叢集平台上考慮記憶體之平行排程演算法

簡克達 張沅合 田尚修 王逸民
靜宜大學資訊管理學系
{g9471001, s9371006, s9371048, ymwang}@pu.edu.tw

摘要

在叢集環境上，為了能有效利用系統資源，從早期的 EASY (the Extensible Argonne Scheduling sYstem, EASY)、FCFS (First-Come-First-Service, FCFS)，進展到具有分時多工性能的 Gang 排程方法，而排程方法考慮的元素也從單一考慮 CPU（中央處理器）性能到同時考慮 CPU 以及記憶體兩者，先前學者已經證實在 Gang 排程方法上同時考慮 CPU 以及記憶體能有效地減少反應時間，然而這個方法嚴格地限制系統資源必須足夠提供工作所需求的記憶體和 CPU，才能允許該工作執行，這樣的作法犧牲了少部分工作的優先執行權利，原先該工作可能只要再多一點點的資源便可以執行，卻因為排程方法的限制導致必須花費相當長的等待時間。

因此我們提出一個有限度放寬記憶體容量上限的方式，此方法為當資源不足以讓工作執行的時候，便將排程方法所考慮的系統記憶體上限略為提高，讓原本無法執行的工作能夠執行，因此該工作的等待時間便能夠縮短，由於此方法會讓所有執行中工作的總需求記憶體比原本高，因此系統會產生內文切換的行為導致所有執行中工作的執行時間將比原本不讓工作執行的情況下還要長，所以總體的執行時間拉長，我們的排程方法便是在總共的等待時間和執行時間兩者中做取捨，我們撰寫模擬器模擬一個對稱式多處理器叢集系統 (SMP Clusters)，使用具有公信力的工作數據，實驗的結果指出排程方法考慮的系統記憶體上限放寬在 10% 左右所得到的平均反應時間為最短，證實我們提出的有限度放寬記憶體容量上限能讓反應時間更進一步的減少。

關鍵詞: 叢集, 工作排程, 記憶體考慮

1. 緒論

在平行運算的叢集系統上, 最主要的研究問題就是要如何有效地利用系統資源，以及如何調節工作之間互相競爭資源的情況，並且在一般情況下不能偏袒任何一位使用者。叢集系統上的資源調配皆由一個主要的排程方法負責運算，這個排程方法必須決定工作在何時要開始執行以及要如何分配資源給不同的工作使用，避免不同工作之間出現競爭資源的情形。

先前學者所作出的研究[3]指出，在不考慮記憶體的因素下，作業系統會頻繁地對程式執行內文切換，這樣的情況會對平行程式內部的相互溝通以及同步造成損傷，因而延長工作的執行時間，而多考慮了記憶體容量之後，雖然工作的等待時間會延長，但是總體的平均反應時間下降。

本研究將工作的等待時間以及執行時間兩者間的取捨再作進一步的研究，我們的研究針對於排程方法的改變, 我們的排程方法不僅考慮了 CPU 需求同時也考慮記憶體需求，而當工作等待排程而缺乏記憶體時，我們的排程方法在對系統的記憶體容量上限這部分所做的考慮，會有限度地提升考慮的記憶體容量上限，更新之後，在排程方法所允許的記憶體容量範圍之內考慮工作是否能夠執行，可以執行的情況下便執行該工作，因此增加所有執行中工作的執行時間，不可執行的情況下該工作便繼續留在等待佇列，我們將分別在 FCFS 以及 EASY 中加入此概念。

本實驗使用 C 語言撰寫一個對稱式多處理器叢集系統模擬器，在工作數據方面採用兩個具有公信力的實驗數據[6][7], 模擬該工作數據在對稱式多處理器叢集系統上運作的情形，由於我們採用的兩個工作數據之間差異性大，能夠驗證我們的排程方法不是僅適用於特定情形的特製排程方法，另外搭配不同的排程方法比較我們改良的排程方法所增進的效能多寡，實驗結果指出所有執行中的工作總合需求記憶體在超出系統記憶體容量上限 10%左右，跟原本的排程方法 FCFS 以及 EASY 比較，能夠更有效地降低總體的平均反應時間。

本論文分為以下幾個部分：2.相關研究，簡單介紹有關排程方法的研究歷史以及記憶體考慮相關的論文介紹；3.有限度開放記憶體容量上限的新排程方法，針對我們所提出來的改良排程方法作解說；4.實驗環境，介紹我們實驗運行的環境；5.實驗結果；6.結論，總結本研究的貢獻及未來發展方向。

2. 相關研究

為了能夠有效地利用叢集系統上的資源, 好的排程方法是不可缺少的。最簡單的方式是 FCFS，此方法依照工作進入系統的時間依序執行，沒有考慮工作需求資源的差異以及需求龐大的工作造成的路障情形，此情形為當有龐大資源需求的工作等待執行，它必須等待系統有足夠資源才能執行，在這段等待期間使得系統有相當多的資源未被使

用，這樣的方式造成整個叢集系統的效能低落。而後一個新的排程概念---回填法(backfilling)被提出來以增進系統使用率，回填的目的在於充分利用系統空閒的資源，著名的有 conservative backfill (FCFS with backfilling)[1][2][4] 和 aggressive backfill (EASY)[1][2][5]。

在 conservative backfill 中，每個在等待佇列中的工作都擁有一個保留權，這個保留權依照該工作進入系統的時間排順序，保留權的功用為確保工作執行的順序與使用資源的權利，每個工作進入系統的時候就會考慮該工作能不能回填，回填的規定有兩個 (1) 現在系統中有足夠資源可供該工作使用 (2) 該工作不能影響到先前進入系統而在等待佇列的工作的執行優先權益，符合以上兩個規定，該工作便能回填。

在 aggressive backfill 中，只有在等待佇列中的第一個工作擁有保留權，而除了第一個工作之外，其他工作都列入可以回填的考慮之中，回填的規定有兩個(1)現在系統中有足夠資源可供該工作使用(2)該工作不能影響到等待佇列中的第一個工作，符合以上兩個規定，該工作便能回填。EASY 能夠將現有的系統資源盡量做最大額度的利用，但是需求龐大的工作在 EASY 排程之下可能會永遠等不到有足夠系統資源的情況，此為 EASY 排程方法的缺點。

上述的排程方法所考慮到的系統資源都限定在只考慮處理器個數是否足夠讓工作使用，由於沒有考慮到記憶體的因素，因此排程方法分配給系統的工作，其所需記憶體可能超出系統負荷，如此系統在執行工作的時候，會頻繁地執行內文切換，進而造成執行時間的上升。在[3]中提出了將 Gang 加進了記憶體容量的考慮因素，當系統內部的記憶體和處理器個數不足以讓一個工作執行的時候，此工作便會繼續放置在等待佇列直到記憶體和處理器個數足夠使用，結果證實加進了記憶體考慮之後，能大幅度地降低平均反應時間，這個方法比較了記憶體容量和沒有考慮記憶體容量兩種不同方式下的差異結果，本研究將這個結果再往前進一步地推演，考慮在不同的記憶體容量之下所造成的影響。

前面這些排程方法促使了我們將記憶體考慮納入 space sharing 方式的排程方法之中，然而同時考慮兩者雖然保證工作的執行時間不會變長，卻增加了工作的等待時間，如果能讓工作提前執行，在系統超出一個限度的記憶體上限之內，那麼工作所減少的等待時間，或許能抵平因為內文切換所增加的工作執行時間，因此使得反應時間能再進一步地降低。本研究基於這個想法，對於 space sharing 的排程方法，加進了記憶體考慮的因素，同時也加進了記憶體上限的考慮方式。

3．有限度開放記憶體容量上限的排程方法

本研究從基本的 space sharing 排程方法為出發點，從原本只考慮處理器個數的排程考慮方式，進展到能同時考慮處理器個數和記憶體容量的排程方法，為了能夠觀察系統超出記憶體上限不同所造成的影響，對於記憶體容量考量的部分作了進一步的修改，公式(1)和(2)為判斷工作能否從等待佇列轉移到執行狀態的判斷式，公式(1)為考慮處理器個數是否足夠讓工作使用，當工作需求的處理器個數小於等於現有系統能提供的處理器個數，公式(1)便能夠成立。公式(2)為考慮記憶體容量是否足夠讓工作使用，並且加入超出記憶體上限的變數 C，C 為排程方法預設的超出記憶體上限百分比，工作需求的記憶體容量小於等於系統能提供的記憶體容量，公式(2)便成立，當公式(1)和(2)皆同時成立，工作才允許執行。

Job require processors ≦ physical processors – used processors (1)

Job require memory ≦ physical memory*(1+C) – used memory, C≧0 (2)

當系統使用的記憶體超出了可用容量上限，也就是我們將公式(2)中的 C 值設定為大於 0 的情況下，系統將會對執行中的工作做內文切換的動作，將可用資料搬移到記憶體內部處理，這個動作會用到頻繁的磁碟讀取和寫入的動作。針對超出記憶體的比例多少而會增加的內文切換發生次數，現在的研究還沒有能準確預測的模型出現，我們根據[8]中對於記憶體容量跟內文切換關聯性的公式，改良成輸入系統原有記憶體容量和現在使用的記憶體容量，便能對系統所增加的內文切換發生次數，乘上一次內文切換所要耗費的執行時間，統合之後所增加的執行時間為結果，算出一個大概的比例。

3.1 加入等待時間考慮因素

由於考慮記憶體容量上限的排程方法結果並不穩定，因此針對擁擠情況下等待中工作提前執行的判定，額外增加等待時間上的判斷，每個工作會因應他所需求的執行時間給予一個相對應的等待時間門檻值(公式(3))，在未超出門檻值的時候，等待中工作依照一般的排程方法排程，也就是只考慮 CPU 和記憶體是否足夠使用，當超出門檻值並且系統在擁擠情況時，等待中工作就能依照我們所提出來的考慮記憶體上限的新排程方法，來決定該工作是否能提前執行。

Wait Time Threshold = Job’s run time * Threshold (3)

圖 1 以及圖 2 為加進記憶體考量因素和等待時間考慮因素之後的新排程方法 FCFS 和 EASY。在新 FCFS 及 EASY 排程方法裡，除了考慮資源方面加進了對記憶體上限的考量，其他排程步驟依然與舊有的排程方法相同。當執行中工作的執行時間增

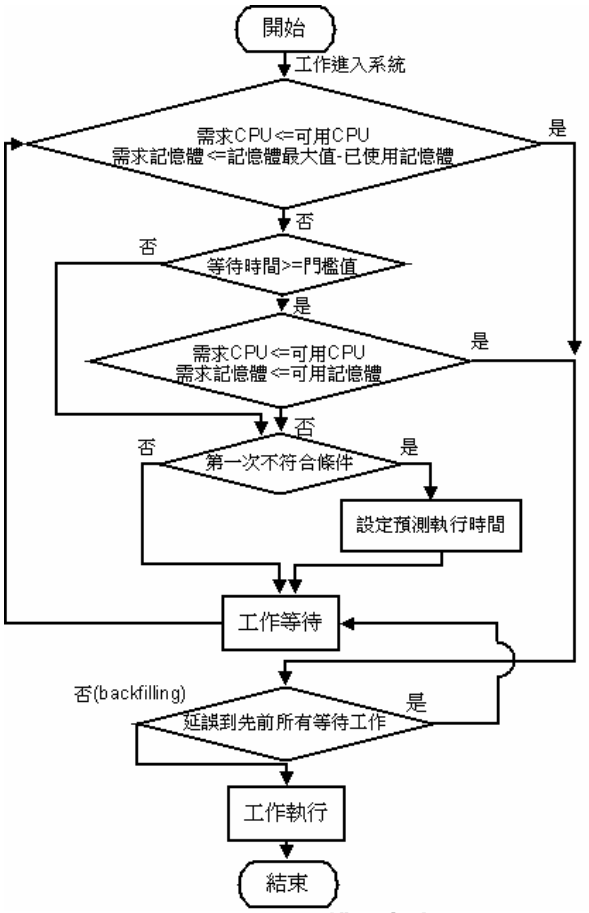


圖 1 新 FCFS 排程方法
可用記憶體=記憶體最大值*(1+C)-已使用記憶體

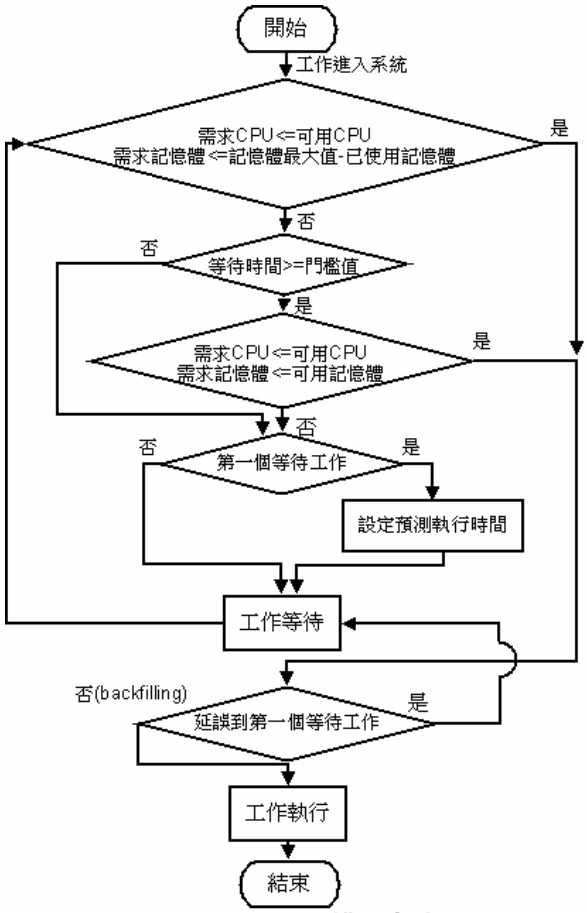


圖 2 新 EASY 排程方法
可用記憶體=記憶體最大值*(1+C)-已使用記憶體

加，則本來預估可以執行的等待工作有可能要再等上一段時間，這多出的等待時間便是因為讓工作提前執行所做的犧牲，我們所期望的就是提前執行所減少的等待時間，能夠勝過因提前執行所增加的執行時間以及後續等待中工作的延誤等待時間。

3.2 記憶體懲罰模型

公式(4)為我們參考[8]所提出來的內文切換增加比例的簡化模型，M*為現在所使用的記憶體容量大小，M 為系統原本記憶體容量大小，N 為我們因為記憶體不足所必須增加執行時間的增加比例，圖 3 為超出記憶體上限程度以及增加比例的對照圖，表 1 為增加比例範例。現在假設有一個執行時間為 10 秒的工作進入系統，而現在的排程方法允許超出記憶體容量上限 60%，當該工作允許執行時剛好超出系統上限 50%，依照我們的增加比例模型，當系統超出記憶體上限為總記憶體容量的 50%的時候增加比例為 1，於是將該工作以及所有執行中工作的剩餘執行時間乘上 2 倍，於是該工作需要執行 20 秒，其他執行中工作的剩餘執行時間也乘上 2 倍。

$$N=\frac{1}{2}\left(H+\sqrt{H^2-4}\right)-1,\ H=1+\frac{M^*}{M}\quad M^*\geq M \qquad (4)$$

計算出增加比例之後，針對現有系統中執行工作的剩餘執行時間，乘上我們所計算出的增加比例，公式(5)結果便是我們因為超出記憶體容量之後的剩餘執行時間，爾後當一些執行中的工作結束之後，此時記憶體壓力減輕，我們要將工作的執行時間回復到原本它應該的剩餘執行時間，我們套用公式(6)，以跟公式(5)相同的比例方式，對所有還在執行中的程式計算原本的剩餘執行時間，爾後當記憶

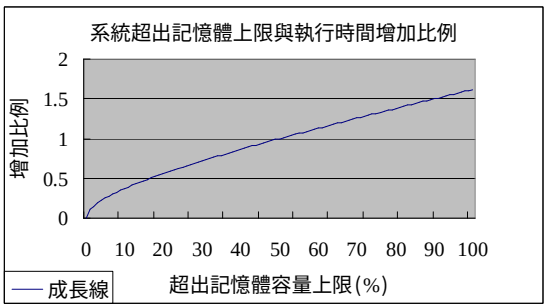


圖 3 系統超出記憶體上限與執行時間增加比例

表 1 超出記憶體上限與時間增加比例範例表

M*/M	H	N
1.1	2.1	0.37
1.3	2.3	0.72
1.5	2.5	1

體再度超出原本系統記憶體容量上限，則會再呼叫公式(5)更改所有執行中程式的剩餘執行時間。

For （all running jobs）
Job's exec_time = Job's exec_time*(N+1) (5)
For （all running jobs）
Job's exec_time = Job's exec_time*1/(N+1) (6)
exec_time 為 execution time

4. 實驗環境

我們運用模擬器搭配上具有可信度的工作數據(Workload trace) DAS2-fs2[6]，來進行本研究的實驗。本研究模擬的平台為一個 SMP Clusters，圖 4 為模擬硬體架構的表示圖，由兩個具有 64 個 CPUs 以及 1G 記憶體的 SMP 系統組成一個 Clusters，表 2 是我們所模擬出來平台的硬體數據，此硬體數據跟本研究所採用的工作數據[7]所運行的實驗平台有關，在處理器個數方面本研究採用跟工作數據相同的處理器個數，處理器個數於 64 時依然能夠執行，然而多數情況下系統資源會限制在處理器個數不足的情況；由於工作數據是以記憶體 1G 為設定環境，因此記憶體容量從 1G 到 2G 之間作測試，以求得在不同程度資源緊縮的情況下排程方法的差異，當記憶體在 2G 以上時由於實驗結果與 2G 差異值甚小，故我們設定記憶體 2G 為記憶體能夠充分使用的情況。

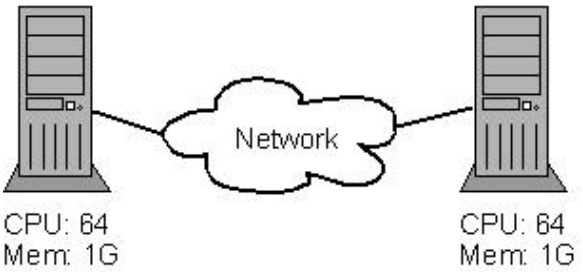


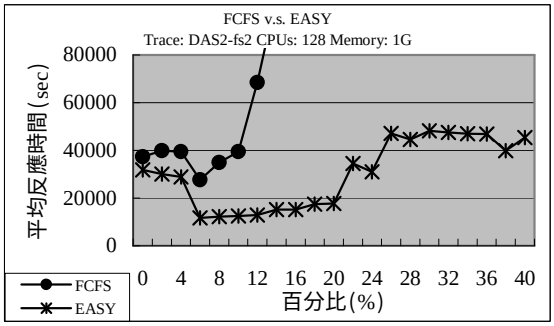
圖 4 128 個 CPUs、2G 總記憶體的 SMP Clusters

表 2 實驗平台硬體設定

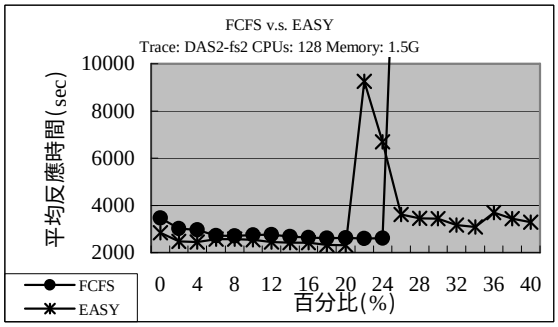
Processors	128
Physical Memory	1-2 GBytes
Seek time	10ms
Transfer rate	40MB/s

4.1 評估標準

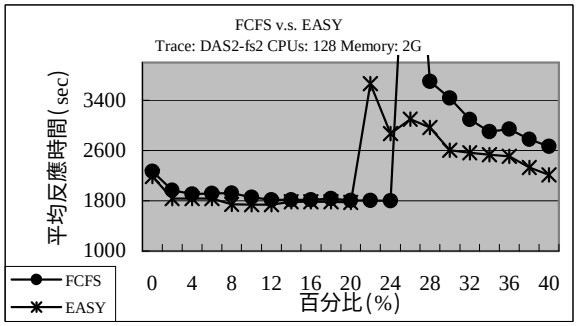
本研究採用[3]所提出來的平均反應時間作為我們評估排程方法的依據，反應時間公式(7)為工作花費在系統內部的時間，從工作到達系統開始計算，到工作完成運算離開系統為止，反應時間增加表示工作停留在系統的時間長，也許是耗費在等待資源或是工作已經執行但因記憶體不足造成執行時間拉長，單以反應時間作為評估標準無法表現出工作耗費在等待資源的時間，但是反應時間縮短，



A



B



C

圖 5 FCFS 與 EASY 在相同環境不同記憶體容量上限的比較 (A)記憶體為 1G (B)記憶體為 1.5G (C)記憶體為 2G

表示工作能夠更快完成它的任務，也代表一個排程方法是有改進效能的成果存在。

Response time = execution time + wait time (7)

5. 實驗結果

首先我們將目標放在相同環境下不同排程方法間的差異性，我們的實驗環境為 128 個 CPUs，工作數據採用 DAS2-fs2，測試不同記憶體容量下 EASY 和 FCFS 排程方法的優劣性。圖 5 為 EASY 與 FCFS 之間的比較，圖 5(A)中在記憶體上限為 0% 的情況 EASY 與 FCFS 處理同樣的工作數據，EASY 所得到的平均反應時間比 FCFS 少了將近 4000 秒，EASY 的排程效率比 FCFS 好上許多；當總記憶體容量為 1.5G 或 2G 時，工作等待系統資源的等待時間減少許多，因此平均反應時間跟 1G 比較之下有

相當大的差異，在記憶體容量比較充足的情況下，EASY 所表現出來的平均反應時間比 FCFS 好。

當排程方法加進有限度開放記憶體上限的考量後，圖 5(A)在開放記憶體上限 10%之內能夠有效的減少反應時間，反應時間上 EASY 表現出來的實驗結果比 FCFS 好；FCFS 排程方法開放記憶體在 26%到 40%時，在 1.5G 或是 2G 的情況下平均反應時間有下降的趨勢，這是由於 26%時執行中工作的執行時間增長，影響到等待中工作的等待時間，然而開放到28%或更高的上限時這些等待中工作有可能可以提前執行，因此等待時間降低造成平均反應時間呈現下降趨勢；開放記憶體到 40%時，EASY 和 FCFS 皆因為記憶體懲罰的因素導致工作的執行時間大幅上升，導致其他等待中的工作要等待更久的時間，因此平均反應時間呈現大幅上升的情況。

實驗結果得知開放記憶體在總記憶體上限 10%以內能夠降低平均反應時間，而在 10%以上減少的等待時間無法抵消增加的工作執行時間，甚至會出現平均等待時間反而延長的情況。之後的測試結果將著重於 EASY 排程方法的改良。

5.1 加入等待時間考量

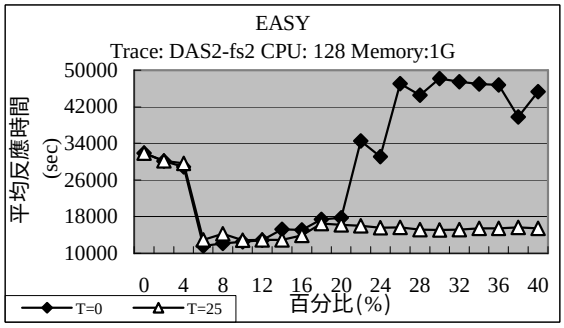
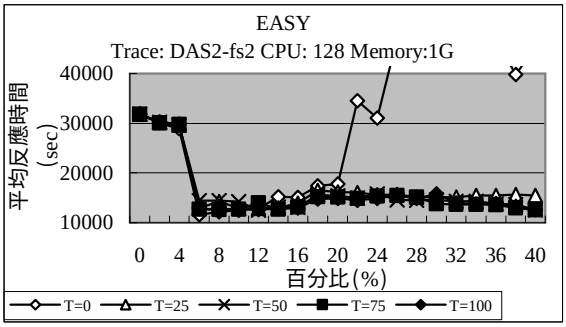
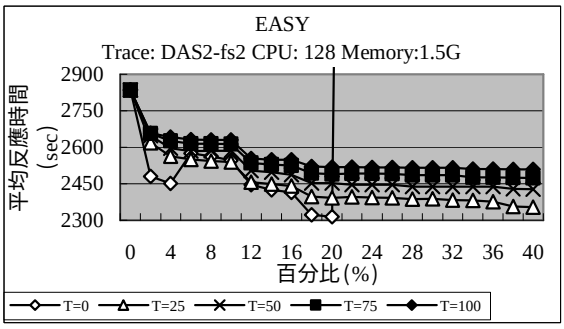


圖 6 加入等待時間考量的 EASY 排程方法

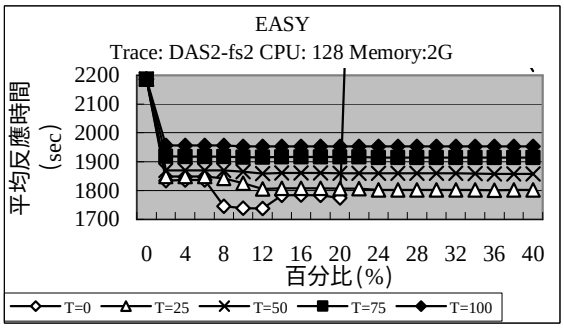
圖6是EASY排程方法加入等待時間考量後門檻值為 0 及 25 的結果，根據門檻值為 25 的實驗結果顯示我們的新排程方法能夠有效並穩定地減少平均反應時間，因此我們會好奇在不同的門檻值和記憶體容量設定下，其平均反應時間的改變是否有不同。圖 7 為門檻值在 0(未使用門檻值的對照組)、25、50、75、100 時所模擬出來的實驗結果，我們可以發現 1G 情況下門檻值為 25 以上時與門檻值為 25 時相差不遠，不同的是隨著門檻值調高，反應時間會更快進入穩定的狀態，而在記憶體容量為 1.5G 以及 2G 時，記憶體資源不足的情況較為緩和，故等待時間比 1G 減少許多並且能夠達到門檻值的工作也比 1G 時要少，因此因為工作提前執行造成等待時間延長的情況發生次數也較少，未使用門檻值的圖表的反應時間有比加入門檻值的結果還要低的現象，表示更低的門檻值或許有更好的實驗結果，然而為了實驗結果的穩定性，在大部分情況下加進等待時間考量的方法會比較好且穩定。實驗結果得知門檻值設為 25 所獲得的結果最好，門檻值



A



B



C

圖7 DAS2-fs2工作數據在不同系統記憶體容量下，不同門檻值的平均反應時間 (A)記憶體為1G (B)記憶體為1.5G (C)記憶體為2G

越高表示受到等待時間考量的排程方法影響到的工作愈少，因此降低的數值不如門檻值為 25 時來的佳。在記憶體為 1G 時門檻值設為 100 為最佳情況。

我們可以獲得一個結論，在記憶體資源不足情況較多的時候，門檻值最好設在 50 以上，減低因為工作提前執行所增加的延誤等待時間，並且將受排程方法影響的工作數量控制在中等程度，而當記憶體資源大部份時間都足夠使用時，建議門檻值設為 25 到 50 之間，在這個門檻值之下針對工作執行時間所做的等待時間考量，能夠充分利用而做到有助於反應時間改善。

6. 結論

系統資源短缺會造成整體系統效能的嚴重下降，本研究針對前人學者所提出的 space sharing 方法作改良，對其加進記憶體考慮的新型排程方法，

同時針對在系統資源短缺的情況下，有限度地開放記憶體上限將能增進排程方法的效能，我們使用模擬器模擬真實工作環境的結果，並且搭配不同硬體規格的環境以求得記憶體容量對工作排程的影響，經過實驗結果指出，我們的排程方法對於 FCFS 能獲得最明顯的改善，最佳的反應時間下降比例約為 30%。

針對 EASY 排程方法，加進等待時間考量的排程方法獲得的效果不錯，在 DAS2-fs2 中最佳情況可以減少 10%以上的反應時間，而在少數情況下未加入等待時間還能獲得更好的反應時間，表示門檻值的選定還沒有達到最佳化的狀態，但是我們門檻值的設定並不是要取得一個最好的平均反應時間，而是要兼顧反應時間的減少以及穩定地降低反應時間，實驗結果可以發現針對 DAS2-fs2 這類需求記憶體小的工作數量較多的工作數據，我們提出的排程方法有很好的穩定減少效果，証實我們的排程方法能夠改善 space sharing 排程方法。

6.1 未來發展

我們的研究工作還沒有結束，本研究並沒有去分析系統現在執行中工作的剩餘執行時間，剩餘時間短的工作很快就會完成它的工作，因此在內文切換次數增加的比例上面，並不會如同長程執行時間的工作來的多，未來將會在space sharing 排程方法上對於放寬記憶體上限的考慮部分，以及系統現存而正在執行中工作的剩餘執行時間做評估，如果系統內部長程工作分布的比例較大，那麼當時便不考慮增加記憶體上限，其內文切換的執行時間不變，所有工作的平均反應時間比起採用增加記憶體上限的情況能夠更進一步地減少。

門檻值的選定對於我們提出的排程方法有顯著的差異，適當的門檻值不僅能夠有更低的平均反應時間，也能使平均反應時間減少的速度更快達到平衡點，因此對於需求記憶體大的工作數量較多的時候，我們可以採用經驗法則統計先前所完成工作的種類動態採用不同的門檻值，在大量需求的工作多的時候採用高的門檻值，而低需求的工作比例佔多數的時候將門檻值降低，希望能結合在不同工作數據上所得到的實驗結果，來將我們所提出的排程方法做到更好的增進效果。

參考文獻

[1] Dror G. Feitelson, Ahuva W. Mu'alem, "Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling", IEEE Transactions on Parallel and Distributed Systems, volume 12, issue 6, pp. 529-543, June 2001.

[2] Dror G. Feitelson, Larry Rudolph, "Parallel Job Scheduling --- A Status Report", 10th Workshop on Job Scheduling Strategies for Parallel Processing, pp. 1-16, June 2004.

[3] Dror G. Feitelson, Anat Batat, "Gang Scheduling with Memory Considerations", In Proceedings of 14th International Parallel and Distributed Processing Symposium, pp. 109-114, 2000.

[4] D. Lifka, "The ANL/IBM SP Scheduling System", 1st Workshop on Job Scheduling Strategies for Parallel Processing, pp. 295-303, April 1995.

[5] J. Skovira, W. Chan, H. Zhou, and D. Lifka, "The EASY-loadleveler API project", 1st Workshop on Job Scheduling Strategies for Parallel Processing, pp. 41-47, April 1996.

[6] Parallel Workload Archive. URL http://www.cs.huji.ac.il/labs/parallel/workload/l_das2/index.html

[7] The DAS2 5-Cluster Grid Logs. URL http://www.cs.vu.nl/das2

[8] Y.C. Tay, Min Zou, "A page fault equation for modeling the effect of memory size", Performance Evaluation, volume 63, issue 2, pp. 99-130, 2006.