東海大學資訊工程研究所

碩士論文

指導教授：陳隆彬博士

適用於具優先權遊戲樹搜尋工作之HPCaaS排程方法

A New Scheduling Approach for Game Tree Search Tasks with Priorities in HPCaaS Platform

研究生：林家瑋

中華民國一百零五年六月

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 林 家 瑋 所提之論文

適用於具優先權遊戲樹搜尋工作之 HPCaaS 排程方法

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

_____

王經篤

_____

指 導 教 授 _____ 簽章

中華民國 105 年 6 月 22 日

摘要

在各種領域的研發都需要用到超級電腦的時代，無論是科學研究或是工商業界都需要高速計算來研發具競爭力的產品與服務。各國無不持續挹注經費來建置高效能計算服務，期能維持科技的創新能力與持續發展。高效能計算系統根據一些排程方法來分配資源給多樣化應用，但這些排程方法在分配的過程中，通常都會產生一些資源碎片，本文將探討使用這些資源碎片的策略和技術。我們提出執行平行蒙地卡羅樹狀搜尋（MCTS）的輕量級單核心工作，去適應不穩定的碎片資源環境。我們的這套系統傾向於將重要的樹節點分配到可靠的計算節點上運行，來確保重要的樹節點優先被探索和順利完成。我們的實驗結果成功的將MCTS計算融合在碎片資源環境中並且獲得有效的資源。

關鍵詞：高效能計算、排程方法 、資源碎片、平行蒙地卡羅樹狀搜尋

Abstract

Resource fragmentation often occurs in modern high performance computing systems due to the complex scheduling criteria. This thesis addresses both of the policy and technical issues of using unstable resources formed by fragmentation. We propose that the parallel Monte-Carlo tree search (MCTS) can be implemented as lightweight single-core tasks to adapt to the unstable fractured resources. Our system tends to assign the promising tree nodes to the reliable and responsive processors to confirm the best-first search. This thesis demonstrates a successful integration in which the MCTS computation gains significant resources.

Keyword：High performance computing, scheduling criteria, resource fragmentation, Monte-Carlo tree search

目錄

圖目錄

表目錄

一、 簡介

現代的高效能計算的工作類型趨於多樣化。運用高效能處理器來執行平行工作，是超級電腦的傳統核心價值，然而近年興起的需求更多是來自於巨量雲端運算與智慧計算等應用。傳統高速平行計算工作被歸類為 moldable task，它們使用固定數目的處理器核心，而且一旦執行就不能變動。而雲端計算的NoSQL資料庫、map reduce、大數據分析、或是遊戲計算AI等工作通常包含巨量的單核心輕量工作，稱為malleable task，這類工作處理持續產生的時序相關或是網路節點收集傳回的資料，彈性地使用處理器來進行計算。

傳統的昂貴的高速計算叢集是一個封閉的環境，平行的moldable task必須指定所需的處理器數量。有很多原因讓處理器閒置，常見原因如系統會收集並保留資源，直到能夠執行某個大型工作為止，被保留的資源就會閒置一段時間，使得系統的資源使用率無法有效地提升。如何讓資源使用率提高，並且能讓各種應用程式都能取用這些資源，就成了受注目的議題。

HPCaaS (High-Performance computing as a service，高效能計算即服務) 是一種將計算資源服務化的概念，透過SOA(服務導向架構)，提供高速計算資源給更多的使用者。

複本技術在malleable task排程系統中扮演關鍵角色，系統會將同一個工作或資料製作R個複本來分配給R個不同主機。像是NoSQL資料庫或是BOINC桌機網格等系統都使用這個機制。在這類系統中，降低R值會減少複本量而增加工作產能，因為同一批主機能做更多不同工作或儲存更多不同資料。另一方面，增加R值則能提升工作或資料可靠度。因此，排程器必須隨時監測系統效能，並適時降低或是增加R值來適應系統可用資源的變化並維持效率。先前學者提出的R值調整演算法是使用爬坡演算法trial and error，隨機增減R值並藉以觀察系統效能指標會變好或是變壞，並以爬坡演算法來回調整多次才會跟上系統的實際變化。

本論文先期研究發現，爬坡演算法無法適應系統劇烈變化。更有甚者，兩個時間點的系統效能指標不具可比性，因為兩個時間點的環境條件不會相同，這造成爬坡演算法在爬坡時無法正確評估效能是變好還是變壞。本論文中，我們發展創新的malleable task排程，我們將效能指標區分為R-同向與R-反向兩類，利用一組正反指標之間的差異，來明確定位出R值要調整的方向與幅度。我們預期本計畫所提出的差動排程方法將能改善舊的方法的缺點，對於建置高效能計算服務將大有助益。

本論文採用 OMNet++模擬整個運行流程，以達到資源可以高效率運用的目的。後續章節安排如下：第二章介紹背景與相關技術，第三章則講述 HPCaaS 排程之工作排程，第四章說明 HPCaaS 之遊戲樹搜尋工作排程方法，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

HPCaaS 名為 High-Performance Computing as a Service 是種高效能計算作為服務的概念，希望能將高效能計算相關的技術轉化成為一種服務。在高校能計算系統中，會根據一些排程方法分配資源給使用者，但這些排程方法在分配的過程中，通常會產生資源碎片，本論文將探討使用資源碎片執行遊戲術搜尋工作的策略與技術，把桌機網格的 worker 運行在高效能計算叢集的資源碎片中，以充分運用碎片資源執行遊戲樹搜尋工作。在本章中，將在 2.1 節先介紹 HPCaaS，在 2.2 節介紹高效能計算叢集，在 2.3 節介紹桌機網格，最後在 2.4 節介紹遊戲樹搜尋工作。

2.1 HPCaaS

HPCaaS 名為 High-Performance Computing as a Service 是種高效能計算作為服務的概念，希望能將高效能計算相關的技術轉化成為一種服務。隨著越來越多人使用高效能系統應用程式服務，新系統需要同時讓多個使用者服務以及應用程式來使用。傳統的 HPC 系統只提供限時的服務給應用程式來避免佔用持續資源，但 HPCaaS 擁有全新概念，使用叢集伺服器和儲存當資源池，使用者能透過網路介面將工作要求傳送至系統，且利用智能排程機制同時排程多個不同的應用程式，提高系統整體的生產力及改善資源的使用率。

圖 2-1 顯示一個 HPCaaS 系統，使用者應用(user application)透過網路服務連結到 submission host 來提交工作。使用者應用可以是傳統平行工作、3D 圖形計算、或是 map reduce 雲端計算工作。使用者也可以透過 submission host 查詢系統狀態。查詢可以用網路服務服務完成，或是利用 GUI 介面來完成。圖 2-1 中，Master host 負責各種工作的排程並派送工作到計算節點 workers，工作主機 worker 是實際執行工作的電腦。

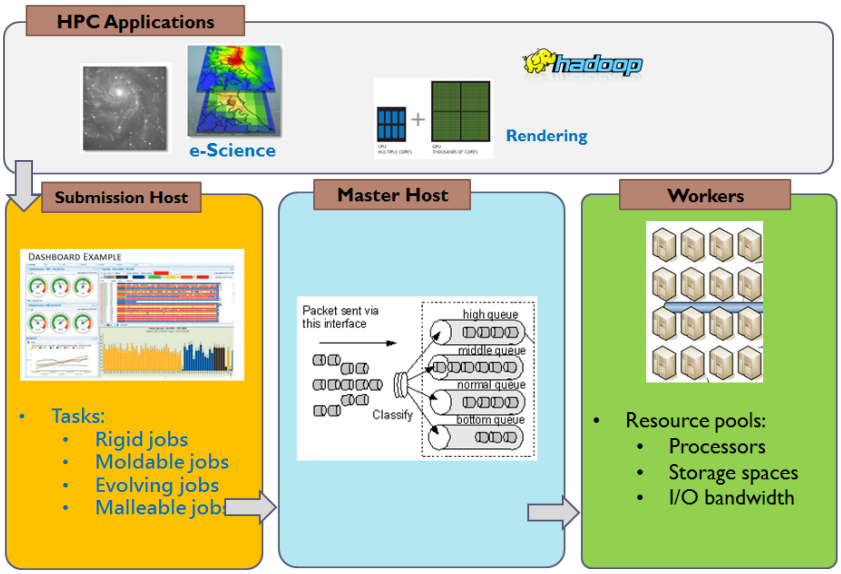

圖 2-1 HPCaaS 示意圖

2.2 高效能計算叢集

本論文使用 IBM Load sharing facility (LSF) cluster 為實驗平台。圖 2-2 顯示 LSF 的架構圖，它是一套排程器，用於 HPC(高效能計算)的平台上，使用者 Job Submission 到 LSF 的 Master Host 上，再經由排程器來將工作丟置 Queue 上，LSF 會監控每個計算節點，當有空閒的機器時就會將 Queue 上面的工作分配出去，計算完的工作會回傳給 Master Host。LSF 採用的排程大多是 FCFS(First Come First Service)的方式，也有 Service level agreement(SLA)、Fairshare scheduling、Preemption，以及 Backfilling 等等，工作的類型也是 Rigid 的工作為主。

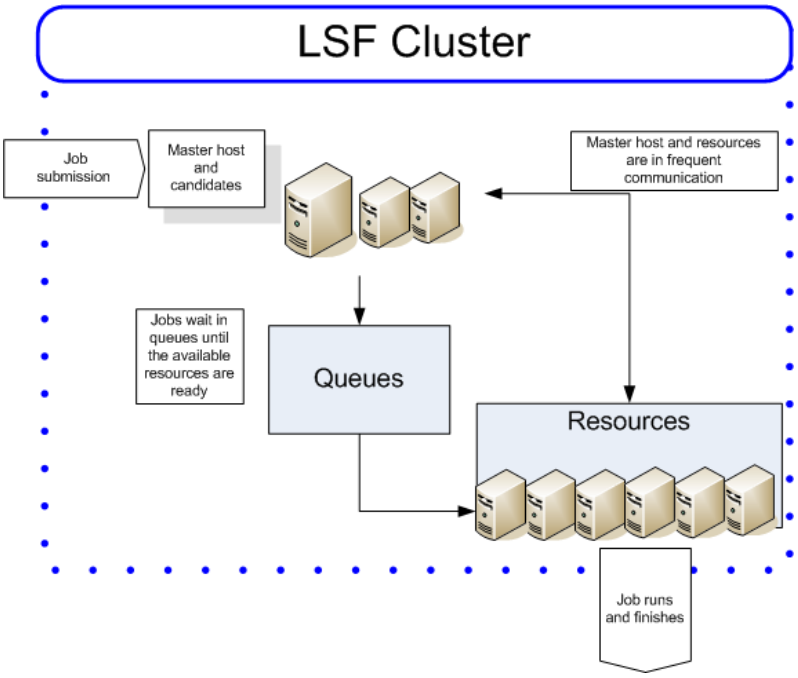

圖 2-2　LSF 叢集計算排程器

2.3 桌機網格(DG)

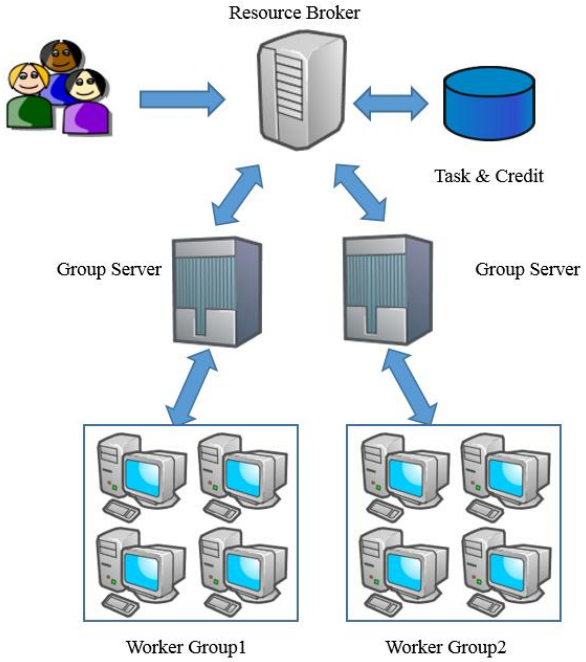

圖 2-3　桌機網格系統架構

桌機網格(Desktop Grid)的系統架構如下表所示。架構包含 user、worker group、group server、broker、和 credit database 等五部分。

表 2-1　桌機網格架構說明

Worker	工作主機，負責執行工作的電腦。
Worker Group	工作主機群組，指的是一群收到同一個工作的複本的 worker。群組內 worker 各自獨立，互不通訊，只跟 broker 聯繫。群組是依據 broker 指派複本而形成，複本執行完成即不屬於群組。
Task	應用程式工作。
Resource Broker	排程伺服器，負責製作工作複本和並派送給 worker 的電腦。broker 也負責接收複本執行結果並決定一個工作是否完成。
Task & 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。

桌機網格的循序圖如圖 2-4 所示，流程圖如圖 2-5 所示，桌機網格的工作流程可以分為以下四個主要部分：

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。

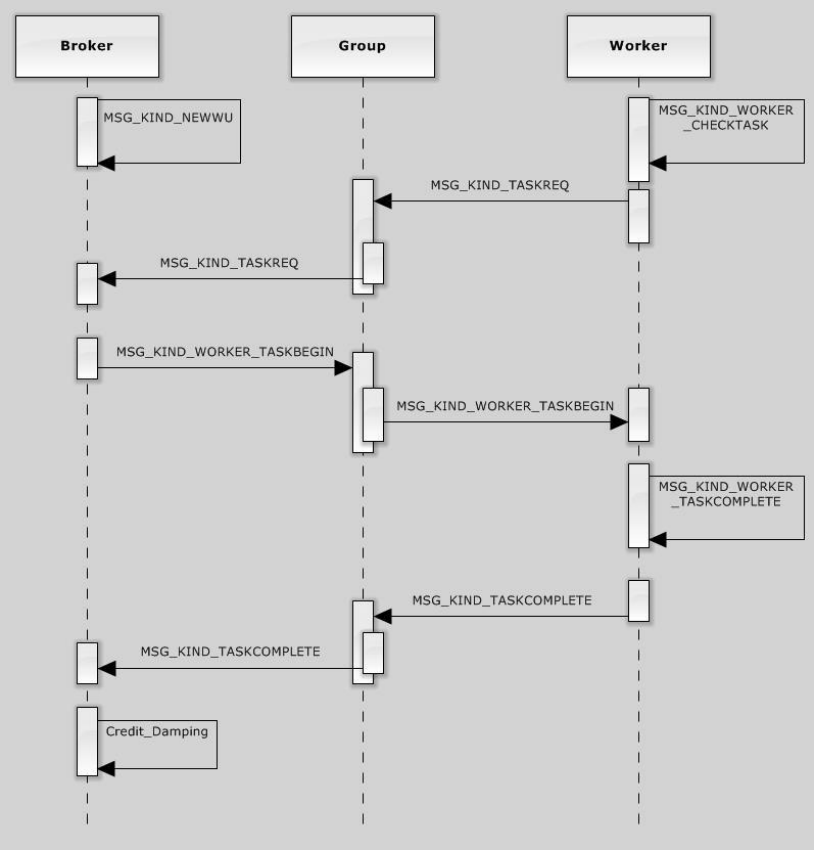

圖 2-4 桌機網格循序圖

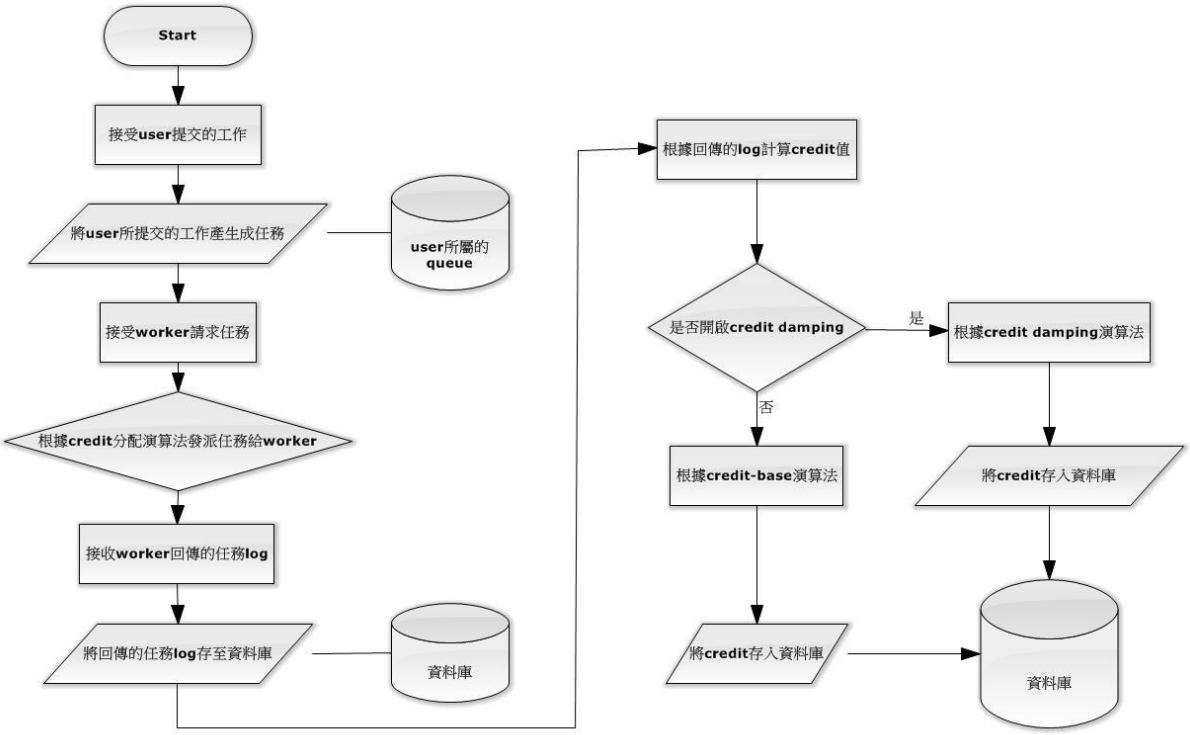

圖 2-5　桌機網格工作流程

2.4 遊戲樹搜尋工作

本論文研究蒙地卡羅樹狀搜尋 Monte Carlo Tree Search(MCTS)平行工作。MCTS 是一種建構在隨機模擬的搜尋樹方法，分為四個階段，分別為選點(selection)、展點(expansion)、隨機模擬(playout)和更新(backpropagation)，如下圖所示。selection 是從根節點開始往下選點直到 leaf；expansion 是將選到的 leaf 的某一可選步作展開；playout 是對新展開的樹節點做模擬，對一個樹節點的模擬即為一個 MCTS 的工作；最後的 backpropagation 是將模擬結果更新至搜尋樹上。在本篇論文中，我們將在資源碎片上運行 MCTS 的工作。

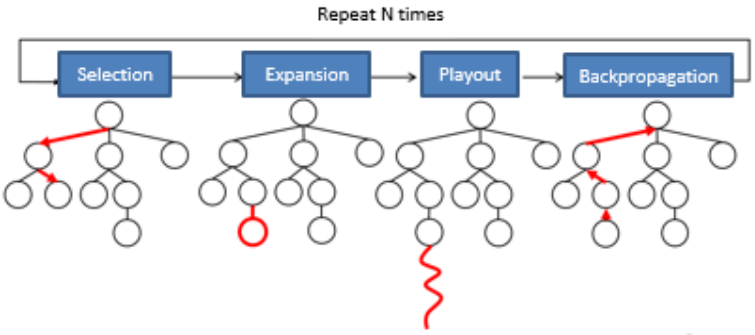
Repeat N times
Selection
Expansion
Playout
Backpropagation

圖 2-6　MCTS

三、 HPCaaS 排程之工作排程

複本技術在 malleable task 排程系統中扮演關鍵角色，系統會將同一個工作或資料製作多個複本來分配給多個不同主機，當它們大多數回報相同結果時該工作才算完成。像是 NoSQL 資料庫、peer-to-peer 計算、或是桌機網格 (desktop grid) 等系統都使用這個機制。本章中，3.1 節說明 HPCaaS 平台的工作類型，3.2 節說明非可靠 worker 計算環境之多數決投票機制的工作複本策略，最後，3.3 節探討可靠 worker 計算環境之至少一票的工作複本策略。

3.1 HPCaaS 平台的工作類型

HPCaaS 擁有極大系統可塑性，在過去每個應用程式都有擁有自己所屬的系統資源可供使用，但 HPCaaS 能讓資源被動態分配給每個工作，大幅度提高資源的使用率，這也是高效能至雲端計算中重要的一個環節。

HPCaaS 平台的工作類型通常可歸納為四類

(1). 剛性工作 (Rigid task)

Rigid 工作在提交時，需要指定固定數量的資源，而且事後不可修改。因為設計容易，rigid 工作是最普遍的一種，但是這種工作因為缺乏彈性而容易引發碎片。以圖 3-1 來作舉例，假設黃色區塊是 rigid 工作，當它被派送進來運行時，會因為前面還有有其他工作正在運行，導致沒有足夠的處理器資源來運行 rigid 工作，必須等待其他工作結束釋放資源才能開始運行,不只造成等待的時間過久,也產生圖 3-2 所謂的碎片(斜線)，

造成整體系統的使用效率不佳。

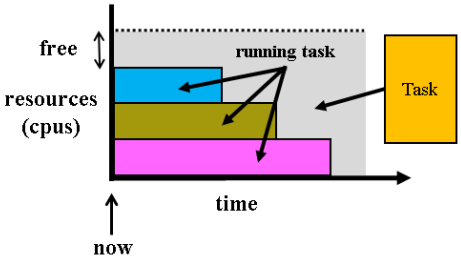

圖 3-1　提交工作範例

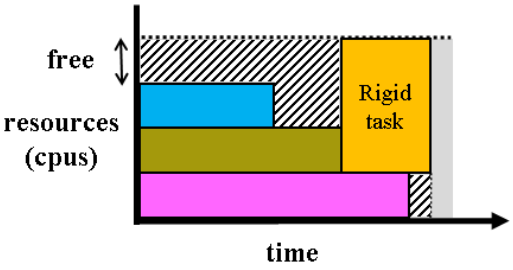

圖 3-2　硬性工作範例

(2). 可塑性工作 (Moldable task)

Moldable 工作，相比 rigid 工作，多了一些彈性，它的處理器數量是一個範圍。moldable 工作使用的資源是事先指定，由排程器依照系統現況在指定範圍內動態給定，但是排程後直到執行結束仍然不能變動 (有些文獻是定義成執行階段可以減少所使用的資源)。以圖 3-3 作舉例，碎片的部分和 rigid 相比，明顯減少很多，但是因為 moldable 在程式執行後不能被改變，所以就算其他工作完成釋放出資源，moldable 還是只能照著程式啟動時指定的數量運行，系統的使用率還是有一個很大空間來讓其他閒置處理器資源處於被使用的狀態。

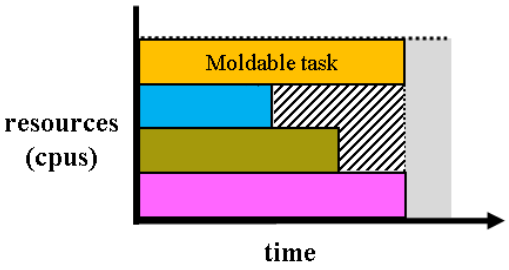

圖 3-3　可塑性工作範例

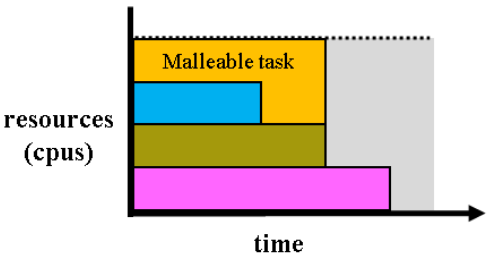

圖 3-4　延展性工作範例

(3). 進化性工作 (Evolving task):

Evolving 工作可以改變分配資源數量，在執行中各個階段，會擁有不同數量的處理器。evolving 工作本身會提出處理器的需求，像是索取或放棄處理器，藉此來控制執行過程中處理器數量。每個 evolving 工作 都設置一個至少最低額度運行的處理器數量。

(4). 延展性工作 (Malleable task):

傳統型工作需要定義所需的處理器數量，malleable 卻可以動態改變處理器數量，不只是在程式排程器和啟動時，也可在執行過程時改變所需處理器數量，免除使用者必須指定處理器數量的麻煩。只要有其他工作完成，釋放出處理器的資源，malleable 工作就可以利用那些資源，縮短原本工作預計的完成工作時間。反之當有更重要工作到達時，malleable 工作可以放棄手上處理器給排程器運用，如圖 3-4 所示。

3.2 非可信 worker 之工作複本策略

桌機網格為志願計算模型，採用公眾匿名參與者的 PC 作為 worker，屬於不可信的計算環境；本文採用 LSF 高效能計算叢集的機器作為 worker，屬於可識別機器的可信計算環境。

在不可信的計算環境中，worker 不是全部可靠的，有可能回傳錯誤的答案，所以定義 worker 可靠度是由 worker 在某個時間區間內工作結果正確率所決定，公式如下：

$$r_i = \frac{n_i + 1}{N_i + 1}$$

r_i 代表第 i 個 worker 的可靠度，n_i 代表第 i 個 worker 做過正確的工作數，N_i 代表第 i 個 worker 總共做過的工作數。

為了適應不可信的 worker 計算環境，我們將一個 malleable 工作複製成多份，派送到一組 worker 上運算，藉由一組 worker 的能力增加工作的完成率。因為 worker 不是全部可信的，所以我們利用多數決的方式決定答案是否正確。判斷一組 worker 的可靠度方法，是計算一組 worker 中，超過半數個 worker 是可靠的機率，公式如下：

$$\lambda(G) = \sum_{m=k+1}^{2k+1} \sum_{\{\varepsilon:\|\varepsilon\|=m\}} \prod_{i=1}^{2k+1} r_i^{\varepsilon_i} * (1 - r_i)^{1-\varepsilon_i}$$

G為一組包含 2k+1 個互相獨立的 worker，注意，這邊使用奇數個 worker，以防止模棱兩可的多數投票。$\lambda(G)$為 G的可靠度，r_i 表示第 i 個 worker W_i 的可靠度，$\varepsilon = \{\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_{2k+1}\}$ 是反映 2k+1 個 worker 的一組向量。ε_i 為 0 表示W_i 回傳錯誤答案，ε_i

為 1 表示W_i 回傳正確答案。$\|\varepsilon\|$ 表示向量ε 中"1"的個數。m 表示 G中達到半數以上的個數。

例如，$G=\{W_1,W_2,W_3\}$，workers 的可靠度r_i 為(0.9, 0.9, 0.7)，m 為 2 和 3，$\varepsilon=\{(0,1,1),(1,0,1),(1,1,0),(1,1,1)\}$ ， $\lambda(G)=(1-0.9)*0.9*0.7+0.9*(1-0.9)*0.7+0.9*0.9*(1-0.7)+0.9*0.9*0.7 \ = \ 0.936$。因此，如果我們將一個工作送到$G$ 上執行，將有 0.936 的機率會完成，也就是有超過半數相同的正確答案。此公式的複雜度為$O(2^{2k})$。

3.3 可信 worker 之工作複本策略

桌機網格的 worker 允許公眾匿名參與者貢獻他們的計算資源作為大型電子科學項目的解決方法，由於參與者是公眾且自願的，因此需要另外探討安全和可靠度等等的議題。本論文以 IBM LSF 叢集系統為研究實作對象。叢集系統的 worker 跟一般桌機網格的 worker 不同，是叢集系統組織內可識別的高可信度 worker，所以當 worker 回傳完成的工作時，只要有一個 worker 回傳完成的工作即確認工作完成。

在本篇論文中，由於我們考慮的 worker 都是可信的，判斷 worker 的可靠度可以有很多種方法，例如：將 worker 空閒的時間比例當作可靠度，或是判斷 worker 忙碌和空閒的切換頻率當作可靠度等等。在本篇論文中，我們是將 worker 在某時間內資源空閒的比例當作 worker 的可靠度，公式如下：

$$r_i(s,t)=\frac{idle_i(s,t)}{t-s}$$

我們定義W_i 代表第 i 個 worker；$r_i(s,t)$ 代表W_i 由時間點 s 到時間點 t 的可靠度，可靠度的值介於 0 到 1 之間。為了底下說明方便，我們將用r_i 來代替 $r_i(s,t)$；$idle_i(s,t)$為W_i 由時間點 s 到時間點 t，CPU 平均的資源空閒時間。當W_i 都是空閒時，可靠度為 1，當W_i 越忙碌時，則可靠度將會越低，以此公式計算 worker 的可靠度。

在本篇論文中，只要至少一個worker回傳答案即可，因此我們將公式修改如下：

$$\lambda(G) = \sum_{\{\varepsilon:\|\varepsilon\|\geq 1\}} \prod_{i=1}^{2k+1} r_i^{\varepsilon_i} * (1-r_i)^{1-\varepsilon_i}$$

$$= 1 - \left(\sum_{\varepsilon=(0,0,\ldots,0)} \prod_{i=1}^{2k+1} r_i^{\varepsilon_i} * (1-r_i)^{1-\varepsilon_i} \right)$$

$$= 1 - \prod_{i=1}^{2k+1} (1-r_i)$$

因為在 G裡的worker是兩兩互相獨立，我們可以將 $\lambda(G)$設為1減去所有W_i 的不可靠度相乘。例如，G包含兩個worker，分別是W_1 和W_2，W_1 的可靠度為0.5，W_2 的可靠度為0.6，則$\lambda(G) = 1-(1-0.5)*(1-0.6) = 0.8$。當 G再加入可靠度為0.7的W_3 後，則$\lambda(G) = 1-(1-0.5)*(1-0.6)*(1-0.7) = 0.94$。由此可以看出，在 G中加入越多worker時，$\lambda(G)$ 的值會越大，且值會介於0到1之間。在本篇論文中，將以此修改後的公式計算一組worker的可靠度。此公式的複雜度為 $O(k)$。

四、 HPCaaS 之遊戲樹搜尋工作排程方法

本論文先期研究發現，爬坡演算法無法適應系統劇烈變化。更有甚者，兩個時間點的系統效能指標不具可比性，因為兩個時間點的環境條件不會相同，這造成爬坡演算法在爬坡時無法正確評估效能是變好還是變壞。

4.1 資源碎片與工作排程系統

在高效能計算系統中，會根據一些排程方法來分配資源給使用者，但這些排程方法在分配的過程中，通常都會產生一些資源碎片，本文將探討使用資源碎片的策略和技術。我們提出以實現在平行蒙地卡羅樹狀搜尋（MCTS）的輕量級單核心任務，去適應不穩定的碎片資源環境。

假設每個節點有 12 核心數，當某個使用者需要執行 48 核心數的工作時，他需要四個計算節點，而在執行此工作之前，此使用者必須等待其它計算節點上正在執行的工作完成，湊齊 4 個計算節的計算資源(12x4)才有辦法執行 48 核心數的工作，在這個等待的過程中，勢必會出現沒有使用到的資源造成節點 idle，這些閒置等待被使用的資源即是資源碎片。

因為無法準確預知每個使用者派送工作的時間，且不易預估每個工作執行時間的長短，造成資源碎片出現的時間既不穩定也不連續，所以很難去使用資源碎片。本文藉由分配新的工作到這些資源碎片上來提高資源的使用率。

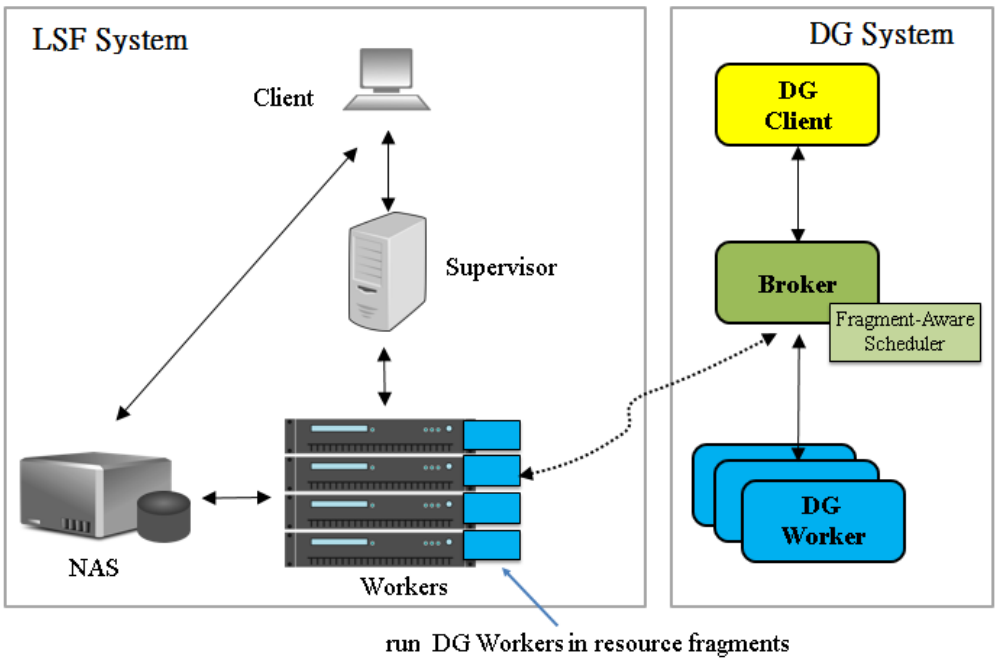

圖 4-1　系統架構

上圖為系統架構，左邊是 LSF 系統，右邊是桌機網格系統，我們會將 DG 的 worker 運行在 LSF worker 的資源碎片中，以充分運用碎片資源執行遊戲樹搜尋工作。

4.2 效能衡量指標

在動態的碎片環境中，我們將派送桌機網格(DG)的 worker 到碎片上運行，為了不影響一般系統使用者的工作，當一般系統使用者有工作要運算時，DG worker 會隨時中斷結束，造成正在 DG worker 上運行的 MCTS 工作失敗，所以我們運用工作複本的技術來解決這個問題，將一個 MCTS 工作複製成多份，派送到不同的 worker 上運行，以減少工作失敗的機率。

工作完成的時間可以有很多種定義，例如，所有工作複本完成的加總時間，或是只計算工作有在執行的時間加總等等。在本篇論文中，定義工作完成時間為C_j，代表從工作送出到完成第一個工作複本的時間；定義標準工作完成時間為S_j，代表完成第一個工作複本所需的時間，如下圖所示。

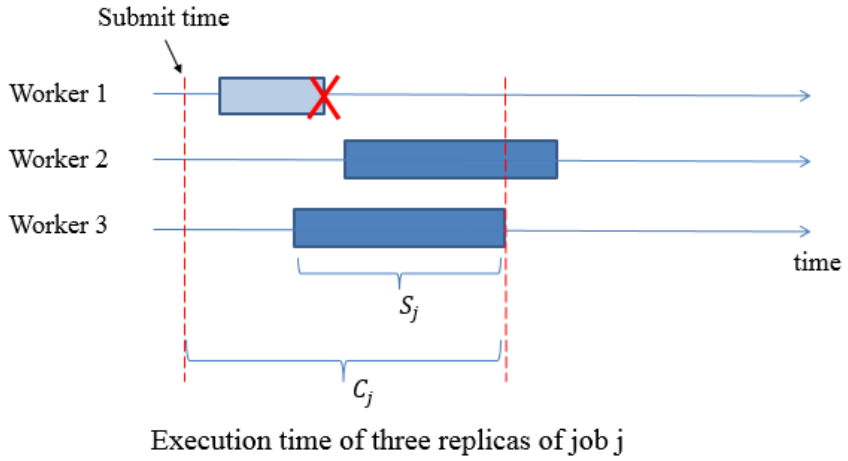

圖 4-2　工作的完成時間C_j和標準完成時間S_j

在本論文中，有兩項衡量標準，分別是產量和加權效率，是用來判斷系統效益的指標。產量(throughput)縮寫為 TH，TH 是某時間區段內所有完成工作的標準完成時間 S_j加總值；加權效率(weighted efficiency)縮寫為 WE，WE 是統計某個時間區段內由每個工作的重要性比重以及工作所花費的時間比例加乘總合，代表全部工作在標準時間完成的比

例。當每個工作都是在標準時間完成($C_j = S_j$)，加權效率為 1，當有工作超過標準時間完成，加權效率將會下降，當工作的重要性越高，影響加權效率的值就越大。

加權效率和產量是相對的，當想要提升產量時，勢必要降低複本的個數，這樣一來，工作被踢掉重做的機率會上升，就會相對增加工作的完成時間，加權效率將會降低。

4.3 演算法

我們定義 malleable 工作 j 的門檻值 ($threshold_j$) 為工作 j 的重要性乘上一個係數：

$$threshold_j = min\{R * \theta_j, 1\}$$

R 是一個大於零的常數；θ_j為第 j 個 malleable 工作的重要性，θ_j的值介於 0 到 1 之間。在本論文中，θ_j越大代表工作越重要，在其他應用中，θ_j可以是不同的定義。當 R*θ_j>1 時，設$threshold_j$=1，使得$threshold_j$的值介於 0 到 1 之間。

門檻值$threshold_j$是用來判斷 worker 群組 G 的整體可靠度是否滿足工作 j 的門檻值，亦即

$$\lambda(G) \geq threshold_j$$

若以上公式成立則我們定義 worker 群組 G 能夠被第 j 個工作所接受，並考慮將第 j 個工作分別派送到此群組中的 worker。

當給定工作 j 和它的門檻值$threshold_j$，本論文採用 First-Fit 演算法來建構能滿足該門檻值的群組 G。First-Fit 演算法會將 worker 依照可靠度排隊在 priority queue 中，每一

次排程器會從最大的 worker 開始挑選並加入 G，直到 r(G)大於$threshold_j$。例如，假設有六個 worker，可靠度分別為 0.7、0.7、0.5、0.4、0.4 和 0.3，$threshold_j$為 0.85。首先選取可靠度最高的 worker 為 0.7，但是不滿足工作的門檻值 0.85，於是按照排序把第二順位的 worker(可靠度 0.7)加入 G，判斷是否滿足門檻值 0.85，於是最後挑出兩個可靠度為 0.7 的 worker。

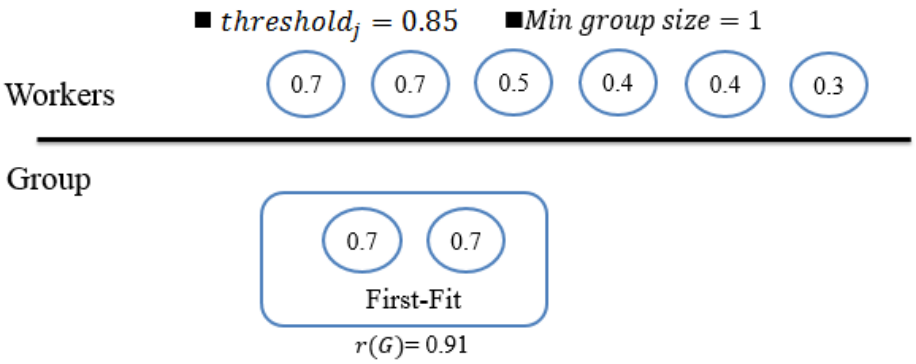

圖 4-3　演算法圖示

First-Fit 演算法如下：

Algorithm First-Fit(G worker list, j job, $threshold_j$)

1: Set $i = 0$

2: Sort G

3: **while** $i < |G|$ **do**

4: Add i worker into the set s from G

5: **if** r(s) exceeds $threshold_j$

6: **break**

7: **else**

8: $i = i + 1$

9: **end if**

10: **end while**

11: **if** such a set s is not found **then**

12: Assign j to the G

13: **end if**

First-Fit 演算法

4.4 遊戲樹搜尋工作之 R 值調整

依照環境特性，調整副本數量

$\lambda(G) \geq R \cdot threshold_j$

R 是一個重要的參數，用來調整複本數量，如果 R 值太大，將相對地造成整體工作的複本數量過多，系統總產量也會相對下降；如果 R 值太小，將相對地造成整體工作的複本數過少，重要工作被砍掉重做的機率上升，導致工作完成時間相對的增加，加權效率下降。

排程器必須隨時監測系統效能，並適時降低或是增加 R 值來適應系統可用資源的變化並維持效率。先前學者提出的 R 值調整演算法是使用爬坡演算法 trial and error，隨機增減 R 值並藉以觀察系統效能指標會變好或是變壞，並以爬坡演算法來回調整多次才會跟上系統的實際變化。

本論文將系統效能分為以下兩類：

- R-正向：加權效率 WE
- R-反向：產量 TH

R 值對於系統的影響如下：

- 給定較低 R 值會降低 $\lambda(G)$門檻，進而減少 worker group 成員數(也就是減少複本量)，也會減少太多 worker 去計算同一個工作而浪費的計算能量，因為同一批主機能做更多不同工作或儲存更多不同資料。但這也會降低工作可靠度。

- 給定較高 R 值會提高 $\lambda(G)$門檻，進而增加 worker group 成員數(也就是增加複本量)，也會使用較多 worker 去計算同一個工作，這也會提高工作可靠度。

加權效率 WE 和產量 TH 的變化從下圖我們的先期研究可以看出：

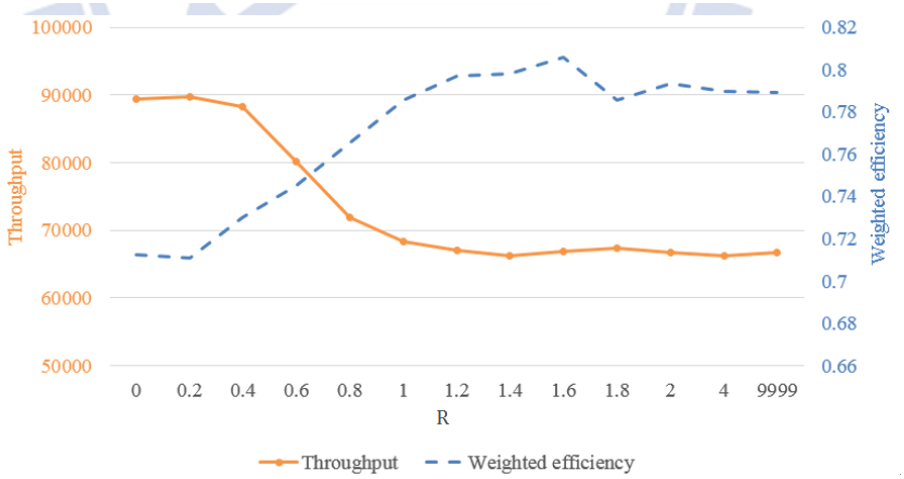

圖 4-4　產量與加權效率之變化圖

我們使用時間 window 觀測統計系統效能變化，若是區間 T_i和 T_{i+1}是呈現上升和下降分別以↑和↓符號表示之，那麼加權效率 WE 和產量 TH 的改變共有四種組合。針對這四種組合，我們歸納出 R 應該要調高或調低：

WE↑ TH↓ ➔ 增加 R	WE↑ TH↑ ➔ 未知
WE↓ TH↓ ➔ 未知	WE↓ TH↑ ➔ 降低 R

4.5 遊戲樹搜尋工作之臨界點分析

由圖 4-4 我們可以觀察出產量和加權效率的關係，當我們要求高加權效率時，必須增加複本的數量以增加成功率，導致產量下降；減少複本數量追求高產量時，則加權效率降低，於是產量與加權效率之變化圖呈現 X 交叉的現象(反向)。另外我們在實驗過程中發現一個現象，請看圖 4-5。

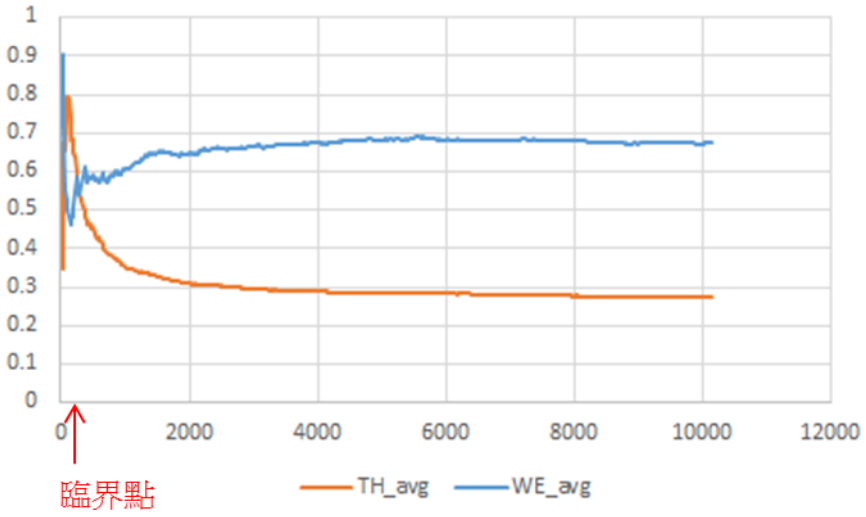

圖 4-5　產量與加權效率之變化圖

由上圖我們可以發現折線圖中，在產量與加權效率交叉呈 X 現象之前，有一小段時間是產量上升而加權效率下降的特別現象。這是因為一開始 R 值趨近 0，幾乎沒有產生工作的複本，造成工作被中斷重作的機率非常高，導致產量低落。因此當 R 值達到基本門檻，有了基本的成功率之後，就會把一開始囤積下來未完成的工作一一完成，呈現加權效率低落，同時出現產量瞬間高升的現象。等環境穩定之後，則如前述所言，產量會隨著複本量增加而降低，加權效率上升而呈現 X 交叉的相反現象(反向)。因此我們將其 R 值基本門檻視為臨界點。

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

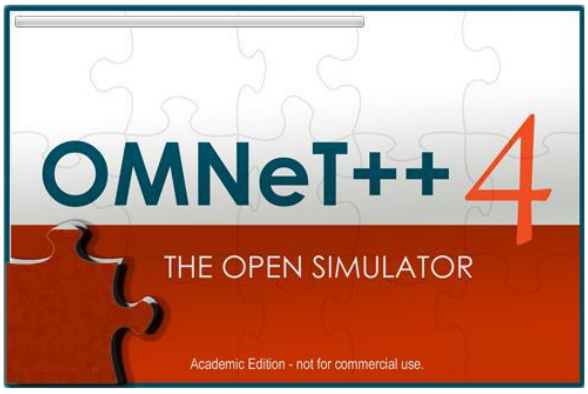

圖 5-1　omnet++圖示

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 模擬程式畫面

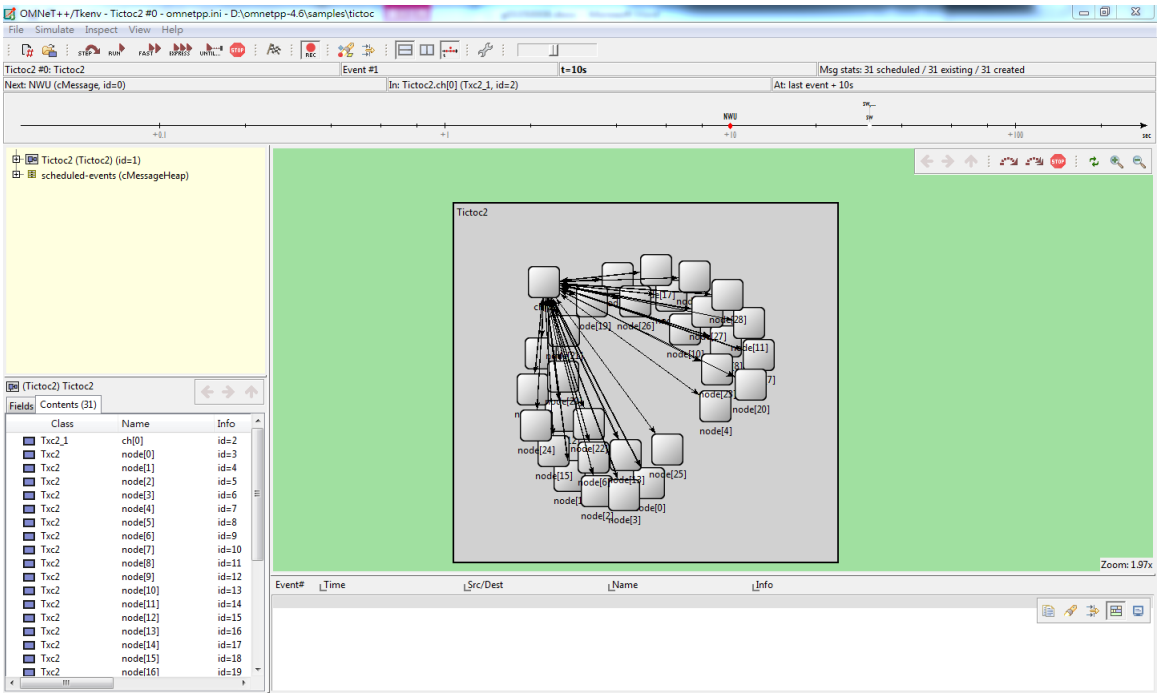

圖 5-2　圖形化使用者介面

Event#	Time	Src/Dest	Name	Info
#257	124	node[25] --> ch[0]	0,0,0,25	id=11098 kind=9
#260	124	node[28] --> ch[0]	0,0,0,28	id=11100 kind=9
#261	124	node[29] --> ch[0]	39,13,50,29	id=11102 kind=8
#278	125	ch[0] --> node[28]	41,13,50,28	id=11104 kind=5
#280	130	ch[0] --> node[25]	42,5,50,25	id=11106 kind=5
#280	130	ch[0] --> node[24]	43,5,50,24	id=11108 kind=5
#285	131	ch[0] --> node[22]	44,12,50,22	id=11110 kind=5
#285	131	ch[0] --> node[18]	45,12,50,18	id=11112 kind=5
#290	136	node[12] --> ch[0]	28,10,50,12	id=11011 kind=2
#295	142	ch[0] --> node[15]	46,14,50,15	id=11115 kind=5
#295	142	ch[0] --> node[14]	47,14,50,14	id=11117 kind=5
#298	145	node[26] --> ch[0]	32,11,50,26	id=11053 kind=2
#301	147	node[21] --> ch[0]	34,7,50,21	id=11057 kind=2
#302	147	ch[0] --> node[26]	48,15,50,26	id=11121 kind=5
#302	147	ch[0] --> node[12]	49,15,50,12	id=11123 kind=5
#308	155	node[2] --> ch[0]	0,0,0,2	id=11125 kind=9
#310	155	node[4] --> ch[0]	0,0,0,4	id=11127 kind=9
#312	155	node[6] --> ch[0]	-1,-1,50,6	id=11129 kind=8
#317	155	node[11] --> ch[0]	0,0,0,11	id=11131 kind=9
#318	155	node[12] --> ch[0]	49,15,50,12	id=11133 kind=8
#319	155	node[13] --> ch[0]	36,9,50,13	id=11135 kind=8
#320	155	node[14] --> ch[0]	47,14,50,14	id=11137 kind=8
#322	155	node[16] --> ch[0]	0,0,0,16	id=11139 kind=9
#323	155	node[17] --> ch[0]	0,0,0,17	id=11141 kind=9
#325	155	node[19] --> ch[0]	35,9,50,19	id=11143 kind=8
#326	155	node[20] --> ch[0]	0,0,0,20	id=11145 kind=9
#329	155	node[23] --> ch[0]	0,0,0,23	id=11147 kind=9
#330	155	node[24] --> ch[0]	43,5,50,24	id=11149 kind=8
#332	155	node[26] --> ch[0]	48,15,50,26	id=11151 kind=8
#334	155	node[28] --> ch[0]	41,13,50,28	id=11153 kind=8
#335	155	node[29] --> ch[0]	0,0,0,29	id=11155 kind=9
#336	155	ch[0] --> node[21]	50,16,50,21	id=11157 kind=5
#336	155	ch[0] --> node[9]	51,16,50,9	id=11159 kind=5
#359	160	ch[0] --> node[29]	52,9,50,29	id=11161 kind=5
#359	160	ch[0] --> node[23]	53,9,50,23	id=11163 kind=5
#363	170	node[7] --> ch[0]	40,13,50,7	id=11071 kind=2
#365	171	ch[0] --> node[20]	54,15,50,20	id=11166 kind=5
#365	171	ch[0] --> node[17]	55,15,50,17	id=11168 kind=5
#369	180	node[25] --> ch[0]	42,5,50,25	id=11106 kind=2
#371	180	ch[0] --> node[16]	56,17,50,16	id=11171 kind=5
#371	180	ch[0] --> node[11]	57,17,50,11	id=11173 kind=5
#375	181	node[22] --> ch[0]	44,12,50,22	id=11110 kind=2
#376	181	node[18] --> ch[0]	45,12,50,18	id=11112 kind=2
#382	186	node[3] --> ch[0]	0,0,0,3	id=11177 kind=9

圖 5-3　文字介面輸出

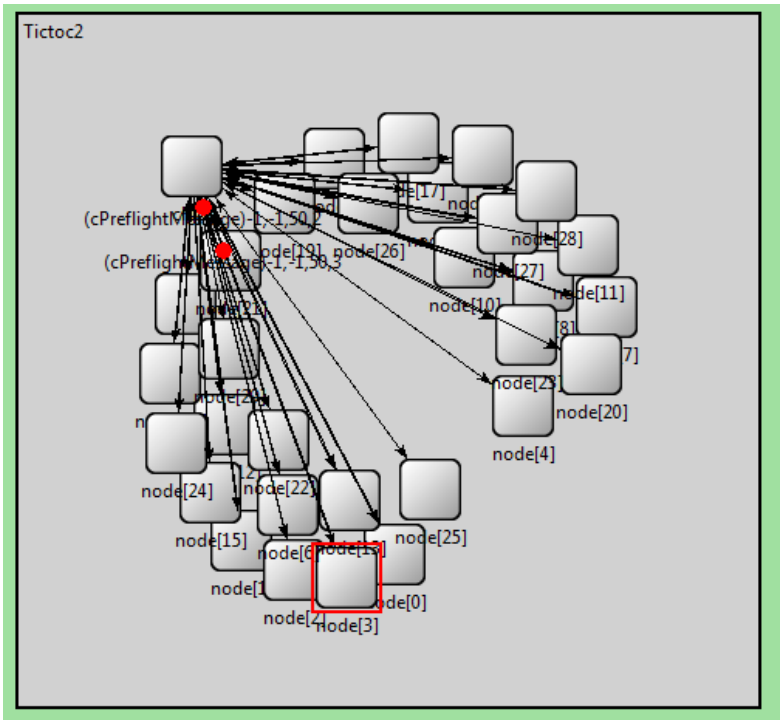

圖 5-4　圖型化輸出視窗

5.3 模擬實驗結果

以下為實驗的參數說明：

表 5-1　實驗參數

worker_num	Worker 數量
user_num	User 數量
failRate	Worker 的失敗率
task_number	任務數量
STANDARD_TASK_LENGTH	任務長度

本論文的實驗的目的是要探討臨界點對不同 failRate 之間的關係。因此，我們執行六種不同 failRate 共 6 組實驗。

表 5-2　六組不同 failRate(F3~F8)的實驗參數設定

F3	F4	F5	F6	F7	F8
failRate=0.7	failRate=0.6	failRate=0.5	failRate=0.4	failRate=0.3	failRate=0.2

以上為我們的預設條件，預設條件除了 failRate，其餘皆為固定參數，worker_num=30、task_number=1000，以下是我們的實驗結果。

以下實驗為 F3-F4 的實驗：

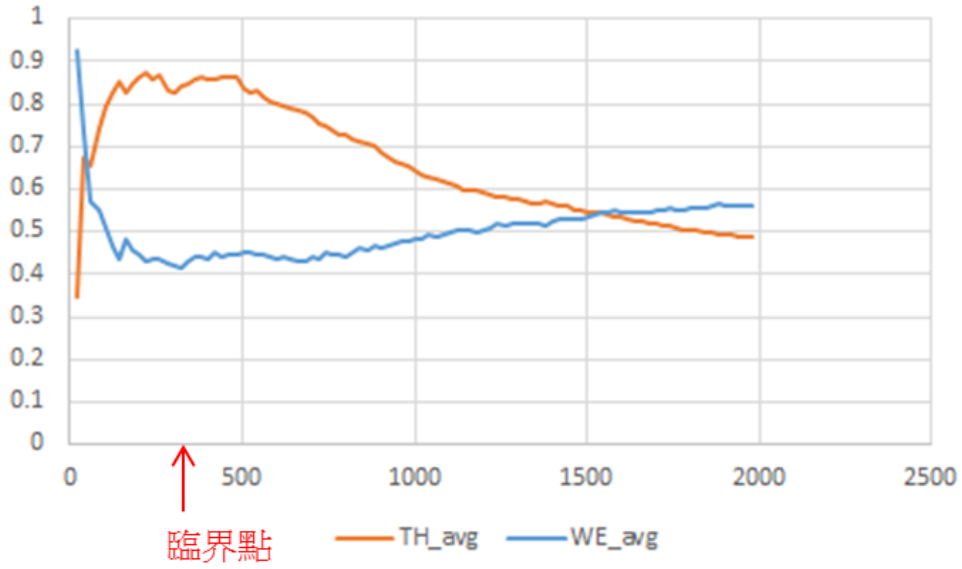

圖 5-5　實驗 F3：failRate=0.7

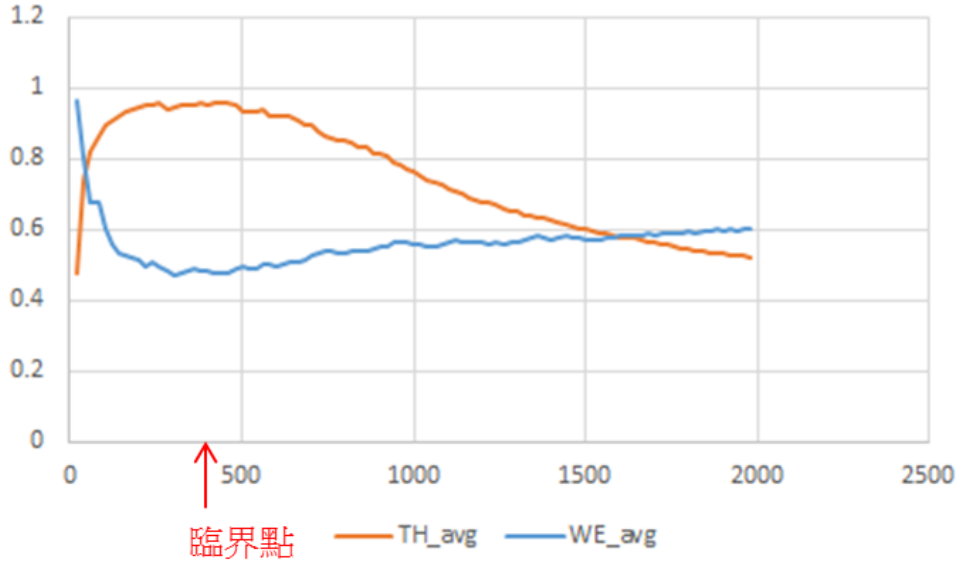

圖 5-6　實驗 F4：failRate=0.6

以下實驗為 F5-F6 的實驗：

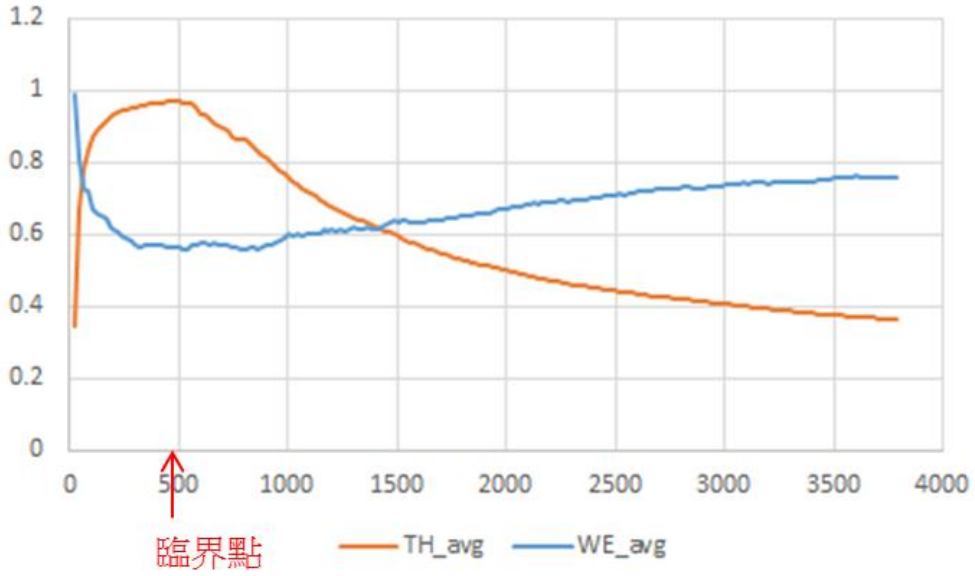

圖 5-7　實驗 F5：failRate=0.5

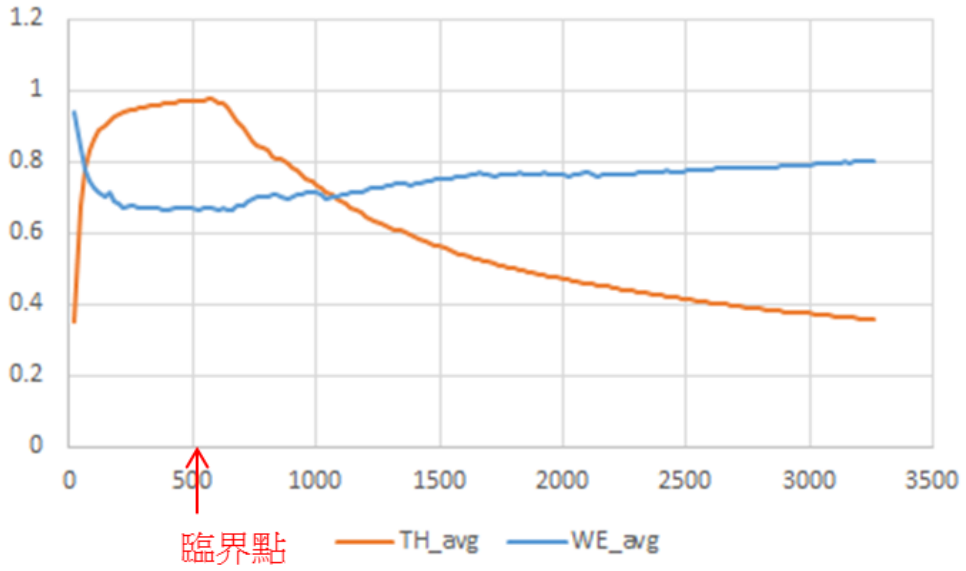

圖 5-8　實驗 F6：failRate=0.4

以下實驗為 F7-F8 的實驗：

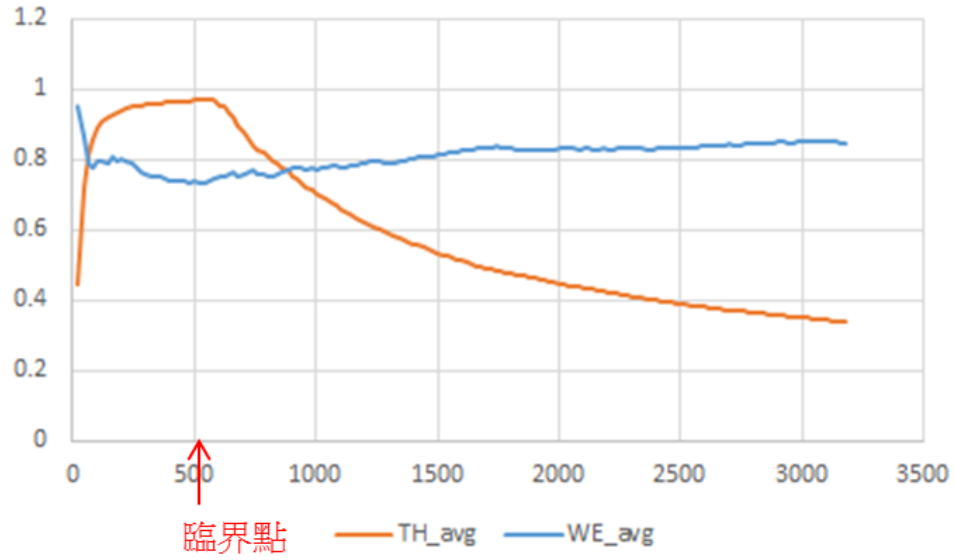

圖 5-9　實驗 F7：failRate=0.3

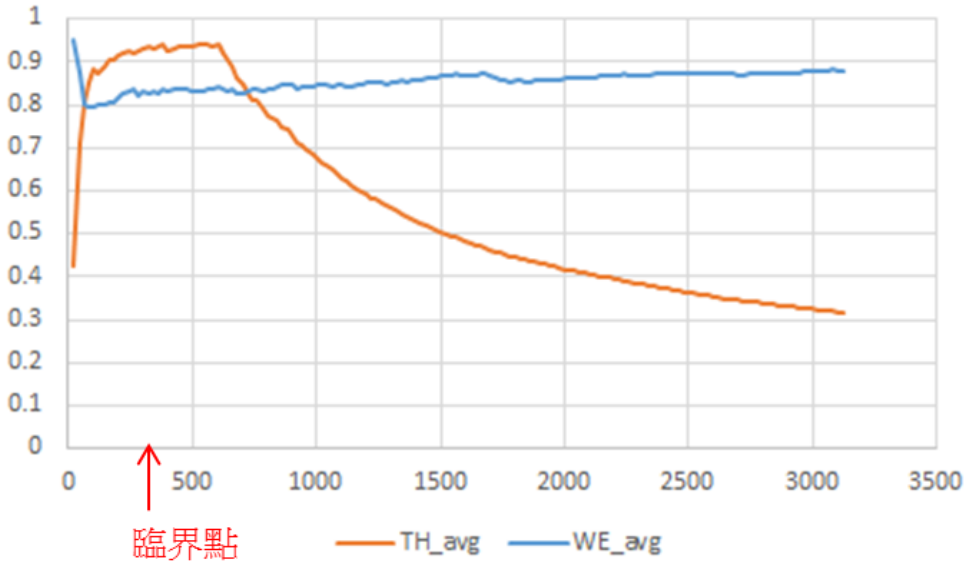

圖 5-10　實驗 F8：failRate=0.2

表 5-3　不同 failRate(F3~F8)的臨界點大小

F3	F4	F5	F6	F7	F8
F3:300	F4:400	F5:500	F6:500	F7:500	F8:300

此次實驗對於環境進行 failRate 的調整，可以觀察到臨界點介於 300~500 之間，所對應到的 R 值為 2~4，顯示出低於 2 以下的 R 值幾乎沒有產生工作的複本，對於系統環境沒有太大影響，因此我們建議使用者對於 R 值的初始值設定大於 2 以上，以穩定觀測系統效能變化。

六、 結論與未來工作

為了解決資源碎片問題，以及重要的 malleable 工作因中斷而延後完成的問題，我們利用工作複本的技術來減少 malleable 工作的完成時間，並套用在遊戲樹搜尋的應用上。從上面的實驗中可以觀察出，在臨界點之前，幾乎沒有產生工作的複本，造成工作被中斷重作的機率非常高，導致產量低落。因此當 R 值達到基本門檻，有了基本的成功率之後，就會把一開始囤積下來未完成的工作一一完成，呈現加權效率低落，同時出現產量瞬間高升的現象；在臨界點之後，複本做的越多，產量就會越低，但加權效率會越高，反之亦然。當碎片資源越細碎，複本做越多的演算法會有較高的加權效率，重要的工作完成的時間相對較短。因為加權效率和產量是相對的，當想要提升產量時，勢必要降低複本的個數，就會相對增加工作的完成時間，加權效率將會降低，本篇論文提供使用者喜好模式，讓使用者可以根據工作類型選取以產量最大化為目標或是以加權效率最大化為目標的不同模式。

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

附錄

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。

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:

表 5-4　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
```

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：

表 5-5　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
```