國 立 交 通 大 學

資訊科學與工程研究所

碩 士 論 文

桌機格網聯盟之資源分配管理

The Study and Design of Resource Allocation Management for the Desktop Grid Federation

研 究 生：張元耀

指導教授：吳毅成　教授

中 華 民 國 一 百零二 年 九 月

The Study and Design of Resource Allocation Management for the Desktop Grid Federation

研 究 生：張元耀　　Student：Yuan-Yao Chang

指導教授：吳毅成　　Advisor：I-Chen, Wu

國 立 交 通 大 學
資 訊 科 學 與 工 程 研 究 所
碩 士 論 文

A Thesis
Submitted to Institute of Computer Science and Engineering
College of Computer Science
National Chiao Tung University
in partial Fulfillment of the Requirements
for the Degree of
Master
in

Computer Science

September 2013

Hsinchu, Taiwan, Republic of China

中華民國 一百零二 年 九 月

桌機格網聯盟之資源分配管理

研究生：張元耀　　指導教授：吳毅成

國立交通大學 資訊科學與工程研究所

摘要

由於電腦對局遊戲的應用問題，必須使用龐大的計算資源，再加上電腦對局遊戲通常具有高動態、低延遲等的特性，因此我們實驗室過去開發一套之適用於電腦對局遊戲之桌機格網系統－Computer Game Desktop Grid (CGDG)，成功地解決一些問題。

為了增進 CGDG 在分配運算資源的公平性和提升 CGDG 所分配運算資源給使用者的效能，本篇論文提出一些改良 CGDG 分配運算資源的方法，這些方法兼具資源分配的公平性和資源使用的效能。此外，本篇論文論述均分、穩定地使用核心數量能提供較好的效能，並且最後以實驗來驗證之。

The Study and Design of Resource Allocation Management for the Desktop Grid Federation

Student: Yuan-Yao Chang Advisor: I-Chen, Wu

Institute of Computer Science and Engineering

National Chiao Tung University

Abstract

We usually need a huge amount of computing resources to solve the computer game applications. Furthermore, these issues of computer game applications are highly dynamic and low latency features. Thus, our laboratory has developed Computer Game Desktop Grid (CGDG) for these applications.

This thesis improves CGDG resource allocation management, including: the fairness between users or organizations, the fairness for multi-threading jobs and the efficiency of allocating computing resources. Furthermore, this thesis also argues that the more stable, the better performance is. Finally, this thesis confirms the argument by experiments

誌謝

這篇論文得以完成，首先要感謝我的指導教授吳毅成教授，藉由老師的建議及指導，使我學會很多程式設計觀念，例如：剛開始的在探討程式架構的過程中，會解釋每項設計的理念，讓我了解其設計原因；老師也會分享他以前程式除錯的經驗，尤其是這部分讓我受益良多。另外也很感謝陳隆彬教授、吳昇教授、江振瑞教授，給予我建議，讓我了解到許多自己沒想到的議題，以及其他應該補強的地方。

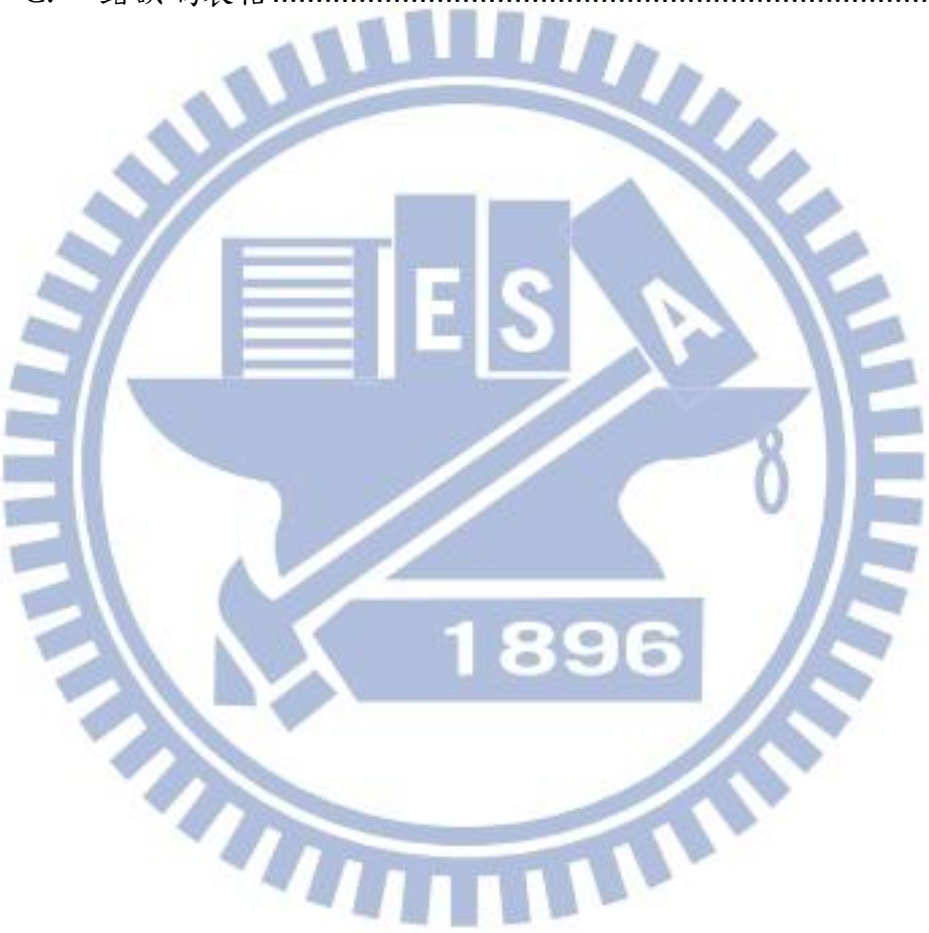
感謝英明神武浩雲哥：在我碩一的時候，總是用非常敏銳的眼光指出我的缺失，幫助我、使得我的研究做到更好。感謝中哥、軒哥、傑哥、廷翰：給了我許多口試投影片以及論文上的建議，讓我可以順利完成。感謝吳東穎、張傑閔、廖挺富：常常和我討論在撰寫程式的過程中，所遇到的種種困難與窘境。感謝拉拉、KK、蛙哥、小吉、嘉嘉、左左：在網路程式設計和高等 JAVA 設計的課程中，不厭其煩地回答我的疑問、給予我額外的課程幫助。感謝黑月、味精、包子、阿賢、庭築：在閒暇時間，一起揪團運動，培養健康的身心。感謝所有實驗室的成員，給予了我平時的幫忙以及建議，讓我得以順利完成我的研究。

感謝我的家人，每次回家，父母都準備很豐盛的餐點，並且很照顧我的身體健康，也不時地給予我鼓勵，讓我在研究的路途上，能夠健健康康、順順利利。

最後僅以這篇論文，獻給所有在我求學路程上曾幫助、關心我的人。大家的支持是我完成此論文及學業最大的助力，謝謝大家。

民國 一百零二 年 九 月　於　新竹交大工三館 CGI-511 實驗室

目錄



圖目錄

表目錄

第一章、緒論

由於電腦對局遊戲的應用問題，必須使用龐大的計算資源，因此可以利用志願型計算(Volunteer Computing)來更有效率地解決此類問題。

在本章節中，將在 1.1 節會先介紹電腦對局遊戲應用，在 1.2 節之中則會介紹本篇論文的研究動機與目的，在 1.3 節為論文大綱說明。

1.1 電腦對局遊戲應用

電腦對局遊戲領域(Computer Game)自從深藍(Deep Blue)[11]－由許峰雄博士所發展的西洋棋人工智慧程式，成功擊敗了人類冠軍棋士 Garry Kasparov 後，使得電腦對局相關領域的研究受到大家的關注，也使得更多學者紛紛投入於該領域之研究，例如：圍棋、象棋、五子棋等知名的棋類遊戲。

到目前為止，我們實驗室已經開發了許多電腦對局遊戲應用程式，例如：六子棋程式之 NCTU6[20]、圍棋程式之 HappyGO、象棋程式之棋謀等。除了我們以外，來自於其他組織的使用者也開發了不少應用，例如：長榮大學許舜欽教授與黑白棋(Othello)[1]、東華大學顏士淨教授與圍棋程式、師範大學的林順喜教授與圍棋[9]程式開發及陳志昌教授與象棋[24]開局庫和澎湖科技大學高國元教授與組合對局遊戲理論(Game theory)[14]。

1.1.1 特性

電腦對局遊戲應用有著以下兩點特性：

1. 高複雜度、大計算量：由於大多數電腦遊戲對局應用的時間複雜度為指數時間複雜度，且需要龐大的計算資源，所以可以利用志願型計算來有效率地解決問題。

2. 高動態性：以圍棋為例，當圍棋應用程式找到必勝步時，就可以把其他搜尋分支砍掉；或是當圍棋應用程式認為快要贏的時候，就會專注在特定分支做搜尋。

對於上述兩點，可以使用志願型計算來加速、且有效率地解決電腦遊戲對局應用問題。

1.1.2 志願型計算

志願型計算(Volunteer Computing)[15]主要是利用許多閒置的計算資源，例如正在閒置的桌機(Desktop Computer)，透過網路，將這些來自不同組織的計算資源，集結成龐大的聯盟(Federation)運算資源。志願者們將這些閒置的計算資源捐出來(cycle stealing)[12]，提供給特定的計畫以完成需要大量計算的應用問題。

1.1.3 CGDG

電腦遊戲桌機格網(Computer Game Desktop Grid)[13]的架構如圖 1 所示，此系統具有以下兩項能力：

1. 能支援、控制高動態的工作[6]。
2. 能支援多個平行運算的應用程式。

過去利用此系統成功地解出六子棋的米老鼠開局[6]以及其他電腦對局的應用問題。

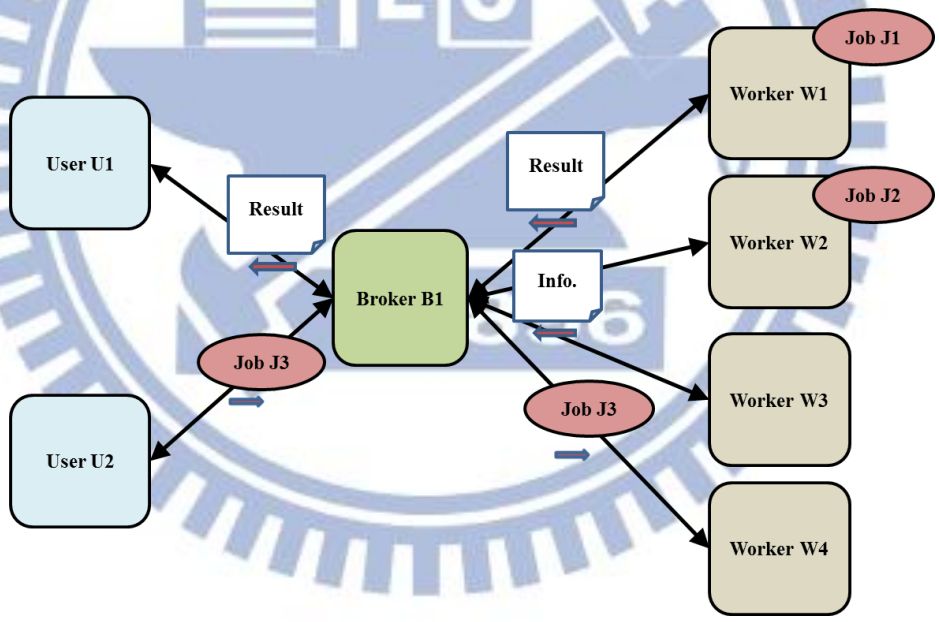


圖 1：CGDG 架構圖

本系統的使用者(User)，主要功能為發送工作(Jobs)，例如 AI 計算、開局庫問題、或其他電腦對局應用問題給系統中樞，仲介者(Broker)。仲介者(Broker)會把資源公平地分配給每個使用者，再把使用者發送的工作分配至

後端的計算資源，也就是工作者(Worker)，當工作者運算完成後，再將結果傳回給使用者。

工作者(Worker)[10]，主要功能為提供計算能力，執行仲介者(Broker)所分配的工作，並將其結果回報給仲介者。除了計算的工作之外，工作者也會定期地回報工作者自己的狀態給仲介者，並且透過仲介者將資訊傳回給使用者。

仲介者(Broker)[7]：這是系統的中樞，主要功能為協調使用者(User)與工作者(Worker)之間的溝通，負責將資源公平地分配給每位使用者，並且將工作發派至合適的工作者上;此外,也會將工作者的資訊傳回給使用者，讓使用者知道目前系統資源的使用狀態。

1.2 研究動機與目標

由於目前已經有許多使用者、許多組織加入使用 CGDG，例如：國立東華大學、國立師範大學、中原大學、東海大學等組織。

然而在眾多的使用者或組織之中，如何將運算資源公平地分配給使用者、如何提升所分配運算資源的使用效能，是一些很重要的議題，因此以下的目標分為兩點：

1. 公平性問題：以下要解決的問題再分為兩點：

   I. 使用者之間或組織之間的公平性問題：由於使用者或組織之間共享資源，如何將運算資源公平地分配給使用者或組織成為重要的課題,因此我們希望設計一套方法,達到分配運算資源的公平性。

II. 使用多執行緒之工作的公平性問題：以往使用者發送的工作都為單執行緒，如今隨著 AI 程式的演變，其中為了增強棋力，多執行緒之工作也漸漸產生，因此我們希望設計一套方法，達到分配資源給多執行緒之工作的公平性。

2. 資源使用效能問題：基於系統的角度，運算資源的使用率越高越好，再加上應用程式使用運算資源的多寡，會影響其效能，因此我們希望設計一套方法，以運算資源的使用率高且分配資源公平為前提，提升應用程式所使用的運算資源效能。

為了增進 CGDG 在分配運算資源的公平性和提升 CGDG 所分配運算資源給使用者的效能，本篇論文提出一些改良 CGDG 分配運算資源的方法，這些方法兼具資源分配的公平性和資源使用的效能。

1.3 論文大綱

在第一章，此篇論文介紹了電腦對局遊戲應用的相關背景，並提出了研究動機與目標。在第二章中，會介紹相關格網系統以及過去實驗室開發的桌機格網系統 (CGDG)。在第三章中則會介紹仲介者的資源分配管理設計，包含使用者或組織之間的公平性、多執行緒工作所使用資源的公平性、分配資源所使用的效能。在第四章中則會介紹仲介者的系統實作及實驗結果。第五章則為論文總結以及說明未來的研究方向。

第二章、研究背景

由於電腦對局遊戲的應用問題，必須使用龐大的計算資源，因此可以利用志願型計算(Volunteer Computing)來更有效率地解決此類問題。

以下將介紹著名的志願型計算系統，首先於 2.1 節介紹 BOINC，2.2 節介紹 XtremWeb，最後 2.3 節則是介紹 CGDG。

2.1 BOINC

BOINC[2][4]是一個有名的志願型計算的中介軟體，由柏克萊大學加州分校的電腦學系於 2,002 年所發展，由 David Anderson 所領導開發，被廣泛的應用在生物科技、地球科學、數學、物理及天文等應用問題，包括 SETI@Home[16]、Background Pi 計畫[3]等。

BOINC 架構如圖 2 所示：

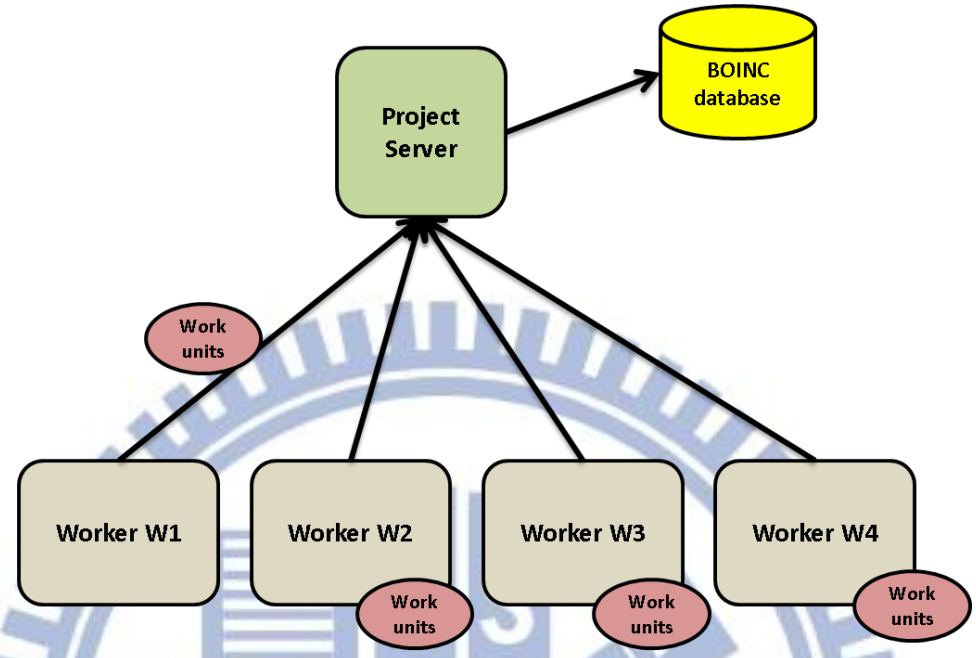


圖 2：BOINC 架構圖

BOINC 具有下列這些性質：

- BOINC 資料庫(BOINC database)：
  - 儲存工作相關資訊(例如：這份工作耗時多久、工作結果...等)。
  - 儲存工作所需要的輸入檔案。
  - 刪除不需要工作或檔案...等。

- 專案伺服器(Project Server)：
  - 將 BOINC 資料庫裡面的工作，包裝成一份工作包(Work units)。
  - 發派工作包、工作包之排程和驗證工作結果等。
  - 每個伺服器只提供一個應用(Project autonomy)。
  - 所有工作包皆由伺服器管理者控制。

- 工作者(Worker)：
  - 採用輪詢模式(Pull model)：定期向伺服器溝通、向伺服器取得要執行的工作包，執行計算該工作包裡面的工作們。
  - 支援離線(off-line)計算模式：當工作者因為網路因素無法與伺服器連線時，工作者仍會繼續執行工作，並將結果存在本地端，直到下次有機會與伺服器重新連線時再傳送結果。
  - 如果志願者想使用工作者的電腦時，工作者會將工作進行記錄點(Checkpoint)並中止工作，所以並不會影響到志願者使用工作者的電腦，透過這種方式來充分利用電腦待機時的計算資源(Cycle Stealing)[12]。

根據上列特色與架構，BOINC 不適用於電腦遊戲應用的原因如下：

- 由於工作者與伺服器之間採用 Connection-less，所以工作結果回傳並不及時，無法滿足低延遲的需求。
- 由於是輪詢模式的設計架構，所以無法達到控制高動態工作，伺服器並沒有辦法直接與工作者連線並中止其工作。

2.2 XtremWeb

XtremWeb[23][8]是由巴黎第 11 大學，在 2,000 年所開發的桌機格網計算平台，它支援多項應用、多位使用者以及跨區域的佈署功能。透過區域網路以及網際網路將資源結合起來，提供高度平行計算的環境。目前被應用於高能物理、生物分子、數學與其他科學與工業領域上。

XtremWeb 架構如圖 3 所示：

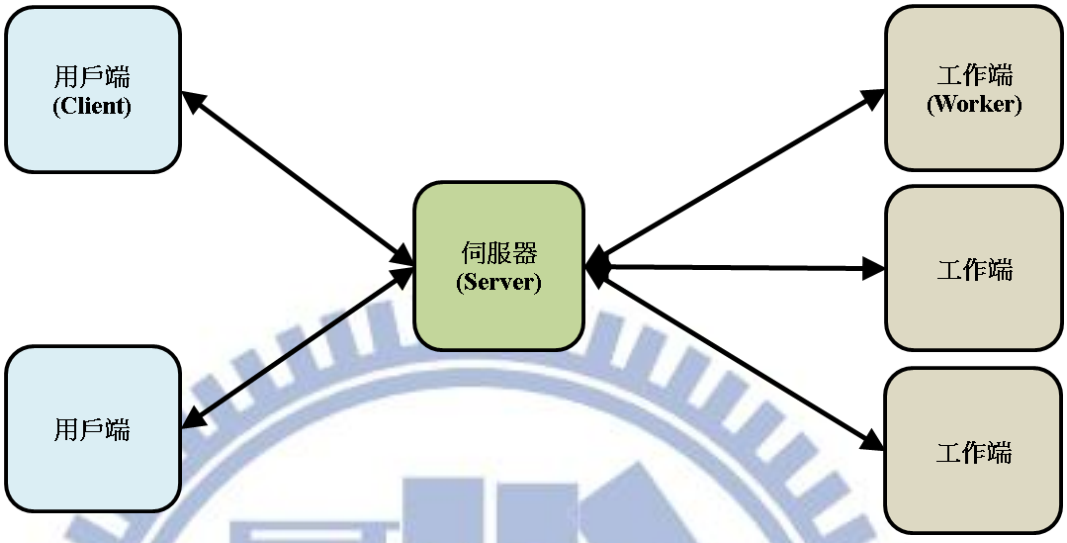


圖 3：XtremWeb 架構圖

XtremWeb 採用三層式的架構，主要由客戶端(Client)、協調者(Coordinator)，即一般的伺服器(Server)以及工作者(Worker)組成。XtremWeb 具有下列這些性質：

- 客戶端(Client)：
  - 負責發送工作給伺服器。
- 伺服器(Server)：
  - 由一組機器組成，具有較高的容錯率。
  - 負責排程與分配工作。
- 工作者(Worker)：
  - 負責執行工作。
  - 當工作者閒置時，使用輪詢模式主動通知伺服器，並且要求分配工作。
  - 工作者與伺服器之間的連線為 Connection-less。

根據上列特色與架構，XtremWeb 不適用於電腦遊戲應用的原因如下：

- 由於工作者與伺服器之間採用 Connection-less，所以工作結果回傳並不及時，無法滿足低延遲的需求。
- 由於是輪詢模式的設計架構，所以無法達到控制高動態工作，伺服器並沒有辦法直接與工作者連線並中止其工作。

2.3 CGDG

如之前 1.1.2 節所述，以下是 CGDG 的架構圖，如圖 4 所示：

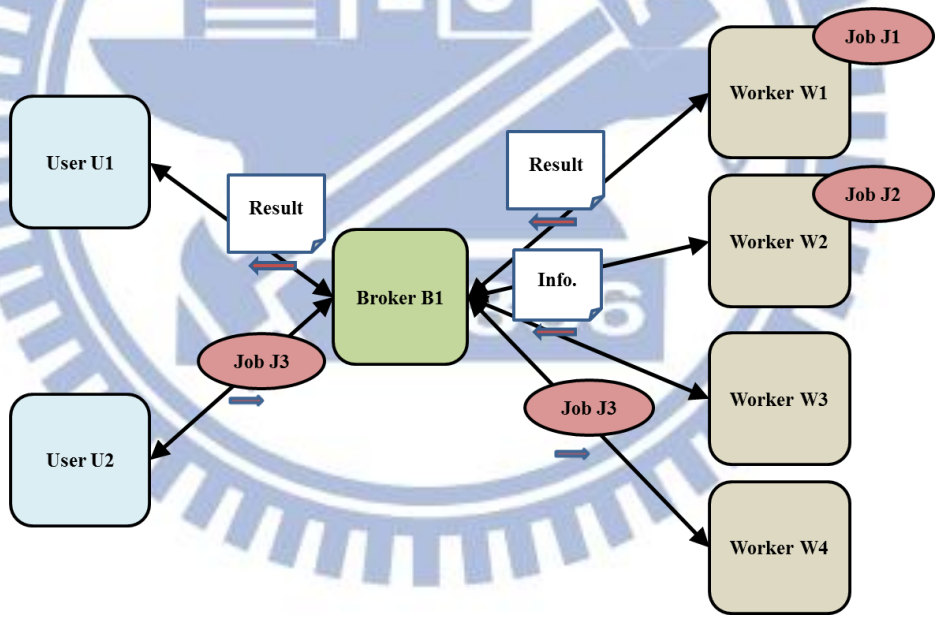


圖 4：CGDG 架構圖

本節主要介紹 CGDG 系統的特色，包含 2.3.1 節伺服器主動發派工作模式與連線導向模式、2.3.2 節容錯機制、2.3.3 節組織化、2.3.4 節工作優先度，總共四項特色。

2.3.1. 伺服器主動發派工作模式與連線導向模式

為了達到控制高動態工作之目的，系統採用伺服器主動發派工作模式(Push model)與連線導向模式(Connection-oriented model)。

因此當某個工作結束後，可以將結果立即回傳給使用者，使用者再根據工作的結果決定是否產生其他工作，或是中斷(Abort)現有的工作，達到控制高動態工作之目的，如圖 5 所示。

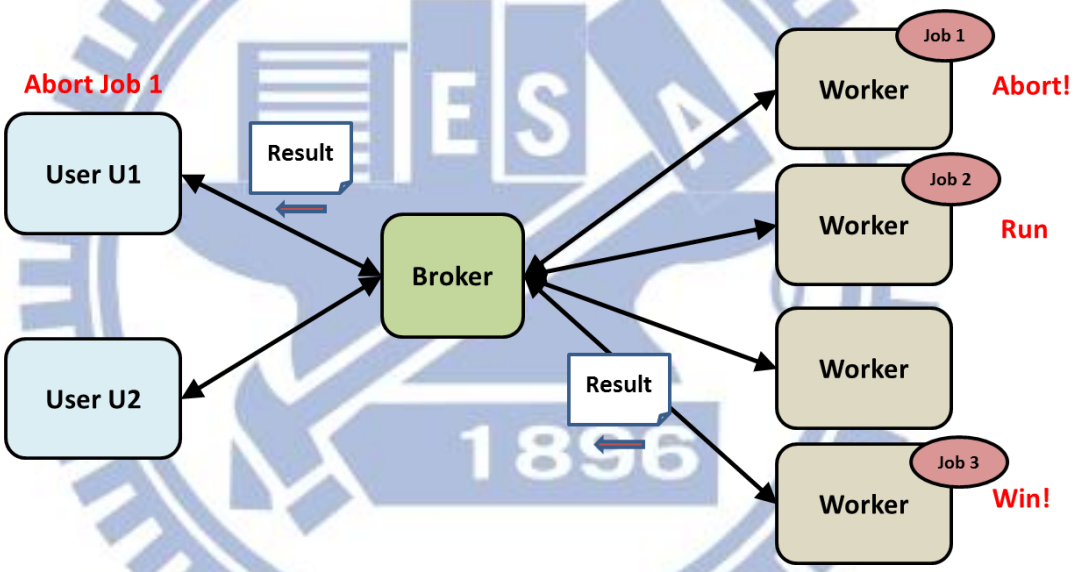


圖 5：連線導向

使用連線導向可以達到控制高動態工作之目的，如圖 5 所示。假設 User U1 分別發送了 Job J1 和 Job J3 工作到仲介者上面，且 Job J1 與 Job J3 是一樣的工作，當工作者已經做完 Job J3 工作，回傳其工作結果給仲介者，此時仲介者會把這工作結果傳送給 User U1 知道，為了提高系統的工作效率，此時 User U1 可以選擇將 Job J1 中斷(Abort)，讓運算資源釋出給其他工作使用。

連線模式不採用 Connectionless model 的原因如下：

- 由於工作者與仲介者之間並非持續連線，所以使用者無法直接將工作中斷。
- 由於使用者與仲介者之間並非持續連線，所以工作者也無法將工作立即地回傳給使用者知道。

2.3.2. 容錯機制

為了提升工作效率，系統提供了一些基本的容錯機制來處理常見的錯誤情況，例如：使用者斷線之處理、工作者斷線之處理、以及工作執行失敗之處理：

- 使用者斷線：仲介者會中斷(Abort)該使用者已送出的工作，如圖 6 所示。

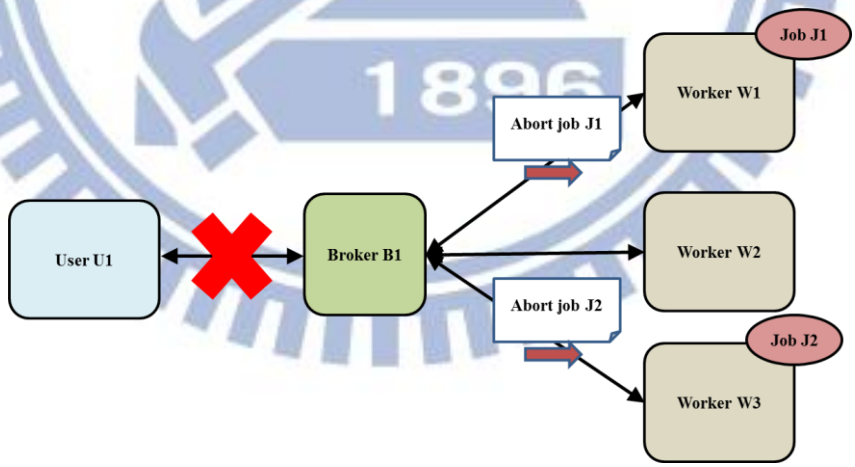


圖 6：使用者斷線圖例

- 工作者斷線：仲介者會將此工作者上的工作，重新分派(Assign)給其他工作者，系統預設每份工作重新分派之次數，以三次為上限，如圖 7 所示。

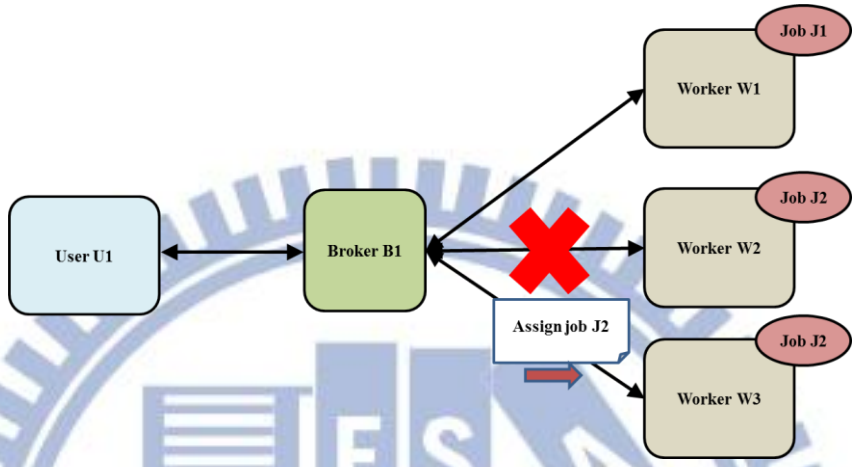


圖 7：工作者斷線圖例

- 工作執行失敗：由於程式執行所需的記憶體不足、作業系統版本不合、程式執行失敗...等錯誤發生時，仲介者會重新分派(Assign)該工作，系統預設每份工作重新分派之次數，以三次為上限，如圖 8 所示。

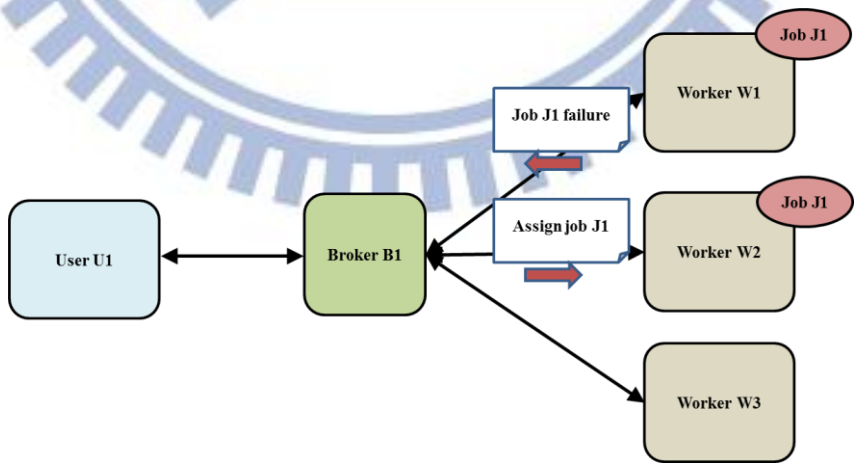


圖 8：工作執行失敗圖例

2.3.3. 組織化

為了便於管理各個組織的資源方分配問題，系統會根據組織來區分使用者與工作者，如圖 9 所示。例如：仲介者會將來自交通大學的工作者與使用者區分在一起。

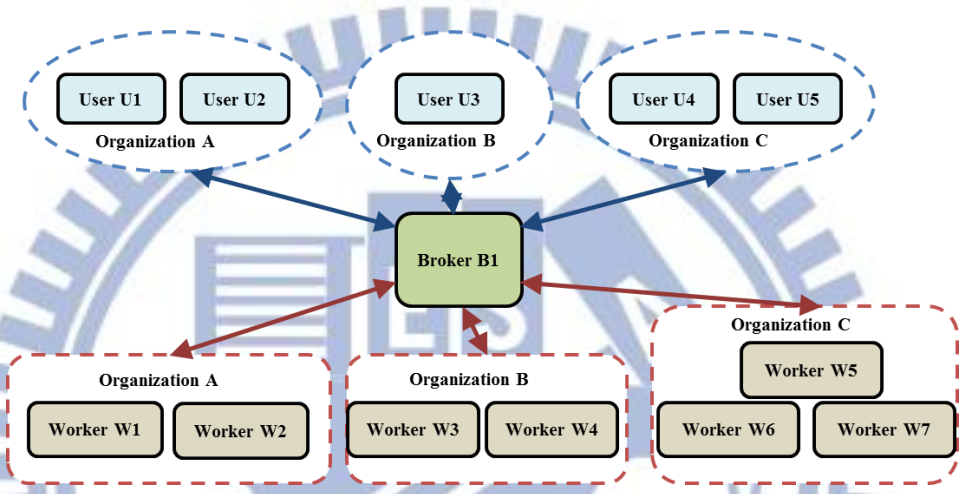


圖 9：使用者與工作者組織化

基於公平性原則，仲介者會優先將使用者發送的工作送到與使用者相同組織的工作者。如果某個組織的使用者沒有在使用該組織的工作者時，仲介者才會將其他組織的工作送到該組織的工作者上。

為了便於管理不同組織的使用者，所以給予使用者不同的身分，各個身分所擁有的權力不盡相同，如圖 10 所示：

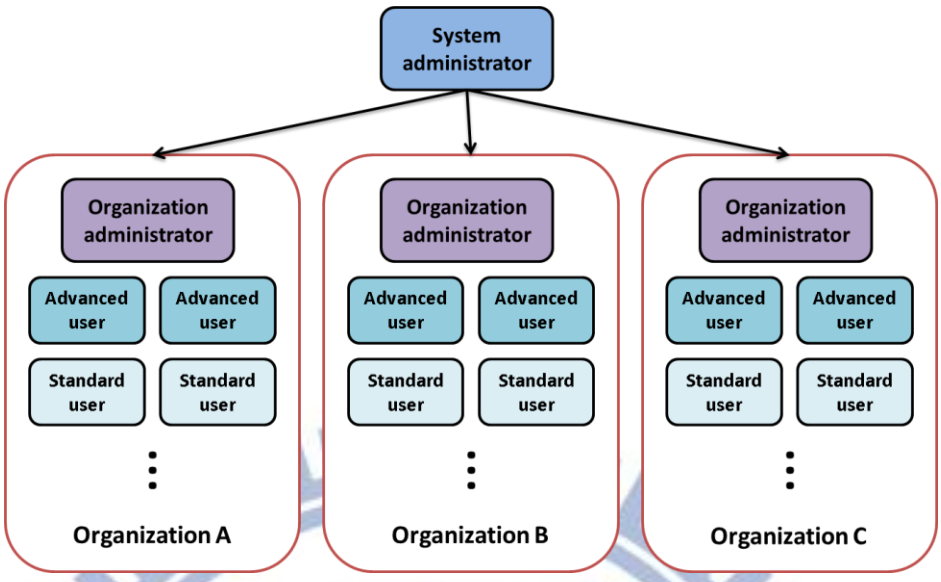


圖 10：使用者階級圖

- 系統管理者(System administrator)：擁有最高的權限，有權利創造、修改所有組織與使用者的資料，也可以創造新的組織管理者，可以發送所有優先等級的工作。
- 組織管理者(Organization administrator)：組織內部的管理者，有權力創造、修改組織內部的使用者資料，以及創造新的使用者，可以發送較高優先等級的工作。
- 高級使用者(Advanced user) ：可以發送較高優先等級的工作。
- 標準使用者(Standard user) ：可以發送一般等級的工作。

2.3.4. 工作優先度

系統將工作劃分為 Class A, B, C 三種不同等級的優先度，如表 1。並且加入組織的概念，例如：交大的使用者想要發送一個工作，該工作的等

級是 B，但此時交大的資源已經用完了，必須取用其他學校的資源(假設為中原)來處理這個工作，那麼此工作的優先度便會降低，成為等級 C 的工作，這是為了讓中原優先度為 B 的工作優先使用自有資源的設計。

表 1 ：工作優先權與組織化

Job Class	自有資源優先度	跨組織資源優先
A1	最高	最高
A2	最高	無法跨組織
B1	<A	視為 C
B2	<B1、A	視為 C
B3	<B1、B2、A	視為 C
C	<A、B	C

A1：等級A的工作有占用機器的特性，因此通常等級A的工作只能使用在自有資源上，防止使用者去佔用別人組織的資源。但需要跨組織實驗的等級時，系統也提供這個需求的工作優先度，就是A1，為了避免爭議，只有系統管理員有權利發布A1的工作。

A2：只可以使用在自有資源上的等級A，系統管理員、組織管理員以及高級使用者有權限發布 。

B1~B3：不同應用有不同的優先度，提供B等級三個優先權讓使用者設定，優先權順序B1 > B2 > B3，所有使用者皆可發布B等級的工作。但是為了防止不同組織的等級B工作去競爭跨組織資源，當等級B的工作被分配到跨組織資源上時，將視為等級C的工作。

C：最低的優先等級，所有使用者皆可發布。

第三章、研究方法

電腦遊戲桌機格網(Computer Game Desktop Grid)[17][21][19][18]，是由我們實驗室所發展的一套志願型計算格網系統，由吳毅成老師、陳靖平、詹宜智、鄒忻芸、陳昱維與陳干越學長等人所設計。

在 3.1 節會介紹學長姐先前設計的資源分配原則，在 3.2 節會介紹我新增或改良的資源分配策略。

3.1. 先前之設計

此章節將說明本系統關於資源管理之原則、資源分配之符號定義和演算法。

3.1.1 資源管理之原則

為了達到公平原則，來管理每位使用者所使用的計算資源，我們提出了下列幾項資源分配的原則[19]：

- 來自於同組織的使用者，必須公平地分享計算資源。
- 在組織 A 中，有一個工作者(Worker)工作結束後，在等待仲介者(Broker)分配新工作時，
  - 會優先將此工作者分配給該組織 A 的使用者。
  - 若該組織 A 沒有使用者需要發送工作，才會考慮分配給其他組織的使用者。

- 每個組織能獲得的計算資源(即工作者)數量與各組織所捐獻的計算資源數量呈正比，也就是該組織捐獻的計算資源量愈多，能使用的計算資源量也愈多，如此一來可以鼓勵每個組織捐獻資源，圖 11 為簡易的範例。

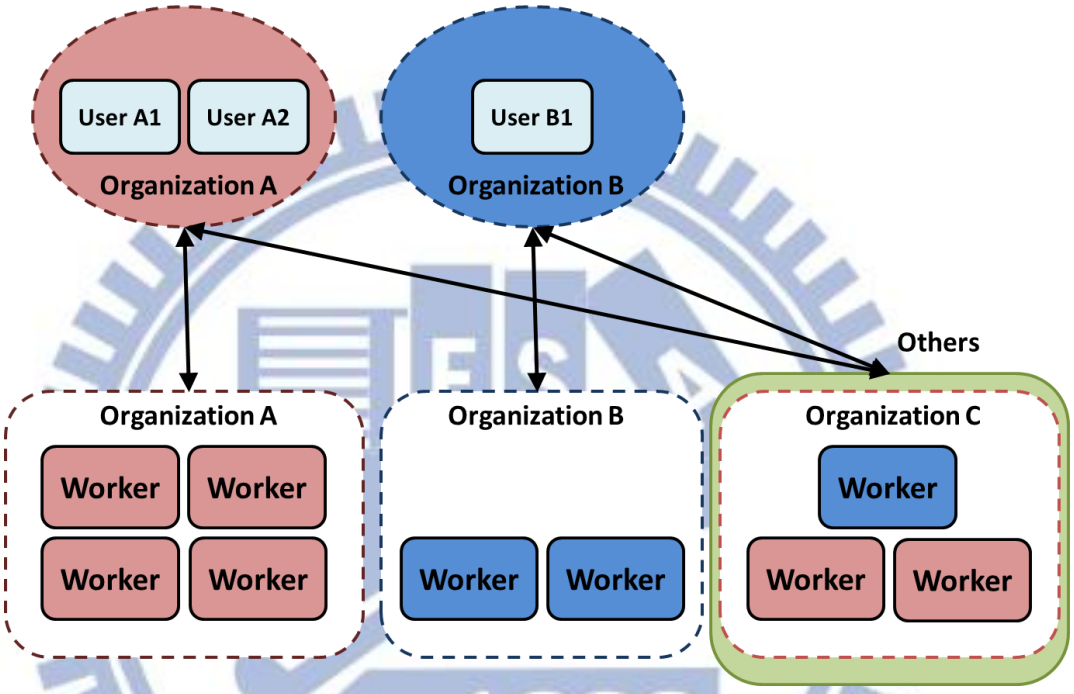


圖 11：組織與資源分配

3.1.2. 資源分配之符號定義

以下將定義一些與資源分配相關的基本符號[19]。

3.1.2.1. 工作者提供的計算量(Computing Power)

對工作者$\mathcal{W}$而言，我們定義$R_{w_i}^{\mathcal{W}}(t)$，代表在時間點 t 時，此工作者w_i所提供的計算效率。每個工作者會定期執行一個計算能力檢測程式(CPU

benchmark)，此程式可以估算出該工作者目前每秒可以執行多少 Giga 的整數計算量，我們利用此值得到$R_{w_i}^{\mathcal{W}}(t)$。

$R_{w_i}^{\mathcal{W}}(t)$使用 Giga 整數計算/每秒(Giga Integer Operations per Second, GINOPS)為單位，我們會選擇整數計算作為計算量的單位，而非選擇浮點數計算的主因為，多數的電腦對局遊戲使用大量整數計算，少數例如圍棋的蒙地卡羅(Monte carlo)演算法[5]才使用大量的浮點數計算。

對工作者$\mathcal{W}$而言，我們定義$P_{w_i}^{\mathcal{W}}(\Delta t)$代表在時間區段Δt時間內，此工作者$w_i$所提供的計算量，單位為 Giga 整數計算量(Giga Integer Operations, GINOP)。我們可以用下列公式得到$P_{w_i}^{\mathcal{W}}(\Delta t)$：

$$P_{w_i}^{\mathcal{W}}(\Delta t) = \int_{\Delta t} R_{w_i}^{\mathcal{W}}(t) \times dt \qquad \textbf{(1)}$$

我們舉一個簡單的例子，假設有一工作者(Worker) w_i的$R_{w_i}^{\mathcal{W}}(t)$ = 2.7，則此工作者兩秒內可以提供多少計算量如下：

$$P_{w_i}^{\mathcal{W}}(\Delta t) = \int_{\Delta t} R_{w_i}^{\mathcal{W}}(t)dt = 2.7 \times 2 = 5.4$$

下圖為示意圖，橘色部位代表工作者w_i所提供的計算量：

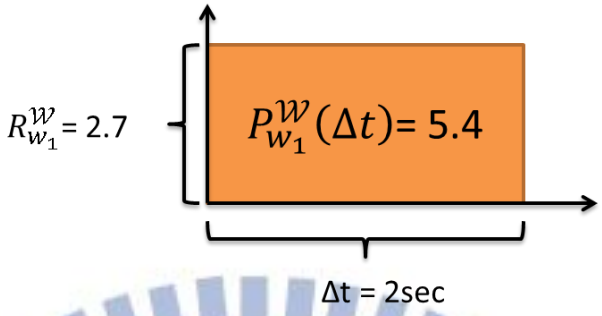


圖 12：工作者計算量圖例

3.1.2.2 使用者消耗的計算量

我們定義$w(u_i, t)$代表一個工作者集合，在這集合中的工作者都有共同的性質：在時間點 t 時，被分配給使用者u_i。

對使用者$\mathcal{U}$而言，我們定義$R^{\mathcal{U}}_{u_i}(t)$代表在時間點 t 時，此使用者計算量的消耗效率，單位為 GINOPS。以下為如何計算本符號的公式：

$$R^{\mathcal{U}}_{u_i}(t) = \sum_{w_j \in w(u_i,t)} R^{W}_{w_j}(t) \quad \textbf{(2)}$$

我們舉一簡單例子，假設現在使用者u_i分配到兩台工作者，其中一台的計算效率為 2.0，另一台為 2.7，則使用者u_i每秒所消耗的計算量如下：

$$R^{\mathcal{U}}_{u_i}(t) = 2.0 + 2.7 = 4.7$$

對使用者u_i而言，我們定義$P^{\mathcal{U}}_{u_i}(\Delta t)$代表在時間區段$\Delta t$內，此使用者消耗的計算總量，單位為 GINOP。

我們可以用下列公式得到$P_{u_i}^{\mathcal{U}}(\Delta t)$：

$$P_{u_i}^{\mathcal{U}}(\Delta t) = \sum_{w_j \in w(u_i,t)} P_{w_j}^{\mathcal{W}}(\Delta t) \quad \textbf{(3)}$$

我們承襲上一個例子，假設現在使用者u_i被分配到兩台工作者，其中一台的計算效率為 2.0，另一台為 2.7，則使用者u_i在 2 秒內所消耗掉的計算量如下：

$$P_{u_i}^{\mathcal{U}}(\Delta t) = (2.0 + 2.7) \times 2 = 9.4$$

下圖為一示意圖，橘色部位代表工作者w_1、w_2所提供的計算量：

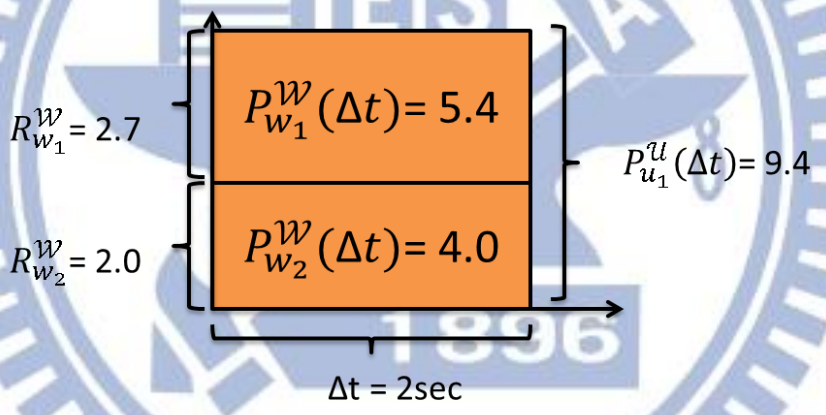


圖 13：使用者消耗之計算量圖例

3.1.2.3 配額(Quota)

我們定義$\mathcal{U}(t) = \{u_1(t), u_2(t), \ldots, u_n(t)\}$代表在時間點 t 時，有這些使用者存在於本系統。

我們定義$R_{total} = \sum_w R_{w_i}^{\mathcal{W}}(t)$代表在時間點 t 時，系統中所有工作者所

提供的計算效率。

對使用者u_i而言，我們定義$Q_{u_i}(t)$代表在時間點 t 時，系統分配給使用者的計算量配額，單位為 GINOPS，換句話說，本符號代表該使用者每秒可以使用多少計算量。理論上同組織內的所有使用者，所獲得的配額(Quota)應該相同，才能保持資源分配的公平性。假設現在系統只有一個組織，則我們可以利用以下公式求得該組織使用者的配額：

$$Q_{u_i}(t) = \frac{R_{total}}{|\mathcal{U}(t)|} \quad \textbf{(4)}$$

我們舉一簡單例子，假設現在系統總共有兩台工作者，其中一台的計算效率為 2.0，另一台為 2.7，而目前系統總共有兩名使用者，屬於同一組織，則每人所獲得的計算量配額如下：

$$Q_{u_i}(t) = (2.0 + 2.7)/2 = 2.35$$

3.1.2.4 信用值(Credit)

對使用者u_i而言，我們定義$D_{u_i}(t)$代表在時間點 t 時，其他使用者尚欠u_i多少計算量，單位為 GINOP。本符號可以說明該使用者是否超用計算資源，或是所分配到的計算資源比應得的量還要少，是用來調整使用者間資源分配公平性的主要依據。以下是如何計算信用值(Credit)的公式：

$$D_{u_i}(t') = D_{u_i}(t) + \int_t^{t'} \left(Q_{u_i}(t) - R_{u_i}^{\mathcal{U}}(t)\right) dt \quad \textbf{(5)}$$

如果有使用者超用資源，則代表有其他使用者分配到的資源量不足，

因此可以推斷出所有使用者的信用值的總和是 0。理論上信用值有守恆的定律。

3.1.3. 演算法

以下將介紹我們的演算法，分成三個部分介紹，包含主迴圈，更新所有數值以及工作分派的演算法。

3.1.3.1. 主迴圈(Main Loop)：

1. 監聽所有事件(Events)，仲介者利用select監聽所有輸入與輸出的事件，每個事件都會觸發進入此主迴圈。
2. 處理接收到的指令(Input Commands)，解析與處理來自使用者或是工作者的指令，並根據這些指令給與對應的回覆。
3. 更新所有數值，包含所有使用者、工作者、組織的資源使用量、配額與信用值等等的數值。將詳述於後文。
4. 工作分派(Job Dispatching)，決定將哪位使用者的工作分配給哪個工作端。將詳述於後文。
5. 處理送出指令(Output Commands)，將指令送給對應的使用者或是工作者。

3.1.3.2. 更新所有數值：

1 對於每個工作者(Worker) w，計算$R_w^{\mathcal{W}}(t)$與$P_w^{\mathcal{W}}(\Delta t)$，計算系統目前有多

少計算資源。

2 對於每個使用者(User) u，計算資源使用率，包含$R_u^u(t)$與$P_u^u(\Delta t)$。更新信用值$D_u(t)$，並根據目前的計算資源總量分配每位使用者下個時間區段可用的配額$Q_u(t)$。

3 對於每個組織(Organization) o，更新信用值$Q_o(t)$，並根據目前的計算資源總量分配每個組織下個時間區段可用的配額$Q_o(t)$。

3.1.3.3. 分派工作：

1. 分派優先權等級A的工作，根據使用者給予的條件選擇合適的工作者，優先分配，不須考慮資源分配的策略與限制。
2. 分派優先權等級B的工作，將每個組織的資源分配給該組織的使用者，根據資源分配的策略決定分派與否，最後若仍有剩餘的計算資源，則分配給資源使用率$R_u^u(t)$最低的使用者。(資源分配策略將詳述於後文)
3. 分派優先權等級 C 的工作，將剩餘的計算資源分配給跨組織的使用者，也必須遵守資源分配的策略。

3.2. 新增或改良之設計

此章節將針對資源分配策略提出新增或改良之方法。

3.2.1 資源分配基本策略

根據 3.1.1 小節提到的資源分配原則，以及 3.1.2 小節所定義的各項符號，我們可以設計出幾項資源分配的基本策略，來解決的資源分配議題。這裡要注意的是，以下提出的策略只適用於除了發送優先權 A 工作之外的使用者。

1. 每一位使用者須獲得一定的量的計算資源，稱為基本配額(Basic Quota)。基本配額的量是配額(Quota)的一半，可以解決避免飢餓的議題。
2. 對於使用者u，若u上傳了一個工作至仲介者(Broker)，仲介者會根據該使用者的信用值(Credit)來決定是否幫該使用者分配工作者來執行此工作，若此使用者的信用值低於全部使用者信用值的平均值，仍可繼續獲得計算資源，直到此使用者佔有的計算資源等於配額為止，才停止為此使用者分派工作(即確保$R_u^u(t) \leq Q_u(t)$)，可以解決公平性及平衡性的議題。
3. 對於使用者 u，若 u 上傳了一個工作至仲介者，若此使用者的信用值高於全部使用者信用值的平均值，則仲介者在幫此使用者分配此工作時，會先檢查若分配資源給該使用者是否會導致該使用者的信用值降至平均值之下，我們可以利用事先得到每個工作的最大執行時間來預估每個工作還需要多久時間結束計算，藉此來得到這個使用者在釋放多餘

的資源時還需消耗多少計算量，若此使用者的信用值不夠支付這些計算量的話，則不幫此使用者分派工作，可以預防長工作取得資源後久久不釋放導致資源分配不平衡的問題 。

3.2.2 組織之間的公平性

此章節將介紹組織的配額(Quota)、組織的信用值(Credit)以及設計中所遇到的議題，最後是組織之間的資源分配策略。

3.2.2.1 組織的配額(Quota)

我們定義$O(t) = \{o_1(t), o_2(t), \dots o_n(t)\}$代表在時間點 t 時，有這些組織存在於本系統。

我們定義$R_{total} = \sum_w R_{w_i}^{W}(t)$代表在時間點 t 時，系統中所有工作者所提供的計算效率。

對組織o_i而言，我們定義$Q_{o_i}(t)$代表在時間點 t 時，系統分配給組織的計算量配額，單位為 GINOPS，換句話說，本符號代表該組織內的所有使用者每秒可以使用多少計算量。組織的計算量配額概念與使用者的計算量配額相似，在 3.1.1 小節我們有提到資源分配的其中一項原則：每個組織能獲得的計算資源與捐贈的資源成正比。

另外，我們定義每個組織的權重為Θ_{o_i}，所以我們可以根據組織的計算資源比例求得組織的配額。

$$Q_{o_i}(t) = R_{total} * \frac{\Theta_{o_i}}{\sum_{o_j \in \mathcal{O}(t)} \Theta_{o_j}} \quad \textbf{(6)}$$

我們舉一簡單例子，假設現在系統總共有兩個組織 A、B，捐獻工作者數量的比例為2: 1，如圖 14 所示，$P_{total} = 10.4$，則各個組織所獲得的計算量配額如下：

$$Q_{O_A}(t) = 10.2 * \frac{2}{2+1} = 6.8$$

$$Q_{O_B}(t) = 10.2 * \frac{1}{2+1} = 3.4$$

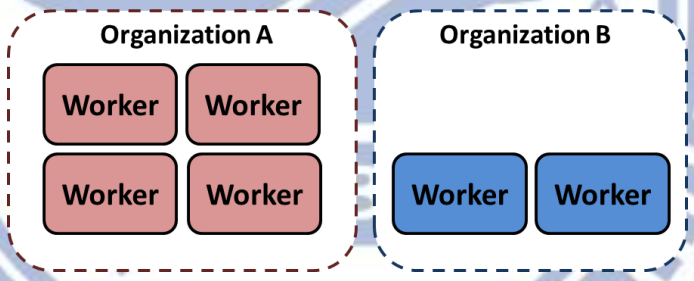


圖 14：組織捐獻比例圖

3.2.2.2 組織的信用值(Credit)

對組織o_i而言，我們定義$D_{o_i}(t)$代表在時間點 t 時，其他組織尚欠o_i多少計算量，單位為 GINOP。本符號可以說明該組織是否超用計算資源，或是所分配到的計算資源比應得的量還要少，是用來調整組織間資源分配公平性的主要依據。

以下是如何計算組織信用值的公式：

$$D_{o_i}(t') = D_{o_i}(t) + \int_t^{t'} \left(Q_{o_i}(t) - \sum_{u_j \in o_i} R_{u_j}^{u}(t) \right) dt \qquad (7)$$

3.2.2.3 權重信用值的議題

我們舉一個例子來敘述此議題，假設系統目前有四個組織 A、B、C 和 D，其中組織 A、B、C 的機器捐獻比例為1:2:3，由於組織 D 沒在使用組織 D 的機器，所以組織 A、B、C 會以1:2:3的比例去使用組織 D 的機器，如圖 15 所示。

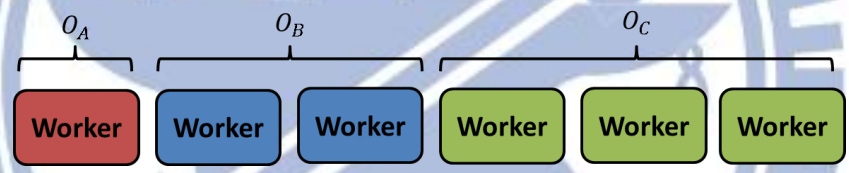


圖 15：依比例使用閒置的機器

如果某時刻組織 A 多用一核、組織 C 少用一核，如圖 16 所示，那麼組織信用值的計算是否要相乘權重，以凸顯各個組織所捐獻的機器數量呢？

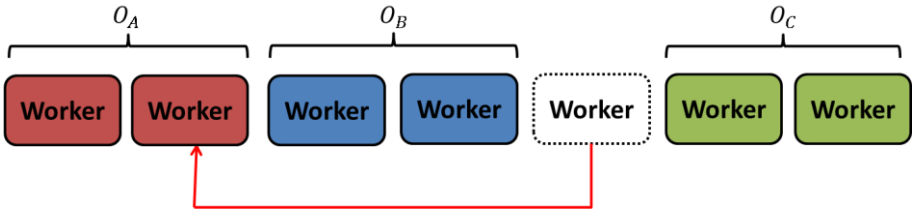


圖 16：組織 A 多用一核、組織 C 少用一核

根據信用值守恆定律，答案是：組織信用值不用相乘權重。以圖 16 為例，組織 A 多用一核的計算量和組織 C 少用一核的計算量，不論乘與比例權重或比例權重之倒數，都無法使所有組織信用值相加為零。因此，組織信用值不相乘權重，才能使得信用值守恆。

3.2.2.4 組織之間的資源分配策略

我們定義$\overline{D_o}(t)$為在時間點 t 時，全部組織信用值的平均值。

我們定義$\Delta D_{o_i}(t)$為在時間點 t 時，某組織的信用值和全部組織信用值的平均值之差，即$\Delta D_{o_i}(t) = D_{o_i}(t) - \overline{D_o}(t)$。

如果某一個使用者$User_{A1}$要使用其他組織的某一台機器，必須全部符合以下條件，才能使用其他組織的資源：

1. 該使用者的組織信用值大於全部組織信用值的平均值，
   即$D_{O_A}(t) > \overline{D_o}(t)$。

2. 該機器所屬的組織，其組織信用值小於全部組織信用值的平均值，
   即$D_{O_{other}}(t) < \overline{D_o}(t)$。

3. 透過 3.1.1 之第 3 點計算該使用者的預估超用量，其值要小於或等於該機器所屬組織的信用值和全部組織信用值之平均值差值的量，
   即$(預估超用量_{A1}) \leq \left|D_{O_{other}}(t) - average\ credit\right|$，確保被搶資源的組織有足夠的信用值能償還使用者 A1 的計算量。

4. 保留一些資源給被搶資源的組織，避免該組織陷入飢餓的議題。

3.2.3 多執行緒工作的公平性

由於系統以往都是分配資源給單核的工作,所以我們希望將系統擴充:能夠公平地分配資源給多核的工作。

此章節將介紹某些情境下會遇到的問題，並且提出解決方法，最後是多執行緒工作的資源分配策略。

3.2.3.1 情境

由於每份工作所使用的核心數量不盡相同，造成現有的空閒核心數量(例如：兩核)小於一份多執行緒工作所需要的核心數量(例如：三核)。

3.2.3.2 解決方法

最直觀的解決方法：將那些無法被多執行緒工作所使用的空閒核心數閒置著，直到閒置核心數量能夠被多執行緒工作所使用。但是該方法所產生的問題是：

1. 浪費資源，亦即沒人使用空閒的核心數。

2. 等待蒐集閒置核心數量的時間，可能會很久。

因此我們提出以下的方法來解決 3.2.3.1 節的問題，如果目前空閒核心數量(例如：兩核)小於一份多執行緒工作所需要的核心數量(例如：三核)：

1. 仍然將現有的空閒核心數量，分配給該份多執行緒工作。

2. 若有多餘的空閒核心，再逐漸地分配給該份多執行緒工作。

如此一來，就不會浪費資源，而且多執行緒工作也無須一直等待空閒核心數量，避免該工作陷入飢餓問題。

3.2.3.3 多執行緒工作的資源分配策略

每次要分配資源給使用者 A 的新工作時，先檢查 A 正在使用資源的工作中，其中某份工作的使用核心數量是否能擴充，擴充的條件是：透過 3.2.1 節之第 3 點來計算 A 的信用值是否能夠支付 A 的預估超用量，如果能支付就讓該工作擴充，透過 JSETCORE 指令擴充(附錄 B)，如圖 17 和圖 18。

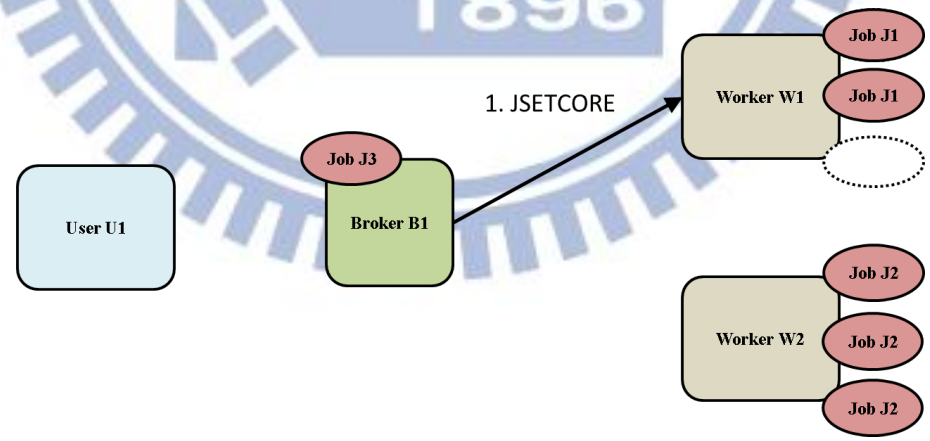


圖 17：擴充某工作的使用核心數(1)

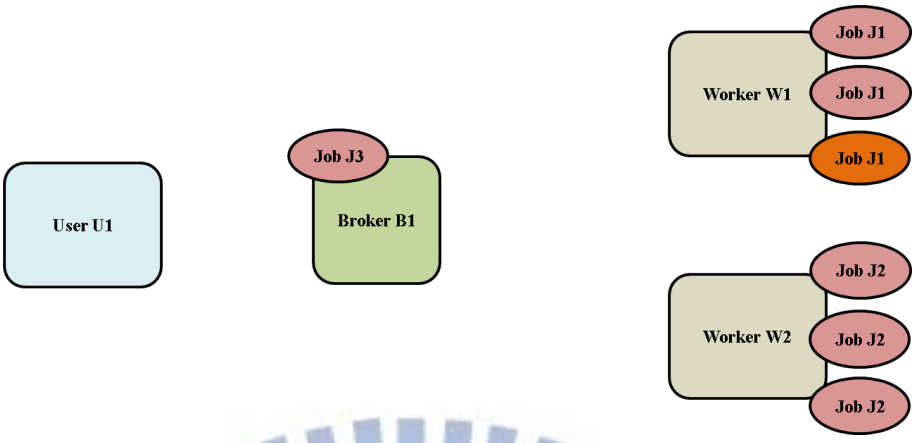


圖 18：擴充某工作的使用核心數(2)

如果 A 正在使用資源的工作中，沒工作能夠擴充使用核心數，若下列條件全部符合，即使空閒核心數小於這份新工作使用的最大核心數，也會將資源分配給這份新工作：

1. 這份空閒資源所屬的機器，正在處理的所有工作和這份新工作，都能擴充到最大使用核心數(擴充的條件同上，所有使用者的信用值都能支付預估超用量)。確保所有工作都能擴充到最大使用核心數。

2. 符合 3.2.1 節所敘述的資源分配基本策略。

3.2.4 資源使用效能

此章節會先介紹議題，接著論證其議題，並提出解決方法，最後是資源分配策略。

3.2.4.1. 議題

使用者使用資源的平衡性和穩定性，會影響使用者工作的整體效能。

3.2.4.2. 論證議題

我們定義T_n代表某程式使用 n 核心，總共需時T_n。

我們定義$S_n = \frac{T_1}{T_n}$，其中S_n代表某程式使用 n 核心的加速倍率(Speedup)。

我們定義$E_n = \frac{S_n}{n} = \frac{T_1}{nT_n}$，其中$E_n$代表某程式使用 n 核心的效能(Efficiency)。因此，我們得知加速倍率和效能有其關係。

在程式使用越多核心數量時，加速倍率的增益會遞減，如圖 19 所示。例如：六子棋驗證系統演算法(Job-Level Proof-Number Search)[22]。

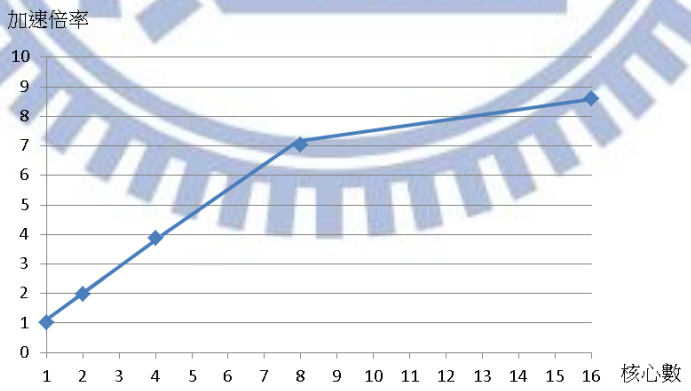


圖 19：加速倍率圖例

為了簡化，我們將S_n認定為連續函式，標示為$S(n)$，其中$S(n) = S_n$，對於所有整數 n，用多項式插入法(Polynomial interpolation)求得其他值，如圖 20 所示。

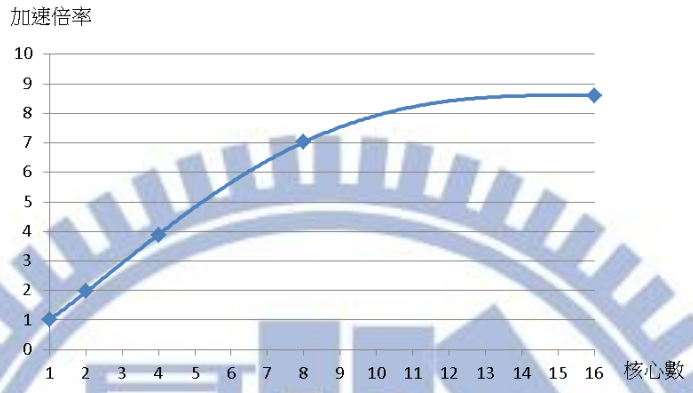


圖 20：加速倍率連續函式圖例

我們針對加速倍率做以下假設：

1. 加速倍率連續函式的微分，$S'(n)$，亦是連續函式。

2. 若 $n_1 < n_2$，則$S_{n_1} < S_{n_2}$。直覺上，使用越多核心數量，加速倍率越大。同時，這能推導出$S'(n) > 0$。

3. $S'(n)$是遞減函式。為了要同步(synchronize)工作之間的結果，而產生額外的執行緒花費(overhead)，造成使用越多核心數量，加速倍率的增益會遞減。

4. 根據以上第 2、3 點能歸納出：$\left(wS_{n_1} + (1-w)S_{n_2}\right) \leq S_{wn_1+(1-w)n_2}$，其中 $0 < w < 1$。當$w = {}^1/_2$時，$\frac{S_{n_1}+S_{n_2}}{2} \leq S_{\frac{n_1+n_2}{2}}$，如圖 21 所示。

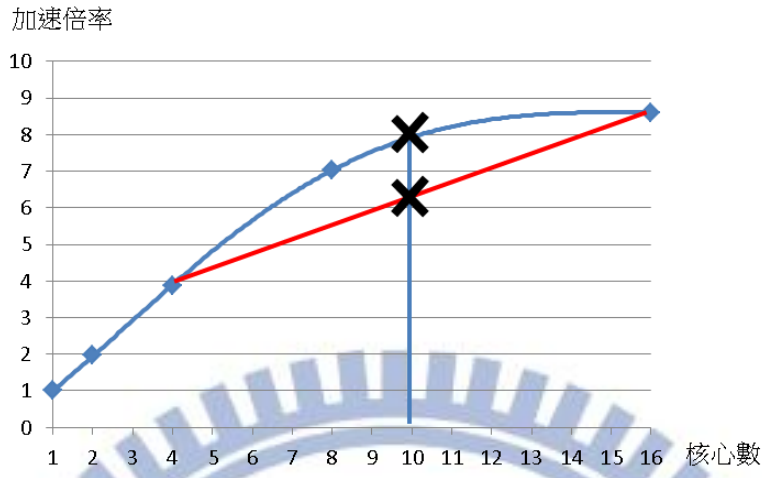


圖 21：加速倍率的歸納假設

由加速倍率公式得知：

$$S_n = \frac{T_1}{T_n} \rightarrow \frac{S_n * T_n}{T_1} = 1 \quad \textbf{(8)}$$

我們先舉個例子，來做一些假設，如圖 22 所示：

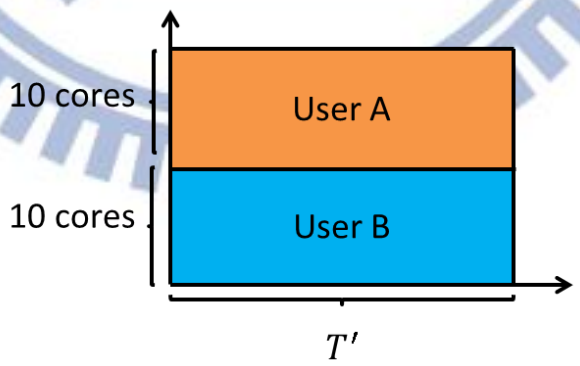


圖 22：論證圖例(一)

根據T_n定義，T_{10}代表某程式使用 10 核心去完成一份工作，總共需時T_{10}，

再根據公式(8)，我們推得$T_{10} \to \frac{S_{10} * T_{10}}{T_1} = 100\%$。

以圖 22 為例，定義T′代表目前花費的總時間，再根據公式(8)，我們推得$T' \to \frac{S_{10} * T'}{T_1} = R$，因此我們定義$R$代表目前完成的工作比例(complete ratio)，如果完成工作比例要 100%，那麼$T' = T_{10}$：

$$R_n = \frac{S_n * T'}{T_1} \quad \textbf{(9)}$$

讓我們舉兩個情況來比較，都有兩個使用者 A、B，資源都有 20 核心，目前花費的總時間都為T′，情況一：A、B 分別平衡、穩定使用 10 核心。情況二：總時間的一半中，A 使用 5 核、B 使用 15 核，剩下的一半時間中，A 使用 15 核、B 使用 5 核，如圖 23 所示。

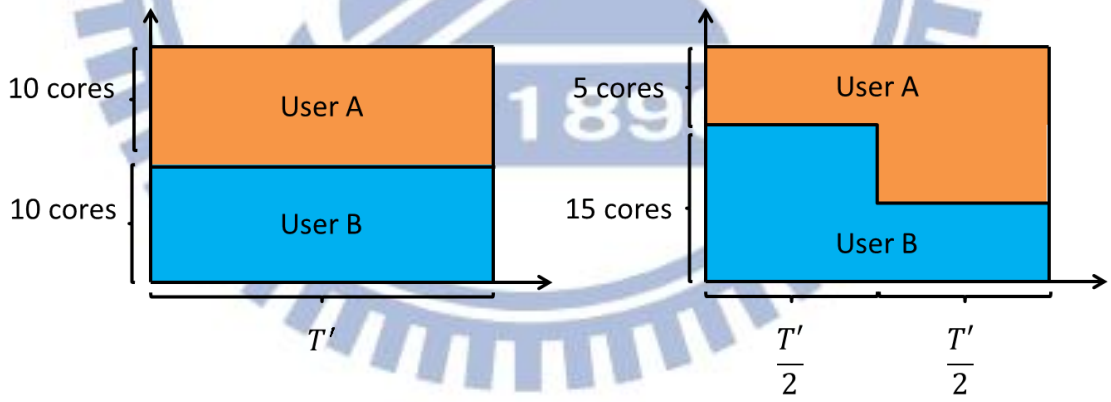


圖 23：論證圖例(二)

上述兩個情況中，我們要比較使用者 A，在哪個情況中的工作完成比例較高，根據**公式(9)**，我們得知：

情況一：

$$R_{10} = \frac{S_{10} * T'}{T_1} = \frac{T'}{T_1} * S_{10} \tag{10}$$

情況二：

$$R_{(5+15)} = \frac{S_5 * \frac{T'}{2}}{T_1} + \frac{S_{15} * \frac{T'}{2}}{T_1} = \frac{T'}{T_1} * \left(\frac{S_5 + S_{15}}{2}\right) \tag{11}$$

根據之前假設中的第 4 點，我們推得：

$$S_{10} \geq \frac{S_5 + S_{15}}{2} \tag{12}$$

由**公式(10)、(11)、(12)**，我們得知情況一的工作完成比例較高，意即加速倍率較高，也就是效能較高，所以得出以下結論：

使用者使用資源的平衡性和穩定性越高，工作的整體效能就越高。

3.2.4.3. 解決方法

我們定義$\Delta QR_u(t)$為在時間點 t 時，系統分配給某使用者的計算量配額和此使用者計算量的消耗效率之差，即$\Delta QR_u(t) = Q_u(t) - R_u^u(t)$。

為了使用者之間的資源使用平衡和穩定性，在公平性原則之下，盡可能地減少$\Delta QR_u(t)$。

3.2.4.4. 資源使用效能的資源分配策略

每次要分配資源給使用者 A 的新工作時，先透過 3.2.1 節中第 3 點檢查 A 的信運值是否能支付預估超用量，如果能支付預估超用量，再繼續判斷以下的新策略：

找出其他使用者中，一個同樣能支付預估超用量，且信用值最高的人，如果找到該使用者，那麼 A 當下使用的消耗計算量，不能超過該使用者，意即$R_A^u(t) \leq R_{other}^u(t)$。

如果 A 的信用值最高，找出所有其他使用者，能支付預估超用量，且信用值大於平均值，但信用值小於 A 的信用值，那麼 A 只能比這些人多用一核計算量，意即$R_A^u(t) \leq R_{other}^u(t) + 1$ 核計算量。

如果要把此策略套用在組織之間，只要將計算使用者的消耗計算量相乘權重即可。

3.2.5 演算法

針對 3.1.3.3 節提到的第 2、3 點中的資源分配策略做改良或新增策略。完整資源分配策略如下：

每次要分配資源給同組織內使用者的新工作時：

1. 透過 3.2.3.3 節的方法判斷該使用者是否有工作能擴充核心數，如果有工作擴充核心數量，就跳回 3.1.3.3 節第 2 點繼續分配資源給同組織其

他使用者。

2. 如果沒工作能擴充核心數量，就採用 3.2.1 節的策略。

3. 如果該使用者的新工作通過 3.2.1 節的第 3 點的策略，就要額外採用 3.2.4.4 的策略，確保能夠穩定使用核心數量，以達到高效能。

每次要分配資源給不同組織使用者的新工作時：

1. 至 3. 點和分配資源給同組織的相同，最後要分配出去前，多做第 4 點判斷。

4. 透過 3.2.2.4 節策略，確認別的組織是否具有足夠的信用值能償還不同組織使用者的預估超用量。

第四章、系統實作

在此章，首先是實驗說明，再來是實驗結果。

4.1. 實驗說明

我們用 3.2.5 節所提出的資源分配策略來進行模擬，並將每個使用者的資源使用率隨時間做紀錄，並以區域圖的方式呈現，可以很容易的看出是否有資源分配不公平或是不平衡的現象。

我們利用模擬的使用者程式連續不斷地發送工作，工作類型分為執行時間較長(約 10 分鐘)的工作，之後簡稱長工作，與執行時間較短(約 20 秒)的工作，之後簡稱短工作，這些工作並不會使用到大量的 CPU，只是簡單的執行睡眠(Sleep)來模擬執行時間，因此我們在實驗中並不討論真實計算量的議題。

在此次實驗中，我們使用來模擬的工作者實體配備如下：

- CPU: Intel Xeon E5620, 2.40GHz, 16 cores
- RAM: 66.1GB

4.2. 實驗結果

將會依序說明新增或改良之資源分配策略的實驗結果。

4.2.1. 組織之間的公平性

我們利用一台實體機器，模擬 48 核計算資源，總共有三位使用者 A、B、C，分別屬於不同組織，他們組織的機器捐獻比例為 1：2：3，由兩名使用者 A、B 都發送短工作，並在中途加入另一個短工作的使用者 C。

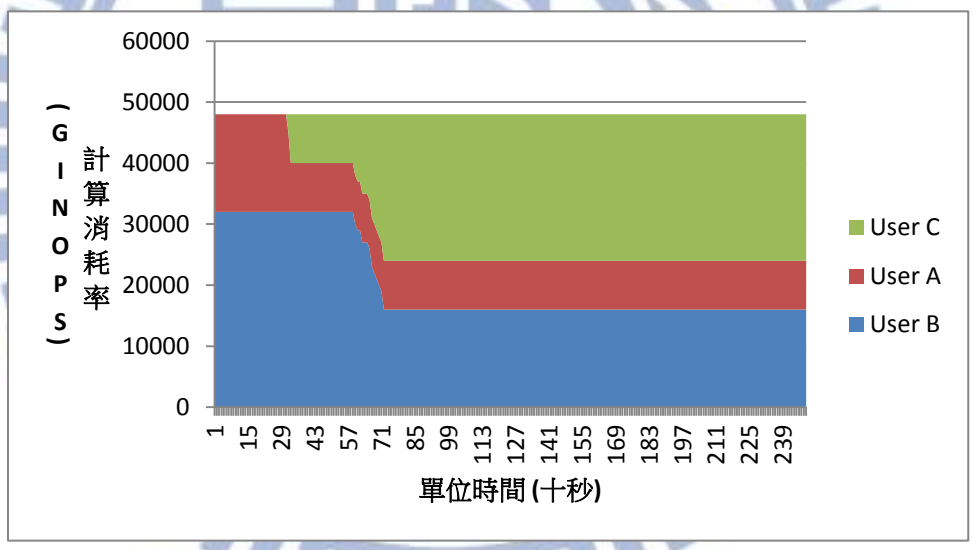


圖 24：資源分配基本策略(組織間)

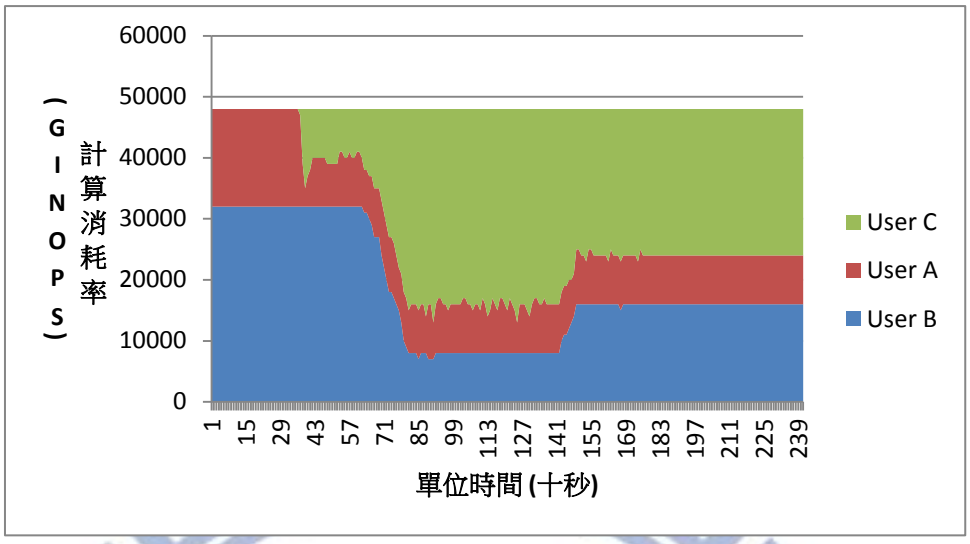


圖 25：資源分配基本策略+組織信用值計算(組織間)

因為使用者 B 分派長工作，造成使用者 C 中途加入時，無法立即獲得資源，必須等待 B 慢慢釋放出資源，C 才可以使用。

圖 24 由於沒計算組織信用值，所以使用者 B 並未償還資源給使用者 A、C，而圖 25 有計算組織信用值，所以使用者 B 在釋放資源的途中，會慢慢還資源給 A、C，之後會再恢復平衡，也就是機器捐獻比例 1：2：3。

4.2.2. 多執行緒工作的公平性

我們利用一台實體機器，模擬 16 核計算資源，總共有二位使用者 A、B，A 分派需要三核心的工作，該工作從個位數 1 計數到第 34 高位元的 1，B 分派需要一核心的短工作，藉此來觀察資源使用狀況，結果如下：

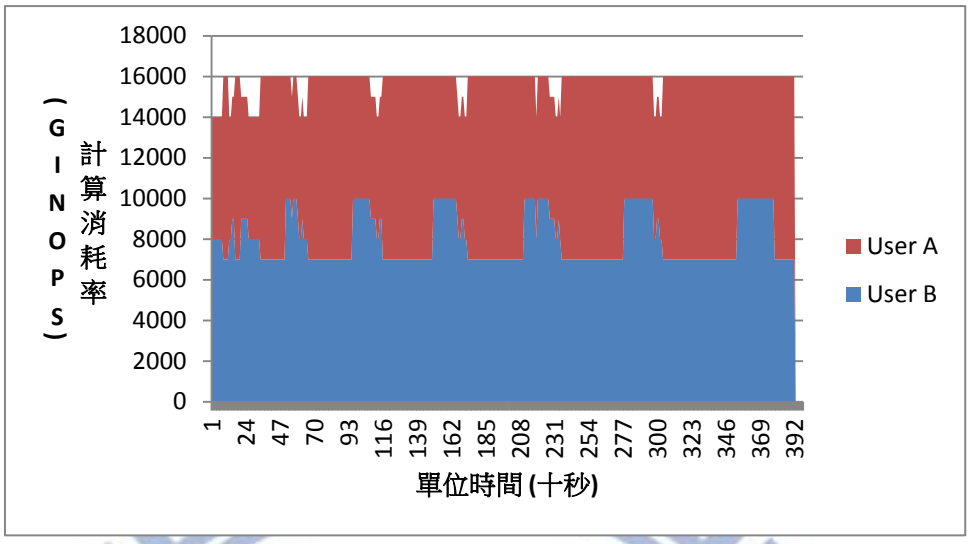


圖 26：部分資源閒置

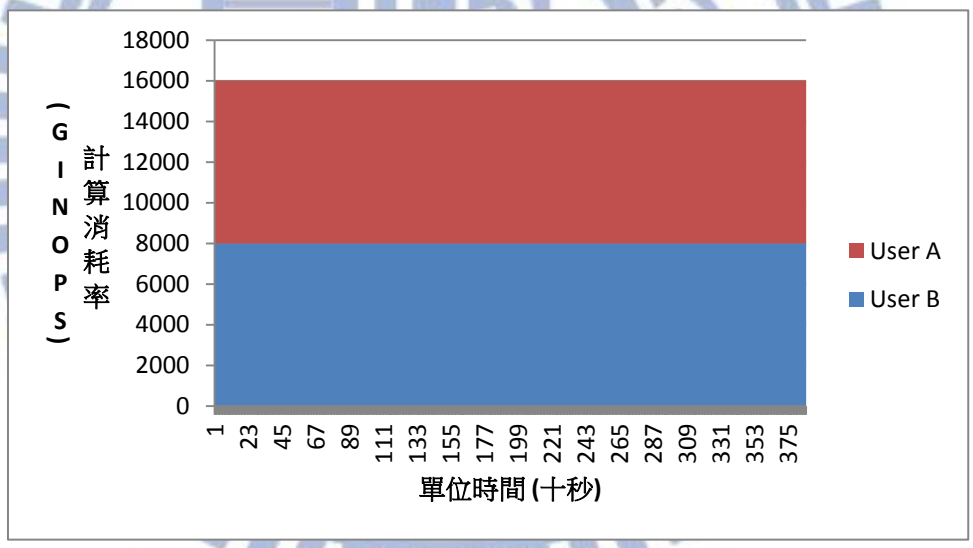


圖 27：資源充分使用

圖 26 可看出有部分的資源閒置，因為空閒的核心數量不足被 A 分派的工作所使用，並且 A 會認為其他使用者欠他資源，所以之後會把資源搶回來使用，但是實際上，B 並沒有超用資源量，所以對 B 來說，A 搶 B 的資源是不公平的。

圖 27 系統能充分使用資源。雖然 A 使用較少的核心數量，從原本工作費時 40 秒，變為平均費時 55 秒，但是系統使用率卻提升不少。

4.2.3. 資源使用效能

先用實驗驗證議題，再運用新增或改良的資源分配策略來實驗。

4.2.3.1. 驗證議題

使用六子棋驗證系統演算法(Job-Level Proof-Number Search)[22]來驗證議題：使用者使用資源的平衡性和穩定性越高，工作的整體效能越高。結果如下：

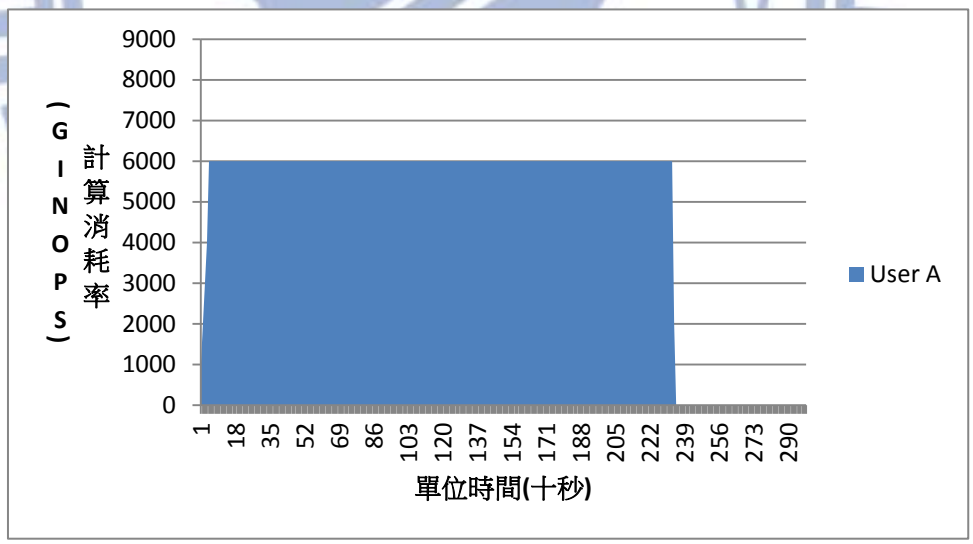


圖 28：穩定使用六核心

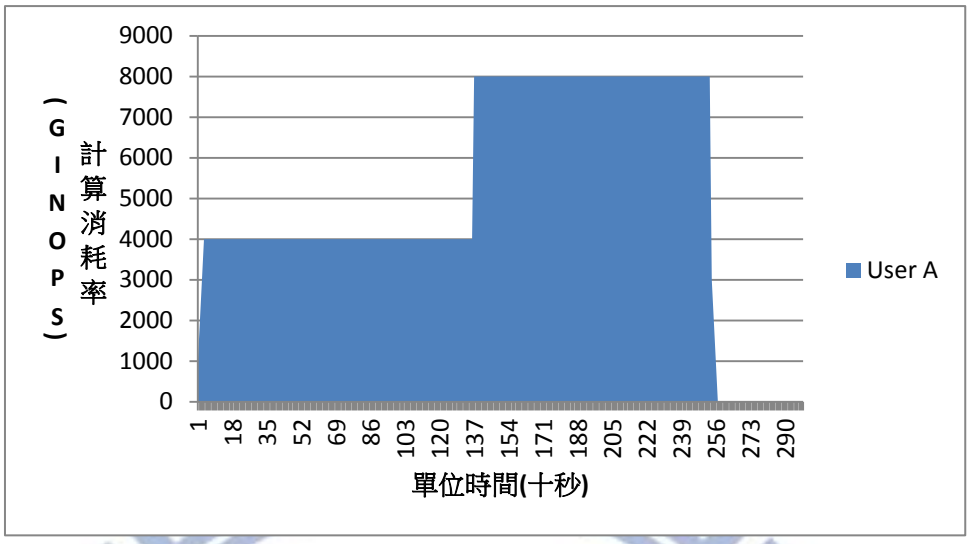


圖 29：總時間的一半用四核、另一半用八核

由圖 28 得知，穩定使用六核心，總共需要 2344 秒；由圖 29 得知，一半時間使用四核，另一半時間使用八核，總共需要 2562 秒，由此可以看出穩定地使用核心數，有較好的工作效能，下面列出較為詳細的數據：

核心數	4	4、8 混用	6	8
加速倍率	2.91	4.32	4.72	5.37

表 2：核心數與加速倍率

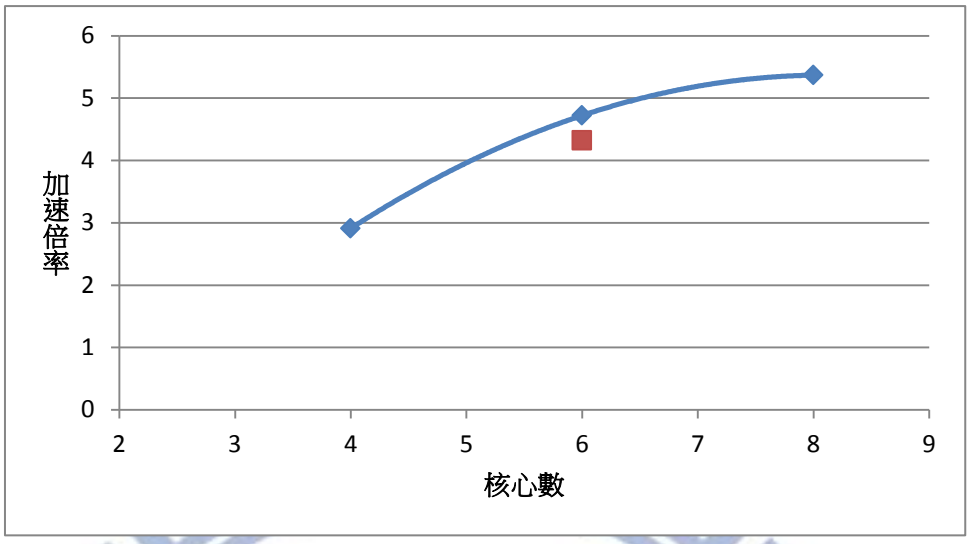


圖 30：核心數與加速倍率

從圖 30 得知，穩定使用 6 核心的加速倍率，大於 4、8 核混用的加速倍率，因此可以佐證推論。

4.2.3.2. 組織內

我們利用一台實體機器，模擬 48 核計算資源，總共有兩位使用者 A、B，一開始由 B 發送短工作，中途 A 再加入發送短工作，結果如下：

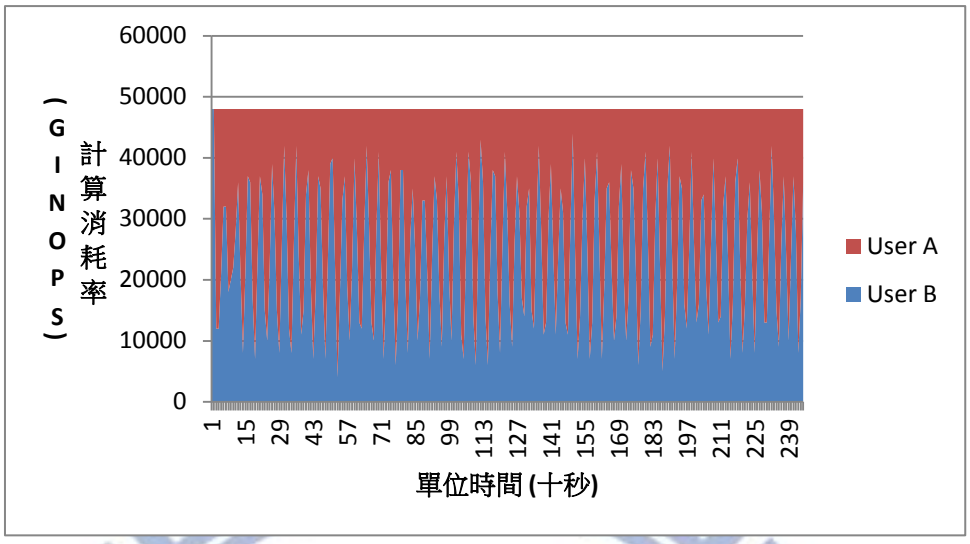


圖 31：資源分配基本策略(只用 1、2 項)

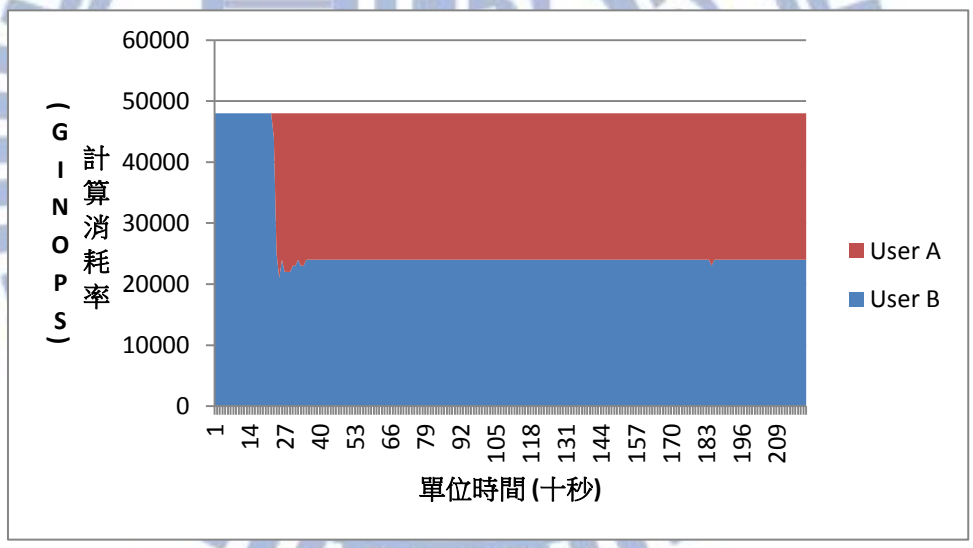


圖 32：資源分配基本策略(3 項全用)

圖 31 沒使用預估超用量策略，造成信用值高的人，會超用太多資源，使得該使用者的信用值會降至平均之下，發生使用核心數不平衡現象，由圖 32 可明顯看出預估超用量對平衡性有很大的幫助。

即使有這樣的設計，但是在某些情況下，還是會不平衡，如下圖所示：

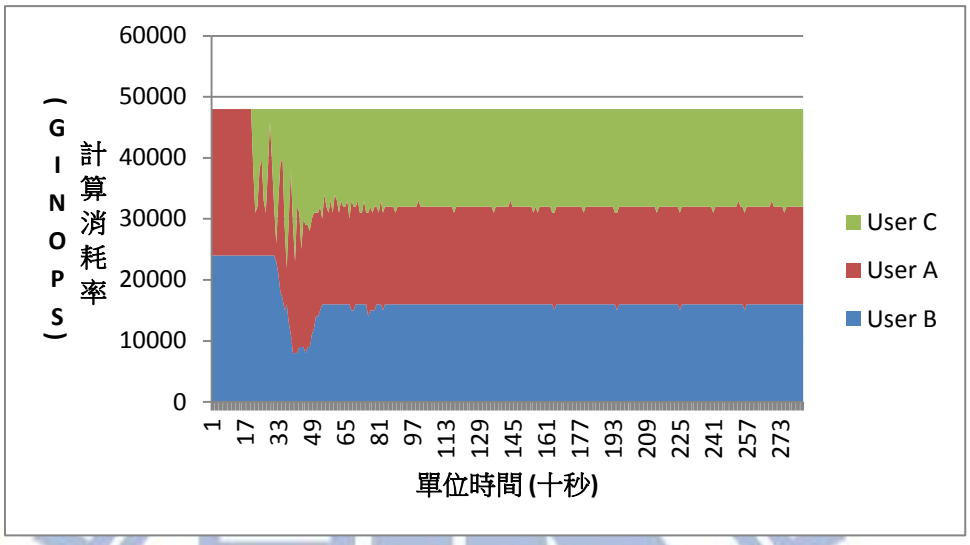


圖 33：使用預估超用量，還是有不平衡之現象

圖 33 的情況是同組織之內，有三位使用者 A、B、C，一開始由 A 發送短工作、B 發送長工作，中途 C 加入發送短工作。由於 C 中途加入，造成 B 積欠資源太多，使得 A、C 的信用值都能支付他們的預估超用量，演變成 A、C 亂搶資源，再加上沒策略規範搶資源的上限，所以 A、C 的資源使用量就不平衡。

根據 3.2.4.4 節的新增策略，用以規範使用者的搶資源上限，使得彼此之間的資源使用量能夠平衡、穩定，結果如下：

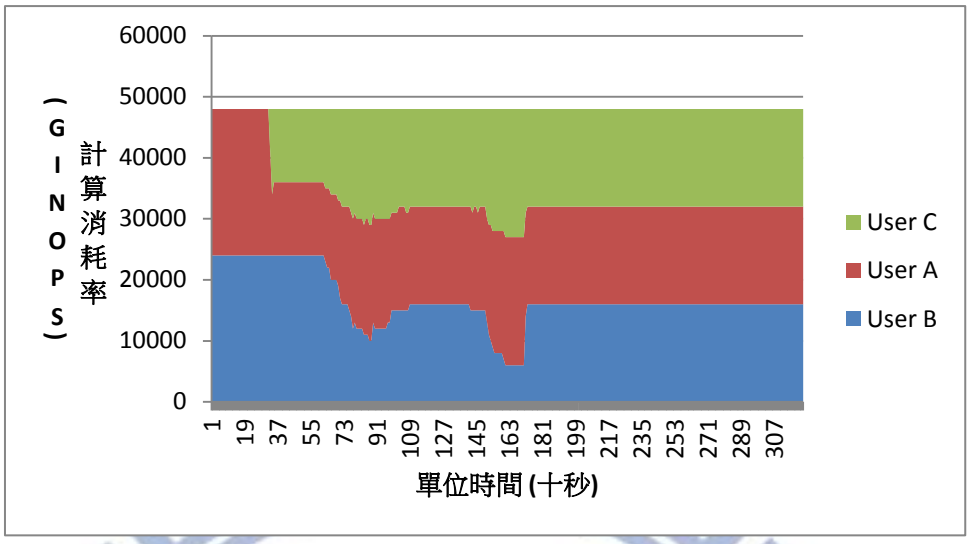


圖 34：規範使用者的搶資源上限

4.2.3.3. 組織之間

在 3.2.4.4 節提出規範使用者的搶資源上限之策略，在組織之間亦能適用(需搭配 3.2.2.4 節計算組織信用值)。

總共有三位使用者 A、B、C，分別屬於不同組織，他們組織的機器捐獻比例為 1：2：3，一開始由 A 發送短工作、B 發送長工作，中途 C 加入發送短工作，結果如下：

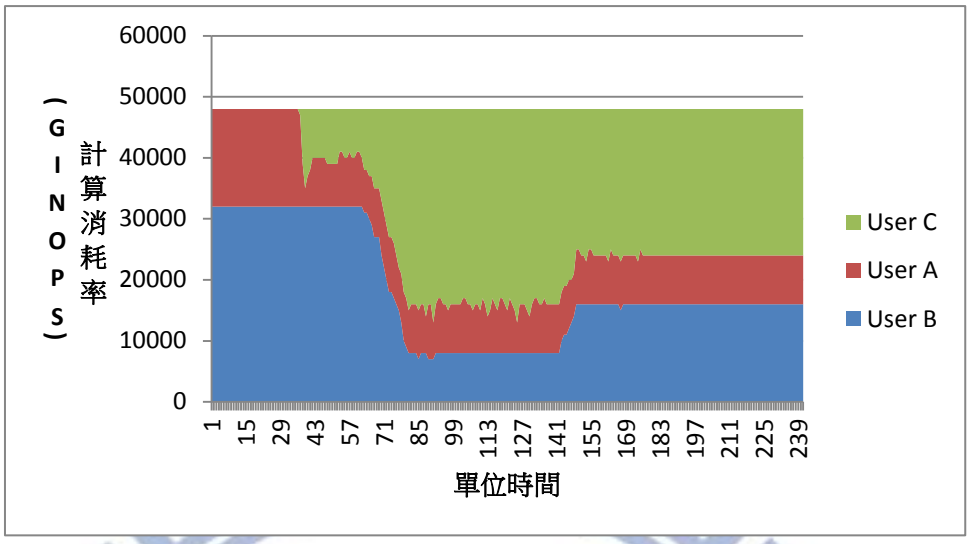


圖 35：組織之間資源使用量不平衡

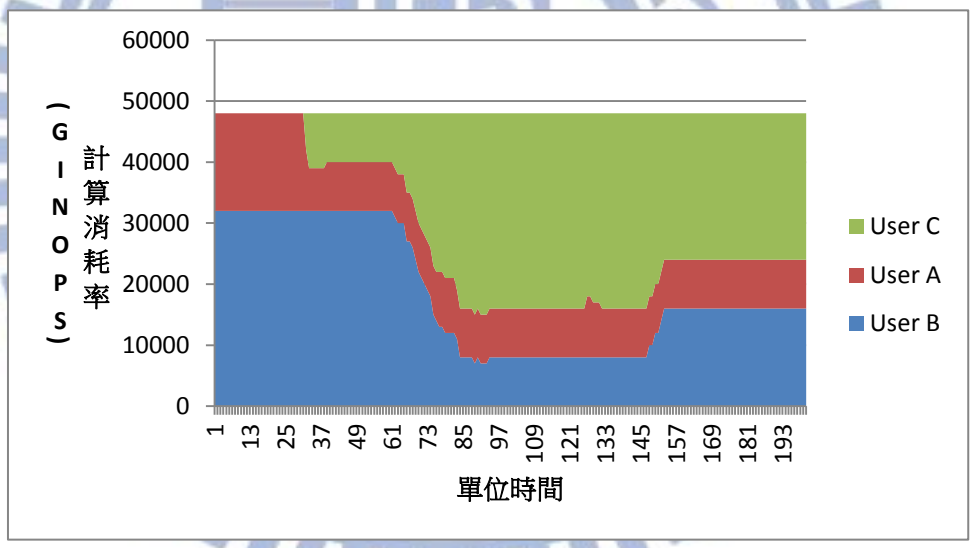


圖 36：規範組織之間使用者的搶資源上限

在 3.2.4.4 節提出規範使用者的搶資源上限之策略，會限制使用者依照組織的機器捐獻比例來搶資源，由於圖 36 使用者依比例搶資源不夠顯著，所以另外設計一個實驗。

共有三位使用者 A、B、C，來自於不同組織，他們組織的機器捐獻比例為 1：2：3，先由 B 送長工作，中途 A、C 加入送短工作，結果如下：

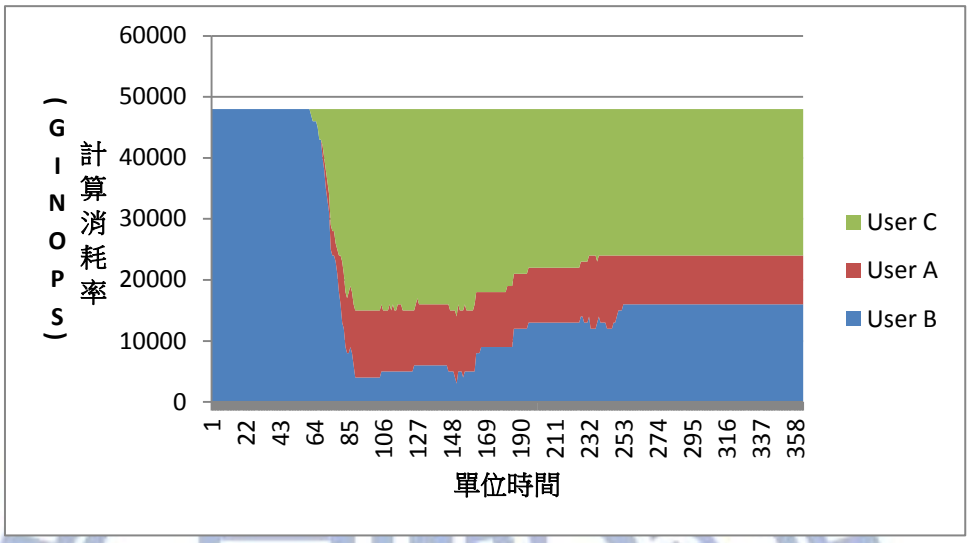


圖 37：實驗結果區域圖

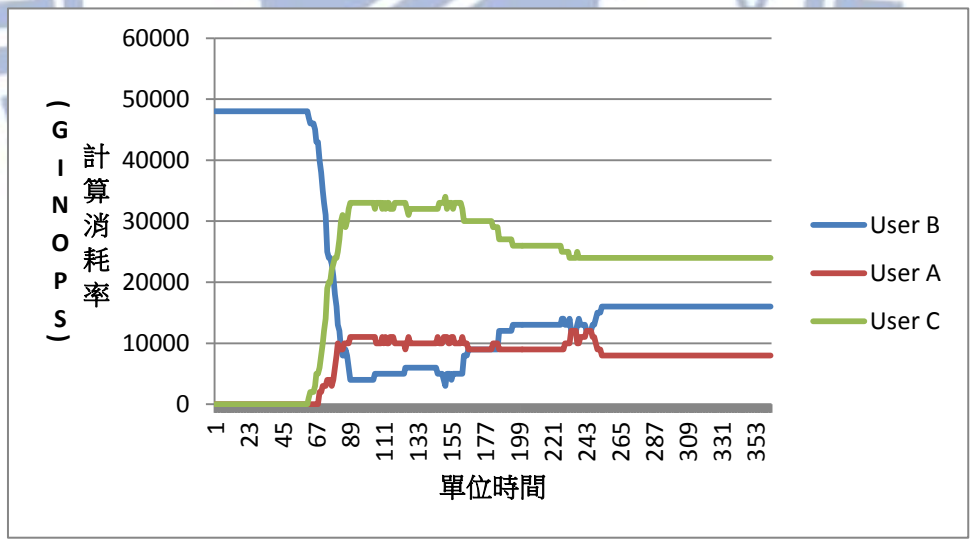


圖 38：實驗結果折線圖

由圖 38 能夠清楚看出使用者 A、C 盡量依照 1：3 的方式，在搶奪使用者 B 釋放出來的資源。

第五章、結論

在此章主要為本篇論文做總結，說明本篇論文的貢獻。

5.1. 貢獻

在本篇論文中，提出了新增或改良的資源分配管理設計與實作，來依序解決以下的問題：

組織之間的公平性：本篇論文將信用值的計算，套用於組織之間(3.2.2.4 節)，使得各個組織依據機器捐獻之比例，做償還資源的事情。相較於本系統以往根據資源分配原則(3.1.1 節)，每個組織只能依照機器捐獻之比例來使用，在資源使用上，能有較大的彈性空間，去償還先前所借的運算資源。

多執行緒工作的公平性：本篇論文將多執行緒工作所使用的核心數量，做動態調整(3.2.3.3 節)，將系統的資源使用率提升。此策略除了提升本系統的資源使用率之外，也將本系統所處理的單核工作，擴充成能夠排程、處理多核工作之間的公平性。

資源使用效能：本篇論文對於『使用者使用資源的平衡性和穩定性，會影響使用者工作的整體效能』，提出加速倍率的假設，並且論證其議題，也提出新策略來規範使用者的搶資源上限，藉此提升使用者使用資源的平衡性和穩定性，亦即提升使用者工作的整體效能。最後也用實驗來驗證該議題，得出相同結論。

藉由新增或改良的資源分配策略，提升本系統在資源分配的公平性和資源使用的效能，希望未來能提供高穩定、高效能的桌機格網計算平台，藉此來吸引更多的志願者或組織加入本系統。

參考文獻

[1] Abramson, B., "Expected-outcome: a general model of static evaluation", IEEE Transactions on PAMI, vol. 12, pp. 182–193, 1990.

[2] Anderson, D. P., "BOINC: A system for public-resource computing and storage", 5th IEEE/ACM International Workshop on Grid Computing, November 2004.

[3] Background Pi, available at http://defcon1.hopto.org/pi/index.php.

[4] BOINC website, avalible at http://boinc.berkeley.edu.

[5] Bruegmann, B: Monte carlo go, 1993.

[6] Chen, C.P., "Desktop Grid Computing System for Connect6 Application", Master thesis, Institute of Computer Science and Engineering, National Chaio Tung University, Hsinchu, Taiwan, 2009

[7] Chen, Y.W., "The Study of the Broker in a Volunteer Computing System for Computer Games", Master thesis, Institute of Computer Science and Engineering, National Chaio Tung University, Hsinchu, Taiwan, 2011.

[8] Fedak, G., Germain, C., Neri, V., and Cappello, F., "Xtremweb: A generic global computing system", In Proceedings of the 1st IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID 2001): Workshop on Global Computing on Personal Devices, IEEE CS Press, Brisbane, Australia, 582-587, 2001.

[9] Gelly, S., Wang, Y., Munos, R., Teytaud, O., "Modification of UCT with patterns in Monte-Carlo Go", Technical Report 6062, INRIA. 2006.

[10] Han, S.Y., "The Study of the Worker in a Volunteer Computing System for Computer Games", Master thesis, Institute of Computer Science and Engineering, National Chaio Tung University, Hsinchu, Taiwan, 2011.

[11] Hsu, F-H (2002). Behind Deep Blue: Building the Computer that Defeated the World Chess Champion, Princeton University Press, 2002.

[12] J. Shoch, J. Hupp, "Computing practices: the 'Worm' programs—early experience with a distributed computation", Comm. ACM 25 (3) (1982) 172–180.

[13] Jou, C.Y., "The Study and Design of the Generic Application Framwork and Resource

Allocation Management for the Desktop Grid CGDG", Master thesis, Institute of Computer Science and Engineering, National Chaio Tung University, Hsinchu, Taiwan, 2010.

[14] Yi-Chang Shan, I-Chen Wu, Hung-Hsuan Lin, and Kuo-Yuan Kao, "Solving Nine Layer Triangular Nim", Journal of Information Science and Engineering (SCI), vol.28, No.1, pp.99-113, January, 2012.

[15] Sarmenta, L.F.G., Volunteer Computing. PhD thesis, "Massachusetts Institute of Technology", June 2001.

[16] SETI@home, available at http://setiathome.ssl.berkeley.edu.

[17] Wu, I.C., Chen, C.P., "Desktop Grid Computing System for Connect6 Application", Institute of Computer Science and Engineering College of Computer Science NCTU, August 2009.SW

[18] Wu, I.C., Chen, K.Y., "The Development of the Multi-Broker Desktop Grid for Computer Games", Institute of Computer Science and Engineering College of Computer Science NCTU, 2012.

[19] Wu, I.C., Chen, Y.W., "The Study of the Broker in a Volunteer Computing System for Computer Games", Institute of Computer Science and Engineering College of Computer Science NCTU, 2011.

[20] Wu, I.C., Huang, D.Y., and Chang, H.C., "Connect6", ICGA Journal, Vol. 28, No. 4, pp. 234-241, December 2005.

[21] Wu, I.C., Jou, C.Y., "The Study and Design of the Generic Application Framwork and Resource Allocation Management for the Desktop Grid CGDG", Institute of Computer Science and Engineering College of Computer Science NCTU, 2010.

[22] Wu, I.C., Lin, H.H., Sun, D.J., Kao, K.Y., Lin, P.H., Chan, Y.C., and Chen, P.T., "Job-Level Proof-Number Search", the IEEE Transactions on Computational Intelligence and AI in Games (SCI), DOI: 10.1109/TCIAIG.2012.2224659, Vol. 5, No. 1, pp. 44-56, March 2013.

[23] XtremWeb website, avalible at http://www.xtremweb.net/

[24] Yen, S.J., Chen, J.C., Yang, T.N., Hsu, S.C., "Computer Chinese Chess", ICGA Journal, vol. 27, no. 1, pp. 3-18, ISSN 1389-6911, March 2004.

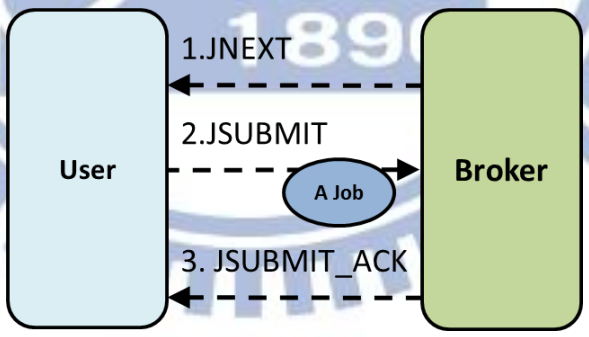
附錄一、CGDG 指令規格

此附錄主要列出使用者、仲介者、工作者指令規格，根據不同的情境(Scenario)來進一步描述使令的用法，以及操作。

本論文新增的指令為：動態更改工作所使用的核心數量指令，其餘的指令皆是沿用原有的設計[13][17][18][19]。

A. Scenario 1: 工作派遣

當仲介者認為使用者有權限發送工作時，便會發送 JNEXT 指令給使用者，當使用者收到 JNEXT 指令時，可以透過 JSUBMIT 指令來發送工作，當仲介者收到此指令時，將會回復 JSUBMIT_ACT 指令給使用者告知工作上傳狀況。



JNEXT 指令格式：

Tag	Type	Description
UID	u_int	User id, given by broker.

以下是一個範例：

```
<ROOT>
        <CMD>JNEXT</CMD>
        <UID>23</UID>
</ROOT>
```

JSUBMIT 指令格式：

Tag	Type	Description
UID	u_int	User id, given by broker.
JID	u_int	Job id, given by user.
APPNAME	string	Application name.
A_VERSION	string	Application version.
PRIORITY	u_int	Job priority. 1: Class A1, 2: Class A2, 3: Class B1 4: Class B2, 5: Class C1, Others: Undefined
CORE	u_short	Num of core job uses.
ABORTABLE (optional)	-	Job abort able flag.
AVG_TIME (optional)	u_int	Average running time.
MAX_TIME	u_int	Max running time.
ARG	string	Job's argument.

以下是一個範例：

```
<ROOT>
        <CMD>JSUBMIT</CMD>
        <UID>5</UID>
        <JID>1</JID>
        <APPNAME>NCTU6</APPNAME>
        <A_VERSION>1013</A_VERSION>
        <PRIORITY>3</PRIORITY>
        <CORE>2</CORE>
        <ABORTABLE>1<ABORTABLE/>
        <AVG_TIME>3600<AVG_TIME>
        <MAX_TIME>7200<MAX_TIME>
        <ARG>
        -checkwintsumego ;B[JJ];W[HH];W[IH]; -
        exp bh
        </ARG>
</ROOT>
```

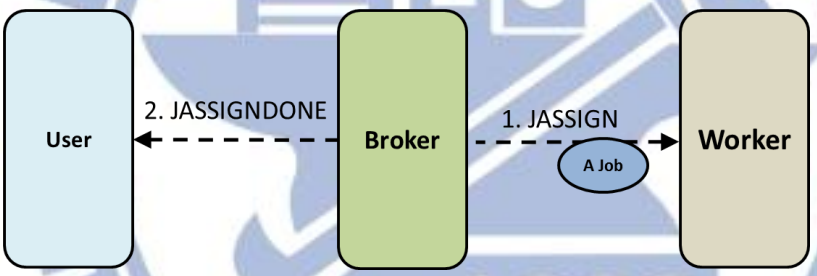
JSUBMIT_ACK 指令格式：

Tag	Type	Description
UID	u_int	User id, given by broker.
JID	u_int	Job id, given by user.
CODE	string	Error code. 001: Success. 030: Privilege denied. 031: Job id repeat. 032: Cannot find matched worker. 033: Number of job limited 034: Job does not register.
MSG	string	Error code message.

以下是一個範例：

```
<ROOT>
        <CMD>JSUBMIT_ACK</CMD>
        <UID>5</UID>
        <JID>1</JID>
        <CODE>001</CODE>
        <MSG>
        Job Submit success.
        </MSG>
</ROOT>
```

當使用者成功的上傳工作後，仲介者會挑選合適的工作者將工作送到此工作者上，仲介者利用 JASSIGN 指令將工作送到工作者上，當成功傳送後，便會利用 JASSGINDONE 指令告知使用者他的工作已成功地派遣到工作者上。



JASSIGN 指令格式：

Tag	Type	Description
UID	u_int	User id, given by broker.
JID	u_int	Job id, given by user.
WID	u_int	Worker id, given by broker.
APPNAME	string	Application name.
A_VERSION	string	Application version.
CORE	u_short	Num of core job uses.

ARG	string	Job’s argument.

以下是一個範例：

```
<ROOT>
        <CMD>JASSIGN</CMD>
        <UID>5</UID>
        <JID>1</JID>
        <WID>30</WID>
        <APPNAME>NCTU6</APPNAME>
        <A_VERSION>1013</A_VERSION>
        <CORE>1</CORE>
        <ARG>
        -checkwintsumego
        ;B[JJ];W[HH];W[IH]; -exp bh
        </ARG>
</ROOT>
```

JASSIGNDONE 指令格式：

Tag	Type	Description
UID	u_int	User id, given by broker.
JID	u_int	Job id, given by user.
WID	u_int	Worker id, given by broker.

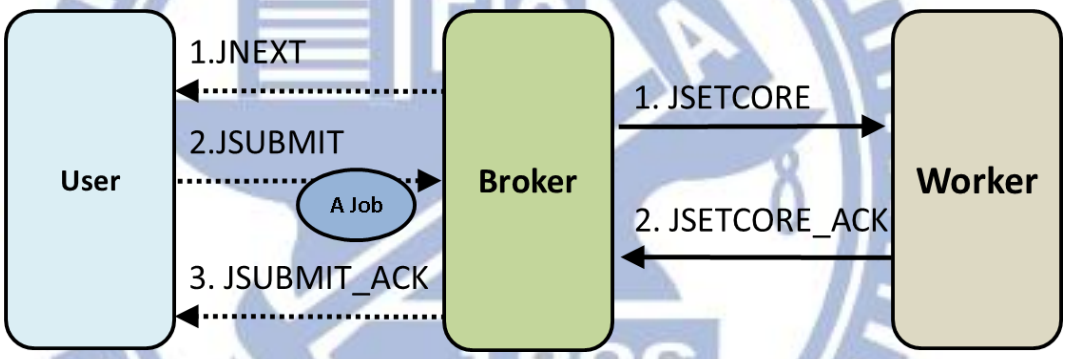
以下是一個範例：

```
<ROOT>
        <CMD>JASSIGNDONE</CMD>
        <UID>5</UID>
        <JID>1</JID>
        <WID>12</WID>
</ROOT>
```

B. Scenario 2: 更改工作使用核心數量

由於工作派遣中，JSUBMIT 有這個工作的使用核心數量，所以仲介者能根據這個值，來更改某個正在運作工作的使用核心數量。

若需要更改某個正在運作工作的使用核心數量時，會發送 JSETCORE 指令，指令包含哪個使用者、哪個工作、更改數量...等。當工作者更改某工作的核心數量之後，不論成功與否，都會回傳 JSETCORE_ACK，告訴仲介者，成功或是失敗的訊息。



JSETCORE 指令格式：

Tag	Type	Description
UID	string	User ID
JID	string	Job ID
WID	string	Worker ID
CORE	string	Current Core Number
SETCORE	string	Set Core Number

以下是一個範例：

```
<ROOT>
	<CMD>JSETCORE</CMD>
	<UID>1341394319</UID>
	<JID>2</JID>
	<WID>1571356308</WID>
	<CORE>5</CORE>
	<SETCORE>7</SETCORE>
</ROOT>
```

JSETCORE_ACK 指令格式：

Tag	Type	Description
UID	string	User ID
JID	string	Job ID
WID	string	Worker ID
CORE	string	Current Core Number
SETCORE	string	Set Core Number
CODE	string	Error code 001: Success 060: Failed
MSG	string	Error code message

以下是一個範例：

```
<ROOT>
	<CMD>JSETCORE_ACK</CMD>
	<UID>1341394319</UID>
	<JID>2</JID>
	<WID>1571356308</WID>
	<CORE>5</CORE>
	<SETCORE>7</SETCORE>
	<CODE>001</CODE>
	<MSG>Case success</MSG>
</ROOT>
```

C. 錯誤碼表格

Code	Case	Description
001	Success	Case success
010	General Failure	Account does not exist or password does not correct, etc.
011	Registration Failure	IP is rejected.
020	Worker Selection Failure	Worker does not exist.
021	Worker Selection Failure	Cross organization.
022	Worker Selection Failure	Cores is not enough.
030	Job Submission Failure	Privilege
031	Job Submission Failure	Job ID repeat
032	Job Submission Failure	Cannot find matched worker
033	Job Submission Failure	Number of jobs limited
034	Job Submission Failure	Job does not register.
040	Job Execution Failure	Cannot invoke application
041	Job Execution Failure	Application crash
050	Job Abort Failure	Cannot find job
060	Job Set Core Failure	Cannot set job core number