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

主從式架構之最佳工作流程元件部署方法

An Efficient Deployment Method of Workflow Components for Client–Server Architecture

研究生：林俞汎

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 俞 汎 所提之論文

主從式架構之最佳工作流程元件部署方法

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

近年來隨著雲端技術的發展，服務導向架構(SOA)在企業流程的系統整合中扮演著不可缺的角色，而 Web service 更是實現 SOA 的重要基礎。在應用程式中可能包含許多個服務元件，而元件的運作成本中包含計算成本和資料傳輸的成本，所以元件的部署的地點會對程式的效能和成本產生關鍵性的影響。本文將探討如何在主從式架構的主機群中部署應用程式的服務元件，找出最佳成本的部署策略。研究顯示，只傳送所需要的物件內容，而不需更新所有的服務物件可以減少雲端服務的運作成本，並且提高整體效能。

關鍵詞：服務導向架構、內容遞送網路、工作流程元件部署。

ABSTRACT

In these years, with the development of cloud computing,Service-Oriented Architecture (SOA) plays an important role in systematic integration of Business Process Management (BPM). Web service is a foundational base to achieve Service-Oriented Architecture (SOA). The web service components of a service application consist of the computing costs and the data transmission costs. Components deployment site can have a crucial impact on effectiveness and cost.The purpose of this thesis is to deploy service component in client–server architecture and find the best cost deployment strategies.Research shows that only transmit content items needed, without having to update all service objects cloud services can reduce operating costs, and improve overall performance.

Key words: service-oriented architectures, content delivery network, workflow component deployment.

CONTENTS

LIST OF FIGURES

一、簡介

系統整合普遍發生在企業的營運活動與業務流程中，然而系統的資源要互相流通因沒有共用性的標準，而造成維護成本偏高。為了讓系統之間有更好的交流以及一套標準化、彈性化的整合方式，提出了服務導向架構(SOA)，透過建立架構模組化，使服務具有可替換性且具重複使用的功能，所以 SOA 已經成為目前電子商務交易的主流。

SOA 主要以元件再利用的方式，提升系統開發的效率，也就是說同一套企業流程元件可以在不同系統中重複使用，減少開發時間並降低開發成本，而為了使元件可以重複使用 SOA 的整合方式是非常重要的，SOA 的整合方式主要可以歸類為資料整合、流程整合和服務整合三種。

資料整合是由企業依據各種資料整合情境分析出適合的整合特性，例如資料屬性、資料數量、即時性等，綜合考量這些特性，來決定合適的整合方法，也就是說所有應用於 SOA 架構系統的資料，都需要清楚的定義，而企業中資料定義的方式則需要一個統一的標準，而對於少量且即時性的資料整合，則採用標準的 Web Service 技術，將資料以 XML 形式傳遞，是目前的主流做法。流程整合的重點在於以業務流程為核心思考企業的價值鏈。流程整合所用的技術大多為了滿足資料整合與服務整合。定義出符合企業的流程模組後，有順序的將各個應用系統所提供的工作流程元件 (Workflow component) 開放成 Web Service 的形態進行流程串接，形成跨應用系統的企業流程。這種把元件當做服務，直接呼叫、引用的系統開發方式，不僅加快了開發、整合系統的速度，同時並確保了一個完整作業流程所需的資訊品質，如此，流程會變得具有彈性，不論是業務流程改變或是再串連其他系統的服務，皆可以較低的維護成本達成。服務整合是應用程式採用 Web Service 為統一的標準，這也是實現 SOA 的基礎。在實作方面，以 WSDL 及 SOAP 為核心技術，同時將服務元件開放，讓外部的系統可以重複使用。

在電子商務(E-Commerce)上 Web Service 常扮演資料傳送的角色，經由 XML 文件的遞送將資料完整的複製到另一個系統。而透過雲端網路的環境，Web Service 被廣泛應用，傳統的網路服務系統只針對個別應用程式來計算其最佳策略，這會造成系統傳送成本增加。所以網路服務元件的部署是一個重要的議題，通常同一流程下的服務物件具有一定程度關聯性，若能一起部署，則可以降低整體營運管理成本。

近年來採用服務導向架構(SOA)開放架構的企業，都是著眼於降低營運成本的考量，SOA 提供了流程的管理及分散式軟體的框架(Framework) ，促使開發人員可以快速且彈性的整合各種資料。內容遞送網路(CDN)的服務模式，就是透過事前的設定與管理有效率的將許多服務元件部署到副本主機(replica server)上。

一個物件可以在 server 或是 client 計算，在網際網路應用中，可以經由 client 端重新計算資料的方式來加速系統的執行效率，例如，在 client 端的計算基於它的前行物件執行轉換[3][4][5][6]的方式來建立物件，所有的內容物件部署成本可以經由網路直接的遞送與 client 端的重新計算來減少整個部署成本。我們可以將內容遞送的問題轉換成網路流量用來建立最佳的部署策略，利用此部署演算法，使分散式內容應用程式有效地運作。本論文將內容遞送的問題轉換成網路流量問題來建立最佳的部署策略，利用此部署演算法，使分散式內容應用程式有效地運作。

在 SOA 環境中，一個應用程式可能包含許多服務元件。這些元件的部署方式對應用程式的整體效能具有關鍵性的影響。如何在資料傳送與主機計算能力的成本考量下做一個平衡的取捨，也是本篇論文主要探討的問題。如果當網路傳送的成本比較高，將不適合做大量資料的傳輸；反之，如果主機計算成本比較高，那由網路直接取得資料比自行計算來的有效率。

本篇論文將服務元件表達為無迴圈有向圖，稱為 ODG (object dependency graph)。藉由將 ODG 轉換為傳統流量演算法(Maximum flow/Minimum Cut)，我們

的演算法能求出最小遞送成本的服務元件的部署策略。

二、工作流程元件部署服務之問題探討

2.1 工作流程元件的部署問題

一個 SOA 應用程式包含許多服務元件，以無迴圈有向圖表達之，稱為 ODG (object dependency graph)。在 ODG G=(V,E)中，每一個節點 (屬於集合 V) 代表一個服務元件，而節點 a 到節點 b 之間的線條(屬於集合 E)，代表服務 a 與服務 b 之間的先後順序關係。各個服務元件都是可執行軟體元件，執行服務所產生的結果，可以成為後續服務的輸入資料。沒有前行者的節點稱為 *初始節點* (initial node)，沒有後行者的節點稱為 *目標節點* (target node) 或是*結束點*。

ODG 代表物件的計算流程的相依性，假設計算相依性是$v_i \rightarrow v_j$，表示元件v_i 計算所產生的 output 會成為元件v_j 的 input，進而觸發連續的計算行為。以圖 2.1 為例，ODG 代表網路上提供的 Web 版的程式語言學習教材，包含了三大主題：Data type、Flow-control、Function，而每個主題又有內容(Content)、網頁呈現(Display)、與測驗(Quiz)等三類物件。每一個元件都是一份 XML 文件，一份 XML 文件可以由它的前行者 XML 文件經過 XSLT 轉換而得之。

在網路部署環境中，我們假設 Server 端伺服器擁有所有 XML 格式的內容元件，亦即 ODG 所有節點。假設計算相依性是$v_i \rightarrow v_j$ ，一個 client 主機有兩種方式得到元件v_j ：

- Server 直接傳輸v_j 的 XML 內容；
- Server 傳輸前行者節點v_i ，client 主機由v_i 自行建構v_j ；

例如，圖 2.1 的 ODG 中，Server 可以傳輸前行者節點 Content1 和 Display1 文件，client 主機收到後再計算出 Display2 文件。

以上兩種方法的取捨是由成本來決定。每一個元件節點 v 有兩類成本：

- $\mathrm{comp}(S_k, v)$ 代表主機 S_k 計算元件 V 的計算成本；
- $\mathrm{net}(S_p, S_k, v)$ 代表主機 S_p 傳送元件 V 至主機 S_k 的成本。

根據主從式架構的問題模型，一個目標物件 V 可以經由 server 端伺服器直接遞送到 client 端，遞送成本為 $\mathrm{net}(S_{\mathrm{server}}, S_{\mathrm{client}}, v)$，另外，該目標物件 V 也可以經由用戶端自行運算所產生，計算成本 $\mathrm{comp}(S_{\mathrm{client}}, v)$。

以圖 2.1 例子來說，在 client 端主機上，若是要得到 Display2 文件，它可以用 net(Display2)=120 的成本得到 Display2 文件。client 端還可以用另一種方式得到 Display2，就是先取 Content2 和 Display1，然後自行計算 Display2，這個方案的成本是 net(Content2) + net(Display1) + net(Quiz2) + comp(Display2) = 30 + 70 + 15 + 30 = 145，很顯然地，以 Display2 物件為例，要取得 Display2 內容經由網路的成本 120 與自行運算的成本 145 相比起來第一種方案顯然較佳。而以上二種部署問題，會因不同的環境而有不同的成本需求。

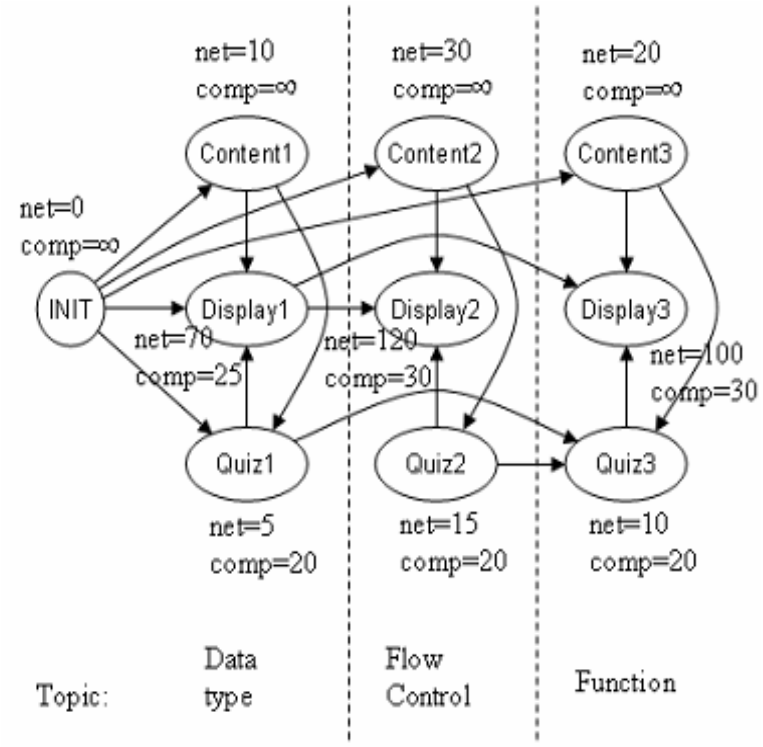


圖 2.1：Web 程式語言教材之 ODG

2.2 雲端部署成本

圖形 H 的節點集合 U 中每一節點 v 均有二種成本 *net(v)*及 *comp(v)*，成本 *net(v)* 為網路成本，*comp(v)*則為物件成本，網路成本取決於網路傳輸費用與傳輸檔案的大小，我們分別以 w_x 和 n_i 表示之，而物件成本則是以主機的計算成本和計算的工作時間來決定，這裡用 k_x 和 c_i 表示。*net(v)*與 *comp(v)*的成本公式如下：

- w_x 代表主機 S_x 傳送成本係數；
- k_x 代表主機 S_x 計算成本係數；
- n_i 代表元件 v_i 傳送成本係數；
- c_i 代表元件 v_i 計算成本係數；

以上兩種方法的取捨是由計算成本來決定。每一個元件節點 v 有兩類成本：

- $\text{comp}(S_x, v_i) = k_x c_i$；代表主機 S_x 計算元件 v_i 的計算成本。
- $\text{net}(S_p, S_x, v_i) = w_x n_i$；代表主機 S_p 傳送元件 v_i 至主機 S_k 的成本。

*net(v)*是網路成本，為 w_x(網路傳輸費用)和 n_i (傳輸檔案大小)的乘積，假設傳送 1mb 的檔案費用為 10 元，要傳送 3mb 大小的檔案必須花 30 元的網路成本；而 *comp(v)*為物件成本，是 k_x (主機的計算成本，例：租用主機的成本)和 c_i (計算的工作時間)的乘積，以 Amazon EC2(圖 2.2)租用主機為例，如果租用 t2.large 主機計算 1 小時的時間花費為 0.104 元，主機共花了 50 小時的時間來計算，則花費的物件成本為 0.104 X 50 = 5.2。

	vCPU	ECU	Memory (GiB)	Instance Storage (GB)	Linux/UNIX Usage
General Purpose - Current Generation					
t2.nano	1	Variable	0.5	EBS Only	$0.0065 per Hour
t2.micro	1	Variable	1	EBS Only	$0.013 per Hour
t2.small	1	Variable	2	EBS Only	$0.026 per Hour
t2.medium	2	Variable	4	EBS Only	$0.052 per Hour
t2.large	2	Variable	8	EBS Only	$0.104 per Hour
m4.large	2	6.5	8	EBS Only	$0.12 per Hour
m4.xlarge	4	13	16	EBS Only	$0.239 per Hour
m4.2xlarge	8	26	32	EBS Only	$0.479 per Hour
m4.4xlarge	16	53.5	64	EBS Only	$0.958 per Hour
m4.10xlarge	40	124.5	160	EBS Only	$2.394 per Hour
m3.medium	1	3	3.75	1 x 4 SSD	$0.067 per Hour
m3.large	2	6.5	7.5	1 x 32 SSD	$0.133 per Hour
m3.xlarge	4	13	15	2 x 40 SSD	$0.266 per Hour
m3.2xlarge	8	26	30	2 x 80 SSD	$0.532 per Hour

圖 2.2： Amazon EC2 Pricing.

2.3 部署策略

本節定義主從式網路架構的*服務部署策略*（*deployment strategy*）。對主機S_k來說，一個服務(ODG 的節點)的部署型式可分成三種：

- *O-node*：該服務不部署在主機S_k。這一類節點以集合O_k表示之。
- *N-node*：該服務部署在主機S_k，且是由其他主機計算後，傳送至主機S_k。這一類節點以集合N_k表示之。
- *C-node*：該服務部署在主機S_k，且是由主機S_k自行計算。這一類節點以集合C_k表示之。

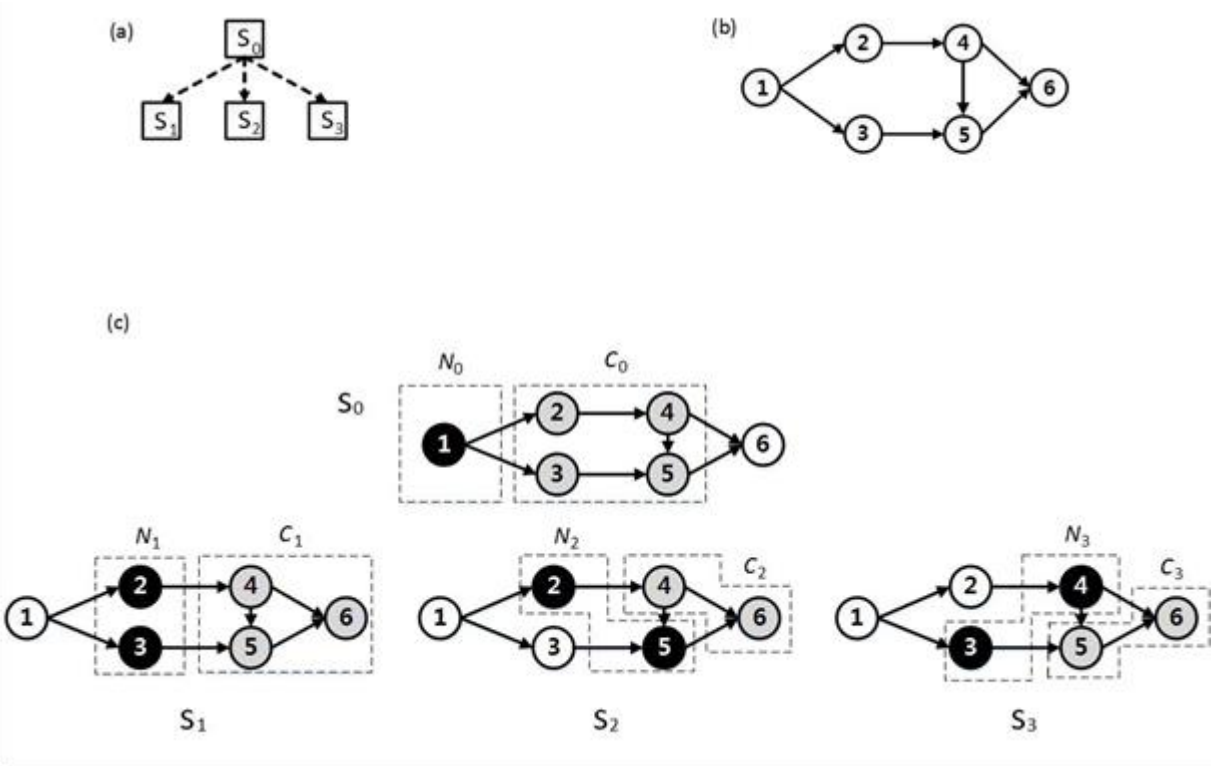


圖 2.3-1 : ODG 部署策略

假設網路 S 有 M+1 個主機 $S_0,S_1,...,S_M$，這 M+1 台主機被組織為主從式架構，以 S_0 為原始主機(Server 端)，S_1 ~S_M 為終端主機(Client 端)。這 M+1 個主機對應到策略集合 $(N_0,C_0),(N_1,C_1),...(N_M,C_M)$。這些 M+1 個部署策略集合起來就形成應用程式(以 ODG 表示之)在主從式架構網路 S 上的部署策略。如圖 2.3-1 所示，有一樹狀圖代表主從式主機結構(圖 2.3-1 (a))，有一個圖形 S=(V,E) 代表 ODG(圖 2.3-1 (b))。S 定義了工作流程元件之間的先後次序關係，而圖 2.3-1 (c)表示應用程式在圖 2.3-1 (a)的網路結構中的部署策略，以圖 2.3-1 (c)表示黑色節點為部署中的 *N-node*，而灰色節點則為 *C-node*。

考量物件的關聯性，並非所有部署項目都是可行的 ，可行的(或稱合法的)部署策略定義如下。

定義 1： 對 M+1 個階層式主機 $S_0,S_1,\ldots,S_M$，可行的(或稱合法的)部署策略 $(N_0,C_0),(N_1,C_1),\ldots(N_M,C_M)$必須要滿足以下的條件：

(1) 在同一主機上，一個節點只能有一種部署方式。因此， $N_k \cap C_k = \phi$，$0 \le k \le M$ 。

(2) 起始節點必須部署在根主機 S_0 (即 Server 主機)，目標節點必須部署在所有終端主機(即 Client 主機)。

(3) 每一台 Client 主機 S_k 的 *N-node* N_k 由 Server 主機 S_p 來提供，因此，$N_k \in (Np \cup Cp)$ 。

(4) 每一台主機 S_k 的 *C-node* C_k 由 S_k 自行計算，因此，PRED $(C_k) \in (N_k \cup C_k)$，PRED (C_k) 表示 C_k 的前行者節點集合。(根據 ODG 模型，當主機 S_y 計算一個 ODG 節點時，它必須有該節點的所有前行者節點。詳見上述 ODG 定義。)

■

以圖 2.3-2 為例，從原始主機 S_0 部署 ODG 到 S_1。在圖 2.3-2 (c)中，S_0 部署項目(N_0,C_0)= ({1}，{2,3}) 和 S_1 部署項目(N_1,C_1)= ({2,3}，{5,6}) 並非可行，因為 S_1 中的計算物件{5}和計算物件{6}需要取得的前行物件{4} 並沒有部署在 S_1。另外，在圖 2.3-3 (c)下，由於 S_1、S_2、S_3 中的所有 Comp 物件都可以從 S_0 原始主機直接取得結果，故其策略為可行。

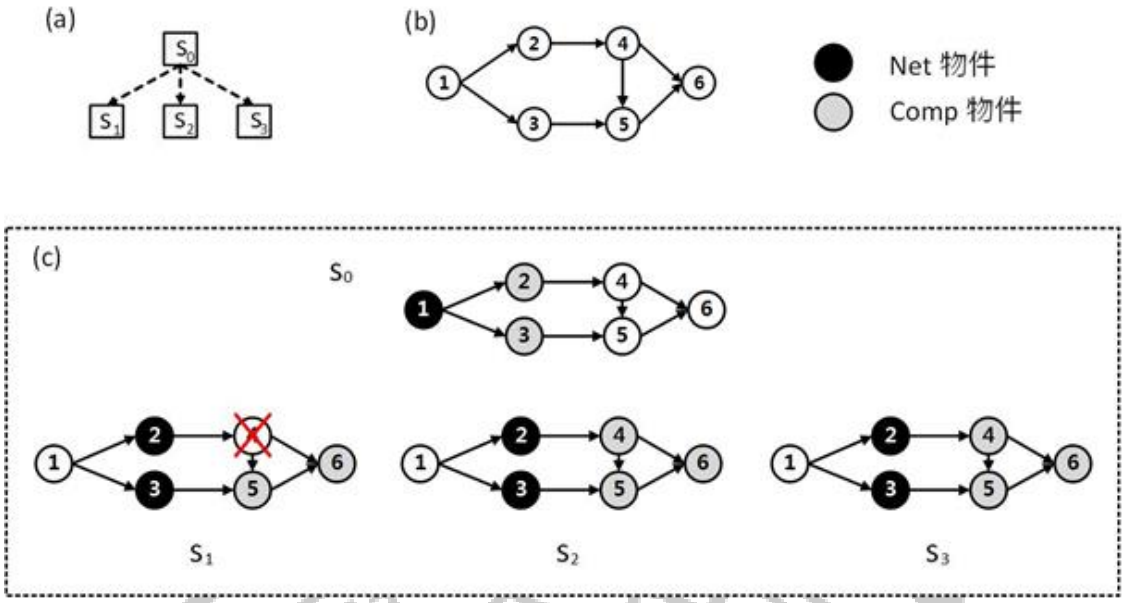


圖 2.3-2：(a)網路結構圖。(b)ODG。(c)不可行部署策略。

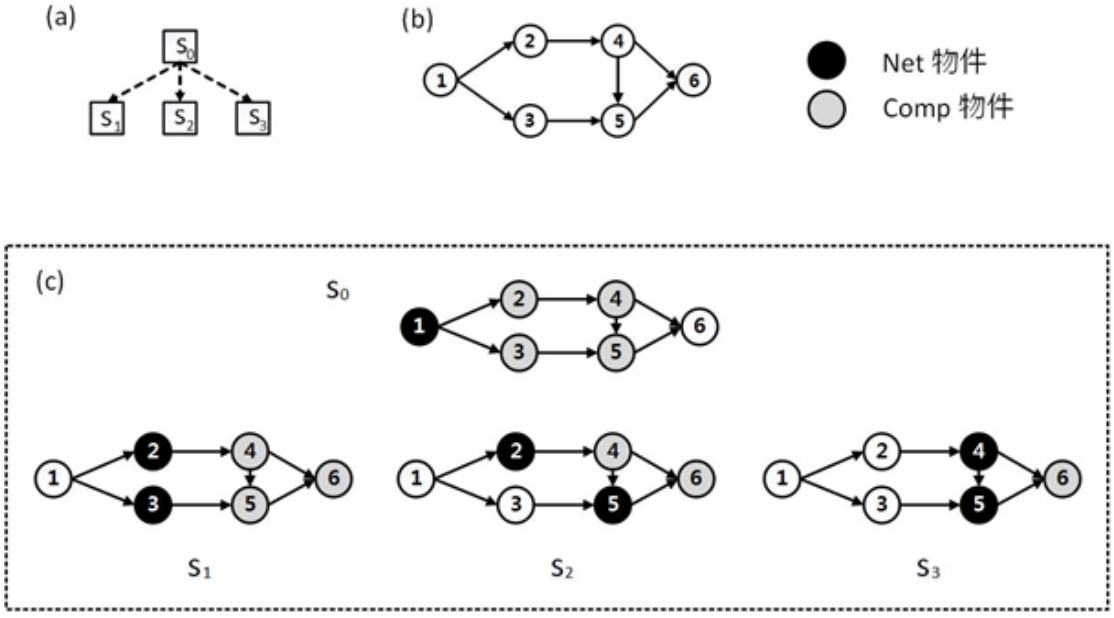


圖 2.3-3：(a)網路結構圖。(b)ODG。(c) 可行部署策略。

本篇論文討論主從式架構的最佳工作流程元件部署方法，在每個主機 S_k 上 ODG 的物件部署策略有二個成本。對於每一個節點 v 分別是：

- comp (S_k, v)：代表元件在某一主機 S_k 中的計算成本。
- net(S_p, S_k, v)：代表元件從 Server 主機 S_p 傳送至 Client 主機 S_k 的成本。

部署策略$(N_0,C_0),(N_1,C_1),\ldots(N_M,C_M)$的成本定義為 (令 S_p 為 S_k 之 parent 主機)：

$$\sum_{0\le k\le M}\left[\sum_{v\in N_k}\text{net}(s_p, s_k, v)+\sum_{v\in C_k}\text{comp}(s_k, v)\right] \qquad (1)$$

在公式(1)中，每一個主機 S_k $(0 \le k \le M)$之 *N-node* 之 net 成本及 *C-node* 之 comp 成本之總和即為總部署成本。

不同的案例有不同的需求，所以我們必須在 comp 和 net 成本中做取捨，本文求最佳部署方法即是解決*最小成本部署策略*問題求出一組策略 $(N_0,C_0),(N_1,C_1)$ $,\ldots(N_M,C_M)$ 使其成本最小。

2.4 ODG 成本

本論文所敘述的 ODG 成本，可以解釋為雲端的租賃費用、運算產生所消耗的能量、網路成本以及儲存空間等可以量化的數據。為了節省更多成本開銷，以上量化質自然是越低越好，而在不同情形下，我們考量使用不同的部署方法。舉例來說，Flash 動畫傳送完整影像會比傳送指令再由 client 自行運算畫面花費更多網路成本，所以比較適合以 client 計算的佈署方式。相反的，如果是氣象預報這種需要大量計算，但計算結果卻只有幾 bytes，特別依賴電腦計算能力的就比較建議由 server 計算出答案在傳送給 client。

2.5 工作流程元件之部署演算法

為了找出部署策略問題的最小成本，常見的演算法如下：

- 線性規劃：將公式(1)視為數學線性規劃問題而解之。這種解法大多運用 SAS 等統計軟體，其缺點是在隨著 ODG 節點數目增加時，有時執行時間會呈現指數遞增之趨勢。
- 動態規劃 (*dynamic programming*)：將 ODG 切割幾個子圖，個別求解之後，再將這些解合併起來。目前這解法無法針對一般化問題(general problem)求解，大多針對特殊問題來求解，例如限制網路為二層樹狀結構(即 client-server 架構) 或是線性結構(linear structured)，也有研究者探討簡化的成本模型。
- 網路流量演算法(Network flow)：將樹狀結構網路以及 ODG 轉換成網路流量問題解之。此種方法效率高但目前實際上只用在二層的樹狀結構(即 client-server 架構)之網路。當樹狀結構超過三層或以上時，轉換極為繁複。

本篇論文探討在主從式架構的最佳工作流程元件部署方法，並探討計算成本和網路成本的取捨，找出最佳部署方法。

三、部署策略問題及成本取捨探討

本章節我們整理幾個常見的 network flow 類型的部署策略演算法，並提出本篇論文的成本取捨方式與部署策略的問題。

3.1 圖形切割問題與部署策略問題之轉換

圖形切割

我們定義"*H-cut*"，一個 H-cut (X,Y)將圖形 H=(U,F)的節點集合 U 分割為 X 和 Y 二部分，其中(U,F)為有向圖，而圖形 H 有 s 節點與 t 節點，分別為來源點與接收點，s 節點(來源點)和 t 節點(接收點)必須分別在 X 和 Y 之中。

轉換演算法

我們把最小成本部署策略轉換成圖形 H 的切割問題，圖形 H=(U,F)。圖形 H 由子圖 $G^0, G^1, \cdots, G^M$ 組成，其中 G^x 對應到主機 S^x。

定義 2：圖形 $G^x = (V^x, E^x)$ 跟 ODG 相似($0 \le x \le M$)，建構方式如下：

- 每個 ODG 節點 v 轉換成線段 $(v_{\mathrm{b}}^{(x)}, v_{\mathrm{e}}^{(x)})$ 和 $(v_{\mathrm{e}}^{(x)}, v_{\mathrm{b}}^{(x)})$，並令 cap$(v_{\mathrm{b}}^{(x)}, v_{\mathrm{e}}^{(x)})$=*net*($v$)和 cap$\left(v_{\mathrm{e}}^{(x)}, v_{\mathrm{b}}^{(x)}\right) = \infty$。(b 表示 begin；e 表示 end。)
- 每個 ODG 線段(u, v)轉換成線段$(u_{\mathrm{e}}^{(x)}, v_{\mathrm{b}}^{(x)})$，並令 cap$(u_{\mathrm{e}}^{(x)}, v_{\mathrm{b}}^{(x)})$=∞。
- 對 G^0，v_{e} 節點連到所有 n_{e} 節點，$1 \le v \le n-1$，並令 cap$\left(v_{\mathrm{e}}, n_{\mathrm{e}}\right)$ = *comp*(v)。(節點 n 表示 ODG 的最後節點)
- 對 $G^1, \cdots, G^M$， 1_{b} 節點連到所有 v_{b} 節點，$2 \le v \le$ n，設 cap$\left(1_{\mathrm{b}}, v_{\mathrm{b}}\right)$ = *comp*(v)。(節點 1 表示 ODG 的第一個節點)

■

G^0對應到主機S^0，$G^1 \cdots, G^M$分別對應到主機$S^x \cdots, S^M$，圖 3.1-1 為 ODG 圖形在主機S^0和S^1中的轉換圖G^0(圖 3.1-1 (a))與G^1(圖 3.1-1 (b))。

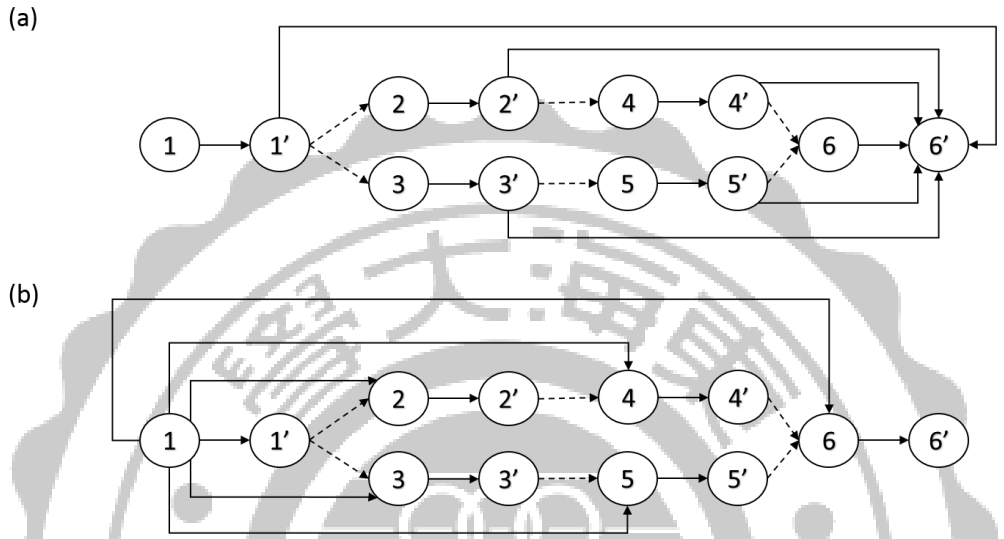


圖 3.1-1： (a) G^0 和 (b) G^1 $(n = 6)$

將所有圖形G^x連接起來即形成圖形 H，連接方式為網路圖 S 和 ODG 圖的乘積運算(Cartesian product)。圖形 H 的定義如下。

定義 3：給定一個網路 S 有 M+1 個主從式結構主機 $S_0, S_1, \ldots, S_M$，ODG 圖 G=(V,E)，圖形 H=(U,F)建構如下：

(1) 建立 s 與 t 分別為圖形 H 的起始點和結束點。

(2) 建立以下 ∞-edges 把G^x連起來：

- $E_{\text{DOWN}} = \{(v_e^{(0)}, v_e^{(k)}) \mid v \in V, S_0$是*server*主機，而$S_k$是*client*主機$\}$ 。
- $E_{\text{TARGET}} = \{(n_e^{(k)}, t) \mid n$是$G$的結束點$, S_k$是主機$\}$ 。
- $E_{\text{SOURCE}} = \{(s, 1_b^{(k)}) \mid 1$是$G$的起始點$, S_k$是主機$\}$ 。

(3) 建立圖形 H 的點集合$U=(V^{(0)}\cup V^{(1)}\cup....\cup V^{(M)}\cup\{s\}\cup\{t\})$，建立圖形 H 的線集合$F=(E^{(0)}\cup E^{(1)}\cup\cdots\cup E^{(M)})\cup E_{\text{DOWN}}\cup E_{\text{TARGET}}$。

■

P1 一個最小成本 H-cut 不會切過任何∞-edge。最小成本切割之成本不會為無限大，故不會切過∞-edge。

利用以上 P1 特性，我們在流量圖中加入了許多成本為∞的 edge ，強迫流量圖的切割與 ODG 部署策略之間存在著一對一的關係。

圖形 H 的節點集合 U 中每一節點 均有二種成本 *net(v)* 及 *comp(v)*，令 $\text{CR(X,Y)}=\{v|(v_{\text{b}}^{(x)},v_{\text{e}}^{(x)})\in E, v_{\text{b}}^{(x)}\in X, v_{\text{e}}^{(x)}\in Y\}$為 N_x 集合。H-cut 的成本公式定義如下：

$$\sum_{v\in CR(X,Y)} net(v_{\text{b}}^{(x)},v_{\text{e}}^{(x)})+\sum_{v\in X} comp(v_{\text{e}},n_{\text{e}})+\sum_{v\in Y} comp(1_{\text{b}},v_{\text{b}}) \quad (2)$$

本論文中，*最小切割*就是在所有 H-cut 中成本最小的那組部署策略。上述 H-cut 與傳統切割(graph cut)定義稍有不同，根據[4]，最小 H-cut 可以利用現有的最大流量演算法解出。我們用水管來比喻，瞬間最大流量就是水管最窄能通過的地方在單一時間下流出的最大流量，H-cut 就是在圖上，找出水管最窄流量的地方。

在子圖 $G^{(k)}$ 中橫跨 H-cut(*X,Y*)切割線的物件 V 集合為 $Z^{(k)}$，$Z^{(k)}$ 將子圖 $G^{(k)}$ 切割成 $X^{(k)},Y^{(k)}$ 二個部分，ODG 圖形與子圖 $G^{(k)}$ 之間的相對應關係與副本策略如下：

- 如果 edge($v_b^{(x)}, v_e^{(x)}$) 在切割線下方，也就是切割子圖$Y^{(k)}$部分的區域，物件v就是一個 *C-node*。
- 如果 edge($v_b^{(x)}, v_e^{(x)}$)橫跨 H-cut 切割線，這類節點在子圖$G^{(k)}$中的集合為$Y^{(k)}$，物件v就是一個 *N-node*。
- 如果 edge($v_b^{(x)}, v_e^{(x)}$)在 H-cut 切割線上方，也就是切割子圖$X^{(k)}$部分的區域，物件v就是一個 *O-node*。

假設有 M+1 台主機，k = 0,…,M，令$(X^{(k)}, Y^{(k)})$為 H-cut (X,Y) 在子圖$G^{(k)}$上的子切割，$Z^{(k)}$為 H-cut (X,Y)的切割線段的節點集合， H-cut 切割的切割線段$Z^{(k)}$對應到 N_k，切割的$X^{(k)}$(上方)與$Y^{(k)}$(下方)分別對應到 O_k與 C_k 。假設 S_0是 server 主機，$S_1,\ldots,S_M$是 client 主機， $1 \le n \le M$，若是節點 v 在$Z^{(n)}$下方，則節點 v 必定也在$Z^{(0)}$下方，否則∞-edge (v_e^0, v_e^n) (在E_{DOWN}中)將會被切割，違反特性 P1。H-cut (X,Y)對應到以下部署策略：

$$N_0 = \{v \mid v^{(0)} \in Z^{(0)}\}, C_0 = \{v \mid v^{(0)} \in Y^{(0)}\}$$

$$N_1 = \{v \mid v^{(1)} \in Z^{(1)}\}, C_1 = \{v \mid v^{(1)} \in Y^{(1)}\}$$

…

$$N_M = \{v \mid v^{(M)} \in Z^{(M)}\}, C_M = \{v \mid v^{(M)} \in Y^{(M)}\}$$

(2)

在圖形 H 中公式(2)的部署策略符合以下定理：

定理 1 圖形 H 的 H-cut (*X*,*Y*) 對應到公式 (2) 中的合法部署策略 $(N_0,C_0),(N_1,C_1),\ldots(N_M,C_M)$ 。

證明： 要證明部署策略是合法的，我們須驗證定義 1 的四項條件。

以下討論假設 $k = 0,1,\cdots,M$。首先觀察圖 3.1-2 的每組切割 $(X^{(k)}, Y^{(k)})$，並對照以下部署策略:

$N_0 = \{2,3\}, C_0 = \{4,5,6\}$

$N_1 = \{2,3\}, C_1 = \{4,5,6\}$

$N_2 = \{2,3\}, C_2 = \{4,5,6\}$

$N_3 = \{4,5\}, C_3 = \{6\}$

圖 3.1-2 中，節點 4 在 $G^{(2)}$ 切割下方，節點 4 也在 $G^{(0)}$ 切割下方。現在，對應到部署策略中，節點 4 在主機 S_2 的 C_2 集合內，節點 4 也必定在 server 主機 S_0 的 C_0 集合。這特性寫成 $(N_2 \cup C_2) \subseteq (N_0 \cup C_0)$，保證了 server 主機 S_0 的部署範圍足以供應 client 主機 S_2 的部署範圍。

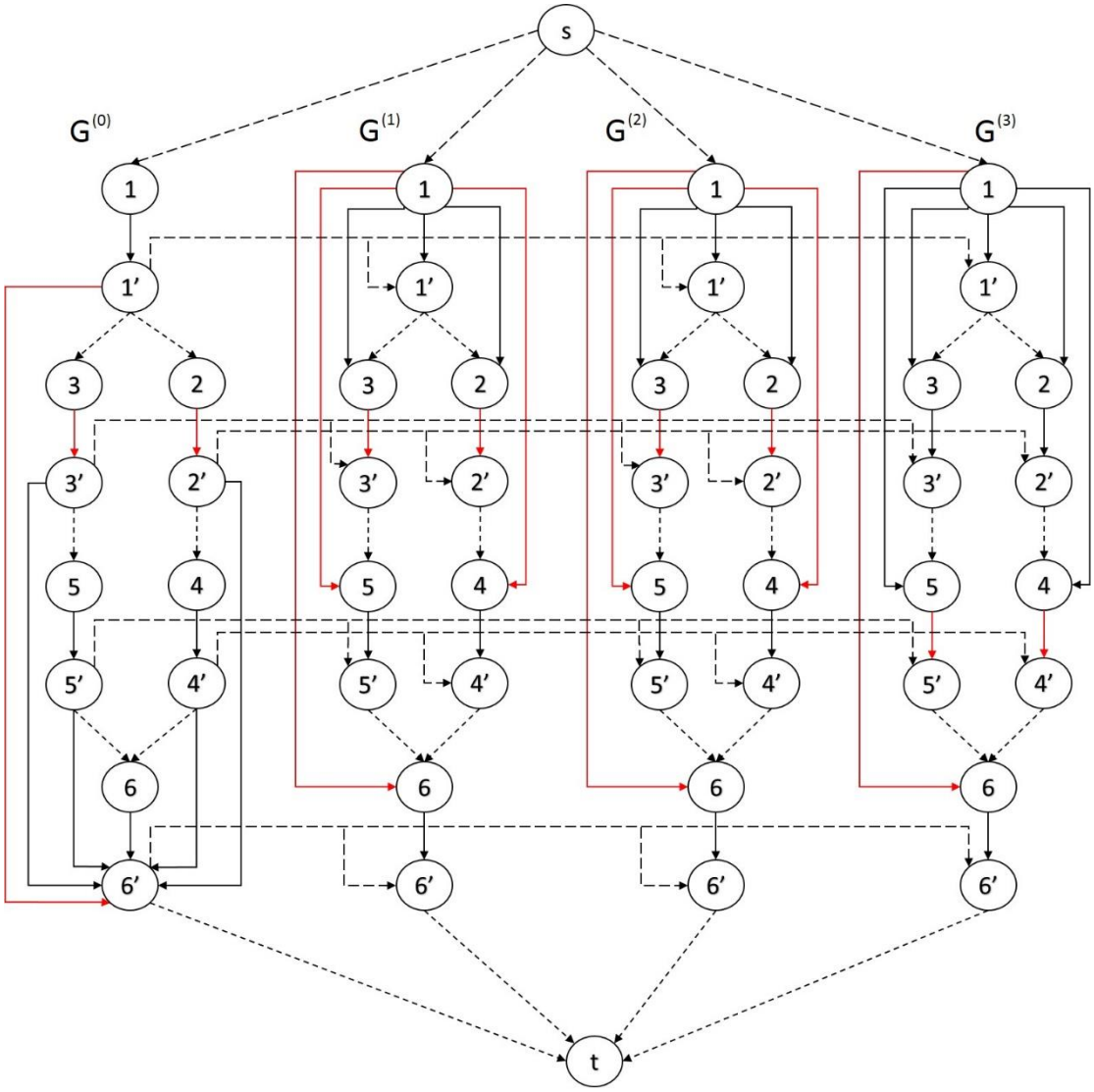


圖 3.1-2：圖 H

3.2 漸進式最小切割演算法

圖形 H 的部署成本分為網路成本 $net(v)$ 與物件成本 $comp(v)$ ，而 w_x 、 k_x 分別為主機 S_x 在網路成本和物件成本中的成本係數，假設網路 S 有 M + 1 個主機 S_0, S_1 ,… , S_M ，而且這 M+1 台主機被組織為樹狀的主從式架構，以 S_0 為原始主機 (Server 端)，S_1 ,… , S_M 為終端主機(Client 端)， w_0 , w_1 , … , w_M 為主機 S_0 , S_1 , … ,

S_M 在網路成本中的成本係數，$k_0, k_1, \ldots, k_M$ 則為主機 $S_0, S_1, \ldots, S_M$ 在物件成本中的成本係數。漸進式最小切割演算法就是在圖形 H 中先求出 S_x 主機的 H-cut (X,Y)，H-cut (X,Y)將 S_x 主機切割成 $X^{(x)}$與 $Y^{(x)}$二個子切割圖形，再從 $Y^{(x)}$圖形中求出 S_{x+1} 主機的 H-cut (X,Y)。

圖形 H 的 H-cut (X, Y) 對應到公式 (2) 中的合法部署策略 $(N_0,C_0),(N_1,C_1),\ldots(N_M,C_M)$，$(X^{(k)},Y^{(k)})$ 為 H-cut (X,Y) 在子圖 $G^{(k)}$ 上的子切割圖形，圖形 H 必定符合以下*漸進式切割*條件：

- $X^{(1)} \subseteq X^{(2)} \subseteq \ldots \subseteq X^{(M)}$ 和 $Y^{(M)} \subseteq Y^{(M-1)} \subseteq \ldots \subseteq Y^{(1)}$。

也就是說當一台主機 S_x 的的 *kw 因子*定義為 $\frac{k_x}{w_x}$。*kw 因子*愈小表示該主機傾向於計算物件，反之則傾向於傳送物件。

我們把終端主機依照 $S_1, \ldots, S_3$ 排序，使 $S_1, \ldots, S_3$ 的 *kw 因子*為 $\frac{k_1}{w_1} \le \frac{k_2}{w_2} \le \frac{k_3}{w_3}$，如果圖形 H 的終端主機切割如圖 3.2 (a)，則 $X^{(1)} \subseteq X^{(2)} \subseteq X^{(3)}$，且 $Y^{(3)} \subseteq Y^{(2)} \subseteq Y^{(1)}$。這表示 S_2 比 S_3 主機傾向於計算物件，而 S_1 又比 S_2、S_3 更傾向於自己在主機做計算，假設在 S_3 主機有更多部分使用計算物件且成本比較划算，如圖 3.2 (b)，則 $Y^{(2)} \subseteq Y^{(N)}$ 並不符合定理 2，如要符合定理 2，則 $Y^{(N)} \subseteq Y^{(2)}$，表示在 S_2 主機做計算的部分必須大於等於 S_3 主機才符合 $Y^{(N)} \subseteq Y^{(2)} \subseteq Y^{(1)}$。

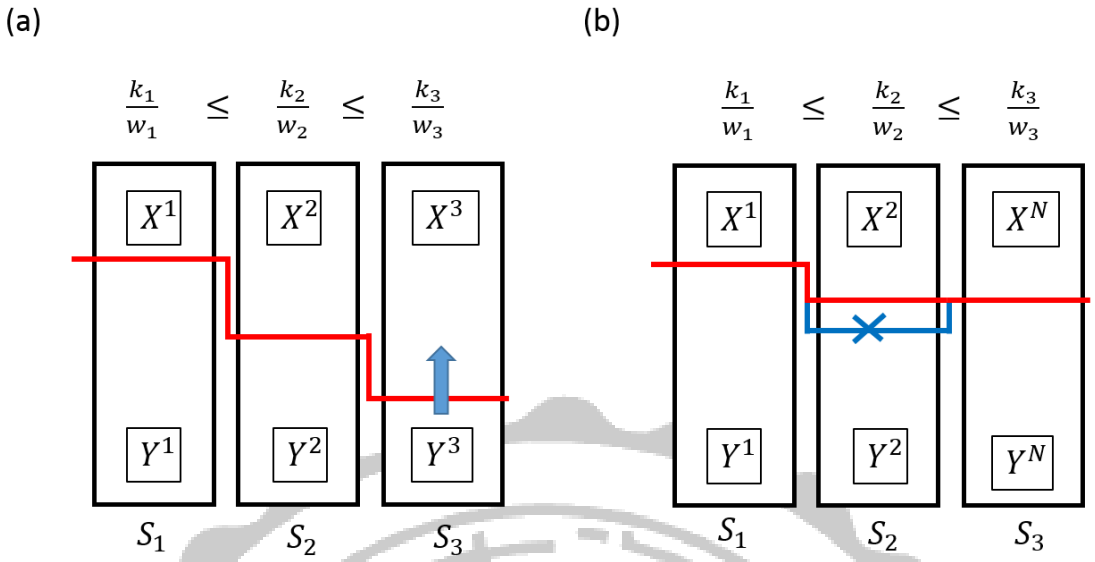


圖 3.2：終端主機切割圖：(a)漸進式切割；(b) S_2 切割必須高於 S_3 切割。

定理 2：給定一個網路 S 有 M+1 個樹狀的主從式結構主機 $S_0, S_1, \ldots, S_M$，令 H 為網路 S 與 ODG G 的部署轉換圖形。若 $\frac{k_1}{w_1} \leq \frac{k_2}{w_2} \leq \ldots \leq \frac{k_M}{w_M}$，則 minimum H-cut (X,Y) 必定滿足漸進式切割條件：

- $X^{(1)} \subseteq X^{(2)} \subseteq \ldots \subseteq X^{(M)}$ 而且 $Y^{(M)} \subseteq Y^{(M-1)} \subseteq \ldots \subseteq Y^{(1)}$。

證明：

令 $\text{comp}(A) = \sum_{v_i \in A} c_i$ and $\text{net}(A) = \sum_{v_i \in A} n_i$。

則 $X^{(1)}, Y^{(1)}$ 的成本為

$$k_1 \cdot \text{comp}(Y^1) + w_1 \cdot \text{net}(Z^1)$$

$X^{(2)}, Y^{(2)}$ 的成本為

$$k_2 \cdot \text{comp}(Y^2) + w_2 \cdot \text{net}(Z^2)$$

我們把終端主機依照 $S_1, \ldots, S_3$ 排序，$S_1, \ldots, S_3$ 的成本係數關係滿足條件 $\frac{k_1}{w_1} \leq \frac{k_2}{w_2}$，這表示 S_1 比 S_2 更傾向於自己在主機做計算，假設在 S_1 主機成本比較划算

的形況下，有更多部分使用主機做計算，也就是 $Y^{(2)} \subseteq Y^{(1)}$。

已知以下條件成立

$$k_2 \cdot \text{comp}(Y^2) + w_2 \cdot \text{net}(Z^2)$$

$$\leq k_2 \cdot \text{comp}(Y^1) + w_2 \cdot \text{net}(Z^1)$$

可以推導出

$$k_2 \cdot \text{comp}(Y^2) + w_2 \cdot \text{net}(Z^2) \leq k_2 \cdot \text{comp}(Y^1) + w_2 \cdot \text{net}(Z^1)$$

$$k_2 \cdot \big(\text{comp}(Y^2) - \text{comp}(Y^1)\big) \leq w_2 \cdot (\text{net}(Z^1) - \text{net}(Z^2))$$

$$\frac{k_2}{w_2} \leq \frac{\text{net}(Z^1) - \text{net}(Z^2)}{\text{comp}(Y^2) - \text{comp}(Y^1)}$$

因為 $\text{comp}(Y^2) - \text{comp}(Y^1) > 0$。

以及

$$k_1 \cdot \text{comp}(Y^1) + w_1 \cdot \text{net}(Z^1) \leq k_1 \cdot \text{comp}(Y^2) + w_1 \cdot \text{net}(Z^2)$$

可以推導出

$$k_1 \cdot \big(\text{comp}(Y^1) - \text{comp}(Y^2)\big) \leq w_1 \cdot (\text{net}(Z^2) - \text{net}(Z^1))$$

$$\frac{k_1}{w_1} \geq \frac{\text{net}(Z^2) - \text{net}(Z^1)}{\text{comp}(Y^1) - \text{comp}(Y^2)}$$

因為 $\text{comp}(Y^1) - \text{comp}(Y^2) < 0$。■

在傳統的部署策略演算法中我們直接從圖形 H 中求最小切割，所需時間為 $O(N^2M)$，$N = \text{c} \cdot \text{n}$，$M = \text{c} \cdot \text{m}$，而漸進式最小切割演算法[4]的時間複雜度則是 $O(n^2m)$，n 為 ODG 節點的個數，m 為 ODG *edges* 的個數。漸進式最小切割演算法因考量到主機的計算與傳送成本，先行在別台成本更低的主機做過計算，而不會有重複計算與更高成本計算的問題，會比傳統切割演算法還要來得節省成本與更有效率。

3.3 部署策略演算法之探討

根據我們及多位研究學者的研究經驗，上述 3.1 節 minimum cut [1]與部署策略的轉換可以延伸到多種不同的網路之中，表 3.3 整理了這些應用。

網路結構	說明
Client-Server 網路	最佳化演算法[5]。
樹狀結構網路	簡化型演算法，方法如 3.1 節所描述。能有效率地求出任意樹狀結構網路的部署策略。
線性結構網路	運用線性結構，簡化節點與線的數量，演算法在[7] 。

表 3.3

表 3.3 所列之方法皆是 ODG G 與網路結構圖 S 轉換成圖形 H，然後再求出圖形 H 的 minimum cut。

四、實驗結果

實驗參數設定包括 ODG 圖形、節點數量、成本與網路結構圖。首先設定 ODG 參數與網路拓樸後，讀入預設元件成本到每個 ODG 節點，並調整不同主機的傳送與計算成本係數，經過乘積後會產生整合圖 H，並根據不同成本的部署策略來分析成本的相對關係。

4.1 Graphviz

Graphviz 是一個運用廣泛的圖形視覺化軟體，使用 dot 語言來描述圖形，我們可以用文字描述 nodes 之間的關係畫出複雜的關係圖，dot 語言能直接陳述圖片上的節點、邊、方向等性質，我們可以利用這套軟體畫出複雜的網路關係圖，觀察網路與 ODG 元件的相依關係與不同成本的部署策略變化。

4.2 實驗建置

進行實驗之前，在程式中預設每個 ODG 元件的傳送成本 n 與計算成本 c，並設置 ODG 的節點數量，程式產生網路結構圖(如圖 4.2 (a))與 ODG 圖形(如圖 4.2 (b))，並依照實驗需求調整主機 S_x 的傳送成本參數 w 與計算成本參數 k，接著執行 maxflow 演算法程式找出最大流量與最小切割部署策略(N_x,C_x)。程式執行後會產生一個文字檔紀錄元件部署圖、元件成本與網路流量，我們將 output 文字檔複製到 Graphviz 軟體執行，在把文字檔轉化為圖形，將不同組主機成本計算出來的部署策略(N_x,C_x)結果做比較，依照問題需求來調整主機的網路傳送成本與主機計算成本。

(a) (b)

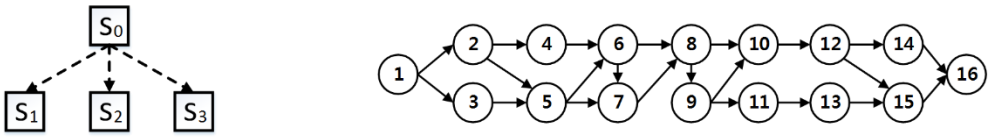


圖 4.2：(a)網路結構圖 S　(b)ODG G

本實驗使用 AMD 六核處理器與 16GB 記憶體在 Windows7 上執行，maxflow 演算法在 Eclipse 開發使用 JAVA 程式撰寫。實驗分別測試三組固定的元件成本 n 和 c，並視情況調整主機的成本係數 w 和 k，並觀察其結果與變化。

4.3 實驗結果

實驗在 client-server 架構上實行，由 1 台 server 主機連接 3 台 client 主機，每台主機有固定的元件成本(*n* 和 *c*)和二組可變化的成本係數(*w* 和 *k*)，以下(表 4.3-1)為實驗的三組固定元件成本，每個主機有 16 個元件節點，*n* 為元件的檔案大小，c 為元件的工作時間，我們隨機調整主機的成本係數 *w* (主機的網路成本)和 *k*(主機的計算成本)，執行 maxflow 演算法程式，並將部署結果以 Graphviz 軟體轉為圖檔。

	n	*c*
A 組	{1,2,3,4,5,6,7,8,1,2,3,4,5,6,7,8}	{6,1,6,1,6,1,6,1,6,7,9,9,8,9,2}
B 組	{8,2,3,4,5,6,7,8,8,6,3,4,5,2,7,8}	{1,2,6,3,5,1,6,8,1,7,9,6,1,7,2}
C 組	{1,2,1,3,1,6,1,8,4,2,8,5,1,2,5,3}	{2,2,4,3,5,1,6,1,1,2,3,2,1,8,2}

表 4.3-1

每組實驗找出三張部署策略圖，每張部署策略圖中有四個子圖，分別為 *Server* (w_0 , k_0) 、*Client 1* (w_1 , k_1) 、*Client 2* (w_2 , k_2) 、*Client 3* (w_3 , k_3)，我們調整 *w* 和 *k* 並觀察其變化，以下為 A、B、C 三組實驗的結果。

A 組實驗的成本係數：

	Server (w_0 , k_0)	*Client 1* (w_1 , k_1)	*Client 2* (w_2 , k_2)	*Client 3* (w_3 , k_3)
A1	(1 , 1)	(20 , 1)	(22 , 2)	(3 , 1)
A2	(1 , 1)	(30 , 2)	(10 , 1)	(4 , 2)
A3	(1 , 1)	(10 , 1)	(10 , 2)	(4 , 3)

表 4.3-2

在 A 組實驗中，我們找出三張部署策略圖，分別為 A1、A2、A3，每張部署策略圖的 *w*、*k* 係數如表 4.3-2。

實驗 A1(如圖 4.3-1)的 4 個主機系數分別為 *Server* (w_0 , k_0) = (1 , 1)與 *Client* 1 (w_1 , k_1) = (20 , 1) 、*Client* 2 (w_2 , k_2) = (22 , 2) 、*Client* 3 (w_3 , k_3) = (3 , 1)。

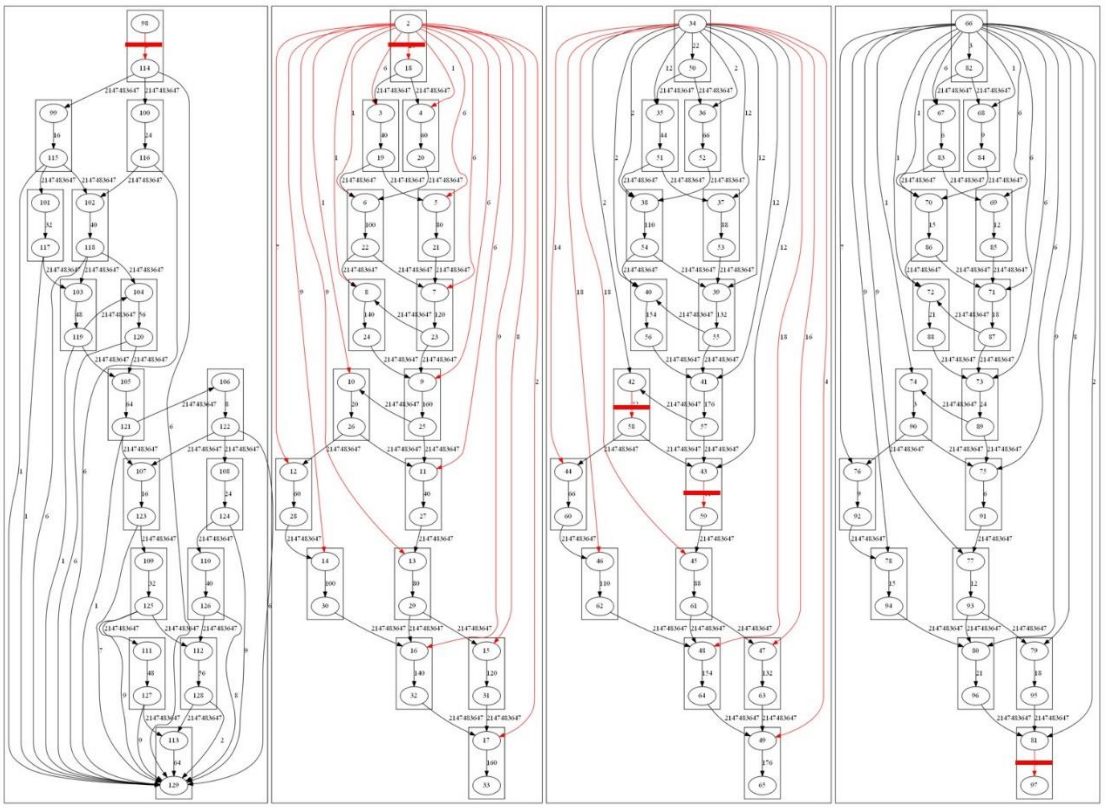


圖 4.3-1：實驗 A1 部署策略圖

實驗 A2(如圖 4.3-2)的 4 個主機系數分別為 *Server* (w_0 , k_0) = (1 , 1)與 *Client* 1 (w_1 , k_1) = (30 , 2) 、*Client* 2 (w_2 , k_2) = (10 , 1) 、*Client* 3 (w_3 , k_3) = (4 , 2)。

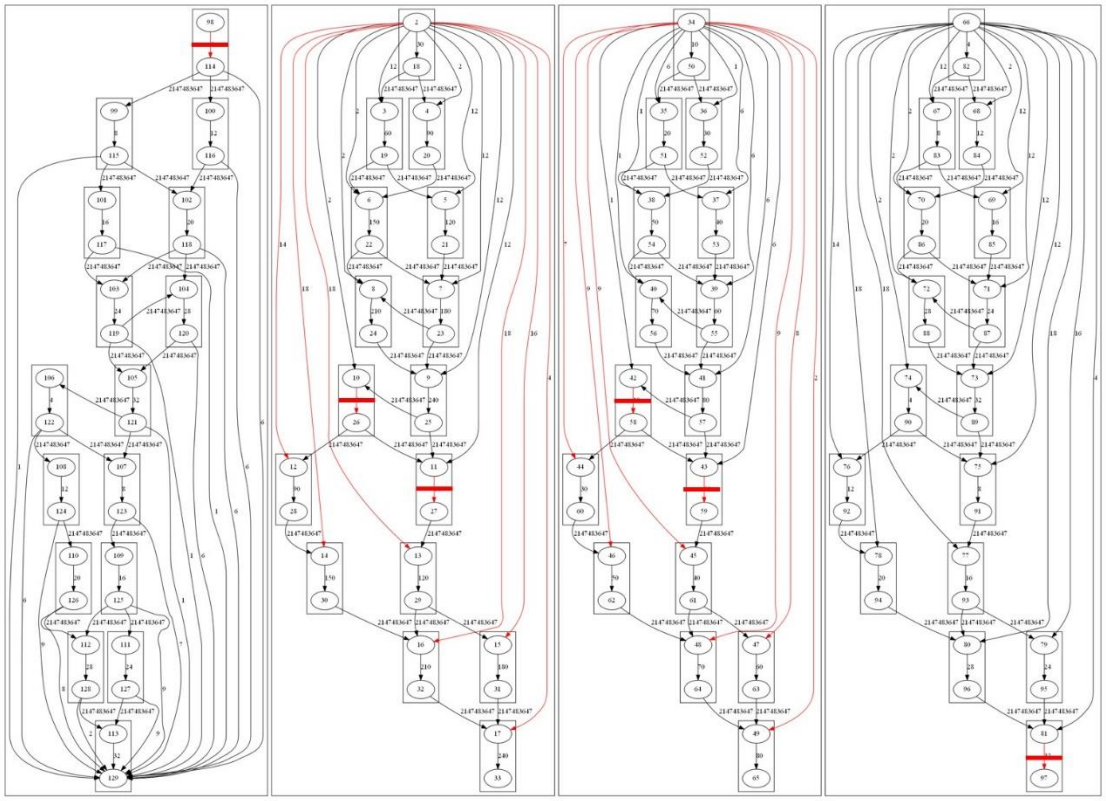
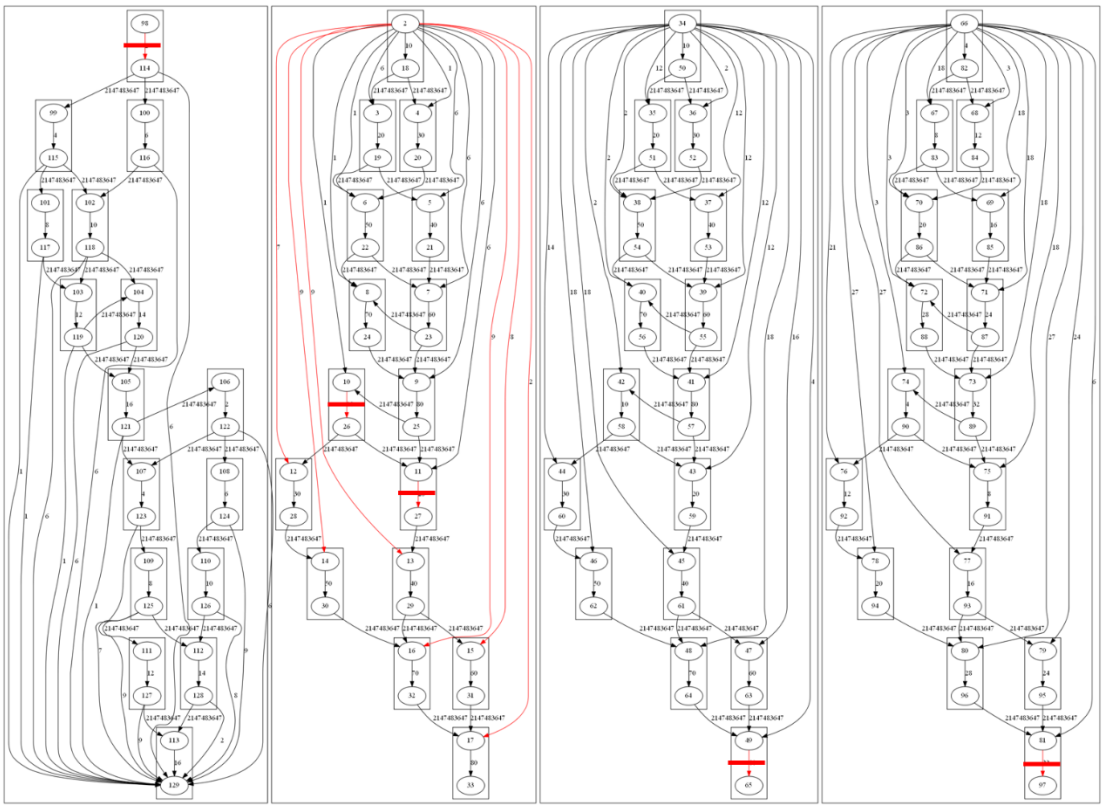


圖 4.3-2：實驗 A2 部署策略圖

實驗 A3(如圖 4.3-3)的 4 個主機系數分別為 *Server* (w_0 , k_0) = (1 , 1)與 *Client* 1 (w_1 , k_1) = (10 , 1) 、*Client* 2 (w_2 , k_2) = (10 , 2) 、*Client* 3 (w_3 , k_3) = (4 , 3)。



Server (w_0 , k_0) = (1, 1)　　*Client 1* (w_1 , k_1) = (10 , 1)　　*Client 2* (w_2 , k_2) = (10 , 2)　　*Client 3* (w_3 , k_3) = (4 , 3)

圖 4.3-3：實驗 A3 部署策略圖

我們把 A 組實驗中 A1、A2、A3 的 k 與 w 係數做相除($\frac{k}{w}$)，得到表 4.3-3 結果，觀察 A 組實驗的係數與部署策略圖的變化。

	Server (k_0/w_0)	*Client 1* (k_1/w_1)	*Client 2* (k_2/w_2)	*Client 3* (k_3/w_3)
A1	$\frac{1}{1}$	$\frac{1}{20}$	$\frac{2}{22}$	$\frac{1}{3}$
A2	$\frac{1}{1}$	$\frac{2}{30}$	$\frac{1}{10}$	$\frac{2}{4}$
A3	$\frac{1}{1}$	$\frac{1}{10}$	$\frac{2}{10}$	$\frac{3}{4}$

表 4.3-3

從表 4.3-3 中我們可以發現當實驗 A1、A2、A3 的 *Client* 主機係數相除結果關係為 *Client 1* (k_1/w_1) < *Client 2* (k_2/w_2) < *Client 3* (k_3/w_3) 時，部署策略圖中 *Client 1*、*Client 2*、*Client 3* 的流量切割線會逐漸往下切割。

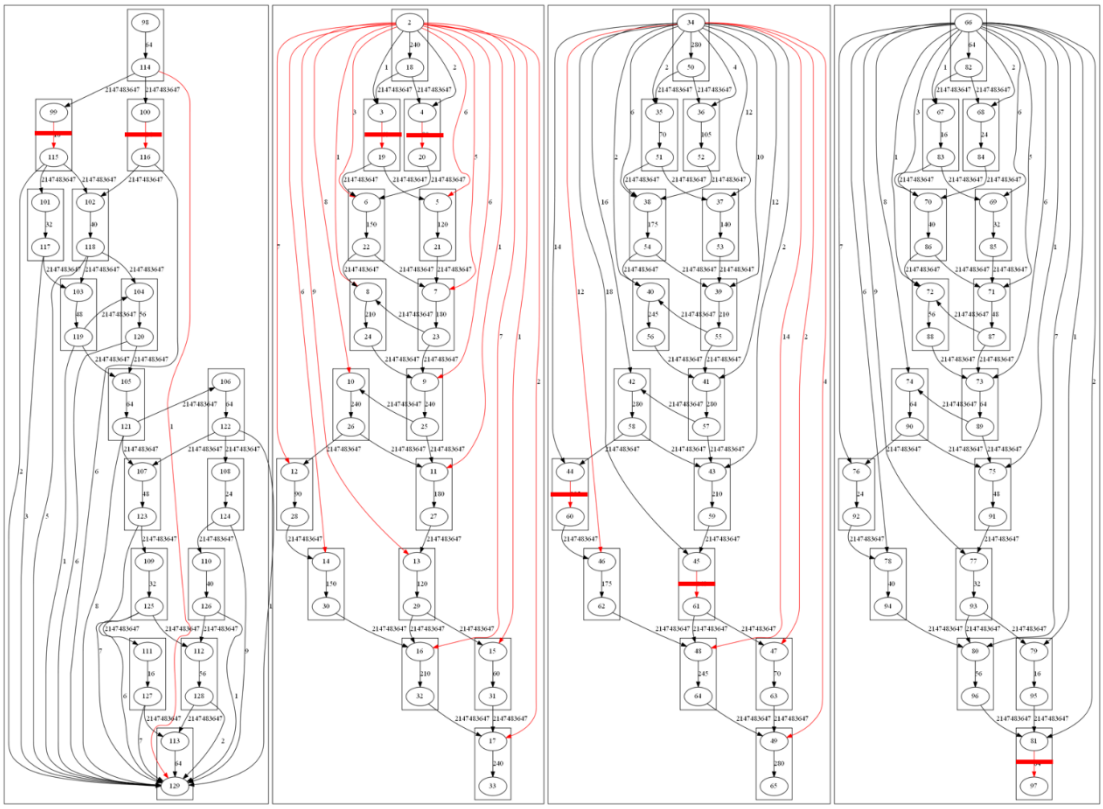
B 組實驗的成本係數：

	Server (w_0 , k_0)	*Client 1* (w_1 , k_1)	*Client 2* (w_2 , k_2)	*Client 3* (w_3 , k_3)
B1	(1 , 1)	(30 , 1)	(35 , 2)	(8 , 1)
B2	(1 , 1)	(42 , 2)	(16 , 1)	(10 , 2)
B3	(1 , 1)	(20 , 1)	(10 , 1)	(4 , 1)

表 4.3-4

在 B 組實驗中，我們找出三張部署策略圖，分別為 B1、B2、B3，每張部署策略圖的 w、k 係數如表 4.3-4。

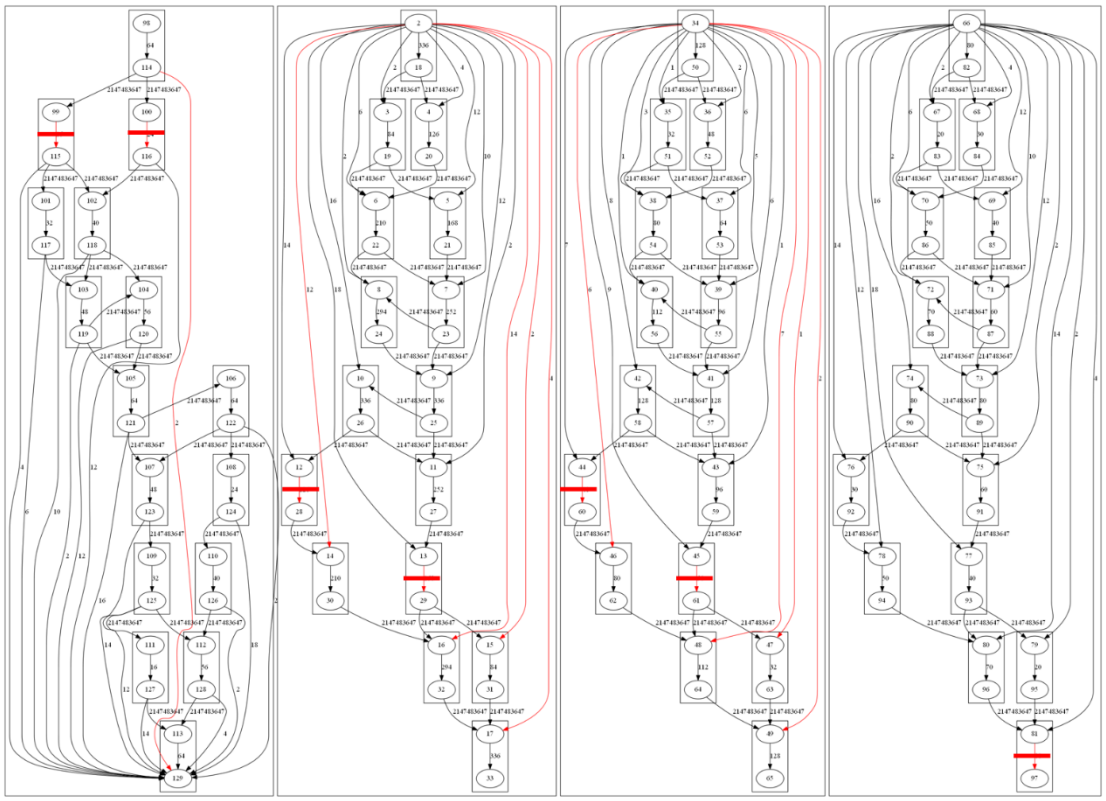
實驗 B1(如圖 4.3-4)的 4 個主機系數分別為 *Server* (w_0 , k_0) = (1 , 1)與 *Client* 1 (w_1 , k_1) = (30 , 1) 、*Client* 2 (w_2 , k_2) = (35 , 2) 、*Client* 3 (w_3 , k_3) = (8 , 1)。



Server (w_0 , k_0) = (1 , 1)　　*Client 1* (w_1 , k_1) = (30 , 1)　　*Client 2* (w_2 , k_2) = (35 , 2)　　*Client 3* (w_3 , k_3) = (8 , 1)

圖 4.3-4：實驗 B1 部署策略圖

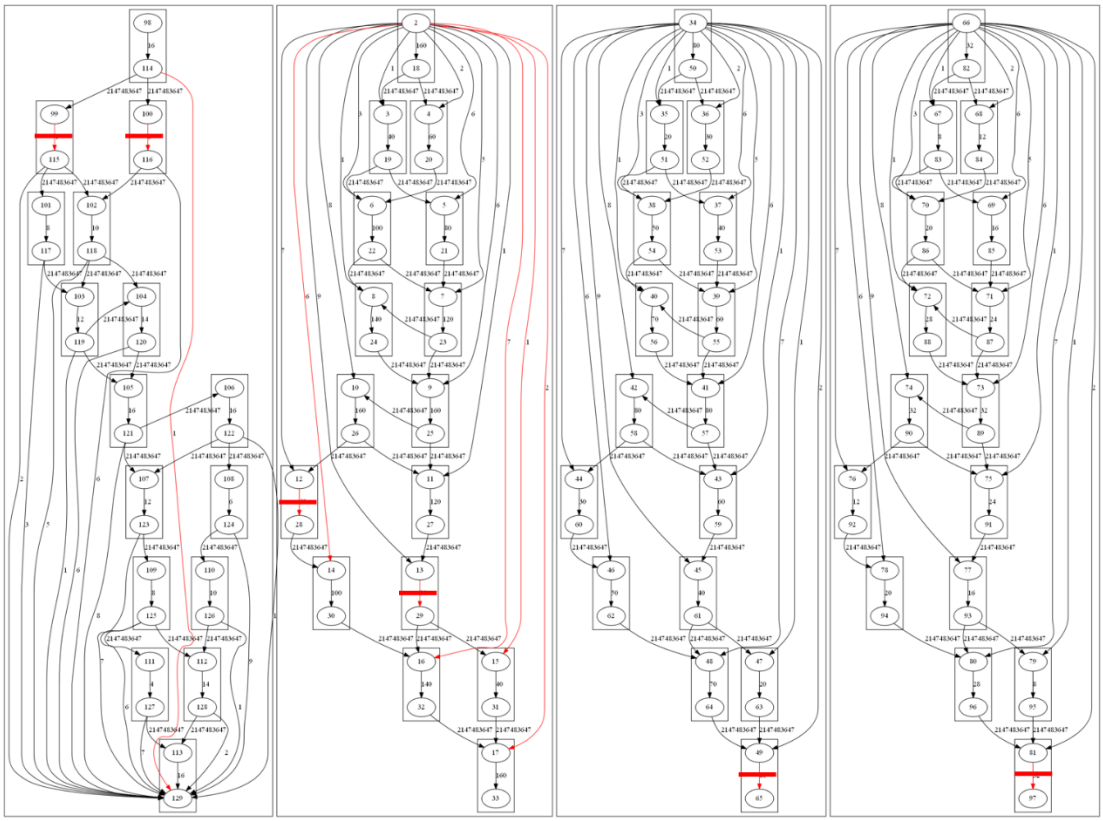
實驗 B2(如圖 4.3-5)的 4 個主機系數分別為 *Server* (w_0 , k_0) = (1 , 1)與 *Client* 1 (w_1 , k_1) = (42 , 2) 、*Client* 2 (w_2 , k_2) = (16 , 1) 、*Client* 3 (w_3 , k_3) = (10 , 2)。



Server (w_0 , k_0) = (1 , 1)　　*Client 1* (w_1 , k_1) = (42 , 2)　　*Client 2* (w_2 , k_2) = (16 , 1)　　*Client 3* (w_3 , k_3) = (10 , 2)

圖 4.3-5：實驗 B2 部署策略圖

實驗 B3(如圖 4.3-6)的 4 個主機系數分別為 *Server* (w_0 , k_0) = (1 , 1)與 *Client* 1 (w_1 , k_1) = (20 , 1) 、*Client* 2 (w_2 , k_2) = (10 , 1) 、*Client* 3 (w_3 , k_3) = (4 , 1)。



Server (w_0 , k_0) = (1 , 1)　*Client 1* (w_1 , k_1) = (20 , 1)　*Client 2* (w_2 , k_2) = (10 , 1)　*Client 3* (w_3 , k_3) = (4 , 1)

圖 4.3-6：實驗 B3 部署策略圖

我們把 B 組實驗中 B1、B2、B3 的 k 與 w 係數做相除($\frac{k}{w}$)，得到表 4.3-5 結果，觀察 B 組實驗的係數與部署策略圖的變化。

	Server (k_0/w_0)	*Client 1* (k_1/w_1)	*Client 2* (k_2/w_2)	*Client 3* (k_3/w_3)
B1	$\frac{1}{1}$	$\frac{1}{30}$	$\frac{2}{35}$	$\frac{1}{8}$
B2	$\frac{1}{1}$	$\frac{2}{42}$	$\frac{1}{16}$	$\frac{2}{10}$
B3	$\frac{1}{1}$	$\frac{1}{20}$	$\frac{1}{10}$	$\frac{1}{4}$

表 4.3-5

從表 4.3-5 中我們可以發現當實驗 B1、B2、B3 的 Client 主機係數相除結果關係為 *Client 1* (k_1/w_1) < *Client 2* (k_2/w_2) < *Client 3* (k_3/w_3) 時，部署策略圖中 Client 1、Client 2、Client 3 的流量切割線會逐漸往下切割。

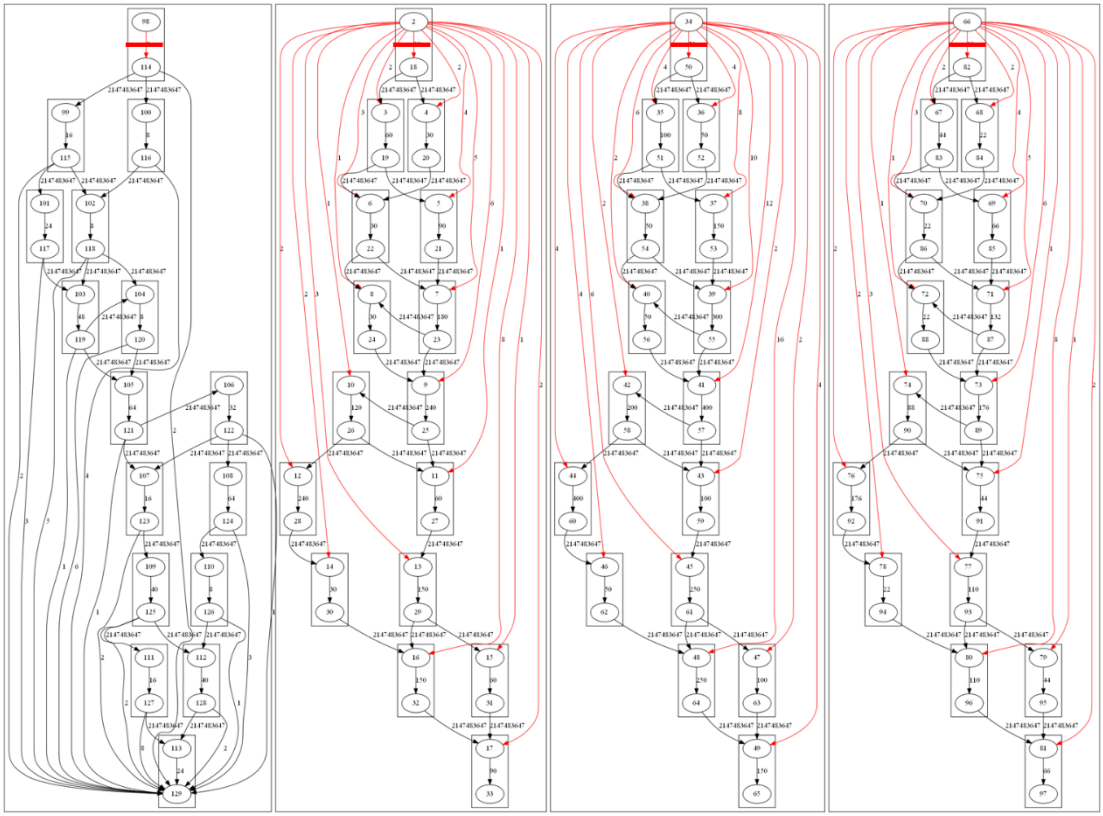
C 組實驗的成本係數：

	Server (w_0 , k_0)	*Client 1* (w_1 , k_1)	*Client 2* (w_2 , k_2)	*Client 3* (w_3 , k_3)
C1	(1 , 1)	(30 , 1)	(50 , 2)	(22 , 1)
C2	(1 , 1)	(48 , 2)	(22 , 1)	(10, 1)
C3	(1 , 1)	(45 , 2)	(20 , 1)	(4 , 2)

表 4.3-6

在 C 組實驗中，我們找出三張部署策略圖，分別為 C1、C2、C3，每張部署策略圖的 w、k 係數如表 4.3-6。

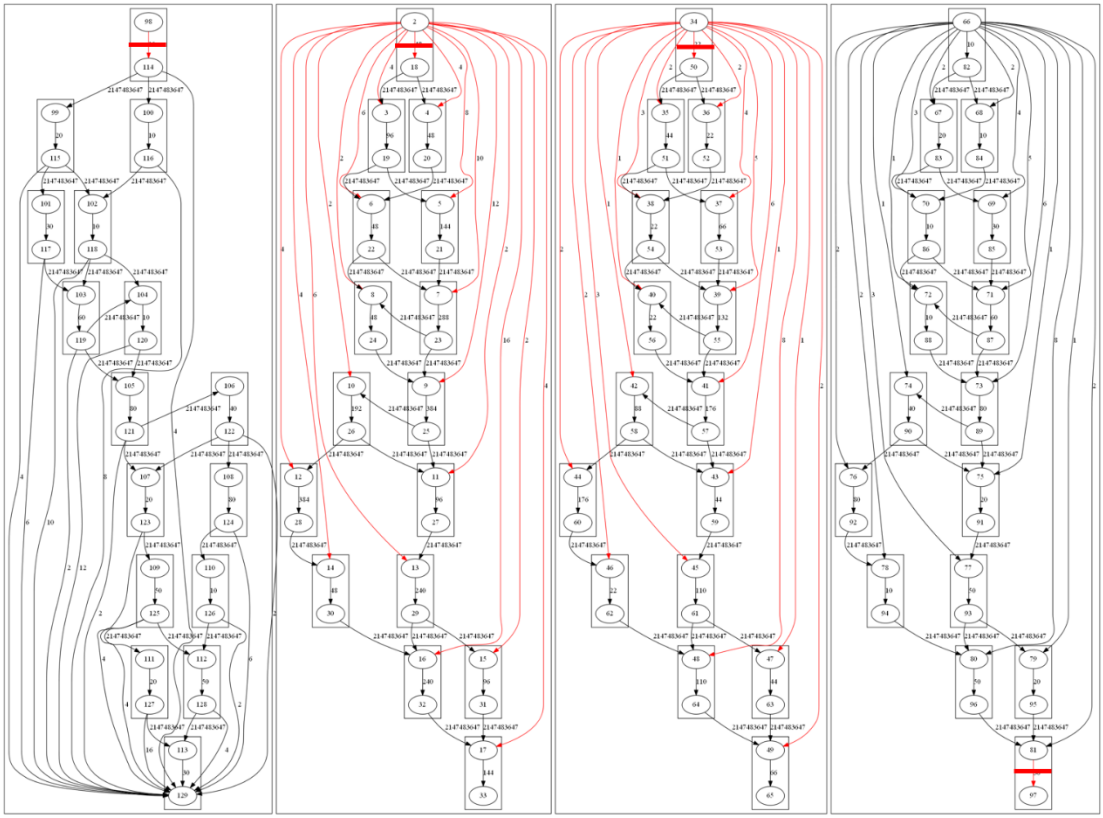
實驗 C1(如圖 4.3-7)的 4 個主機系數分別為 *Server* (w_0 , k_0) = (1 , 1)與 *Client* 1 (w_1 , k_1) = (30 , 1) 、*Client* 2 (w_2 , k_2) = (50 , 2) 、*Client* 3 (w_3 , k_3) = (22 , 1)。



Server (w_0 , k_0) = (1 , 1)　　*Client 1* (w_1 , k_1) = (30 , 1)　　*Client 2* (w_2 , k_2) = (50 , 2)　　*Client 3* (w_3 , k_3) = (22 , 1)

圖 4.3-7：實驗 C1 部署策略圖

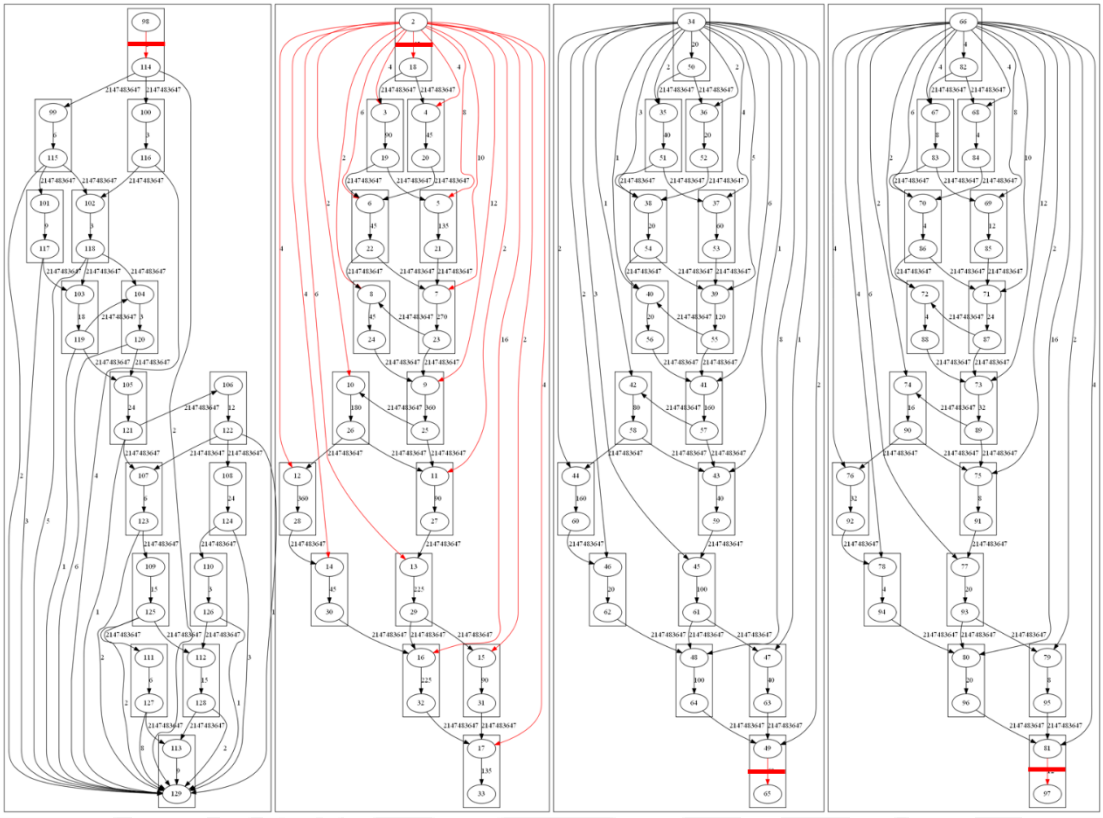
實驗 C2 (如圖 4.3-8)的 4 個主機系數分別為 *Server* (w_0 , k_0) = (1 , 1)與 *Client* 1 (w_1 , k_1) = (48 , 2) 、*Client* 2 (w_2 , k_2) = (22 , 1) 、*Client* 3 (w_3 , k_3) = (10 , 1)。



Server (w_0 , k_0) = (1 , 1)　　*Client 1* (w_1 , k_1) = (48 , 2)　　*Client 2* (w_2 , k_2) = (22 , 1)　　*Client 3* (w_3 , k_3) = (10 , 1)

圖 4.3-8：實驗 C2 部署策略圖

實驗 C3(如圖 4.3-9)的 4 個主機系數分別為 *Server* (w_0 , k_0) = (1 , 1)與 *Client* 1 (w_1 , k_1) = (45 , 2) 、*Client* 2 (w_2 , k_2) = (20 , 1) 、*Client* 3 (w_3 , k_3) = (4 , 2)。



Server (w_0 , k_0) = (1 , 1)　　*Client 1* (w_1 , k_1) = (45 , 2)　　*Client 2* (w_2 , k_2) = (20 , 1)　　*Client 3* (w_3 , k_3) = (4 , 2)

圖 4.3-9：實驗 C3 部署策略圖

我們把 C 組實驗中 C1、C2、C3 的 *k* 與 *w* 係數做相除($\frac{k}{w}$)，得到表 4.3-7 結果，觀察 C 組實驗的係數與部署策略圖的變化。

	Server (k_0/w_0)	*Client 1* (k_1/w_1)	*Client 2* (k_2/w_2)	*Client 3* (k_3/w_3)
C1	$\frac{1}{1}$	$\frac{1}{30}$	$\frac{2}{50}$	$\frac{1}{22}$
C2	$\frac{1}{1}$	$\frac{2}{48}$	$\frac{1}{22}$	$\frac{1}{10}$
C3	$\frac{1}{1}$	$\frac{2}{45}$	$\frac{1}{20}$	$\frac{2}{4}$

表 4.3-7

從表 4.3-7 中我們可以發現當實驗 C1、C2、C3 的 *Client* 主機係數相除結果關係為 *Client 1* (k_1/w_1) < *Client 2* (k_2/w_2) < *Client 3* (k_3/w_3) 時，部署策略圖中 *Client 1*、*Client 2*、*Client 3* 的流量切割線會逐漸往下切割。

從 A、B、C 三組實驗中我們可以發現當 *Client 1* 、 *Client 2* 、 *Client 3* 依次排序，且 $\frac{k_1}{w_1} < \frac{k_2}{w_2} < \frac{k_3}{w_3}$ 時，部署策略的切割線會逐漸往下切。這表示在傳送成本相當的情況下，當 *client* 主機的計算成本大於其他主機，它直接從網路上取得資料並直接傳送到主機比較划算；相反的，當 *client* 主機的計算成本小於其他主機，表示盡量在 *Client* 主機做計算會比較划算。

五、結論

隨著雲端技術的發展，企業為節省成本開始租用雲端主機服務，而應用程式中的服務部署對程式的效能與成本造成關鍵性的影響，我們在計算成本相對較低的主機中多做計算，讓計算成本相對較高的主機盡量以傳送資料為主，使服務流程元件可以在主從式網路的架構，求得較少的服務元件部署成本。從實驗中我們發現應用程式中各個流程服務元件與成本的關連性，漸進式最小切割演算法評估不同主機的計算與傳送成本來做取捨，並同時部署多台主機，先行在別台成本更低的主機先做計算，而不會有重複計算與更高成本計算的問題，會比傳統切割演算法還要來得節省成本與更有效率。

參考文獻

[1] R.K. Ahuja T.L. Magnanti and J.B. Orlin. Network Flows:Theory, Algorithms, and Applications. Prentice-Hall, 1993.

[2] A.Ehuchi S.Fujishige and T.Takabatake. , “A polynomial-time algorithm for the generalized independent-flow problem,”Journal of the Operations Research ,Vol .47:pp.1–17, 2004

[3] K.H. Y.L.Chin and W.Zhang, “Multimedia object placement for transparent data replication,” IEEE Trans. Parallel and Distributed Systems, vol.18,no.2, pp.212-214,2007

[4] Lung-Pin Chen, I-Chen Wu, William Chu, Jhen-You Hong, and Meng-Yuan Ho , “Incremental Digital Content Object Delivering in Distributed Systems”, IEICE Transactions on Information Systems , vol. E93-D , no. 6, June, 2010.

[5] X.Tang and S.T.Chanson. , “Minimal cost replication of dynamic web contents under flat update delivery ,” IEEE Transactions on Parallel and Distributed Systems, 15(5):431–441,2004

[6] J. Challenger, A. Iyengar, K. Witting, C. Ferstat, and P. Reed, “A Publishing System for Efficiently Creating Dynamic Web Content,” Proc. IEEE INFOCOM, pp. 844-853, Mar. 2000.

[7] Lung-Pin Chen, Yu-Hsiang Huang, and Ke-Chiang Chih, “Efficient Reconstruction of Dynamic Web Contents in Hierarchical Environment,” The 2010 Conference on Technologies and Applications of Artificial Intelligence, Hsinchu, Taiwan, Nov, 2010