東海大學資訊工程學系研究所

碩士論文

指導教授：陳隆彬博士

# 適用於遊戲樹搜尋之高效能桌機網格資源分配方法

# Efficient Resource Sharing in Desktop Grids of Game Tree Search

研究生：歐政昌

中華民國一百零四年六月

# 東海大學碩士學位論文考試審定書

東海大學資訊工程學系 研究所

研究生 歐 政 昌 所提之論文

適用於遊戲樹搜尋之高效能桌機網格資源分配方法

經本委員會審查，符合碩士學位論文標準。

學位考試委員會
召 集 人 _____ 簽章

委 員 _____

_____

_____

指 導 教 授 _____ 簽章

中華民國 104 年 6 月 18 日

# 摘要

隨著桌上型電腦的能力愈來愈強大，人們開始研究如何將桌機的閒置的資源集中化，組成所謂的桌機網格來進行大型科學計算專案(e-Science)。本論文將多組織的桌機網格聯盟應用於遊戲樹搜尋，透過資源共享來解決單一組織不易達到的大型遊戲樹搜尋應用。傳統的桌機網格的 broker 依據使用者 credit 來分配資源，由於遊戲樹搜尋應用可以動態生成並且分割工作，在多人共享資源的環境中，credit 很容易劇烈變化，造成資源分配的震盪現象，產生不必要的資源分配的切換。本論文探討工作主機的數量、系統可用資源的數量、工作數量等因素對於 credit 的影響`。我們在 broker 與 credit database 之間加入一個 credit damping 元件，此元件能緩和 credit 快速的相互作用，避免不必要的資源分配的切換來提高系統的性能。

關鍵字: 桌機網格、資源分配演算法、使用者信用、遊戲樹搜尋

# Abstract

A desktop grid federation has enabled the organizations to solve large-scale game tree search applications via resource sharing. Since such applications can generate/prune tasks dynamically, the simple credit-based resource broker can easily lead to resource thrashing, causing a high computing overhead. We develop a credit damping component that is added between the broker and the credit database. By applying certain equalization function on the credit data, the broker avoids rapid interact to credit updates, therefore prevents thrashing and then enhances the system performance.

Keywords: Desktop Grid、Credit、Resource sharing、Game tree search

# 目錄

# 圖表目錄

# 一、介紹

今日的個人電腦遠比十年前的工作站甚至大型機房更強大，因此我們認為建構低成本的桌面電腦來解決大型的計算應用程式是可行的。桌面網格是一個網路的計算模組，可以收集來自桌面電腦的計算結果。主要伺服器的節點用於協調桌面網格的應用程式的執行。伺服器分配任務至大量的計算節點上，等待執行結果，並將最終結果做驗證來確認最終結果。這種計算模式具有顯著的韌性，因為計算節點可以是不同的操作系統，而且不一定會有關聯性的連接。

如今很多研究機構已經建立的桌面網格作為大型電子科學項目的解決方法。志願計算伺服器負責分配任務給公眾匿名參與者，即為志願者。由於志願者是自願的，會不時上線或離線。在統計學上，資源可用性可以維持在一定的水平，具有足夠數量的志願者，單一工作者的回應時間通常不會造太大的問題。

桌面網格的另一個模式是由多個組織組成，每個組織會貢獻許多 worker 主機。這些主機由各個組織自行維護和直接控制。worker 主機可以是員工的個人電腦，或是專門的計算節點來提供穩定的計算資源。

桌面網格聯盟使企業通過資源共享來解決大規模的應用程式，本文研究這個計算平台的大規模遊戲樹的搜索任務，由於在計算節點的系統運作，可以任意終止，並重新發送，桌面網格適合延展性應用程式，可以在執行過程中改變自己對處理器的要求。因此，一個遊戲樹的搜索專案可以要求任務動態的生成與調整，可以很容易在桌面網格上實現可延展的應用程式。

桌機網格的 broker 負責統計各個組織的貢獻度與資源消耗量，將一段時間內的統計資料量化為 Credit 的數值。資源伺服器在分配資源時，便以各個組織的 credit 作為依據。由於遊戲樹的搜尋應用程序可以動態生成與調整的任務，傳統的依據 Credit 來分配資源的 broker 很容易造成資源的震盪，同一時間，多個組織都送出工作，若此時資源不足，便會產生競爭現象，造成同一個資源在短時間內，一下子分配給組織 A，一下子分配給組織 B，造成資源管理成本增加。

為了改善這種現象，我們研究資源震盪的現象在桌機網格聯盟上，本論文提出一個改良這個問題的方法。我們在資源分配伺服器(broker)與 credit database 之間插入一個 credit damping 元件。此元件能控制 credit 的變化以避免快速的相互作用讓 credit 的更新產生震盪，來提高系統的性能。我們開發了資源分配演算法來確保公平與穩定的資源分配，來提高聯盟的總體效率。

本論文採用 OMNet++模擬整個運行流程，並在 Desktop Grid Federation 平台環境下，將震盪的可能情況做模擬測試，以達到資源可以高效率的運用的目的，後續章節安排如下：第二章介紹桌機網格平台與應用、桌機網格聯盟，第三章則講述實驗的系統架構，第四章說明傳統 Credit 基於資源分配在桌機網格上的運作以及討論加強資源分配的演算法，第五章說明實驗結果，第六章分析結論及未來工作。

# 二、 背景與相關技術

## 2.1 Desktop Grid Applications

相較於傳統 HPC Cluster 或是平行電腦，現今的 Desktop Grid 的模式是相對便宜，那麼桌機網格是從何而來?其實桌機網格不是新的名詞在概念上取自於電力網格(Electric Power Grid)而來，在日常生活中，各電廠資源可以便利調度，人們只要接上電源插頭，就可以使用電器用品，而不需要指定或知道電從何而來，網格即是將電力網格這樣的概念賦予網路世界中的新意義，也就是說，各式網路資源(如計算、儲存、資料庫、以及儀器設備等資源)可以便利調度，任何人只要連上網路，便可自由使用網路世界中各類資源，在網格架構下，將原本各自運作的電腦連結成一部超級大電腦，成為最具整合性的分散式計算系統，使用者只須要提出計算需求，系統便會自動分配資源，而無需考慮資源的實際位置。

因此，網格不僅提供了網路世界中資源溝通管道，更促使人們得以處理大量的資料、更複雜的計算、並建立超越地域的合作。而且網格也已經應用在各種領域，比如高能物理、藥物設計－抗禽流感藥物、生物資訊－蛋白質摺疊、地震減災應用、氣候模擬，天文－探索外太空文明等等，應用的領域相當廣泛。

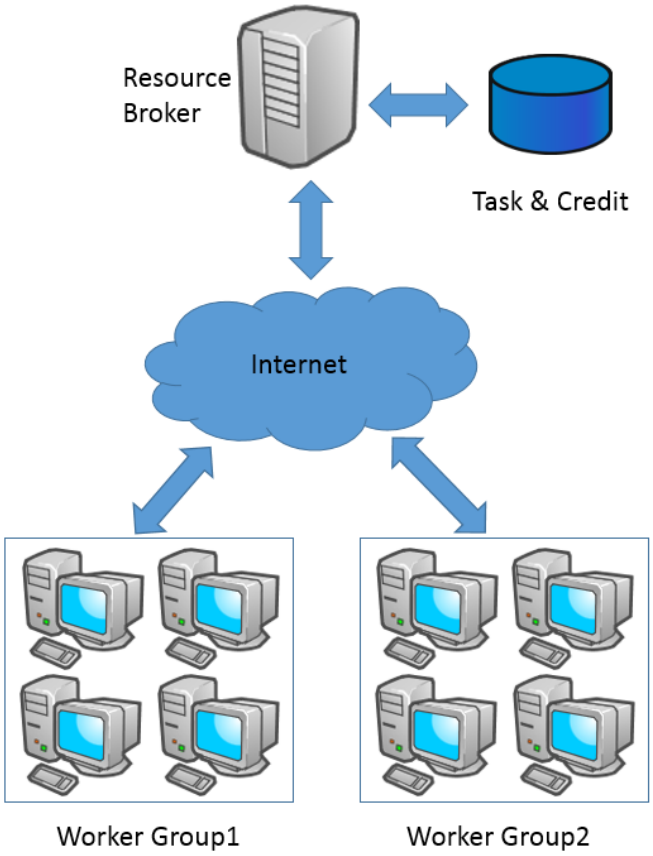
## 2.2 Desktop Grid Federation

TWGrid 是由台灣的中央研究院網格計算團隊 (Academic Sinica Grid Computing, ASGC)與中研院的高能物理團隊、台灣大學物理系與中央大學物理系開始共同導入與佈建全球網格系統，以支援國內高能物理 ATLAS 與 CMS 實驗研究人員進行數據分析模擬。數年來，藉由積極參與全球網格計畫之策畫、技術研發及全球網格系統維運，如今台灣已獲得 CERN 與其他國際合作團隊之認可，並於 2005 年 12 月由中研院與 CERN 簽屬備忘錄(MoU)，受邀擔任目前亞洲唯一的 Tier-1 中心，此外，台灣亦獲 EGEE 計畫邀請擔任亞洲區總協調，負責帶領亞洲地區建立網格應用環境與長期營運之機制。

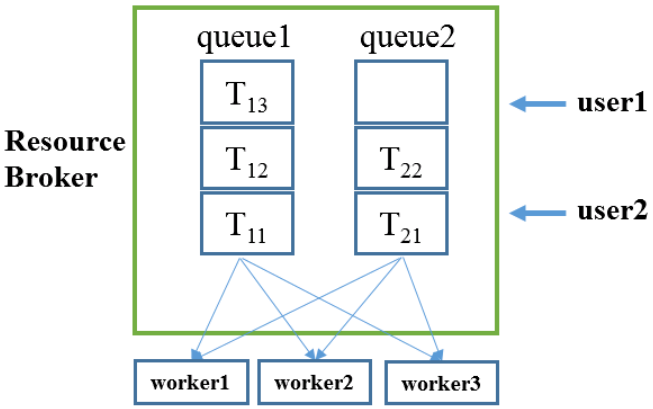
目前大部分的網格計畫主要是建立在 Globus 中介軟體的環境上。Globus 是由位於芝加哥附近的 Argonne 國家實驗室網格計算教父 Ian Foster 教授以及洛杉磯南加州大學資訊科學研究所的 Carl Kesselman 教授共同開發。它是一個開放原碼的基礎平台，提供了網格所需要的基本功能與服務，如安全性、資源選取與匹配、儲存資源管理與資料存取等。

我們開發 desktop grid federation(DGF)的模擬系統，做為計算的模擬，一個 DGF 系統包含組織、客戶(使用者)、worker 還有 broker。如圖表 1 的 DGF 擁有兩個組織的 worker。使用者不定時會需要計算，所以會提交任務到 broker。Worker 會請求 broke 分配任務。Broker 則是協調使用者與 Worker 之間的問題。我們使用基本的推播的通訊方式，在於 Broker 與使用者或 worker 之間。由於使用者跟 worker 與 Broker 之間的所有連線都是專用的，可以立即送出任務或訊息。

例如，使用者發送任務給 broker，Broker 馬上將任務發給 worker，Worker 做完任務後，結果立即回傳給 broker，Broker 會將回傳的任務做統整後，再回覆給使用者。因使可以實現即時的互動與高度的動態地調度。如圖表 2 所示。




圖表 1　Desktop Grid Federation

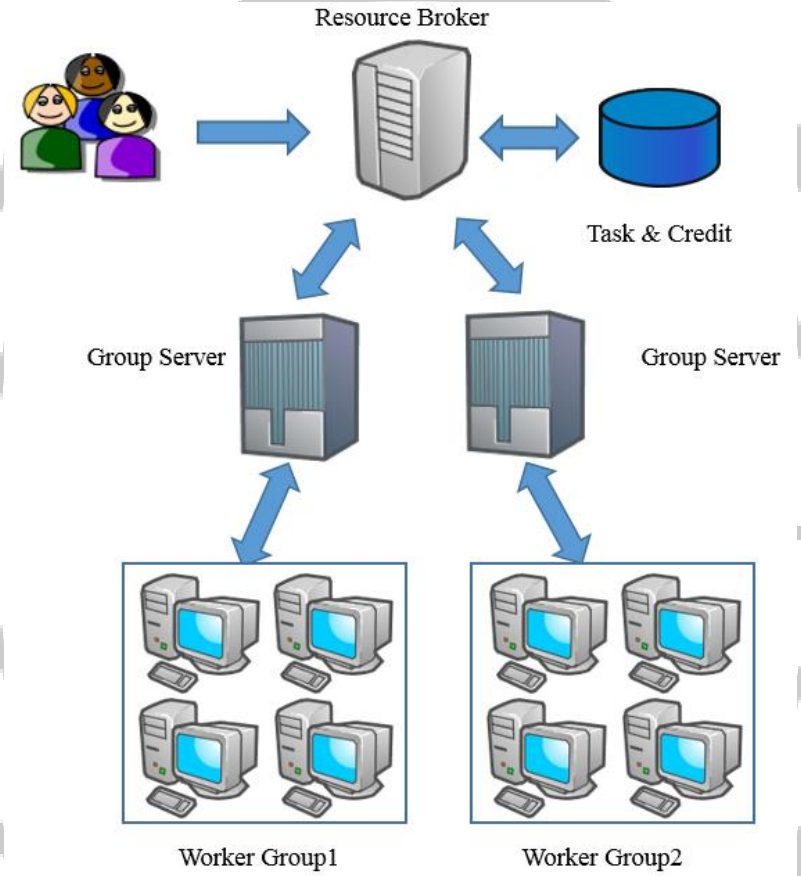



圖表 2　Job queue

# 三、 DGF 架構與 Credit 資源分配方法

在桌機網格中，broker 會依據每個 user 的貢獻度來設定其 credit 值。Broker 會依據 user credits 來分配資源。本章中，3.1 節說明桌機網格聯盟的架構，3.2 節說明傳統 credit 計算方式，最後，3.3 節探討傳統 credit 計算方式會引發的資源分配震盪現象及其缺點。

## 3.1 DGF 系統設計




圖表 3 DGF 系統架構

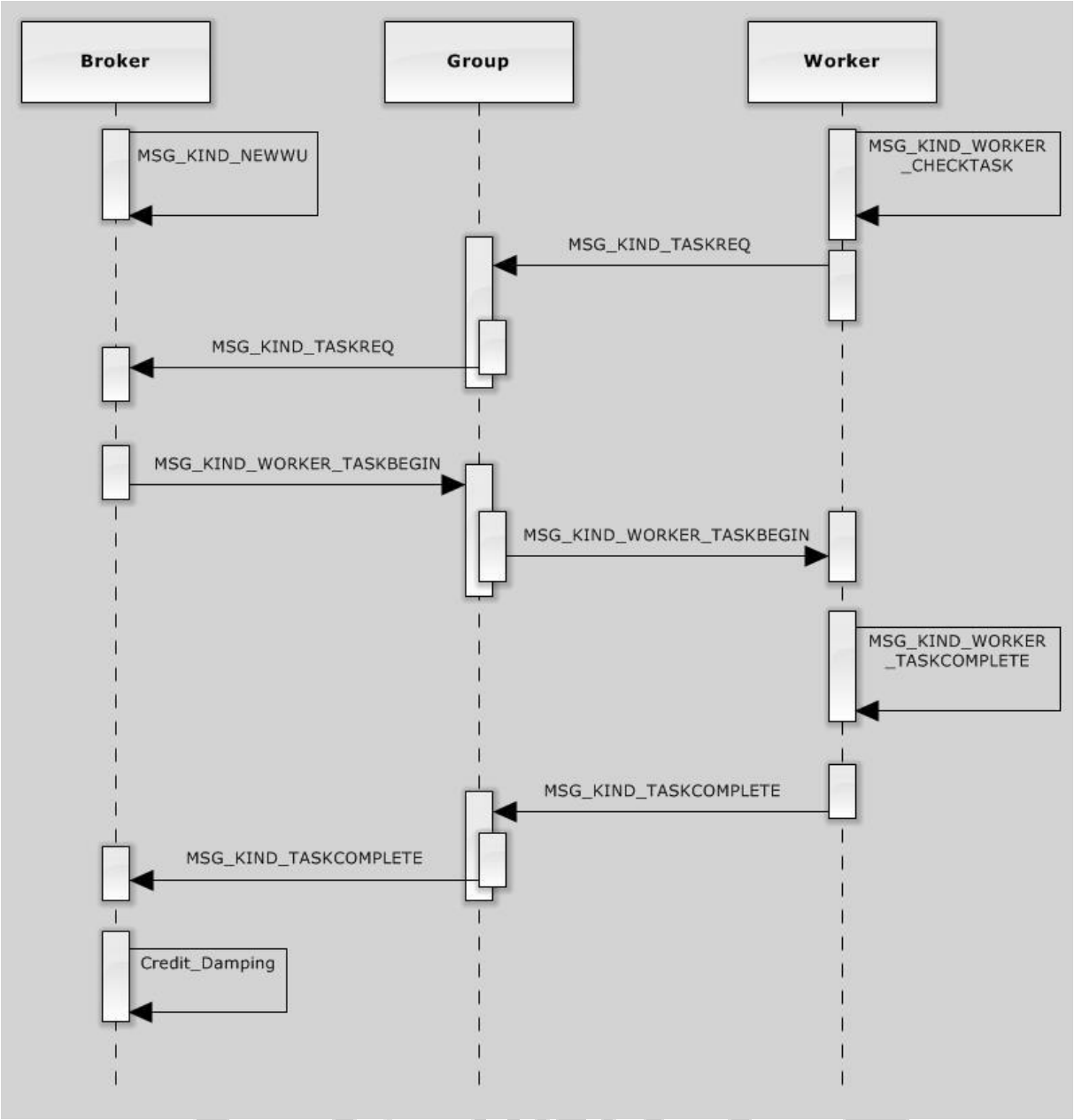
DGF (桌機網格聯盟)的系統架構如圖表 3 所示。DGF 包含了 user、worker group、group server、broker、和 credit database 等五部分。

圖表 4　DGF 架構說明

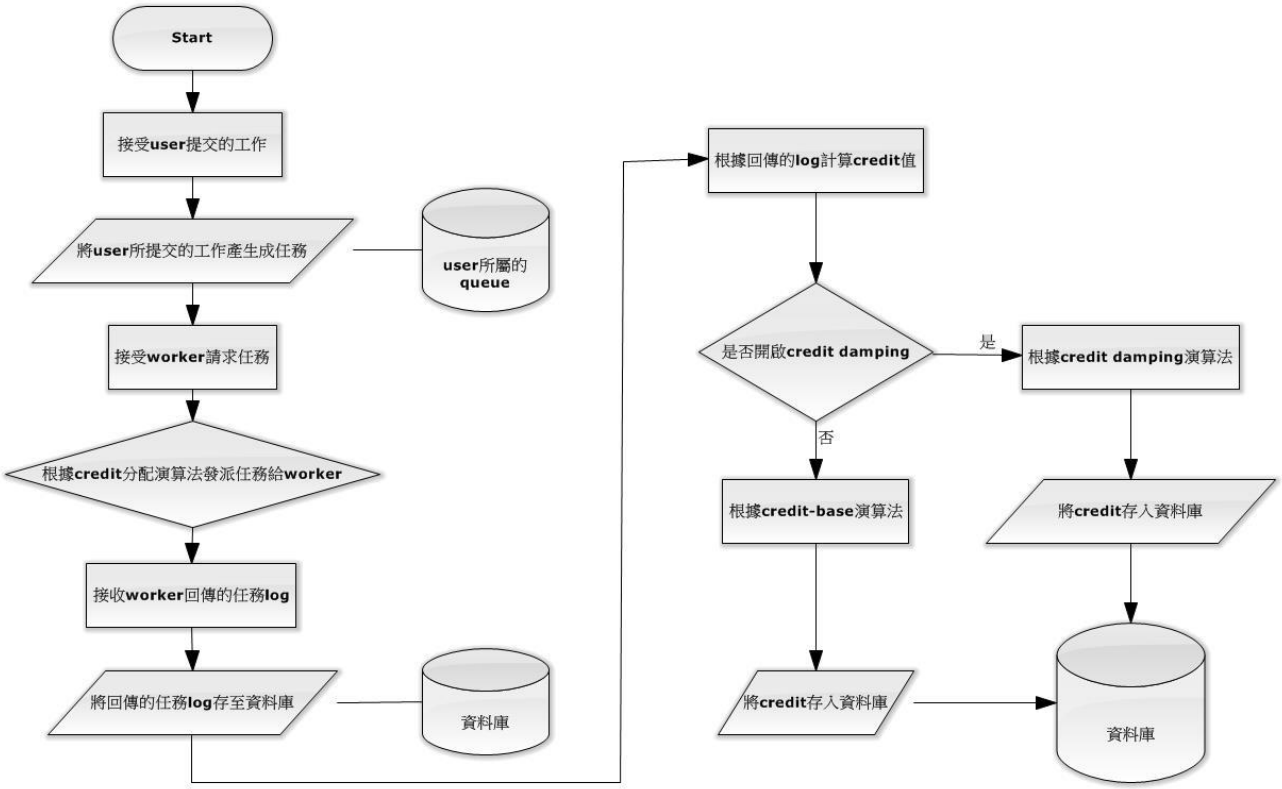
| | |
|---|---|
| Worker | 接受 broker 所發配的任務來計算，計算完成後，回傳至 broker。 |
| Worker group (或簡稱 group) | 一個組織會建立並擁有一個 worker group。一個 worker group 包含許多稱為 worker 的電腦主機。 |
| User | 一個組織擁有一個 user 帳號，每個組織以其 user 帳號提交工作。Broker 以 user 帳號識別工作是由哪個組織發出的。 |
| Broker | 接受 user 所提交的工作，並產生任務。根據 user 的優先權來發配任務至 worker。 |
| Credit database | Broker 會依據每個 user (或是組織)的貢獻度來設定其 credit 值。Credit database 儲存 user 的 credit 值。 |

本論文的 DGF(桌機網格聯盟)是以事件驅動模型(event driven model)來運作。DGF 的循序圖如圖表 5 所示，流程圖如圖表 6 所示，DGF 的工作流程可以分為以下五個主要部分：

1. broker 產生工作：user 發出新工作訊息(MSG_KIND_NEWWU)到 broker，broker 收到此訊息會產生新任務並加到其組織 queue 中。
2. Worker 向 broker 要求工作：worker 發出任務請求訊息(MSG_KIND_TASKEQ)給 broker。
3. Broker 分配工作：收到 worker 任務請求訊息後，會發出任務與任務訊息(MSG_KIND_WORKER_TASKBEGIN)給 worker。
4. Worker 完成工作：任務完成後，回傳任務與任務訊息(MSG_KIND_TASKCOMPLETE)給 broker。
5. Credit damping：關閉 CD 的情況下，根據回傳的任務計算 credit 更新到資料庫，開啟 CD 的情況下根據回傳的任務計算 credit 更新後，將 credit 值放進訊息(credit damping)中進行資料庫更新。


Broker
Group
Worker
MSG_KIND_NEWWU
MSG_KIND_WORKER
_CHECKTASK
MSG_KIND_TASKREQ
MSG_KIND_TASKREQ
MSG_KIND_WORKER_TASKBEGIN
MSG_KIND_WORKER_TASKBEGIN
MSG_KIND_WORKER
_TASKCOMPLETE
MSG_KIND_TASKCOMPLETE
MSG_KIND_TASKCOMPLETE
Credit_Damping

圖表 5　DGF 循序圖




圖表 6　DGF 工作流程

## 3.2 傳統 Credit-Based 資源分配方法

在單一桌機網格中，broker 會嘗試增加系統的處理能力，所以在選擇 worker 時，會找可靠度達到門檻的 worker。但對於桌機網格聯盟，除了 worker 能力，還必須考慮互惠共享的因素。以下是桌機網格聯盟的 broker 要額外考慮的因素：

- Fairness：在組織中的參與者，資源可用度與用戶的貢獻成正比。
- Starvation-free：每個參與者都能獲得最低限度的資源，來完成自己的任務需求。

為了滿足以上兩個條件，大部分的系統會依照各個組織的貢獻度(credit)來決定它們所能分得的資源。每一個組織 A 的 credit 值(寫成 credit(A))代表它的資源貢獻量減去資源使用量。Credit 值通常使用 GINOP (Giga Integer Operations)和任務的執行時間的長度的乘積估算。此處我們採用 GINOP 而非 FLOPS (Floating-point operations per second)，因為遊戲樹的搜尋偏向整數計算。

Credit 的計算方式如下，當組織 A 的 worker 完成一件工作時，組織 A 的 credit 會增加 $C \cdot GINOP_T \cdot L_T$，其中$L_T$ 是該工作的執行時間的長度，$GINOP_T$是該 worker 的 GINOP 值，而$C$ 是一個常數。另外，該工作的發起者組織 B 的 credit 會減少相同的量。注意，若是組織 A 即是組織 B，則 credit 不變。

以下我們先講解一下傳統以 credit 為基礎的的資源分配演算法如下:

1. 某個用戶組織 A 提交任務給 broker，然後在註釋中描述任務的屬性，包括估計的執行時間、處理器的需求、應用程式的類型(互動、一般、或是長期的任務)等等。
2. 令$\lambda_A = \frac{W_A}{W_1+W_2+\cdots+W_N}$，其中$W_x$是組織$x$的 worker 數量。若是組織 A 的 credit 大於$\lambda_A$，borker 會分配資源給組織 A，反之則組織 A 無法取得資源。
3. 為了避免有飢餓的問題，每個組織成員可以獲得由管理員分配的基本額度。
4. Broker 接收到任務時，它會先嘗試將任務分配給該用戶組織內的 worker。如果該組織本身無法接收新的任務，Borker 才會把任務分配到另一個組織的 worker，在選擇組織 worker 時也會選擇較低的 credit 的組織。

## 3.3 Credit Thrashing

傳統以 credit 為基礎的資源分配方式對於 DGF 系統中有著不平衡長度的任務時，很容易導致不穩定的資源分配。舉一個簡單例子，比如系統有一資源 R，而且$\text{credit(B)} - \text{credit(A)} = \Delta$。在時間區段$T_1$中，因為組織 B 擁有較高 credit，資源 R 會被分配給組織 B。組織 B 使用完畢之後，組織 A 的 credit 值增加而組織 B 的 credit 值減少。在下一時間區段$T_2$中，可能組織 A 擁有較高 credit，可以使用資源 R。假設在巧合的情況下，在時間區段$T_1$和$T_2$中，credit 值的增加和減少都是$\Delta$，那麼一直重複$T_1$和$T_2$的資源分配模式，便會使組織 A 和組織 B 的 credit 值在$-\Delta$和$+\Delta$之間變化，形成震盪現象。

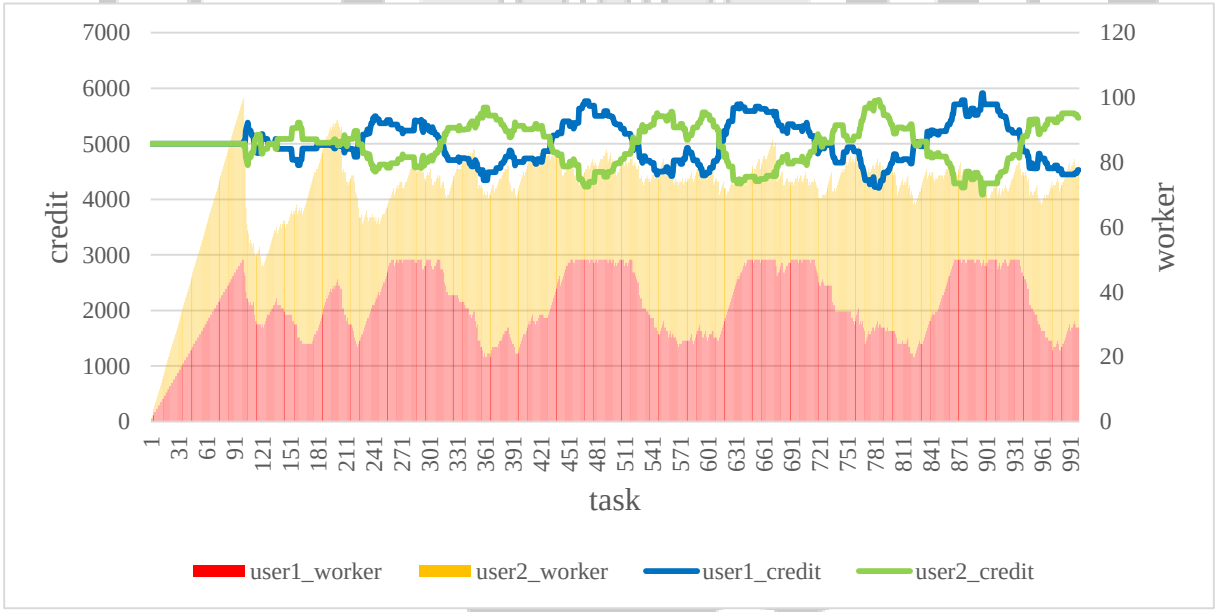
在我們的實驗中，兩個或兩個以上的用戶擁有不同計算長度的任務，很容易造成震盪，還有另一個可能的原因就是，在網格平衡狀態中，有用戶加入或離開。

觀察圖表 7 所示的統計數據，Credit thrashing 不會違反 Fairness 的規則，因為在兩個用戶的資源使用率上幾乎相等。但是，在桌機網格聯盟下，Resource thrashing 會引起計算資源浪費，由於以下原因：

- Worker 在很短的時間內有很多任務進行切換，加上有 context-switching overhead 會影響系統的效率。
- 屬於相同用戶的任務分配給不同的 worker，導致更高的通信成本。
- Broker 具有動態地記錄以及追蹤任務分配給不同的組織的作用。
- 網路程式的資料多半具有 locality 屬性，也就是一個資料會在一段時間內頻繁地被參照使用。若是 worker 一直切換著執行不同 User 或是組織的工作，那麼 worker 必須重新載入資料或程式，這會花費不少時間成本。

圖 3 呈現一個 credit thrashing 的現象。實驗預設條件如下，Credit_dampig=未開啟、Worker_num=100 、User_num=2 、User_credit=5000 (預設使用者的 credit 值) 、Task_number=500，以上為我們的預設條件，首先說明，各 user 的 credit 值預設為 5000，可以供 broker 做為資源優先權的參考。

實驗一開始，因為 credit 值都為 5000，所以每次分配任務，以亂數分配為主，雖然以亂數分配，但從分配結果可以得知，兩位使用者各占一半的資源，如**錯誤! 找不到參照來源。**所示，在第一個任務完成回傳的時候，所記錄下來的 Core 數。雖然以亂數分配的方式來讓兩者使用資源的分配各站將近一半的比例，但是還是會有誤差，以圖 3 來說，在 task=211 時，User1 的 credit 值逐漸上升，這時 Broker 在分配工作會以 user1 為主，除非 user1 沒有提交工作，不然資源使用權根本不可能會分配給 user2，當 user1 大量占用資源後，user2 使用的資源趨近於 0，但是在 301 至 331 的任務完成時，user1 與 user2 產生交叉，意思就是說當 user1 已經將剛剛所占用的資源以 credit 值的方式回饋給 user2，這時反過頭來 user2 開始大量占用資源，迫使 user1 無法取得資源，而產生一個惡性循環，我們稱它為過度補償。




圖表 7　Credit 值與 Core 值之變化圖

在我們的實驗結果中，當有兩個或兩個以上的使用者時，很容易產生震盪，然而在 DGF 系統當中，主要影響震盪的因素有：

- Arrival rate of tasks：當時間越長時，任務產生的密集度越低，使得 worker 分配到的任務相當緩和，也就不會產生震盪的現象，但反之，則會產生震盪。
- CPU utilization of workers：此變數是指 worker 本身的計算能力，當計算能力較弱的 worker 還在計算時，相對於計算能力較強的 worker 可能已經將 job 的計算結果

回傳給 broker，由此一來就會影響到消化任務的進度，相對來說就會造成震盪現象。

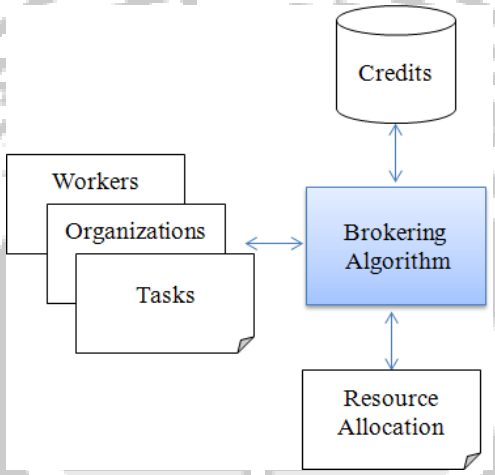
- Number of workers：當 worker 數量越多時，越可以緩衝任務太多而造成的震盪情形，但基準點是能將佇列中的任務完整的消化，否則 worker 數太多，也無法減輕震盪的情況。
- Imbalance of task granularities：Users 的工作顆粒不一致而造成的震盪情形。

# 四、 改良式的 Credit 資源分配方法

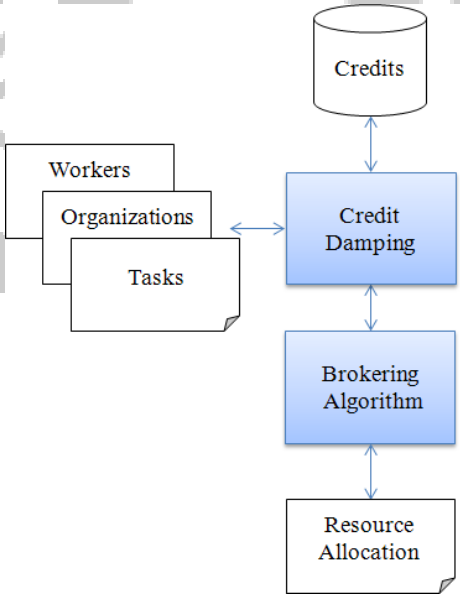
我們研發兩種改良式的 credit 資源分配演算法，試圖避免因為快速的交互作用以及 credit 的更新，而產生 credit thrashing 現象。

## 4.1 Credit Damping Algorithm

我們開發的第一種了資源分配演算法稱為 credit damping 方法。圖表 8 和圖表 9 顯示了傳統的 credit 資源分配方法和 credit damping 方法的差異之處。圖表 9 中，credit damping 元件 (簡稱 CD 元件)被加入到 broker 與 credit database 之間。當 broker 讀取 credit 資料庫時，查詢的資料將會被 CD 元件進行調變。Broker 會根據調變後的資料進行資源分配。因此，控制調變函數將可以影響 broker 行為。




圖表 8　A simple credit-based brokering algorithm




圖表 9　The credit damping (CD) component

本實驗中，我們設計了常數增加 (credit constant ) 的 CD 元件。這個方法是取一個基本量δ，當有組織 A 要增加 credit Δ時，系統不會一次就調整Δ的量，而是定時增加δ的量直到Δ為止：

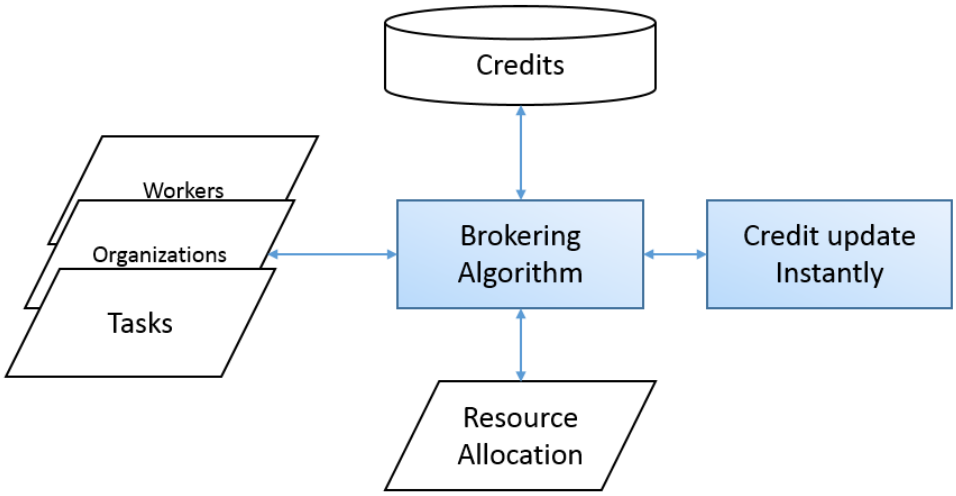
```
class Credit {
   private:
      int value= 0;
      int changeUnit=δ;
   public:
void updateCredit(int  Δ) {
         If (value ≥ Δ ) {
            value = Δ;
            changeUnit =δ;
         } else {
            value += changeUnit;
         }
      }
}
```

在以上 class Credit 中，CD 元件每次呼叫 updateCredit 時，credit 的值會增加 changeUnit 的量，每次呼叫的時間不能間隔太久，因為若是 credit 值不能反映真實貢獻度，則會影響公平性。

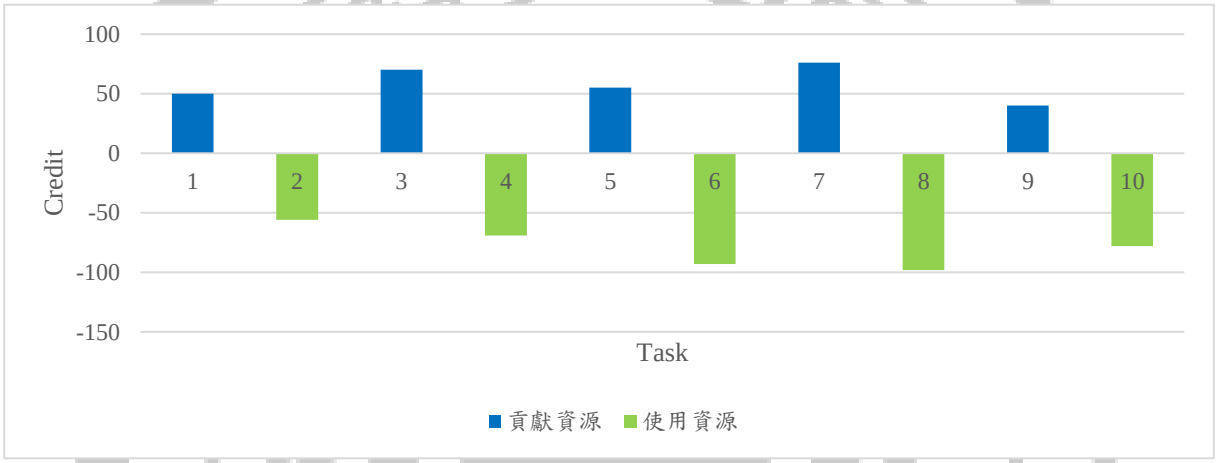
## 4.2 Credit 即時更新

這是我們開發的第二種演算法，Credit 立即更新。我們在 CD 元件的演算法中，主要用來減輕因為 credit 更新的交互作用而產生的震盪，原本 credit 值計算都是任務運算完成回傳，才開始計算並更新，但是因為回傳的時間必定會有時間差，再加上每個組織之間 credit 值的拉扯，而產生震盪的現象，尚不能有很大的改善，所以希望可以將 credit 立即計算並且更新，讓時間有一致性、而組織之間的 credit 值的拉扯也可以達到抵消，對於公平性以及震盪的改善有很大的幫助。

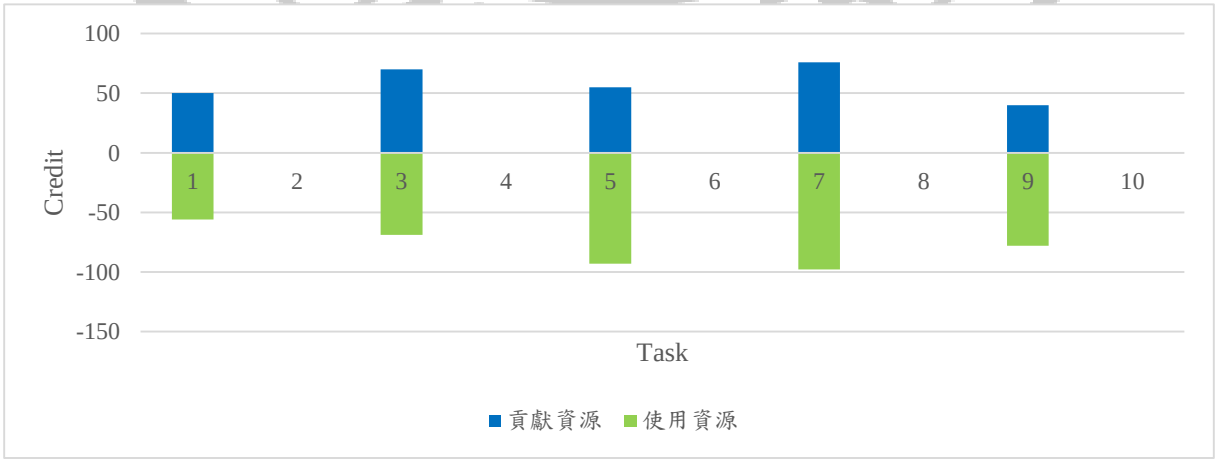
以下我用例子來說明，有兩個組織，A 與 B 組織，各會有因為貢獻資源而增加 credit 值或是使用資源而減少 credit 值。在圖表 11 中，我們可以看到任務回傳後，計算完 credit 值的更新，A 組織使用資源及貢獻出的資源，任務之間的回傳是有時間差的。在圖表 12 中，可以看到因為立即更新的關係，計算任務的 credit 值因為沒有時間差，所以可以抵消兩組織之間的貢獻及使用資源的部分，然而讓 credit 值不會產生太大的變動。




圖表 10　Credit 即時更新演算法




圖表 11　A 組織 credit 值

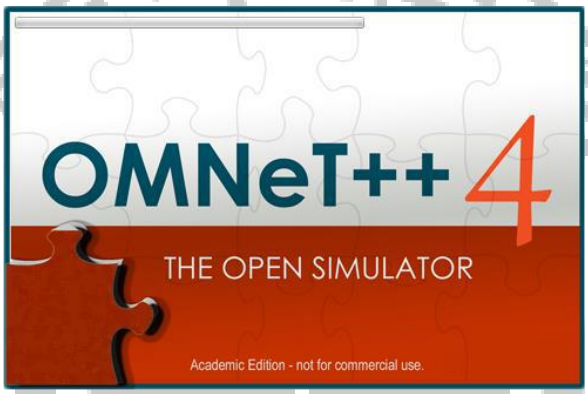



圖表 12　A 組織 credit 值之立即更新

# 五、 模擬實驗結果

本文使用 OMNeT++來模擬實際的網路環境，OMNeT++是個以 C++為基底元件的模擬函式庫(Simulation library)，同時，它具有延展性及框架(Framework)並能將整個網路環境模組化，而它最大的特色，就是除了模擬網路環境外，還能模擬一些更加複雜的環境，例如:IT 系統、佇列網路以及硬體架構等，目前這套 OMNeT++模擬工具，應用於學術研究上可免付費使用，在學術界以及全球科學界都有廣泛應用。

## 5.1 OMNeT++簡介




圖表 13　omnet++圖示

在 OMNeT++的模擬器當中，所使用的嵌入式模型有一部分是採用分層式架構，而對於嵌入式模型來說，它的深度是可無限延伸、擴充的，當使用者想建構實際網路系統在模擬環境中，系統會透過建構系統的邏輯結構來決定是否允許使用者建立此模擬環境，各個模組間藉由傳遞訊息來進行構通，而眾多訊息中不乏結構複雜的數據，每個模組更能擁有各自的參數集，這些參數集將被作為模組的行為規範。

Eclipse 是 OMNeT++所使用的開發環境，擁有眾多外掛模組支援靈活性佳，同時具備圖形及命令列介面，在操作上更加得心應手，並可開發於各種作業系統中，包括 Microsoft Windows、Mac OS X、Linux 及其他 Unix-like 系統。OMNeT++的模型由 3 個部分所組成:

(1) NED 語言拓樸描述(NED language topology description，.ned 檔)：用來定義網路組態，包含組成模組組件及連結 gate 方式，並可使用參數進行調整。訊息定義(Message definitions，.msg 檔)。

(2) 簡單模組原始碼(Simple module sources，.cc 檔或.h 檔)：用來描述網路成員的行為及事件處理，包含初始化以及接受訊息之後的反應。

(3) OMNeT++訊息區包含幾個訊息，其中比較常用有，Problem → 如果在 compile 過程有問題會顯示在這區塊。Event Log → 用來顯示模擬所以是順序以內容。

Process → 用來看目前哪些事件正在進行，哪些在等待。

### 5.1.1 OMNet++的 cSimpleModule 重要函數簡介

#### 5.1.1.1 虛擬函數

- void initialize()：在模組被建立之後被呼叫。
- void handleMessage(cMessage *msg)：在模組接收到 msg 之後被呼叫。
- void finish()：在模擬成功結束之後被呼叫，通常用於收集模擬過程中的統計記錄。

#### 5.1.1.2 成員函數

- send（）：發送訊息給其他模組。
- scheduleAt（）：發送訊息給自己。
- simTime（）：目前模擬程式中的時間。
- setKind（）：設定訊息種類。
- getKind（）：讀取回傳訊息種類。
- setName（）：設定訊息名稱。

## 5.2 程式架構

每個 OMNet 模擬器專案使用兩個檔案，分別是.ned 檔與.cc 檔。其中.ned 檔定義網路組態以及每次模擬的定義參數，.cc 檔描述模組主機行為及事件處理。

### 5.2.1 ned 檔

一個網路包含許多模組敘述，ned 檔就是專門定義平台組態，定義伺服器主機數量以及網路通道連接方式架構，程式架構如下:

圖表 14　ned 檔之程式架構

```
simple　Borker //（simple module）
gates: //定義 submit 的輸出及輸入
simple　Group //（simple module）
gates: //定義 submit2 的輸出及輸入
simple　Worker //（simple module）
```

```
gates: //定義 master 的輸出及輸入
network net //定義 net 組態
submodules: //定義組態中子模組及其數量
broker:
group:
worker:
connections: //定義網路連結方式
```

### 5.2.2 cc 檔

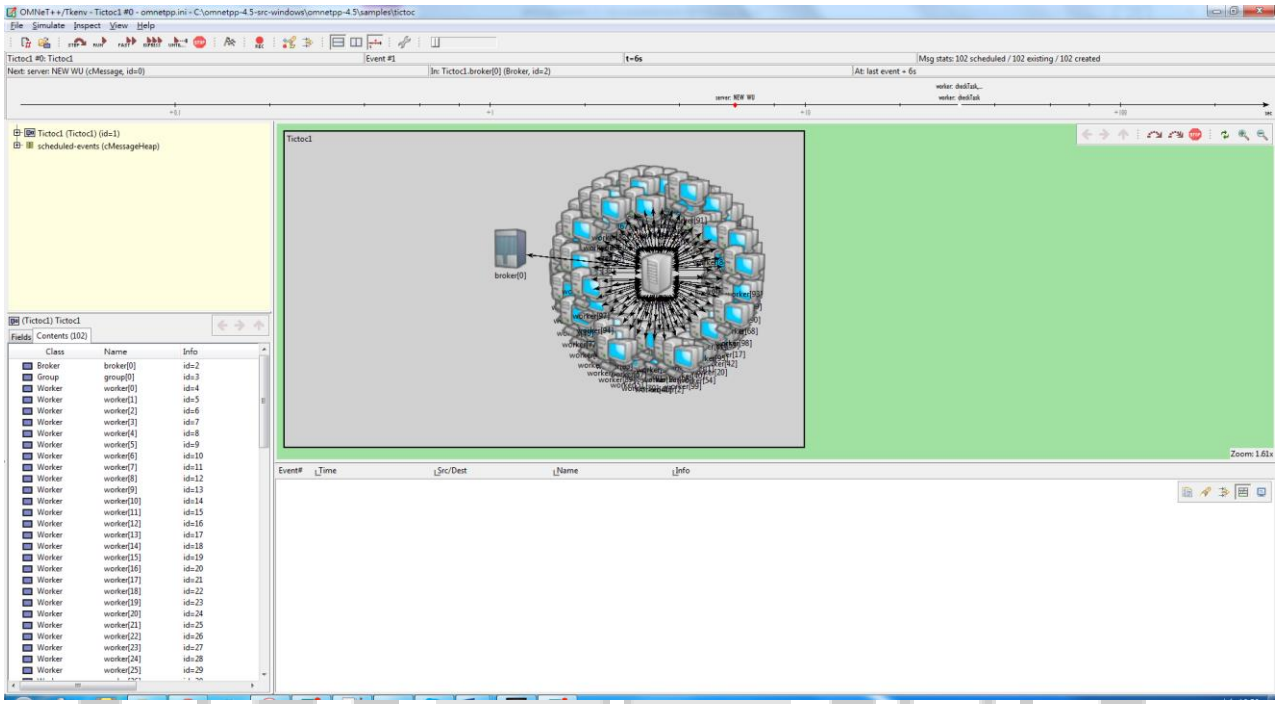
cc 檔主要定義模組溝通其行為，包含定義傳遞的訊息、運算能力，宣告各模組、工作單元、副本，描述接收訊息後的行為等等，程式定義訊息架構如下：

圖表 15　CC 檔之程式架構

```
//定義訊息種類
#define MAX_TRIGGER_DELAY 5
//自定義一個產生動作的時間來推動系統運行
#define STANDARD_TASK_LENGTH 50
//定義所有工作的長度
#define WORKER_CHECK_TASK_PERIOD 30
//worker會呼叫自己來檢查自己是否在計算任務的時間
#define MSG_KIND_NEWWU 0
//broker收到此訊息會產生任務
#define MSG_KIND_TASKREQ 1
//worker送至broker的任務請求訊息
#define MSG_KIND_TASKCOMPLETE 2
//worker送至broker的完成任務訊息
#define MSG_KIND_WORKER_CHECKTASK 3
//worker呼叫自己檢查工作的訊息
#define MSG_KIND_WORKER_TASKCOMPLETE 4
//worker完成工作的訊息
#define MSG_KIND_WORKER_TASKBEGIN 5
//worker的工作請求訊息
#define MSG_KIND_CREDITS_TEMP 6
//每當任務結束後,計算值credit值進入credit damping元件的訊息
#define Credit_Damping 0
```

```
//CD元件是否開啟
#define Credit_Damping_value 15
//CD 元件的緩衝值
```

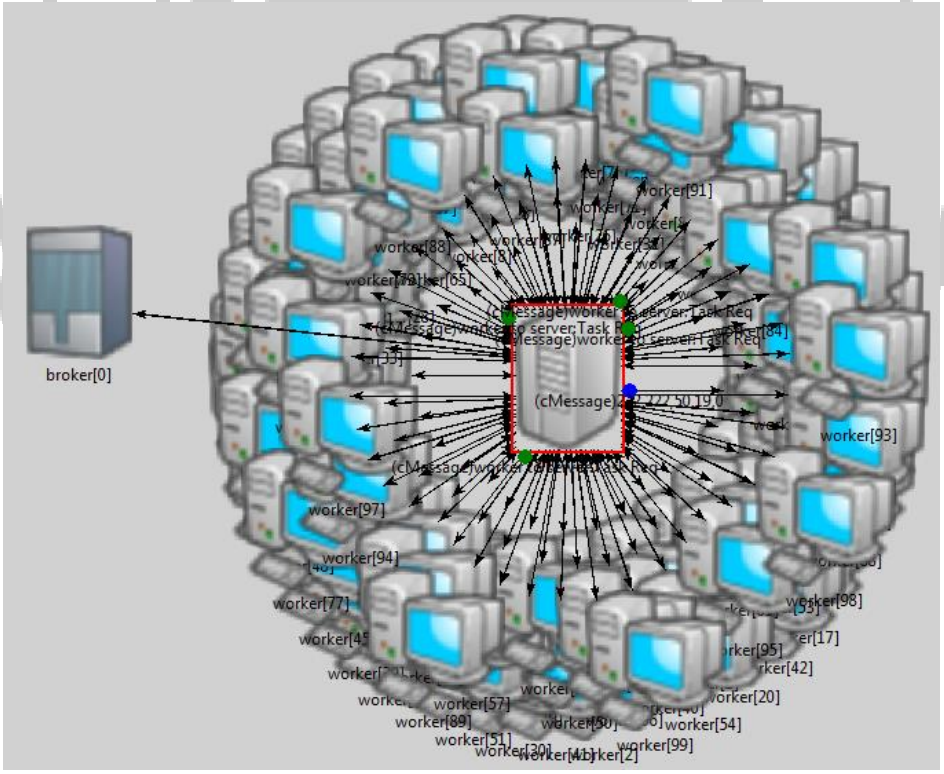
## 5.3 模擬程式畫面



圖表 16 圖形化使用者介面

| Event# | Time | Src/Dest | Name | Info |
|---|---|---|---|---|
| #1614 | 184 | worker[93] --> group[0] | 104,104,50,93,0 | id=684 kind=2 |
| #1615 | 184 | worker[91] --> group[0] | 172,172,50,91,0 | id=937 kind=2 |
| #1619 | 184 | group[0] --> broker[0] | 104,104,50,93,0 | id=684 kind=2 |
| #1620 | 184 | group[0] --> broker[0] | 172,172,50,91,0 | id=937 kind=2 |
| #1623 | 185 | worker[42] --> group[0] | 114,114,50,42,0 | id=738 kind=2 |
| #1624 | 185 | worker[0] --> group[0] | 168,168,50,0,0 | id=923 kind=2 |
| #1626 | 185 | worker[79] --> group[0] | 198,198,50,79,0 | id=1114 kind=2 |
| #1627 | 185 | group[0] --> broker[0] | 114,114,50,42,0 | id=738 kind=2 |
| #1628 | 185 | group[0] --> broker[0] | 168,168,50,0,0 | id=923 kind=2 |
| #1629 | 185 | group[0] --> broker[0] | 198,198,50,79,0 | id=1114 kind=2 |
| #1633 | 186 | worker[12] --> group[0] | 148,148,50,12,0 | id=859 kind=2 |
| #1636 | 186 | group[0] --> broker[0] | 148,148,50,12,0 | id=859 kind=2 |
| #1638 | 187 | worker[59] --> group[0] | 182,182,50,59,0 | id=1038 kind=2 |
| #1639 | 187 | worker[69] --> group[0] | 125,125,50,69,1 | id=1056 kind=2 |
| #1640 | 187 | worker[27] --> group[0] | 127,127,50,27,1 | id=1069 kind=2 |
| #1641 | 187 | group[0] --> broker[0] | 182,182,50,59,0 | id=1038 kind=2 |
| #1642 | 187 | group[0] --> broker[0] | 125,125,50,69,1 | id=1056 kind=2 |
| #1643 | 187 | group[0] --> broker[0] | 127,127,50,27,1 | id=1069 kind=2 |
| #1650 | 189 | worker[33] --> group[0] | 126,126,50,33,0 | id=786 kind=2 |
| #1651 | 189 | worker[97] --> group[0] | 156,156,50,97,0 | id=885 kind=2 |
| #1652 | 189 | worker[36] --> group[0] | 192,192,50,36,0 | id=1082 kind=2 |
| #1653 | 189 | worker[20] --> group[0] | 196,196,50,20,0 | id=1101 kind=2 |
| #1655 | 189 | worker[10] --> group[0] | worker to server:Task Req | id=1388 kind=1 |
| #1656 | 189 | worker[84] --> group[0] | worker to server:Task Req | id=1390 kind=1 |
| #1657 | 189 | group[0] --> broker[0] | 126,126,50,33,0 | id=786 kind=2 |
| #1658 | 189 | group[0] --> broker[0] | 156,156,50,97,0 | id=885 kind=2 |
| #1659 | 189 | group[0] --> broker[0] | 192,192,50,36,0 | id=1082 kind=2 |
| #1660 | 189 | group[0] --> broker[0] | 196,196,50,20,0 | id=1101 kind=2 |
| #1661 | 189 | group[0] --> broker[0] | 10 | id=1388 kind=1 |
| #1662 | 189 | group[0] --> broker[0] | 84 | id=1390 kind=1 |
| #1667 | 189 | broker[0] --> group[0] | 226,226,50,10,0 | id=1388 kind=5 |
| #1668 | 189 | broker[0] --> group[0] | 228,228,50,84,0 | id=1390 kind=5 |
| #1669 | 189 | group[0] --> worker[10] | 226,226,50,10,0 | id=1388 kind=5 |
| #1670 | 189 | group[0] --> worker[84] | 228,228,50,84,0 | id=1390 kind=5 |
| #1673 | 190 | worker[67] --> group[0] | 120,120,50,67,0 | id=759 kind=2 |
| #1674 | 190 | worker[99] --> group[0] | 164,164,50,99,0 | id=893 kind=2 |
| #1677 | 190 | worker[12] --> group[0] | worker to server:Task Req | id=1404 kind=1 |
| #1680 | 190 | group[0] --> broker[0] | 120,120,50,67,0 | id=759 kind=2 |
| #1681 | 190 | group[0] --> broker[0] | 164,164,50,99,0 | id=893 kind=2 |
| #1682 | 190 | group[0] --> broker[0] | 12 | id=1404 kind=1 |
| #1685 | 190 | broker[0] --> group[0] | 230,230,50,12,0 | id=1404 kind=5 |
| #1686 | 190 | group[0] --> worker[12] | 230,230,50,12,0 | id=1404 kind=5 |
| #1688 | 191 | worker[44] --> group[0] | 154,154,50,44,0 | id=883 kind=2 |
| #1690 | 191 | worker[93] --> group[0] | worker to server:Task Req | id=1412 kind=1 |
| #1691 | 191 | group[0] --> broker[0] | 154,154,50,44,0 | id=883 kind=2 |
| #1692 | 191 | group[0] --> broker[0] | 93 | id=1412 kind=1 |
| #1694 | 191 | broker[0] --> group[0] | 232,232,50,93,0 | id=1412 kind=5 |
| #1695 | 191 | group[0] --> worker[93] | 232,232,50,93,0 | id=1412 kind=5 |
| #1697 | 192 | worker[70] --> group[0] | 144,144,50,70,0 | id=855 kind=2 |
| #1700 | 192 | group[0] --> broker[0] | 144,144,50,70,0 | id=855 kind=2 |
| #1702 | 193 | worker[30] --> group[0] | 160,160,50,30,0 | id=889 kind=2 |
| #1703 | 193 | worker[37] --> group[0] | 178,178,50,37,0 | id=981 kind=2 |
| #1704 | 193 | worker[38] --> group[0] | 117,117,50,38,1 | id=1014 kind=2 |
| #1705 | 193 | worker[42] --> group[0] | worker to server:Task Req | id=1423 kind=1 |

圖表 17 文字介面輸出




圖表 18 圖型化輸出視窗

## 5.4 模擬實驗結果

### 5.4.1 Credit Damping 元件

以下為實驗的參數說明：

圖表 19 實驗參數

| | |
|---|---|
| credit_damping (CD) | 開啟(true)或關閉(false)減震元件 |
| CD_Value | 執行 credit damping 時所使用的δ值。 |
| worker_num | Worker 數量 |
| user_num | User 數量 |
| user_credit | User 的 credit 值 |
| task_number | 任務數量 |
| STANDARD_TASK_LENGTH | 任務長度 |

本論文的實驗的目的是要驗證減震方法對執行不同長度的工作的效益。如**錯誤! 找不到參照來源。**所示，我們執行三種工作長度和四種減震參數共 12 組實驗。

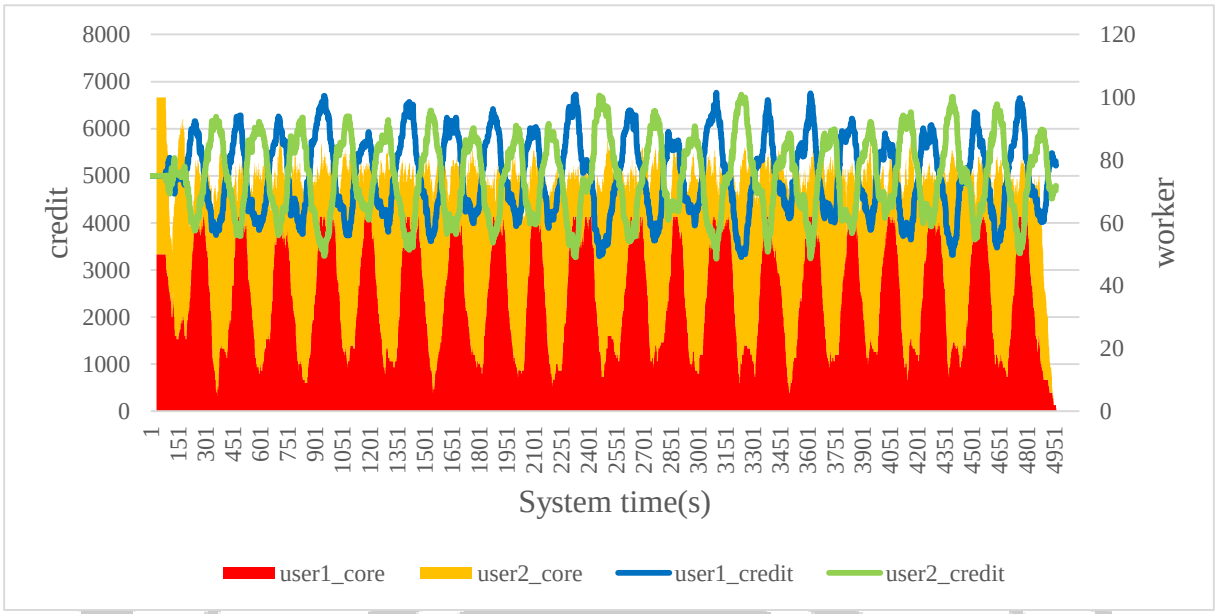
圖表 20 不同工作長度(L1~L3)對各種減震參數(C1~C4)的實驗參數設定

| L1 STANDARD_TASK _LENGTH =50 | L2 STANDARD_TASK _LENGTH =80 | L3 STANDARD_TASK _LENGTH=150 |
|---|---|---|
| L1C1: CD 關閉 | L2C1: CD 關閉 | L3C1: CD 關閉 |
| L1C2: CD_value=15 | L2C2: CD_value=30 | L3C2: CD_value=50 |
| L1C3: CD_value=10 | L2C3: CD_value=20 | L3C3: CD_value=35 |
| L1C4: CD_value=5 | L2C4: CD_value=10 | L3C4: CD_value=20 |

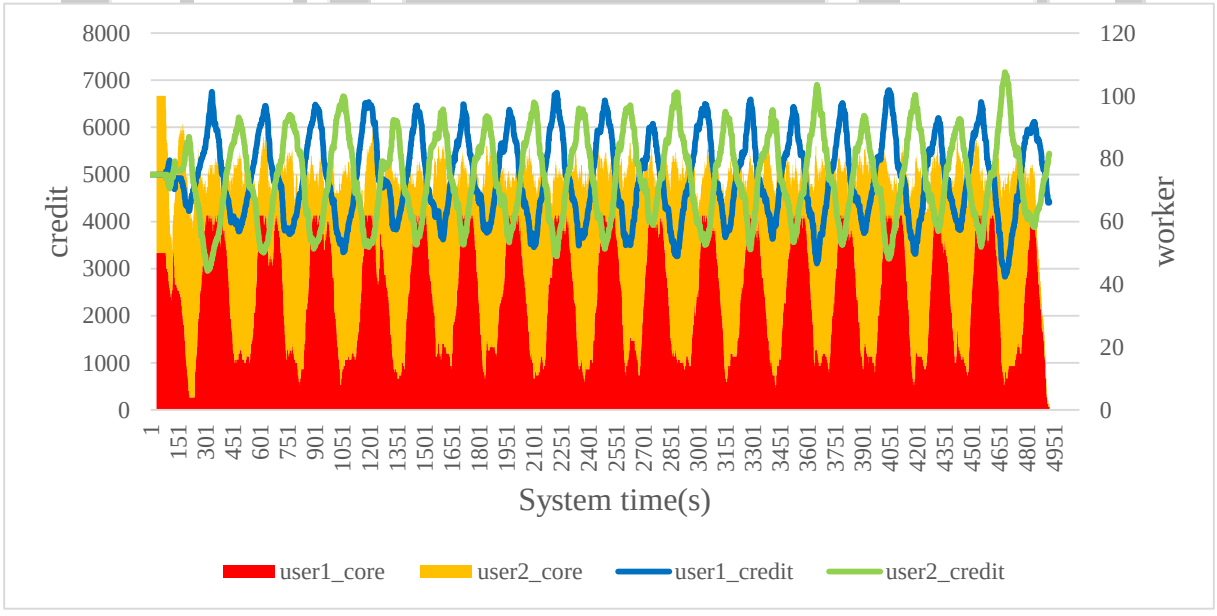
以下是我們的實驗結果，預設條件除了工作長度與減震參數，其餘皆為固定參數，Worker_num=100 、User_num=2 、User_credit=5000(預設使用者的 credit 值) 、

Task_number=5000，以上為我們的預設條件，首先說明，各 user 的 credit 值預設為 5000，可以供 broker 做為資源優先權的參考。

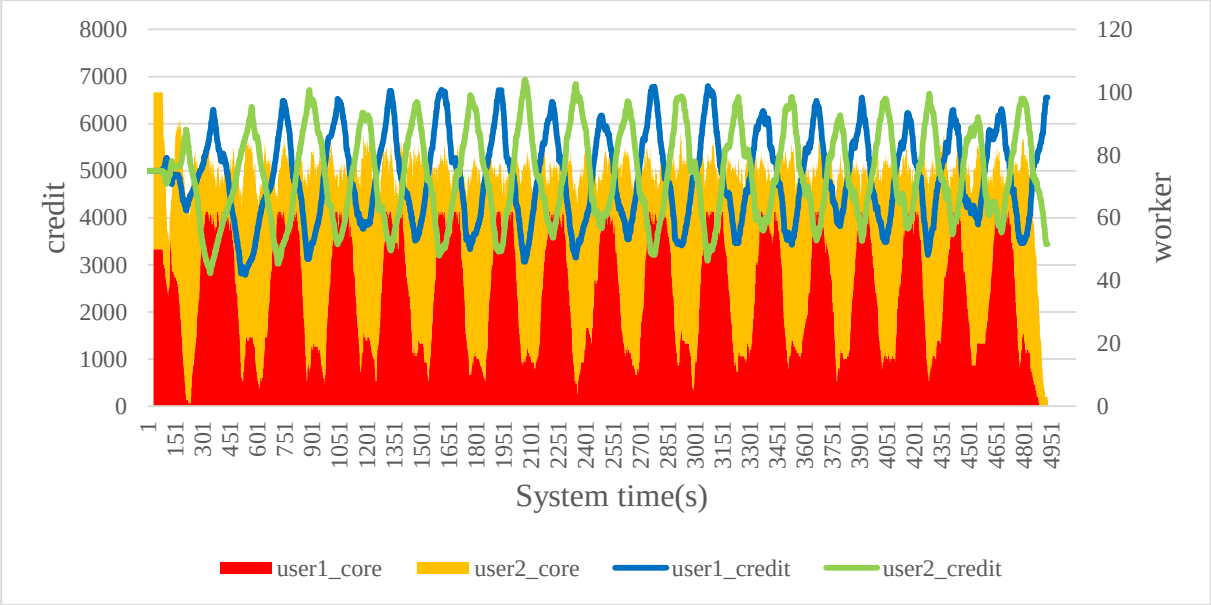
以下實驗對於工作長度為 50 的任務做 C1-C4 的實驗：



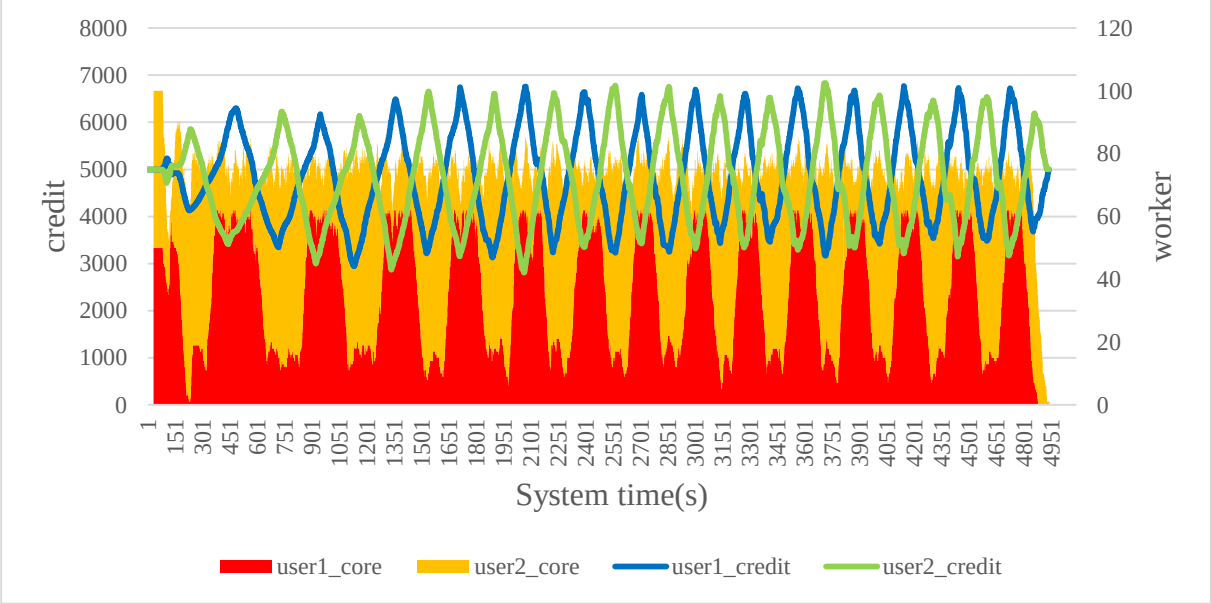

圖表 21 實驗 L1C1：工作長度=50、CD 未開啟




圖表 22 實驗 L1C2：工作長度=50、CD_value=15



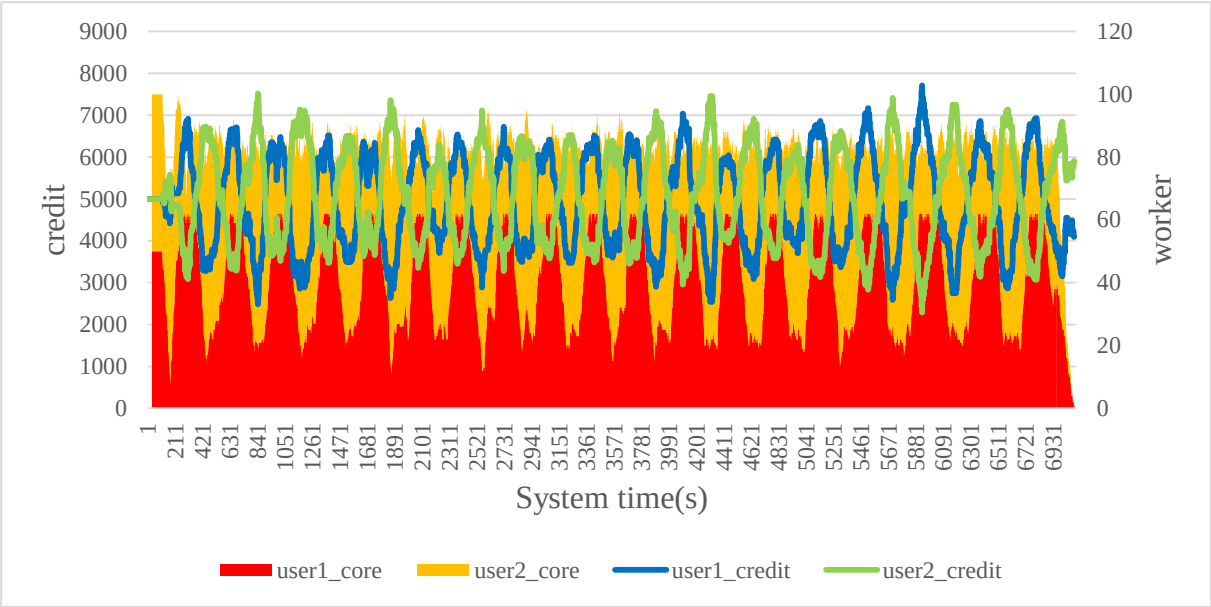

圖表 23 實驗 L1C3：工作長度=50、CD_value=10



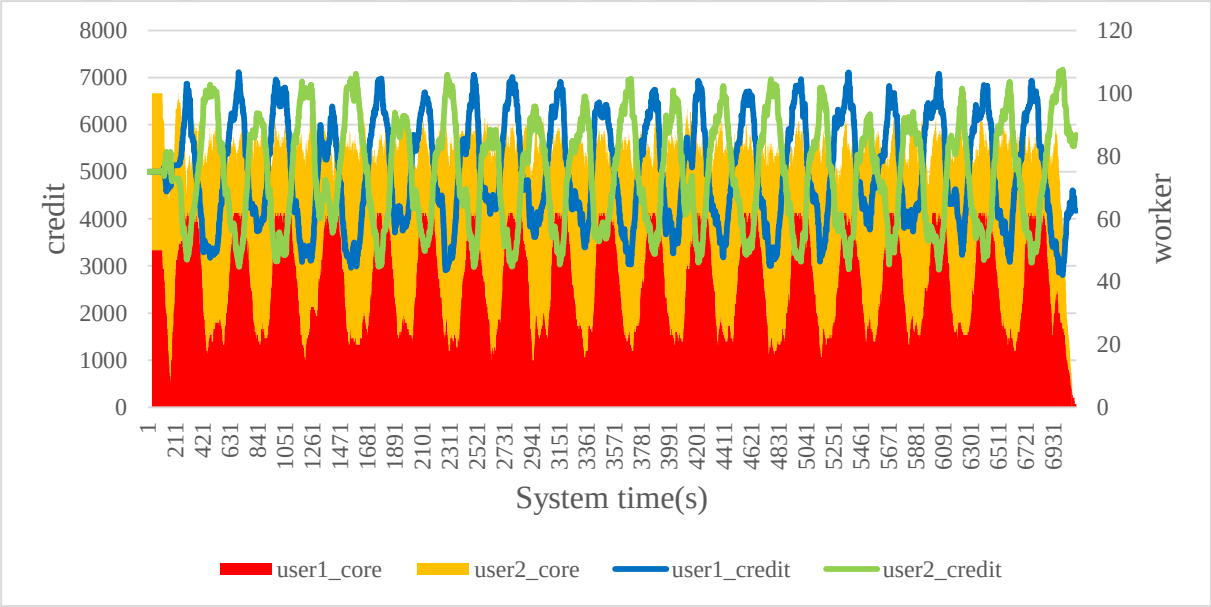

圖表 24 實驗 L1C4：工作長度=50、CD_value=5

此次實驗對於短工作，工作長度為 50 的任務進行 CD 開啟與未開啟的實驗，當 credit_damping_value 越低時，更容易趨緩資源震盪的情況。

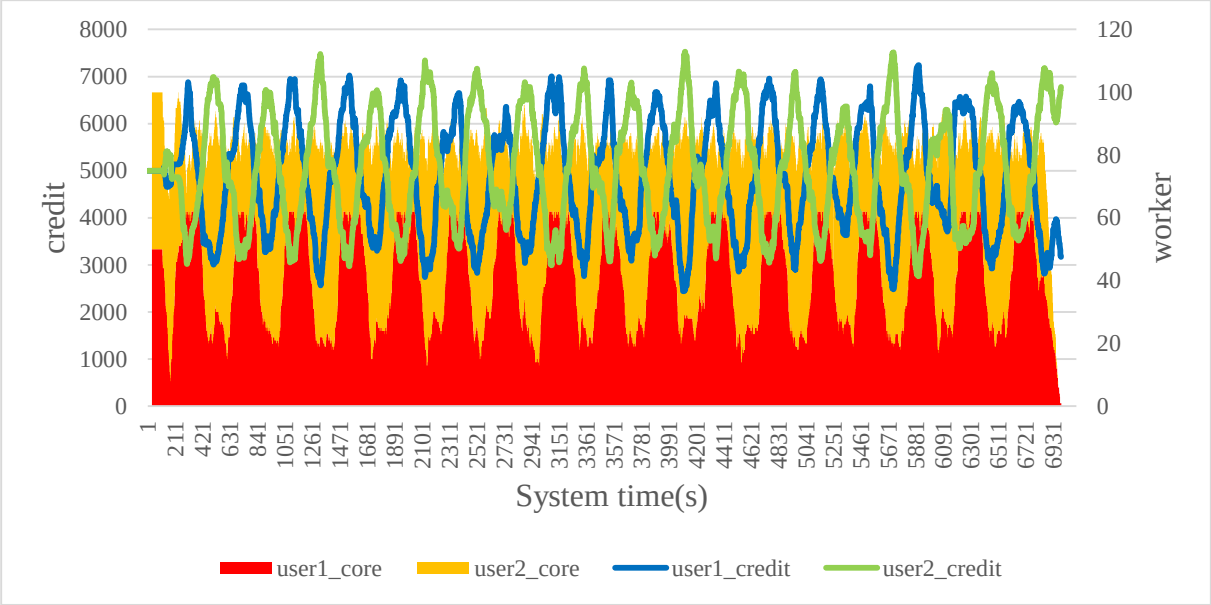
以下實驗對於工作長度為 80 的任務做 C1-C4 的實驗：



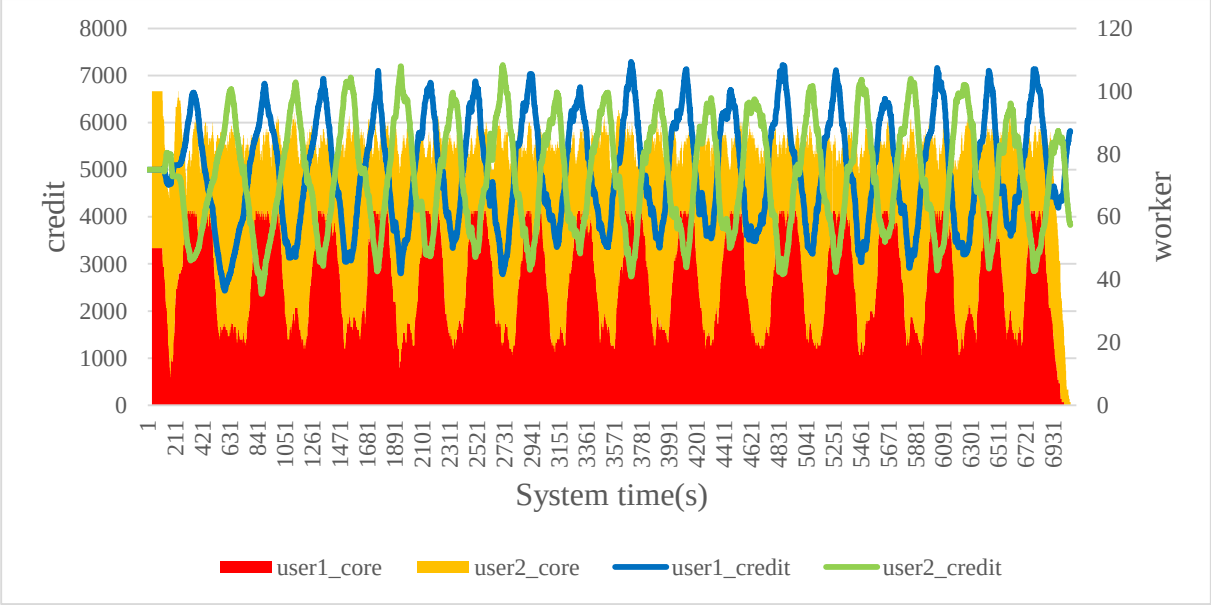

圖表 25 實驗 L2C1：工作長度=80、CD 未開啟




圖表 26 實驗 L2C2：工作長度=80、CD_value=30



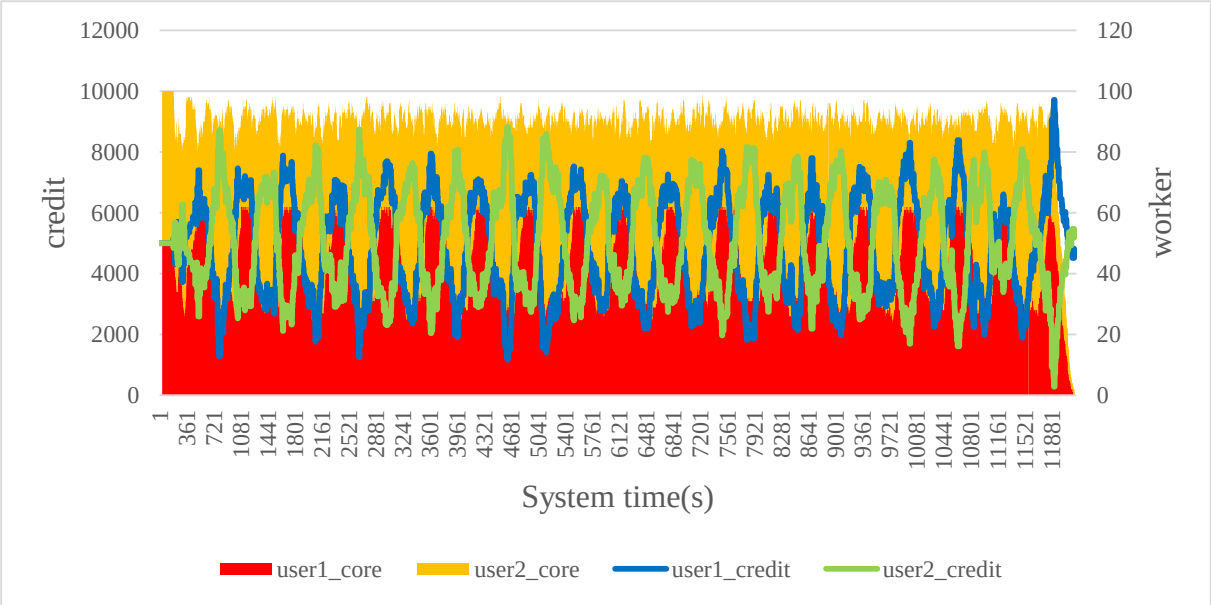

圖表 27 實驗 L2C3：工作長度=80、CD_value=20



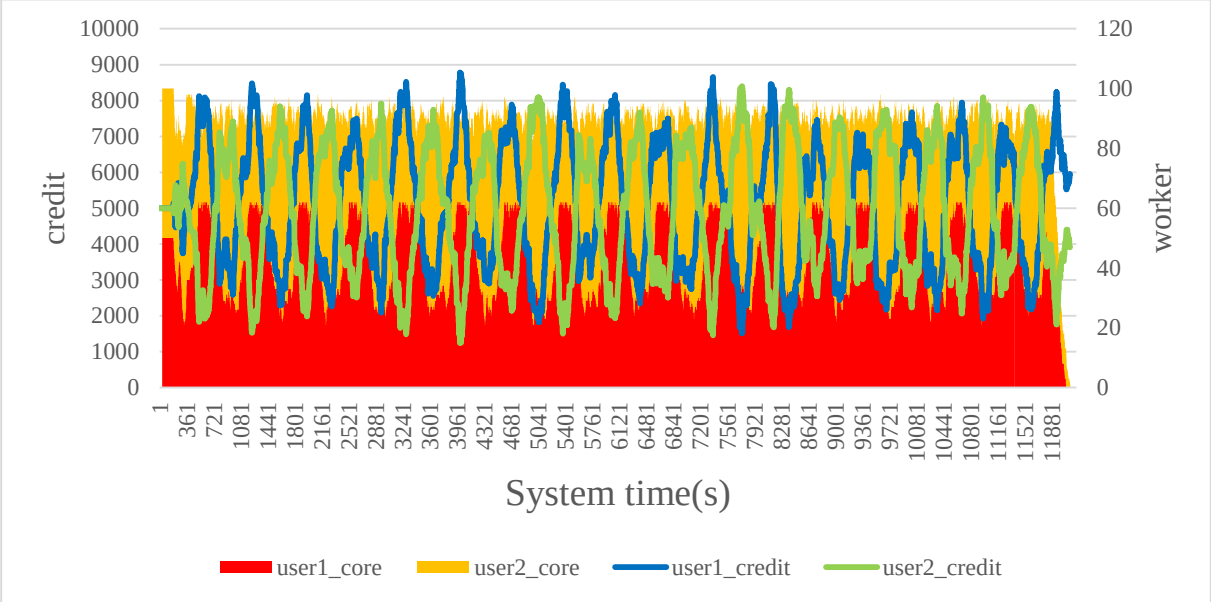

圖表 28 實驗 L2C4：工作長度=80、CD_value=10

此次實驗為對於短工作較長的工作，工作長度為 80 的任務進行 CD 開啟與未開啟的實驗，當 credit_damping_value 越低時，會趨緩資源震盪的情況。但是效果卻沒有遠先的那麼好。

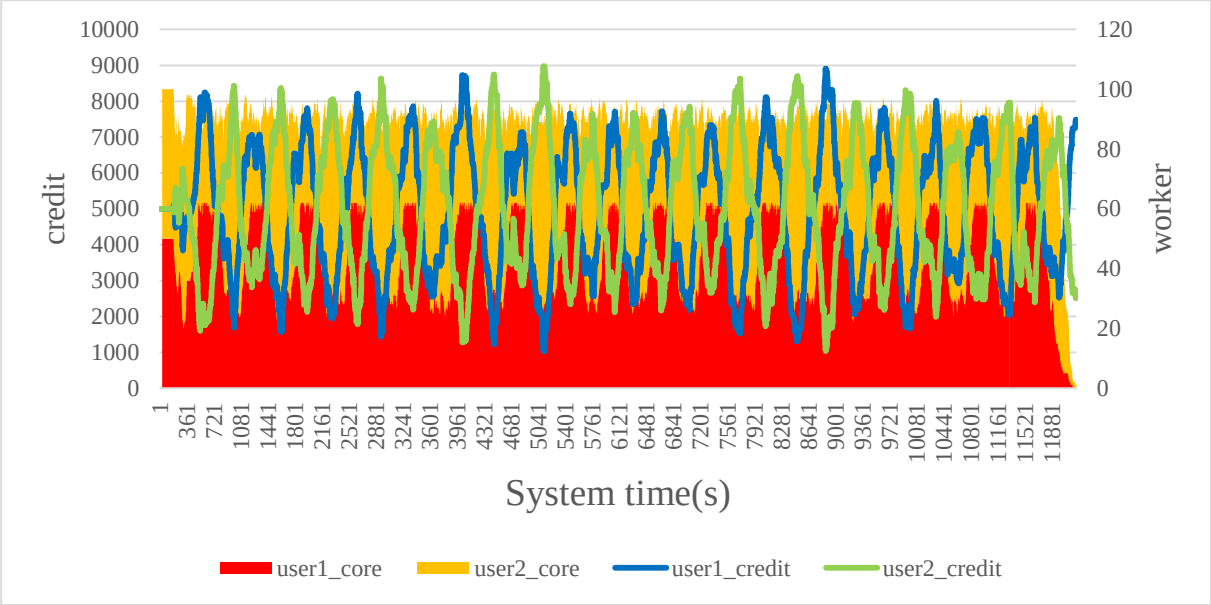
以下實驗對於工作長度為 150 的任務做 C1-C4 的實驗：



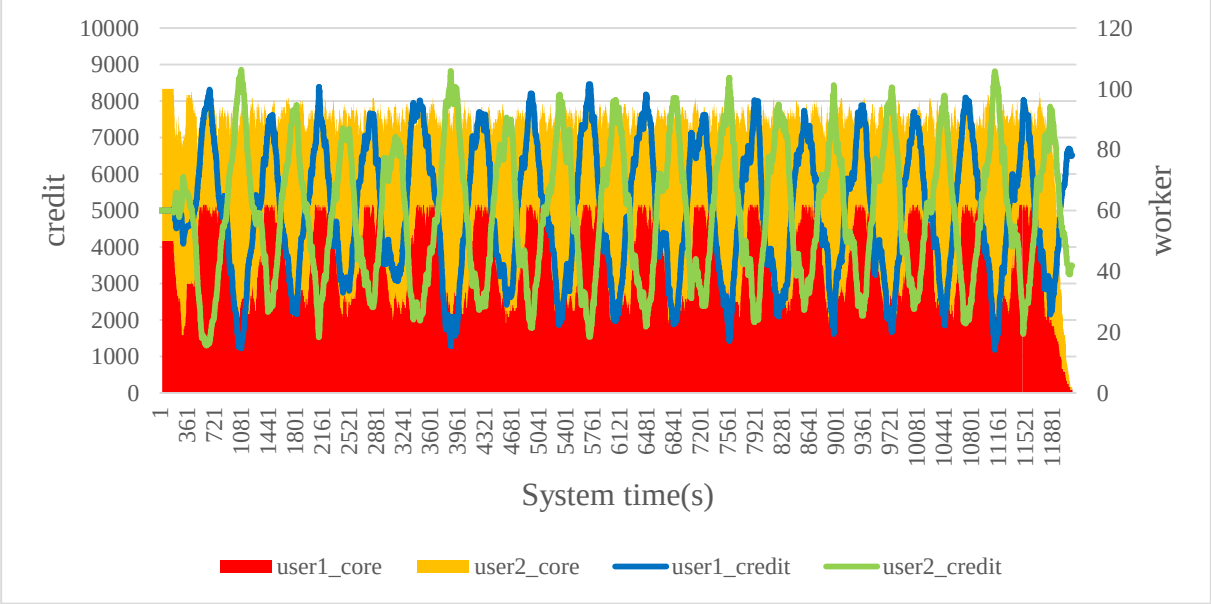

圖表 29 實驗 L3C1：工作長度=150、CD 未開啟




圖表 30 實驗 L3C2：工作長度=150、CD_value=50




圖表 31 實驗 L3C3：工作長度=150、CD_value=35




圖表 32 實驗 L3C3：工作長度=150、CD_value=20

此次實驗為對於工作長度為 80 較長的工作，工作長度為 150 的任務進行 CD 開啟與未開啟的實驗，當 credit_damping_value 越低時，會趨緩資源震盪的情況。但是由於資源震盪的幅度只有 30 台 worker 左右，所以可以發現，當工作時間越長時，震盪情況沒有改善，但是震幅卻大幅下降，使得資源更加穩定的分配。

圖表 33 不同工作長度(L1~L3)對各種減震參數(C1~C4)的震盪次數

| L1<br>STANDARD_TASK<br>_LENGTH =50 | L2<br>STANDARD_TASK<br>_LENGTH =80 | L3<br>STANDARD_TASK<br>_LENGTH=150 |
|---|---|---|
| L1C1: 21 | L2C1: 20 | L3C1: 20 |
| L1C2: 19 | L2C2: 20 | L3C2: 19 |
| L1C3: 17 | L2C3: 18 | L3C3: 18 |
| L1C4: 15 | L2C4: 18 | L3C4: 18 |

### 5.4.2 Credit 即時更新

以下為實驗的參數說明：

圖表 34 實驗參數

| | |
|---|---|
| credit_update | 開啟(true)或關閉(false)credit 即時更新 |
| worker_num | Worker 數量 |
| user_num | User 數量 |
| user_credit | User 的 credit 值 |
| task_number | 任務數量 |
| STANDARD_TASK_LENGTH | 任務長度 |

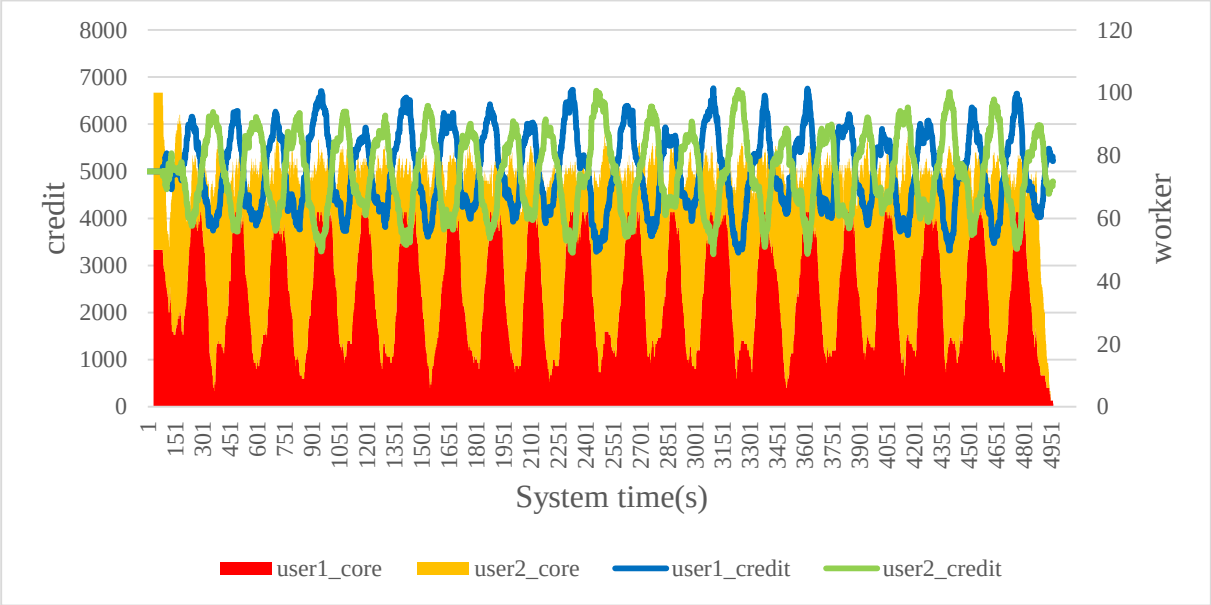
此次實驗主要是實驗 credit 即時更新，是否可以使任務發配更穩定及公平。以下是我們的實驗結果，預設條件除了工作長度與 credit_update 參數，其餘皆為固定參數，Worker_num=100 、User_num=2 、User_credit=5000(預設使用者的 credit 值) 、Task_number=5000，以上為我們的預設條件，首先說明，各 user 的 credit 值預設為 5000，可以供 broker 做為資源優先權的參考。

本論文的實驗的目的是要驗證減震方法對執行不同長度的工作的效益。如圖表 34 所示，我們執行三種工作長度和兩種減震參數共 6 組實驗。

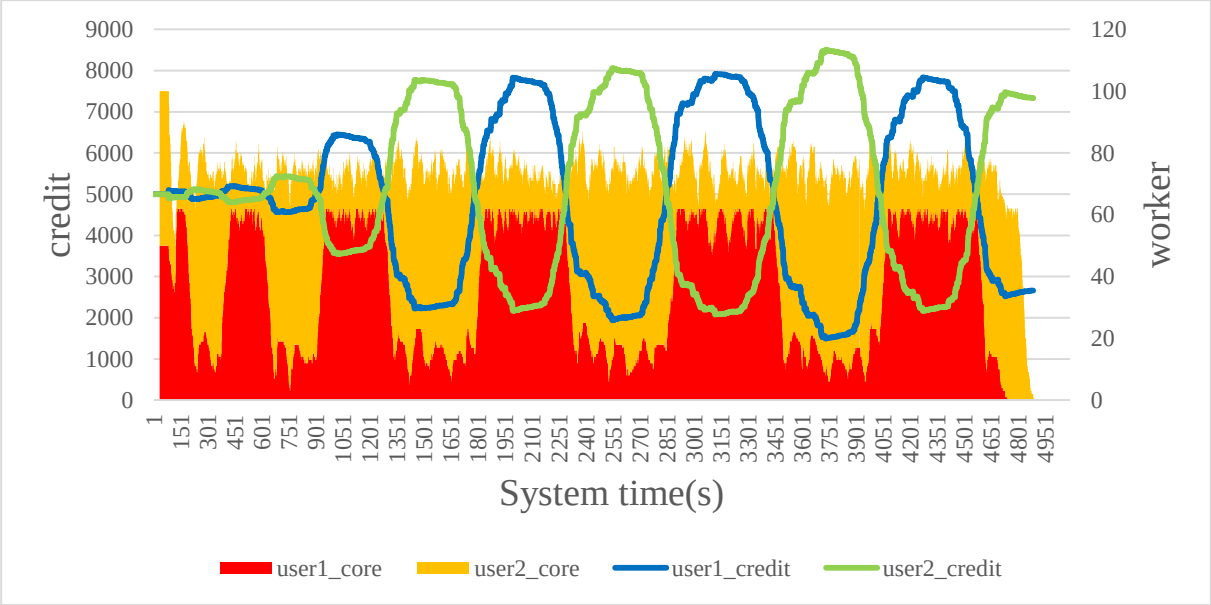
圖表 35 不同工作長度(L1~L3)對各種減震參數(U1~U2)的實驗參數設定

| L1<br>STANDARD_TASK<br>_LENGTH =50 | L2<br>STANDARD_TASK<br>_LENGTH =80 | L3<br>STANDARD_TASK<br>_LENGTH=150 |
|---|---|---|
| L1U1: Credit_update 關閉 | L2U1: Credit_update 關閉 | L3U1: Credit_update 關閉 |
| L1U2: Credit_update 開啟 | L2U2: Credit_update 開啟 | L3U2: Credit_update 開啟 |

以下實驗對於工作長度為 50 的任務做 U1-U2 的實驗：

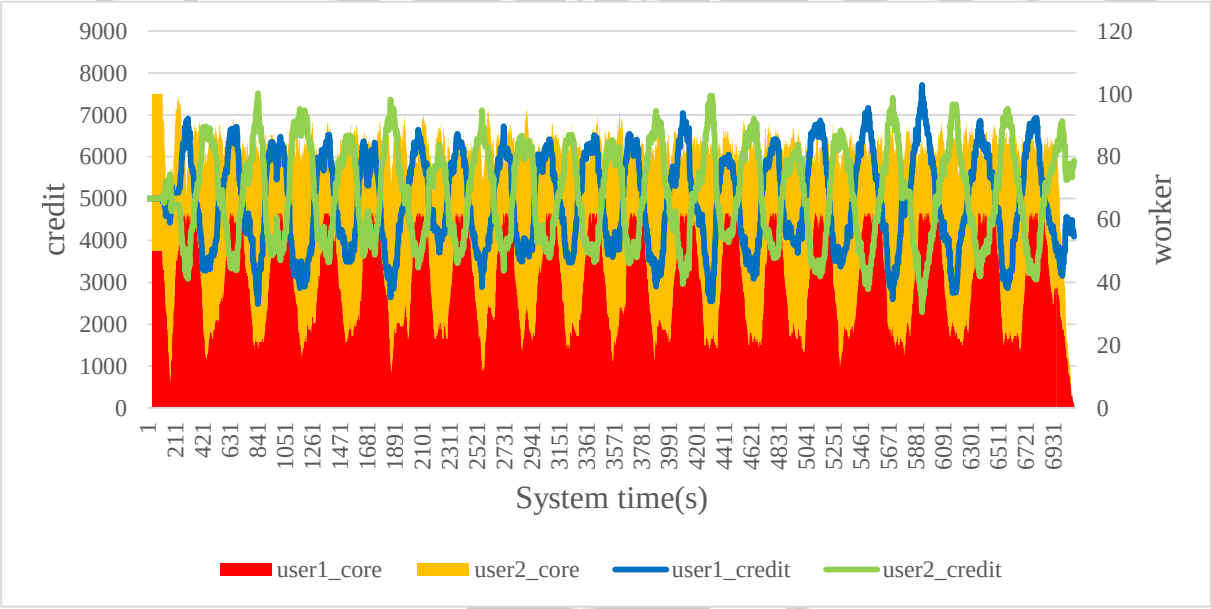



圖表 36 實驗 L1U1：工作長度=50、Credit_update 關閉

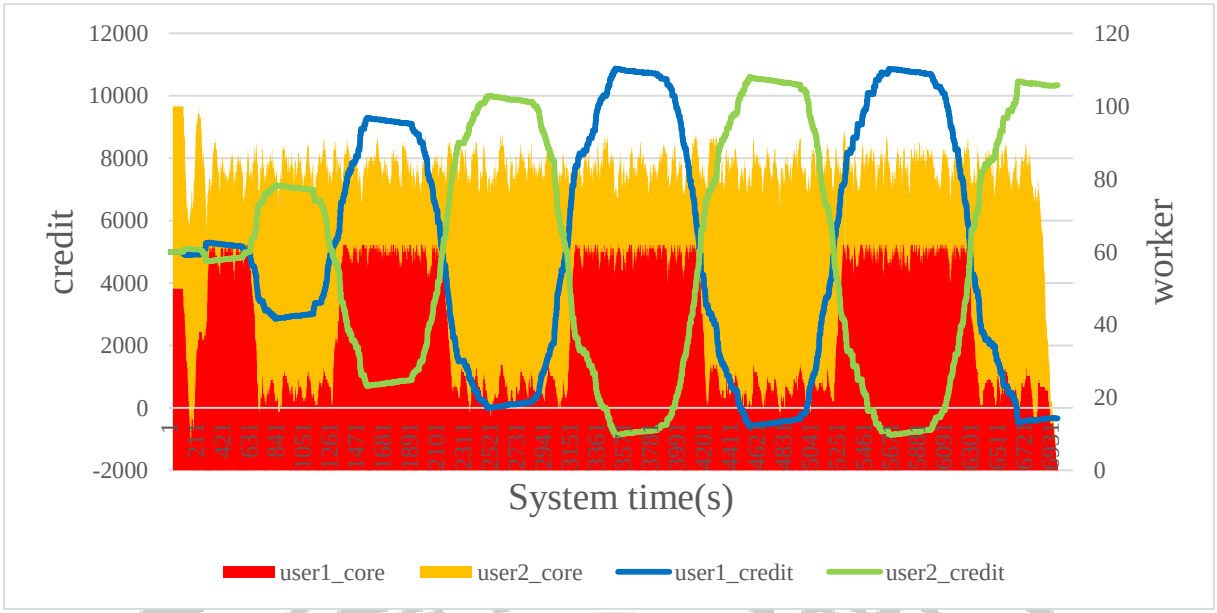



圖表 37 實驗 L1U2：工作長度=50、Credit_update 開啟

以下實驗對於工作長度為 80 的任務做 U1-U2 的實驗：



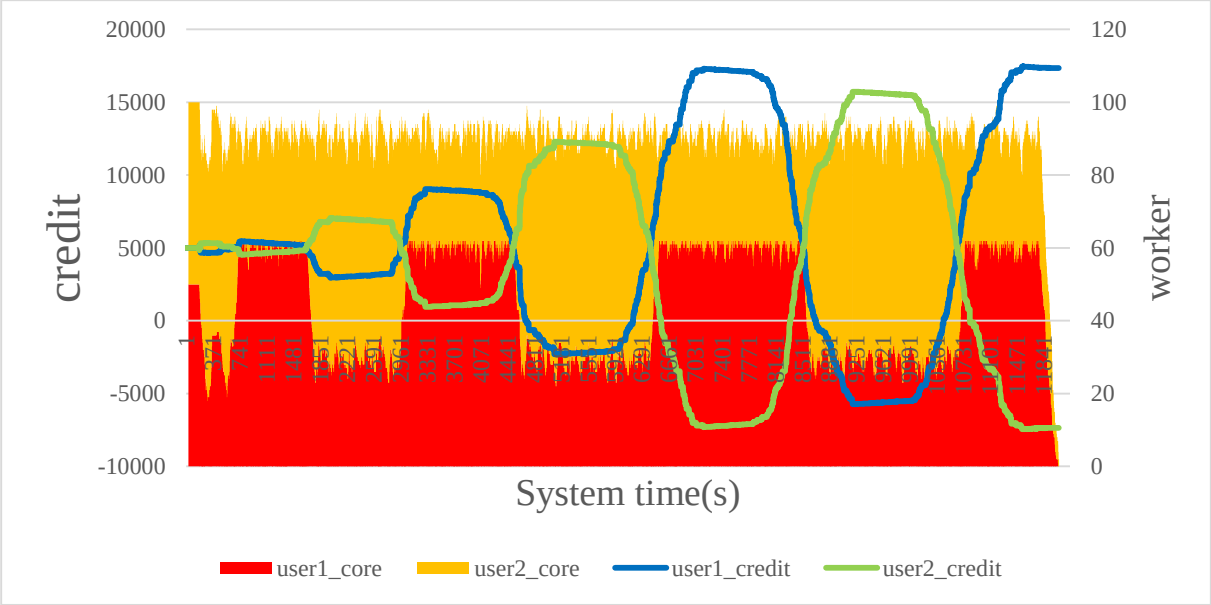

圖表 38 實驗 L2U1：工作長度=80、Credit_update 關閉




圖表 39 實驗 L2U2：工作長度=80、Credit_update 開啟

以下實驗對於工作長度為 150 的任務做 U1-U2 的實驗：


圖表 40 實驗 L3U1：工作長度=150、Credit_update 關閉

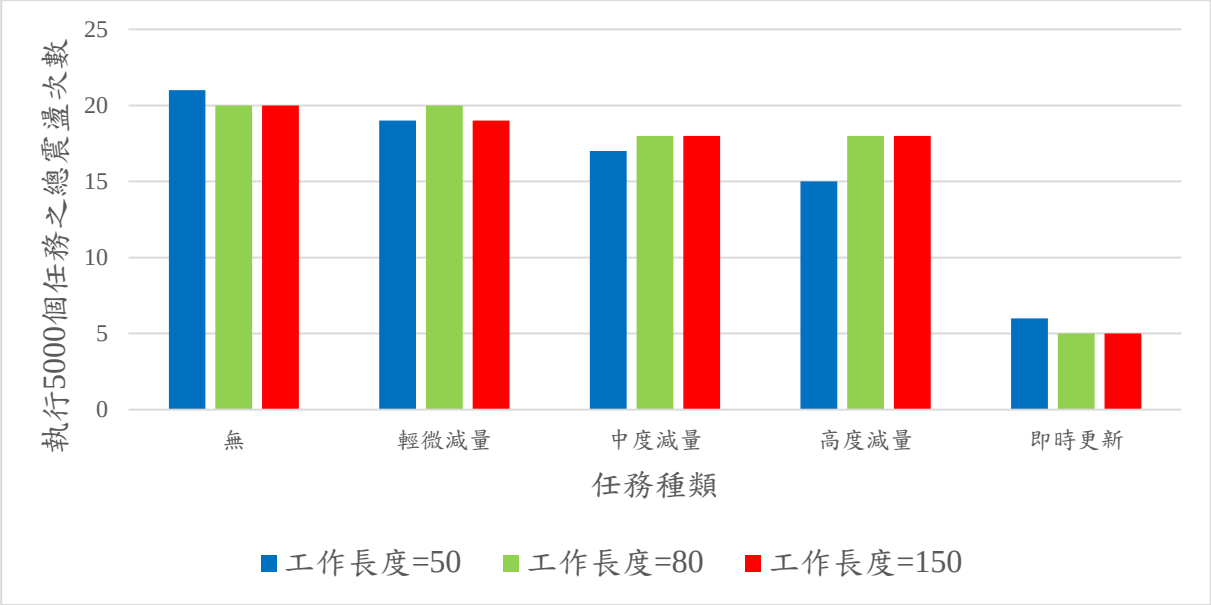



圖表 41 實驗 L3U2：工作長度=150、Credit_update 開啟

圖表 42 不同工作長度(L1~L3)對各種減震參數(U1~U2)的震盪次數

| L1<br>STANDARD_TASK<br>_LENGTH =50 | L2<br>STANDARD_TASK<br>_LENGTH =80 | L3<br>STANDARD_TASK<br>_LENGTH=150 |
|---|---|---|
| L1U1: 21 | L2U1: 20 | L3U1: 20 |
| L1U2: 6 | L2U2: 5 | L3U2: 5 |

觀察於立即更新的部分，因為採取非傳統回傳的方法來做減震。立即更新不只在時間上，不需要等待，可以馬上做 credit 更新，也可以將組織間的貢獻用量以及使用資源量做平衡，使得減震非常明顯。




圖表 43 執行 5000 個任務之總震盪次數比較圖

圖表 43 主要表示，工作長度不同時，對於震盪的影響，還有 credit-damping 的減量就是相較於原本更新的量要再低，當減少的量越多就等於是中度或高度減量，使得 credit 更新的量較為緩慢，還有採取及即時更新的方法。

# 六、 結論與未來工作

本文討論的桌機網格聯盟使企業透過資源共享來解決大規模的應用程式計算。然後我們設計了新的分配演算法，確保了公平與穩定的資源分配。實驗結果證明我們的方法可以減少震盪，使得系統能穩定的分配資源並提升效能。

本實驗稍有分流概念卻未納入整個系統，以求焦點放置在減振的元件，再作其他方面的精進，未來會增加多群 Group 的 Server 來進行資料量的分流實驗以達到有效的減輕 IO 瓶頸限制。

# 參考文獻


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

[2] 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.

[3] Foster, I., Kesselman, C. *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann Publishers, Inc., 1999.

[4] Taiwan UniGrid website, available at http://www.unigrid.org.tw/info.html

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

[6] 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

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

[8] Wu, I.C., Han, S.Y., “The Study of the Worker in a Volunteer Computing System for Computer Games”, Institute of Computer Science and Engineering College of Computer Science NCTU, 2011.

[9] Speedup, “http://en.wikipedia.org/wiki/Speedup”, 2013.

[10] Laxmikant V. Kale,´ Sameer Kumar, Jayant DeSouza, “A Malleable-Job System for Timeshared Parallel Machines”, 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid, 2002.

[11] TWGrid ,”http://www.twgrid.org/cht/index.php”,2015

[12]張元耀，桌機網格聯盟之資源分配管理，國立交通大學資訊工程研究所碩士論文，2013

[13]Viktors Bertis, Raphaël Bolze, Frédéric Desprez, Kevin Reed,” From Dedicated Grid to Volunteer Grid: Large Scale Execution of a Bioinformatics Application”,2009