一個適用於資料廣播的混合式快取策略

A hybrid caching policy for data dissemination

呂永和、張朝宗、洪振洲

國立臺灣科技大學資訊管理研究所

台北市基隆路四段四十三號

E-mail: yhl@cs.ntust.edu.tw

摘要

在行動計算環境中，因為資料廣播具有同時服務大量客戶端的特性，因此利用廣播方式傳送資料給大量客戶端是一個有效的方法。廣播環境中，選擇式調校(Selective tuning)是一種被廣泛用來節省能源消耗的方法。而利用快取技術能同時降低能源消耗與等待時間。目前快取技術方面的研究，都是只單獨考慮快取資料與快取索引。我們認為將索引或資料放入快取中都可能增進系統的效益，因此提出同時快取資料與快取索引的混合式快取策略。在我們所提出的混合式快取策略中，我們利用三個因素來衡量一個資料或索引對系統的價值，並以此判斷是否要快取該資料項：這三個因素分別是(1)資料或索引的使用機率、(2)資料或索引被重複使用時，所能降低的能源與等待時間、(3)資料異動的頻率。混合式快取策略能根據資料異動的頻率決定快取中最佳的資料與索引比例。根據實驗顯示，我們所提出的混合式快取策略與單獨考慮快取資料或快取索引的策略相比，確實能達到節省能源與降低資料取得時間的目的。

關鍵詞：資料廣播、能源節省、快取策略

Abstract

Since a broadcast server can serve many mobile clients simultaneously, data broadcasting is adopted for data dissemination in mobile computing environments. In data broadcasting, selective tuning can be used for reducing energy consumption, while data caching can be used for reducing both energy consumption and access time. Existing cache policies either cache data items or indices. It would be beneficial to cache both of them. In this paper, we propose a hybrid cache policy which cache both data items and indices. In the hybrid caching policy, a profit function is used to measure the value of keeping an item (a data or an index) in the cache. When cache space is full, the item with the lowest profit is dropped from the cache. The profit function considers three factors: (1) the access probability of an item, (2) the amount of energy and waiting time saved by caching the item, and (3) the update frequency of the item. According to the update frequencies of data items, the hybrid policy determines the optimal proportion of data items to indices in the cache so as to reduce the mean access time and tuning time. The experiments show that the hybrid policy outperforms those policies that cache only data items or only indices.

Key words： Data Dissemination, Energy Saving, Caching Policy

1. 緒論

近年來，因為資訊科技的日新月異以及無線網路技術的快速進步，使得人們可以不受”線 ”的拘束而在任何地方甚至在是移動中的狀況下使用行動運算裝置(如行動電話、筆記型電腦、PDA 等)。

在廣播環境下，節省電力與降低資料獲取時間是目前大家追求的目標。廣播的資料傳送方式為伺服器端在廣播頻道上週期性地廣播資料，當行動客

戶端需要某個資料項時就進入頻道等待，直到自己需要的資料項被播放後，再從頻道上離開。當行動客戶端進入頻道等待伺服器端廣播的資料時，會由所謂的「休眠模式」 (Doze mode)轉為「活躍模式」(Active mode)，而休眠模式是以較省電的方式運作，所以轉到活躍模式後會比較耗電。在廣播環境中的效能評估方面，一般是以調校時間(Tuning time)衡量擷取資料過程中消耗的電力，而以獲取時間(Access time或Response time)代表獲取資料的平均時間。調校時間是指行動客戶端進入廣播通道中，監聽(Listening)頻道所花費的時間，也就是以活躍模式運作的時間，這段時間可用來測量行動客戶端在獲取所需要資料項時所消耗的能源。至於獲取時間則是指行動客戶端從提出資料需求一直到取得資料項所花費的時間。如圖1為行動客戶端隨機進入頻道取得data 2的調校時間和獲取時間，調校時間為黃色區域的時間總和。

在廣播環境下，根據T. Imielinski等在[1]提到，bucket是廣播的最小單位，bucket分為index bucket和data bucket二種，連續相鄰的index bucket構成index segment；連續相鄰的data bucket構成data segment。當行動客戶端提出資料請求後，必須先取得索引(Index)以得知需要的bucket被伺服器端廣播的時間，之後便可先離開頻道進入休眠模式，而不用持續監聽頻道上的資訊，等到所請求的bucket被播放時，再進入頻道恢復成活躍模式接收bucket，減少擷取資料所需花費的調校時間，以達到節省能源的目的。行動客戶端可在廣播頻道中利用樹狀結構的索引使資料的擷取更有效率，此樹狀結構稱為索引樹，如圖2(a)所示，在樹的最底層是資料項，其餘節點則是索引。行動客戶端取得資料項的方式是根據索引樹的指引，拿取相關的索引後，便可擷取到需要的資料項。例如客戶端要取得data 22，

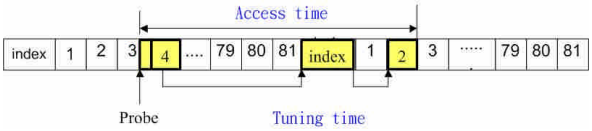
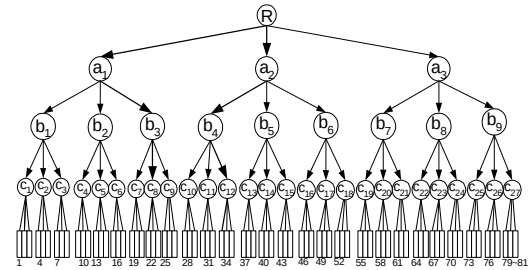
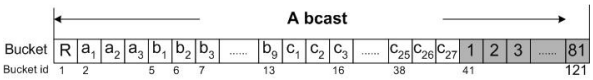


圖 1　Access time 與 Tuning time



(a)



(b)

圖 2　廣播結構 (a)索引樹 (b)廣播排程

只要依序取得 R、a_1、b_3、c_8 後，便可擷取到 data 22。相對應的廣播排程則如圖 2(b)所示。

在廣播環境下，除了利用索引之外，使用快取技術也可使資料擷取更有效率。因為擷取資料前必須先獲取相關索引，因此可分快取資料[3，4]與快取索引[2]兩方面做探討。在快取資料方面的研究，一般都是考慮(1)資料被使用的機率、(2)資料被播放的頻率、(3)下載資料所花費的成本。來決定放入快取的對象。如果客戶端需要的資料項存在於快取中，就可直接從快取中獲得，不必再花費任何的調校時間與獲取時間來取得所需的資料項。但是如果資料有異動的情形發生，則快取的效益可能會被打折扣，一旦快取中的資料被異動，此筆資料在快取中就無法發揮任何效用。在快取索引方面，因為擷取資料前必須先獲取相關索引，因此將適當的索引放入快取，也可增進資料擷取的效率。且快取索引所帶來的效能將不會因為資料有異動情形發生而受影響，資料異動後，快取中的索引仍可使用。但與快取資料相比，快取索引的方法在取得所需資料過程中，利用快取中的索引後，仍舊得進入頻道擷取資料，因此節省的調校時間與獲取時間較快取資料的方法少。

事實上，快取資料與快取索引都可能為系統帶

來效益。在每個 bucket 被存取的機率方面，因為以索引樹為基礎的資料擷取方式，在取得所需資料前必須獲取相關索引，因此位於索引樹中越上層的節點被使用的機率越高，而資料項是位於索引樹的最底層，因此索引被使用的機率普遍較資料高。本篇文章的主要目的是降低調校時間與獲取時間，就這方面而言，快取資料所降低的調校時間與獲取時間較快取索引多。至於 bukcet 失效的問題，因為索引不會異動，而資料則有異動的可能，因此快取資料會有資料失效的風險存在。由以上可得知，若是單獨考慮快取資料或是快取索引都可能使快取管理機制過於僵化。因此我們提出整合快取資料與快取索引的混合式(Hybrid)快取策略，我們認為在資料有可能異動的情形下，應該同時考慮快取資料與快取索引。

本篇文章之後的章節編排如下：在第二節是研究方法，主要是介紹我們所提出的混合式快取策略。第三節是實驗的部分，我們將比較不同的參數對調校時間與獲取時間的影響。第四節是結論。

2. 混合式快取策略

2.1 影響系統效能的因素

在廣播環境下，利用快取技術使資料擷取更有效率的方式中，有三個因素會影響到系統的效能：(1)bucket 的使用機率、(2)bucket 被重複使用時，所能降低的調校時間與獲取時間、(3)bucket 異動的機率。我們所提出的混合式快取策略，主要就是以這三個因素為主要考量，決定快取的對象

2.2 效益函式

我們將三個會影響系統效能的因素量化後，建立成效益函式(Profit Function)，利用效益函式可衡量出將每個 bucket 放入快取後，所能為系統帶來的效益，以效益值來決定快取的對象，使得利用快取所獲得效益能進一步擴大。

效益函式(Profit Function)

$$Profit(x)=P(x)\cdot R(x)\cdot U(x)$$

Profit(x)：將 x 放入快取後，能為系統帶來的效益
P(x)：bucket x 的存取機率
R(x)：使用 bucket x,可節省的調校時間與獲取時間的比例
U(x)：bucket x 放入快取後,client 下次要使用 x 時,x 尚未異動的機率

其中 *Profit(x)*是代表將 x 放入快取後，期望能降低的調校時間與獲取時間的總和。接著我們將分別針對效益函式的三個組成因素做仔細的探討。

(1) bucket 的存取機率— P

我們假設每個資料項被存取的機率為已知，因此可推算出每個索引的存取機率。算法會是將每個索引所包函的資料範圍內的資料項存取機率加總。

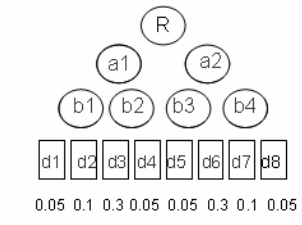


圖 3 索引樹

如圖 3 所示，d1~d8 是代表資料項，其他則是索引，每個資料項下面的數字代表該資料項被存取的機率。假如我們現在要計算 b3 存取的機率，算法是將 b3 包含的資料範圍內的資料項(d5、d6)存取機率加總，因此 P(b3)=P(d5)+P(d6)=0.05+0.3=0.35。若是我們考慮將一個 bucket 放入快取，而此 bucket 在索引樹中的直接子節點若已存在於快取中，則此 bucket 的機率要做一些調整。我們假設 b3 已經存在於快取中，而我們現在要計算 a2 的存取機率，因為 b3 已經存在於快取中，若是行動客戶端需要

d5、d6 時，會使用 b3 而不會使用 a2，因此 a2 所包含的資料範圍由原本的 d5、d6、d7、d8 變為 d7、d8。因此 b3 存在於快取的情形下，P(a2)=P(d7)+P(d8)=0.1+0.05=0.15。同樣的，當快取中某一個 bucket 將被從快取中移除時，若此 bucket 在索引樹中的直接父節點已經存在於快取中，則此直接父節點的存取機率要做調整。我們假設現在 a2、b3 同時存在於快取中，若要將 b3 從快取中移除，a2 的存取機率要做調整。當 b3 在快取中的情形下，a2 包含的資料範圍是 d7、d8，當 b3 從快取中移除後，a2 的資料範圍變為 d5、d6、d7、d8，因此 a2 的存取機率要調整為 P(a2)=P(d5)+P(d6)+P(d7)+P(d8)=0.5

(2) 使用 bucket，可節省的調校時間與獲取時間的比例— R

在廣播環境下，調校時間是能源消耗的指標，獲取時間則是獲取資料平均花費的時間，兩者對系統而言都是越低越好。一般都是將調校時間與獲取時間分開比較，但在一些應用中，我們可能要同時考慮這兩個時間。在[5]中作者利用一個指標來將兩者做結合。在這邊我們利用類似的概念，將調校時間與獲取時間做結合，使之成為一個指標。

計算步驟如下：

1. 計算快取每個 bucket 所能降低的 access time 與 tuning time，分別以 RAT (Reduced Access Time)與 RTT (Reduced Tuning Time)表示。其中 RAT 與 RTT 的算法將在附錄做介紹。

2. 將 RAT 與 RTT 的最大值與最小值分別以 RAT_{max} ,RTT_{max} ,RAT_{min} ,RTT_{min} 表示，並另外記下,以方便之後做正規化。

3. 利用 Min-max normalization 把 RAT 與 RTT 正規化可得到 RATR(Reduced Access Time Ratio) RTTR(Reduced Tuning Time Ratio)。

$$RATR(x)=\frac{RAT(x)-RAT_{\min}}{RAT_{\max}-RAT_{\min}}$$

$$RTTR(x)=\frac{RTT(x)-RTT_{\min}}{RTT_{\max}-RTT_{\min}}$$

4. 利用 RATR 與 RTTR 可算出 RAVR(Reduced Average Time Ratio)，我們所求的 R 即為 RAVR。

$$RAVR(x)=\frac{1}{2}(RATR(x)+RTTR(x))$$

$$R(x)=RAVR(x)$$

若是要計算某個 bucket 的 R 值時，此 bucket 在索引樹的直接父節點已存在於快取中，則此 bucket 的 R 值，會因為其直接父節點的存在而被削減，因此要扣掉直接父節點的 R 值才是真實的 R 值

(3) bucket 放入快取後，client 下次要使用時，尚未異動的機率— U

我們假設伺服器端異動資料的間隔時間服從指數分配，如圖 4 中的 u；而客戶端查詢的間隔時間一樣也呈現指數分配，如圖 4 中的 q。假設 u 與 q 獨立。

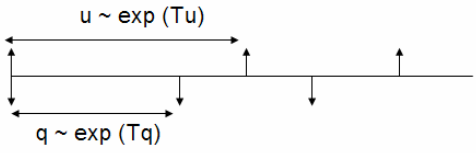


圖 4 client 查詢間隔時間與 server 異動間隔時間

我們要推導的是客戶端查詢資料的時間點還是有效的，因此所求 U=P(u>q)。

因為 u~exp(Tu)以及 q~exp(Tq)

因此 $f(u)=\frac{1}{Tu}e^{\frac{-u}{Tu}}$, $u>0$ 且 E(u)=Tu

$$f(q)=\frac{1}{Tq}e^{\frac{-q}{Tq}} \quad , q>0 \quad \text{且 E(q)=Tq}$$

u 與 q 獨立

所以 f(u,q)=f(u)·f(q)= $\frac{1}{TuTq}e^{-(\frac{u}{Tu}+\frac{q}{Tq})}$

P(u>q)

$$=\int_0^\infty\int_q^\infty f(u,q)dudq=\int_0^\infty\int_q^\infty\frac{1}{TuTq}e^{-(\frac{u}{Tu}+\frac{q}{Tq})}dudq=\frac{Tu}{Tu+Tq}$$

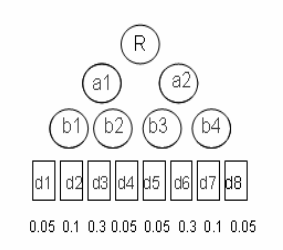
因為 Tu 代表平均異動時間，當平均異動時間越長，則行動客戶端拿到失效資料項的機率自然會比較低，也就是 U 會越大。在 $\frac{Tu}{Tu+Tq}$ 中，當 Tu 上升，則 $\frac{Tu}{Tu+Tq}$ 會變大，即 U 會越大，與之前推論的結果相符。

因為 Tq 代表平均查詢時間，當平均查詢時間越長，則行動客戶端拿到失效資料項的機率會越高，也就是 U 會越小。在 $\frac{Tu}{Tu+Tq}$ 中，當 Tq 上升，則 $\frac{Tu}{Tu+Tq}$ 值會降低，即 U 會變小，一樣與之前的推論結果相符。

我們所提出的混合式快取策略只有在快取滿的時候，才會啟動運作的機制。在快取中的每一個 bucket 經由效益函式計算後都可得到一個效益值。當快取滿了，而又有一個新的 bucket 考慮放入快取時，系統會先利用效益函式計算此新 bucket 放入快取後的效益值，之後再將此效益值與快取中所有 bucket 中的最低效益值相比，若新 bucket 的效益值大於快取中的最低效益值，則此新 bucket 就將快取中具有最低效益值的 bucket 取代掉，反之則不做任何動作。

2.3 實例說明

我們以圖 5 的廣播結構為例，將效益函式的計算方式以實際的例子做說明。圖 5(a)是索引樹，索引深度 TL=3，圖 5(b)則是相對應的廣播排程。其中 indexsegment=7，datasegment=8，cycle=15。



(a)

R	a1	a2	b1	b2	b3	b4	d1	d2	d3	d4	d5	d6	d7	d8

(b)

圖 5 廣播結構 (a)索引樹 (b)廣播排程

當每個 bucket 在索引樹的直接父節點不存在於快取中時，在索引樹中同一階層的節點其 RAT 與 RTT 值會是相同的。因此我們必須先將索引樹中每一階層的 RAT 與 RTT 算出，才可把 RAT_{max}、RAT_{min}、RTT_{max}、RTT_{min} 求出。

利用附錄的公式可求出，RAT_{max}=18.5、RAT_{min}=2、RTT_{max}=5.5、RTT_{min}=1，及 RAT(d6)=18.5、RTT(d6)=5.5

若我們要計算將 d6 放入快取的效益

Profit(d6)=P(d6)·R(d6)·U(d6)

P(d6)=0.3

$$R(d6)=RAVR(d6)=\frac{1}{2}(RTTR(d6)+RATR(d6))$$

$$RATR(d6)=\frac{RAT(d6)-RAT_{\min}}{RAT_{\max}-RAT_{\min}}=\frac{18.5-2}{18.5-2}=1$$

$$RTTR(d6)=\frac{RTT(d6)-RTT_{\min}}{RTT_{\max}-RTT_{\min}}=\frac{5.5-1}{5.5-1}=1$$

所以 RAVR(d6)=1=R(d6)

假設 U(d6)=0.7

則 Profit(d6)=0.3*1*0.7=0.21

3. 實驗結果與分析

3.1 實驗參數

在這一節，我們將把之前介紹的方法，利用系統模擬的方式做實驗分析。我們使用 C++撰寫模擬程式。利用調校時間與獲取時間這兩個指標來做為效能評估的標準。比較對象則是在[2]中提出的 CPF 法與傳統的 LRU 快取策略。其中 CPF 法只快取索引， LRU 則快取資料。另外我們會觀察資料異動頻率對快取中 index bucket 比例的影響，以 index ratio 表示。

實驗的參數的設定如表 1 所示。採用分支度為 4，深度為 6 的索引樹，資料項位於索引樹的最底層，資料項共有 4096 個。

表 1 實驗參數

參數值	預設值	調整範圍
資料數目	4096(bucket)	
快取空間	100(bucket)	100~4096
θ	0.8	0~4.5
平均資料異動間隔時間 (DUI)	10(廣播週期)	0.0125~10

實驗參數的設定如下:

θ : Zipf distribution 的參數，用來表示對資料項請求的傾斜度（data skew）模式；θ 越大表示對不同資料的請求越不平均。

資料異動間隔時間服從一個指數分配，我們使用 Data Update Interval(DUI)表示該指數分配的期望值，也就是平均資料異動間隔時間。而 DUI 是以廣播週期為單位。

以 u 表示資料異動的間隔時間, 則:

$$f(u) = \begin{cases} \frac{1}{DUI} e^{\frac{-u}{DUI}} & , u > 0 \\ 0 & , o.w. \end{cases}$$

3.2 實驗分析

DUI 越大表示平均的資料異動間隔時間大，本實驗 Cache Size 設為 100 個 Bucket。從圖 6 可看出，DUI 的大小對 CPF 的 Tuning Time 幾乎沒有影響，主要原因是 CPF 的快取空間只存放索引，而索引並不會異動，因此資料是否異動，對 CPF 的 Tuning Time 都沒有任何影響。而 Hybrid 與 LRU 均允許快取存放資料，隨著 DUI 的增加，其 Tuning Time 越來越低，但 Hybrid 的調校時間卻比 LRU 低很多。主要是因為當資料異動間隔時間很短時(即 DUI 很小)，資料失效的機率相當高，而 LRU 的快取中只允許存放資料，行動客戶端需要資料時，快取中的資料大部分都失效了，因此還是得從索引樹根節點開始擷取，所以 LRU 的 Tuning Time 會很高。而 Hybrid 因為允許快取中存放索引與資料，當資料異動時間很短時，Hybrid 會快取多一點索引少一點資料，因此還是可利用索引來有效降低調校時間。當 DUI 值很小時，Hybrid 與 CPF 的調校時間相當接近。但是當 DUI 值越來越大時，Hybrid 會開始增加快取資料的比例，因此調校時間會漸漸降低。

在圖 7 中，CPF 的獲取時間並不會隨著 DUI 的變化而有所影響，主要原因還是 CPF 僅快取索引。而 LRU 與 Hybrid 都允許快取資料，隨著 DUI 越來越大，LRU 與 Hybrid 的獲取時間均越來越低。

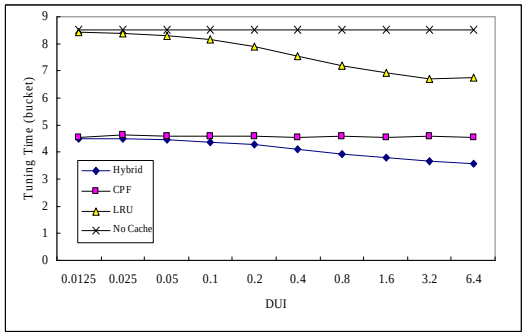


圖 6 DUI 對 Tuning Time 的影響

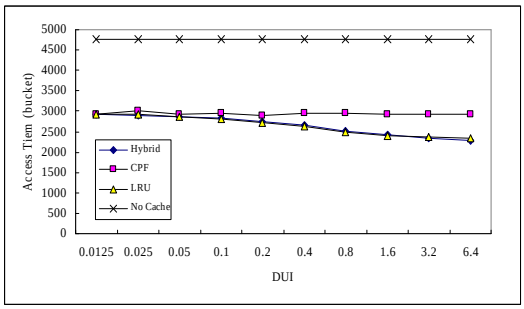


圖 7 DUI 對 Access Time 的影響

在圖 8，我們將每個快取策略的 Tuning Time 與 Access Time 分別利用 Min-max normalization 做正規化後，再做加總，即為 RAVR。我們想觀察 DUI 對 RAVR 的影響，發現 CPF 的 RAVR 值維持固定不變，因為在圖 6 與圖 7 的 CPF 其 Tuning Time 與 Access Time 均不會受 DUI 的變化而有所影響。至於 LRU 與 Hybrid 在圖 7 的 Access Time 雖然幾乎相同，但圖 6 的 Tuning Time 方面，Hybrid 則比 LRU 效果好很多，因此 Hybrid 的 RAVR 值會比 LRU 低。

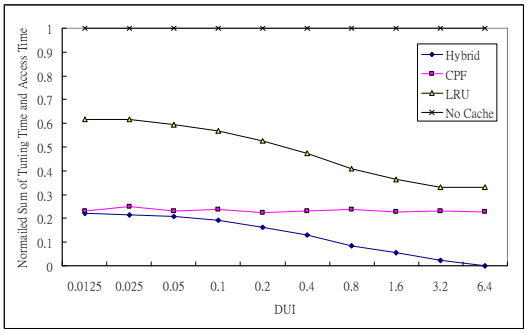


圖 8 DUI 對 RAVR 的影響

圖 9 主要是比較快取空間對 Tuning Time 的影響，隨著快取空間的加大，Hybrid、CPF 與 LRU 的 Tuning Time 也都跟著降低。因為 Hybrid 同時可快取資料與索引，它能適當調整快取中資料與索引的比例，使快取的帶來的效益有效發揮，因此 Tuning Time 都另外兩者低。而 CPF 因為只允許快取索引，所以調校時間始終比 Hybrid 高，當某些資料能為系統帶來很大效益時，CPF 並不會快取這些資料，頂多快取它上端的索引。至於 LRU 則因為只能快取資料，除非需要的資料都在快取中，否則就得從索引樹根節點重新開始擷取，因此調校時間較 Hybrid 與 CPF 高。但當快取空間越來越大，LRU 大部分資料都可從快取中滿足，因此 Tuning Time 會比 CPF 低。

圖 10 可看出 CPF 的 Access Time 始終無法大幅度降低，因為唯有需要的資料就在快取中才能使 Access Time 有效降低，所以 Hybrid 與 LRU 均能隨著快取空間的變大，Access Time 也跟著降低。

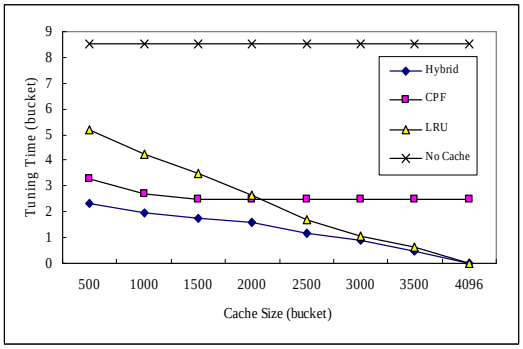


圖 9 快取空間對 Tuning Time 的影響

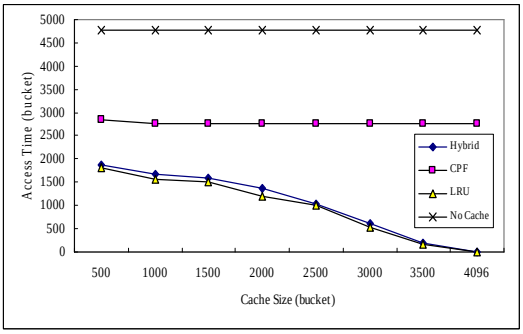


圖 10 快取空間對 Access Time 的影響

因為我們提出的混合式快取策略，允許同時快取資料與快取索引，因此我們希望探討隨著資料異動區間的變化，對行動客戶端中快取索引與快取資料的比例帶來怎麼樣的影響。圖 11 所示，整體而言，當 DUI 變大時，快取索引的比例愈低，因為 DUI 越大表示資料被異動的機率越低，因此快取資料能帶來越大的效益。另外，針對 θ=0~2 做比較，

當 θ=0 時，資料需求呈現均勻分配，也就是行動客戶端對每個資料項需求的機率都一樣，因為每個資料項被使用的機率都很低(Cache Size = 100 而 Data Size = 4096)，放入快取後下次再用到的機率也不高，因此快取資料不能為系統帶來太大的效益，所以會大部分快取索引。當 θ=0.5~2 時，資料需求漸漸越來越偏向某部分資料項，快取索引比例跟著降低，因為該部分資料被使用的機率變高，快取資料的效益增加，這種現象隨著 θ 值越接大越明顯。

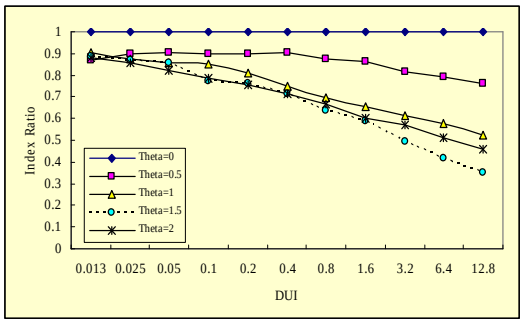


圖 11　DUI 對快取索引比例的影響

4　結論

在行動計算環境下，行動客戶端能源的節省以及等待資料項時間的降低，都是大家關注的議題。我們提出利用快取技術來達到這兩個目的。在資料項有可能異動的情形下，我們提出將快取資料與快取索引這兩個方法整合的混合式快取策略。混合式快取策略的主要概念是：同時考慮快取資料與快取索引，並以效能為依據，將能為系統帶來最大效益的資料或索引放入快取中，並可隨系統環境的變化，將快取中的資料與索引的比例做適當的調動，使得利用快取能為系統帶來的效益進一步擴大。

實驗顯示，我們提出的方法與只快取索引的 CPF 或只快取資料的 LRU 相比，確實能有效的降低調校時間與獲取時間，以達到省電與降低資料項等待時間的目的。

參考文獻

1. T. Imielinski , S. Viswanathan, and B. R. Badrinath, “Data on Air: Organization and Access,” IEEE Trans.Knowledge and Data Engineering, Vol. 9,No. 3, pp353-372 , May/June 1999.
2. Jen-Jou Hung and Yungho Leu,” Efficient index caching schemes for data broadcasting in mobile computing environments” *Database and Expert Systems Applications*, 2003. Proceedings. 14th International Workshop, pp. 139 – 143, Sept. 2003.
3. S.Acharya et al. “Broadcast Disks: Data Management for Asymmetric Communications Environments,” *Proc. ACM SIGMOD Conf.*, pp.199-210,May 1995.
4. D.A. Tran, K.A. Hua, and K. Prabhakara, “On the Efficient Use of Multiple Physical-Channel Air-Cache,” *Proc. IEEE Wireless Comm. and Networking Conf. (WCNC ’02)*, Mar. 2002.
5. Xu Yang and Athman Bouguettaya, “Adaptive Data Access in Broadcast-Based Wireless Environment,” *IEEE Trans. Knowledge and Data Eng.* , Vol 17, pp. 326-338,March. 2005.

附錄

我們以圖 x 的 Tune_opt 廣播排程為例，來介紹擷取一個資料所耗費的 Tuning Time 與 Access Time 的算法

圖 x Tune_opt 廣播排程

根據[1]指出，以索引樹為基礎的資料擷取方式，擷取一個資料項需要耗費的 Tuning Time 為 1.5+TL+1，其中 TL 表示索引樹中的索引深度。

至於 Access Time 方面，在 Tune_opt 的結構中，行動客戶端要擷取任何資料項前，必須先擷取索引樹的根節點，之後再根據行動客戶端需要的資料項擷取相關的索引，最後便可獲取需要資料項。我們以 indexsegment 表示索引區間的長度，datasegment 表示資料區間的長度，而整個廣播週期以 cycle 表示，cycle=indexsegment+datasegment。因此行動客戶端拿取資料的順序會是：

1. 行動客戶端首先進入頻道任意接收一個完整的封包，利用此封包可得知從接收一個完整的封包到下次根節點被播放所需等待的時間。這段距離就是下圖 9 中的 a。假設行動客戶端在 probe in 的時間點進入頻道，接收任意一個完整封包到下個廣播週期起始點的平均距離

   a=$\frac{1}{2}$ cycle。

2. 獲取根節點後，再根據行動客戶端需要的資料項 X，擷取相關的索引封包，因為在 Tune_opt 的結構中資料封包是擺在索引封包之後，因此要獲取所需的資料項，一定要先等全部的索引封包播放完。全部索引封包的總長度就是圖中 b = indexsegment。
3. 等完所有索引後，就可擷取行動客戶端所需要的資料項，需要的資料項有可能會在資料區塊任一個地方，平均來講是全部資料區塊的一半，如圖中的所示，需要的是 X，因此需要等待 c 長度才可獲取 X ，c=$\frac{1}{2}$ datasegment。

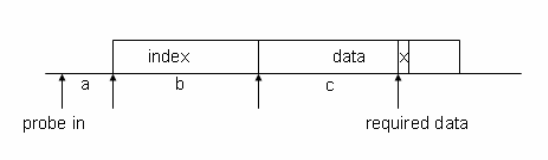


圖 1　Tune_opt 廣播排程

所以要推導的獲取時間

=a+b+c

=$\frac{1}{2}$ cycle+indexsegment+$\frac{1}{2}$ datasegment

=$\frac{1}{2}$ cycle+$\frac{1}{2}$ (indexsegment+datasegment)

+$\frac{1}{2}$ indexsegment

=cycle+$\frac{1}{2}$ indexsegment

接著我們討論有快取的情形下，獲取時間與調校時間的算法。因為行動客戶端快取的對象有 index bucket 與 data bucket，所以我們分成兩部分討論。先考慮將 data bucket 放入快取的情形。一旦將 data bucket 放入快取後，行動客戶端需要這些資料時，只需從快取中獲取，不必再進入頻道中擷取，因此將資料封包放入快取後，行動客戶端要擷取資料時所耗費的 Tuning Time=0，Access Time=0。因此將 data bucket 放入快取後，節省的 Tuning Time 與 Access Time，也就是 RAT 與 RTT。

RTT=1.5+TL+1

RAT= cycle+$\frac{1}{2}$ indexsegment

再來我們考慮將 index bucket 放入快取後，Tuning Time 與 Access Time 的算法，先討論 Tuning

Time 的部分。假設存放在快取的 index bucket 是位於索引樹的第 K 層。在沒有快取的情行下，當行動客戶端要擷取資料需要花費的 Tuning Time 是 1.5+TL+1，而利用快取中的索引封包後，其 Tuning Time 會變為 1.5+(TL-K)+1。所以節省的 Tuning Time 是 K，因此 RTT=K。

Access Time 的算法，假設放入快取中的索 index bucket 位於索引樹的第 K 層，其 Access Time 必須考慮兩種情形：第一種情形是下一個要獲取的相關封包是 index bucket，第二種情形是下一個要獲取的相關封包是 data bucket。

情形一：下一個要獲取的相關封包是索引封包

行動客戶端擷取到所需的資料項過程會是：

(1) 先進入頻道接收任意一個完整的封包，之後再利用此封包與快取中的 index bucket，可得知從接收一個完整的封包到下一個要獲取相關 index bucket(圖 10 中的 B)的距離，假設行動客戶端在 probe in(A)的時間點進入頻道，這段長度就是圖 10 中 A->B 的距離。A->B= $\frac{1}{2}$ Cycle。

(2) 獲取下一個要拿的相關 index bucket 後(即 B)後，因為 B 是 K+1 層的 index，必須把 K+1 層的 index 都等完，再把剩下相關的 index bucket 都拿完。因為 Tune_opt 廣播結構的特性，要獲取需要的資料項前，必須把所有的 index bucket 都等完，所以要等待 B->D 的距離。因為 B 可能在 K+1 層 index 的任一個地方，因此取平均的距離。B->D=(K+1 層剩下的 index)+(K+1 層以下的所有 index)= $\frac{1}{2}$ (K+1 層的 index)+(K+1 層以下的所有 index)

(3) 等完全部 index bucket 播放後，行動客戶端即可擷取需要的資料項，資料項有可能位於在 datasegment 任一地方，假設如圖中的所示，需要的是 C，獲取 C 所需等待的長度是 D->C，D->C= $\frac{1}{2}$ datasegment。

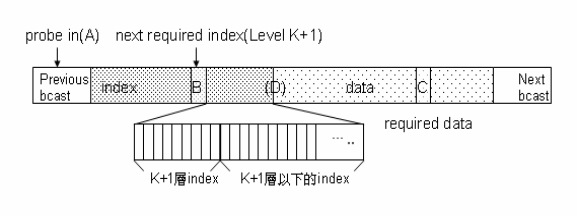


圖 2　Tune_opt 廣播排程

所要推導的獲取時間是

A->C=(A->B)+(B->D)+(D->C)= $\frac{1}{2}$ (cycle+datasegment)+ $\frac{1}{2}$ (K+1 層的 index)+(K+1 層以下的所有 index)

因此

RAT= $\frac{1}{2}$ cycle- $\frac{1}{2}$ (datasegment-indexsegment)- $\frac{1}{2}$ (K+1 層的 index)-(K+1 層以下的所有 index)

情形二：下一個要獲取的相關封包是資料封包

在這個情形下，表示快取中的索引封包是位於索引樹中最底層的索引節點。行動客戶端擷取到所需的資料項過程會是：

先進入頻道接收任意一個完整的封包，之後利用此封包以及快取中的索引封包，可得知從接收一個完整的封包到下一個要獲取資料封包(圖 11 中的 C)的距離，假設行動客戶端在 probe in(A)的時間進入頻道，所以這段長度就是 A->C ，因此 access time=A->C= $\frac{1}{2}$ cycle。

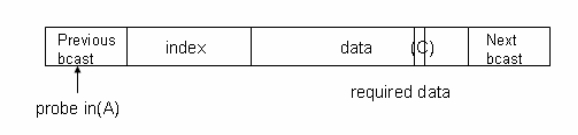


圖 3　Tune_opt 廣播排程

因此 RAT= $\frac{1}{2}$ (cycle+indexsegment)