在 Linux 系統上設計與實現負載平衡設施*
The Design and Implementation of Load Balancing Facility in Linux

楊竹星，曾群偉，馬志銘
C.S. Yang, C.W. Tseng, C.M. Ma

國立中山大學資訊工程研究所
Institute of Computer and Information Engineering
National Sun Yat-Sen University, Kaohsiung, R.O.C.
csyang, cwtseng, jmma@cie.nsysu.edu.tw

摘要

本論文提出以 Offset 觀念爲主的負載平衡策略，並在 Linux 作業系統設計與實作負載平衡設施，文中定義工作負載爲佇列長度、記憶體使用率及 Disk I/O 佇列三項指標加權相加，使系統可動態調整其加權比重以適應不同的工作型態。並將工作放置到合適的節點上執行以達到負載平衡，提高系統輸出量，改善系統的整體效能。

關鍵字：負載平衡，負載指標，工作負載

Abstract

In this paper, we propose a Candidate Group load balancing strategy, and implement a load balancing facility in Linux kernel. Our workload is the sum of the queue length, memory utilization and Disk I/O queue length indexes. The load balancing mechanism reduce average job execution time and increase system throughput.

Keyword : load balancing, load index, workload

1. 簡介

由於微處理機的進步及網路技術的提升，使得計算環境從過去的單一大型主機演變成今天以網路連接數十個節點而成的分散式系統，具有資源分享、容錯、可擴充及可靠等特性。然而在實際的經驗中，我們發現並不是網路上每個節點都能充分發揮它的效能，有些節點的工作負載很重，而其它節點的工作負載卻很輕、甚至是閒置的情形，使得整個系統負載分佈不平衡。

因此，如何改善系統的效能，成爲分散式系統重要的議題。要若解決各個節點負載不均的情況，系統必須具備將負載重的節點的負載分攤到負載輕的節點上的能力，防止負載不均的情形繼續發生。也就是說，需要有負載平衡的措施（ Load Balancing Facility ），使得每一個工作都能得到比較多的資源，加快執行的速度、降低系統的平均回應時間及提升整體的效能。

1.1 研究動機與目的

本論文的目的是在作業系統中，建制一個負載平衡措施，有效地將負載分配到適當的節點，避免工作過度集中於某些節點，造成壅塞的現象，而相對其他節點卻使用率低落。就行程管理來說，新的工作進入節點，只能在該節點執行，一般作業系統雖然有提供遠端執行指令 rsh ，但使用者在使用前需先決定要在那一台機器上執行，並設定環境；而且工作從開始執行直到結束或發生錯誤後才會離開，完全不具通透性及效能。故本論文設計負載平衡措施，監視網路上各工作站的負載情形，根據負載平衡策略，動態地將工作分配到合適的節點執行。因此負載平衡的過程完全是由系統依所設計之演算法調整，對使用者來說，仍然只是面對一台電腦。此外，一個 Task 可能由幾個 sub-task 所構成，有負載平衡的功能，即可分散執行這些 sub-task ，充份利用網路上其它節點的運算能力，即可增加整個系統的運算能力，達成分散式系統的擴充性。

1.2 相關背景介紹

從工作排程的觀點來看，負載平衡設施是將工作站內本地排程(local scheduling)擴展成整體排程(global scheduling)[1]：網路上的各工作站，皆可以是作業平台，工作不受限於本地執行，亦可在遠端執行。如此便產生了兩個問題：一、如何選擇執行機器?二、如何讓工作在選定的機器執行?

對於一個工作來說，是否有一個最佳的執行地點?這裏所謂的最佳，指的是工作可以得到最佳的回應時間（ response time ），有一些研究即是針對此來發展負載平衡設施[2 、 3]。 然而有一些限制，如在預測程式的行爲方面，比較不易達到精確的要求，而且也要付出一些效能上的代價。

工作站上行程使用的資源有 CPU 、記憶體、硬碟等，這些資源都具有可共享的特性；程式能夠分配到的資源越多，便可縮短其回應時間。負載不平衡的問題即在於：有些工作站上有太多的行程要輪流行使用有限的資源，而有些工作站卻經常閒置；故從負載平衡的策略上我們以降低系統的平均回應時間，增加資源使用率爲目標。如此整個系統的效能亦可提升。此外，許多負載平衡的研究皆是以此爲努力的方向[4 、 5 、 6 、 7] 。

在經過仔細研究負載平衡問題之後，我們歸納出要在系統上建構負載平衡設施，必需具備以下兩種能力：

- 有效的負載平衡策略
- 遠端執行的能力

第一項屬於策略（ Strategy ），第二項則屬於機制 Mechanism ）。以下就這兩個項目分別說明：

A. 有效的負載平衡策略

首先，我們所面臨的第一個問題是：何時要起將負

*本論文由國科會計畫編號： NSC 86-2213-E-110-032 所補助

載由壅塞的節點分攤到閒置的節點上?也就是。系統在何種狀況下，負載是不平衡的？很顯然，"負載不平衡"只是一個抽象的語句，我們有必要加以定義與量化。所以，我們先要定義何謂負載，並要掌握監控網路上各節點的負載情形，最後才能根據各節點負載的情形，提出一個準則，來判定什麼情形下，系統是處於負載不平衡的狀態，再去平衡負載。

第二個問題是：負載如何平衡? 由壅塞的節點自覺地將工作負載移到閒置的節點，或是由閒置的節點主動向壅塞的節點要求分攤工作負載? 是集中由一個節點來統籌，或是分散到各個節點同時進行呢?

爲解決上述問題，我們擬定負載平衡策略。負載平衡策略在過去的研究裏面， 主要分成三個政策：資訊收發政策（ Information Policy ）、轉移政策(Transfer Policy ）及地點政策（ Location Policy ）[5]。其內容分述如下：

(1) 資訊收發政策(Information Policy)

交換節點之間的個各類訊息，如節點的負載大小等資料。

- 作業方式

範圍可以整個 cluster 爲單位或者以分群、階層式的方式進行。[13]

- 描述評量節點的負載情形定訂 Load Index

一般所謂節點的負載情形指的是節點上各資源的使用情形導致再執行新程式的能力[3] 常見的是 CPU 的 queue length[9]，正在執行中的行程總個數。其它如記憶體的使用情形、硬碟存取要求佇列的長度，皆可作爲負載情形的指標。

- 監控系統各節點負載的方法

1. 集中管理-負載資訊集中於一個節點管理的方式。
2. 分散管理-各個節點記載所有的負載資訊的方式。

- 各節點溝通交換負載資訊的方法

採用各節點負載有所變動時才動作。

(2)轉移政策 (Transfer Policy)

- 作業方式

與資訊收發政策相同。

- 判斷負載不平衡的情況，決定執行遠端執行的執行主體

由一個節點來負責；由擁塞的節點尋找閒置的節點分攤工作或由閒置的節點尋找擁塞的節點，進行負載平衡。

- 如何判斷負載不平衡的情況，何時決定執行遠端執行

首先針對節點的負載情形，訂定擁塞、閒置的標準，便可決定何時系統不平衡；若此時新工作進來時，即作遠端執行，並持續偵測判斷系統上各節點的情形。

(3) 地點政策(Location Policy)

- 決定負載被移出或接受負載的節點

隨機的選擇或以負載的情形作爲依據。

B. 遠端執行的能力

遠端執行是達成負載平衡的手段。依運用時機與方式的不同，分爲兩種：一、新工作放置方法(Initial Job Placement)，二、程序遷移方式(Process Migration)。[7]

(1) 新工作放置方法

當新工作進入系統時，即根據負載平衡策略，選定一個適當的地節點執行，而非一定要在原來的節點執行。

(2) 程序遷移方式

負載平衡機制監視系統的負載情形，若發現負載不平衡，則動態地將擁塞節點上正在執行的行程暫停執行，遷移到閒置的節點繼續執行。

2. 系統設計與架構

在設計負載平衡設施時，必須考量通透性,延展性與容錯等特性，才能符合分散式系統的需求。本節將說明本系統的設計方式與基本架構。

2.1 系統設計

爲了在 Linux 設計與實作一個高效率且具通透性的負載平衡設施，在設計上我們決定採用三種方式：

1. 在 Linux 的核心實作負載平衡設施

以修改核心的方式設計，使用者的程式完全不需修改，就可以如往常般執行，並且能夠使用負載平衡的功能，完全符合分散式系統通透性的要求。

2. 遠端執行支援新工作放置方法

本系統在工作進入節點時支援新工作放置方式，以決定工作在本地或遠端執行。除此之外，工作在遠端執行之後不再接受第二次遠端執行，防止工作發生連續搬移的情形而造成系統資源的浪費。

3. 系統使用 NFS 以提供一個具有一致性之檔案命名空間

採用 NFS 使得工作在任一節點上皆能以同一名稱存取檔案，以解決檔案的地點相依問題，並且讓不同的使用者可以存取同一檔案，達成資源共享的目的。

2.2 基本架構

本系統的基本架構如圖 2-1 所示，每個參與的節點都有三個模組：負載資訊管理模組(Information Manager)、工作放置管理模組(Job Placement Manager)及遠端執行機制模組(Remote Execution Manager)。功能如下:

1. 負載資訊管理(IM)模組：此模組負責負載資訊的接收與傳送。每一個參與節點的 IM 偵測自己的負載資訊，並且傳送給其它節點上的 IM ；同時接受其它節點上的 IM 送來的負載資訊，提供給 JPM 模組作爲負載平衡的判斷依據。
2. 工作放置管理模組(JPM)：新的工作進入接收機器時，JPM 開始運作，參考 IM 所提供的負載資訊，決定是否要作遠端執行，並且爲工作尋找適合的目的機器來執行，再決定是否要啓動 JEC 模組，進行負載平衡。
3. 遠端執行機制模組(REM)：進行遠端執行時，接收機器上的 REC(REM Client)會與目的機器上的 RES(REM Server)建立連結，發出遠端執行需求，將工作的 pathname 及使用者的 security 資料傳送給目的機器，然後等待對方的回應。若一切無誤，工作就可以在目的機器上執行。

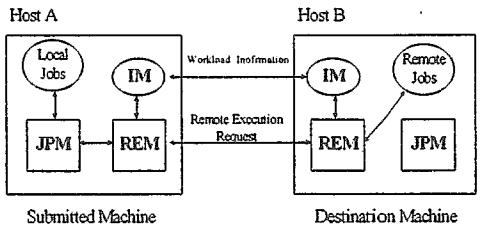


圖 2-1 基本架構

3. 負載平衡策略

負載平衡策略是負載平衡研究中的核心問題。有好的負載平衡策略才能使負載平衡發揮功用，提高系統的效能，避免系統因負載不平衡造成系統資源的浪費。我們的負載平衡策略包含兩個部份：一個是 Load Policy ，另一個是 Job Placement Policy 。

3.1 Load Policy

Load Policy 主要在決定如何訂定負載指標及節點與節點之間如何交換負載資訊。以下分別作說明。

3.1.1 負載指標(Load Indices)

一般負載平衡設施最常以 CPU queue length 作爲工作負載指標（ load index ）。不可否認， CPU 是電腦內部最重要的資源，而其它如記憶體（ memory ）、硬碟空間、網路存取頻率等，亦是影響程式行爲的重要因素。

因此，在設計工作負載指標時，我們考量程式的不同需求，除了 CPU queue length 外，亦將記憶體的使用情行形及硬碟的 I/O queue length 作爲指標一。

我們對節點的工作負載作如下定義：

Workload = (a × Process queue length/ maximum CPU queue length)+(b × (1- free memory/total memory))+ (c × Disk I/O queue length / maximum Disk I/O queue length)

其中 $a+b+c=1$ $0 \leq a, b, c, workload \leq 1$ 。(maximum CPU queue length 及 maximum Disk I/O queue length 爲 Linux 系統之上限)

我們將三項 load index 加權後相加後，便得到量化後的工作負載。 a 、 b 、 c 這三個加權用的係數，可以動態調整，以適合 cluster 不同的工作型態，有效地平衡負載。如在 Database 伺服器組成的 cluster，便可以提高 c 的比重，以降低這些 I/O bound 工作的回應時間。

3.1.2 負載資訊的交換

本系統採用分散式的架構，只有當節點自身的工作負載變動時，才有必要傳送負載資訊給其它節點。如此不但可以減輕網路的負擔，又可以維持負載資訊的正確性。因此， Load Policy 的演算法如圖 3-1 所示。

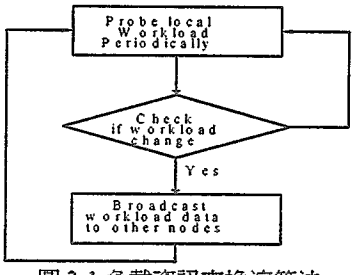


圖 3-1 負載資訊交換演算法

3.2 新工作配置政策

負載平衡策略裏最重要的部份，是決定 job 是否要接受遠端執行及如何選擇一個適當的執行節點，我們稱爲新工作配置政策(Job Placement Policy)。以下說明其內容。

3.2.1 判斷系統的負載情形

過去負載平衡策略的研究，在認定負載不平衡的狀況，常以臨界值(Threshold)的方式[5]來定義系統上壅塞及閒置的節點。當節點的工作負載高於臨界值時，爲壅塞的；當節點的工作負載低於臨界值時則爲閒置的節點。當負載不平衡時，便將新進工作或正在執行的行程從壅塞的節點遷移到閒置的節點。這樣的二分法雖然方便簡單，但有很大的缺點。如圖 3-2 所示， host A 與 B 之工作負載差距不大，但依設定之臨界值來看，一個是 busy ，另一個卻是 idle 。若將 A 、 B 負載平衡之，效果並不大。再來看圖 3-3 ， node B 、 E 工作負載顯然比 A 、 C 、 D 重得多，但由於其工作負載皆低或高於臨界值，故 B 、 E 沒有機會讓 A 、 C 、 D 分攤其負載。因此臨界值定得太高，會失去其作用；若定的太低，則遠端執行發生過於頻繁，則通訊損耗相對變大。

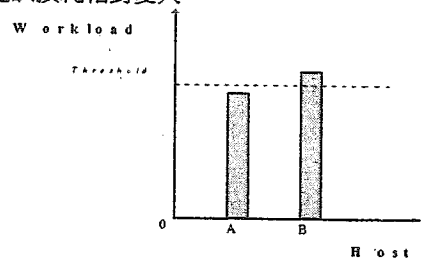


圖 3-2

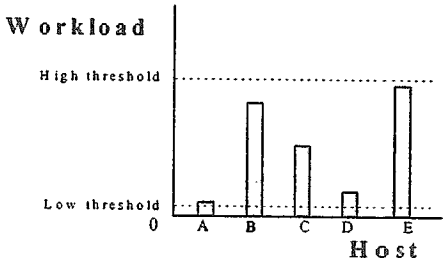


圖 3-3

由於這種二分法的諸多缺點[12]，有的研究便提出使用兩個以上 Threshold 的方法，但僅解決部份的問題。舉例來說，使用兩個 Threshold 的方法將節點分成 heavy load 、 middle load 及 light load 三個 group ，負載平衡的方式爲從 heavy load 的節點送到 light load 的節點上執行。雖然已有效地將壅塞與閒置的節點隔開。避免不必要的遠端執行。然而從圖 3-4 所示，可知仍存在單一 Threshold 的缺失：因爲跟 Threshold 失之交臂，節點 A 與 B 的負載無法負載平衡。

因此我們需要有一個更好的負載平衡方法。如果我們再重新思考負載平衡問題，我們發現壅塞與閒置是相對的概念；我們應該以節點間工作負載的差額（ offset ）作爲判斷依據：當節點間的差額達到某一程度時，工作負載高的節點就是壅塞的節點，工作負載低的則是閒置的節點，我們認爲此時系統即出現負載不平衡的現象；而不應固定地以臨界值作爲節點屬性的分水嶺。如圖 3-5 所示，每種情況皆可以差額來判斷出負載不平衡的狀況。

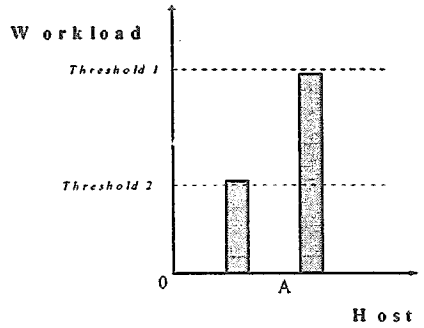


圖 3-4

因此我們需要有一個更好的負載平衡方法。如果我們再重新思考負載平衡問題，我們發現壅塞與閒置是相對的概念；我們應該以節點間工作負載的差額（ offset ）作爲判斷依據：當節點間的差額達到某一程度時，工作負載高的節點就是壅塞的節點，工作負載低的則是閒置的節點，我們認爲此時系統即出現負載不平衡的現象；而不應固定地以 threshold 作爲節點屬性的分水嶺。如圖 3-5 所示，每種情況皆可以差額來判斷出負載不平衡的狀況。

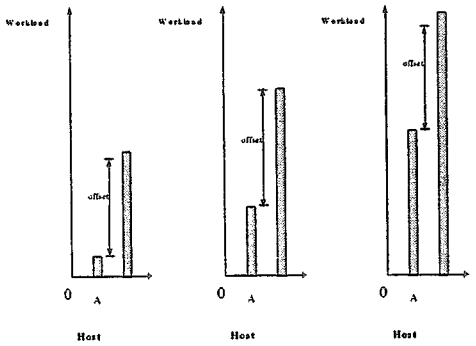


圖 3-5 offset 與不同負載之間的關係

因此在本論文中，提出了一個較完善的新工作配置政策，期望能更有效、機動地決定負載平衡的時機。

3.2.2 候選機器群(Candidate Group)

對一個節點來說，因爲擁有其它節點的負載資訊，將其加以排序便可知道有哪些節點的負載是比自己小的。換句話說，可找出相對於自己是屬於“閒置”的節點。由圖 3-3 ，負載比 B 小的節點有 A 、 C 和 D 。於是我們令 offset 爲 X 。負載比節點 B 的負載小 X 以上的節點我們認爲其相對於節點 B 的關係即爲“閒置”。

接下來是挑選一適當節點作爲節點 C 遠端執行的地點。直覺地我們會挑選 A 爲目的機器（即遠端執行的地點），這也是其它研究常見的作法[9]。但是如此節點 B 、 E 同時對 A 作遠端執行的機率很高，會造成節點 A 負載驟然上升，又製造另一負載不平衡的情形，即所謂“工作碰撞” [8]問題。

A 固然是負載最輕的點，而 D 的負載跟 A 十分接近，似乎也很適合作爲目的機器以執行工作，而 C 對 B 來說雖然也是閒置的，但由圖 3-3 觀察出， C 跟 A 、 D 有一段差距。我們認爲， D 、 A 比 C 更適合作爲目的機器。

根據以上的觀察，我們提出候選機器群(Candidate Group)的觀念，即建立所有可能當作目的機器的集合，再從其中挑選一節點做爲真正的目的機器，以降低工作碰撞的機會。

我們定義候選機器群如下：

假設 N_1 、 N_2 、 N_3 .....Nn 爲 cluster 上的 n 個節點，其負載大小關係爲 $L_1 < L_2 < L_3$....，則任一節點 N_i 的候選機器群 CG(i)可由下列方式求得：

(1) CG(I)= ϕ

(2) For j=1 to n

```
{
 if ((Li - Lj) > X )) then
     CG( i )=CG( i ) + { Nj }
 else
     goto(3)
}
```

(3) We get Candidate Group

最後是從候選機器群中選一個節點當作目的機器。我們以隨機的方式挑選一節點。如此，發生工作碰撞的機會便大爲減少。如圖 3-6 所示，節點 C 的候選機器群由 A 和 D 兩節點所組成。

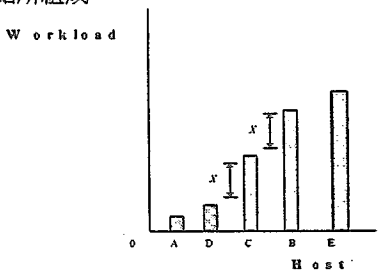


圖 3-6

所以新工作放置政策的流程是：當新的 job 進入節點後，就計算它的候選機器群，假如是空集合，則表示目前系統的負載處於平衡的狀態，工作留在本地端執行即可，無需送至遠端執行；假如不是空集合，則表示有其它閒置的節點。接著便從候選機器群中隨機地選擇一個節點作爲目的機器，進行遠端執行之程序。

4. 遠端執行(Remote Execution)

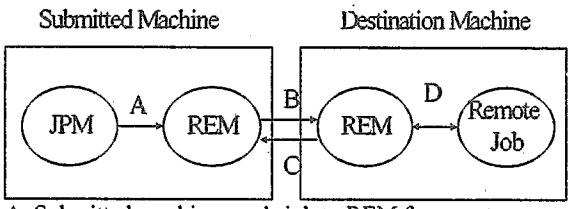
負載平衡策略透過遠端執行機制，使得工作能在比接收機器更適合的目的機器上執行。本節說明我們的系統所採用的遠端執行機制。

4.1 新工作放置方法(Initial Job Placement)

爲了讓遠端執行能符合效率的要求，我們採用新工作放置方法的機制。因爲採用這樣的方式接收機器只需要傳送工作的 pathname 與 security 的資料給目的機器，即可在目的機器上執行。假如工作已經變成一個執行中的行程，就要用行程遷移的方式，但是由於行程與核心已經產生密切的關係(如核心內屬於該行程的資料結構)，要將行程遷移到別的節點，要付出相當大的代價。

4.2 新工作放置模式

如圖 4-1 所示，我們採用 client-server model 的架構：當新進工作決定要以遠端執行的方式來完成時，JPM 將 job 的相關資料(job pathname 及 user name)和目的機器的 IP 位址交給 REM ，REM 就與目的機器上的 REM 建立連結，並傳送遠端執行需求；目的機器上的 REM 在確認 job 的資料正確無誤之後，即執行所指定的工作，當資料不正確時，會傳送錯誤訊息給接收機器上的 REM 。



A: Submitted machine sends job to REM for remote execution
B: REM of the submitted machine send remote execution request to destination machine
C: REM of the destination machine checks pathname and user authentication, then send acknowledgement back to REM of the submitted machine
D: REM creates remote job

圖 4-1 新工作放置模式

5. 系統實作

如 3.2 所述，負載平衡設施在實作上有兩種方法：一是在 user level 上架一層中介軟體，二是在核心內部增加負載平衡的功能。基於效能與通透性的考量，本篇論文選擇以延伸核心功能的方式來進行。本節將描述如何在核心內部建置負載平衡設施、系統的整體架構及系統運作方式。

5.1 延伸核心功能

基本上，核心可視爲 service routines 的集合，提供服務給在 user level 的行程使用。行程與核心的介面爲系統呼叫(system call)，行程是以系統呼叫的方式，來使用核心所提供的功能。因此，藉由修改或增加系統呼叫，我們就可以增加或改變核心的功能，同時不被使用者察覺。

經過長時間的追蹤 Linux 核心的原始碼，我們決定：

(1)修改系統呼叫 sys_execv

加入 JPM 模組，把新進的工作攔截住，並且根據 JPM 的演算法爲工作尋找一適當的執行節點。

(2).增加一些系統呼叫

由於核心是處於被動的角色，只有使用者呼叫系統呼叫時，才能執行核心的程式碼。因此，我們將核心要增加的功能，寫成新的系統呼叫，再由 user level 的程式呼叫這些新的系統呼叫,並以迴圈的方式停留在核心，形成核心 daemon 。如此一來這些核心 daemon 便能像 user level 的 daemon 般週期性地執行。除此之外，各個節點上的核心 daemon 皆以 RPC 的方式互通訊息。

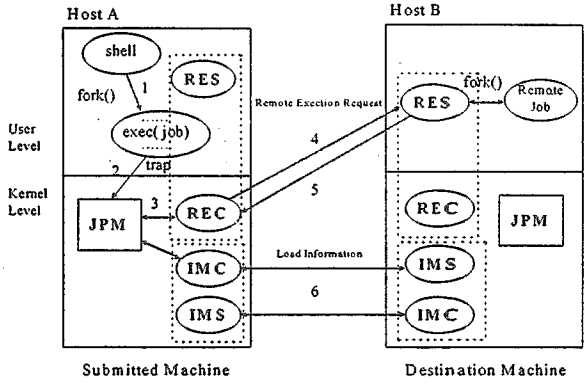
5.2 系統整體架構

系統的整體架構如圖 5-1 所示。我們把 REM 分成兩個部份：Remote Execution Client(REC) 與 Remote Execution Server(RES) ； IM 也分成 Information Management Client(IMC) 與 Information Management Server(IMS)。接收機器上的 REC 與目的機器上的 RES 以 client-server model 的方式完成遠端執行；每個節點上的 IMC 負責偵測自身負載的變化情形，並以廣播的方式將負載資訊傳送給其它節點上的 IMS ； JPM 則是負責 Job Placement Policy 。其中， REC,IMC 及 IMS 是屬於 kernel daemon ， JPM 是加在 system call execv 的模組，而 RES 是在 user level 的 daemon 。

5.3 系統作業流程

Linux 系統上要執行一個工作，必定是由一個 parent process 執行系統呼叫 fork() ，產生 child process 之後，再由 child 執行系統呼叫 execv ，將工作載入並執行。我們以圖 5-2 來說明系統是如何運作的。

假設我們的使用介面是 shell,當我們在 shell 的提示符號後輸入工作名稱後， shell 就"fork"一個 child process ，接著 child 便執行系統呼叫 execv() ， trap 進入核心，此時加在 execv()裏的 JPM 就會根據 IMS 提供的系統負載情形，判斷是否要作遠端執行；答案是否定的話，則工作依原來 execv()的邏輯，留在本地執行。若 JPM 決定要作遠端執行，就將目的機器的位址、工作的路徑名稱及使用者的資料傳送給 REC ； REC 依照 JPM 所給的位址與目的機器上的 RES 建立連結，發出遠端執行要求，內含工作的路徑名稱與使用者的資料。 RES 收到要求之後，檢查工作路徑與使用者的資料是否正確。若正確無誤， RES 執行系統呼叫 fork() ，產生一個子行程，子行程接著以系統呼叫 execv() ，執行所指定的工作；否則 RES 會將錯誤訊息傳回給接收機器上的 REC ，宣告遠端執行失敗。



Remote Execution

1. Job is submitted and shell "fork" a child process
2. Shell's child process "execv()",and trap into kernel with job pathname and user data
3. When JPM decides to do remote execution, it send destination address, job pathname and user data to REC for remote execution 。
4. REC establishes a connection with RES in destination machine, and send remote execution request
5. RES in destination machine check the request. If both job pathname and user data are correct, RES fork and execute the job specified; if not, RES send back error message.

Load Distribution

6. IM in each node probe its workload and broadcast to each other periodically.

圖 5-1 系統執行流程

6. 實驗結果

本系統在量測負載平衡設施時，採用的測試模式是讓系統處於閒置的狀態，再施予某種 Job Pattern ，再觀察負載平衡設施對系統整體效能的影響，量測節點工作平均執行時間當作系統效能的指標。

系統的測試環境爲以 Ethernet 10-based 連結的網路，五臺 AMD K5 爲 CPU 、 16MB RAM 的 PC 工作站爲平臺，再加上一臺 Pentium 133 、 16MB RAM 的 PC 工作站爲 NFS Server 。

本系統以不同的 Job Arrival Rate 來測試在沒有負載平衡設施，動態的 Threshold-based ，靜態的 Threshold-based ， Candidate Group I及 Candidate Group II(註)五種負載平衡策略下得到的平均工作執行時間。 Job Arrival Rate 由每秒 0.3 個到每秒 0.64 個，共八種 Job Arrival Rate ，每種持續 10 分鐘。量得的工作執行時間單位爲 millisecond，並且以求出平均工作改善率來比較各個負載平衡策略的效能。其公式爲 ：

Promotion Percentage ：

[(X - Y)/ X]* 100 %

其中, X : 沒有負載平衡的平均工作時間

Y : 有負載平衡的平均工作時間

測試結果如圖 6-1 ，我們觀察在不同的 job arrival rate 時，每個負載平衡策略的表現。由圖 6-1 可以看出，整體上，本論文提出的負載平衡策略(C.G.I, C.G.II)表現最好，動態的 Threshold-based 負載平衡策略次之，靜態的 Threshold-based 負載平衡策略最差。 Candidate Group 的負載平衡策略與動態的 Threshold-based 負載平衡策略相比較，在系統負載輕時差異較小，當系統負載增加時，前者就明顯地比後者有較佳的表現。 C.G.I 與 C.G.II 的不同在於定義 Candidate Group 的條件，前者較寬，而後者強調 Candidate Group 的 locality，即 Candidate Group 中皆是工作負載輕而且相接近的節點；結果顯示 C.G.II 的效能優於 C.G.I 。至於靜態的 Threshold-based 負載平衡策略，由於有本篇論文前面所述的諸多缺點，其效能比其它的負載平衡策略遜色許多。由數據顯示，本篇論文所提出的由 offset 的觀念延生而成的 Candidate Group 負載平衡策略確實爲一種優秀的負載平衡策略。

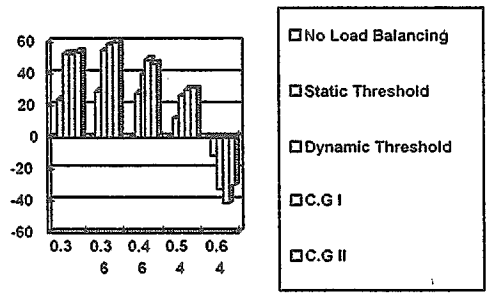


圖 6-1 平均工作時間改善率

(註: Candidate Group II 即爲本篇論文所定義的 Candidate Group，而 C.G. I 與 C.G II 不同點在，在求 Candidate Group 時，將條件放寬，只要跟該節點工作負載的差意異大於 X(系統所定義)的節點，即可納入 Candidate Group)

7. 參考文獻

[1] M. M. Theimer and K. A. Lantz, " Finding Idle Machine in a Workstation-based Distributed System," IEEE Transactions of Software Engineering ,Vol. 15, No.11, pp.112-122, Nov. 1989.

[2] K. K. Goswami, M. Devarakonda and R. K. Iyer, "Prediction-Based Dynamic Load-Sharing Heuristics," IEEE Transactions on Parallel and Distributed Systems, Vol. 4,No. 6, pp.638-648, June 1993.

[3] 陳 躍 仁 , "Prediction-Based Load Balancing in Heterogeneous Systems,"國立中山大學資訊工程研究所碩士論文, 1994.

[4] S. Zhou, "A Trace-Driven Simulation Study of Load Balancing, "IEEE Transactions of Software Engineering, Vol. 14, No. 9, pp.1327-1341, Sep. 1988.

[5] D. L. Eager, E. D. Lazowska, and J. Zahorjan, "Adaptive Load Sharing in Homogeneous Distributed Systems," IEEE Transactions of Software Engineering, Vol. 12, No. 5, pp.662-675, May 1986.

[6] A. Hac and X. Jin, "Dynamic Load Balancing in a Distributed System Using Decentralized Algorithm," Department of Computer Science, The John Hopkins University, Baltimore, Maryland, 1987.

[7] O. Kremien and J. Kramer, "Methodical Analysis of Adaptive Load Sharing Algorithms," IEEE Transactions on Parallel and Distributed Systems, Vol. 3,No. 6, pp.747-760, Nov. 1992.

[8] C. J. Hou, K. G, Shin and T. K. Tsukada, "Transparent Load Sharing in Distributed Systems : Decentralized Design Alternatives Based on the Condor Package," IEEE, 1994.

[9] Shepherd S. B. Shi , D. H. Lin and C. S. Yang, "Dynamic Load Sharing Services with OSF DCE," IEEE The First International Workshop on Service in Distributed and Networked Environments, June 27-28, Prague, pp. 178-186, 1994.

[10] 楊敏玲, "在 Linux 上設計與實現行程轉移," 國立中山大學資訊工程研究所碩士論文, 1995.

[11] 杜威慶, "在 Linux 系統中使用轉遞機制解決行程遷移的 IPC 問題," 國立中山大學資訊工程研究所碩士論文, 1996.

[12] O. Kremien, J. Kremer, "Scalable, Adaptive Load Sharing for Distributed Systems, " IEEE Parallel & Distributed Technology, pp.62-70, August 1993.

[13] John G. Vaughan ,"Incorporating Job Size in Distributed Load Balacing," Micro-processing and Microprogramming 41, pp.111-119, 1995.

[14] S. Zhou, J. Wang, X. Zheng, and P. Delisle, "UTOPIA: A LOAD Sharing Facility for Large, Heterogeneous Distributed Computer Systems, "Technical Report CSRI-257, April 1992

[15] W. Zhu, C.F. Steketee, "An Experimental Study of Load Balancing on Amoeba," IEEE, 1995.