國立臺灣大學管理學院資訊管理學系

碩士論文

Department of Information Management
College of Management
National Taiwan University
Master Thesis

在考慮地域鄰近性且群組化的點對點範圍查詢系統下的搜尋與快取機制

Search and Cache Mechanism in A Proximity-Aware and Group-Based Peer-to-Peer System for Range Queries

黃曉梅

Hsiao-Mei Huang

指導教授：莊裕澤 博士

Advisor: Yuh-Jzer Joung, Ph.D.

中華民國 97 年 7 月

July, 2008

謝詞

論文能順利完成最要感謝的人莫過於辛勤指導我的莊裕澤教授，在研究所兩年的生涯裡，莊教授不厭其煩地指導對學問懵懂無知的我、啟發我論文相關的靈感，並且學到正確且嚴謹的求學態度，在做人做事方面也從莊教授身上學到許多往後受益無窮的觀念。接著要感謝黃榮達學長指導我實驗方面的程式與模擬器的相關問題，由於黃學長打下良好的基礎，使我接續他的研究成果後能順利進行其他領域的探討，在此由衷感謝學長的幫助。

研究所的同學們在實驗進行的階段提供機器使實驗能順利進行，感謝莊柏穎同學提供數台 Linux 電腦供實驗使用，也感謝黃慈霖同學在專業知識與實作技術上的協助，以及同實驗室的莊昆隆、郭家禎、鄭雅玲、許棣堂、巫志彰在平時遇到問題時能彼此互相討論與幫助，讓研究生涯進行得更順利。

最後感謝家人毫無保留的支持，且忍受在面臨時間的壓力下脾氣變得暴躁的我，沒有爸爸媽媽的鼓勵與安慰，無法撐到論文完成。感謝一路走來身邊所有人的幫助。

黃曉梅 謹致

台大資訊管理研究所

97 年 7 月

Abstract

Graduate Institute Of Information Management

Nation Taiwan University

Student: Hsiao-Mei Huang July, 2008
Advisor: Yuh-Jzer Joung

Search and Cache Mechanism in A Proximity-Aware and Group-Based Peer-to-Peer System for Range Queries

Peer-to-peer systems have emerged as a powerful platform for large-scale distributed information systems in recent years; important functionalities, such as searching, have been added to improve the system's lookup capability. The efficient support of range queries in a P2P system is still a challenging problem. To resolve this problem, some other related issues must first be addressed, such as high communication overheads, load imbalance among peers and preservation of object availability. Donuts satisfies above requirement and maintain object availability and effectively range queries. Within a grouping environment, peers are divided into several groups to provide the flexibility of proximity and feasibility of load balance in a range queries system. However, Donuts' search algorithm with cooperative search is not efficient. The search cost is tremendous. Also the query pattern in Donuts is not similar to real word.

This thesis provide more factual query pattern to simulate search behavior. Considering locality and popularity, we try to find the best search and cache strategy. The result of experiment prove that chordal ring is the better system data structure. We also compare the performance of local cache with the one of cache by path. Finally, we compare cooperative neighbor cache with traditional cache and make a conclusion for best search and cache method.

Keywords: Peer-to-Peer, Range Queries, Proximity, Grouping, Load Balance

論文摘要

作者：黃曉梅　　民國九十七年七月
指導教授：莊裕澤教授

論文題目：

考慮地域鄰近性與群組化的點對點範圍查詢系統的搜尋與快取機制

點對點系統近年來成為大規模分散式資訊系統的強大平台，搜尋系統資源是此種平台的一種十分重要的功能，並且已經有不少的研究提出改善系統搜尋效能的方法；然而，點對點平台並不是對於任何類型的查詢都能有效的支援，例如對於範圍查詢的支援仍然有限。要在點對點平台上有效的支援範圍查詢，首先我們必須解決影響效能的相關問題：系統內部的高通訊成本、點與點之間的負載不平衡以及維持系統裡物件的有效性。Donuts 是滿足上述效能要求的點對點範圍查詢系統，它可以維持物件的有效性及有效地支援範圍查詢，利用群組化的概念來解決地域鄰近性與負載平衡之間的衝突；然而 Donuts 提供的搜尋演算法-“先搜尋鄰居列表與群組成員鄰居列表、再利用繞路表搜尋”的方式所花費的搜尋成本過於昂貴，且模擬的搜尋樣式與真實點對點系統的情形並不相符。

在本篇論文裡我們提出與實際搜尋行為較相似的搜尋樣式-“考量地域性與物件受歡迎程度的搜尋樣式”，並以之為基礎研究搭配 Donuts 特性的搜尋與快取策略。我們的實驗結果證明了 Chordal Ring 是適合 Donuts 的系統資料結構，並且比較了傳統快取方法-本地端快取與沿路快取的搜尋效能；最後比較搭配 Donuts 特性的合作式鄰居搜尋的快取方式與傳統快取的效能，深入討論最適合的搜尋與快取策略。

關鍵字：點對點網路、範圍查詢、鄰近性、群組化、負載平衡、快取

目錄

圖片目錄

表格目錄

第一章簡介

1.1 背景介紹

近年來由於網路的普及與便利性，越來越多的網路應用與服務大大地改變人們的使用行為，例如點對點(Peer-to-Peer)網路的應用；點對點網路的優勢在於能提供大規模、可靠的、且能自我組織的系統，能省去維護主機、頻寬等資源的成本，而點對點檔案分享是行之有年且極受歡迎的網路應用，吸引了學術界與企業界的高度關注，像是 Gnutella[1]、Freenet[2]、Bittorrent[3]；這些網路的結構已非傳統主從式架構(Client-Server)，而是點對點的平等式架構，每個節點兼具主人與客人的身分，可以分享包含電腦運算能力、儲存空間、檔案給其他節點之外，也能搜尋點對點網路中是否有自己想要的資源；在上述列舉的非結構化的點對點網路裡，節點的加入與離開對整個點對點網路的影響不大，能以比較彈性與經濟的方式來維持整個點對點網路的正常運作，但昂貴的搜尋成本一直是非結構化點對點網路的一大缺點，像 Gnutella 與 Freenet 是用洪水氾濫法與盲目搜尋整個網路的搜尋策略，通常要繞一大圈才能得出搜尋結果，幸運一點可以順利找到想要的檔案，但若花費昂貴的搜尋成本最後卻得出搜尋失敗的結論，就不是我們樂於見到了。

與非結構化點對點網路相反，結構化點對點網路的設計在於希望提供結構穩定的系統並使搜尋成本有效降低；節點的加入與離開有一定的運作規則，物件則透過分散式拼湊表(DHT)的對應，必須加在對應到的特定節點上，使用 DHT 的結構化點對點網路有 CAN[4]、Chord[5]、Pastry[6]…等，DHT 的優點是能保證搜尋時有合理的搜尋成本，假設整個網路的節點數是 N，繞路花費步數的上限為 O(log N)。在〝考慮地域鄰近性且群組化的點對點範圍查詢系統〞(Donuts [28,29])裡，系統的基本架構是類似 Chord 的環狀結構(圖 1-1)，每個節點會在路由表裡紀錄 2× log N 個 high-links，log N 個 links 指向排列在之前的節點，另 log N 個 links

指向排列在之後的節點，以便未來搜尋時能利用指向前後的 high-link 快速逼近搜尋目標以減少跳躍搜尋的次數；每個物件透過 DHT 的轉換能得到一把特殊的〝Key〞，而每個節點負責一段範圍，若某物件的 Key 值落在某個節點所負責的範圍裡，該物件必須放置到此節點上，以圖 1-2 為例，節點 S 負責(U,A〕的範圍，放在 S 的物件就會像是 Uncertain、Wednesday、Audacious 諸如此類的 Key 值；在搜尋的時候，我們可以透過繞路表查到搜尋目標的節點，再到此節點作範圍查詢。

由於範圍查詢經常會在環狀結構上連續幾個點作搜尋，Donuts 的設計概念是想盡量讓在實體位置上相近節點在系統中的位置也是相近的，這樣能讓實體位置相近的這些節點彼此間的傳輸成本最經濟；又由於在範圍查詢的系統裡若想實現節點間負載趨近平衡的目標，節點在系統中的位置必須要保持一定的彈性，這個要求與考量地域鄰近性的目的有衝突，Donuts 設計了群組化的方法來解決這個問題，如圖 1-3 所示：把相鄰的節點們分配到數個小組裡，每個節點再各自用 GroupList 來記錄離每個小組實體距離上相距最近的節點是哪一個;以圖 1-3 為例，S 的 GroupList 中記錄了第三小組中離 S 最近的節點是 V、第二小組中離 S 最近的是 U。

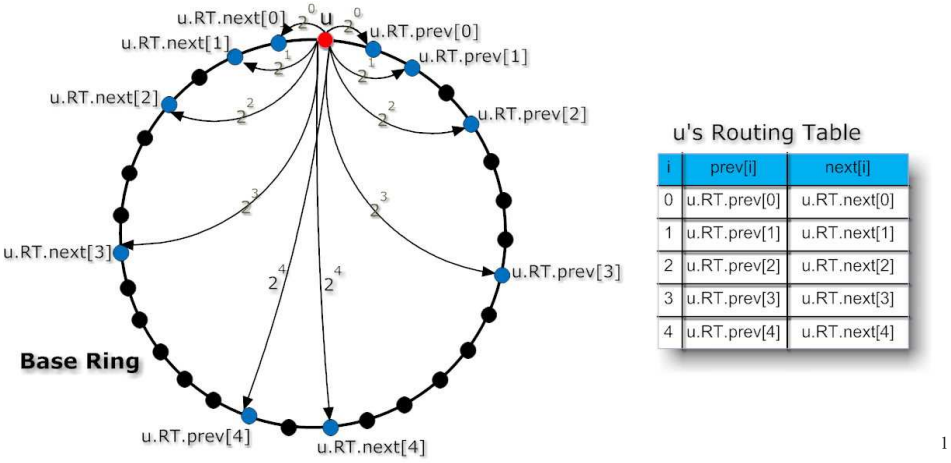


圖 1-1：The high-links of u for efficient routing

[1] 圖 1-1~圖 1-3 來源：Wing Tat Wong, A Proximity-Aware and Group-based Peer to Peer System for Range Query.

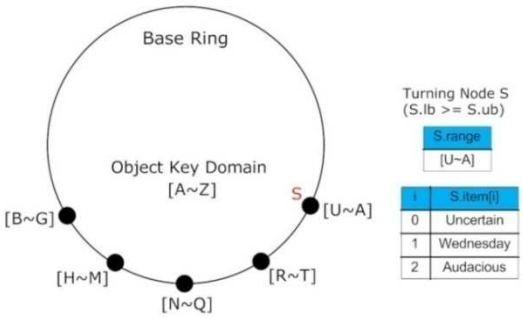


圖 1-2：The basic structure

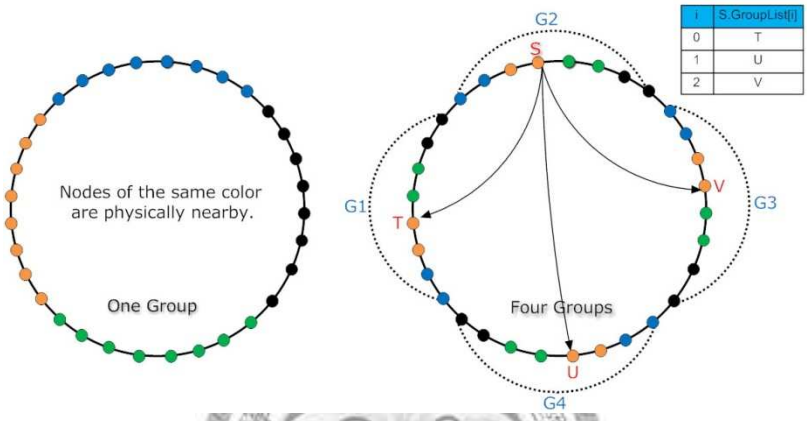


圖 1-3：Left-Overlay without grouping, Right-Overlay with 4 groups

快取(Cache)已被證實是一種能改善搜尋效能的技術，當節點發一個搜尋的請求後若有成功搜尋到標的物，節點會複製一份在快取儲存區裡，日後若又對同一個標的物發出搜尋請求時，先到快取去尋找且成功找到了，這樣就節省一次網路搜尋成本。在眾多的快取技術裡很多都是以點對點系統為設計架構，目前快取方法的設計可分為本地端快取(Local cache)與合作式快取(Cooperative cache)；本地端快取是讓每個節點本身配置一塊快取儲存區，用來儲存節點先前成功搜尋到的物件，若此節點位於其他節點的搜尋路徑上，也能順便儲存其他節點找到的物件。合作式快取則是讓數個節點彼此共享快取儲存區，假設 A、B、C 三個節點一起做合作式快取，對 A 節點而言，它可以存取的快取儲存區是原本作本地端快取的三倍大，若 A 想搜尋一個物件時，發現 C 的快取儲存區裡有此物件，A 只要跟 C 要這份物件即可，省去搜尋網路的成本；若 A 搜尋到一個物件是 A、B 和 C 的快取儲存區都沒有的，再把此物件加入快取儲存區裡，這樣的作法可避免數個節點

上重複的物件過多，尤其是對大受歡迎且搜尋頻率高的檔案而言，可以有效控制檔案的重複率。

而節點要遴選哪些成員一起做合作式快取的選擇方法又是一門學問，若是挑選的節點不適合自己，會造成合作式快取的使用率過低，這樣一來就無法達成快取的設計目的—降低搜尋成本，而且還浪費記憶體空間；更糟一點的情形，若一起做合作式快取的成員在實體位置上相距過遠，即使快取使用率很高，但實際傳輸花費的成本比搜尋一個實體距離較近的節點的成本還高，這樣也不是一個好的合作式快取成員；而 Donuts 正需要一個適合此系統特性的合作式快取方法來改善搜尋的效能。

1.2 動機

Donuts 最大的特色即是在系統結構中保留實體網路裡節點相對遠近的關係(Proximity)，而在[29]裡，搜尋樣式僅實做隨機搜尋—隨機挑選節點發搜尋請求、搜尋隨機物件，隨機搜尋的搜尋樣式與實際點對點網路的搜尋模式並不相符，搜尋樣式應再考慮地域性因素與物件受歡迎的程度，因此本論文的第一個動機即是做出與實際點對點網路搜尋習慣較為相似的搜尋樣式，並搭配 Donuts 獨有的鄰近性特色去研究搜尋與快取的機制該如何運作能有最好的搜尋效能。

[29]實作合作式快取的方法僅選擇在環狀結構上彼此相鄰的數個節點一起做合作式快取，這樣的選擇方法能讓合作式快取成員間的傳輸成本較為低廉，前提是這些成員們的搜尋喜好必須很相似，這樣才能使快取使用率真正提高；但這樣的情形不太可能發生，因為搜尋喜好必須要透過長時間觀察才能得出一個節點的搜尋模式，不可能事先規範或預知一個節點可能的搜尋喜好。[29]搭配合作式快取的搜尋方式是在做遠端繞路前先搜尋合作式快取成員的快取空間，若計算此種方式的平均搜尋路徑長度，會發現比原先只做遠端繞路的平均搜尋長度高出甚多(如圖 1-4 所示)，由此可知合作式快取成員個數過多，使整體搜尋長度過高，同

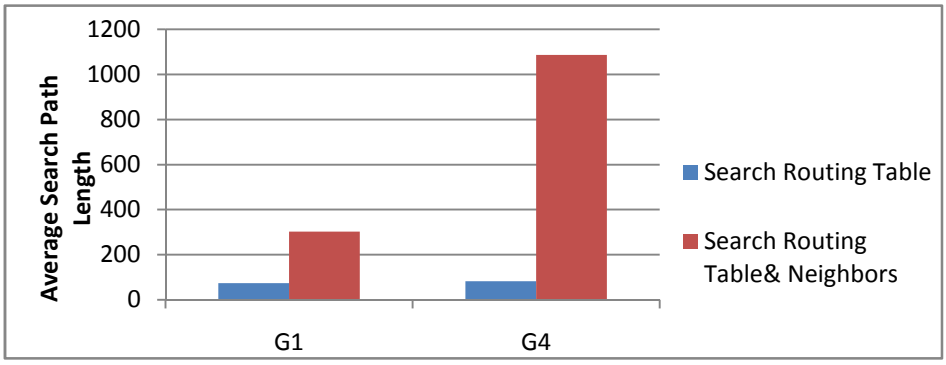


圖 1-4：Compare Average Search Path Length under Different Search Strategy

時合作式成員的快取使用率亦非常低，對於如何挑選合作式快取成員仍存在很大的改進空間。圖 1-5 是節錄自[29]實驗結果的圖表，此圖的實驗目的是想驗證合作式快取成員以不同的快取儲存空間來做搜尋時的平均搜尋路徑長度有無明顯差異，比較圖中藍線與綠線可知：當快取儲存空間變大時，平均搜尋路徑長度並沒有明顯比本地端快取減少很多，減少的長度小於 1 個 hop 數，可見在此系統的合作式快取策略仍存在很大的改進空間值得我們去探討研究。而 Donuts 的特性是節點在 overlay 的相對位置配置考量了實體位置的相對遠近，若我們能設計與此特性相搭配的合作式快取，除了提高快取使用率，減少實體傳輸成本也是改善搜尋成本的一環。

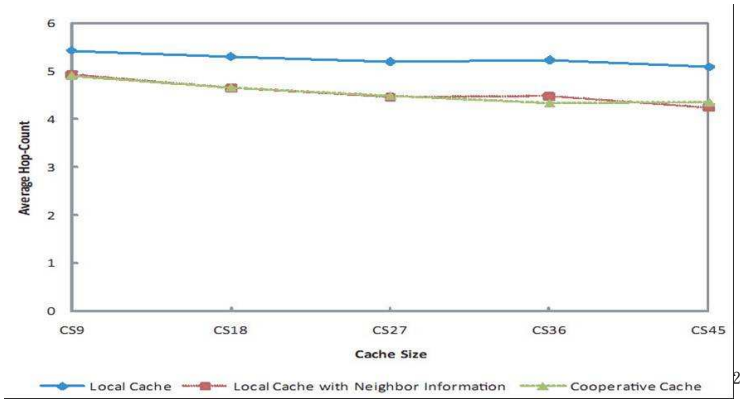


圖 1-5：Average hops under different cache size

[2] 圖 1-5 來源：Wing Tat Wong, A Proximity-Aware and Group-based Peer to Peer System for Range Query.

1.3 研究目標

我們在 Donuts 上討論搜尋與快取機制，討論最適合範圍搜尋的 Donuts 系統設定，並比較本地端快取、沿路快取傳統快取方法與合作式快取的效能，希望找出最適合 Donuts 系統特性的快取策略。

Donuts 的特性是系統中節點的位置與節點紀錄的檔案會適度反映實體位置的相對距離，也就是說能知道哪些是距離較近的節點，能使傳輸成本較經濟，而它實做合作式快取時僅考慮傳輸成本，並未考慮更多因素，因此造成快取儲存區的使用率成效不彰，並未有效減少搜尋成本；我們想要善用 Donuts 的系統特性去設計搭配搜尋演算法的合作式快取機制,讓合作式快取儲存區的使用率能有效提高，進而降低搜尋成本。

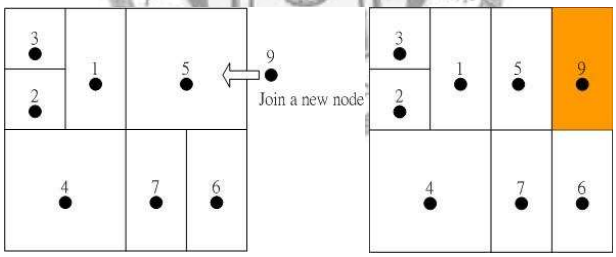
此外我們需要實驗驗證合作式快取機制與傳統快取方法的成效，評估何種方法最適合 Donuts，歸納出在何種情形下最恰當的搜尋設定。

第二章 文獻探討

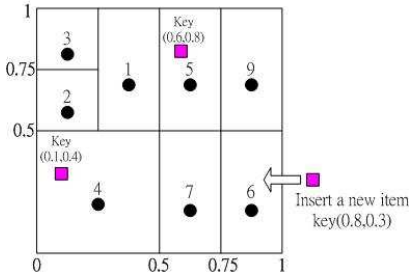
2.1 結構化點對點系統

2.1.1 CAN

CAN[4]是以分散式雜湊表為基礎的系統，將 D 維笛卡兒空間切割成 N 塊相異的區域，圖 2-1 是 2 維的 CAN 範例。新加入的點隨機挑選一個區域，該區域則一分為二由兩個點負責，如圖 2-1(a)所示，新加入的節點 9 與原本區域負責的節點 5 平均分配原有的區域；區域相鄰的節點則互稱”鄰居”，每個節點最多有 2D 個鄰居(以 2 維 CAN 來說)，圖 2-1 中節點 7 的水平方向鄰居是節點 5、6，垂直方向的鄰居是 5。要配置到 CAN 的物件首先將物件名字經過特定的雜湊函數操作得到一把”Key”，key 的值對應到 CAN 裡面的其中一塊區域，物件也由此區域負責的節點儲存；以圖 2-1(c)為例，欲加入一個新的物件，其 key 值為(0.8, 0.3)，對應到節點 6 負責的區域，於是將物件放到節點 6。



(a) (b)



(c)

圖 2-1:(a) Before node 9 joins.(b) After node 9 joins, node 9 splits the zone with node 5.(c) Insert a new item

當節點想搜尋某物件時，將欲搜尋的物件名字以同樣的雜湊函數操作得到對應的 key 值，搜尋路徑的選擇則是依照貪婪法則－每一步都走訪與 key 值最接近的鄰居，平均搜尋路徑長度為$(\frac{d}{4}) \times N^{\frac{1}{d}}$，複雜度上限為$O(dN^{\frac{1}{d}})$，以圖 2-2 為例，節點 6 發出搜尋請求，算出 Key 值決定走訪哪個鄰居，再逐步逼近目的地，由於路徑的選擇是非決定性的，節點 6 到節點 1 有數條路徑可供選擇，但由於繞路是採用貪婪法則，CAN 並未提供選擇繞路往哪個鄰居的彈性。

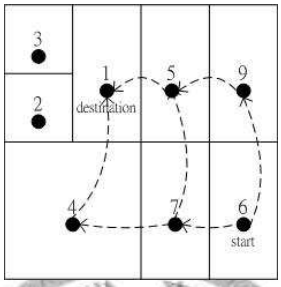


圖 2-2：Route selection flexibility in 2-D CAN

2.1.2 Chord

Chord[5]的基礎結構是一個環，節點的位置與其所負責的物件範圍是由一致性雜湊決定的一維環狀空間。如圖 2-3 所示，要新加入一個節點到 Chord ring 之前，將約定好的節點資訊(如區域名稱或 IP 位址)經過雜湊運算得到節點的 ID，此 ID 決定該節點加入到環狀結構的什麼位置之上，圖 2-3(a)中新增一個 ID 為 7 的節點，其位置在節點 1 與節點 13 的中間。要插入物件到 Chord 的做法與新增節點類似，先將物件名稱經過雜湊運算得到物件的 ID，物件由節點 ID 大於物件 ID 但不小於的節點所儲存，圖 2-3 中節點 7 未加入之前，節點 13 所負責的物件 ID 依序是 4、6、10、12，皆介於 1~13 之間，節點 7 加入之後，依照前述的物件管理規則，節點 13 將物件 4 和 6 移轉給節點 7。在 Chord 裡，節點與物件大致上被均勻分布到環狀結構上，每個節點所負責的物件個數差異並不大。

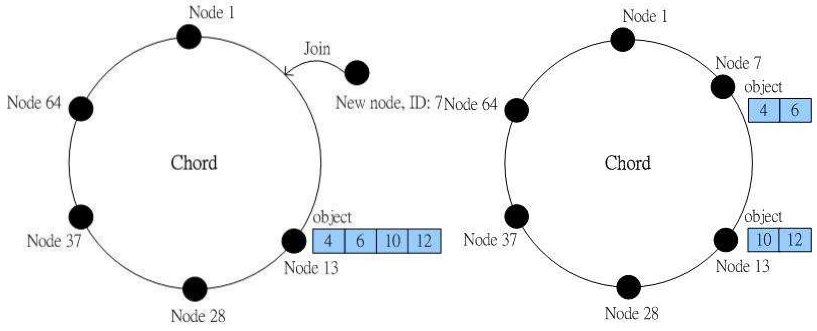


圖 2-3：Chord (a) New node join. (b) Object shift.

為了增進搜尋的效率，每個節點維護一張大小為 log N 的”Finger Table”，N 是節點總數，Finger Table 中的第 i 項元素是一個指標，指向離自己順時針方向 2^i 的節點，以圖 2-4 為例，節點 6 的 Finger Table 中的的 3 個元素指向節點 14($6+2^3$)，若繞路時依照貪婪法則每次都選擇最佳的”Finger”(離目標最近的節點)，則最多需要 O(log N)步。又為了不讓 Chord 因為節點的離線造成環狀結構斷裂，每個節點額外再維護大小為 k 的 successor list。與 CAN 相同，Chord 在選擇繞路對象時也是採用決定型演算法，並無提供選擇的彈性。

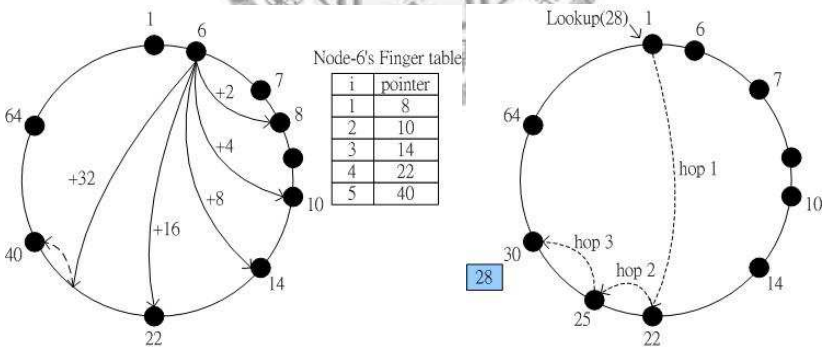


圖 2-4：Chord (a)Routing Table (b)Search Routing

2.1.3 Pastry

Pasrty[6]的實作方式與 Chord 同樣是環狀結構，不同之處在於 Pastry 的繞路演算法是採用 PRR[7]。每個節點以自己的名稱經過雜湊運算得到一個特殊的 ID，

節點根據 ID 的大小以順時針方向排列；同時每個節點都要維護一張 $\log_2{}^bN$ 列×2^b 行的繞路表(N 為總節點數，b 為設定參數)，Level i 的分錄必須繞路到至少 i 個前序位元吻合的節點，PRR 繞路演算法即是在每一步驟都使一個單位前序位元吻合，再逐步接近目標，假設圖 2-5 中的節點 2130 想搜尋 key 為 0111 的物件，首先會先從 Level 0 中挑選第一個位元與 0111 吻合的節點，也就是 0321；接著再查節點 0321 的繞路表第 Level 1，再繞路到與 0111 有 2 個前序位元吻合的節點 0123，以遞迴的方式逐步接近目標，假設繞路到 0113 發現已是最接近目標 ID 的節點了，則在此節點找到欲搜尋的物件，繞路在此結束。由於節點 ID 的長度最多為 O(log N)，可推理得知 PRR 演算法最多需要 O(log N)步驟。Pastry 提供選擇繞路鄰居的彈性，但到 level 較高的分錄則因為前序位元限制變長了，鄰居選擇的彈性較 level 低的分錄較低。

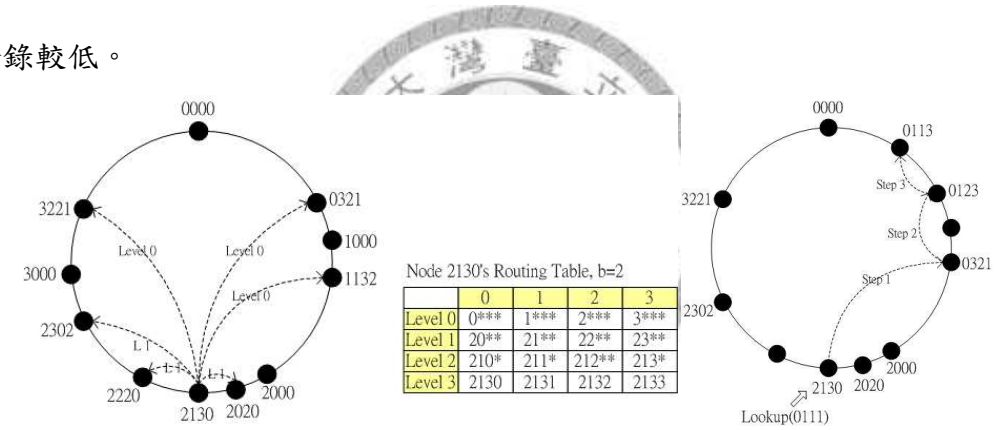


圖 2-5：Pastry (a)Routing table (b)Search routing

2.1.4 Skip graph

Skip graph[30]是以 Skip list 為基礎的分散式資料結構，可支援範圍查詢，Skip graph 並未使用 DHT 的配置方式，因此我們可以依照自己想要的順序去建立節點的相關位置與配置物件的存放處，物件的相對關係可以反應在資料結構上。Skip list 是一種搭配隨機演算法產生的隨機平衡樹，搜尋的複雜度上限可保證在 O(log N)之內(N 是 Skip list 的節點總數)，Skip graph 是 Skip list 的集合，而 Skip list 又是一群 linked list 的集合，如圖 2-6 所示，在 Skip graph 的底

層 Level 0 是一個 base ring，每個節點都有雙向指標指向與自己相鄰的節點；第 Level i 的指標預期有 $1/2^i$ 的機率可建立指向與自己相距 2^i 的節點，透過每層 Level 的這些指標，可讓搜尋的進行以”跳躍”的方式逐步接近目標，Skip graph 平均會有 O(log)個 Levels (N 是節點總數)。

在 Level i 的指標指向與節點本身前序 i 位元相同的節點，一共由 2^i 種不同的 linked lists 組成，每個節點根據自己的 ID 來決定屬於哪一條 linked list，而節點的 ID 是隨機決定的，每個節點平均維護 O(log N)個指標，搜尋的運作是由最高的 Level 開始，在 Level i 可以一次縮短 2^i 單位的距離，以遞迴的方式一層層 Level 往下到 Level 0 結束。

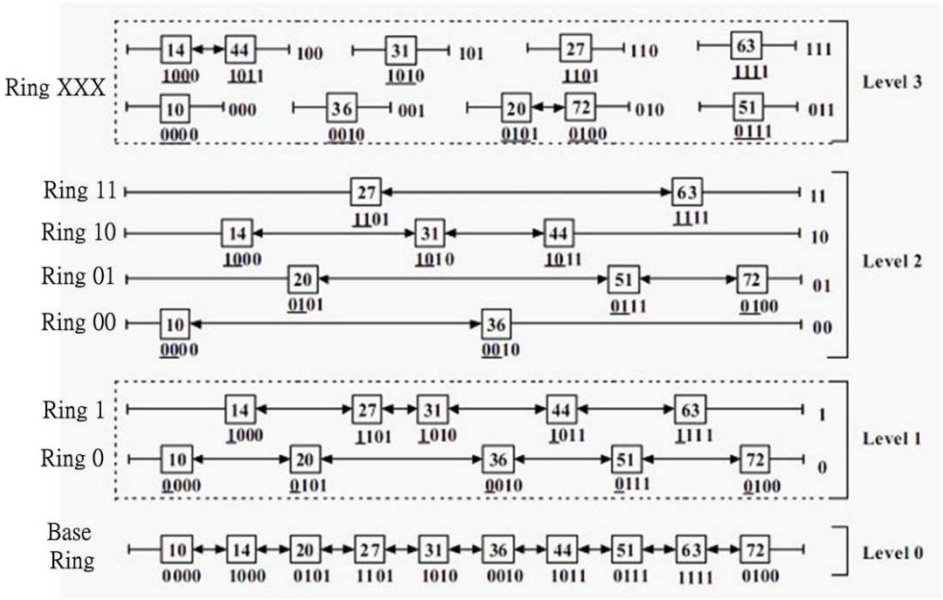


圖 2-6：The structure of Skip graph

2.1.5 總結

前文所述以 DHT 為基礎的結構化點對點系統－CAN、Chord、Pastry，以及特別用來支援範圍查詢的 Skip graph 都能滿足分散式系統的設計理念：量化、錯誤容忍、完全分散式。然而 DHT 系列的系統中，節點所屬位置的決定與物件的配置皆經過雜湊運算，物件資訊與物件間的相關性都消失了，使得範圍查詢難以進行，節點的位置也被雜湊機制給強制決定，造成負載不平衡的問題。反觀 Skip graph

由於並未使用 DHT 的方式建立系統,保留了物件彼此間的相關性以利做範圍查詢，節點在系統中的位置也具有較高的彈性，適度減緩負載不平衡的問題，但 Skip graph 的結構較為複雜,在動態的環境中(節點上下線頻繁時)若想維持 Skip graph 的正確性，必須要付出較多的維護成本。

2.2 搜尋樣式的相關議題

實際網路中的搜尋樣式有地域性的特徵。[40]的研究觀察 Napster 與 Gnutella 網路中節點的行為，發現檔案的地域性呈現長尾分布，在系統中將受歡迎的檔案事先放到節點的快取裡，可提升系統整體的搜尋效能；[41]提出利用空間性地域特色來改善點對點系統的效能，每做一次搜尋時，會順便回傳與搜尋目標相關的其他物件的地理位置，根據檔案存取有空間性地域的特徵，若日後要搜尋的物件恰巧在先前的搜尋時已得知地理位置了，則可省去繞路成本。

網頁快取機制的研究也提供很多快取方法的啟發;為了加速網頁讀取的效率，Proxy 主機在網路架構中扮演快取的角色,而 Proxy 主機會架設在離客戶端較近的地方，使存取成本較低。[43]研究時間性地域因素對網頁 Proxy 效能的影響，觀察長期受歡迎程度與短期時間性地域特徵，通長短期時間性地域的影響較大是客戶端沒有做本地端快取造成。

一些觀察實際搜尋行為的研究讓我們得知實際網路中的搜尋樣式的基本特色，善用時間性地域可讓未來搜尋受歡迎物件的搜尋成本降低，利用空間性地域則可預知未來發搜尋的節點約在何處，或未來搜尋哪些相關物件，而系統中會不同程度顯現兩種地域性特徵，在設計搜尋與快取機制時考量地域性因素，可讓搜尋與快取效能更有效率。

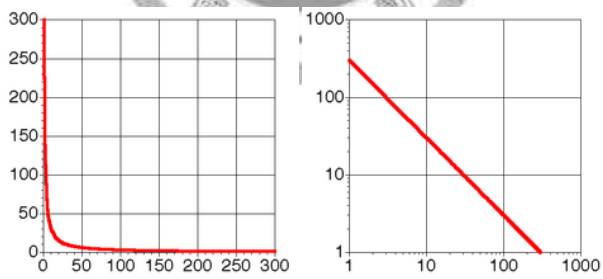
2.3 快取

快取是用來改進搜尋效能的一種很普遍的技術，在點對點網路裡過去文獻提出的快取方法主要分成兩類：本地端快取與合作式快取，本地端快取是由個別節

點獨自作快取，合作式快取則是由數個節點一起做快取，快取儲存區可共享使用，我們將針對這兩類快取機制做文獻探討的說明，並說明過去快取取代機制的研究成果。

2.3.1 本地端快取

E. P. Markatos 的研究[8]指出：在 Gnutella 網路裡，搜尋請求時常會重複且搜尋請求的分布會類似 Zipf dtstribution，若分析一個物件在 Gnutella 網路裡受歡迎的程度與被搜尋的頻率，發現最受歡迎的檔案數很少，但被搜尋的頻率卻非常高；而大部分的檔案都是不受歡迎的，被搜尋的頻率也不高，如圖 2-7(a)所示，300 個檔案裡，最受歡迎的檔案被搜尋的次數將近 300 次；而大部分的檔案被搜尋的次數卻趨近於零次，圖 2-7(b)則是將座標軸換成對數規模。這個研究也分析了在 Gnutella 網路裡快取的效率並評估各種快取取代策略。E. P. Markatos 指出本地端快取在點對點系統是能有效降低搜尋成本的，然而若網路環境變動太快或節點加入與離開系統的頻率太高，快取儲存的物件會很容易過時，也就不適合當時的搜尋了，因此本地端快取的效能提升很有限。



(a) (b)

圖 2-7：(a) Linear scale of Zipf-like dist. (b) Log scale of Zipf-like dist.

C. Wang 等人[9]也研究過本地端快取，發現本地端快取會造成相鄰節點的快取儲存區裡有大量重複與不必要的物件，除了造成節點本身電腦儲存空間的浪費外並沒有明顯幫助搜尋效能提升；因此我們可以推論相鄰節點若搜尋喜好類似，就不應該使用本地端快取的方法，以免造成搜尋效能不彰。S. Patro[10]等人提出中央集權式的主機在組織對外的門戶上做快取的方法，並證實這樣的作法比在

個別節點上做本地端快取更有效率；然而，這些架構並不適合完全分散式的系統。

2.3.2 合作式快取

合作式快取主要的構想是希望消除節點的快取儲存區裡重複的物件，對非常受歡迎的檔案而言，一個節點對它有興趣，同樣地其他節點對它也有興趣的機會很高，因此沒有必要每個節點都把受歡迎的檔案放到快取裡。B. M. Duska 與 M. Kacimi 等人的研究[11][12]證實合作式快取能提供比本地端快取更為顯著的搜尋效能改善。Summary Cache[13]提出一個合作式快取的系統架構，在這個系統裡，每個節點都可使用其他所有節點的快取，節點之間透過定期交換快取資訊的方式來聯絡告知目前快取儲存區裡有哪些物件，使用的方法是 Bloom filter[3]，由於節點之間交換的資訊僅是 Bit vector，資料量不大，即時時常分享快取資訊並不會造成巨大的頻寬消耗；但另一方面也有一個明顯的缺點存在—每個節點必須要完整紀錄系統中其他所有節點的快取資訊；又由於可以知道其他節點的快取資訊，若節點發現自己的快取裡某個物件遺失了，而其他節點的快取仍存在此物件，就可以利用其他節點的快取。

有一些研究為了避免節點交換快取資訊，提出藉由將每個物件配置給特定的節點管理的方法，Consistent hashing[14]的機制下，一個節點可以在不用事先知道快取資訊的情形下配置物件的位置，並且能允取合作式快取成員經常性的更改快取物件；使用這類方法的系統像是 Squirrel[15]、Kache[16]，是建構在結構化系統之上，用來提供有效率地且支援大規模網路的合作式快取。雖然這些系統實做合作式快取能減少大量的頻寬消耗，但他們的架構並不適合用來支援範圍查詢的系統。前面提到的合作式快取方法中，每個節點仍要消耗特定量的傳輸頻寬，Roussopoulos 等人的研究[17]提出一個鼓勵機制讓每個節點可決定何時需要傳送更新的快取摘要，節點在做決策之前會考量一個物件在本節點受歡迎的程度和快取資訊的改變率。

[3] 一種用 bit vector 來表示節點目前的物件集合的方式，所有節點共同使用數個 hash 函數

2.3.3 快取取代策略

當快取儲存區已滿而想加物件進去的時候，節點必須要從中挑選一個物件並移除之，讓多出來的空間可加入新的物件；快取移除策略根據[31]的分類主要分成 5 種：以最近發生的事為基礎的快取取代策略(Recency-Based Strategies)、以頻率為主要考量因素的策略(Frequency-Based Strategies)，還有結合兩種因素的最近發生的事/頻率為基礎的策略(Recency/Frequency-Based Strategies)，此外也衍生了把各種因素一起考量的以函數為基礎的策略，最後是隨機策略。

早期的研究是以最近發生的事為基礎的快取取代策略，是根據過去的經驗法則來管理本地端的快取儲存區，以 LRU(least recently used)為基礎衍生出各種演算法；例如[33]加入物件大小因素的考量，LRU-Threshold 移除物件大小大於特定門檻的物件、移除最少量物件的 LRU-Min，PSS(Pyramidal Selection Scheme)[34]以物件大小做分類，每一種類別裡用獨立的 LRU 表管理物件；也有純粹依物件被存取的時間間隔來區分物件重要性的 EXP1[35]。

以頻率為基礎的快取取代策略(LFU)考量的是物件被存取的次數，不同的物件受歡迎的程度也不同，其中的差異會反應在物件被存取的次數，LFU 即是考量物件被存取的次數來為未來做決策。在本地端的 LFU 演算法通常只能從現有的快取物件去評估物件有多常被存取，LFU-Aging[36]利用門檻的機制來做：若平均物件存取次數超過門檻設定值，所有的頻率計數器會減半；LFU-Aging 過於依賴門檻參數的設定值，容易受參數設定影響整體的演算法效能，[36]另提出 LFU-DA，當快取中某物件被存取了，會驅動每個物件的 K 值，K 值的計算方式：$K_i = f_i + L$，L 是老化參數，初始設為 0，f_i 是物件目前被存取的頻率，新的 K 值會當作下一次計算 K 值時的 L 參數。LFU 系列的演算法都有一些共通的缺點：計算方式複雜、快取儲存區的汙染，而且物件被存取的頻率極有可能相同，必須衍生區分此種情形的對應策略。

結合 LRU 與 LFU 特色的演算法是進一步發展出來的快取取代策略，SLRU[36]

與LRU-hot[37]的方法都是區隔受歡迎物件與非受歡迎物件,兩者以不同的LRU表、不同的計數器來記錄物件被存取的頻率；然而這類演算法若要再額外考量其它因素，例如物件的大小，快取取代機制的設計將變得較複雜。以函數為基礎的策略則是事先定義用來計算物件的特徵值的函數；LLF(lowest latency first)[18]和GD-size(Greedy Dual size)[19]的作法考慮兩個快取移除演算法最常利用的物件特徵—物件大小與下載物件的時間,LLF 移除下載時間最小的物件，以減少快取遺失的影響;GD-size 演算法則結合物件大小與下載物件時間兩者來決定移除哪個物件；此外還有考量其它因素，像是物件低相關的機率-LRV[45]，而這類以函數計算特徵值的演算法最大困難處在於如何挑選適當的參數。

第三章 系統模型

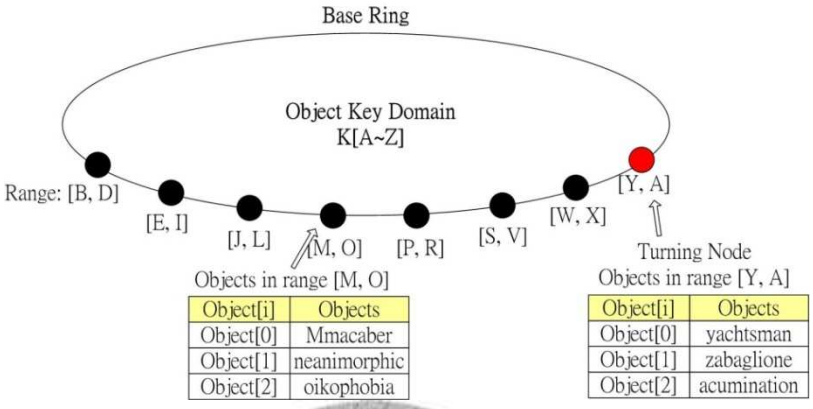
3.1 系統概述

首先介紹用來做範圍查詢的系統平台－Donuts。Donuts 的設計理念是以點對點分散式系統為參考指標，我們假設 Donuts 裡的成員除了提供私有的儲存空間儲存系統物件外，也維護了一塊共享的快取儲存功間，用以提高搜尋的效率；系統裡的成員並不知道所有其他成員的完整資訊、也不知道實體網路的情形，而系統成員會對其所擁有的資訊做出合理可預期的行為；同時我們也假設系統成員可以在任意時間加入或離開 Donuts，且不用通知其他成員。

欲加入到系統中的物件，先透過轉換函數將物件填上一把特殊的”Key”，Key 值的範圍是我們事先定義好的 K，Key 值會介於 0~K 之間，而所有的物件將由 Key 值的大小排定先後順序；我們將 K 切割成數塊小範圍的區段，Donuts 中每個成員將負責保管一段區段的物件，而區段與區段之間沒有重疊的部分，並且所有區段的集合恰好是 K。如圖 X 所示，我們將 K 切割成 8 個區段[lb,rb)，並分配給系統中的成員，每個成員負責的區段範圍除了 turning node 以外，左邊邊界 lb 的值都小於右邊邊界 rb 的值；假設物件的 key 值是 o.key 落在某個區間[lb,rb]，則此物件 o 由負責此區間的成員所保管，例如”oikophobia”落在[M,O]區間，則放置到此區間的節點上。而節點負責的區間是可以動態變更的，例如若有某節點下線了，其負責的區間與此區間的物件將移轉給其他仍在線上的節點。我們所做的範圍查詢即是在一特定的範圍區間裡做搜尋，若欲搜尋某 key 值的物件，必須先找到負責此區間的節點，才對此節點所擁有的物件做搜尋的動作。

Donuts 提供了以下基本的功能：節點的加入與離開、物件的插入與刪除，以及物件搜尋；根據我們的需求，可以選擇 Donuts 建立的基本結構是 Chordal graph 或是 Skip graph，兩者最底層的結構皆是如圖 3-1 的環狀結構，節點必須與左右鄰居保持連結以免環狀結構斷裂；再者，兩種系統結構皆可保證繞路演算法的複

雜度在 O(log N)以內，可預期在少數的步數裡就找到搜尋目標，而 Chordal graph 的結構比 Skip graph 的結構較為簡單，雖然兩者都適合用來實做範圍查詢，但前者在系統維護上的複雜程度較低，而 Donuts 的最大特色即是整合了鄰近性與負載平衡的問題，這兩個互相衝突的因素在 Donuts 裡得到妥善的解決。



Object[i]	Objects
Object[0]	Mmacaber
Object[1]	neanimorphic
Object[2]	oikophobia

Object[i]	Objects
Object[0]	yachtsman
Object[1]	zabaglione
Object[2]	acumination

圖 3-1：Base Ring structure

3.2 基礎環狀結構

Donuts 的底層架構是一個環，所有的節點在邏輯上會排列成一個循環的環狀結構(如圖 3-2 所示)，每個節點 p 會有一個 p.next 指標指向後一個節點、一個 p.prev 指標指向前一個節點，假設 D_N 表示一個基礎環狀結構為 N 的 Donut 系統，|N|表示環狀結構上的節點個數，這時在 D_N 的搜尋方式很類似於在一般連結串列上做搜尋，假設想要搜尋 Key 值為 k 的物件，必須要沿著環狀結構逐一走訪每個節點，查詢走過的節點是否有 Key 值為 k 的物件，如此較沒有效率的搜尋方式的複雜度為 O(|N|)。

為了讓搜尋更有效率，每個節點需維護一張繞路表，繞路表的建立方是遵照 Chord 的定義，依照 2 的指數次方遞增，而 Donuts 裡的節點紀錄了向前與向後的繞路指標，以圖 X 為例，u.RT.prev[i]的指標指向節點 u 往前數第 2^i 個節點，u.RT.next[i]的指標指向節點 u 往後數第 2^i 個節點，i 表示繞路表裡的階層，階層越高的指標可以指向環狀結構裡越遠的節點，表示在做搜尋時可透過高階層的指標到離自己範圍較遠的節點做範圍查詢，Donuts 上的每個節點記錄了 2log|N|個

指標，第 i 階層的指標指向離自己 2i 個節點遠的節點，如此可支援雙向搜尋，且讓搜尋複雜度從原先的 O(|N|)變成 O(log|N|)。

繞路表的建立時機從節點找到環狀結構上的所屬位置後開始，當基礎環狀結構上只有一個節點時，此唯一的節點只有第 0 階層的指標，該指標指向自己本身；而之後加入的節點根據不同的加入演算法找到環狀結構上的位置之後，以”插隊”的方式插入，並通知其前後的節點更新第 0 階層的指標，新加入的節點 p 的繞路表則根據基礎環狀結構來建立，p.RT.prev[0]和 p.RT.next[0]最先完成，而 p.RT.prev[1]則是 p.RT.prev[0]的前一個節點，也就是 p.RT.prev[0].RT.prev [0]，同理可推 p.RT.prev[i]= p.RT.prev[i-1].RT.prev[i-1]，繞路表由低階層逐步建立到高階層，當 p.RT.prev[i].rb＜p.lb 時繞路表建立程序即終止。當新加入一個節點到環狀結構裡時，只有新加入節點的左右鄰居被通知要更新繞路表，而其他節點並不知道新節點的加入是否會影響到自己的繞路表正確性，但通知所有節點所花費的代價太高，Donuts 的做法是讓節點定期修正自己的繞路表，以免繞路表過時太久。

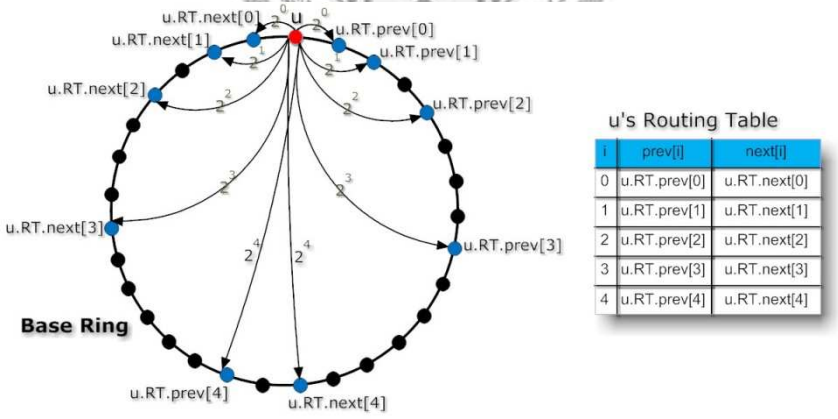


圖 3-2：Routing table of Donuts peer

3.3 鄰近性

節點加入Donuts遵循鄰近性加入演算法，讓實體距離上相近的節點，在Donuts的結構裡也是相近的，如此一來在做範圍查詢時若需要走訪 Donuts 上連續的數個節點，相鄰節點之間的實際傳輸成本會較低。在前文我們提到”負載平衡”的議

題，在一個支援範圍查詢的系統裡若想滿足負載平衡的需求，則節點在系統中的位置必須要能彈性變動，這麼一來就與系統中節點的相對位址須考慮鄰近性的目標有所衝突了，Donuts 利用”群組化”的概念來解決鄰近性與負載平衡的衝突：將 Donuts 上的節點給予分群，屬於同一群的節點們再依照鄰近性排列在環狀結構上，因此在實體距離上相近的點會被平均分到數個群組裡，為了讓實體距離相鄰的點仍可彼此溝通，每個節點須記錄一組 GroupList(如圖 3-3 所示)，用來記錄每個群組裡與自己實體距離最近的節點，如此可滿足鄰近性的節點仍可彼此溝通的目標。

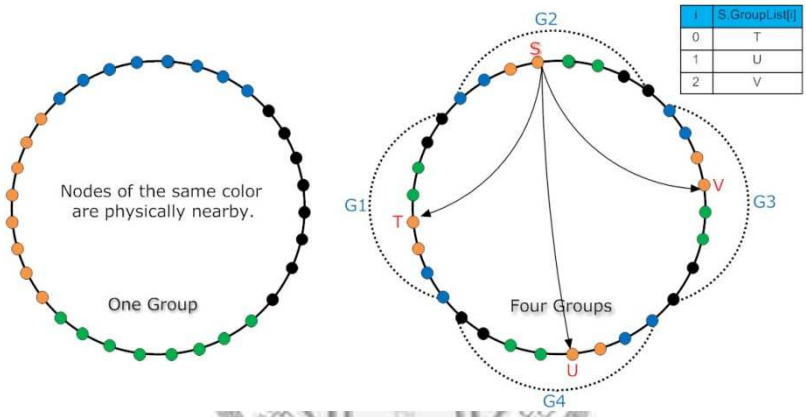


圖 3-3：Left-Overlay without grouping, Right-Overlay with grouping

Donuts 的鄰近性加入演算法是一階層式加入過程，當一個新的節點 n 要加入到基礎環狀結構時，會先找一個已經存在 Donuts 的節點，如圖 3-4 中以節點 v 當作鄰近性演算法的起始，首先從 v.RT 查到最高階層 h0 的指標分別指到節點 h0_1 與節點 h0_2，n 可查自己到 v、h0_1、h0_2 的實體網路最短路徑長分別是 d(n,v)、d(n,h0_1)和 d(n,h0_2)，假設 d(n,h0_1)是三者中最短的，表示 n 離 h0_1 最近，於是 n 接著利用 h0_1.RT 查詢次高階層 h1 的節點，如此以遞迴的方式直到最低階層，也就是最終的基礎環狀結構，如圖 3-4 最終 n 找到 h3_1 就終止了，於是最後 n 要決定加入到 h3_1 的左邊或是右邊；若 d(h3_1.RT.prev[0], n)+d(n, h3_1)小於 d(h3_1.RT.next[0], n)+d(n, h3_1)-d(h3_1.RT.next[0], h3_1)，則 n 加入到 h3_1 的左邊，並通知其左右的節點更新第 0 階層的繞路表。

新加入的節點在挑選最初的”v”時，Donuts 提供了兩種選法：對稱式加入與非對稱式加入，前者的 v 可隨機挑選系統中任一在線上的節點，後者則是選一固定的節點當作演算法操作的起始，兩種做法所花費的空間複雜度皆是 O(log N)；由先前的實驗結果[29]得知在群組數不多的時候，非對稱式加入一般來說較能反應出實體網路的鄰近性。

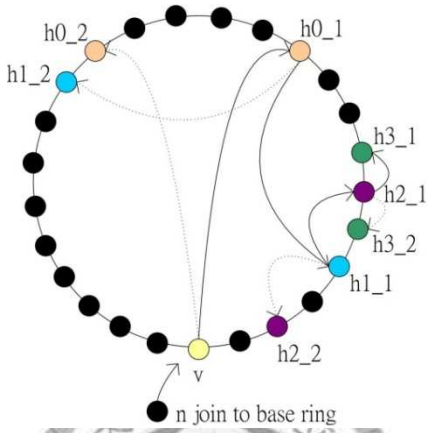


圖 3-4：Proximity join in Chordal ring

3.4 錯誤容忍機制

在動態的網路環境裡，節點的上線與離線會影響 Donuts 結構的完整，若節點離線了，而其他節點的繞路表或是鄰居列表(Leafset)仍紀錄此節點，當在做搜尋的時候走訪到繞路表中離線的節點是無用的，既而將搜尋請求往下一階層傳遞，這時搜尋路徑有很大的機會將大於預計的 log N 步；離線的影響對 Donuts 的基礎環狀結構更大，因為繞路表正是查詢環狀結構所建立起來的，若節點離線無法即時更新環狀結構造成結構斷裂，將會有更多繞路資訊是錯誤的；接著我們將說明 Donuts 如何保持鄰居列表與繞路表是最新的狀態。

3.4.1 鄰居列表

為了基礎環狀結構的緊密連結，每個節點都要儲存一份鄰居列表，用來記錄

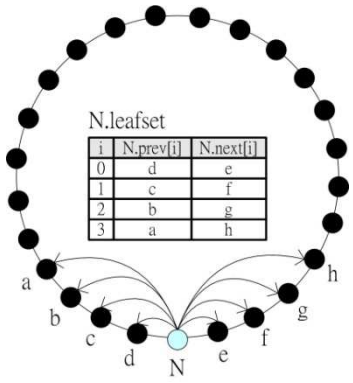


圖 3-5：Leafset of Donuts peer

數個左右鄰居的節點，以免只記錄左右相鄰各一個時因鄰居離線造成結構斷裂；每個節點需要記錄一份大小為 log N 的鄰居列表，N 為節點總數，其中$\frac{\log N}{2}$記錄左邊的鄰居，另$\frac{\log N}{2}$記錄右邊的鄰居，若鄰居列表中有節點離線了，仍可靠列表中其他仍在線上的節點保持與鄰居的相連，除非 log N 個鄰居同時都離線了才可能造成環狀結構的斷裂，圖 3-5 是鄰居列表的示意圖。Donuts 維護鄰居列表的方法是定期更新每個節點鄰居列表的狀態，逐一檢查列表中的鄰居是否仍在線上，若其中有一個離線了，則將其之後的鄰居往前推移一個位置，多出來的位置則再補入新的鄰居。為了保持有最新狀態的鄰居列表，更新列表的時間間隔不能太短，以免因為短暫的鄰居列表不一致影響到搜尋的效能；在更新鄰居列表的同時，節點也必須更新目前所負責物件的 Key 範圍。

3.4.2 繞路表

當新節點的加入或有節點離線時，其他節點有些會需要修正自己的繞路表，而哪些節點需要更新繞路表我們無從得知，Donuts 的做法是讓每一個點定期修正自己的繞路表；首先節點 n 從繞路表裡的最高階層開始檢查指標所指的節點是否仍在線上，檢查的順序由高階層的指標逐一往下，當找到繞路表中第一個錯誤的指標時，假設是 n.RT.next[i]，就由第 i 階層開始往高階層做繞路表修正的動作，當 n.RT.next[i]錯誤時，可由 n.RT.next[i-1].RT.next[i-1]得知正確的繞路資訊，若 n.RT.next[i-1]也是離線狀態，則再往下找 n.RT.next[i-2]，最壞的情況

是查最底層的 n.RT.next[0]；同理上一階層的 n.RT.next[i+1]由 n.RT.next[i].RT.next[i]取代之，直到更新完整張繞路表為止。而以繞路表為建立基礎的其他繞路資訊，例如之後會提到的 Express Table、Proximity Table，在繞路表被更新的同時也需要被一併更新，因此修正繞路表的動作實則是一連串繞路資訊的大更新，並不適合在短時間內更新太多次。

3.5 物件的可利用性

Donuts 以複製物件到其他節點上的方式來改善物件的可利用性。Donuts 裡一個節點負責一段 Key 區段的物件，若此節點離線則其他節點將找不到這個區段的物件，因此節點會把所負責的物件複製一份到環狀結構上的左右鄰居，由於 Donuts 的基礎環狀結構的建立考量了鄰近性的問題，相鄰的節點其實際距離是較短的，傳輸複製物件的成本並不算大。複製的數量可事先決定，複製階層 r 若定為一，則節點的左右兩個鄰居各存一份複製品，若為 r 則左右 r 個鄰居都要存一份，當 r 定的越大，物件的可利用性越高，但每個節點需要提供更多的空間儲存複製物件，頻寬的消耗也較高

3.6 負載平衡機制

節點負責的物件數量並非一致化，會有某些節點負載較多物件、有些節點卻負載少量物件的情形發生，負載平衡演算法的目的即是把物件從負載較大的節點移往負載較小的節點上。首先我們必須定義”負載失衡”的衡量指標，l_n 代表節點 n 的負載指數，也就是物件的數量，當兩個節點 n_1、n_2 的負載變成 l_{n1} 大於 $\alpha \times l_{n2} + \beta$ 時(α 是 n_1 與 n_2 的負載比例，β 是 n_1 必須擁有的最少物件數量)，表示需要啟動負載平衡機制了。Donuts 有兩種負載平衡的機制，分別是鄰居平衡(Neighbor Balance)與節點重排(Peer Reorder)。

3.6.1 鄰居平衡(NB)

鄰居平衡的做法是讓環狀結構上相鄰的兩個負載失衡節點平均分攤彼此的物

件，例如當節點 u 發現自己的負載大於相鄰的節點 v 時($l_u > \alpha \times l_v + \beta$)，負載高的 u 必須移轉部分物件給 v 保管，同時 u 和 v 也要跟著調整 Key 範圍，如圖 3-6 中所示。

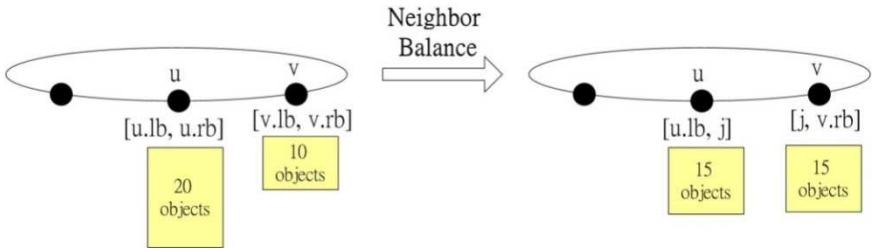


圖 3-6：Neighbor balance

3.6.2 節點重排(PR)

節點重排的做法是讓負載低的節點離開目前在環狀結構上的位置，加入到負載較高的節點旁邊並平分負載物件。在方法實作上，負載低的節點先行離線，此離線的節點所負責的物件範圍將由其原先相鄰的節點所接收，之後這個節點再從新加入到 Donuts 裡，並分攤其鄰居的負載。鄰近性與節點重排的做法有所衝突，Donuts 裡由於節點都被分成數個群組，每個節點會用 GroupList 記錄其他群組裡離自己最近的節點，因此會被挑選來做PR的節點只會是GroupList裡的成員如此，如此鄰近性的原則才不會被破壞；例如節點 u 發現自己的負載超過 u.GroupList 裡的成員，則 GroupList 裡負載最小的節點實行 PR 的步驟：離線、加入到 u 的旁邊並分攤負載物件。

3.7 搜尋

當節點發出搜尋請求時，首先檢查欲搜尋的物件是否就在自己身上，依序查閱自己負責的物件範圍、複製備份、和快取儲存區，若三者都找不到即確定要到系統中去做遠端繞路搜尋，基本的繞路演算法於後說明之；繞路資訊的選擇除了最原始的繞路表之外，根據系統資料結構的不同，另有鄰近表與快捷表可套入搜尋的演算法使用。

```
Algorithm n.Search(key)
1.   if key ∈ n.range
2.        if key exists in the set of objects hosted by n
3.             then return object with the key
4.        else NOT_FOUND
5.   else for i = n.RT.height down to 0
6.             do if key > interval
7.                  then if key ≥ n.RT.next[i].interval
8.                            then n.RT.next[i].Search(key)
9.                  else
10.                      if key ≤ n.RT.prev[i].interval
11.                           then n.RT.prev[i].Search(key)
```

3.7.1 鄰近表(Proximity Table)

繞路表是每個節點在做搜尋時用來查詢繞路資訊的基本資料，繞路表可以幫助搜尋離自己負責範圍較遠的節點，每個節點皆與自己相鄰 2^i 個位置，繞路表裡的節點其相對位置固定，但可能走訪的實際最短距離卻是較長的，表示我們需要花費較多的搜尋成本，因此當我們希望繞路時每一步都走每階層的最短路徑、同時也能搜尋較遠處的節點而不在意每次都要跨越最多 2^i 個節點時，可以利用鄰近表(Proximity Table)來繞路。

鄰近表的建構方式須參照繞路表，當我們要建立鄰近表的第 i 項時，參考繞路表的第 i 項與第 i+1 項，鄰近表的第 i 項指標指向繞路表第 i 項到第 i+1 項之間距離最近的節點；如圖 3-7 所示，節點 v 欲建立鄰近表第 3 項時，須參考繞路表的第 3 項與第 4 項，這段範圍內節點稱之為候選集合，v 從候選集合當中挑選離自己最近的節點當作鄰近表的第 3 項指標。挑選的方法與階層式加入做法類似，首先利用 v.RT.next[3].RT.next[2]找到候選集合的中心點－s.RT.next[2]，在從此中心點為準，每次跨越一半的方式逐步檢查候選集合中是否有距離 v 更短的節點。

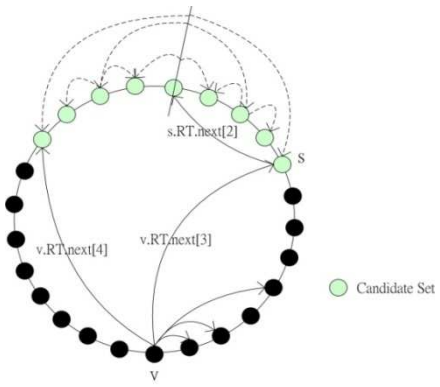


圖 3-7：Build Proximity table

3.7.2 快捷表(Express Table)

快捷表是 Skip Graph 專屬的繞路表。Skip Graph 裡每個節點有專屬 ID，一般以二進位方式表示之，快捷表的第一項記錄節點在第零階層基礎環狀結構的左右鄰居，第二項則記錄第一階層裡第一個前序 ID 一位元相同的左右鄰居、第三項記錄第二階層裡第一個前序 ID 二位元相同的左右鄰居，接著以此類推每個階層指標的意義。由於快捷表的定義是考慮前序 ID 對應到的個數，當物件的配置方式搭配節點 ID 時，可以由節點 ID 去反推物件之間的關聯性，例如可以將一系列相關的物件放置到前序 ID 皆是 010 開頭的節點之上，透過快捷表的第四項則可快速找到相關的物件；而 Donuts 物件配置的方式是依照字母順序分配到第零階層的環狀結構上，並未考慮節點 ID 與物件相關性因素，但仍可以透過快捷表逐步逼近搜尋目標。

3.7.3 搜尋與快取

快取已被證實是改善資料存取速度的重要技術，在 Donuts 裡快取機制的設計目的是希望能減少搜尋的經常花費，若想有效降低搜尋成本並提高快取使用率，快取的設計必須從根本的搜尋樣式思考起。與搜尋相關的網路應用，例如 Web Search 和 P2P 網路，搜尋樣式會反應地域性因素，地域性又分時間性地域與空間性地域；搜尋者端的時間性地域是在一段時間裡由同樣的節點發出搜尋請求，客戶端的時間性地域則是在一段時間裡某些物件經常被重複搜尋；空間性地域則是

當某一節點發出搜尋請求時,在地理上與它相近的節點也會發出同樣的搜尋請求，例如住在同一城市裡的人會對某項發生在此的新聞有興趣，會不約而同去搜尋同樣的標題新聞；而當某項物件被搜尋了，另外其他某物件也會被搜尋，這則是客戶端的空間性地域。時間性地域可反應一段特定時間內物件受歡迎的程度，空間性地域則可反應特定地理區域的搜尋頻率。

當地域性特徵明顯時，本地端快取能滿足搜尋者端的時間地域性，當節點發出重複搜尋請求的比例很高時，本地端快取能節省很多繞路搜尋成本；沿路快取則是將物件複製一份放到搜尋路徑上的節點，若搜尋路徑上的節點也搜尋同樣的物件，則會在它自己的快取中找到，進而省去繞路成本；沿路快取能快速散播物件到整個系統中，若散播的是受歡迎的檔案，對系統整體的平均搜尋成本有所幫助，反之若是較不受歡迎的檔案，沿路快取反而會使節點的快取空間快速達到飽和，並且無法有效提高快取使用率。合作式快取能讓數個節點的快取空間共享，選擇適合的合作式快取成員與快取的效能有很大的關係，後文將說明在 Donuts 環境下該如何實做合作式快取較有效率。我們希望 Donuts 在一般反應地域性特色的 LSZ0 搜尋樣式下，找出適合 Donuts 的搜尋方式與最佳的快取方法。

3.7.3.1 合作式鄰居搜尋

Donuts 可利用鄰近性加入讓地理上相近的節點在環狀結構上也是相近的，當空間性地域特色出現在 Donuts 時，表示環狀結構上某一區段內的多數節點頻繁發出搜尋請求；而受歡迎的物件被搜尋的次數非常高，若每個節點都做沿路快取，則受歡迎的物件會普遍分布在某區段環狀結構上節點的快取儲存區裡，若節點在做遠端繞路搜尋前先搜尋相鄰節點的快取，很有可能就找到欲搜尋的物件，省去繞路的成本，同時相鄰節點間的溝通成本也是較低的。我們做了預備實驗來驗證上述的推理：檢查搜尋目標物件在鄰居列表分布的情形，即是在遠端繞路前先檢查節點鄰居列表成員的快取裡是否有該物件，檢查完畢才做遠端繞路搜尋。

我們先以鄰近性加入演算法建立一個有 5000 個節點的 Donuts 系統，系統參

數設定如表 3-1 所示。我們做三種搜尋樣式，分別是隨機來源-隨機物件(RSR0)、隨機來源-Zipf 物件(RSZ0)、地域性來源-Zipf 物件(LSZ0)，隨機表示挑選的方式是亂數選取，Zipf 物件則是將搜尋物件挑選集合(Q_{set})依 Zipf 分配物件選取的

G	群組數	4
L	Leafset 的大小	14
P	維護基礎環狀結構的時間間隔	30 秒
F	修正繞路資訊的時間間隔	180 秒
α_{NB}, β_{NB}	鄰居平衡演算法需要的參數	1.2,20
α_{PR}, β_{NB}	節點重排演算法需要的參數	2.0,20
Q_{set}	搜尋物件挑選集合的大小	10000
C_S	快取儲存區的大小	15
C_R	快取取代策略	LRU

表格 3-1：系統參數設定

機率，Zipfen 函數的參數設為 1；地域性來源則是定義發搜尋請求的節點依距離遠近挑選較相近的 100 個節點。我們在一實體網路結構上，每一回合的實驗裡我們發出 5000 個搜尋請求，每種搜尋樣式皆做 6 個回合的實驗，搜尋方式參考[29]的做法：先檢查節點自身和 groupList 的鄰居列表裡目標物件分布的情形，最後利用鄰近表做遠端繞路；快取方式則採用沿路快取。

圖 3-8 是統計在 RSR0 搜尋樣式底下鄰居列表的平均快取物件分布曲線，我們發現隨機搜尋樣式的快取分布沒有一定的規則(鋸齒狀曲線)，無論鄰居列表或 groupList 鄰居列表的平均快取存取次數都非常少，皆在 3 次以下，可見在 RSR0 這種搜尋樣式下，在遠端繞路前搜尋鄰居列表與 groupList 的鄰居列表付出昂貴搜尋成本卻沒有實質的效率，快取的使用率亦非常低。與圖 3-8 相比，圖 3-9 是 RSZ0 搜尋樣式的結果，將物件選取機率依照受歡迎的程度去分配，越受歡迎的物件被搜尋的次數越多，且極受歡迎的物件數量占 Q_{set} 的極少數部分，我們發現 RSZ0 搜尋樣式的平均快取物件分布曲線仍然沒有一定規則(鋸齒狀曲線)，但整體的平均快取分布均超過 90 次，因此我們得知搜尋物件依受歡迎程度選取能有效提高快取使用率。

若搜尋樣式加入地域性因素：發搜尋請求的節點侷限於一群彼此相近的節點們，平均快取物件分布曲線則如圖 3-10 所示，Leafset-RJ 和 Leafset-PJ 是統計發搜尋請求節點本身的鄰近列表中搜尋物件出現的次數；Group leafset-RJ 和 Group leafset-PJ 則是統計發搜尋請節點的 GroupList 成員的鄰近列表的搜尋物件平均出現的次數，例如群組數為 4 時，須統計 3 個 GroupList 成員的鄰近列表的快取物件分布。無論是隨機加入演算法或鄰近性加入演算法建構的系統結構，分布曲線都以發搜尋請求的節點為中心向左右兩側鄰居列表遞減，而 groupList 的鄰居列表分布次數皆非常低，因此我們推論搜尋 groupList 鄰居列表較無效率；由此圖可知節點本身的快取物件以及左右鄰居的快取物件能幫助節省搜尋成本：在本地端快取找到物件，搜尋成本是零；在左右鄰居找到物件，僅有與左右鄰居的溝通成本，因此我們進一步推論不用搜尋整個鄰居列表，只要搜尋較相近的數個鄰居即可找到。合作式鄰居搜尋即是先搜尋左右鄰居，再採用鄰近表做遠端繞路。

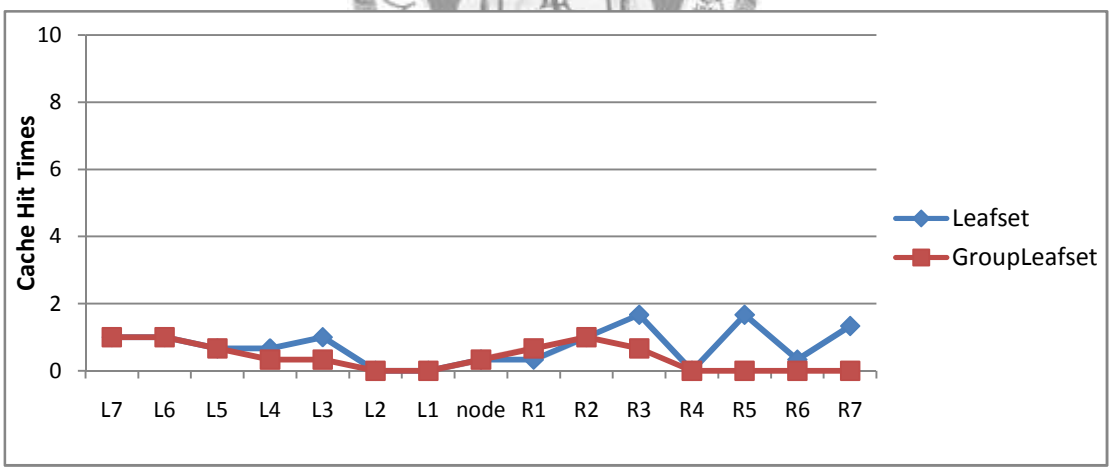


圖 3-8：Compare Average Cache Hit of Leafset and GroupLeafset in RSR0 pattern

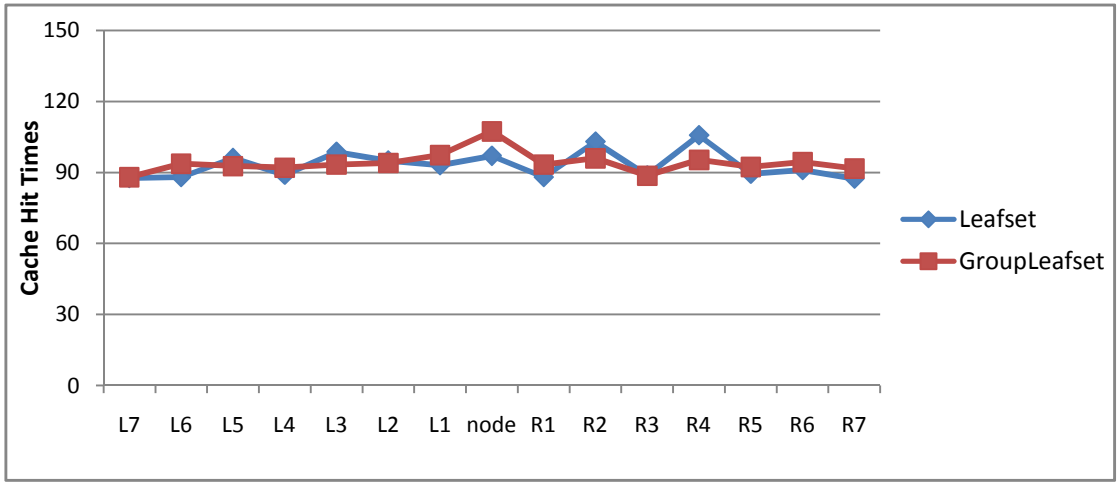


圖 3-9：Compare Average Cache Hit of Leafset and GroupLeafset in RSZO pattern

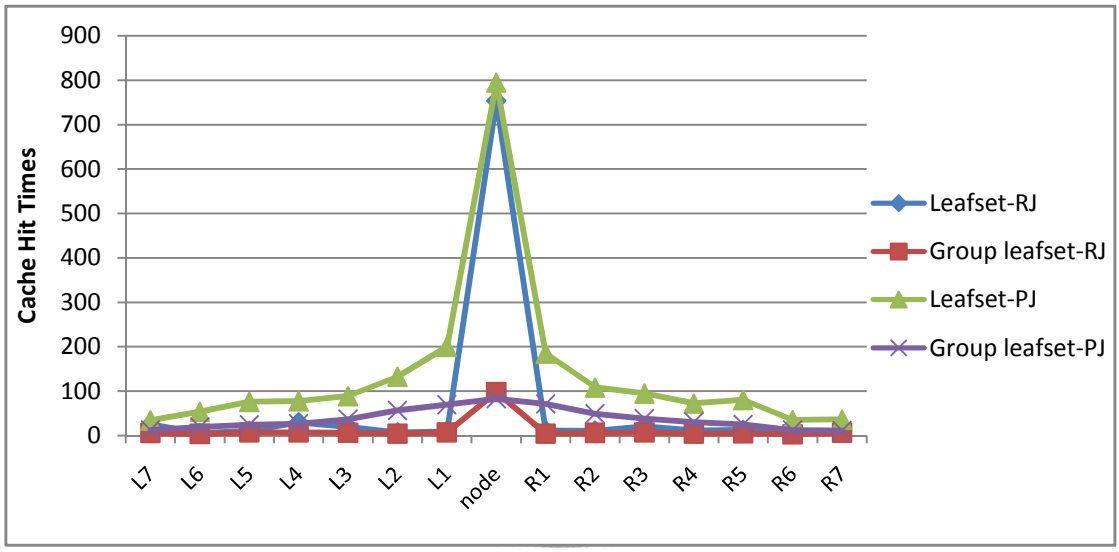


圖 3-10：Compare Average Cache Hit of Leafset and GroupLeafset in LSZO pattern

快取取代策略的差異

LRU 與 LFU 是最常見的快取取代策略，前者會移除快取空間裡最近最少被使用的物件，衡量依據物件在 Queue 中的先後順序；後者是移除快取物件中被存取次數最少的物件，依照每個物件計數器的值來衡量何者該移除；前文的預備實驗使用的快取移除策略是 LRU，我們另外做了 LFU 的實驗來比較，圖 3-11~圖 3-13 是比較在 LRU 與 LFU 設定下節點本身的鄰居列表的快取物件分布狀況，在 RSRO 搜尋樣式底下，分布情形一樣是沒有規則的曲線，快取擊中次數亦同樣低；RSZO 搜尋樣式底下則是快取擊中次數雙雙提高在相似的範圍區間裡，分布情形一樣是

沒有規則；在 LSZO 則是多了地域性因素，分布情形以發搜尋的節點為中心向兩側鄰居遞減，LRU 與 LFU 的曲線幾乎重疊，因此我們知道在 Donuts 之上做快取時，兩種主流的快取取代策略造成影響的差異很小，在本論文之後的實驗裡，我們皆選用最多人使用的 LRU 快取策略。

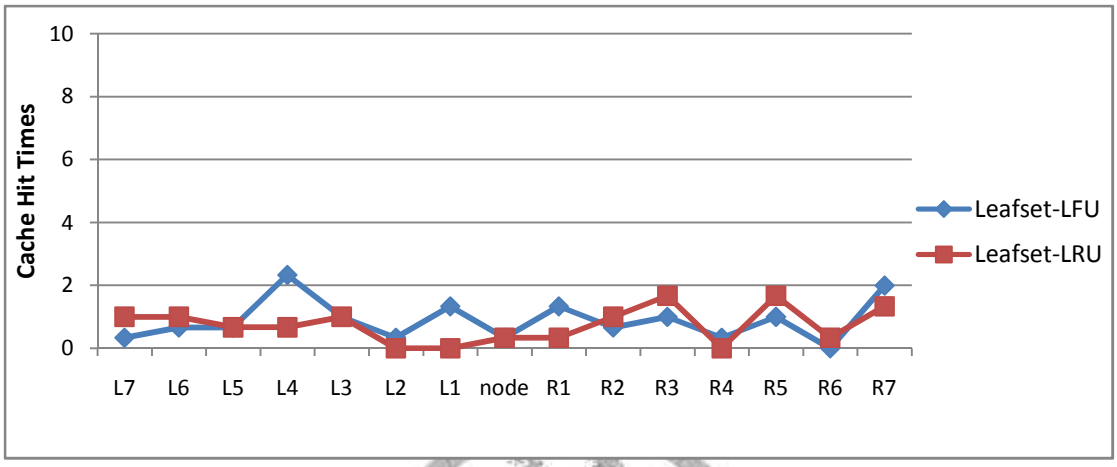


圖 3-11：Compare Average Cache Hit of Leafset in RSR0 pattern

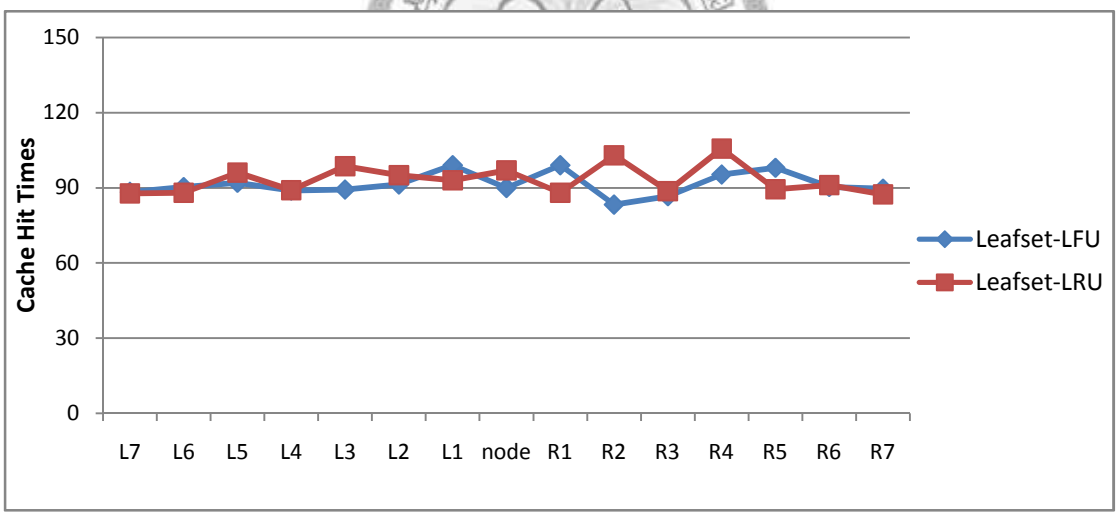


圖 3-12：Compare Average Cache Hit of Leafset in RSZ0 pattern

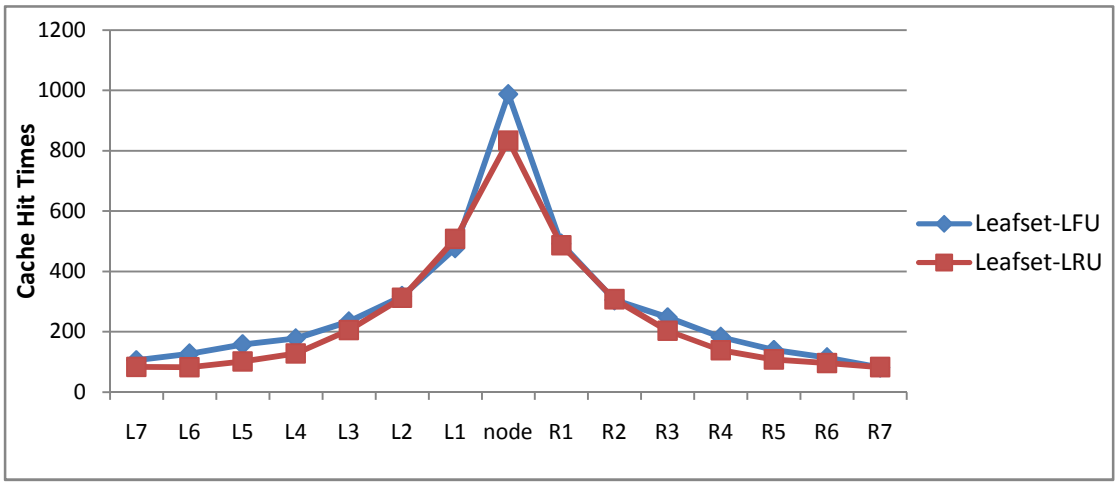


圖 3-13：Compare Average Cache Hit of Leafset in LSZO pattern

3.7.3.2 合作式快取加沿路快取

由前文得知在 LSZO 搜尋樣式下搜尋鄰居的快取找到欲搜尋物件的機率很高，而沿路快取是點對點網路中經常應用的快取策略，對受歡迎的檔案而言，沿著搜尋路徑放一份複製的物件可以讓檔案快速散布到整個系統之中，提高整體的搜尋效率，但沿路快取會造成大量的重複物件出現在節點的快取儲存中，快取快速存滿其它節點的複製物件達到飽和，而且沿路快取的物件回傳成本與搜尋成本相同，消耗很多網路頻寬；我們希望節點與左右鄰居做合作式快取，減少環狀結構上某一區段快取重複物件過高的情形，同時做沿路快取將受歡迎物件快速散播到環狀結構上的遠處。

除了沿路快取之外，我們讓節點與其左右相鄰的節點一起做合作式快取，在做搜尋繞路前除了先檢查自己的快取，也檢查左右鄰居的快取有無欲搜尋的物件；找到搜尋物件之後快取機制就被啟動了，合作式鄰居快取主要分成兩個步驟：快取進入許可、快取取代機制，快取進入許可決定把物件放到哪個節點的快取儲存當中，若快取達到飽和又想加入新物件則啟動快取取代機制，決定移除哪一個舊物件讓新物件加入。

Algorithm *CooperativeNeighborSearch(key)*

1. **if** *key* ∈ *n.range*
2. **if** *key exists in the set of objects hosted by n*
3. **then return** *object with the key*
4. **else** *NOT_FOUND*
5. **else**
6. **if** *n.rightLeafset[0].cache contains key* **or** *n.leftLeafset[0].cache contains key*
7. **then return** *object with the key*
8. **else for** *i = n.PT.height* **down to** *0*
9. **do if** *key > interval*
10. **then if** *key* ≥ *n.PT.next[i].interval*
11. **then** *n.PT.next[i].Search(key)*
12. **else**
13. **if** *key* ≤ *n.PT.prev[i].interval*
14. **then** *n.PT.prev[i].Search(key)*

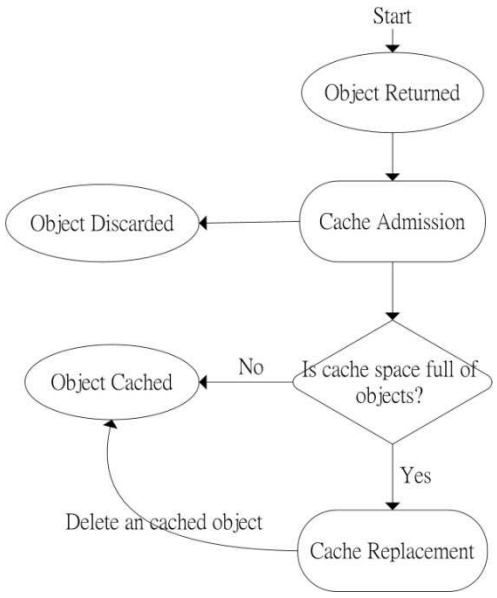


圖 3-14：Cache State Diagram

第四章 實驗結果

在這個章節，首先說明實驗模擬環境的建立方法，包含基礎的實體網路來源、動態環境模擬與 Donuts 系統建構到可以做搜尋的過程。我們做各種實驗想找出最適合 Donuts 特性的搜尋方式，首先評估不同系統資料結構在不同搜尋樣式底下的搜尋效能，接著比較傳統的本地端快取與沿路快取對搜尋效能的改善，再由地域性來源與 Zipf 物件特性搭配 Donuts 特有的鄰近性環狀結構，評估可能的搜尋方式，進而比較合作式快取與傳統快取的差異，找出最適合 Donuts 的搜尋方式與快取策略。

4.1 模擬方法

實驗用的模擬器是以 Java 語言撰寫而成，並在裝有 JVM 6.0 的電腦上做單機模擬的實驗；模擬器的設計架構採用單程序-事件驅動模型來模擬節點在特定時間點的行為。以下依序說明實體網路結構如何產生、以及如何模擬節點在系統中的各種行為。

4.1.1 實體網路

我們利用實體網路產生器 Brite[27]來產生類似 Internet 環境的實體網路結構，Donuts 再以實體網路結構為基礎去建立自己的系統環狀結構。Brite 支援各式種類的實體網路架構，根據[38]從實際網路數據的研究中得知，Internet 環境具有 power-law 的現象，而 Internet 是由數個自治系統所組合成，因此我們選用 Brite 提供的 Top-Down 模型(如圖 4-1 所示)來建立兩階層式的實體網路架構，第一階層是自治系統(Autonomous System)，第二階層為路由系統，網路的建構方法是先建立好第一階層的自治系統環境,才逐一建立每個自治系統底下的路由系統。節點與節點的連結機制我們採用最小邊數模型，每一個連結所分配到的對內與對外頻寬則是根據長尾分配來指派;節點在地理空間的配置方式也是根據長尾分配，綜合各種參數設定皆是以造出一個最接近 Internet 環境為目標。　實體網路中

任意兩個節點的溝通成本在本論文的實驗裡只考慮傳播延遲(Propagation Delay)；我們假設整個實體網路是穩固不壞的，如圖 4-1 中的節點 S 與 D 的溝通成本是計算兩節點間的最短路徑的傳播延遲，以 d(S,D)代表最短路徑長。

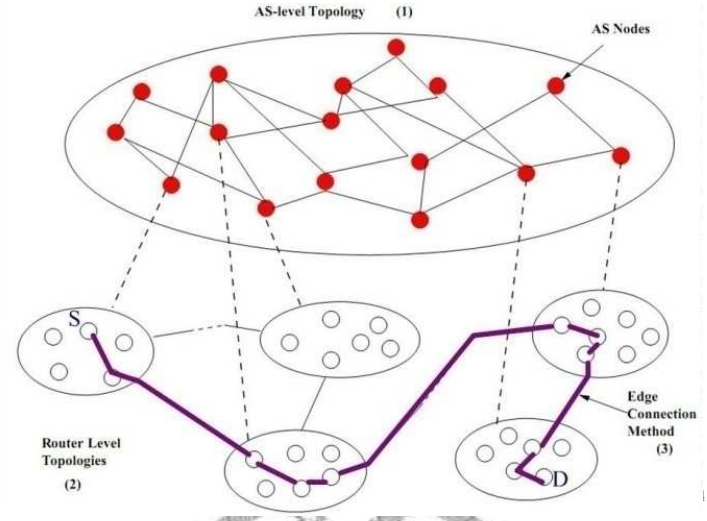


圖 4-1：Top-down model

4.1.2 動態環境模擬

在動態網路環境中，節點的加入與離線會改變部分環狀結構的成員相對關係，我們希望不管在環狀結構的建構過程或是維護階段都能保持整體結構不至於有斷裂的情形發生。在建立環狀結構的階段，一開始加入第一個節點到系統中時，環狀結構上的成員只有一個節點，也只有第一階層的左右指標，且皆指向自己；當第二個節點欲加入時，根據加入演算法決定新加入的節點在環狀結構中的位置，以插隊的方式加入到已在環狀結構上的成員中間，並且新節點設定自己第一階層的指標指向左右兩邊的節點，同時也通知左右兩邊的節點調整第一階層的指標，之後新節點開始建構自己的繞路表，如同第三章所描述。當節點的離線使繞路表過時，利用定期修正每個節點的繞路表來解決因節點離線或節點非預期錯誤造成的問題。

為了模擬動態的節點行為，我們利用實際的資料[38]來分配節點的上線時間

[4] 圖 4-1 來源: A. Medina, A. Lakhina, I. Matta, and J. Byers. BRITE: Universal Topology Generation from a User's Perspective

(session time)，當節點加入到 Donuts 時，我們就為它分配一個上線時間，當上線時間倒數到零該節點就離線，[38]觀察實際 Napster 與 Gnutella 網路資料，發現上線時間的累積分布函數(CDF)如圖 4-2 所示。圖 4-2 中的觀測時間是 720 分鐘，我們令 T 代表觀測時間區間，每段上線時間則是 T 乘上對應的百分比。利用[38]的資料我們可以計算出上線時間的機率分布函數(PDF)，當一個節點加入系統中，根據上線時間的機率分布函數得到一個機率值 δ，依照模擬器需要模擬的時間可計算出節點停留在系統中的時間。依據圖 4-2 的累積分布函數可計算出平均上線時間為 0.195T，在我們的模擬實驗裡，定義節點的平均上線時間為 36000 個時間單位，模擬器裡每推進一個時間單位代表時間經過 6 秒鐘，平均上線時間則為 60 小時，模擬器可依照實驗的需求去更動模擬的總時間單位，分配的方式仍是以同樣的機率分布函數去配置。

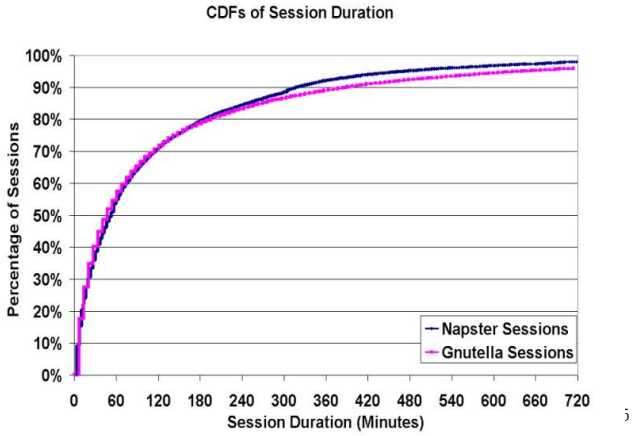


圖 4-2：CDF of system session lengths

在每一次的模擬當中，必須建立並維護一個大小為 N 的環狀結構。首先我們讓前 100 個節點以每個時間單位加入一個節點的方式依序加入到系統中，為了讓剛開始的基礎環狀結構保持高度完整性，在這 100 個節點的加入過程中不允許有節點離線；待大小為 100 的基礎環狀結構完成後，開始分配每個存在系統中的節點的上線時間，並從時間單位 100 時開始倒扣上線時間，當上線時間歸零時該節

[5] 圖 4-2 來源: S. Saroiu, P. K. Gummadi, S. D. Gribble, “A Measurement Study of Peer-to-Peer File Sharing Systems”

點則離線，為了保持系統的穩固性，此時必須從離線的節點當中隨機挑選一個節點重新加入到系統中，以維持系統中節點總數的平衡；在時間單位 100 之後，每隔兩個時間單位加入一個新節點，若我們預計 Donuts 大約是 N = 5000 的環狀結構，在時間單位 9900 時系統保持約有 5000 個節點的狀態，圖 4-3 是模擬器整個模擬過程的示意圖。

物件的加入緊接在環狀結構建立完成之後，模擬器會從等待加入的物件集合當中隨機挑選一個物件加入到系統中，物件集合是來自韋伯字典的英文字串，字串總數為 235,118，每一個英文字代表一個物件，物件之間的相對關係則依照字母順序由A排列至Z,而每一個物件經過運算會得到一個不與其他物件重複的Key值，物件的加入方法即是找到系統中負責保管此物件的節點，也就是節點的物件範圍區間涵蓋 Key 值，則把該物件放置到此節點上，接著依照複製物件個數 γ 的參數設定，當 γ 設為 2，物件會各複製一份到所屬節點的左右兩邊節點上。當物件開始加入到系統中時，每個節點所負責的物件範圍區間皆相同，很明顯會有負載不平衡的問題產生，也就是有些區間的物件會特別多，例如分配到母音字母開頭的範圍區間，因此在加入物件的過程中，模擬器會一邊運作負載平衡演算法，鄰居平衡演算法的參數 α_{NB} 設為 1.2、節點重排演算法的參數 α_{PR} 設為 2.0，β_{NB} 與 β_{PR} 皆設為 20，參數的設定參考[29]，模擬器以 5000 個時間單位來完成所有物件的加入，因此在 14900 時間單位的時候所有物件加入完成。

在物件都加入完成且系統中的節點總數達到穩定狀態時，我們開始做搜尋的實驗。為了模擬搜尋的過程，我們必須挑選某些節點發出搜尋請求，挑選的方式

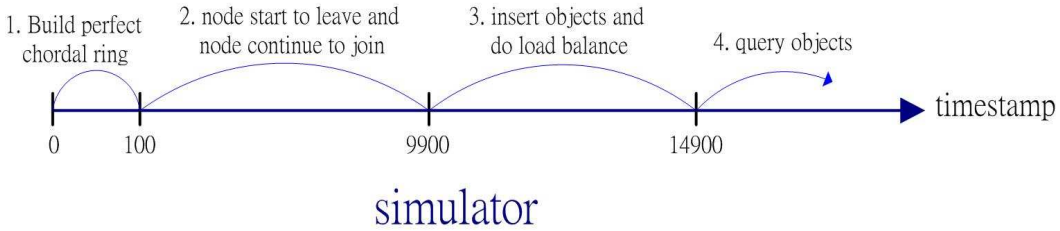


圖 4-3：Event-Driven Simulator

可隨機挑選、也可依照地域性選擇；至於被搜尋的物件由給定的詢問檔案中挑選目標物件，挑選的方式一樣可以隨機挑選，另外也能依照受歡迎的程度、以 Zipf 分配的方式挑選，搜尋樣式即是由發搜尋的節點與目標物件搭配而成，模擬器提供四種搜尋樣式的選擇，後文將做詳細介紹。

N	Donuts 中的節點總數	5000
L	Leafset 的大小	14
P	維護基礎環狀結構的時間間隔	30 秒
F	修正繞路資訊的時間間隔	180 秒
C	快取儲存區的大小	15
α_{NB}, β_{NB}	鄰居平衡演算法需要的參數	1.2,20
α_{PR}, β_{PR}	節點重排演算法需要的參數	2.0,20

表格 4-1：System Parameters

4.2 衡量指標

為了衡量搜尋的效能，我們定義搜尋所花費溝通成本的測量方法。假設 d 是計算實體網路裡兩點間最短路徑所花費的溝通成本，若存在節點 u 與 v，d(u,v)代表 u 到 v 該花費的最小溝通成本，d 的計算結果是固定且對稱的，d(u,v)與 d(v,u)的結果是一樣的。當環狀結構上的某一節點 S 發出搜尋請求，經過 h 次的跳躍搜尋最終在節點 D 找到目標物件，整個過程的搜尋成本計算方式是$\sum_{i=1}^{h} d_i(u,v)$，又最終將目標物件由 D 傳給 S 需要回傳成本 d(S,D)，每一個單獨的搜尋花費總成本即是搜尋成本加上回傳成本，如圖 4-4 所示。

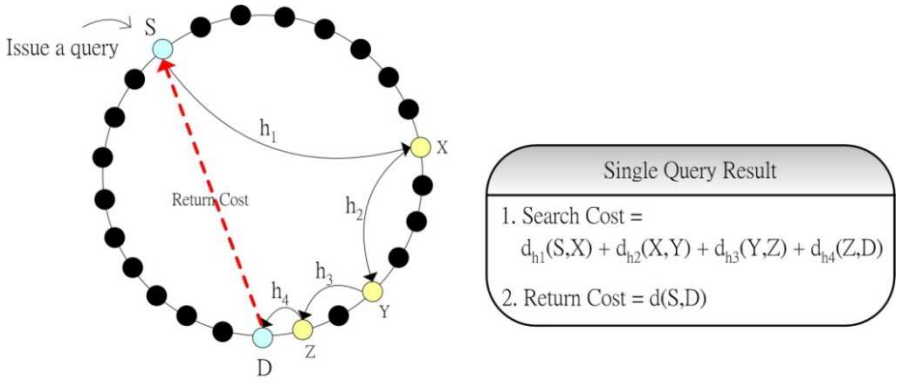


圖 4-4：Total Search Cost

我們令每一次的搜尋實驗發出 Q_n 個搜尋請求，平均搜尋成本即是

$$\text{Search Cost}_{avg} = \frac{\sum_{q=1}^{Q_n}\sum_{i=1}^{h_q} d_{q_i}(u,v)}{Q_n}$$

，而平均回傳成本則根據快取策略的不同有兩種回傳方式：若物件回傳方式是沿搜尋路徑傳遞，則平均回傳成本與平均搜尋成本相同；若從搜尋物件所在節點直接回傳，稱為平均直接回傳成本，計算方式如下

$$\text{Return Cost}_{avg} = \frac{\sum_{q=1}^{Q_n} d_q(S,D)}{Q_n}$$

，平均總成本是兩者相加：$\text{Total Cost}_{avg} = \text{Search Cost}_{avg} + \text{Return Cost}_{avg}$

若想計算平均每次搜尋在實體網路的搜尋路徑長，將 Search Cost_{avg} 除以實體網路的平均連結長即可。

$$\text{Physical Search Hops}_{avg} = \frac{\text{Search Cost}_{avg}}{\text{Physical Edge Length}_{avg}}$$

4.3 搜尋與快取的效能

為了評估 Donuts 支援搜尋的效能，我們設計了一些實驗模擬真實的整體點對點網路搜尋行為。在每一回合的搜尋實驗中會產生 5000 個搜尋請求，被搜尋的物件由一固定大小為 10000 個字的字串集合中挑選，為了保證搜尋的物件一定會在環狀結構上，這 10000 個英文字由韋伯字典 235,118 個字中隨機挑選出來，而韋伯字典所有的英文字皆按照字母順序依序分配到對應的環狀結構節點上。

模擬器除了原先支援的隨機搜尋樣式之外，我們額外增加了地域性來源與 Zipf 目標物件的組合。模擬器隨機挑選一個正在環狀結構上的節點當作發搜尋請求的開端，欲搜尋的物件也是以隨機的方式從 10000 個字串集合中挑選，這種搜尋樣式稱為隨機來源-隨機物件(RSRO)；若來源的選擇仍是隨機挑選，而物件的選法依照受歡迎的程度挑選之，越受歡迎的物件被搜尋的次數就越高，此種搜

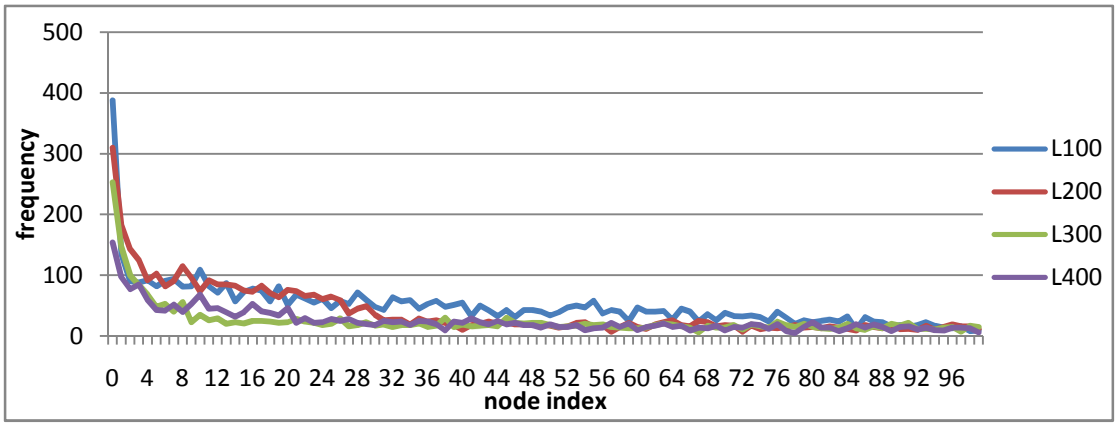


圖 4-5：Simulate Locality-Source by Bernoulli Model

尋樣式為隨機來源-Zipf 物件(RSZO)；當搜尋來源加入地域性的考量時，我們從地理上較相近的特定節點當中挑選發搜尋請求的來源，越近的節點被挑選到的次數會越多，而此時不考慮物件受歡迎的程度，這種搜尋樣式是地域性來源-隨機物件(LSRO)，反之若考慮物件受歡迎的程度，則是地域性來源-Zipf 物件(LSZO)。

地域性來源的模擬方式我們選用 Bernoulli 模型，首先隨機挑選一個節點當作地標，計算所有其他節點到此地標的實體最短距離，並以此距離當作 Bernoulli 模型的影響變量；搜尋來源會先由距離最小的節點開始嘗試挑選，若挑選失敗則嘗試挑選距離第二小的節點，依此類推直到挑選到為止。地域性區域依序為 100、200、300、400 個節點時，這些節點發搜尋請求的頻率如圖 4-5 所示，當地域性區域越小時，發大量搜尋請求的節點集中在少數相近的某些節點上，例如地域性區域為 100 時，離地標最近的前 30 個節點發出了 2580 個搜尋請求，占總搜尋請求的一半以上；地域性區域為 400 時，前 30 個相近的節點只發 1404 個搜尋請求，因此，當我們希望發搜尋請求的節點越集中在以鄰近性演算法建構的環狀結構上某個區段時，地域性區域的參數應該設越小越好。

模擬物件受歡迎的程度則是採用 Zipf 模型，我們將 N 個被挑選的可能物件以 Zipf 模型去分配物件受歡迎的程度，排名第 k 的物件被搜尋的頻率如下公式定義：

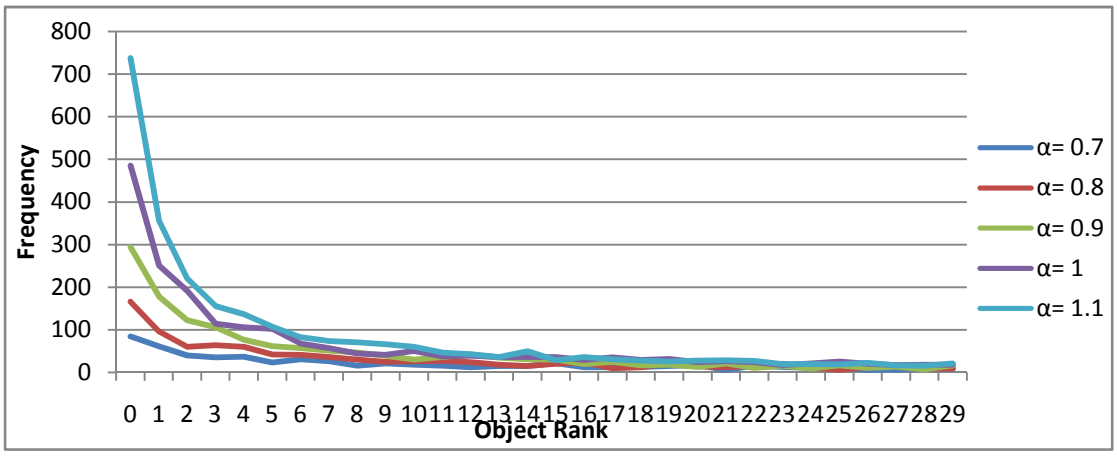


圖 4-6：Simulate Zipf-Object by Zipf Model

$$f(k,s,N) = \frac{1/k^s}{\sum_{n=1}^{N} 1/n^{\alpha}}$$ [6]

圖 4-6 是參數 α 分別為 0.7、0.8、0.9、1、1.1 時排名前 100 的物件被搜尋的頻率，物件個數 N 為 10000，很明顯能發現：被頻繁搜尋的物件數量非常稀少，大部份的物件被搜尋的次數皆不多；當 α 越小時，排名第一名的物件被搜尋的次數越少，例如 α=0.7 時，最受歡迎的物件只被搜尋了 84 次，若 α=1.1，最受歡迎的物件被搜尋了高達 734 次；當搜尋的過程搭配沿路快取時，若最受歡迎的物件被搜尋過多次，會造成排名高的物件存在大量節點的快取之中，造成空間的浪費；反之若受歡迎的物件被搜尋太少次，表示其它節點會搜尋它的機會不大，整體而言會造成快取的使用率較低，因此我們在評估快取空間重複率與使用率之下取折衷的參數—以 α=1.0 當作之後模擬物件受歡迎程度的參數設定。

四種搜尋樣式裡，LSZO 是最接近實際點對點網路的搜尋樣式，考慮了節點的地域性因素，也考慮了物件受歡迎的程度，當 Donuts 搭配鄰近性加入演算法時，沿路快取會讓快取物件大部分聚集在環狀結構上的某段區間上(參考 3-7)，圖 4-7 是在不同地域性個數設定下，節點本身與鄰居列表的快取物件分布狀態，當地域性區域越小時，分布曲線越陡峭；地域性區域越大時，分布曲線越趨近平緩，集

[6] 來源：http://en.wikipedia.org/wiki/Zipf's_law　Zipf's law can be approximated with a Zipfian distribution, one of a family of related discrete power law probability distributions.

中的現象較無地域性區域小時來的明顯。圖 4-8 則是在不同 Zipf 參數設定下，快取物件在節點與其鄰居列表的分布曲線，我們發現參數越大(α =1.1 時)，分布的現象最為集中在節點左右，參數越小(α =0.7 時)分布曲線則越趨近平緩，集中的現象較不明顯。

綜合上述不同設定對快取分布的影響，接下來的實驗則依照對地域性區域大小的不同要求給予調整，在討論系統資料結構的影響時，地域性因素設為所有節點總數的十分之一(LS =500)，而在討論快取的實驗裡，我們希望提高地域性因素的影響，故將地域性區域設為 100，讓快取物件分布較為集中。而 Zipf 參數我們選用 α =1.0，由圖 4-8 可發現，α =1.0 的分布曲線在鄰居列表的出現頻率最高，表示鄰居列表的快取物件使用率會是最高的。

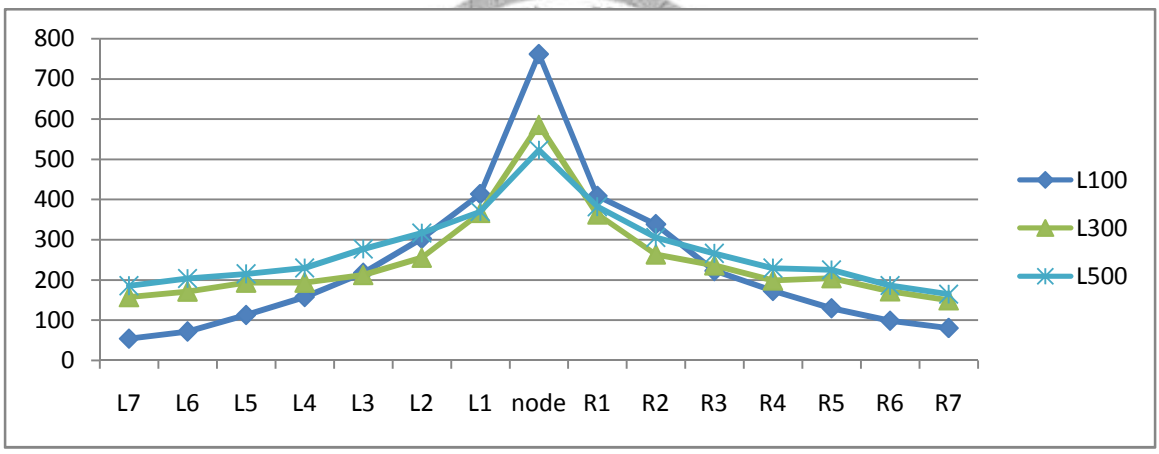


圖 4-7：Average Cache Hit of Leafset under different locality size

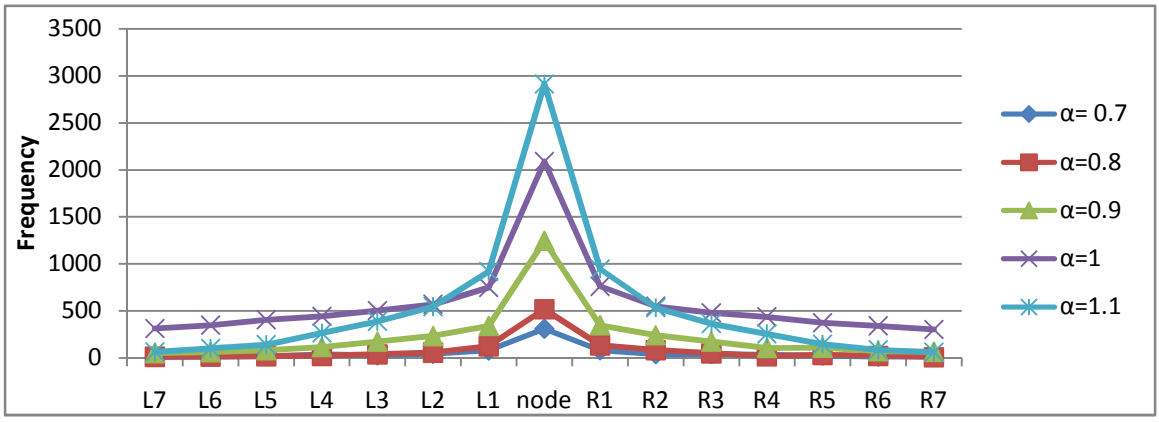


圖 4-8：Average Cache Hit of Leafset under different Zipf parameter

4.3.1 系統資料結構的影響

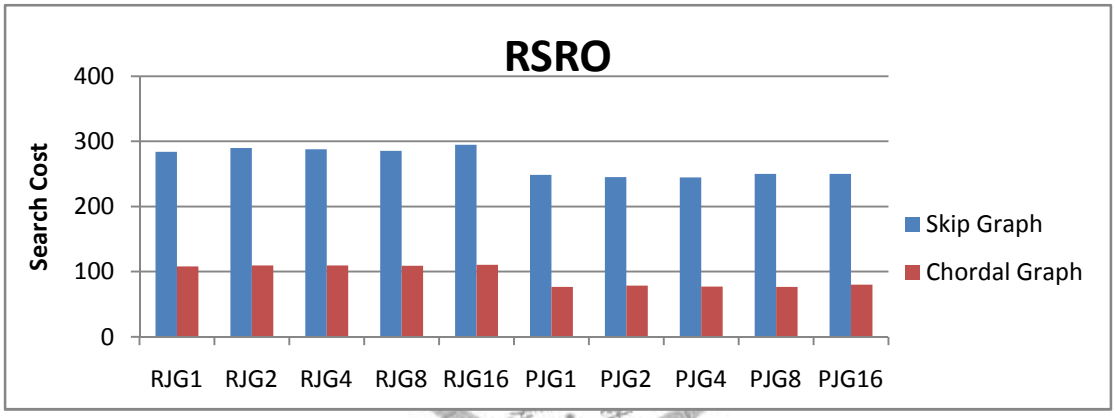
Chordal Ring 與 Skip Graph 都是適合做範圍查詢的系統資料結構，在第一個實驗裡，首先檢測兩種系統資料結構在搜尋效能上的差異。Chordal Ring 的搜尋是使用鄰近表的繞路資訊，Skip Graph 的搜尋則是查閱快捷表，在搜尋的過程中不做任何快取處理；我們搭配前文提到的四種搜尋樣式、群組數與加入演算法的不同來比較搜尋成本的差異，詳細數據如圖 4-5 所示，其中地域性的節點個數我們設定為 500 個彼此相近的節點，實驗用一實體網路結構，在 RSRO 搜尋樣式底下我們做了九個回合，我們發現 Skip Graph 花費的搜尋成本明顯比 Chordal Ring 所需的搜尋成本高出甚多，礙於時間因素，另三種搜尋樣式我們僅各做三個回合。Skip Graph 的搜尋成本在任何一種搜尋樣式底下皆比 Chordal Ring 高出甚多，Chordal Ring 花費的搜尋成本平均比 Skip Graph 的三分之一還少。

若比較不同搜尋樣式下搜尋成本的差異，我們發現 Chordal Ring 和 Skip Graph 皆在 LSRO 搜尋樣式下花費最高的搜尋成本，其餘三種搜尋樣式的差異則較不明顯，由於地域性來源侷限於環狀結構上的某些節點，而搜尋物件隨機選擇，使平均搜尋路徑比其他三種搜尋樣式長，因此搜尋成本較高；其中 LSZO 是最接近真實點對點網路的搜尋樣式：發搜尋請求的節點有地緣性關係，相近的節點有興趣的物件也是類似的，因此在 LSZO 搜尋樣式下所花費的搜尋成本與實際點對點網路的情形較為相近。

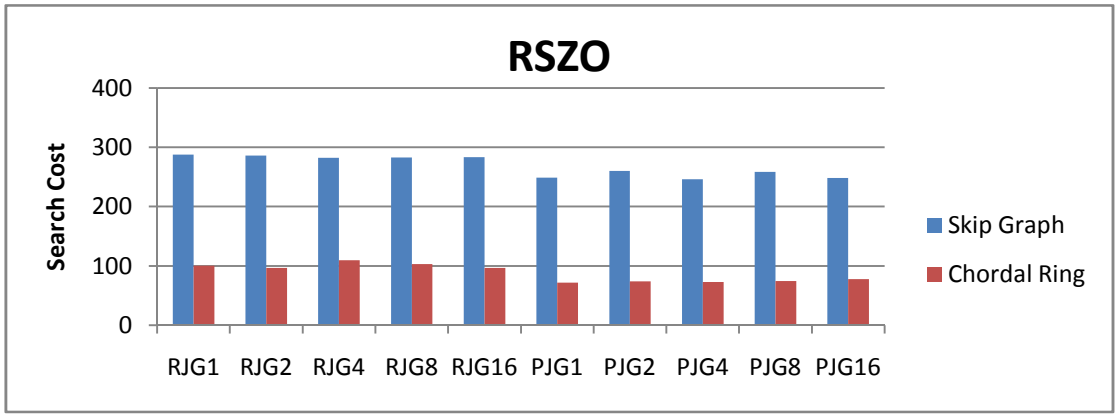
至於加入演算法的影響，兩者的建構方式採用鄰近性加入演算法時，由於系統資料結構反應了實際網路情形，使得個別的搜尋成本比隨機加入演算法平均減少 21.78%與 13.59%，可見採用鄰近性加入演算法建立的環狀結構較適合用來做搜尋。群組數的不同所造成的差異較不明顯，G=1 時表示並未分群，搜尋成本比有分群的情形稍小，由於分群會使實體距離相近的一群節點平均分配到數個群組當中，也就是配置到環狀結構上的數個區段之上，分群會稍微破壞鄰近性的完整度，造成搜尋成本上升。回傳成本在兩種系統資料結構上的差異僅在搜尋樣式採用 LSRO

時有明顯的不同，當發搜尋請求的節點侷限於環狀結構上某一區段的節點、而搜尋物件的範圍是整體環狀結構時，除非搜尋物件剛好挑中位於發搜尋請求的區段節點，平均須付出較昂貴的回傳成本，無論系統資料結構採用 Chordal Ring 或 Skip Graph 皆是。

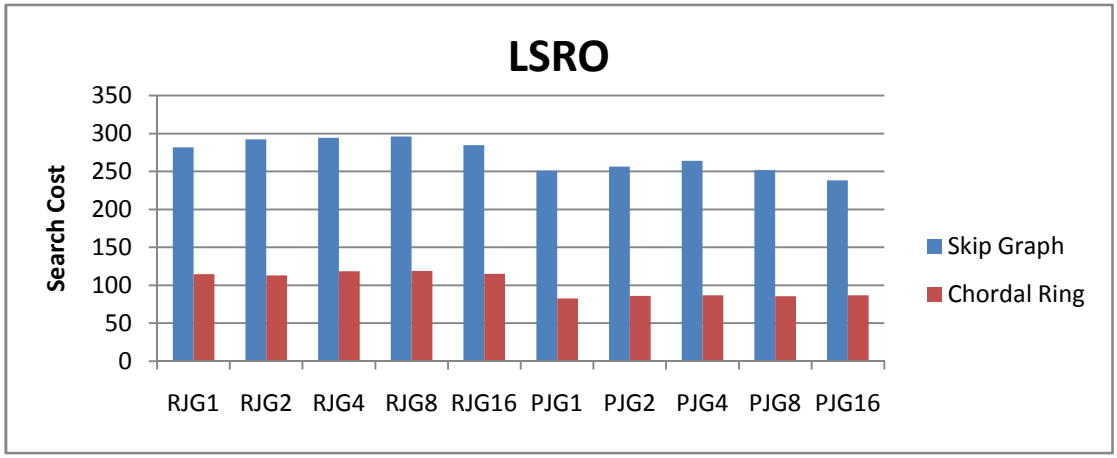
Chordal Ring 的繞路搜尋使用鄰近表，Skip Graph 則是使用快捷表，由於鄰近表每一階層的指標皆是指向該階層最近的節點，快捷表的指標則沒有這個特色，因此我們得到一個結論：利用 Chordal Ring 搭配鄰近性加入演算法產生的系統結構來做範圍搜尋，可以最小化整體的搜尋成本，之後的實驗皆以 Chordal Ring 為系統資料結構。



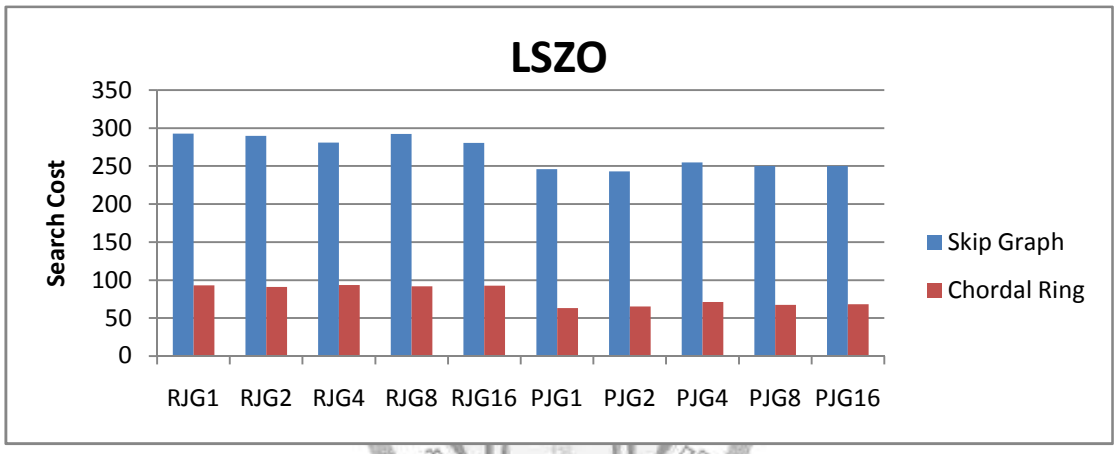
(a) Query Pattern：Random Source-Random Object



(b) Query Pattern：Random Source-Zipf Object



(c) Query Pattern：Locality Source-Random Object



(d) Query Pattern：Locality Source-Zipf Object

圖 4-9：Compare Average Search Cost under Different Query Patterns

Chordal Ring	RSRO	RSZO	LSRO	LSZO
RJG1	32.68	33.44	42.86	33.28
RJG2	33.11	32.28	42.12	33.36
RJG4	32.79	33.3	42.24	32.5
RJG8	33.19	33	41.89	33.58
RJG16	33.18	33.08	43.27	34.6
PJG1	33.5	32.66	43.23	33.63
PJG2	33.42	33.97	43.41	32.73
PJG4	33.84	33.27	43.69	32.75
PJG8	33.11	33.11	43.64	32.03
PJG16	33.74	33.33	41.99	34.16
Average	**33.26**	**33.14**	**42.83**	**33.26**

Skip Graph	RSRO	RSZO	LSRO	LSZO
RJG1	32.95	33.17	32.57	34.72
RJG2	33	32.9	29.57	29.07
RJG4	33.32	33.83	30.57	30.04
RJG8	33.23	34.19	34.21	37.43
RJG16	33.17	33.8	34.67	35.28
PJG1	33.63	33.91	34.46	35.27
PJG2	33.58	33.67	35.59	34.61
PJG4	33.8	33.06	40.32	29.21
PJG8	33.15	33.79	31.01	26.19
PJG16	34.15	33.47	31.33	30.71
Average	**33.4**	**33.58**	**33.43**	**32.25**

表格 4-3：Return Cost in 4 Query Patterns

4.3.2 快取空間對搜尋效能的影響

首先我們討論快取空間大小對快取效能的影響。在 LSZO 搜尋樣式底下，搜尋方式採用鄰近表繞路搜尋，搭配沿路快取策略，地域性區域大小設為 100，Zipf 參數為 1.0，我們用三個實體網路做實驗，每種設定皆做三個回合；圖 4-10 是快取空間大小分別為 15、30、45 時的搜尋成本，當快取空間越大，搜尋成本越小，可見在 LSZO 搜尋樣式底下，放大快取空間能換取搜尋效能的改善，而群組數越小時，鄰近性的特性越明顯，讓地域性搜尋者來源在環狀結構上越集中，快取的效能越明顯。

當快取空間變大，可以提高本地端快取的擊中次數(圖 4-11)，而這是犧牲更多的記憶體空間換來的效能改善，對節點本身會造成更多的負荷；我們希望在快取空間較小的情形下尋求搜尋效能的改善，因此之後的實驗將快取空間大小訂為 15。

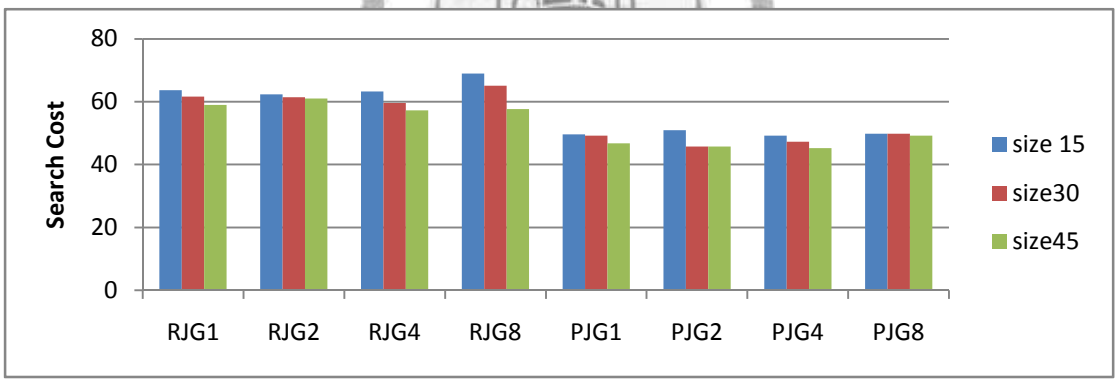


圖 4-10：Compare Search Cost under Different Cache Size

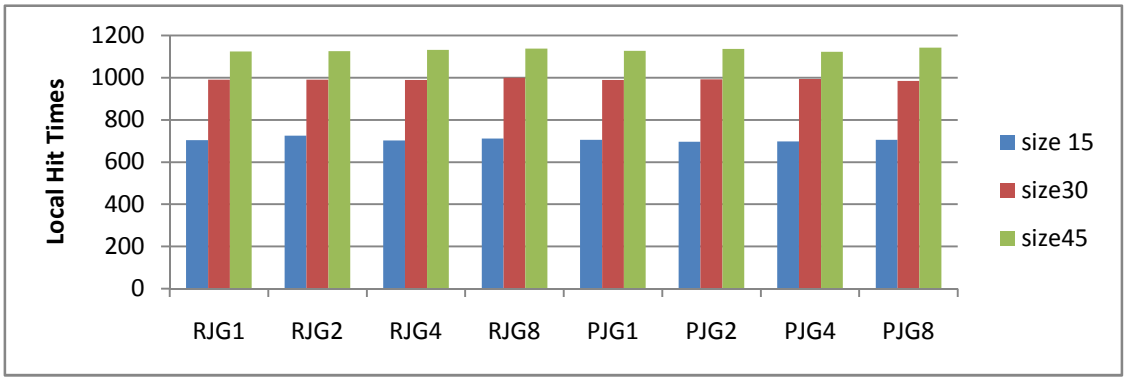


圖 4-11：Compare Local Hit Times under Different Cache Size

4.3.3 傳統快取對搜尋鄰近表的影響

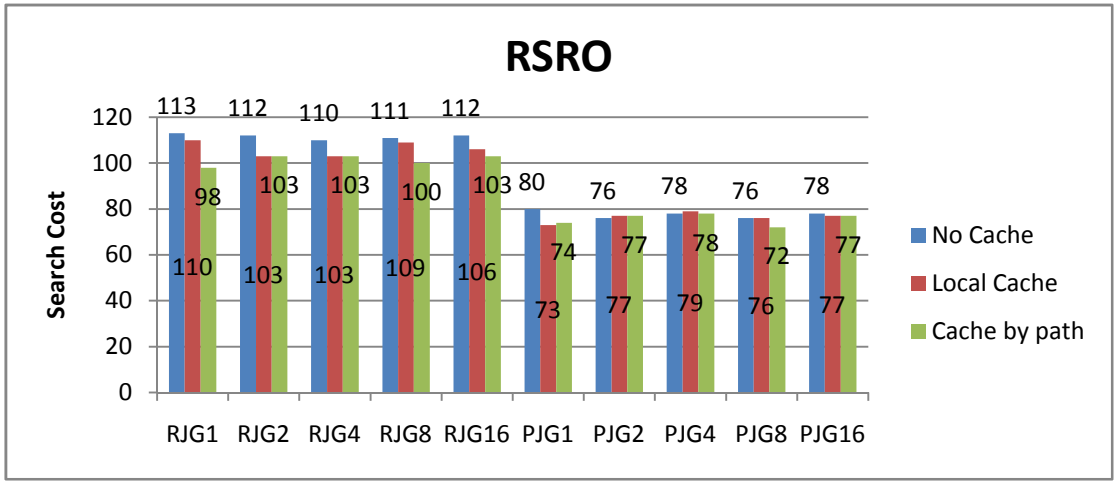
在 Chordal Graph 的系統資料結構下，在做搜尋時我們查閱鄰近表的繞路資訊，並搭配不同的節點加入演算法與不同的搜尋樣式，來比較沒有快取、僅做本地端快取和沿路快取三種快取方式的搜尋成本，其中地域性區域大小設為 100，Zipf 參數設為 1，快取取代方式是 LRU、快取空間大小是 15，我們的實驗用三個實體網路模型，每個模型做三個回合的搜尋實驗。

在 RSRO 搜尋樣式下(圖 4-12(a))，做本地端快取與沿路快取的搜尋成本與沒做快取的搜尋成本差異均不大，尤其是搭配鄰近性加入演算法時，無論群組數的多寡，本地端快取與沒做快取的搜尋成本幾乎相同，表示在 RSRO 搜尋樣式下，本地端快取對改善搜尋效能無益，沿路快取對搜尋效能的改善也無明顯幫助，由於發搜尋的節點是隨機挑選，搜尋的物件也是隨機挑選，即使搭配快取，放到快取中的物件在日後無論是節點自己重複搜尋或是被其他節點搜尋的機會皆很小，因此快取無法發揮效用。在 RSZO 搜尋樣式下(圖 4-12(b))，採用隨機加入演算法時，本地端快取平均減少了 6.9%的搜尋成本，沿路快取則大幅減少了 30.9%的搜尋成本；在鄰近性加入演算法時，本地端快取的搜尋成本幾乎與沒做快取時一樣，沿路快取則減少約 23.5%的搜尋成本；整體而言，採用 RSZO 搜尋樣式比 RSRO 搜尋樣式的搜尋成本較小，可見搜尋物件有受歡迎程度之分時，本地端快取與沿路快取可提供不同程度的搜尋效能改善，但由於是隨機選擇節點發搜尋請求，節點發出重複搜尋請求的機會較低，連帶降低快取空間的使用率，節點在自己的快取裡找到物件的機會也隨之降低，搭配本地端快取策略只能減少一點點搜尋成本，若搭配沿路快取則減少較多的搜尋成本，可見快速將受歡迎檔案散布到整個系統結構中能有效減低平均搜尋成本。

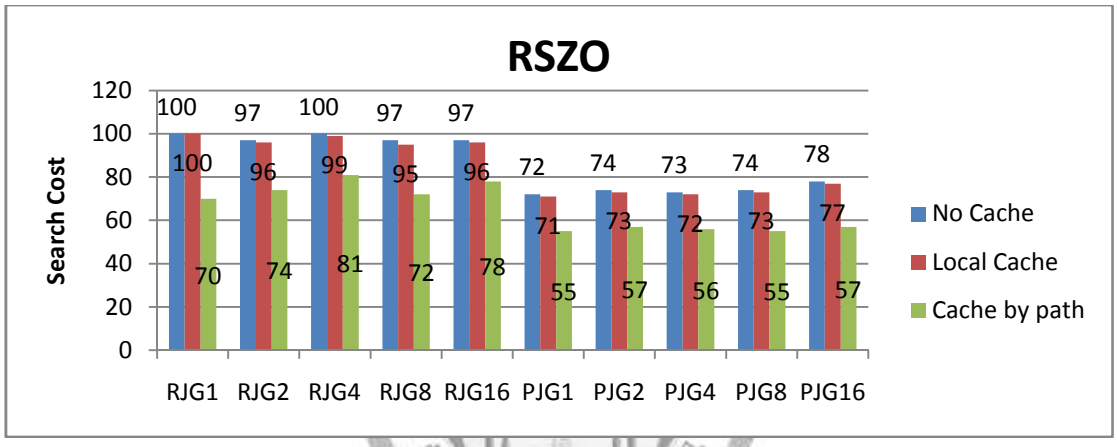
在 LSRO 搜尋樣式下(圖 4-12(c))，地域性區域限定了發搜尋請求的節點，當採用隨機加入演算法時，這些節點平均散布在環狀結構上，反之鄰近性加入演算法則大部分散布在環狀結構的某一區段上，當搜尋物件是隨機從整個環狀結構上

的物件挑選時，搜尋成本與 RSRO 的情形一樣—快取無法改善搜尋效能，反而因為加入地域性限制使搜尋成本比 RSRO 時還多，由於限定了發搜尋請求的節點，使環狀結構上的平均搜尋路徑較長，平均回傳成本也較高，整體的搜尋總成本也高於其他三種搜尋樣式。在 LSZO 搜尋樣式底下的快取效益較為明顯，本地端快取平均減少了 14.8%的搜尋成本，沿路快取則是大幅減少了 34.6%的搜尋成本；LSZO 搜尋樣式所花費的搜尋成本也比其他三種搜尋樣式還少。LSZO 是最接近一般實際點對點網路的搜尋樣式，發搜尋的節點之間有地域性的分布，且搜尋的物件也有受歡迎程度之分；LSZO 搭配 Donuts 特有的鄰近性節點分布，搜尋成本比隨機加入的方式更少，因此我們可推論 Donuts 適合一般點對點網路的範圍搜尋。

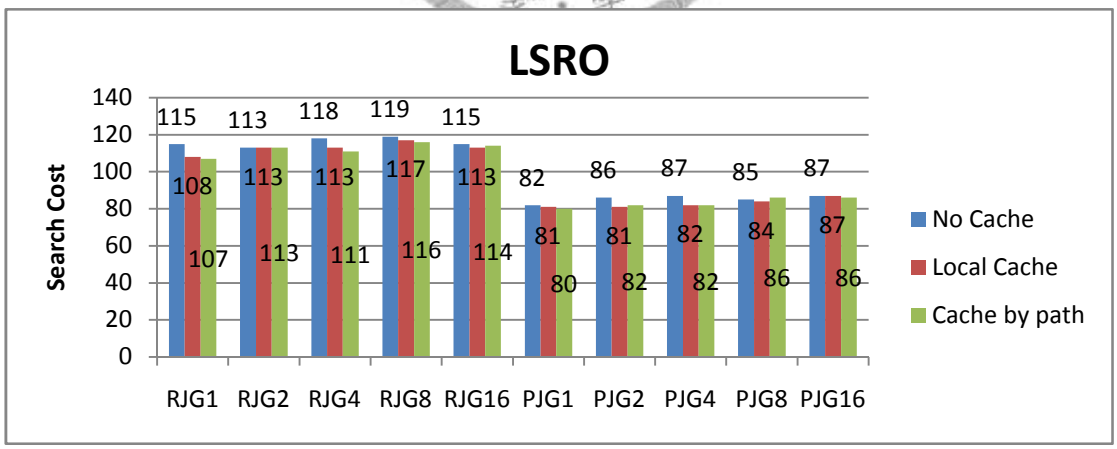
綜合上述的分析並搭配表 4-3 回傳成本的結果，我們發現只有在 LSZO 搜尋樣式之下回傳成本才有明顯的減少，與沒做快取時相比，LSZO 搭配本地端快取減少了 18.1%回傳成本，可見只有在 LSZO 兩種搜尋樣式下，本地端快取才有明顯的幫助。圖 4-14 分別是在四種搜尋樣式下的總搜尋成本，沿路快取的回傳成本與搜尋成本相同，由此圖明顯看出快取對搜尋效能的改善情形，本地端快取平均減少 16.3%的總搜尋成本，而沿路快取在 RSRO 與 LSRO 下的總搜尋成本皆非常高，在 RSZO 與 LSZO 下則較低，特別是在 LSZO 搜尋樣式搭配鄰近性加入演算法時，總搜尋成本比沒做快取時還低，因此我們推論若要做沿路快取，一定要搜尋物件有受歡迎程度之分，並且搭配鄰近性加入演算法，才能得出較為經濟的平均搜尋成本。沿路快取由於受歡迎的物件被放到很多節點的快取儲存區裡，節點可能在搜尋這些受歡迎的物件時發現本地端的快取已存有一份，也可能在查閱鄰近表繞路時中途即找到物件而減少跳躍搜尋的次數，不管是哪種情形都能減少搜尋成本與回傳成本。我們發現在每種搜尋樣式與不同節點加入演算法之下，本地端快取的總搜尋成本都是最低的，在 LSZO 搜尋樣式底下的總搜尋成本是四種搜尋樣式中最低的，可見在較接近一般具有地域性與物件受歡迎程度之分的搜尋行為下，本地端快取是最經濟的快取策略。



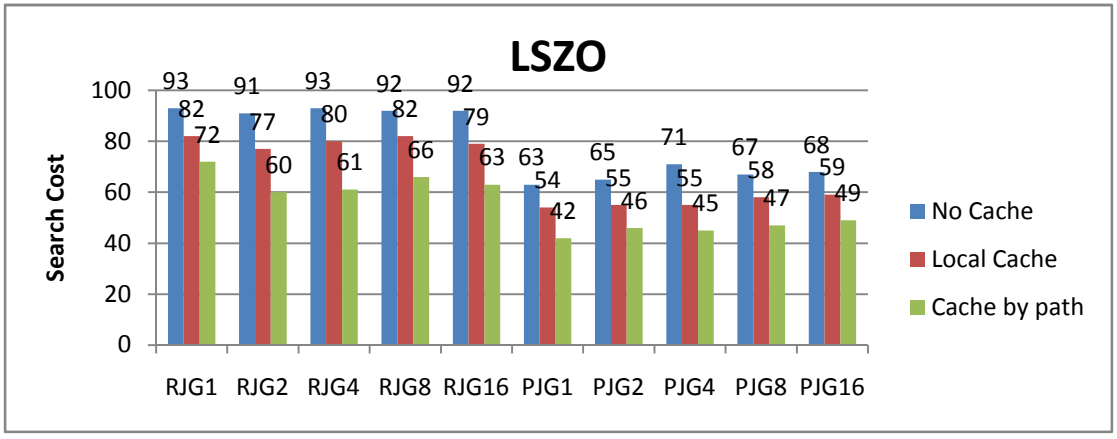
(a) Query Pattern：Random Source-Random Object



(b) Query Pattern：Random Source-Zipf Object



(c) Query Pattern：Locality Source-Random Object



(d) Query Pattern：Locality Source-Zipf Object

圖 4-12：Compare Search Cost under Different Query Patterns and Cache Methods

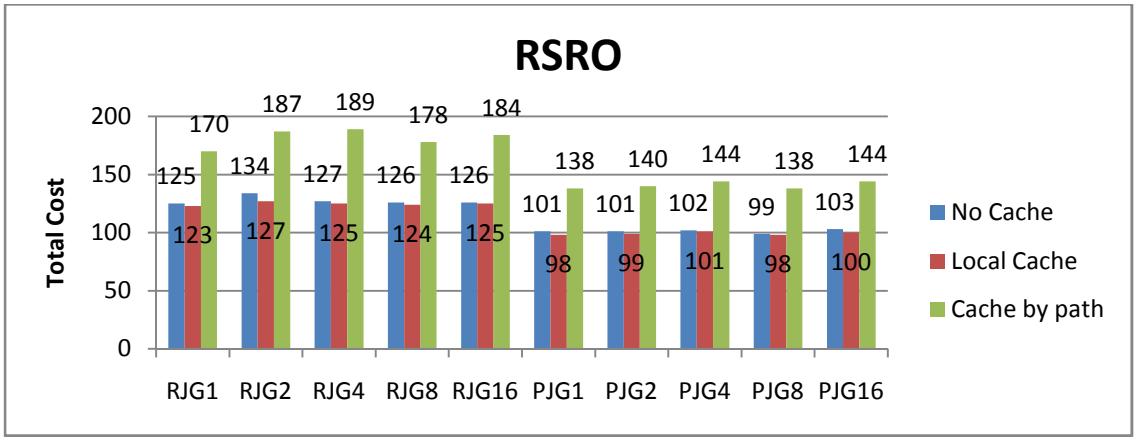
x 軸-RJ=Random Join, PJ=Proximity Join, G=Group Number

y 軸-Average Search Cost

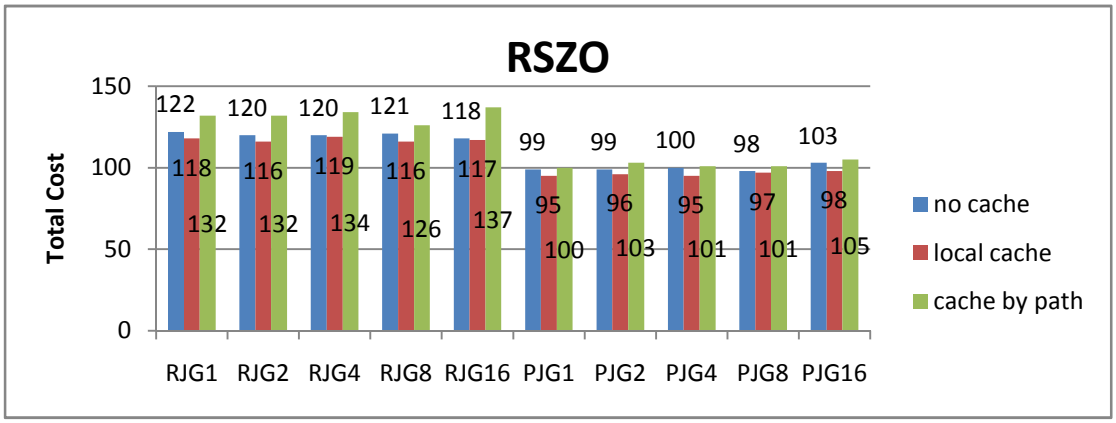
No Cache	RSRO	RSZO	LSRO	LSZO
RJG1	32.68	33.44	42.86	33.28
RJG2	33.11	32.28	42.12	33.36
RJG4	32.79	33.3	42.24	32.5
RJG8	33.19	33	41.89	33.58
RJG16	33.18	33.08	43.27	34.6
PJG1	33.5	32.66	43.23	33.63
PJG2	33.42	33.97	43.41	32.73
PJG4	33.84	33.27	43.69	32.75
PJG8	33.11	33.11	43.64	32.03
PJG16	33.74	33.33	41.99	34.16
Average	**33.26**	**33.14**	**42.83**	**33.26**

Local Cache	RSRO	RSZO	LSRO	LSZO
RJG1	32.62	32.76	42.8	27.48
RJG2	33.11	31.59	42.02	27.18
RJG4	32.81	32.81	42.15	26.66
RJG8	33.26	32.44	41.87	27.67
RJG16	33.19	32.46	43.27	28.63
PJG1	33.58	32.14	43.22	27.66
PJG2	33.41	33.17	43.38	26.64
PJG4	33.83	32.63	43.66	26.36
PJG8	33.11	32.39	43.46	26.81
PJG16	33.77	32.58	41.92	27.31
Average	**33.27**	**32.5**	**42.78**	**27.24**

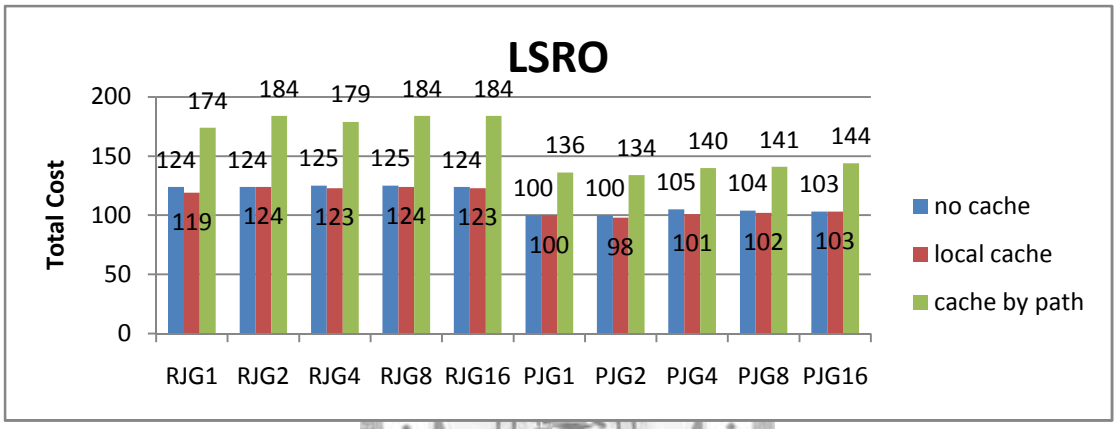
表格 4-2：Return Cost in 4 query patterns



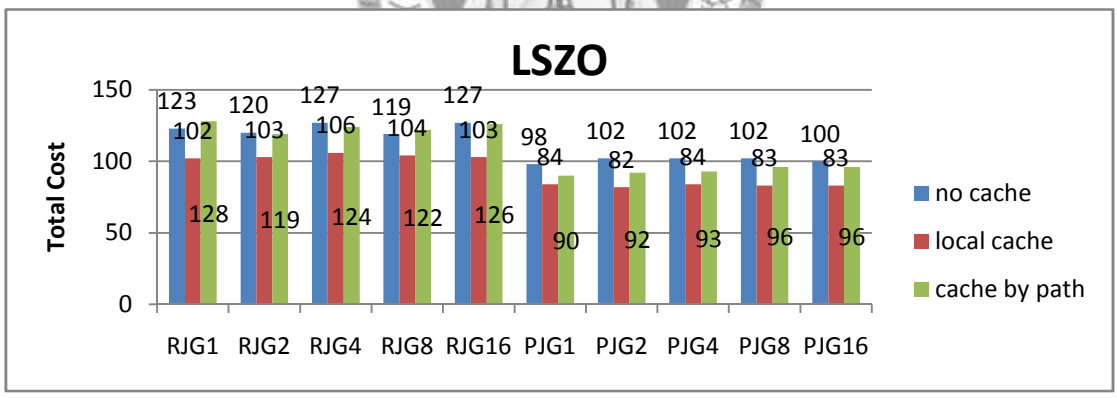
(a)Query Pattern：Random Source-Random Object



(b) Query Pattern：Random Source-Zipf Object



(c)Query Pattern：Locality Source-Random Object



(d) Query Pattern：Locality Source-Zipf Object

圖 4-13：Total Cost on Four Kinds of Query Pattern. x 軸-RJ = Random Join, PJ = Proximity Join, G = Group Number, y 軸-Average Total Cost

4.3.3 本地端快取對鄰居搜尋加鄰近表繞路的影響

Donuts 的系統環狀結構能適度反應節點在實際網路的相對鄰近性，在環狀結構上彼此相鄰的節點其實際距離上也是較相近的，Donuts 的鄰近性特色搭配搜尋樣式 LSZO 時，搜尋搭配本地端快取會有快取物件群聚在環狀結構上某一區段的現

象出現，因此在搜尋的過程中，由於地域性區域的因素，若一開始先查詢相鄰數個節點的快取儲存區，有很大的機會可以找到欲搜尋的物件；先搜尋鄰居最大的優點是溝通成本遠低於高階層的繞路指標花費的成本,且若順利在鄰居找到物件，也省去遠端繞路的搜尋成本，同時回傳成本也較低。我們做了實驗來驗證上述的推理並研究該搜尋多少個鄰居才是有效率的，其中地域性區域大小設為 100，Zipf 參數設為 1,快取取代方式是 LRU、快取空間大小是 15,我們在一實體網路模型上，每種設定皆做三個回合的模擬。圖 4-14 是搜尋順序依序是鄰近列表成員、鄰近表，並搭配本地端快取的搜尋成本，x 軸代表搜尋左右鄰居的個數，1 表示搜尋左右相鄰的鄰居，2 表示搜尋左右相鄰 2 個的鄰居，由於鄰近性加入演算法保留了節點的相對鄰近性，搜尋越多鄰居的搜尋成本成長率不像隨機加入演算法來的快速；圖 4-14 中“no cache”的線表示沒有做快取的搜尋成本，我們發現只有在系統結構以鄰近性加入演算法建立時,多搜尋左右鄰居的搜尋成本比沒有做快取時來的少，表示多搜尋兩個以上的左右鄰居就是沒有效率的搜尋了。

圖 4-15 是比較隨機加入演算法與鄰近性加入演算法的回傳成本，我們發現只要多搜尋兩個左右相鄰的鄰居，就可減少 7%的回傳成本，而隨機加入演算法並無法有效減少回傳成本；同時我們也發現搜尋整個鄰居列表是沒有效率的，不但搜尋成本過高，也無法更有效降低回傳成本，只要多搜尋左右相鄰的鄰居即可。[29]的搜尋方式除了搜尋鄰居列表，還多搜尋了 groupList 的鄰居列表，由圖 4-14 可發現搜尋鄰居列表成員個數的搜尋成本遞增狀態，多搜尋 groupList 的鄰居列表花費的搜尋成本一定比只搜尋鄰居列表時高出甚多，因此我們得知再額外搜尋 groupList 的鄰居列表並無法幫助改善搜尋效率,因為搜尋成本一定會比沒做快取時還要高。

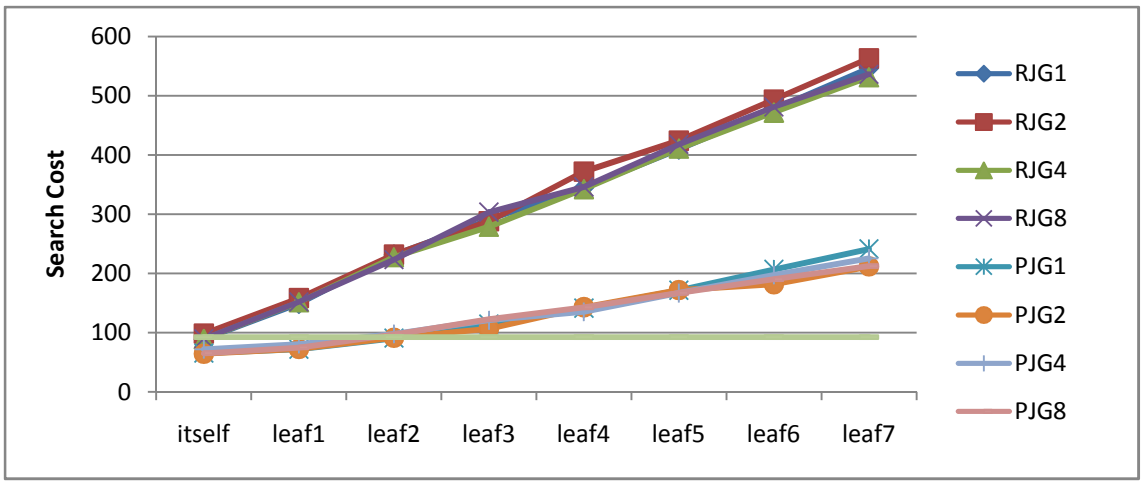


圖 4-14：Neighbor Search and Search by PT-Local Cache：Search Cost

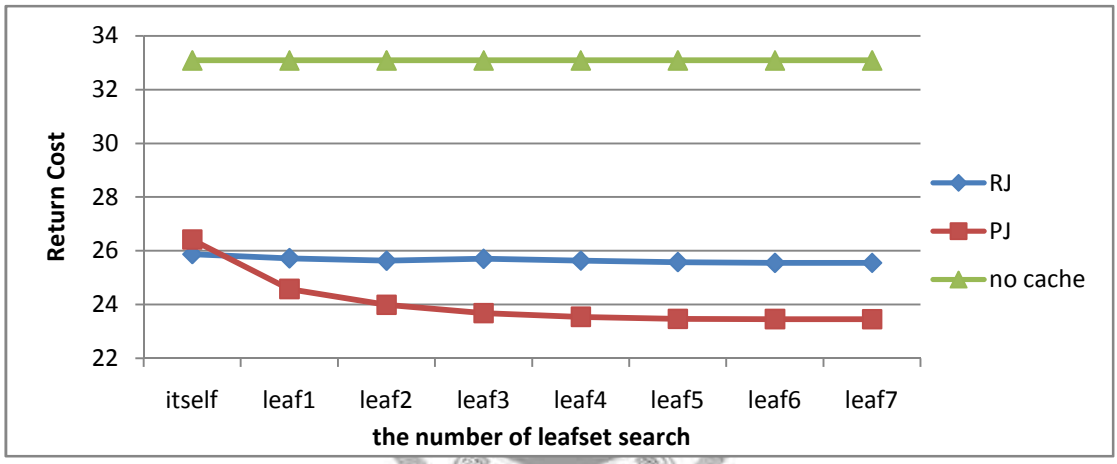


圖 4-15：Neighbor Search and Search by PT-Local Cache：Return Cost

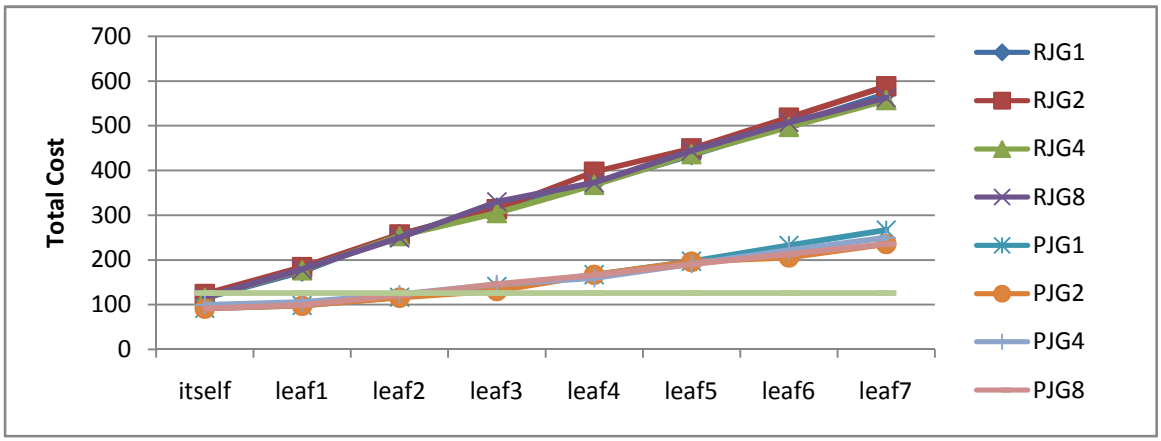


圖 4-16:Neighbor Search and Search by PT-Local Cache：Total Cost

4.3.4 合作式快取加沿路快取與傳統快取的比較

依前面的實驗結論得知，在鄰近性加入演算法建構的環狀結構中，LSZO 搜尋樣式搭配本地端快取並先搜尋左右鄰居能減少遠端繞路的次數，可見地域性與 Zipf 物件選取能讓左右鄰居快取的可利用率提高,而與左右鄰居的溝通成本較低，因此節點左右相鄰的鄰居是合作式快取成員的最佳選擇，如此一來，快取空間大小變成原來的三倍，也可減少快取物件的重複率。先搜尋左右鄰居再利用鄰近表遠端繞路的搜尋方式我們稱之為合作式鄰居搜尋，由前面的實驗我們知道沿路快取搭配搜尋鄰近表會有最低的總搜尋成本，因此我們想比較合作式鄰居搜尋搭配合作式快取加沿路快取與原先的鄰近表繞路搭配沿路快取與本地端快取的差異。快取策略除了與左右鄰居共享快取物件之外，也做沿路快取到遠端繞路走過的節點，讓受歡迎的物件快速散布到系統之中。

此部分的實驗，我們用了兩個實體網路模型來做實驗，每種模型搭配每種設定各做 6 個回合，並將地域性區域大小設為 100，Zipf 參數設為 1，快取取代方式是 LRU、快取空間大小是 15，搜尋方式選用合作式鄰居搜尋，圖 4-17 是比較四種快取策略的平均搜尋成本，依序是本地端快取、沿路快取、合作式鄰居快取搭配沿路快取、鄰居推銷搭配沿路快取。我們發現本地端快取的搜尋成本最高，沿路快取的搜尋成本除了在 PJG1 以外，其餘多群組的設定時皆是最低，而合作式鄰居快取搭配沿路快取則比沿路快取的總搜尋成本略高；由於合作式鄰居搜尋的搜尋方式額外多搜尋了合作式快取成員，比只搜尋鄰近表的方式多了兩個搜尋節點，造成搜尋成本上升；又因鄰近表是每個繞路階層中距離最近的節點，而節點左右鄰居會因為負載平衡演算法造成在環狀結構中位置的變動，使得鄰近性的特色稍微被破壞，相鄰節點的距離反而比鄰近表中的節點還長，造成鄰居間的溝通成本無法保證一定能比鄰近表還低，因此造成搭配合作式快取無法顯現出合作式快取該有的效能提升。

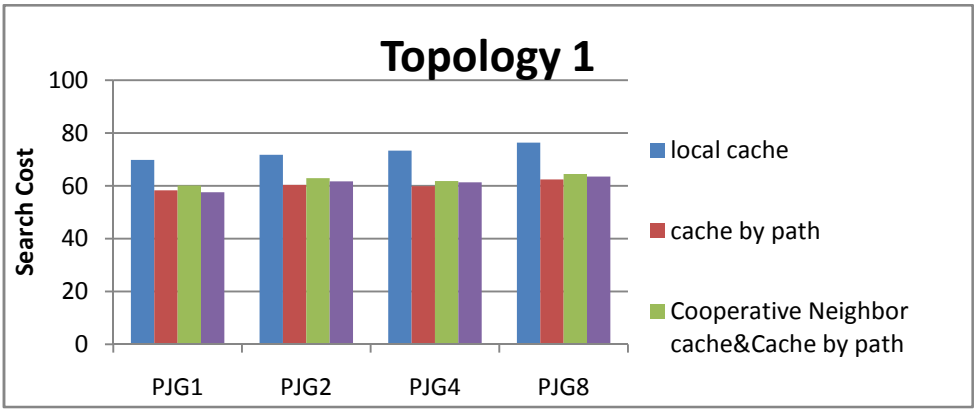


圖 4-17：Under different cache method with Neighbor&PT search for Topology 1

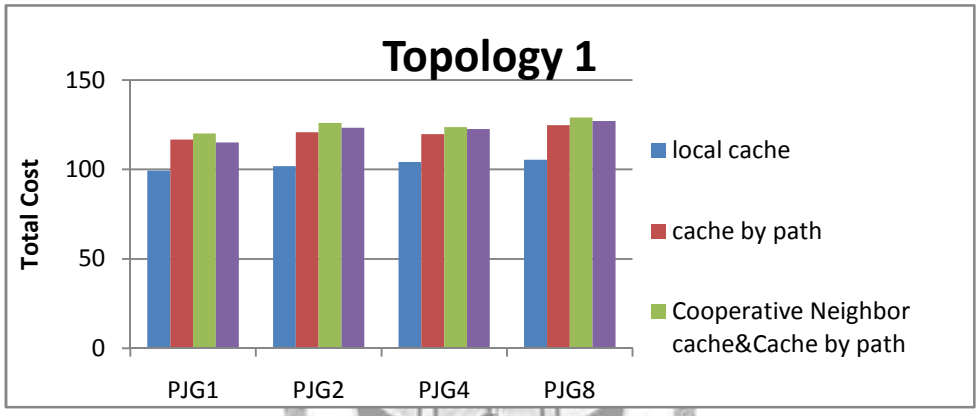


圖 4-18：Total Cost for Different Cache Method for Topology 1

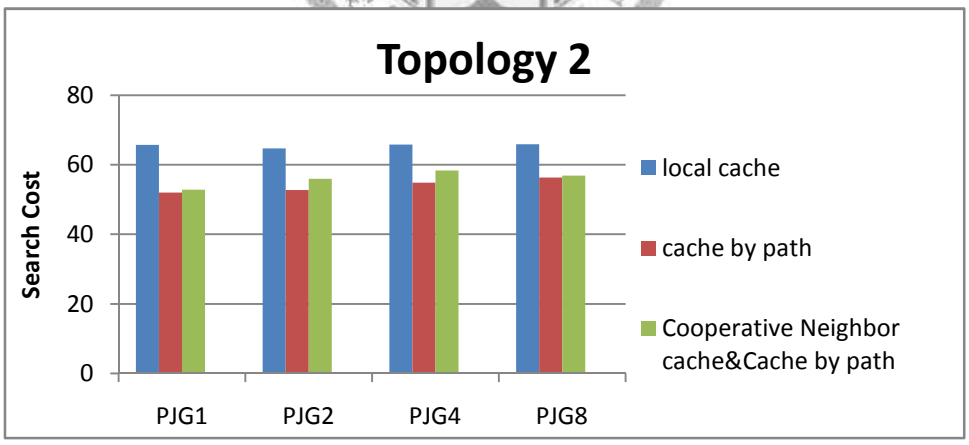


圖 4-19：Under different cache method with Neighbor&PT search for Topology 1

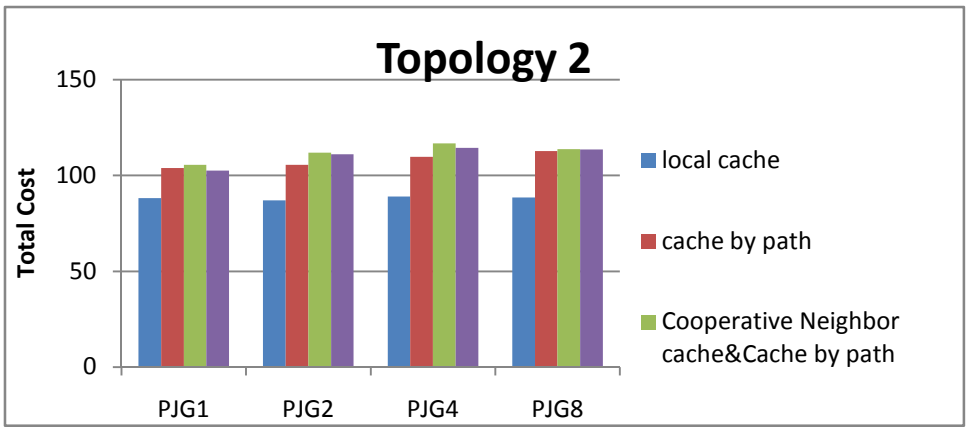


圖 4-19：Total Cost for Different Cache Method for Topology 2

我們比較圖 4-17 與圖 4-19 兩個實體網路模型的搜尋成本，發現在不同的群組數設定下，皆是合作式鄰居搜尋搭配沿路快取有最低的搜尋成本，若考慮總搜尋成本則需在加上回傳成本，本地端快取的回傳成本是直接回傳，其餘三種皆研搜尋路徑回傳，與搜尋成本相同，圖 4-18 與圖 4-20 比較總搜尋成本時，是本地端快取有最低的總搜尋成本。

4.4 總結

由前一章節的結論我們知道合作式鄰居搜尋搭配沿路快取有最好的搜尋效能，與遠端繞路搜尋搭配沿路快取和本地端快取比較搜尋成本的高低，是遠端繞路搜尋搭配本地端快取的搜尋成本最低(如圖 4-21 所示，比較同一實體網路模型下每種設定各做六個回合)；由於多了搜尋左右鄰居就多了兩段與鄰居的溝通成本，儘管合作式鄰居搜尋使本地端擊中次數增加能減少直接回傳成本，但減少的成本仍比鄰居溝通成本還高，使得合作式鄰居搜尋策略變得較不經濟，我們分析造成合作式鄰居搜尋較無效率的原因在於鄰近表的特性-每一階層的指標皆是指向該階層距離最近的節點，而 Donuts 為了維持物件負載平衡，會定期做負載平衡演算法-節點重排，使節點離開原本的位置去加入到負載較高的節點旁邊，這樣的做法會破壞環狀結構原先的鄰近性特色，使相鄰節點的距離不再是原先距離較近的；反之鄰近表定期更新一次正確性，使它確實指到每階層最近的節點，因此鄰近表的成員比左右鄰居的溝通成本還低的可能性就更高了。

在 LSZO(LS=100，Zipf=1.0)這種最接近實際搜尋樣式的情形下，搭配 Donut 的 Cordal Ring 環狀結構與鄰近性加演算法，無論群組數的設定為何，鄰近表繞路搜尋搭配本地端快取皆有最低的總搜尋成本。

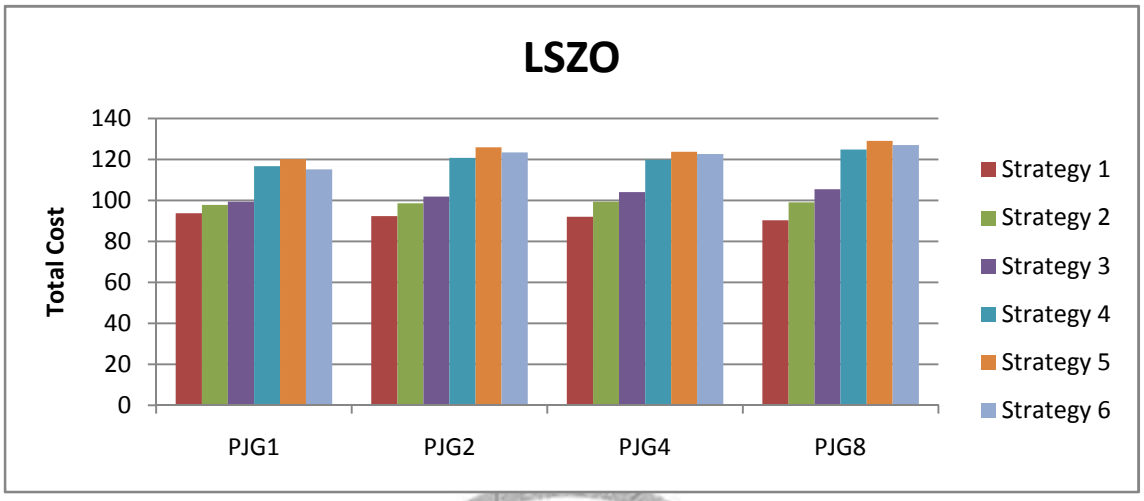


圖 4-21：All Cache Methods

Strategy 1：鄰近表搜尋搭配本地端快取　　Strategy 2：鄰近表搜尋搭配沿路快取

Strategy 3：合作式鄰居搜尋搭配本地端快取　　Strategy 4：合作式鄰居搜尋搭配沿路快取

Strategy 5：合作式鄰居搜尋搭配合作式鄰居快取&沿路快取

Strategy 6：合作式鄰居搜尋搭配鄰居推銷&沿路快取

第五章 結論

Donuts 是一保有鄰近性特色與負載平衡的範圍查詢系統，在本篇論文當中，我們在 Donuts 上討論搜尋與快取的相關議題，提出與實際點對點網路較為相似的搜尋樣式：考慮地域性搜尋者與物件受歡迎的程度，並以此為基礎衍生搭配的快取策略。由實驗結果得知，在每一種搜尋樣式底下，Chordal Ring 皆是較適合做範圍查詢的系統資料結構，除了平均總搜尋成本較少之外，整體系統的執行速度也較快；在比較傳統快取方法的實驗中，得知在 RSZO 與 LSZO 兩種搜尋樣式底下才能透過快取明顯改善搜尋效能；而沿路快取是能減少最多搜尋成本的快取策略。

利用 Donuts 能反應鄰近性的特色，提出與之搭配的搜尋演算法與快取策略：合作式鄰居搜尋與合作式鄰居快取加沿路快取，讓 Donuts 系統結構上相鄰節點的快取能互相分享，我們的實驗結果顯示這樣的策略是能提高本地端快取擊中次數，省去遠端繞路搜尋，並讓平均回傳成本降低。歸納所有的搜尋與快取策略，在 LSZO 此種較接近一般搜尋樣式的情形裡，我們得知鄰近表繞路搜尋搭配本地端快取是最適合 Donuts 系統特性的搜尋與快取策略，在快取空間不大的情形下，不僅有較低的搜尋成本，還能使快取空間有最好的使用效能。

參考目錄

[1] Gnutella http://www.gnutella.com.
[2] Freenet http://freenet.sourceforge.com
[3] Bittorrent http://www.bittorrent.com/
[4] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A scalable content-addressable network. In Proceedings of the 2001 conference on appli-cations, technologies, architectures, and protocols for computer communications (SIG-COMM), pages 161–172. ACM Pres, August 2001.
[5] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In Proceedings of the 2001 conference on applications, technologies, architectures, and protocols for computer communications (SIGCOMM), pages 149–160. ACM Press, Auguest 2001.
[6] Antony Rowstron and Peter Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In Proceeding of IFIP/ACM International Conference on Distributed Systems Platforms, pages 329–350, November 2001.
[7] C. Greg Plaxyon, R. Rajaraman, and A. W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In Proceeding of the Ninth Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA 1997), pages 311~320. ACM Press, June 1997.
[8] E. P. Markatos. Tracing a large-scale peer to peer system: An hour in the life of gnutella. In Proceedings of the 2nd IEEE / ACM International Symposium Cluster Computing and the GRID (CCGRID 2002), page 65-74. IEEE Computer Society, 2002.
[9] C. Wang, L.Xiao, Y. Liu, and P. Zheng. Distributed caching and adaptive search in multiplayer P2P networks. In Proceeding of 24th International Conference on Distributed Computing Systems (IDCS' 04), 2004
[10] S. Patro and Y. C. Hu. Transparent query caching in peer-to-peer overlay networks. In Proceedings of International Parallel and Distributed Processing Symposium (IDCS' 03), 2003
[11] B. M. Duska, D. Marwood, and M. J. Freeley. The measured access characteristics of World-Wide-Web client proxy caches. In Proceedings of the USENIX Symposium on Internet Technologies and Systems (ITS' 97), 1997
[12] M. Kacimi and K. Yetongnon. Evaluation study of a distributed caching

based on query similarity in P2P network. In Proceedings of The 2nd International Conference on Scalable Information Systems (INFOSCALE 2007), 2007
[13] L. Fan, P. Cao, J. Almeida and A. Z. Border. Summary Cache: A scalable wide-area Web cache sharing protocal. In IEEE/ACM Transactions on networking, 8(3): 281-293, 2000
[14] D. Karger, E. Lehman, F. L. Leighton, M. Levine, D. Lewin and R. Panigrahy. Consistent hashing and random tree: Distributed caching protocals for relieving hot spots on the World Wide-Web. In Proceedings of the 29th ACM Symposium on Theory of Computing, 1997.
[15] S. Iyer, A. Rowstron and P. Druschel, Squirrel: A decentralized peer-to-peer web cache. In Proceedings of the 21th ACM Symposium on Principles of Distributed Computing (PODC 2002), page 213-222, 2002
[16] P. Linga, I. Gupta and K, Birman. A churn-resistant peer-to-peer web caching system, In Proceedings of ACM workshop on Survivable and self-regenerative systems, 2003
[17] M. Roussopoulos and M. Baker. CUP: Controlled Update Propagation in Peer-to-Peer Networks. In Proceeding of International Workshop on Peer-to-Peer Systems (IPTPS' 04), 2004
[18] R. P. Zhao, J. D. Kibiatowicz, and A. D. Joseph. Tapestry: An infrastructure for faul-tolerant wide-area location androuting. Technical report, University of California, Berkeley, April 04, 2001
[19] R. Bhagwan, K. Tati, Y. C. Cheng, S. Savage, and G.M. Voelker. Total Recall: System support for automated availability management. In Proceedings of the 1st Symposium on Networked Systems Design and Implementation (NSDI), 2004
[20] D. J. Watts*, S. H. Strogatz. Collective dynamics of 'small-world' networks. In Nature, 1998
[21] J. Kleinberg. The Small-world Phenomenon: An Algorithmic Perspective*. In Proceedings of the thirty-second annual ACM symposium on Theory of computing, 2000
[22] H. Zhang, A. Goel, R.Govindan, Using the small-world model to improve Freenet performance. In Science Direct-Computer Network 46: 555-574, 2004
[23] S. Merugu*, S. Srinivasan, E. Zegura. Adding structure to unstructured peer-to-peer networks: the use of small-world graphs. In J. Parallel Distrib. Comput. 65(2005) p.142~p.153
[24] K. Sripanidkulchai, B. Maggs, H. Zhang. Efficient content location using interest-based locality in peer-to-peer systems. In INFOCOM 2003.

Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies. IEEE
[25] M. Berry, Z. Drmac, and E. Jessup, Matrices, Vector Spaces, and Information Retrieval, In SIAM Review, 41(2):335-362, 1999.
[26] M. W. Berry, S. T. Dumais, and G. W. O'Brien, Using Linear Algebra for Intelligent Information Retrieval, In SIAM Review, 37(4):573-595, 1995.
[27] BRITE. http://www.cs.bu.edu./brite/, 2003.
[28] Yi-Fang Chou, Donuts: A Network-Aware and Load-Balanced System for Range Query
[29] Wing Tat Wong, A Proximity-Aware and Group-based Peer to Peer System for Range Query.
[30] J. Aspenes and G. Shah. Skip graphs. In Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2003), pages 384~393. Society for Industrial and Applied Mathematics, Jan. 2003.
[31] S. Podlipnig, and L. Ormenyi. A Survey of Web Cache Replacement Strategies. In ACM Computing Surveys (CSUR), pages 374~398, 2003
[32] Abrams M., Standridge C. R., Abdulla G., and Williams S. Caching proxies: Limitations and potentials. In Proceedings of the 4th International World Wide Web Conference. 1995
[33] AGGARWAL,C.C.,WOLF ,J.L., AND YU, P. S. 1999. Caching on the World Wide Web. IEEE Trans. Knowl. Data Eng. 11, 1 (Jan.), 94 - 107.
[34] ARLITT , M., FRIEDRICH, R., AND JIN, T. 1999a. Workload characterization of a Web proxy in a cable modem environment. ACM SIGMETRICS Perform. Eval. Rev. 27, 2, 25 - 36.
[35] REDDY,M. AND FLETCHER, G. P. 1998. Intelligent Web caching using document life histories: A comparison with existing cache management techniques. In Proceedings of the 3rd International Web Caching Workshop.
[36] CHANG, C.-Y., MCGREGOR,T., AND HOLMES, G. 1999. The LRU* WWW proxy cache document replacement algorithm. In Proceedings of the Asia Pacific Web Conference.
[37] RIZZO,L. AND VICISANO, L. 2000. Replacement policies for a proxy cache. IEEE/ACM Trans. Netw. 8, 2 (Apr.), 158 - 170.
[38] S. Saroiu, P. K. Gummadi, and S. D. Gribble, A Measurement Study of Peer-to-Peer File Sharing Systems. In ACM SIGCOMM Computer Communication Review, Jan. 2002.
[39] A. Medina, I. Matta and John Byers. On the Origin of Power Laws in Internet Topologies*. ACM SIGCOMM Computer Communication Review, April

2000
[40] J. C. Chu, K. S. Labonte, and B. N. Levine. Availability and locality measurements of peer-to-peer file systems. In Proceedings of SPIE Vol. 4868, 2002
[41] Zhiyong Xu and Yiming Hu. Exploiting Spatial Locality to Improve Peer-to-Peer System Performance. In Proceedings of the The Third IEEE Workshop on Internet Applications (WIAPP), 2003
[42] J. Wang. A survey of web caching schemes for the internet. ACMComputer Communication Review, 29(5):36–46, 1999
[43] A. Mahanti, D. Eager and C. Williamson. Temporal locality and its impact on Web proxy cache performance. Special issue on internet performance modeling, pages: 187 - 203, 2000
[44] A.R. Bharambe, M. Agrawal and S. Seshen. Mercury: supporting scalable multi-attribute range queries. In Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications. Pages 353~366, 2004
[45] Rizzo L. and Vicisano L. Replacement policy for a proxy cache. In Networking, IEEE/ACM Transactions on, April 2000.