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

階層式 Web 服務佈署策略之簡化與改進方法

On Refining the Strategies of Deploying Web Service Components in Hierarchical Network Servers

研 究 生：陳暐璽

中華民國一百零二年一月

摘要

一個雲端服務應用程式可能包含多個服務元件，這些元件的運作成本包含了計算成本以及資料傳輸成本。客戶通常會將元件副本佈署在對計費方式較有利的伺服器節點中。研究顯示，只傳送所需要的物件內容，而不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。找出佈署策略之最低成本是一個困難的任務，因此很多研究將重點放在求較佳解的方法。本論文探討階層式環境的服務元件的佈署策略的較佳解，首先先求出一組近似解，接著分析服務元件之間的流程相依關係，微調原先之近似解並進一步精簡。研究結果顯示靠著消除多餘的物件，佈署成本能夠降低並且能夠提高網路服務效能。

關鍵字：資料副本、動態網頁、Web service、雲端計算

ABSTRACT

A cloud service applications may contain multiple service components, the operation of these components cost includes the cost of computing and data transfer costs. Customers usually will copies of the components to be deployed in a more advantageous billing server node. The research shows, only to send the object content, cloud services can reduce operating costs and improve overall performance without updating all the service object. Find out deployment strategy of minimum cost is a difficult task, so a lot of the research will focus on seeking better solutions. This thesis explores the deployment strategy of the service components of the hierarchical environment better solution, we first find a set of approximate solution, then analysis of the processes between the service components dependency, fine-tuning the original approximate solution to further streamline. The research results show that by eliminating redundant objects, deployment cost can be reduced to improve the network service performance.

Keywords：Data replication, Dynamic web, Web service, Cloud Computing.

CONTENTS

LIST OF FIGURES

CHAPTER 1　簡介

Web 應用是由 server 端執行複雜的商業邏輯運算，提供應用程式資料給 client 端的程序。透過雲端網路服務，web 服務應用程式已成為具備全面功能的應用程式。為增加應用程式的可用性，系統會透過複製內容元件並散布於多個網路節點[1][2][3]。網路的內容服務佈署一直是一個基本的應用問題，吸引了學術界和業界[4][17][21]的研究投入。本論文中，我們將討論在樹狀的階層式網路中[22]，由根節點擔任 server 主機，將新的內容版本更新到底層主機的問題。

如何在網路頻寬與主機計算能力下做一個平衡的取捨，這是個需要來探討的問題[24][25]。如果當網路頻寬變小而嚴重延遲時，將不適合做大量的資料傳輸；相反的，如果使用的電腦運算速度是老舊較慢的，那麼由網路取得資料反而比自行計算還要有效率的多。文章[3][23]中，作者研究發展有效的演算法，找出最小[19]成本內容副本策略。在網路應用中，經由 client 端重新計算資料的方式來加速系統的執行效率，client 計算基於他的前行元件執行轉換[3][8][15][16]來建立物件，內容物件成本可經由網路直接傳送與 client 的重新計算。

一般檔案副本的更新是透過檔案傳輸來達成，而 Web 服務物件的更新較為複雜，包括在分散式環境中依照相依關係來建構 Web 物件。研究顯示，只傳送所需要的物件內容，而不更新所有的服務物件可以

減少雲端服務[27]的運作成本，並且提高整體效能。先前已有研究探討如何在階層式網路環境中求較佳解的 Web 物件更新方法。本論文探討階層式環境的服務元件的佈署策略的較佳解，首先先求出一組近似解，接著分析服務元件之間的流程相依關係，微調原先之近似解並進一步精簡。研究結果顯示靠著消除多餘的物件，佈署成本能夠降低並且能夠提高網路服務效能。本文中的重點是在階層式網路中，使用每個主機的$direct$直接與$InDirect$間接相依關係來消除部分成本，第三章會加以詳述以及規範條件內容。

藉由將 *ODG* 轉換為傳統流量演算法(Maximum flow/Minimum Cut)[13]，我們的演算法能針對階層式網路求出較小遞送成本服務元件的佈署策略，使分散式內容應用程式有效地運作。實驗方面使用 *OMNet++*軟體來進行網路仿真模擬，*OMNet++*軟體在第四章有更詳細的說明。實驗結果顯示，消除$direct$與$InDirect$元件相依關係的精簡方法比先前的演算法，更能降低(減少)系統的執行時間。

接下來的章節安排如下：第二節定義問題的模型。第三節介紹了主要的演算法。在第四節中，實驗架構說明與結果表明。最後，第五部分討論本篇結論和今後發展的工作方向。

CHAPTER 2　問題定義

2.1　Object Dependency Graph (ODG)

Web內容應用程式以非循環的有向圖*ODG* (Object Dependency Graph)來代表，我們假設$G=(V,E)$是*ODG* G表示式，$V=\{v_0,v_1,\ldots,v_n\}$，$v_i\in V$表示web上的服務物件，鏈結$(v_i,v_j)\in E$表示物件v_i連向v_j的相依關係，0 ≤ i，j ≤ n。

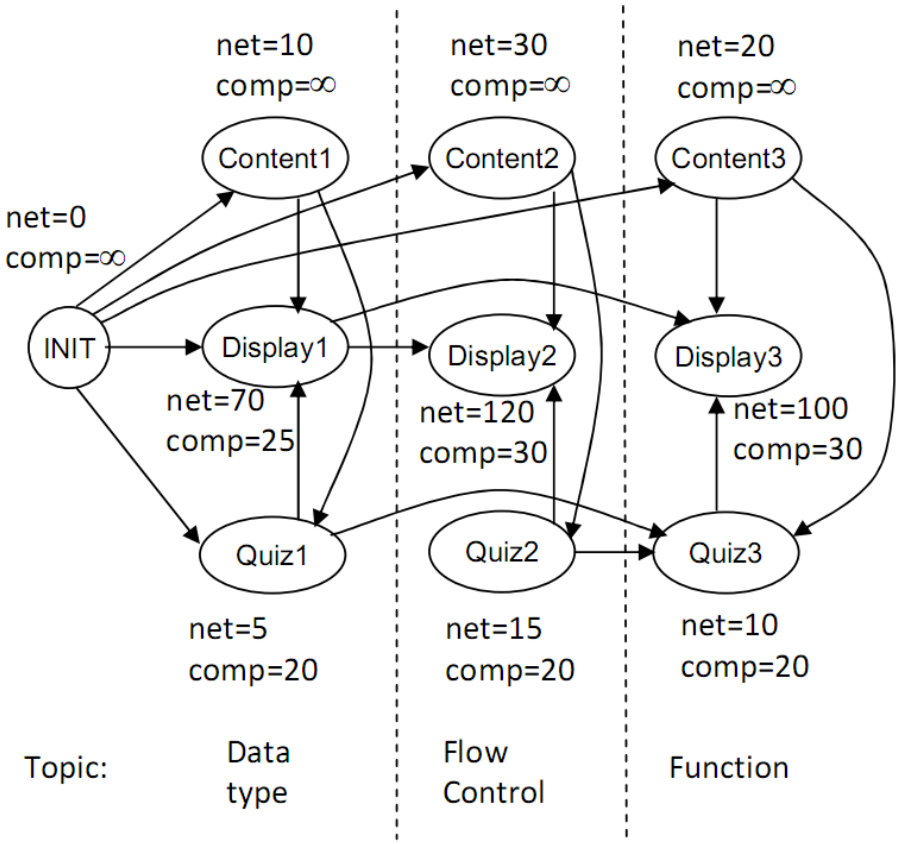
ODG G圖上，一段edge上$(v_i,v_j)\in E$，假設一條途徑從v_i連向v_j，表示成$v_i\rightarrow v_j$。v_i相對於v_j是個前行節點$\mathrm{PRED}(v_i)=\{v_j|(v_j,v_i)\}$，$v_j$相對於$v_i$也就是後繼節點$\mathrm{SUCC}(v_i)=\{v_j|(v_i,v_j)\}$。如果$\mathrm{PRED}(v_i)=\{\emptyset\}$，表示$v_i$是第一個節點並稱為initial節點(*I*)；如果$\mathrm{SUCC}(v_i)=\{\emptyset\}$，$v_i$則是最後的終端節點稱為target節點(*J*)。假設集合$W\subseteq V$，$\mathrm{PRED}(W)=\{v_j|v_j\rightarrow v_i,v_i\in W,v_j\in V\backslash W\}$　且　$\mathrm{SUCC}(W)=\{v_j|v_i\rightarrow v_j,v_i\in W,v_j\in V\backslash W\}$(我們使用簡易的數學表示方法來表達$X\backslash Y$，將$X$-$X\cap Y$的一種集合)。正因為*ODG* G是一個非循環的有向圖，所以$\mathrm{PRED}(W)\cap\mathrm{SUCC}(W)=\{\emptyset\}$。

ODG G概念是由物件所組成，且物件部分是可執行的。假設資料傳輸方向是$v_i\rightarrow v_j$，v_i計算產生的output會成為v_j的input，進而觸發連續的計算行為。物件是否可執行計算的其中一個條件，是必須接收其物件的$\mathrm{PRED}(N_x)$資料，N_x為傳輸節點。物件成本方面，每個節點v_i

有兩個成本：$\text{comp}(v_i, s_x)$代表物件在主機s_x中計算的成本；$\text{net}(v_i, s_y, s_x)$代表資料從主機$s_y$傳送至主機$s_x$的成本。以圖表 1為例，*ODG*圖代表網路上提供的web版程式語言學習教材，電子書包含三大主題：Data type、Flow-control、Function，每個大主題下又包含內容(Content)、網頁的呈現(Display)、與測驗(Quiz)等三類物件。Content是每個主題的知識內容，且該內容無法由其他物件導出，因此Content計算成本設為∞，強迫Content物件向INIT取得資料。每個元件都是一份XML文件，XML文件可以由它的前行者XML文件經過XSL轉換而得XSLT，需要XSLT才可顯示在使用者HTML的瀏覽器上。INIT伺服器擁有以XML格式儲存的電子書內容物件正本，每個物件net成本設定需以物件的大小與頻寬來斟酌以設定成本。

以圖表 1為例，想要得到Quiz3內容可由網路取得net(Quiz3)=10成本。使用者也可利用自行計算的方式得到Quiz3，但必須取得Quiz3所有前行節點的資料，才可以接著進行計算Quiz3內容，其計算成本為 net(Quiz1) + net(Quiz2) + net(Content3) + comp(Quiz3) =5+15+20+20=60。很顯然地，以Quiz3物件為例，要取得Quiz3內容經由網路(10)成本較自行運算成本(60)低，這說明Quiz3物件並不適合自行運算。上述兩種可能的佈署trade-off 問題，就如同大型系統的patch更新。直接傳送大量或完整文件往往比傳送小的patch檔，再由

用戶端自行計算還花費更多網路成本。舉例來說，Flash動畫傳送完整影像會比傳送指令由client自行運算畫面花費更多網路成本。相反的，如果是氣象預報這類需要大量的計算，計算的結果卻只有幾bytes，特別依賴電腦的計算能力就不適合透過網路進行大量資料傳輸。



圖表 1：web 版程式語言電子書 ODG。

2.2 佈署策略

本節定義階層式主機網路佈署策略(deployment strategy)[20]在系統中執行服務與傳送執行結果的策略。某個網路主機s_x，一個物件服務(*ODG*節點)的佈署型態分三種情況：

- *O-node*：節點不佈署服務在s_x。
- *N-node*：由前行主機計算後，接收資料的節點。
- *C-node*：*N-node*接收資料後，由本地端執行計算的節點。

對主機s_x來講，集合(N_x, C_x)表示一種**佈署項目**，其中：

- N_x表示主機s_x的*N-node*集合。
- C_x表示主機s_x的*C-node*集合。
- $V \backslash (N_x \cup C_x) \in$ *O-node*。

在同一主機上，一個物件只能有一種佈署方式，因此，$N_x \cap C_x = \{\emptyset\}$，其中$0 \le x \le M$。階層式網路主機中，假設由$M+1$個主機$(s_0, s_1, \dots, s_M)$組成的樹狀結構。$M+1$個主機就有$M+1$種佈署項目，表示為$\big((N_0, C_0), (N_1, C_1), \cdots, (N_M, C_M)\big)$。將這些項目集合起來就形成應用程式*ODG G*在網路上各主機的佈署策略，定義主機如何在網路上進行合作以獲得目標物件的重建或內容更新。階層式網路中以主機s_0為樹狀網路的根主機，位於葉節點位置的主機稱為終端主機。在階層式網路架構下，每個主機s_x(除了根主機之外)都會有一個父節點主機$\text{parent}(s_x)$，子節點稱為$\text{child}(s_q)$。樹狀結構的網路會形成一個供需鏈，上游伺服器會提供下游伺服器物件所需的計算結果。*ODG G*是有方向性且無循環的物件圖，禁止下游主機往上游主機進行佈署物件或傳輸計算結果。對每個主機s_x來講，副本佈署在s_x上的目標物件

集合稱為T_x。考量物件在網路上的關聯性，並非所有佈署項目都是可行的或合法的，可行的佈署策略定義如下：

定義 2.1（可行的佈署策略）：*ODG* $G=(V,E)$，階層式主機$(s_0,s_1,\ldots,s_M)$，對應副本策略$((N_0,C_0),(N_1,C_1),\cdots,(N_M,C_M))$是“可行的”佈署項目：

(1) (Computable)： s_x主機的*C-node* C_x由s_x自行運算，因此$\mathrm{PRED}(C_x)\subseteq(N_x\cup C_x)$，也就是說當主機$s_x$計算一個*ODG*節點時，其前行節點不可為*O-node*。

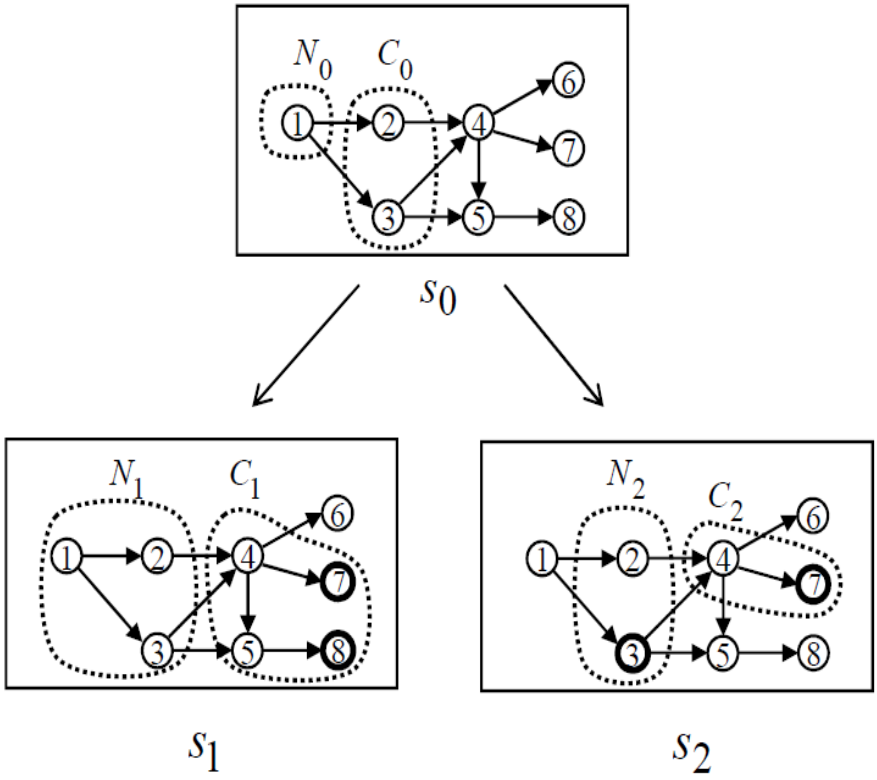
(2) (Supply-Demand)：若s_x是根主機以外的主機，$s_x=\mathrm{parent}(s_q)$，則s_q的N_q由s_x提供：$N_q\subseteq(N_x\cup C_x)$。

(3) (Fulfillment)：終端節點必須佈署在終端主機s_M (底層葉節點終端主機)。$T_x\subseteq(N_x\cup C_x)$。

(4) (Initial) ：根主機s_0，$N_0=I$。起始節點必須佈署在根主機s_0(root主機)。■

Computable 屬性限制了initial節點永不能被計算。圖表 2表示一個合法的階層式主機(s_0,s_1,s_2)佈署策略。圖中的s_2主機對應的佈署項目(N_2,C_2)，$N_2=(2,3)$，$C_2=(4,7)$，其中$\mathrm{PRED}(C_x)=\mathrm{PRED}(\{4,7\})=\{2,3\}\subseteq\{2,3,4,7\}$滿足Computable 條件。在child主機$s_1$與$s_2$所有的*N-node*$(N_q)$都可由parent主機$s_0$來提供，滿足Supply-Demand 條件。

實際上，副本策略定義了一個物件的形式(*O,N,C*)和傳輸(net)運作，從初始節點s_0依照層次結構的序列往底部傳送資料，直到所有目標物件被重建(更新)過，滿足Fulfillment 條件。



圖表 2：可行的階層式主機(s_0, s_1, s_2)佈署，粗體圓圈是目標物件 T。

針對階層式網路，利用佈署策略演算法使分散式內容應用程式有效地運作。圖表 2顯示一個二層式的網路架構，根據[5]，外層由主機形成樹狀結構，而樹狀架構的節點為複合式節點，包含了物件與物件形成的*ODG G*架構。利用主機*ODG*內的節點分別與各子主機的相應節點來將內外層連結成一個網路服務佈署。以圖表 2為例，每個複合式節點都有相同的*ODG*物件，而主機(S_0, S_1, S_2)之間形成樹狀的階層

式網路。完成建構圖形後再將其視為一個單一的*ODG*圖形求解，最後將單一的*ODG*合併求得完整的網路流量。

2.3 最小成本之佈署策略問題

每個主機s_x在*ODG*物件佈署策略上有兩個成本。對於每一個節點v_i分別是：

(1) $\text{comp}(v_i, s_x)$：在主機s_x，物件v_i計算成本。

(2) $\text{net}(v_i, s_x)$：上游主機$\text{parent}(s_x)$傳送資料到主機s_x v_i接收的網路成本。

$\text{comp}(v_i, s_x)$、$\text{net}(v_i, s_x)$被稱為成本函數，*N-node*提供$\text{net}(v_i, s_x)$成本，*C-node*則提供$\text{comp}(v_i, s_x)$成本，計算總成本時*O-node*沒有被佈署亦不會提供任何成本。假設主機s_x佈署策略項目(N_x, C_x)，$1 \le x \le M$，則成本總和為：

$$\text{cost}((N_x, C_x)) = \sum_{v_i \in N_x} \text{net}(v_i, s_x) + \sum_{v_i \in C_x} \text{comp}(v_i, s_x) \qquad (1)$$

以本篇論文來講，**最小成本更新傳輸問題**[26]就是在階層式的環境下$(s_0, s_1, \dots, s_M)$，*ODG* *G*找出一組可行的佈署策略$((N_0, C_0), (N_1, C_1), \dots, (N_M, C_M))$符合**定義 2.1**，且總成本 $\sum_{x=0}^{M} \text{cost}\big((N_x, C_x)\big)$是最小值，

也就是在單一時間下網路可通行的最大流量。

本論文討論網路流量演算法運用在階層式伺服器網路的部署策略的解法，由於最佳解的轉換極為複雜，本論文討論較佳解，並透過實驗來驗證我們提出的方法之效能。

2.4 ODG 成本

本篇論文討論網路服務應用程式，佈署在階層式樹狀的網路結構中，內容遞送網路 CDN[28]服務模式是透過事先的設定與管理，有效將服務元件佈署到主機上。以圖表 2 為例，s_0根節點為最主要的 server 也是存放所有資料的地方，往下游s_1, s_2子節點 server 佈署所需要更新的部分內容元件，不需要大規模的全面更新子節點 server 內容元件，以減少部分成本。更新子節點 server 的目的是提供 client 就近來存取更新的資料，減少每個 client 都與根主機 server 來交換資料，一方面也分擔 server 的負載量。而子節點 server 資料就由上游 server 來做更新或傳輸資料的佈署。

前面章節所敘述的 *ODG* 成本，這邊可解釋為雲端的租賃費用或計算時間、執行應用程式的 respond time、運算產生的排碳量或消耗的能量、或者是可量化的損耗值等等。前述的這些量化值當然是越低越好，節省較多的成本開銷，轉換成問題目標也就是我們所要求的最小成本(費用)佈署策略問題。

CHAPTER 3　佈署策略問題與演算法

3.1　圖形切割與佈署策略之轉換

近似值演算法解決副本策略問題是藉由流量圖轉換成圖形切割[6]的問題。定義如下，$H = (U, F, s, t, cap)$是由五個元素組成的整合圖，其中(U, F)為有向圖，每個edge$(u, v) \in F$且edge是非負值的容量cap(u, v)。整合圖H中，s節點與t節點分別為資料來源點與接收點，s節點輸出流量必須與t節點輸入流量一致。

最小成本佈署策略制定是透過流量切割的概念[14]，我們定義H-cut為整合圖H的切割。H-cut將U集合分割為(X,Y)，切割的左邊為X區域，右邊為Y區域，$\{s\} \subseteq X$，$\{t\} \subseteq Y$。我們將每個ODG節點v_i和轉成流量圖$v_{i,b}$和$v_{i,e}$兩點（表示begin和end節點）。另外，集合F的edge分為一般容量與無限容量：

- $\mathrm{cap}(\mathrm{s}, \mathrm{v_{i,b}}) = \mathrm{comp}(\mathrm{v_i})$
- $\mathrm{cap}(\mathrm{v_{i,b}}, \mathrm{v_{i,e}}) = \mathrm{net}(\mathrm{v_i})$
- $\mathrm{cap}(\mathrm{v_{i,e}}, \mathrm{v_{i,b}}) = \infty$

為方便起見，無限容量的edge又稱為$\infty -$ edge。

以下特性 P1 證明H-cut不可順向切過任何$\infty -$ edge否則成本會變成∞：

P1 對H-cut (X,Y)與任何∞-edge(v,w)，條件$v \in X \wedge w \in Y$不成立。反過來說，$(v \in X \wedge w \in X) \vee (v \in Y \wedge w \in Y) \vee (v \in Y \wedge w \in X)$成立。

流量圖中加入許多成本為∞的edge，強迫流量圖的切割與*ODG*佈署策略之間存在著一對一的關係。令$(X,Y) = \{v|(v_{i,b}, v_{i,e}) \in E, v_{i,b} \in X, v_{i,e} \in Y\}$為$N_x$的集合，整合圖$H$的節點集合$U$中，每一個*ODG*節點$v_i$均有兩種成本net($v_i$)與comp($v_i$)，最小切割就是所有H-cut中成本最小的那組佈署策略。上述H-cut與傳統切割(graph cut)定義有部分不同，根據[8]，最小H-cut可用現有的最大流量演算法解出。從一條水管來看，瞬間最大流量就是取決於水管最夾窄能通過的地方，也就是這條水管單一時間下流出的最大流量，H-cut就是在圖上劃出水管最窄流量的地方。H-cut的成本公式定義：

$$\mathrm{cost(X,Y)} = \sum_{\mathrm{v \in H(X,Y)}} \mathrm{net(v_{i,b}, v_{i,e})} + \sum_{\mathrm{v \in Y}} \mathrm{comp\left(s, v_{i,b}\right)} \quad (2)$$

最小切割問題有效的解決時間複雜度$O(nm \log(n^2/m))$[6]。

將 *ODG* 與網路進行乘積運算(Cartesian product)產生整合圖 H，如圖表 3。整合圖 H 的建構演算法定義如下：

定義 2.2： 假設網路 s 有 $M+1$ 個樹狀結構主機$s_0, s_1, \ldots, s_M$，$ODG\ G = (V, E)$。整合圖$H = (U, F)$建構如下：

(1) 對所有主機$s_k(0 \le k \le M)$，將所有點和線加上標籤(k)，建立子圖$G^{(k)} = (V^{(k)}, E^{(k)})$，$G^{(k)}$的建立方式如下：

- For each object v_i in the ODG
  - Add two vertices $\mathrm{v_{i,e}}^{(k)}, \mathrm{v_{i,b}}^{(k)}$ to graph H.
  - Add edge $(\mathrm{v_{i,e}}^{(k)}, \mathrm{v_{i,b}}^{(k)})$ to graph H.
- For each edge $(\mathrm{v_i}, \mathrm{v_j})$ in the ODG
  - Add edge $(\mathrm{v_{i,e}}^{(k)}, \mathrm{v_{j,b}}^{(k)})$ to graph H.

(2) 建立整合圖 H 的點集合 $U = (V^{(0)} \cup V^{(1)} \ldots \cup V^{(M)} \cup \{T\})$，整合圖 H 結束點為 T。

(3) 建立以下三類$\infty -$ edges：

- $E_{DOWN} = \{(v_{i,b}{}^{(p)}, v_{i,b}{}^{(k)}) | v_i \in V, s_p$是$s_k$的 parent 主機$\}$。
- $E_{SOURCE} = \left\{(s, v_{i,b}{}^{(k)}) \middle| v_i \in I \text{ 是 initial 節點}, 0 \le k \le M\right\}$。
- $E_{TARGET} = \{(v_{i,e}{}^{(k)}, T) | v_i \in V, v_i$是$G$的結束點$, 0 \le k \le M\}$。

(4) edges 成本：$E_{COMP} = \{(s, v_i{}^{(k)}) | v_i \in V \setminus I, 0 \le k \le M\}$。

- 對edges$(s, v_{i,b}{}^{(k)})$，cap$\left(\left(s, v_{i,b}{}^{(k)}\right)\right) = \mathrm{comp}(v_i)$。
- 對edges$(v_{i,b}{}^{(k)}, v_{i,e}{}^{(k)})$，cap$\left(\left(v_{i,b}^{(k)}, v_{i,e}^{(k)}\right)\right) = \mathrm{net}(v_i)$。

(5) 建立整合圖 H 的線集合：$F = (E^{(0)} \cup E^{(1)} \ldots \cup E^{(M)}) \cup E_{DOWN} \cup E_{SOURCE} \cup E_{TARGET}$。■

由(X,Y)導出$G^{(x)}$的一組切割項目$(X \cap V^{(x)}, Y \cap V^{(x)})$，副本策略演算法回傳 $\left(\left(X \cap V^{(0)}, Y \cap V^{(0)}\right), \left(X \cap V^{(1)}, Y \cap V^{(1)}\right) \ldots, \left(X \cap V^{(\mathrm{M})}, Y \cap V^{(\mathrm{M})}\right)\right)$。假設$(X,Y)$在整合圖 H 是最小切割，我們可以倒出以下部署策略 R：

$$\boldsymbol{R} = (\left(X \cap V^{(0)}, Y \cap V^{(0)}\right),$$
$$\left(X \cap V^{(1)}, Y \cap V^{(1)}\right),$$
$$\ldots,$$
$$\left(X \cap V^{(M)}, Y \cap V^{(M)}\right))$$

(X,Y)切割跟整合圖 H 之間的相對應關係與副本策略如下：

- 如果$\mathrm{edge}(v_{i,b}^{(x)}, v_{i,e}^{(x)})$在切割線右邊，物件$v_i$就是一個 *C-node*。$F_{COMP}$ 中$\mathrm{edge}(s, v_{i,b}^{(x)})$會被 H-cut 切割到，且成本$\mathrm{cap}\left(\left(s, {v_{i,b}}^{(k)}\right)\right) = \mathrm{comp}(v_i)$。
- 如果$\mathrm{edge}(v_{i,b}^{(x)}, v_{i,e}^{(x)})$橫跨 H-cut 切割線，物件$v_i$就是一個 *N-node*。切割線在$\mathrm{edge}(v_{i,b}^{(x)}, v_{i,e}^{(x)})$，其成本為$\mathrm{cap}\left(\left(v_{i,b}^{(k)}, v_{i,e}^{(k)}\right)\right) = \mathrm{net}(v_i)$。
- 如果$\mathrm{edge}(v_{i,b}^{(x)}, v_{i,e}^{(x)})$ 在 H-cut 切割線左邊，物件v_{i}就是一個 *O-node*。沒有 edge 與$v_i^{(x)}$有相關聯，也就是無佈署服務成本為 0。

圖表 3(a)表示一個 ODG 圖，圖表 3(b)階層式樹狀網路 servers。*ODG* 與 network 的乘積可得到整合圖 *H*(圖表 4)。整合圖$H = (U,F)$中包含 H-cut 與相對應的副本策略，黑色粗體實線為 H-cut，所有的資

料都是由 s 節點出發，經過中間主機傳輸計算更新後，最終將全數流回 t 節點。令$(X^{(x)}, Y^{(x)})$為 H-cut(X, Y)在子圖$G^{(x)}$上的子切割。根據[7]，圖表 4 的佈署策略是合法的。H-cut (X, Y) 對應到以下佈署策略：

$$N_0 = \{v | {v_{i,b}}^{(0)} \in X^{(0)}\}, C_0 = \{v | {v_{i,e}}^{(0)} \in Y^{(0)}\} \qquad (3)$$

$$N_1 = \{v | {v_{i,b}}^{(1)} \in X^{(1)}\}, C_1 = \{v | {v_{i,e}}^{(1)} \in Y^{(1)}\}$$

$$\cdots$$

$$N_M = \{v | {v_{i,b}}^{(M)} \in X^{(M)}\}, C_M = \{v | {v_{i,e}}^{(M)} \in Y^{(M)}\}$$

要證明佈署策略是合法的，需驗證**定義 2.1** 四項條件都是滿足的情況[7]。最後讀出合法的較佳解佈署策略後，透過提出的方法將多餘的成本消除，可進一步減少佈署策略的成本。較佳解演算法步驟如下：

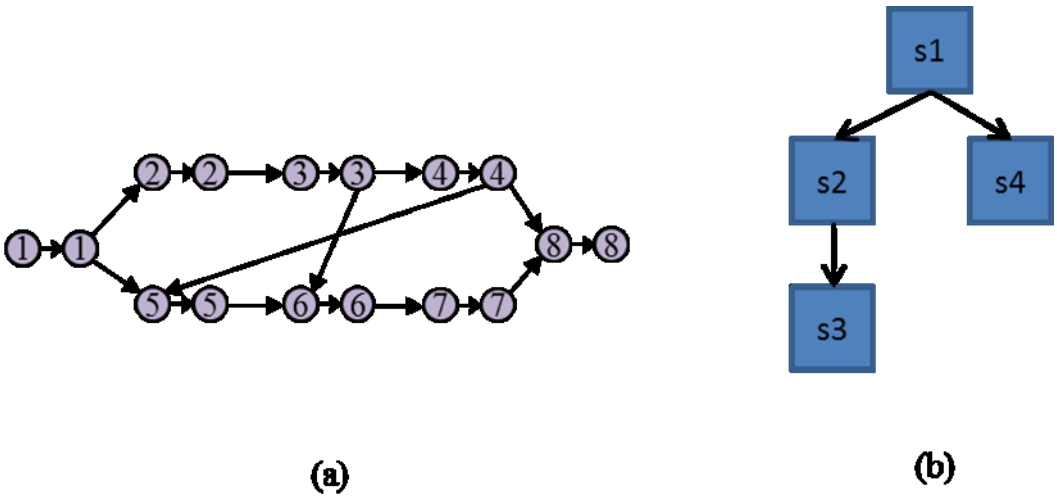
1. 將佈署策略問題轉換成整合圖 H。
2. 找出整合圖 H 上最小切割$C_{min} \in$ H-cut。
3. C_{min}對應一組合法的佈署策略(**定義 2.2**)。
4. 將利用**定義 3.1**、**3.2** 方法消去多餘的物件成本。

步驟 1 與步驟 2 轉換方法是一個較早的演算法，線性網路結構原理的延伸[8]，步驟 4 是這篇論文所提出的方法。較早的演算法可以解出線性網路結構的最佳解。然而在本文中延伸應用，它只能找到階層式網路結構的較佳解方案。精簡成本的方法將在接下來的小節討論。

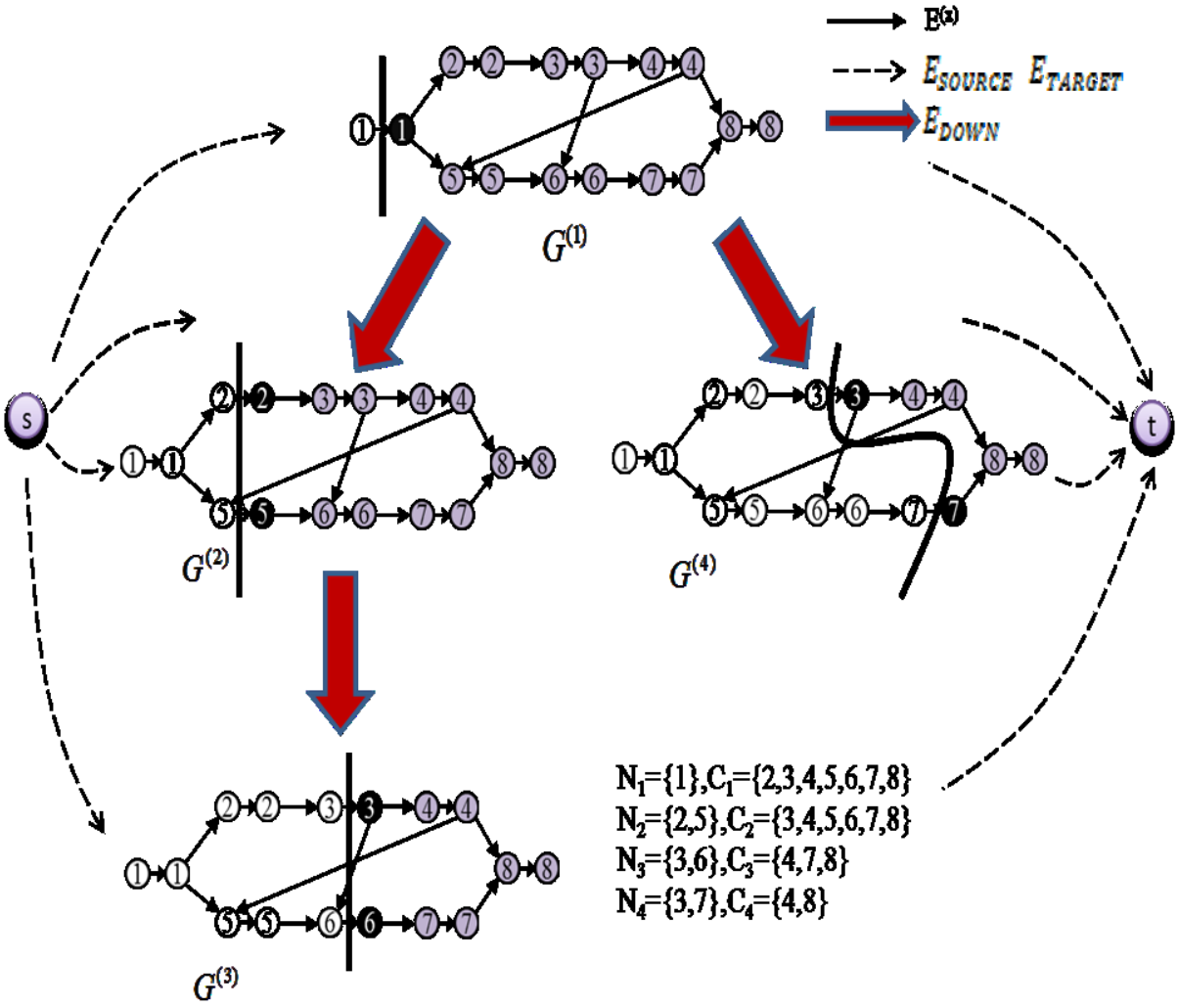
本篇論文探討在階層式的架構下應用，而另外線性架構的網路已經有先前的學者研究[9]。首先設定主機之間的網路相依關係與主

機內的*ODG*模型，產生實驗需要的成本，開啟佈署圖形轉換器(T)[9]，將預先產生好的數據在圖形轉換器(T)中產生佈署流量整合圖 H，依照既定的順序將成本一對一的附著到每個物件。確保成本的大小符合實驗需求與滿足圖形的屬性(物件各自有一組成本)，這在接下來的實驗中是很重要的，假如其中一個物件附屬錯誤的成本，會導致接下來的計算實驗結果都是錯誤的。首先讀入 *ODG* $G = (V, E)$，$V = (v_1, v_2 \ldots, v_n)$，成本函數 net 與 comp 還有目標物件集合 $(T_0, T_1 \ldots, T_M)$。接下來將成本與 *T* 依照順序佈署到主機的 *ODG* 物件上，接下來執行最大流量演算法找出最小切割線佈署 *O,N,C* 的結果如圖表 4。

從生產者及消費者的角度分析，若s_k是s_q的 parent 主機，在主機s_k內只需佈署物件$N_k \cup C_k$**(定義 2.1-(2))**。若某一物件v_i不在s_q子主機的N_q需求範圍內，那麼s_k就不用佈署物件v_i，相同物件v_i在s_q子主機會與s_k主機重疊計算造成多餘的成本。



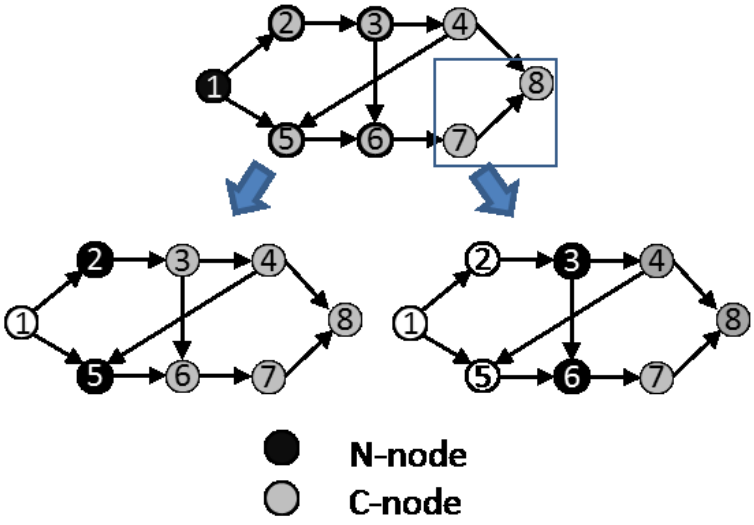
圖表 3：(a)*ODG G*，(b)*network S*。



圖表 4：整合圖 *H* 的切割線。

3.2 Direct 副本策略精簡方法

在主機s_x中，副本策略(N_x, C_x)將會被重新的佈署 *N-node* 與 *C-node*。事實上，是在滿足策略**定義 2.1** 且是個可行的佈署策略下扣除多餘的成本，可進一步的精簡成本。



圖表 5　parent 主機s_x，child 主機s_y、s_z，$N_x = \{1\}$，$N_y = \{2,5\}$，$N_z = \{3,6\}$，$Dir(N_x, C_x) = \{1,2,3,4,5,6\}$。

我們考慮一個情況，假設兩層主機的階層式網路，網路架構與圖表 2 相同，但 *ODG* 架構不相同。parent 主機s_x有兩個 child 主機s_y、s_z，而它們對應的副本策略分別是主機s_x (N_x, C_x)、主機s_y (N_y, C_y)與主機s_z (N_z, C_z)。符合**定義 2.1-(2)**條件，parent 主機s_x佈署範圍必須含蓋 *N-node* N_y、N_z，如沒含蓋到N_y、N_z則無法提供主機s_y、s_z計算所需的資料，也就違反了**定義 2.1-(2)**的條件。如圖表 5，討論$direct$相依物件的路徑，$N_x = \{1\}$，$N_y = \{2,5\}$，$N_z = \{3,6\}$，$Dir(N_x, C_x) =$

{1,2,3,4,5,6}，7,8 元件成本可在主機s_x扣除。原因是同樣的元件在下游主機中還會再被計算過一次，造成多餘的計算次數，以消費者角度來看，本地端也只需計算到下游節點所需位置即可，再多的計算元件都是多餘的。

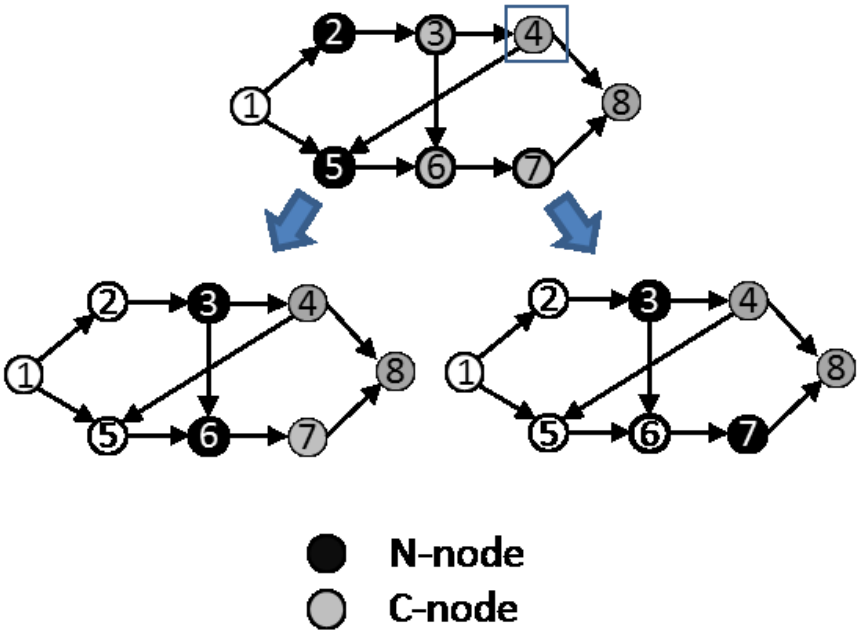
定義 3.1： 根據公式(3)佈署對應項目：

1. $Dir(N_x, C_x) = PRED(N_y) \cup N_y \cup PRED(N_z) \cup N_z$。
2. Set of nodes in $V \backslash Dir(N_x, C_x)$ can be removed from (N_x, C_x)。

■

3.3 InDirect 副本策略精簡方法

加強 3.2 小節的$Dir(N_x, C_x)$的延伸，Dir已扣除部分多餘成本，這章節將利用更進一步方法$InDir(N_x, C_x)$來精簡執行時間。



圖表 6　parent 主機s_x，child 主機s_y、s_z，$N_x = \{2,5\}$，$N_y = \{3,6\}$，$N_z = \{3,7\}$，$Dir(N_x, C_x) = \{2,3,4,5,6,7\}$，$InDir = \{4\}$。

如圖表 6，InDir(N_x, C_x)代表一個節點的集合，$InDir(N_x, C_x) \subseteq Dir(N_x, C_x)$，集合$InDir$的條件是$C_x$節點往$N_y \cup N_z$的路徑上，需透過$N_x$才能連接的$C_x$集合；換句話說，並沒有路徑直連到$N_y \cup N_z$，需間接的經過$N_x$來連向$N_y \cup N_z$，因為$N_x$的前行節點 *C-node* 之計算結果並不影響$N_x$結果，所以可扣除$InDir$集合中符合條件的成本。保留$N_x$後繼節點 *C-node*$\backslash InDir$ 佈署N_y、N_z。圖表 6 中，$InDir$節點包含在$Dir(N_x, C_x) = \{2,3,4,5,6,7\}$其中$C_x = \{3,4,6,7\}$。須考慮到有無其他 *C-node* 計算需要的前行節點，例如計算N_y節點 6 必需要其全部的前行節點 3 與 5，因此節點 3 並不算是多餘的節點，而N_x節點 5 並不需要前行節點 4 的計算結果，因此$InDir = \{4\}$。節點 4 成本扣除可再進一步精簡成本。

定義 3.2：根據公式(3)佈署對應項目

1. InDir$(N_x, C_x) = \{v_i | v_i \in Dir(N_x, C_x)\ and\ \forall(v_i, v_j)E,\ v_j \in PRED(N_y) \cup PRED(N_z) \Longrightarrow v_i \in PRED(N_x) \setminus N_y \cup N_z\}$。
2. Nodes in set InDir(N_x, C_x) can also removed from (N_x, C_x)。

■

CHAPTER 4　實驗結果

實驗參數設定包括 *ODG* 形狀(line or tree)、節點數量與成本，網路建構圖(line or tree)與隨機產生的預設實驗成本(net and comp)。首先設定 *ODG* 參數與網路拓樸後，讀入預設成本到每個 ODG 節點，經過乘積後會產生整合圖 H，接著執行 maxflow[6][18]演算法找出最大流量與最小切割佈署策略(N_x, C_x)。將這組(N_x, C_x)來進行 omnet 模擬執行時間，首先主機會配置與連結到樹狀結構裡，所有的 *N-node* N_x *C-node* C_x會被複製到主機s_x。多重主機佈署在 *OMNet++*後開始執行程式，主機會由 node 0 開始廣播，每個主機s_x中的 *C-node* 會一直等待，直到從 parent 主機接收全部的 *N-node*(N_x)。在那之後主機s_x才會開始計算，完成節點計算後s_x再廣播給 child 主機的 *N-node* 來觸發連續的傳輸與計算行為，直到目標節點完成計算取得最後結果。整個過程結束時，除了 *O-node* 以外所有的物件會被重新建構，直到終端主機完成計算才算執行結束。實驗系統架構圖表示在圖表 7，這是一個管理控制組件的實驗工作流程，模擬器可以在接近真實的網路環境下進行各種狀況的模擬，不必受限於真實機器數量的問題與人為操作的問題。數據產生器隨機產生所有的實驗數據，依照問題需求來設定網路拓撲結構、*ODG* 圖形和成本，將部署結果輸入到模型中進行多回合的模擬。根據每次不同

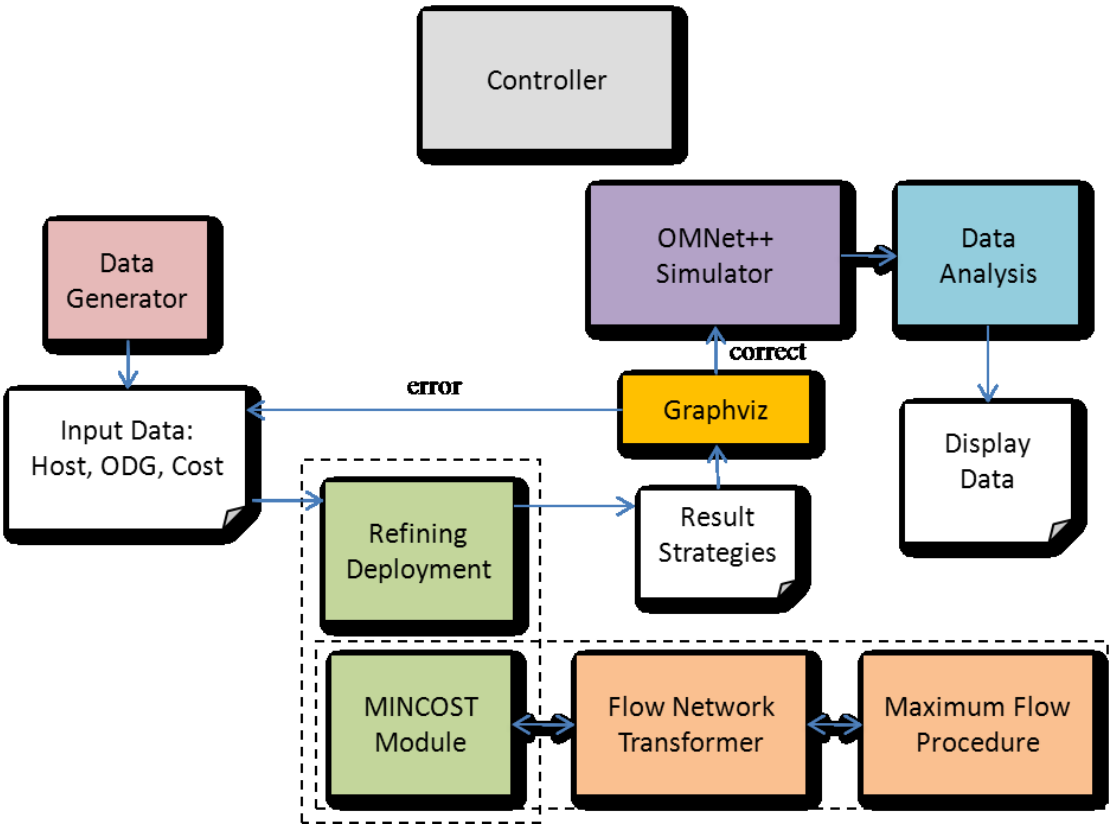
的成本與佈署策略取平均來分析相關數據。

4.1 OMNet++

此小節介紹模擬環境的軟體 *OMNet++*，一套 discrete event network 的模擬平台軟體，使用 C++物件導向程式的技術配合網路描述模組(*NED*)，Message 的種類與行為皆可以程式化定義，非常有彈性而且可重複使用它可利用多種機制來進行大量網路節點的模擬[10]。模擬方式是實作真實的 web service，開發環境是 eclipse，模擬時有圖形化介面可觀察網路串流情況，軟體提供可在任何時間停止或繼續模擬網路傳輸[11]的功能，由於 *OMNet++*具有外接真實網路的 gateway，我們可以將真實應用程式的網路訊息導入 *OMNet++*模擬網路，這樣既能模擬大量主機運作又能實際進行 web service 的運作。在 *OMNet++*中，模擬程式可以結合許多的模組，其中描述網路模組的複檔名為.ned，組態配置檔 omnetpp.ini 整合所有系統參數。系統模型也包含了子模型，而子模型又可以再包含子模型，即允許使用者在模擬的環境中繪製實際系統的邏輯運算結構。方便使用者開發測試模擬程式，完整的執行、記錄、分析、除錯等工具且能精確的分析模擬工作結果。

最後完成所有模擬結果數據組件將會分析顯示實驗結果的總合並記錄。所有的程序實驗均使用 Intel 雙核處理器與 4G 記憶體與

Windows 7。目前 *OMNet++*支援 Windows 以及 Linux 作業系統，學術使用為開放原始碼授權而商業化用途則需取得 Simulcraft 公司授權。



圖表 7：實驗系統架構。

圖表 7 元件說明：

Data Generator：使用數據產生器亂數取得實驗成本。

Input Data：設定實驗目標的網路與 ODG 拓樸，並且讀入實驗成本。

Refining Deployment and MINCOST Module：利用本篇提出的演算法與最小成本模型來進行實驗模擬。

Result Strategies and Graphviz：執行程式後的最小成本佈署策略，放

到 Graphviz 軟體[12]確認是否為實驗所符合的結果。

OMNet++ Simulator：使用此套模擬軟體[11]模擬執行整個傳輸過程的所需時間。

Data Desplay：由 OMNet++模擬結果比較上述的演算法之間的差異性，統計並做結論。

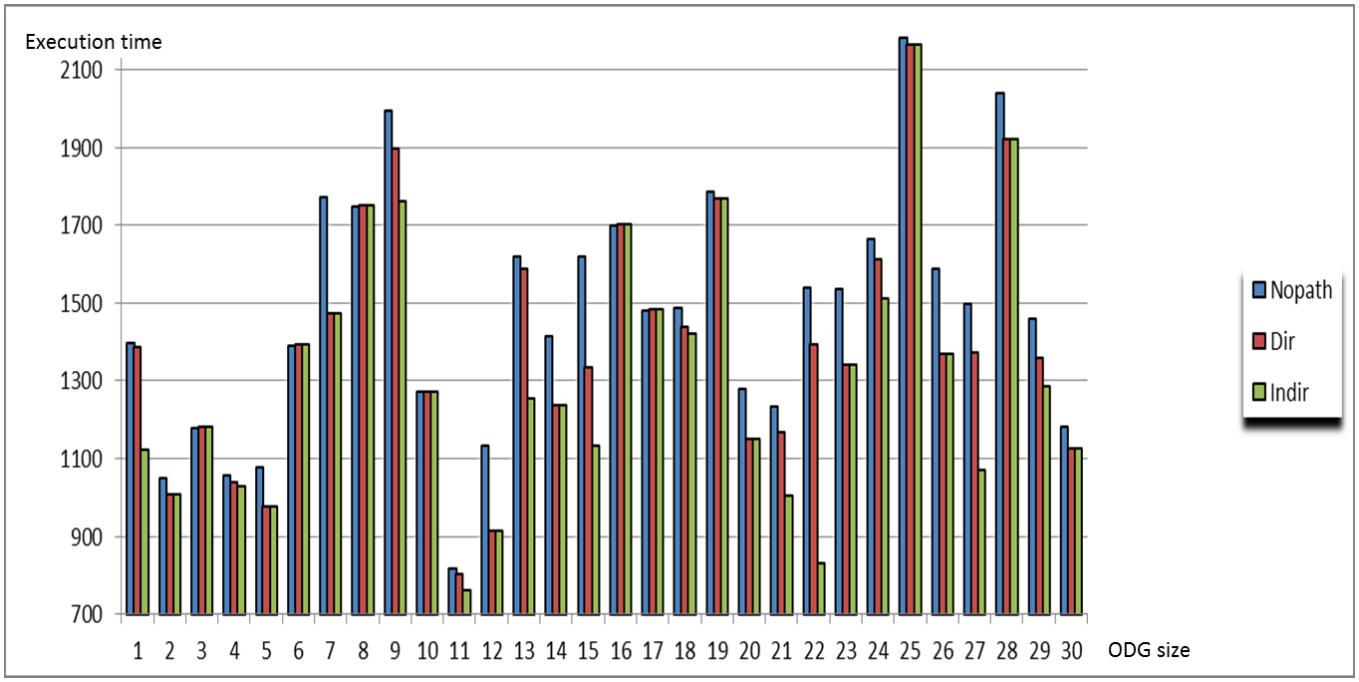
4.2 實驗細節

在進行實驗之前，必須由數據產生器亂數產生數組 net 與 comp 的 cost.text 檔，內容含每個 *ODG* 物件的成本、*ODG* 節點數量，需與網路拓樸一致，否則會造成成本配對錯誤導致執行結果誤差。數值設定最低 0 至最高至∞(以 100 萬代表為上限)，現實中並沒有負值的計算能力與網路頻寬。程式中 odg1 與 odg2 分別表示網路拓撲與 *ODG* 圖，依照實驗需求調整參數用於邏輯設計圖。odg1 代表階層式主機$(s_0, s_1, \ldots, s_p)$之間的相依關係圖，而 odg2 代表每個主機裡 *ODG* 中物件與物件的執行先後順序。整合圖 H　odg3 是由 odg1 與 odg2 進行乘積運算(Cartesian product)產生。程式執行後會產生一個文字檔紀錄物件的佈署圖(*O,N,C*)、成本與流量，以 0 代表 *O-node*，1 代表 *N-node*，2 代表 *C-node*。標示為 0 的節點成本將不列入最後的總成本計算。為了方便驗證網路與流量結果是否正確，我們將 output 文字檔複製到 Graphviz (Graph Visualization Software)[12]軟體執行文字轉圖的動作。

Graphviz 由一種被稱為 DOT 語言的圖形描述語言，我們利用這套軟體將程式可畫出複雜的網路關係圖，以便我們驗證網路與 *ODG* 物件相依關係的正確性。待確認網路關係與流量正確後再將最大流量演算法產生的佈署策略，以文字檔形式複製到 *OMNet*++模擬器，執行網路模擬成本的數據分析，模擬數據結果記錄在 result.txt。經過多組的成本與不同演算法各取平均來比較優缺點。

4.3 實驗討論

這章節我們來討論實驗結果，我們提出一個在階層式網路下，可增強系統效能的方法。我們實驗目標是去驗證提出的方法，比未精簡的算法還要增進效能。這項實驗是在 *OMNet*++模擬環境下執行所記錄的結果。



圖表 8：模擬效能的三種方法：*Nopath*、*Dir* 與 *InDir*。

圖表 是實驗結果,Y 軸代表程式執行時間,而 X 軸代表 *ODGs* 大小。Y 軸上的每個數據都是由 200 個執行時間取平均而得。由 *Nopath* 透過$Direct$減少執行時間，可再一次經過$InDirect$減低執行成本，也由實驗結果顯示$InDirect$路徑的精簡方法比另外兩種方法還要好，以 *Nopath* 方法為最差，$Direct$次之。

CHAPTER 5　結論與未來發展

現今的科學化社會，生活對於網路的依賴是越來越重，如何的來減低執行時間，respond time 也顯得更為重要。在這篇論文提出了一種針對階層式網路的內容服務的精簡成本演算法，透過消除$Direct$與$InDirect$的相依關係，可進一步刪除多餘佈署的元件中，進而降低成本。

從實驗結果來看，我們新提出的演算法比先前的演算法更能降低且改善成本，然而改善的幅度取決於 *ODG* 的拓樸形狀。從**定義 3.2** 可發現，$InDirect$只發生在往前的元件連接線上，例如圖表 6 中s_x節點 4，越多往前連接的元件，則滿足$InDirect$條件就會越多。

在實驗過程中發現，如果符合$InDirect$條件的元件數量不多或者條件達成的元件成本偏低，那麼實質上能有效減少的執行時間相對也較少，在圖表 8 中有些組的$InDirect$減少成本的情況並不明顯，如第 4 組實驗。相反的，符合條件的元件若是佔很大的計算成本比例，則效果將很明顯，如第 13 組實驗。最差的情況則如第 10 組所示，$Direct$與$InDirect$完全無符合條件的原件，以致執行時間沒有下修的情況出現。目前本篇論文只求到較佳解的佈署策略，未來研究方向應是朝向求最佳解，但是這是一個有挑戰性的題目。

參考文獻

1. B. Krishnamurthy F. Douglis, A. Feldmann and J. Mogul. Rate of change and other metrics: A live study of the World Wide Web. In Proceedings of the USENIX Symposiumon Internet Technologies and Systems, December 1997.

2. A. Labrinidis and N. Roussopoulos. Webview materialization. In Proceedings of the 2000 ACM SIGMOD international conference on Management of data table of contents, May 2000.

3. X. Tang and S.T. Chanson. Minimal cost replication of dynamic web contents under at update delivery. IEEE Trans. Parallel and Distributed Systems, 15(5): 431-441, May 2004.

4. J. Ravi, Z. Yu, W. Shi. A survey on dynamic web content generation and delivery techniques. Journal of Network and Computer Applications, 32(5):943-960, 2009.

5. L.P. Chen, “Optimal Update Delivery Strategy for Hierarchical Content service Environment”, Private communications, Dec 2009.

6. A. Goldberg and R. Tarjan. A new approach to the maximum flow problem. Journal of the ACM, 35(4):921–940, October 1988.

7. L.P. Chen, C.Y. Lai, “A Simplified Deployment Strategy Management Service for Workflow Components”, National Digital Library of Theses and Dissertations in Taiwan, June 2012.

8. L.P. Chen, I.C. Wu, William Chu, J.Y. Hong, and M.Y. Ho , “Incremental Digital Content Object Delivering in Distributed

Systems", IEICE Transactions on Information Systems , vol. E93-D_, no. 6, June, 2010.

9. L.P. Chen, Y.H. Huang, and K.C. Chih, "Efficient Reconstruction of Dynamic Web Contents in Hierarchical Environment." The 2010 Conference on Technologies and Applications of Artificial Intelligence, Hsinchu, Taiwan, Nov, 2010
10. A. Varga. Using the omnet++ discrete event simulation system in education. IEEE Transactions on Education, 42(4), 1999.
11. Omnet++ setup http://www.wretch.cc/blog/wulu2005/342870 July 2012.
12. Wikipedia. Graphviz.

    http://zh.wikipedia.org/wiki/Graphviz , June, 2012.
13. R.K. Ahuja T.L. Magnanti and J.B. Orlin. Network Flows:Theory, Algorithms, and Applications. Prentice-Hall, 1993.
14. 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.
15. K. Li, H. Shen, Francis 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
16. 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.

17. L.B. Chan and I.C. Wu. Detection of summative global predicates. IEICE Transactions Information and Systems, E86-D: 976-980, 2003.

18. L.R. Ford and D.R. Fulkerson. Maximal flow through a network. Can. J. Math., 8:399-404, 1956.

19. K. Watanabe H. Tamura K. Nakano and M. Sengoku. The p-collection problem in a flow network with lower bounds. IEICE transactions on fundamentals of electronics, communications and computer sciences, E80-A: 651-657, 1997.

20. A. Ehuchi S. Fujishige and T. Takabatake. A polynomial-time algorithm for the generalized independent-flow problem. JOper Res Soc Jpn, 47:1-17, 2004.

21. C.E. Leiserson T.H. Cormen and R.L Rivest, “Introduction to Algorithms,” The MIT press, 1989.

22. L.P. Chen and P.M. Tsai. Enhanced Service Deployment for Hierarchical Content Delivery Networks. National Digital Library of Theses and Dissertations in Taiwan, 2010.

23. A. Labrinidis, Q. Luo J. Xu and W. Xue. Caching and materialization in web databases. Foundations and Trends in Databases, Dec 2009.

24. R.M. Rahman, R. Alhajj, and K. Barker. Replica selection strategies in data grid. Journal of Parallel and Distributed Computing, Dec 2008.

25. J.J Wu, Y.F.Lin, and P.F.Liu. Optimal replica placement in hierarchical data grids with locality assurance. Journal of Parallel and

Distributed Computing, Dec 2008.

26. J. Wang, J. Wang, B. Chen, and N. Gu. Minimum cost service composition in service overlay networks. World Wide Web, Aug 2010.

27. Wikipedia. Cloud computing.

    http://en.wikipedia.org/wiki/Cloud_computing , Dec 2012.

28. Wikipedia. Content delivery network (CDN).

    http://en.wikipedia.org/wiki/Content_delivery_network , Dec 2012.