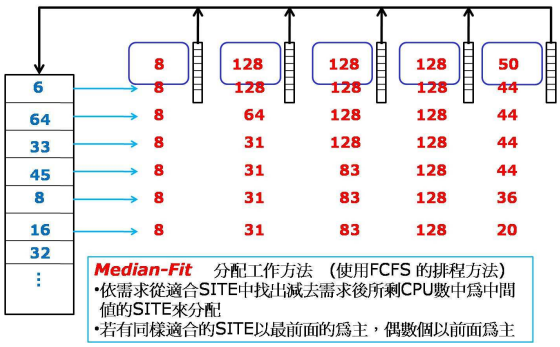


圖.3.7 Median-Fit 的分配方法例子

Random-Fit: 從能配置的Site中，用亂數去選擇配置的Site，因此，沒有固定的行為模式，因為是隨機的，若處理器夠分配，每次分配的都不一樣，任意分配。

以上六種是本模擬平台，內部提供處理器配置方法給使用者使用的方法。

四、實驗及討論

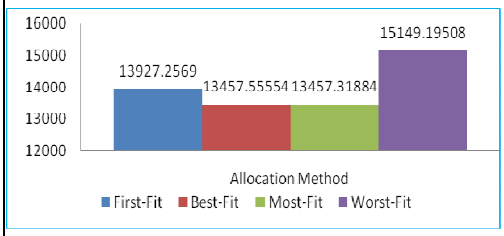
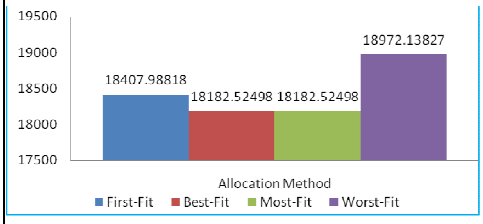
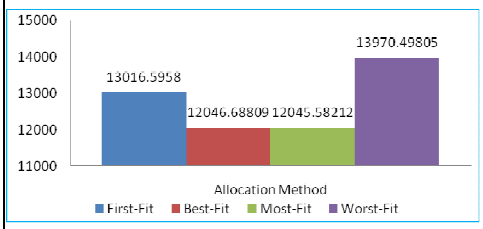
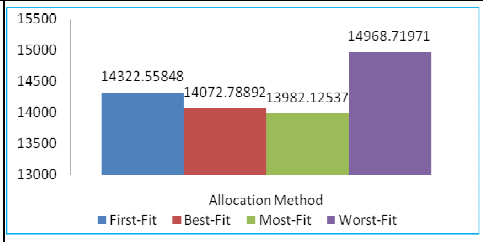
這一章敘述實驗的環境和參數設定以及實驗數據的結果與討論。

(一) 環境描述、參數設定及實驗數據呈現

分為一般工作檔和機率工作檔的模擬實驗分析。一般工作檔的部份，比較不同工作排程和處理器配置方法之間的好壞優劣。首先參數設定，跟據SDSC SP2記錄檔，設定各Site處理器數目為 8，128，128，128，50，而速度比(Speed)設定以1：1：1：1：1，限制佇列長度為範圍9-11以決定平均的負載比(Load)。

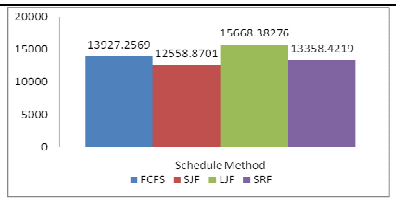
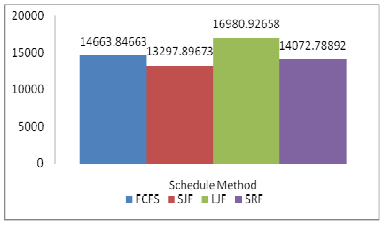
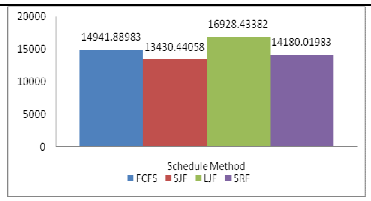
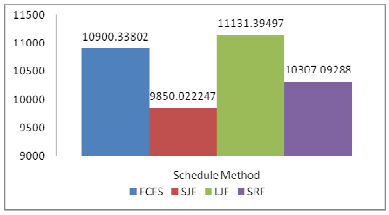
處理器配置方法比較為表4.1，此實驗固定工作排程方法，進行處理器配置方法之間的比較，觀察實驗數據中發現，使用不同的工作排程方法，在佇列有中等負載時(佇列長度 9 – 11)，Most-Fit的處理器配置方法表現是最好的，而Best-Fit次之。

表 4.1 SDSC SP2 處理器配置方法比較

佇列長度 / 排程方法	9 – 11
FCFS	 Most-fit < Best-fit < First-fit < Worst-fit
SJF	 Best-fit = Most-fit < First-fit < Worst-fit
LJF	 Most-fit < Best-fit< First-fit< Worst-fit
SRF	 Most-fit < Best-fit < First-fit < Worst-fit

固定處理器配置方法，工作排程方法和上方使用同樣的參數設定，只是將比較對象改為工作排程方法，結果如表4.2所示，觀察實驗數據中發現，工作排程方法的效能優劣順序皆是SJF < SRF < FCFS < LJF ，表示SJF在效能表現上一直是最好的。

表 4.2 SDSC SP2 工作排程方法比較

佇列長度 處理器 配置方法	9 – 11
First-Fit	 SJF < SRF < FCFS < LJF
Best-Fit	 SJF < SRF < FCFS < LJF
Most-Fit	 SJF < SRF < FCFS < LJF
Worst-Fit	 SJF < SRF < FCFS < LJF

比較不同排程和配置方法的組合與原始FCFS/First-Fit效能差異。環境設定各Site處理器數目為8，128，128，128，50 ，速度比1：1：1：1：1 ，負載比的乘積為2.2，比值1：1：1：1：1，佇列長度約為10，比較FCFS/ First-Fit、FCFS/Best-Fit、SJF/First-Fit、SJF/Best-Fit這四種組合，模擬結果如圖.4.1所示。實驗中SJF /Best-Fit組合效能為最佳，擁有最短的完成時間。從數據中可以推測出，好的排程方法改善完成時間比處理器配置方法更加有效益，而同

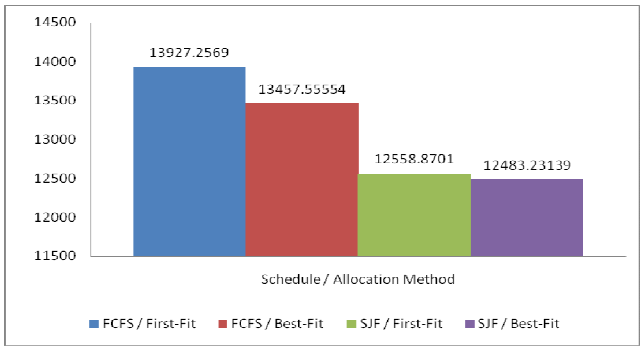


圖.4.1 SDSC 不同排程和配置方法的比較

時搭配一個好的排程方法和好的配置方法則可以達到加乘的效果。

比較不同機率分佈所產生的差異，環境設定各Site處理器數目為128，128，128，128，128，速度比(Speed) 1：1：1：1：1，限制佇列長度為範圍9-11以決定平均的負載比(Load)，工作到達時間(SubTime)上使用Possion機率分佈，記錄檔用參數600來產生，600是由SDSC工作記錄檔中各個工作的到達時間差，取平均值得之。在執行時間和處理器分佈的部份使用Normal、Uniform、Exponential機率分佈交互比較，執行時間參數為5481，由SDSC工作記錄檔中各個工作的平均執行時間所得。實驗結果的效能比較請看表4.3、表4.4、表4.5、表4.6，分別為不同排程方法下使用機率分佈方法的比較結果。由表4.3可知在使用FCFS排程方法時，不同機率分佈以Best-Fit最好而與Most-Fit差距接近。由表4.4可知在使用SJF排程方法時，Most-Fit和Best-Fit二者皆相等，都是最好，只有一個情形First-Fit比較好。由表4.5可以得知在使用LJF排程方法時，Most-Fit和Best-Fit在不同機率分佈有不同的優勢，也有二種相同的情況。由表4.6可看到使用SRF排程方法時，幾乎是Most-Fit最好，少數和Best-Fit一樣並列最好，只有一個是Best-Fit最好的情形。由以上四個表可知在不同的排程方法中，都幾乎是Most-fit和Best-Fit擁有最短的完成時間。

表 4.3 FCFS 排程與不同處理器配置方法比較

執行時間 / NP (64)	Normal	Uniform	Exponential
Normal	Most- fit = Best-fit < First-fit < Worst-fit	Most- fit = Best-fit < First-fit < Worst-fit	Most- fit = Best-fit< First-fit < Worst-fit
Uniform	Best-fit< Most -fit< First-fit < Worst-fit	Best-fit < Most -fit< First-fit < Worst-fit	Most -fit < Best-fit < First-fit< Worst-fit
Exponential	Best-fit< Most -fit < First-fit < Worst-fit	Best-fit < Most -fit < First-fit < Worst-fit	Most -fit < Best-fit < First-fit < Worst-fit

表 4.4 SJF 排程與不同處理器配置方法比較

執行時間 / NP (64)	Normal	Uniform	Exponential
Normal	First-fit< Most-fit= Best-fit< Worst-fit	Most-fit= Best-fit< First-fit< Worst-fit	Most-fit= Best-fit< First-fit< Worst-fit
Uniform	Most-fit= Best-fit< First-fit< Worst-fit	Most-fit= Best-fit< First-fit < Worst-fit	Most-fit= Best-fit< First-fit< Worst-fit
Exponential	Most-fit= Best-fit< First-fit< Worst-fit	Most-fit= Best-fit< First-fit< Worst-fit	Most-fit= Best-fit< First-fit< Worst-fit

表 4.5 LJF 排程與不同處理器配置方法比較

執行時間 / NP (64)	Normal	Uniform	Exponential
Normal	First-fit <Most-fit= Best-fit< Worst-fit	Most-fit= Best-fit< Worst-fit< First-fit	Most-fit= Best-fit< First-fit< Worst-fit
Uniform	Most-fit< Best-fit< First-fit< Worst-fit	Best-fit<Most -fit<First-fit< Worst-fit	Best-fit< Most-fit< First-fit< Worst-fit
Exponential	Best-fit< First-fit< Most-fit <Worst-fit	Most-fit< Best-fit< First-fit< Worst-fit	Best-fit< Most-fit< First-fit< Worst-fit

表 4.6 SRF 排程與不同處理器配置方法比較

執行時間 / NP (64)	Normal	Uniform	Exponential
Normal	Most-fit= Best-fit< First-fit< Worst-fit	Most-fit= Best-fit< First-fit< Worst-fit	Most-fit= Best-fit< First-fit< Worst-fit
Uniform	Most-fit< Best-fit< First-fit< Worst-fit	Most-fit< Best-fit< First-fit< Worst-fit	Most-fit< Best-fit< First-fit< Worst-fit
Exponential	Most-fit< Best-fit <First-fit <Worst-fit	Most-fit< Best-fit< First-fit< Worst-fit	Best-fit< Most-fit< First-fit< Worst-fit

再來就取機率分佈的案例來比較排程和處理器配置方法所帶來的效能改進幅度。環境設定各Site處理器數目為128，128，128，128，128為主，速度比 1：1：1：1：1，負載比的乘積為0.835，比值為1：1：1：1：1，機率分佈使用Possion (600)、Exponential (5481)、Normal(64)。比較FCFS/First-Fit、FCFS/Best-Fit、SJF/ First-Fit、SJF/Best-Fit這四種組合，結果為圖.4.2。結果和之前SDSC工作記錄檔的比較相同，一樣是排程方法所帶來的效能改進幅度較大，而SJF Best/Fit組合有最短的完成時間。

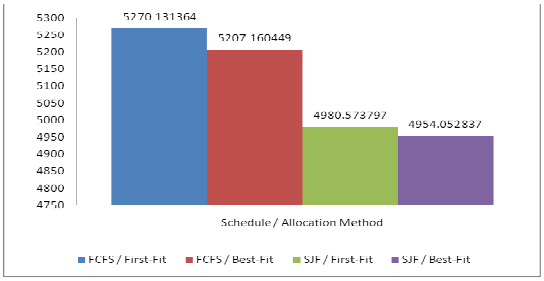


圖.4.2 機率分佈下不同排程和配置法的比較

(二) 實驗結果討論

整合上述的實驗結果，歸納出以下幾個具參考價值的重要結論：

1. 當 NP 的平均值占單一 Site 總處理器數目較少的比例時(例如 9/128)，Most-Fit 表現最佳，Best-Fit 次之，接著 First-Fit，而 Worst-Fit 表現最差，尤其在與 SJF 排程方法搭配時，Most-Fit 表現甚佳。
2. 當 NP 平均直接近每個 Site 處理器數目的一半時(64/128)，Most-Fit 及 Best-Fit 整體表現仍優於其它方法，當搭配 FCFS、LFJ 排程方法時 Best-Fit 表現優於 Most-Fit，而當搭配 SJF，SRF 排程方法時，Most-Fit 表現優於 Best-Fit。Most-Fit 在搭配 SJF 排程方法時表現最佳。
3. 在工作排程方法比較方面，結果相當一致，SJF 表現最佳，SRF 次之，此結果顯示出工作所需處理器數目對於排程效益影響大於工作所需時間。其中 LJF 的表現比 FCFS 還差，主要是當一個大的工作佔住資源後，其它許多個小的工作都被迫等候，整體平均的 TurnaRound Time 因此拉長（可由 LJF 方法導致較長的平均貯列長度看出）。
4. 單獨改變工作排程方法所能帶來的效能改進遠大於單獨改變處理器的配置方法。

五、結論與未來研究方向

本文針對格網計算環境開發一個工作負載管理的模擬平台，此平台可讓管理者藉由設定格網個數、格網內的處理器個數、速度比、負載比等參數來自行定義各種規模之格網環境。利用此模擬平台可進行排程方法或處理器配置方法的測試及分析比較。讓管理者能開發系統中的工作負載模式需搭配何種排程或處理器配置方法才能達到最大效益。藉由分析不同排程或處理器配置方法的優缺點原因，更可因此而設計出更具效益的方法。

我們實際利用此模擬平台來研究格網平台上的資源碎裂(Resource Fragmentation)問題，嘗試分析不同的工作排程與處理器配置方法，以期找出最能降低資源碎裂發生機率的方法。降低資源碎裂的發生機率即代表提升格網平台的整體資源使用效率，反映在使用者的感受上，就是可降低使用者執行工作所需的TurnaRound Time。研究過程中，本文提出的Most-Fit處理器配置方法，實驗結果顯示能在某些情況的工作負載下達成最佳的系統效能。

本文的實驗結果指出資源碎裂問題確實是

影響格網平台計算效能的一個重要因素，而選擇適當的工作排程與資源配置方法可大幅提升整體的計算效能，其中排程方法改良所能帶來的效能改進更大於資源分配方法的改善。實驗結果顯示SJF(Smallest Job First)的工作排程方法搭配Most-Fit的資源配置方法在大部分的工作負載模式下可達到最佳的效果。

本文利用所開發的模擬平台來進行資源碎裂問題的初步研究，未來可針對此論文實驗所得到的一些現象進行更深入的分析。例如進一步分析為何在某些情況下，Best-Fit方法能表現得比Most-Fit方法來得好的原因等等。此外，本論文實驗所假設的格網環境為一同質性的環境，由於此模擬平台可支援異質性格網環境的模擬，因此，異質性格網環境中工作負載管理方法的研究亦是未來一個重要的研究方向。

誌謝

本論文中之研究成果為執行國科會專題研究計畫之產出，計畫編號: NSC 96-2221-E-142 -006 -MY3

參考文獻

[1] Atsuko Takefusa, “Bricks: A Performance Evaluation System for Scheduling Algorithms on the Grids”, JSPS Workshop on Applied Information Technology for Science, 2001.

[2] Hyo J. Song, Xin Liu, Dennis Jakobsen, Ranjita Bhagwan, Xianan Zhang. “The MicroGrid: a Scientific Tool for Modeling Computational Grids”, Proceedings of Super Computing 2000, 2000.

[3] Henri Casanova. “SimGrid: A Toolkit for the Simulation of Application Scheduling”, Proceedings of the First IEEE/ACM International Symposium on Cluster Computing and the Grid, pp. 430-437, 2001.

[4] Rajkumar Buyya and Manzur Murshed. “GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing”, The Journal of Concurrency and Computation: Practice and Experience, Volume 14, Issue 13-15, Wiley Press, 2002.

[5] Xin Liu and Andrew Chien. “Traffic-based Load Balance for Scalable Network Emulation”, Proceedings of the ACM Conference on High Performance Computing and Networking, 2003.

[6] Fred Howell and Ross McNab. “SimJava: A Discrete Event Simulation Package For Java With Applications In Computer Systems Modelling”, First International Conference on Web-based Modelling and Simulation, Society for Computer Simulation, 1998.

[7] C. Ernemann, V. Hamscher, R. Yahyapour, and A. Streit, “Enhanced Algorithms for Multi-Site Scheduling”, Proceedings of 3rd International Workshop Grid 2002, in conjunction with Supercomputing 2002, pp. 219-231, Baltimore, MD, USA, November 2002.

[8] C. Ernemann, V. Hamscher, A. Streit, R. Yahyapour, “"On Effects of Machine Configurations on Parallel Job Scheduling in Computational Grids", Proceedings of International Conference on Architecture of Computing Systems, ARCS 2002, pp. 169-179, 2002.

[9] R. Buyya, D. Abramson, J. Giddy, H. Stockinger, “Economic Models for Resource Management and Scheduling in Grid Computing”, Special Issue on Grid Computing Environments, The Journal of Concurrency and Computation: Practice and Experience(CCPE), May 2002.

[10] D. England and J. B. Weissman, “Costs and Benefits of Load Sharing in Computational Grid”, 10th Workshop on Job Scheduling Strategies for Parallel Processing, Lecture Notes In Computer Science, Vol. 3277, June 2004.

[11] K. C. Huang and H. Y. Chang, “An Integrated Processor Allocation and Job Scheduling Approach to Workload

Management on Computing Grid”, Proceedings of the 2006 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'06), pp. 703-709, Las Vegas, USA, June 26-29, 2006.

[12] M. Brune, J. Gehring, A. Keller, A. Reinefeld, “Managing Clusters of Geographically Distributed High-Performance Computers”, Concurrency – Practice and Experience, 11(15): 887-911, 1999.

[13] A. I. D. Bucur and D. H. J. Epema, “The Performance of Processor Co-Allocation in Multicluster Systems”, Proceedings of the Third IEEE International Symposium on Cluster Computing and the Grid (CCGrid'03), pp. 302-, May 2003.

[14] W. Zhang, A. M. K. Cheng, M. Hu, “Multisite Co-allocation Algorithms for Computational Grid”, Proceedings of the 20th International Parallel and Distributed Processing Symposium, pp. 8-, April 2006.

[15] K. C. Huang, “Performance Evaluation of Adaptive Processor Allocation Policies for Moldable Parallel Batch Jobs”, Proceedings of the Third Workshop on Grid Technologies and Applications, December 7-8, 2006, Hsinchu, Taiwan.

[16] W. Cirne, F. Berman, “Using Moldability to Improve the Performance of Supercomputer Jobs”, Journal of Parallel and Distributed Computing, Volume 62, Number 10, pp. 1571-1601, October 2002.

[17] Feitelson, D. G., “A Survey of Scheduling in Multiprogrammed Parallel Systems”, Research Report RC 19790 (87657), IBM T. J. Watson Research Center, Oct. 1994.

[18] Krueger, P., Lai, T. H., Radiya, V. A., “Job Scheduling is More Important Than Processor Allocation for Hypercube Computers”, IEEE Transactions on Parallel and Distributed Systems, May 1994, pp. 488-497.