Resource Allocation on SMP Clusters

YI-MIN WANG*
Department of Computer Science and Information Management, Providence University, Taiwan

ABSTRACT

This paper studies the resource allocation of parallel jobs on SMP clusters. Previous parallel job scheduling algorithms, such as EASY (Extensible Argonne Scheduling sYstem), a backfilling algorithm, focus only on the allocation of CPUs. As communication cost becomes a bottleneck and gradually dominates the performance of program executions on multicomputers, some processor allocation policies suggest that as processors are allocated to a job, the allocated processors must be as continuous as possible. However, some of the mentioned algorithms will lead to another external fragmentation problem. The problem occurs as sufficient processors become available for the requested job; however, these processors are dispersed throughout the system. Thus jobs must wait for some time until there are sufficient contiguous processors. In summary, as adjacent processors are allocated, we may improve the run time of jobs, however, this will increase the wait-time of jobs.

In this paper, we suggest that an effective processor allocation policy should well balance communication cost and waiting delay. The principal is simple. When a network is busy or the communication cost is high, the processors must be allocated as continuously as possible. On the other hand, when the communication cost is light, to shorten the waiting time, the processors should be allocated as soon as possible.

Key words: cluster of SMPs; Resource Allocation; Communication Cost.

SMP Clusters 上資源分配之研究

王逸民*
靜宜大學資訊管理系,台灣

摘要

本論文主要研究平行工作(parallel jobs)在 SMP 叢集上，CPU 和網路資源分配之探討。首先在平行工作 scheduling 演算法方面，如果分配給這些 jobs 的 CPUs 的位置，是集中在少數 SMPs 上，communication 的機會較小，因此對網路的負擔也較小，但會增加 waiting time；反之，如果分散到許多 SMPs 上，waiting time 會降低，但 communication 較多，對網路的競爭也較嚴重。

本論文以效能較佳的 backfilling 演算法為實驗對象，我們在模擬器上，利用有公信力的 trace，在不同的網路通訊負擔的假設之下，執行各項實驗。本實驗證實：當系統要分配 CPUs 給 job 時，可以依照當時的網路的負載程度的輕重，動態調整這些 CPUs 的集中程度(Tight 值)，原則是網路的負載程度重時則要求集中，網路的負載程度輕時則放寬集中程度。然後去尋找是否有足夠的 CPUs 個數且符合 Tight 值，若有則將這些 CPUs 分配出去，否則等待其他 job(s)釋出 CPUs 後再行判斷是否可以進行後續的分配工作。

關鍵詞：叢集電腦、資源分配、通訊負擔。

* E-mail: ymwang@pu.edu.tw

壹、簡介

本論文主要研究在 SMPs (symmetric multiprocessors) 叢集上，平行工作資源分配(parallel jobs resources allocation)之問題。先前許多研究顯示，在平行電腦上的資源分配策略，例如常見的 First Come First Serve (FCFS)及各種 backfilling scheduling algorithms 等，大多只注重 CPUs (or processors) 的分配，主要目的是增加 CPUs 的使用率，也就是盡量讓 CPUs 分配出去，減少閒置(Feitelson，2005; Mu'alem & Feitelson，2001; Tsafrir, Etsion & Feitelson, 2007;Talby & Feitelson, 2005)。

另一方面，有些研究則建議分配給 parallel jobs 的 CPUs，儘量集中在一起（或稱為連續），這些研究以 Mesh 架構(Hosseini-Moghaddam & Naghibzadeh, 2006; Seo, 2005; Zhu, 1992)或 Torus 架構(Mao, Chen & Watson III，2005)的平行電腦為研究對象。在這些架構下，若分配到某 job 的 CPUs 太過分散，communication cost 會非常大。因此，這一類方法的主要觀念是：寧願多花一點時間等待，讓得到的 CPUs 儘量集中在一起，以降低 communication overhead。上述方法，在所謂 communication intensive jobs，因為 communication cost 是主要的瓶頸，效能會比不考慮 CPUs 位置的方法好，缺點是有 waiting time 過長，以及 external fragmentation 的問題。

本論文所假設的 clusters of SMPs 架構，如圖一所示，有 Q 個 SMPs，每個 SMP 包含 P 個 CPUs。在此架構下，parallel jobs 的執行，除了前面提及有關 CPUs 分配的問題之外，其實還有其他個因素要考量：例如在 SMP 內部共享記憶體的競爭，以及整個系統對於網路的競爭。其中前者在 Wang, Chien, Chang and Tien(2008) 有詳細的討論，本論文則著重在後者的探討。

理想的資源分配演算法希望能達到下列目標：整體 parallel jobs 的平均 slowdown 和 turnaround time 要降低、整體 CPU 使用率要增加。當然整體 jobs 的公平性也要注意，不能因為顧及整體的效能，嚴重犧牲某些 jobs 的權益。

為了驗證我們的實驗，除了實際建構 cluster of SMPs 之外，提供一個好的模擬環境也是好的選擇，因此，本實驗使用 C 語言撰寫一個叢集系統模擬器，在工作數據(trace)方面採用具有公信力的實驗數據(Parallel Workload Archive)，模擬該工作數據在叢集系統上運作的情形，並比較不同方法的效能。本實驗證實：當系統要分配 CPUs 給 job 時，可以依照當時網路的負載程度的輕重，動態調整這些 CPUs 的集中程度，原則是網路的負載程度重時則要求集中，網路的負載程度輕時則放寬集中程度。

除了簡介之外，本論文分為以下幾個部分：2. 相關研究，簡單介紹有關先前排程方法的研究；3. 研究方法及介紹實驗環境、實驗結果；4. 結論，總結本研究的成果。

貳、相關研究

早期 resources 分配的研究都著重在 CPU 的分配，在 multicomputer 環境下，parallel job 的 CPU scheduling 通常採用 space sharing approach，研究 parallel job scheduling 的專家 Feitelson 指出，這一類研究的基本假設如下(Feitelson, 2005; Mu' alem & Feitelson, 2001; Tsafrir, Etsion & Feitelson, 2007)：

(1) 一個新的 job 到達系統時，系統會依照這個 job 要求的 CPUs 個數，根據本系統各 queues 包含的 jobs 的分佈及當時 CPUs 的分配情況，執行某些策略決定何時分配 CPUs 給此 job。

(2) 有些 job scheduling algorithms，例如 backfilling scheduling，規定使用者必須提供此 job 執行時間的預估值。使用者若提供較小的預估值，有機會讓此 job 提前執行，但往後真正的執行時間若大於此預估值，此 job 可能會被 killed。

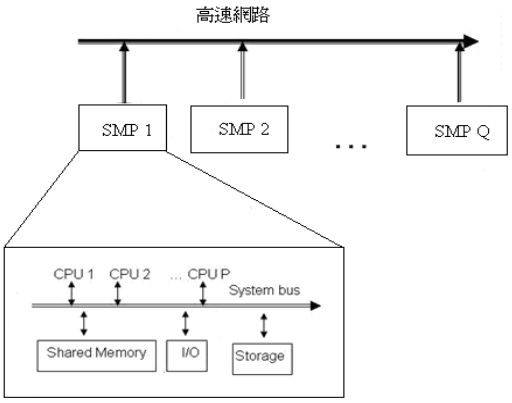

圖 1 本論文假設的 cluster of SMPs 架構。

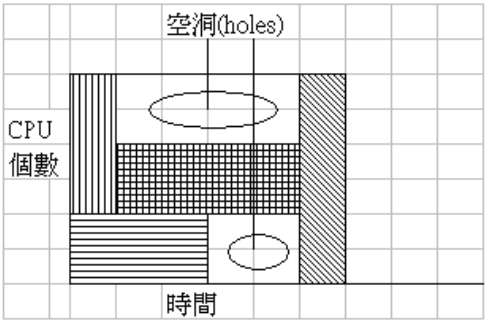

圖 2 Parallel Job scheduling 執行時 Jobs 分配圖。

上述的情形可用一個二維的圖來表示，如圖二所示，原點表示現在的時間，而水平與垂直軸分別代表時間與 CPUs 個數，每個 job 都可以視為一個被填滿的矩形，矩形的長度是此 job 執行時所需的(或預估的)時間，高度則是此 job 執行時所需的 CPUs 個數。當此 2D 圖裡的矩形之間出現空洞(holes)，則表示有許多 CPUs 正在閒置。具有 backfilling 機制的系統會將 waiting queue 後面較小的 jobs 往前移動，設法將此空洞補滿，以增進系統整體的效能 (Mu'alem & Feitelson, 2001; Tsafrir, Etsion & Feitelson, 2007)。

在 Mu'alem and Feitelson(2001)及 Tsafrir, Etsion and Feitelson(2007)，針對 parallel job scheduling algorithms，有很詳盡的介紹和分析：最簡單的 parallel job scheduling algorithm 叫 FCFS(first-come first-serve)，CPUs 的分配完全以 jobs 到達系統的先後次序決定，當目前系統剩下的閒置 CPUs 個數無法滿足目前 waiting queue 最前頭的 job 的需求，排在 queue 後面的 jobs 必須等待，即使後面有其他 CPUs 需求較小的 jobs，也無法提早使用這些閒置的 CPUs。FCFS 的優點是簡單，容易實作，也不用使用者提供 jobs 的預估執行時間，但是卻會面臨很多的 holes 的碎裂(fragmentation)情形，如果有一些需要大量 CPUs 的 jobs 擋在前面，會嚴重影響系統的效能。

為了解決 FCFS 產生的碎裂問題，backfilling methods 是一個好的改善方案，backfilling methods 建議前頭的 job 所要求的 CPUs 個數無法被系統提供時，可以允許其後面有限個數 CPUs 要求較小的 jobs 往前移，以獲得資源先行執行，這一類方法可以改變 jobs 執行的次序，但是使用者必須提供 job 執行時間的預估值，給系統做判斷。backfilling 的策略中，有兩種最常見：分別是積極的(aggressive) backfilling 和保守的(conservative) backfilling (Mu'alem & Feitelson, 2001)。

EASY (Extensible Argonne Scheduling sYstem)是 aggressive backfilling 中的代表，它由 IBM 在 SP1 平行電腦的環境下所發展出來，因為效能不錯，後來也應用在 SP2 的電腦上(Mu'alem & Feitelson, 2001)。為了增加 job backfilling 的機會，只要不會延遲到位於 queue 最前頭的 job 的執行，即可將 CPUs 要求較少的 jobs 往前搬移優先執行，詳細的演算法在 Mu'alem and Feitelson (2001) 有描述。

另一種方法叫 conservative backfilling，這種方法與 aggressive backfilling 類似，queue 後面較小的 job 可以往前移。當每一個 job 進入系統時，系統都會給予一個保留，保障其最晚開始執行的時間。但是這種方法較保守，在後方往前移動的 job 不能延遲所有在 queue 前面中任何一個 job 的執行，其詳細的演算法在 Mu'alem and Feitelson (2001)有詳述。

另外一個議題是，assign 給 job 的 CPUs 的集中程度如何。有一種希望 assign 到某個 job 的 CPUs，儘量集中在一起（或稱為連續），這一類方法主要在 mesh connected 及 Torus 的 multicomputers 上提出。我們知道，在 mesh 及 torus 上，system topology 扮演非常重要的角色，因此，若分配出去的 CPUs 較為分散，communication cost 會比較大。針對這些架構早先方法建議寧可多花一點 waiting time，讓得到 CPUs 必須集中在一起，以降低 communication overhead。但是會有 waiting time 過長，以及 external fragmentation 的問題。為了平衡 waiting time 及 communication overhead，許多研究開始著重所謂 non-continuous

allocation strategies，例如 Hosseini-Moghaddam and Naghibzadeh (2006)、 Seo (2005)及 Zhu (1992)建議在 mesh 上，而 Mao, Chen and Watson III (2005)則在 multi-dimensional Torus 上，他們共同的特色是建立 non-continuous 但 compact 的 processor allocation strategy，除了減少 waiting time，又可以減低 communication cost。

前面討論的方法，比較注重如何 allocate CPUs 給需要的 job，但忽略下面幾個問題：

(1) 這些 jobs 執行時對 memory resource 的要求較少提及，如果 system 剩下來的 memory 不夠，執行這些 jobs 時會產生大量的 page faults，結果會浪費許多的時間。解決方案在 Wang, Chien, Chang and Tien (2008) 有詳細的討論。

(2) 這些 jobs 執行時對 network bandwidth 的要求，如果分配給這些 job 的 CPUs 的位置，是集中在少數 SMPs，network communication 的機會較小，因此對 network 的 load 也較小，但會增加 waiting time；反之，如果分散到許多 SMPs 上，waiting time 會降低，但 communication 較多，對 network 的 contention 也較嚴重。

參、研究方法及實驗結果

為了比較各種分配法在 cluster of SMPs 的效能，我們建立一套 cluster of SMPs 的模擬系統，以模擬各種策略的效能。另外，我們採用由 Dr. D. Feitelson 所維護的 Parallel Workloads Archive 網站上的 traces(Parallel Workload Archive)。目前這個網站包含了許多 Logs of Real Parallel Workloads from Production Systems，幾乎是相關議題研究一定要採用的資訊。例如 Swedish Royal Institute of Technology (KTH) in Stockholm 的 trace file 中包含在 100-node 的 IBM SP2 上 11 個月的記錄，一共有 2 萬 8 千 RECORDS，每一 RECORD 包含某一個 job 進入 system 的時間、開始及結束的時間、執行時間、需要的 memory，CPUs 的個數等。表一表示這些 JOBS 的執行時間分配情形，圖三則是這些 jobs 的 request CPUs 個數的分配情形。

表一 **Jobs 的執行時間分配情形**

執行時間 (秒)	Jobs 個數
0-1	203
1-10	1391
10-100	7995
100-1000	5195
1000-10000	6664
10000-100000	6805
>100000	237

註：最大執行時間(秒): 226709
平均執行時間(秒): 8876

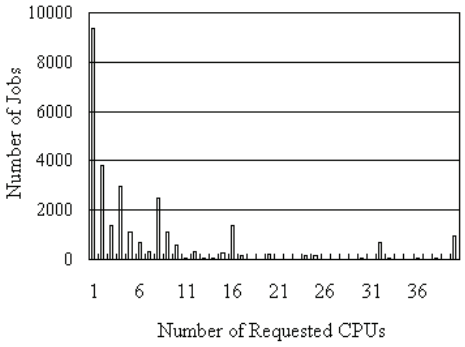

圖 3 Jobs 的 request CPUs 分配情形。

本實驗用到的假設及符號：

(1) 此 system 共有 10 個 symmetric multiprocessors (SMPs)，每個 SMP 包含 10 個 CPUs。

(2) CPU scheduling 採用 space sharing approach。

(3) Memory 是否足夠使用的標準，是設定一個 memory threshold，此 memory threshold 的 size 剛好是此 job 執行過程中是否會大量產生 pages faults 的臨界值，也就是說，此 job 得到的 memory size 總合若小於此值，page faults 就會經常產生，若大於此值 page faults 就會很少發生。至於 page fault cost function (or model)在 Wang, Chien，Chang and Tien (2008)有詳述。本論文則注重 communication cost 之探討。

(4) Communication cost 的計算方面，會因為 assign 到的 CPUs 的分散程度有所差異，這裡我們建立一個簡單的 function，依實際的分配情形來計算其 cost。

Communication cost = BASE * number_of_SMPs * Level

其中 BASE 是 communication 的基本 cost(本實驗假設 0.05)；Level(從 1 到 5)則表示快速網路的負載程度，值越大表示 communication cost 越大；number_of _SMPs 則表示 Job 執行時，system 分配給此 job 實際分布的 SMPs 個數。詳情請見表二。舉例來說，假設某個 job 需要 4 個 CPUs，如果 4 個 CPUs 在同一個 SMP 上，communication cost 等於 0.05 * 1 * Level；但是如果分散在 4 個 SMPs 上，communication cost 等於 0.05 * 4 * Level。

系統分配給欲執行的 job 所需 CPUs，system 允許這些 CPUs 分佈的 SMPs 總數不可以超過 (Tight + Minimum_number_of_SMP)；其中 Minimum_number_of _SMP 為 CPUs 分佈的 SMPs 最小值，例如某 job 需要 24 CPUs，則 Minimum_number_of_SMP 為 3，亦即這 24 個 CPUs 必須集中在某 3 個 SMPs 上;Tight 則是 job 所得之 CPUs 的集中程度，Tight 值越小表示分配到的 CPUs 落在的位置越集中，反之則較分散。例如 Tight=0 時，job 需要的 CPUs 必須集中在 Minimum_number_of _SMP 個 SMPs 中，當 Tight=10，則完全不

限定所得到的 CPUs 的位置。

表二 **本實驗假設的 communication cost function**

Job 分布的 SMPs 個數	通訊負擔比值 (BASE=0.05)		
	Level=1	Level=2	Level=4
1	0.05	0.1	0.2
2	0.10	0.2	0.4
3	0.15	0.3	0.6
4	0.20	0.4	0.8
5	0.25	0.5	1.0
6	0.30	0.6	1.2
7	0.35	0.7	1.4
8	0.40	0.8	1.6
9	0.45	0.9	1.8
10	0.50	1.0	2.0

我們在模擬器上，利用前面提及的 trace (Parallel Workload Archive)，在不同的網路通訊負擔(即調整不同的 Level 值)的假設之下，執行下列的實驗：以分配到欲執行的 job 所需 CPUs 的 SMP 分佈的集中程度當參數(Tight)，實作 FCFS 及接近 EASY 的 Backfilling 二種 job scheduling algorithms。至於實驗的測量值，包含所有 jobs 的下列平均值：Turnaround time、Waiting time、Slowdown、Communication cost。

圖四為 Backfilling 和 FCFS 在各種 Level 下，以 Tight 為參數所測出 trace 中 2 萬多個 jobs 平均 slowdown 之比較。本實驗證實，在所有情況之下，Backfilling 的效能的確比 FCFS 好很多。因此，後面的數據僅顯示以 Backfilling 為基礎的結果，FCFS 的部份將省略。

圖五是 Backfilling 演算法在各種 Level 下，以 Tight 為參數所測出 trace 中 2 萬多個 jobs 之平均 communication cost 之比較，由此圖可見：快速網路的負載程度越大(Level 的值越大)或/及分配之 CPUs 較不集中(Tight 值越大)，則 communication cost 越大。

	Level=0		Level=2		Level=4	
Tight 值	Backfilling	FCFS	Backfilling	FCFS	Backfilling	FCFS
Tight=0	150	268	219	442	346	793
Tight=2	68	186	119	363	253	963
Tight=4	64	179	121	385	430	1180
Tight=6	62	178	119	393	563	1255

圖 4 Backfilling 和 FCFS 平均 slowdown 之比較。

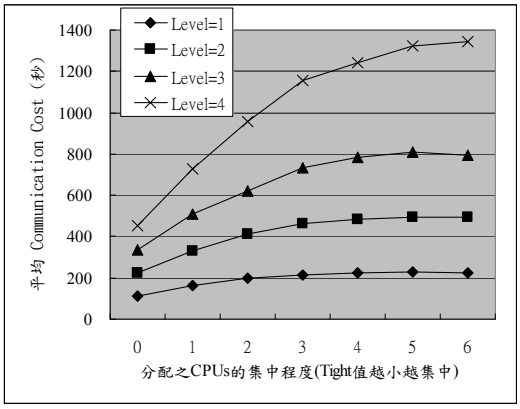

圖 5 Backfilling 演算法在各種 Level 下，以 Tight 為參數所測出之平均 communication cost。

圖六是 Backfilling 演算法在各種 Level 下，以 Tight 為參數所測出 trace 中 2 萬多個 jobs 之平均 turnaround time 之比較，由此圖可見：1. 快速網路的負載程度較小時(Level 的值越小)，turnaround time 會隨著 Tight 增加而遞減，例如 Level = 0 時，當 communication cost 很低時，分配之 CPUs 不用考慮是否集中，因此 waiting time 都不大，所以 Tight 越大 turnaround time 越小。2. 快速網路的負載程度較大時(Level 的值越大)，除了一開始之外，turnaround time 會隨著 Tight 增加而遞增，Level 越大(例如= 4 時)，因為 communication cost 很大，分配之 CPUs 若不集中，communication cost 則會快速增加，因此 turnaround 也會隨著快速增大。

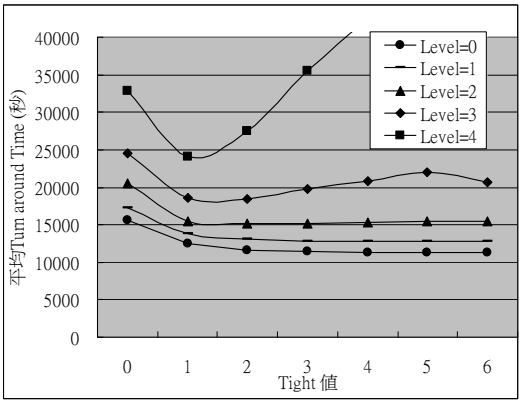

圖 6 Backfilling 演算法在各種 Level 下，以 Tight 為參數所測出之平均 turnaround time。

圖七是 Backfilling 演算法在各種 Level 下，以 Tight 為參數所測出 trace 中 2 萬多個 jobs 之平均 waiting time 之比較。一般而言，若不考量分配到的 CPUs

是否集中(即 Tight 值較大時)，因為不用等待 CPUs 的落點，因此 waiting time 會隨著 Tight 增加而減少，例如圖中的 Level =0 和 Level= 1。然而當 Level 值(communication cost)變大時，因為 system 來不及消化接踵而至的 jobs，因此新進來的 jobs 越積越多，等待處理，因此 waiting time 亦隨之增加。

圖八是 Backfilling 演算法在各種 Level 下，以 Tight 為參數所測出 trace 中 2 萬多個 jobs 之平均 slowdown 之比較，此圖和圖六的 turnaround 情形類似：1. 快速網路的負載程度較小時(Level 的值越小)，turnaround time 會隨著 Tight 增加而遞減，例如 Level = 0 時，因為 communication cost 很低時，分配之 CPUs 不用考慮是否集中，因此 waiting time 也都不大，因此 Tight 越大 slowdown 越小。2. 快速網路的負載程度較大時(Level 的值越大)，除了一開始之外，slowdown 會隨著 Tight 增加而遞增，Level 越大(例如= 4 時)，因為 communication cost 很大，分配之 CPUs 若不集中，communication cost 則會快速增加，因此 slowdown 也會隨著快速增大。

從上述的實驗可以發現，當我們設計 processor allocation 演算法時，若將 Tight 值固定，絕非好的方法，如表三所示，從 Tight=0 到 Tight=6，沒有一個固定 Tight 值在不同 Level 值下均有好的表現。當網路的負載程度較小時，為了減少 waiting time，我們可以不限定 assigned CPUs 的落點，也就是允許較大的 Tight 值，例如表三的 Level=0 時，slowdown 最佳值在 Tight=6 時，然而隨著網路的負載程度的增加，我們必須開始對 Tight 值設限，例如 Level=1 時，slowdown 最佳值在 Tight=4 時，Level=4 時，slowdown 最佳值在 Tight=1 時。

我們可以將 processor allocation 演算法做如下調整：當 system 要 assign CPUs 給 job 時，可以依照當時的網路的負載程度的輕重，動態調整這些 CPUs 的集中程度(Tight 值)，原則是網路的負載程度重時則要求集中，網路的負載程度輕時則放寬集中程度。然後去尋找是否有足夠的 CPUs 個數且符合 Tight 值，若有則將這些 CPUs assign 出去，否則須等待其他 job(s)釋出 CPUs 後再行判斷是否可以進行後續的分配工作。

例如我們可以利用一非常簡單的方法，稱為可調法：令 Level 值加上 Tight 的值為常數，以表三為例，可令 Level + Tight = 6，可調法的值都很接近最佳值，比所有固定 Tight 法好。

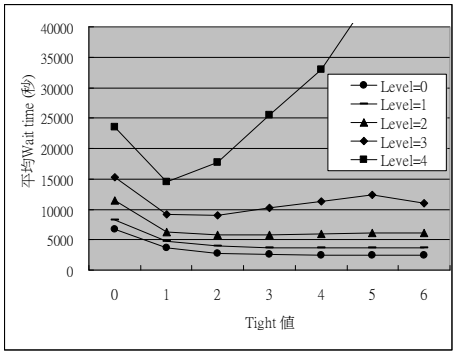

圖 7 Backfilling 演算法在各種 Level 下，以 Tight 為參數所測出之平均 waiting time。

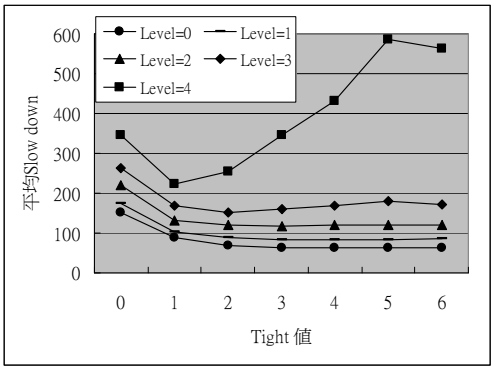

圖 8 Backfilling 演算法在各種 Level 下，以 Tight 為參數所測出之平均 slowdown。

表三 **可調法與固定 Tight 值法 slowdown 之比較**

	Level=0	Level=1	Level=2	Level=3	Level=4	Level=5
Tight=0	150.0	174.2	218.8	263.1	346.0	466.1
Tight=1	87.3	103.5	131.7	168.4	222.9	354.1
Tight=2	67.7	88.2	119.1	150.8	253.4	486.9
Tight=3	64.0	83.8	117.6	159.4	346.1	701.8
Tight=4	63.8	82.5	121.3	167.4	430.3	1221.1
Tight=5	61.7	83.4	119.3	179.4	584.8	2263.5
Tight=6	61.6	84.6	119.5	171.1	562.6	2692.8
可調法	**61.6**	**83.4**	**121.3**	**159.4**	**253.4**	**354.1**
最佳值	**61.6**	**82.5**	**117.6**	**150.8**	**222.9**	**354.1**

肆、結論

本論文主要研究在 cluster of SMPs 上，parallel jobs 的資源分配之問題。在此架構下，除了 CPUs allocation 的問題之外，還有在 SMP 內部 shared memory 的競爭和整個 system 對於 network 的競爭。其中前者在 Wang, Chien, Chang and Tien (2008)有詳細的討論，本論文則著重在後者的探討。針對此一問題，有二個不同的看法：1. 在 parallel computers 上的 resource allocation policies，大多較注重 CPUs (or processors) allocation，主要目的是增加 CPUs 的 utilization(盡量讓 CPUs 分配出去，減少閒置)，以降低 waiting time 進而改善 slowdown 及 response time (Feitelson, 2005; Mu'alem & Feitelson, 2001; Tsafrir, Etsion & Feitelson, 2007;.Talby & Feitelson, 2005)。然而上述方法，在所謂 communication intensive jobs，效能會因為分散 CPUs 的 communication cost 較大而變差。2. 另一方面，有些研究則建議 assign 到 parallel jobs 的 CPUs，儘量集中在一起（或稱為連續），主要觀念是，寧願多花一點時間等待，讓得到 CPUs 儘量集中在一起，以降低 communication overhead。但這種方法較嚴格限制分配到的 CPUs 必須集中，就算整個系統閒置的 CPUs 夠多，但不夠集中，卻不會分配。因此

雖然降低 communication cost，但會增加 waiting time(Hosseini-Moghaddam & Naghibzadeh, 2006; Seo, 2005;. Zhu, 1992; Mao, Chen & Watson III，2005)。

最簡單的 parallel job scheduling algorithm 叫 FCFS(first-come first-serve)，另一種效能較佳的稱為 EASY (Extensible Argonne Scheduling sYstem) backfilling 演算法(Mu'alem & Feitelson，2001)。這些 jobs 執行時對 network bandwidth 的要求，如果分配給這些 job 的 CPUs 的位置，是集中在少數 SMPs，communication 的機會較小，因此對 network 的 load 也較小，但會增加 waiting time；反之，如果分散到許多 SMPs 上，waiting time 會降低，但 communication 較多，對 network 的競爭也較嚴重。

為了比較各種分配法在 cluster of SMPs 的效能，我們建立一套 cluster of SMPs 的模擬系統，以模擬各種策略的效能。另外，我們採用由 Dr. D. Feitelson 所維護的 Parallel Workloads Archive 網站上的 traces。目前這個網站包含了許多 Logs of Real Parallel Workloads from Production Systems，是相關議題研究一定要採用的資訊。

我們在模擬器上，利用前面提及的 trace，在不同的網路通訊負擔(即調整不同的 Level 值)的假設之下，執行各項實驗。首先本實驗證實，在所有情況之下，Backfilling 的效能的確比 FCFS 好很多。因此，後面的數據僅顯示以 Backfilling 為基礎的結果，FCFS 的部份將省略。我們的結論是：當 system 要 assign CPUs 給 job 時，可以依照當時的網路的負載程度的輕重，動態調整這些 CPUs 的集中程度(Tight 值)，原則是網路的負載程度重時則要求集中，網路的負載程度輕時則放寬集中程度。然後去尋找是否有足夠的 CPUs 個數且符合 Tight 值，若有則將這些 CPUs assign 出去，否則等待其他 job(s)釋出 CPUs 後再行判斷是否可以進行後續的分配工作。

致謝

本論文之研究經費，部分由國科會計劃編號 NSC97-2221-E-126-004 及 NSC 98-2221-E-126-004 提供，在此致謝。也感謝前後期協助計劃的同學，包含研究生以及大學部專題學生。

參考文獻

Feitelson, D. G. (2005). Experimental Analysis of the Root Causes of Performance Evaluation Results: A Backfilling Case Study. *IEEE Transactions on Parallel and Distributed Systems*, *16*(2), 175-182.

Hosseini-Moghaddam, S.-M. & Naghibzadeh, M. (2006). A New Processor Allocation Strategy Using ESS (Expanding Square Strategy). *14th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, 137-140. Montbeliard-Sochaux, France.

Mao, M., Chen, J. & Watson III, W. (2005). Efficient Subtorus Processor

Allocation in a Multi-Dimensional Torus. *Proceedings of the 8th International Conference on High-Performance Computing in Asia-Pacific Region*. Beijing, China.

Mu'alem, A. W. & Feitelson, D. G.. (2001). Utilization, Predictability, Workloads, and User Runtime Estimates in Scheduling the IBM SP2 with Backfilling. *IEEE Transaction on Parallel and Distributed Systems*, *12*(6), 529-543.

Parallel Workload Archive, http://www.cs.huji.ac.il/labs/parallel/workload.

Seo, K. H. (2005). Fragmentation-Efficient Node Allocation Algorithm in 2-D Mesh-Connected Systems. *Proceedings of the 8th International Symposium on Parallel Architectures, Algorithms and Networks*, 318-323. Las Vegas, USA.

Tsafrir, D., Etsion, Y. & Feitelson, D. G. (2007). Backfilling Using Runtime Predictions Rather Than User estimates. *IEEE Transactions on Parallel and Distributed Systems*, *18*(6),789-803.

Talby, D. & Feitelson, D. G. (2005). Improving and Stabilizing Parallel Computer Performance Using Adaptive Backfilling. *Proceedings of the 19th International Parallel and Distributed Processing Symposium*. Denver, USA.

Wang, Y. M., Chien, K. T., Chang, Y. H. & Tien, S. H. (2008). Memory Consideration for Parallel Job Scheduling on SMP Clusters. Journal of *Internet Technology*, *9*(5), 417-423.

Zhu, Y. (1992). Efficient Processor Allocation Strategies for Mesh-connected Parallel Computers. *Journal of Parallel and Distributed Computing*, *16*(4), 328-337.

王逸民(Yi-Min Wang)，1984 年交通大學資訊科學系畢業，1986 年清華大學計算機管理決策研究所碩士，1996 年交通大學資訊科學博士。1986 年到 1989 年擔任中山科學研究院技士，1989 年至 1992 年擔任資訊工業策進會工程師。1996 年起任教於靜宜大學，目前是靜宜大學資訊管理系教授。主要研究領域為叢集計算、作業系統和計算機結構等。