SMP 叢集上資源預估對 Parallel Job 排程演算法效能之影響

許俊傑
靜宜大學資訊管理系大學部學生
e-mail: s9571090@pu.edu.tw

王逸民
靜宜大學資訊管理系教授
e-mail：ymwang@pu.edu.tw

摘要

本論文主要討論在 SMP(symmetric multiprocessors) 叢集上， jobs 所須執行時間和記憶體的使用者預估值，對 parallel jobs 演算法效能影響之研究。許多資源分配演算法執行前，必須先得到 jobs 執行時所須各項資源的預估值，然後根據這些值安排 jobs 執行的順序。先前的研究只對預估執行時間的準確性，有各種不同的研究及看法。本論文建議，除了執行時間預估值的影響研究之外，也希望探討 job 所需要的記憶體的使用者預估值，對系統效能影響之研究。實驗結果顯示：不論執行時間預估值或所需記憶體預估值，若能比真正所需要的做適當的增加，反而會有比較好的表現，然而預估值若太大，則無法得到改善。

關鍵詞：SMP 叢集；資源分配；資源預估；工作排程。

Abstract

The paper studies the impact of execution time prediction and memory prediction on resources allocation for SMP cluster. Previous backfilling scheduling algorithms typically require run time prediction from the user. In this paper, we suggest that the information provided by users should also include memory prediction. Experimental result shows that as larger predictions are given, better performance will be achieved.

Keywords: SMP Clusters; Resource Allocation; Resource Prediction; Job Scheduling

1. 前言

叢集系統是一項非常重要的高性能計算環境研究議題。本論文所假設的 SMP (symmetric multiprocessors)叢集的架構圖，如圖一所示：假設系統有 *Q* 個 nodes，每個 node 是一個包含 *P* 個 CPUs 的 SMP。在此架構下的資源分配議題，[21]建議除了 CPUs 分配(allocation)之外，亦須同時考量記憶體及通訊負擔(communication cost)。而本論文的重點，則是使用者對 jobs 的執行時間預估值，及記憶體需求預估值的正確與否，對整個系統效能影響的研究。

先前的資源分配策略，例如常見的各種回填排程演算法 (backfilling scheduling algorithms)，大多只注重 CPUs (或 processors)的分配，主要目的是增加 CPU 的使用率[1, 2, 7, 8, 10, 13, 14, 15, 18, 19]。另一方面，在某些 mesh connected 及 hypercube connected 的架構下，有些研究則建議分配到 parallel jobs 的 CPUs，儘量集中在一起（或稱為連續）[4, 5, 9, 12, 20]，因為若分配的 CPUs 太分散，通訊負擔會非常大，這些研究建議寧願多花一點等待時間，讓得到的 CPUs 集中在一起，以降低通訊負擔。上述方法，在需要較多通訊負擔的 jobs 上，因為通訊負擔是主要的瓶頸，效能會比不考慮 CPUs 位置的方法好，但是仍然有等待時間過長，以及外部碎裂(external fragmentation)的問題。

另一方面，常見的回填排程演算法，會要求使用者，除了提出欲執行的 jobs 的需求 CPU 個數之外，也要提出執行時間預估值，以作為安排這些 jobs 的參數。因為大部分使用者的預估值都不準確，有些系統還會幫助使用者調整，或自動幫使用者根據紀錄，產生這些預估值[1, 6, 10, 11, 15, 16, 17]。使用者的預估值的準確性與系統效能之間，有下列有趣的現象：

1) 如果使用者為了增加回填的機會，將預估值盡量緊縮，雖然增加回填機會，但可能會因為執行時間超過預估值而被 killed；

2) 如果預估值較大，以避免因為超過預估值而被 killed，卻可能因預估值太長，而錯失了本身可以回填的機會[15]。

由[21]得知，好的 resource allocation policy

必須同時兼顧 CPU、記憶體和通訊負擔等資源。相同的道理，預估值的研究也一樣要多方考量。先前的研究大都只著重在執行時間的預估，本論文除了執行時間之外，使用者必須提出 jobs 所需要的記憶體預估值，讓系統做出正確的資源分配策略，也是非常值得探討的議題；實驗結果顯示: 記憶體預估值的影響和執行時間預估的影響類似，預估值較大時有時候反而比較好，原因正如[15]提及，提供較大的預估值，讓小的 jobs 提前執行的機會大大增加，整體效能會增加。但是預估值太大則無法得到改善。

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及預估值影響的研究；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。

2.相關研究

根據研究 parallel job scheduling 的專家 D. Feitelson 的說法，這一類研究的基本假設如下[1, 2, 3, 14, 15, 18]：

- 當一個新的 job 到達系統時，系統會依照這個 job 要求的 CPUs 個數、當時系統中每個 queue 所包含 jobs 的個數及當時 CPUs 的分配的情況，執行某些策略，來決定何時分配 CPUs 給此 job。而這些被分配出去的 CPUs 會固定給此 job 使用，一直到此 job 完成工作，再將這些 CPUs 歸還給系統。
- 除了最簡單的 FCFS 之外，backfilling scheduling 是 parallel job scheduling algorithms 中最常見且有效的策略，在此機制下，使用者必須提供此 job 執行時間的預估值。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。

我們可用如圖二，一個 2-Dimension 的圖來表示 Parallel Job scheduling 的執行。圖中左下角的原點表示現在的時間，而水平與垂直軸分別代表時間與 CPU 個數，每個 job 都可以視為一個矩形，矩形的長度(橫軸方向)是此 job 執行時所需的時間，高度(縱軸方向)則是此 job 執行時所需的 CPUs 個數[7, 10, 13]。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 backfill 機制的系統會將 waiting queue 後面較小的 jobs 往前移動，以設法填補這些空洞。

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，使用者也不用提供 jobs 的預估值，缺點是會面臨很多的 holes 的碎裂(fragmentation)情形，如果有一些大的 jobs 擋在前面，會嚴重影響系統的效能[1, 7, 10, 13, 14]。

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 user 必須提供 job 執行時間的預估值，給系統做判斷[1, 3, 7-10, 13, 14, 15, 16, 18, 19]。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，現簡述如下：

EASY (Extensible Argonne Scheduling sYstem)是 aggressive backfilling 中的代表，它主要是應用在 IBM 的 SP1 及 SP2 平行電腦上[1, 10, 13, 14, 18]。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 要求較少或/及執行時間較短的 jobs(我們常稱為較小的 job)，往前搬移優先執行，詳細的演算法在[10]有描述。

Conservative backfilling 與 aggressive backfilling 類似，queue 後面較小的 job 也可以往前移。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，排在後方但往前移動的 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[10]有詳述。

Backfill algorithms (例如 EASY) 的確可以改善 FCFS algorithm 的效能[21]，然而使用者必須提供所有 submitted jobs 的執行時間預估值。這個預估值當作系統安排 jobs 執行順序的重要依據。有關於使用者提供的預估值與系統效能相關的研究，較重要的議題詳述如下：

- 使用者的預估值到底準確不準確：[10] 實際用 EASY 做實驗發現，使用者的預估值並不準確，然而因為估計太小而遭 killed 的 job 並不多，因為使用者預估值普遍有偏向預估值較大的情形。其原因應當是怕因為預估值太小時，萬一執行時間超過反而

遭 killed。為了探討使用者做預估值時的心理，[6]建議先將執行時間超過預估值時會被系統 killed 的限制移除，結果顯示大約有一半的使用者會提供較佳的執行時間預估值，然而對整體的平均預估值準確性仍然沒有太大的幫助。另外，[16]發現，使用者的預估值會用比較籠統的數據，實驗證實，使用者的值大概落在幾個樣本值上，例如半小時、一小時等等，這些籠統的數據當然準確度不佳。

- 預估值的準確性對整體效能的影響：在 backfilling algorithms(例如 EASY)環境之下，使用者提供執行時間的預估值是一定需要的，現在討論的是，對整體系統的效能來說，這些預估值愈準確愈好，還是不準確反而較好。這個議題見仁見智，有人覺得不準確反而有助於整體的效能提升[1, 10, 14]，甚至故意將使用者預估值都乘以某個大於 1 的數，以當成最後的預估值；但是另外一些學者卻認為，許多使用者因為害怕預估值太緊湊，如果最後執行時間超過預估值會遭 killed，因此普遍提供較大的預估值，如此一來讓小的 jobs 提前執行的機會大大增加，scheduling algorithm 其實是變成了 shortest job first (SJF) [15]。一般而言，SJF 整體效能會較佳，但是對一些需要 resources 較多的 jobs 較不公平。為了保障需要 resources 較多的 jobs 的公平性，[15]建議保留原先執行之順序，將 algorithm 改良成 shortest job backfilled first (SJBF)。
- 系統提供預估值的相關研究：我們知道，backfill algorithms (例如 EASY)的環境下，使用者必須提供所有 submitted jobs 的執行時間預估值給系統參考，然後系統可以直接利用這些值安排 jobs 執行的順序，也可以經過 algorithm 調整執行時間預估值後再決定[17]，下面則介紹一些調整預估值的相關的方法：
  - ✓故意讓這些預估值不準確，將放大的預估值當系統 prediction：正如前面提及，有人覺得不準反而有助於整體的效能提升[1, 10, 14]，[17] 試著將使用者預估值加倍(故意把預估值的準確性降低)，結果發現效能竟然比原先以原始的預估值的效能還要好，如同前面的討論，原因應該是執行時間小的 jobs 紛紛提前執行的機會大大增加，scheduling algorithm 其實是變成了 shortest job first (SJF)所致。
  - ✓利用歷史資訊: 許多大型系統的工作具有高度的重複性，也就是說有些使用者(或系統)會重複執行相同的程式，有些程式也會有高度的依賴相關性，這種情形之下可以利用歷史資訊來預估 jobs 的執行時間。[10]建議可以用 application file name, user, 或 node number 當編號，收集相同編號所有 jobs 的執行時間，然後用平均值(或加上一些值，例如 1.5 倍的 standard deviation，以避免預估值不夠而遭系統 killed)當 prediction 預估值。
- 是否還有其他 resources 在 submitted 到系統時要提供預估值? 由前面的討論可以知道，resource allocation 是 cluster 相關很重要的研究議題，Resource allocation algorithms 若能同時考量 CPU、memory、network 等的影響，整體的效能一定會較佳。既然我們希望 CPU、memory 的影響並重，所以也需要提供 memory 等需求的預估值，這個預估值準確與否，對系統效能的影響如何？則是本論文研究之重點。

3.研究方法及實驗結果

我們用 C 語言撰寫一個模擬器，假設系統有 100 個 CPUs，記憶體共有 100M，我們比較的測量值，包含所有 jobs 的下列平均值：

- 所有 Jobs 的等待時間(Waiting time)平均值：等待時間定義為 Job 開始執行的時間點和來到系統的時間點的差。等待時間越短表示效能較佳。
- 所有 Jobs 的 Turnaround time 的平均值：turnaround time 定義為 Job 完成的時間和進入系統時間的差。Turnaround time 越短表示效能越好。
- 所有 jobs 的 Slowdown 的平均值，slowdown 定義是 Turnaround time/執行時間，為了避免執行時間太小的 job 會過度放大 slowdown，我們採用和[15]類似的觀念，只要執行時間或 turnaround time 小於 10 秒，一律以 10 秒代替。

至於輸入模擬器的 trace 包含 200 個 jobs，每個 job 有如下資訊：

- 真正執行時間：從 1 到 100 任取的亂數(單位秒)。
- 真正執行所需的記憶體：從 1 到 30 任取的亂數(單位 Mbytes)。
- CPU 的需求個數：每個 job 所需 CPU 為從

1 到 100 任取的亂數(單位個)。

- 到達系統的時間點：第”從 1 到 100 任取的亂數”(單位秒)。
- 執行時間預估值：根據實驗的參數設定。
- 所需記憶體預估值：根據實驗的參數設定。

我們將實驗分為三個步驟，其結果如下：

步驟一：假設所有的記憶體預估值都正確，測量執行時間預估值的效應

執行時間預估值設定方式如下：因為預估值若比實際的執行值小時，job 會被移除，因此我們只有考量比真正執行時間大或相等的預估值。每個 job 的執行時間預估值設定為(真正的執行時間+從 1 到 T 任取的亂數)，其中 T 為時間設定的參數，T=0 表示真正執行時間和預估值相同，T 越大表示預估值比實際執行時間差異越大。

從表一可以發現，以 T=0(即真正執行時間和預估值相同)為測試的基準點，預估值較大時有時候反而比較好，例如在 T=50 最佳，原因正如[15]提及，提供較大的預估值，讓小的 jobs 提前執行的機會大大增加，整體效能會增加。但是預估值比真正值大太多(T=1500)或太少(T=25)則無法得到改善。

步驟二：假設所有的執行時間預估值都正確，測量記憶體預估值的效應。

真正執行所需記憶體之設定方式如下：因為預估值若比實際的執行值小時，job 會被移除，因此我們只有考量比真正執行所需記憶體大或相等的預估值，所以每個 job 的記憶體預估值=(真正所需的記憶體+從 1 到 M 任取的亂數)，其中 M 為記憶體設定的參數，M=0 表示真正需要的記憶體和預估值相同，M 越大表示預估值比實際所需記憶體差異越大。

從表二可以發現，以 M=0(即真正所需記憶體和預估值相同)為測試的基準點，預估值較大時有時候反而比較好，例如在 M=5 最佳，M=10 時次之。原因正如[15]提及，提供較大的預估值，讓小的 jobs 提前執行的機會大大增加，整體效能會增加。但是預估值比真正值大太多(M=20)則無法得到改善。

步驟三：測量執行時間預估值及記憶體需求預估的效應。

這部份的實驗採用前二個步驟中，第一部份的執行時間預估假設和第二部份的記憶體需求預估假設，其他和前面的假設相同。從圖三可以發現和前面類似的結論：最佳情形發生在 T=50、M=15 時。然而 T 太大時(例如 T=500 時)在各種 M 值時都普遍不佳。當 M 太大時，例如 M=30 時在各種 M 值時表現不穩定(T=50 最好，但 T=100 時最差)。

4.結論

叢集系統是一項非常重要的高性能計算環境研究議題。本論文的重點，是使用者對 submitted jobs 的執行時間預估值，及記憶體需求預估值的正確與否，對整個系統效能影響的研究。

使用者對系統要求各種資源時，大部分的系統除要求使用者提出欲執行的 jobs 的需求 CPU 個數之外，也要求使用者提出執行時間預估值，以作為安排這些 jobs 的參數。因為大部分使用者的預估值都不準確，有些系統還會幫助使用者調整，或自動幫使用者根據紀錄，產生這些預估值[1, 6, 10, 11, 15, 16, 17]。

[21]建議，好的 resource allocation policy 必須同時兼顧 CPU、memory size 和 communication cost。相同的道理，預估值的研究也一樣要多方考量。因為先前的研究大都只著重在 run time 的預估，本年度論文，建議除了 run time 之外，也希望協助或調整使用者提出欲執行的 jobs 所需要的 memory 的使用者預估值，讓系統做出正確的資源分配策略。

實驗結果顯示：適當的增加預估值有時候反而比較好，原因正如[15]提及，提供較大的預估值，讓小的 jobs 提前執行的機會大大增加，整體效能會增加。但是預估值太大則無法得到改善。

致謝：

本論文之研究經費，主要由國科會計劃 NSC 98-2221-E-126-004 及部分由國科會計劃 NSC 99-2221-E-126-007 提供，在此致謝。也感謝前後協助的同學，特別是楊雅婷及游暄富二位同學。

參考文獻

[1] D. Feitelson, L. Rudolph, and U. Schwiegelshohn, Parallel Job Scheduling—A Status Report, *Springer-Verlog*, 2004.

[2] D. Feitelson, Experimental Analysis of the Root Causes of Performance Evaluation Results: A Backfilling Case Study, *IEEE Transactions on Parallel and Distributed*

Systems, Vol. 16, No. 2, February 2005, pp.175-182.

[3] E. Frachtenberg and D. Feitelson, Pitfalls in Parallel Job Scheduling Evalution, *Job Scheduling Strategies for Parallel Processing*, June 2005.

[4] SM. Hosseini-Moghaddam and M. Naghibzadeh, A New Processor Allocation Strategy Using ESS (Expanding Square Strategy), *Proceedings of the 14th International Conference on Parallel, Distributed, and Network-Based processing*, 2006.

[5] V. J. Leung, E. M. Arkin, M. A. Bender, D. Bunde, J. Johnston, J. S. B. Mitchell, C. Phillips, and S. S. Seiden, Processor Allocation on Cplant: Achieving General Processor Locality Using One-Dimensional Allocation Strategies, *Proceedings of the IEEE International Conference on Cluster Computing, 2002*.

[6] C. B. Lee, Y. Schwartzman, J. hardy, and A. Snavely, Are User Runtime Estimates Inherently Inaccurate?, *Job scheduling Strategies for Parallel Processing*, pp. 253-263, 2005.

[7] J. Y. Lie and Y. M. Wang, Communication Cost Consideration for Job Scheduling on Clusters, *Proceedings of the First Workshop on Grid Technologies and Applications*, December 2004.

[8] V. Lo and J. Mache, Job scheduling for prime time vs. non-prime time, *Proceedings of 2002 Cluster Computing*, Sept. 2002, pp. 488 – 493.

[9] W. Mao, J. Chen, and W. Watson III, Efficient Subtorus Processor Allocation in a Multi-Dimensional Torus, *Proceedings of the eighth International Conference on High-Performance Computing in Asia-Pacific Region*, 2005.

[10] A. W. Mu'alem and D. Feitelson, Utilization, Predictability, Workloads, and User Runtime Estimates in Scheduling the IBM SP2 with Backfilling, *IEEE Transaction on Parallel and Distributed Systems*, vol.12, no.6, June 2001, pp.529-543.

[11] I. Rao and E. N. Huh, A Probabilistic and Adaptive Scheduling Algorithm using System-Generated Predictions for Inter-Grid Resource Sharing, *The Journal of Supercomputing*, Vol. 45, 2008, pp.185-204.

[12] K. H. Seo, Fragmentation-Efficient Node Allocation Algorithm in 2-D Mesh-Connected Systems, *Proceedings of the 8th International Symposium on Parallel Architectures, Algorithms and Networks*, 2005.

[13] S. Srinivasan and R. Kettimuthu et al., Characterization of Backfilling Strategies for Parallel Job Scheduling, *IEEE Parallel Processing Workshops*, 2002, pp.514 – 519.

[14] D. Talby and D. Feitelson, Improving and Stabilizing Parallel Computer Performance Using Adaptive Backfilling, In: *19th International Parallel and Distributed Processing Symposium*, April 2005

[15] D. Tsafrir, Y. Etsion, and D. Feitelson, Backfilling Using System-Generated Predictions Rather Than User Runtime Estimates, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 18, No. 6, June 2007, pp.789-803.

[16] D. Tsafrir, Y. Etsion, and D. Feitelson, Modeling User Runtime Estimates, *Job Scheduling Strategies for Parallel Processing*, pp. 1-35, June 2005.

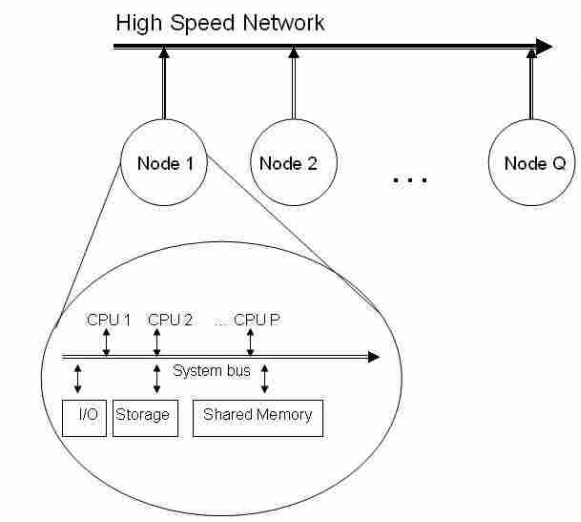
[17] D. Tsafrir, Y. Etsion, and D. Feitelson, Backfilling Using Runtime Predictions Rather Than User estimates, Technical Report 2005-5, school of Computer science and Engineering, *The Hebrew Univ*., Feb. 2005.

[18] Y. Zhang, H. Franke, J. Moreira, and A. Sivasubramaniam, An integrated approach to parallel scheduling using gang-scheduling, backfilling, and migration, *IEEE Transactions on Parallel and Distributed Systems*, Volume 14 ,Issue 3 ,2003, pp.236 – 247.

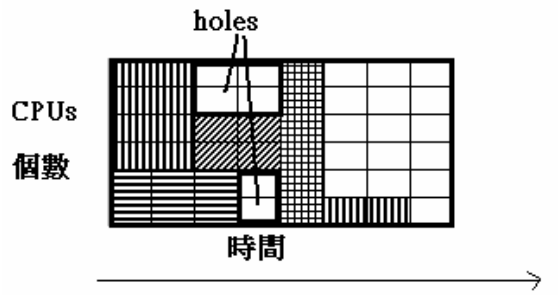
[19] Y. Zhang, H. Franke, J. E. Moreira, and A. Sivasubramaniam, A, Improving parallel job scheduling by combining gang scheduling and backfilling techniques, *Proceedings of 14th Parallel and Distributed Processing Symposium*, May 2000, pp.133-142.

[20] Z. Zhu, Efficient processor allocation strategies for mesh-connected parallel computers. *Journal of Parallel and Distributed Computing*, 16:328-337, 1992.

[21] 王逸民, SMP Clusters 上資源分配之研究, *Asian Journal of Arts and Sciences*，第 1 卷，第 1 期，頁 129 -140。



圖一 本論文假設的 SMP 叢集



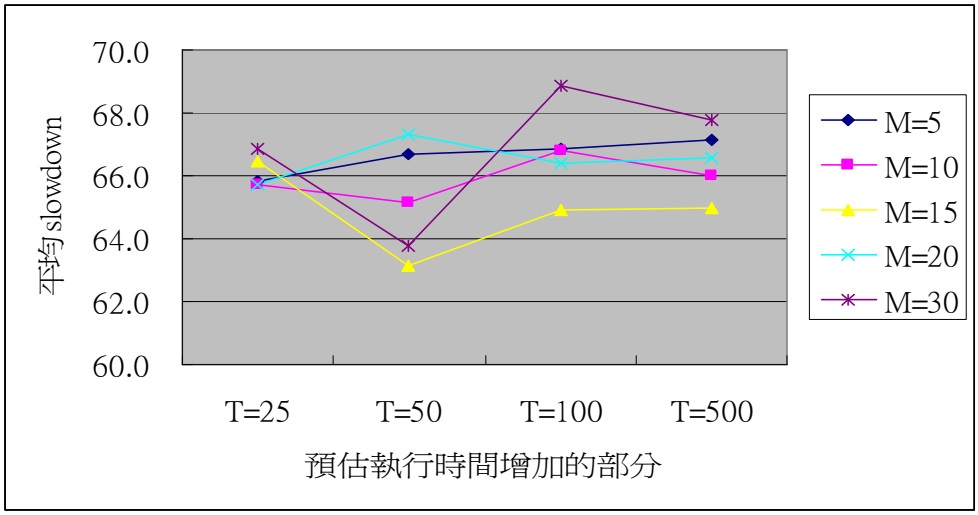
圖二 Parallel Job scheduling 執行時 Jobs 的分配圖

表一 假設所有的記憶體預估值都正確，僅考量執行時間預估值的效應

	T=0	T=25	T=50	T=100	T=500	T=1500
Average of Wait Time	1757.9	1781.3	***1712.2***	1740.1	1740.5	1782.8
Average of Turnaround Time	1801.5	1824.9	***1755.8***	1783.7	1784.1	1826.4
Average of Slowdown	67.0	68.8	***66.1***	67.1	66.5	69.2

表二 假設所有的執行時間預估值都正確，僅考量記憶體預估值的效應

	M=0	***M=5***	M=10	M=15	M=20
Average of Wait Time	1757.9	***1670.4***	1707.1	1737.6	1776.6
Average of Turnaround Time	1801.5	***1714.0***	1750.6	1781.2	1820.2
Average of Slowdown	67.0	***64.0***	64.5	66.7	69.5



圖三 同時考量執行時間預估值記憶體預估值的效應