無所不在的資料串流環境之資源感知以密度為基礎的分群

趙景明
東吳大學資訊管理學系教授
chao@csim.scu.edu.tw

李紫麗
東吳大學資訊管理學系研究生
96756019@scu.edu.tw

摘要

隨著資訊科技的進步,許多應用領域不斷地產生資料，這些資料以串流的形式出現。由於資料串流具有大量、潛在無限、快速、連續抵達與即時的特性，傳統的資料探勘技術並不適用於探勘串流資料，因此資料串流探勘是近年來非常重要的研究議題。另一方面，由於行動設備處理能力的增強，使傳統的資料探勘演算法的輕量版本可以在資源有限的行動設備上進行資料探勘，因此無所不在的資料串流探勘的概念被提出。然而，資料串流探勘的技術無法適用在此資源有限的環境，因此本論文提出 RA-DCluster 分群演算法，其建置在行動設備中探勘串流資料，並且依據行動設備目前可用的記憶體與電力來執行適應性程序，使其可以在較低的資源下持續進行分群採勘，並且更節省電力消耗。

關鍵詞：資料串流、資料探勘、分群、無所不在的資料串流探勘。

1.前言

隨著資訊科技的進步,許多應用領域不斷地產生資料，這些資料以串流的形式出現，例如網路封包、網頁瀏覽行為、電信通聯紀錄、感測資料、股票交易等，這些資料串流隱藏許多有用的資訊與知識。由於資料串流具有大量、潛在無限、快速、連續抵達、即時與不可預測的特性[2]，傳統的資料探勘技術並不適用於探勘串流資料，因此資料串流探勘是近年來非常重要的研究議題之一。

另一方面,由於行動設備處理能力的增強，使傳統的資料探勘演算法的輕量版本可以在資源有限的行動設備上進行資料探勘，因此無所不在的資料串流探勘的概念被提出。無所不在的資料串流探勘代表下一代的資料串流探勘系統，其可以在任何時間、任何地點提供智慧資料分析。無所不在的資料串流探勘的問題和挑戰如下[2,9,10,12]：如何處理連續的資料串流、如何減少行動設備的電力消耗、如何適應行動設備有限的計算能力與記憶體、如何在行動設備的小螢幕呈現資料探勘結果。由於行動設備僅具備有限的資源，使得原先可以應用在資料串流探勘的演算法無法適用在此資源有限的環境下。

許多文獻[2,8,9,10,12]指出在無所不在的環境中探勘資料串流主要受限於行動設備的記憶體大小還有電力問題，然而過去關於無所不在的資料串流分群探勘的研究[6,7,16] ，雖然可以有效地在有限的記憶體下持續進行探勘，但是卻沒有考慮到電力限制的問題，雖然記憶體足夠但是一旦電力不足時就會中斷探勘的工作。因此，本論文提出 RA-DCluster 演算法，其依據行動設備目前可用的資源做適應性處理，使其可以在有限的記憶體與電力下持續進行探勘工作。

接下來的章節由以下幾個部分所組成。第二節將探討資料串流分群、資源感知探勘技術，以及無所不在的資料串流分群的相關研究。第三節將說明 RA-DCluster 演算法，其中會談到資料取樣、對應格子矩形、建立動態式摘要格子、資源監控、演算法細粒度設定以及分群。第四節是針對 RA-DCluster 所做的實驗與分析。第五節是本論文的結論。

2.文獻探討

本節將探討資料串流分群、資源感知探勘技術、以及無所不在的資料串流分群的相關研究。

2.1 資料串流分群

近年來已有很多資料串流分群演算法被提出， Aggarwal et al. [1]提出 CluStream 演算法，主要分為線上階段及離線階段，線上階段將資料串流形成微群集(micro-cluster)，而離線分群階段則利用微群集的特徵向量進行分群，其使用金字塔時間框架(Pyramidal Time Frame)保存資料串流的概要資訊，在快速變化的資料串流環境下仍可以維持高品質的分群結果。

由於基於 K-Means 的資料串流分群的文獻上並無法有效處理任意形狀群集的資料串流且其需要預先假設群集的個數，因此 Cao et al. [3]提出 DenStream 演算法，擴充傳統密度演算法 DBSCAN (Density-Based Spatial Clustering of Applications with Noise)，DenStream 演算法採用 CluStream 的二階段處理架構，線上階段處理資料串流並產生潛在微群集(p-micro-cluster) 和孤立點微群集(o-micro-cluster)，當新的資料點抵達時，先試著合併到最近的微群集，若其半徑超過設定的範圍，則加入孤立點微群集；離線階段根據所產生的微群集，使用 DBSCAN 演算法進行分群處理。Ren and Ma [15]提出 SDStream 演算法，使用移動視窗框架來處理資料串流，並採用 CluStream 的二階段處理架構，線上階段使用群集特徵指數直方圖 (Exponential Histogram of Cluster Feature, EHCF)來儲存微群集，而離線階段進行分群處理，實驗結果證明其可以有效的處理任意形狀的群集且具有較高的精確度。

格子為基礎的分群演算法擅長於處理資料量大的資料，因此 Park and Lee [13,14]提出 Statistical grid-based 分群演算法，將資料串流切割成格子，並將每個維度的格子以成員清單(sibling list)儲存，而維度間的關係則是建立成員樹(sibling tree)，一旦格子內的資料量大於門檻值時才會再將此格子進行切割，其不僅可以分析出全部維度所構成的群集，更可以有效的分析由子空間所構成的群集。Chen and Tu [4]提出 D-Stream 演算法，是結合格子與密度的資料串流演算法，其採用 CluStream 的二階段架構，線上階段將資料串流切割成多個格子，並採用衰退因子反映資料串流隨著時間的演變過程，而離線階段首先一開始先擴充 DBSCAN 演算法進行初始群族的分群，隨後在固定的間隔時間再以最新的格子密度執行群集的分割或合併，實驗結果證明其處理資料串流的效率比 Clustream 快速，且對於多維度的資料串流具延展性，其可以適用在具有任意形狀之群集的資料串流。

2.2 資源感知探勘技術

探勘資料串流所面臨的挑戰是串流資料是高速、無限、連續抵達，但記憶體是有限的，因此資料串流探勘會採取以下方法來適應有限的記憶體：取樣(sampling)、過濾(filtering)、聚合(aggregation)和減少負荷(load shedding)，這些方法主要是在資料串流輸入時進行處理，取樣是使用統計方法在資料串流中抽取樣本作為分析代表，過濾是分析每個資料的重要性，將對探勘結果影響較不重要的資料丟棄，聚合是用統計方法對資料進行統計並將結果做為資料探勘的輸入。

AOG (Alogrithm Output Granularity) [7,9]是 Gaber et al. 提出的資源感知方法，其使資料串流探勘技術能夠意識到資源並且根據可用的資源改變它的操作，其適用在發展輕量型的資料探勘上，由於不用複雜的統計運算因此適合行動設備有限的計算能力。AOG 是一個資源感知概念，不同於上述在輸入端處理的技術，主要是藉由調整輸出細粒度來適應可用的資源進行資料串流探勘的工作，其主要分為探勘(Mining)、適應(Adaptation)及知識整合(Knowledge Integeration)三個階段，圖 1 為 AOG 處理程序。探勘階段是執行探勘演算法，當執行完一次後會依據資料速率與可用的記憶體，透過適應階段來調整門檻

值，接著判斷記憶體是否耗盡，如果是則進入知識整合階段來合併探勘的結果，否則回到探勘階段繼續執行。

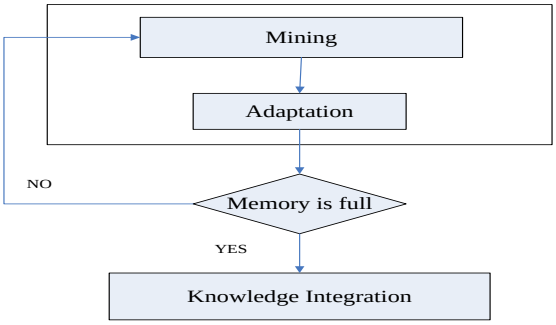

圖 1 AOG 處理程序[7]

AOG 已經被 Gaber et al. 應用在輕型探勘技術上[6,7]，LWC 是應用 AOG 在無所不在的資料串流中進行分群，一旦記憶體不足時會透過知識整合階段合併分群的結果並輸出；LWF 則是應用 AOG 在資料串流中進行頻繁項目集的探勘；LWClass 是應用 AOG 在無所不在的資料串流的分類演算法，其是一個輕量型的最近鄰 K (K-Nearest-Neighbors)分類法。Gaber et al. 基於 AOG 的概念提出 RA-UDM 系統架構，將上述 LWC、LWF 及 LWClass 三個演算法應用在無線網路進行資料串流探勘工作，其它還有很多基於 AOG 所提出的資料串流分群感知文獻將在下節說明。

2.3 無所不在的資料串流分群

Gaber et al. [6] 提出的 LWC (Light-Weight-Clustering)演算法是一個輕量型的漸增式的分群演算法，基於 AOG 的概念，一旦記憶體不足時就會合併分群的結果並輸出。Shah et al. [16] 提出 RA-VFKM (Resources-Aware Very Vast K-Means)，引用 VFKM 演算法[5]有效的串流分群技術，並且利用 AOG 資源感知技術擴充 VFKM 無法在有限資源中進行探勘的問題。當可用記憶體到達關鍵階段時，藉由增加可允許的錯誤值(ε*)和可允許的錯誤值的機率值(δ*)，來減少執行的次數與樣本數。其利用增加錯誤值與機率值的策略而妥協在最後輸出結果的準確性上，在關鍵情況時，能夠合併結果和避免執行失敗。

Horovitz et al. [11]提出以模糊邏輯(Fuzzy Logic)為基礎的分群演算法，其主要分為三個階段，第一個階段執行線上資料串流的處理，第二個階段先應用 LWC 分群演算法，然後使用模糊規則將分群的結果以專家、領域的知識把群集標籤化，第三個階段是線上分類階段，對於新進入的資料串流將其分群到已被加上標籤的群集，此方法可以不需人類來分析探勘結果，有效地減少使用者決策制定的時間。

Gaber and Yu [8]提出的 RA-Cluster (Resource-Aware Clustering)演算法是第一個可以適應不同可用資源的資料串流分群，主要分為線上微群集處理階段與分群處理二個階段。每隔一段固定的時間執行資源感知程序來適應目前的資源狀況，當記憶體不足時會調整演算法的距離門檻值，並將目前微群集中包含最少資料的微群集或最久沒有活動(inactive)的微群集刪除，即使沒有活動的微群集包含很多資料也會被刪除。當電力不足時則會調整資料取樣的大小。利用調整演算法的輸出細粒度或調整輸入的資料取樣數，使其在資源不足時可以持續進行探勘工作。但是由於其資源感知程序是每隔一段固定的時間才會執行，因此若資源在此時間內消耗完，則會導致探勘工作中斷。

3. RA-DCluster 演算法

由上一節的文獻探討，我們發現無所不在的資料串流探勘的挑戰是行動設備中有限的記憶體與電力，過去的資料串流探勘演算法並無法適用在此環境下，而之前關於無所不在的資料串流分群的研究雖然可以有效地在有限的記憶體下持續進行探勘，但是卻沒有考慮到電力限制的問題，雖然記憶體足夠但是一旦電力不足時，就會中斷探勘的工作，且無法在電力危急時，即時輸出探勘結果。因此，本論文提出 RA-DCluster (Resource-Aware Density-based Clustering)演算法來解決此問題。RA-DCluster 演算法主要分為線上維

護與離線分群兩個階段如圖 2 所示。

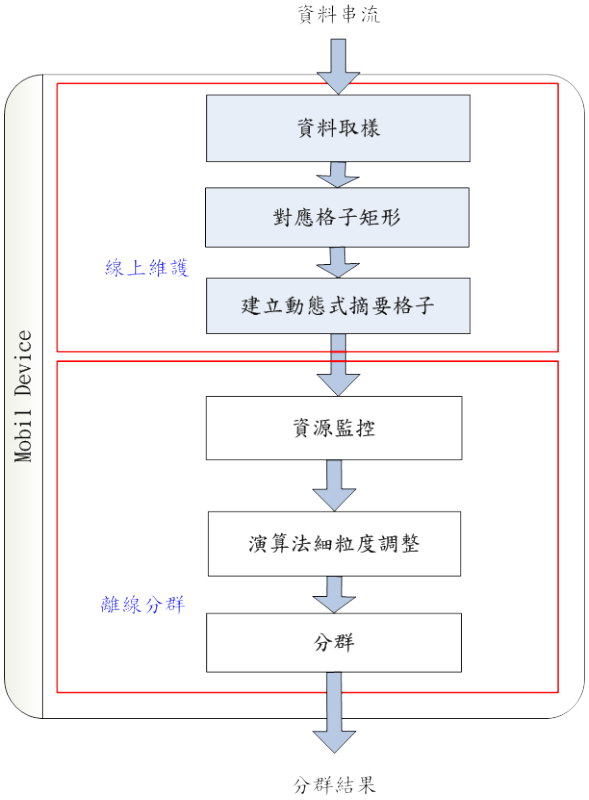

圖 2 RA- DCluster 的整體流程

在線上維護階段中，主要是維護串流中資料點對應的格子矩形(grid-cell)的統計資訊，使離線分群階段能夠利用這些資訊來做探勘，而不是探勘串流中的每一筆資料，得以減少計算的時間與複雜度。首先，利用移動視窗(Sliding Window)模式對串流資料取樣，取樣的資料是以視窗大小為單位。接著，針對移動視窗中的資料計算其歸屬的格子矩形，並且漸進式更新格子矩形的統計值。最後，使用動態式摘要格子(Dynamic Summary Grid)來儲存每個格子矩形的特徵向量，同時根據可獲取的資源來調整動態式摘要格子的細粒度。當資源不足時，將目前摘要格子的細粒度合併到較大的細粒度來減少需要處理的資料量，進而達到縮減資源消耗的目的。

在離線分群階段中，主要是根據目前記憶體來調整線上維護階段的設定，並且利用動態式摘要格子所儲存的統計資訊來進行分群。首先，我們利用資源監控程序(Resource Monitor Procedure)來計算記憶體使用量、記憶體剩餘率與電力剩餘率，並且判斷記憶體與電力資源是否不足。當記憶體不足時則會調整視窗大小與動態式摘要格子的細粒度。當電力不足時則會調整視窗大小。最後，利用以密度為基礎的分群方法針對動態式摘要格子進行分群，當記憶體或電力不足時會增加疏稀格子密度門檻值(sparse grid threshold)，使分群演算法減少所需要處理的資料量。相反地，當資源充足時則減少疏稀格子密度門檻值，以提昇分群結果的精確度。

3.1 線上維護

3.1.1 資料取樣

資料串流取樣採用移動視窗模式，圖 3 顯示移動視窗取樣範例，其中 Stream 代表一個資料串流，而 t0、t1、…、t20 分別代表一個時間點。假設使用者設定移動視窗的大小為 10，其代表一次處理資料串流中的十筆資料，因此移動視窗第一次取出十筆資料時間點為 t1 到 t10。在處理完視窗的資料點後，視窗會往右移動一次來處理接下來的 10 筆資料。在這個範例中，視窗總共移動了二次，第二個移動視窗取出的十筆資料，其分別為 A、B、C、D、E、F、G、H、I、J，其時間點為 t11 到 t20。

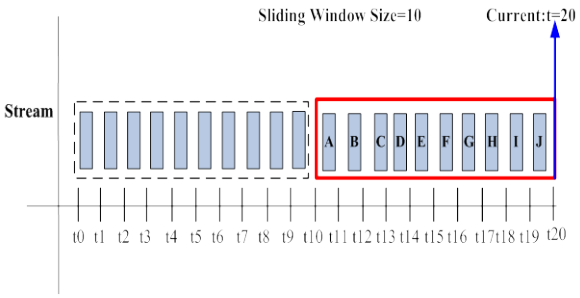

圖 3 移動視窗取樣範例

3.1.2 對應格子矩形

我們切割多維度的資料空間為分離的細小格子矩形，假設輸入資料點有 d 個維度空間 $S=S_1 \times S_2 \times \ldots \times S_d$，將每一個維度的資料空間 S_i 等分成 P_j 個區間，維度 $S_i=S_{i,1}$

$\cup S_{i,2} \cup \ldots \cup S_{i,pj}$，將這些區間用 1~j 來編號，對於一個格子矩形 gc 是由維度空間 $S_{1,p1} \times S_{2,p2} \times \ldots \times S_{d,pd}$ 所組成，我們將其表示為 $gc=(j_1,j_2,\ldots,j_d)$，$j_i=1,2,\ldots P_j$。切割後的每個格子矩形的大小是相等的。移動視窗取樣後，我們對所取出的資料點計算其所對應的格子矩形。接著，我們紀錄每個格子矩形所包含的資料點的統計資訊，產生格子矩形特徵向量(Gird-Cell Feature Vector)，其總共有 4 個資料項，表示方式為(n,ts,tm,cluster)，每個資料項的定義如下:

- n 為格子矩形所包含的資料點數量。
- ts 為格子矩形最初被建立的時間戳記。
- tm 為格子矩形最後被更新的時間戳記。
- cluster 為格子矩形所屬的群集代號，預設為 no_class。

以下是產生對應格子的步驟：

Step 1： 針對視窗取樣後的資料計算其所歸屬的格子矩形，計算方式是將資料的每個維度值除以各維度的間距。

Step 2： 判斷該對應的格子矩形是否已存在，若是則更新格子矩形)的特徵向量值，否則產生新的格子矩形。

3.1.3建立動態式摘要格子

產生格子矩形之後，接著透過動態式摘要格子來儲存每個格子矩形的特徵向量，初始時動態式摘要格子的細粒度等級 Size=0，表示目前動態式摘要格子的細粒度為最小等級，系統會依目前的記憶體來動態式調整格子的細粒度。離線分群階段則會利用目前動態式摘要格子的細粒度等級來進行分群。當記憶體不足時會自動調整格子的細粒度，如圖 4 所示，以降低分群時記憶體空間與計算時間的耗費。當記憶體充足時，會再調回原來的細粒度等級。

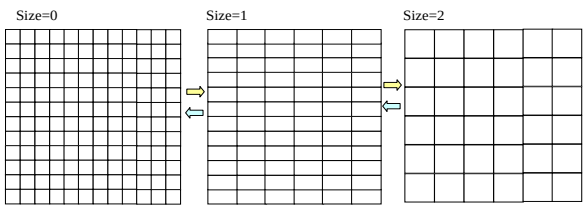

圖 4 調整動態式摘要格子的細粒度

格子的細粒度調整是利用動態式摘要格子其具有格子合併(Grid Merged)與格子解析(Grid Resolution)的兩種功能。當記憶體不足時會透過格子合併將細粒度較小的相鄰格子合併成細粒度較大的格子，以降低分群時記憶體空間與計算時間的耗費。當記憶體充足時會透過格子解析將細粒度較大的格子轉換回細粒度較小的格子。

動態式摘要格子的表示方法如圖 5 所示：

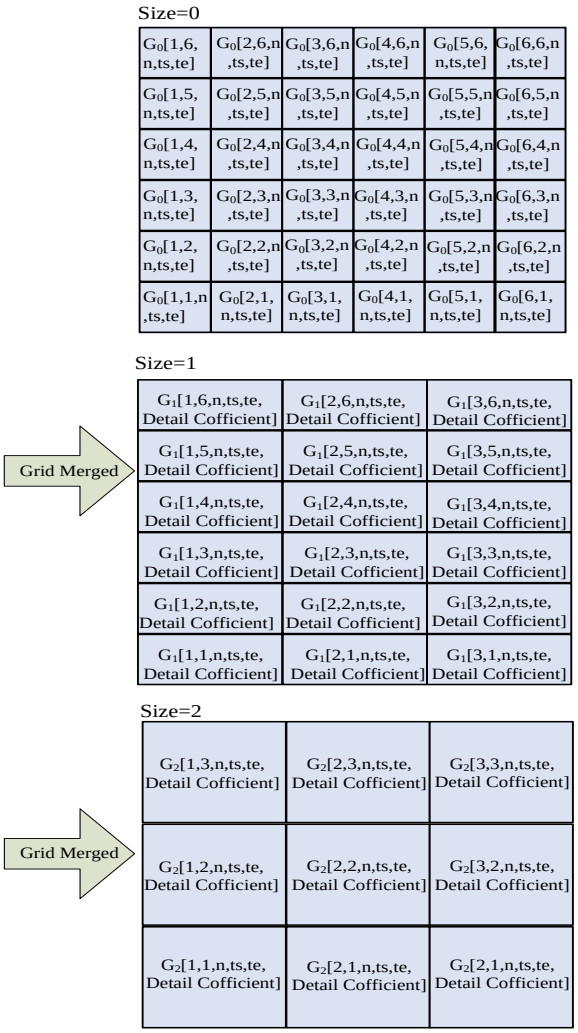

圖 5 動態式摘要格子

- Size 是用來表示目前動態式摘要格子的細粒度等級，每個等級的細粒度是由多個格子(Grid)所組成，且每一個格子儲存一個格子細胞的特徵向量。
- 每個格子可以用 $G_i[P_j,ts,te]$來表示，i 標示細粒度等級，P_j 標示格子編號，ts 代表開始儲存時的時間戳記，te 代表儲存結束時的時間戳記。

- 在動態式摘要格子 Size=0 以上的等級都會另外附加明細係數(Detail Coefficient)欄位，其用來儲存資料之間的差值，目的是為了後續進行格子解析的處理。

格子合併與格子解析的處理是利用哈爾小波轉換(Haar Wavelet Transform)，如下所述：

首先，定義動態式摘要格子的表示方法，每個細粒度等級的格子摘要模式是由多個格子(Grid)所組成，每一個格子存放一個格子矩形的特徵向量，其可以用 $G_i[P_j,ts,te]$來表示，i 標示細粒度等級，P_j 標示格子編號，ts 代表開始儲存時的時間戳記，te 代表儲存結束時的時間戳記，此種標示方法主要是用來顯示此格子所儲存的時間區間，並且當格子合併時可以快速且明白的表示透過合併處理產生的格子其所對應之較小細粒等級的格子區間範圍。格子的建立是從最小的細粒度等級開始，當產生格子矩形後會被儲存到此暫時的格子中，並且記錄格子特徵向量資料。產生出 Size=0 的細粒度模式之後，離線分群階段就可以使用此模式來做後續的處理，但是當行動設備可獲取的記憶體資源不足時，我們可以利用合併操作將 Size=0 的細粒度合併成(Size+1)的細粒度，合併的過程是從第一個維度開始，當記憶體仍不足時，再往下一個維度進行合併，當所有維度皆合併過後，若記憶體仍不足時，則會再從第一個維度開始合併，重覆以上的合併操作直到記憶體足夠。例如細粒度 Size=1 的格子 $G_1[1,1]$是合併 Size＝0 的格子 $G_0[1,1]$與 $G_0[2,1]$，細粒度 Size=2 的格子 $G_2[1,1]$是合併 Size=1 的格子 $G_1[1,1]$與 $G_1[1,2]$。格子合併的操作是利用哈爾小波轉換，其是一種壓縮資料的方法。由於其具有計算快速與容易理解的特性，因此很適合應用在運算能力有限的行動設備上。此轉換可以視為一連串計算和值與差值的處理。當記憶體充足時我們利用格子解析將合併後的格子轉換回細粒度較小的格子，其是利用格子合併時所得到的明細係數來做處理。例如細粒度 Size=1 的格子 $G_1[1,1]$是由細粒度 Size=2 的格子 $G_2[1,1]$與其明細係數進行相減再除以格子數 2，格子 $G_1[2,1]$是由細粒度 Size=2 的格子 $G_2[1,1]$與其明細係數進行相加再除以格子數 2。

3.2 離線分群

3.2.1 資源監控與演算法細粒度調整

在離線分群階段，我們利用資源監控程序來監控記憶體與電力，並且依據行動設備目前可用的記憶體與電力執行適應程序。資源監控程序包含二組參數，第一組是監控記憶體參數 T_m、U_m 與 LB_m，其分別代表記憶體總共的大小、目前的記憶體使用量與記憶體使用量的最低門檻值，第二組是監控電力參數 T_b、U_b 與 LB_b，其分別代表目前電力的百分比、目前的電力使用百分比與電力使用量的最低門檻值。我們會分別計算記憶體剩餘率 $R_m=(T_m-U_m)/T_m$ 與電力剩餘百分比 $R_b=(T_b-U_b)$，當 $R_m<LB_m$ 時表示記憶體不足，當 $R_b<LB_b$ 時表示電力不足，一旦發生資源不足時，系統會調整演算法細粒度 (Algorithm Granularity)。演算法細粒度調整是指降低演算法輸入時資料的詳細程度，以減少演算法執行時所耗費的資源。因此，當記憶體不足時我們藉由調整動態式摘要格子的細粒度與視窗大小，以減少記憶體的耗費。當電力不足但記憶體充足時，則調整視窗大小，以減少電力的消耗。表 1 為資源監控程序針對各種資源不足時所採取的控制策略。

表 1 資源控制策略

資源狀況	資源控制策略
記憶體不足，電力充足	以剩餘記憶體計算新的視窗大小，並調整動態式摘要格子的細粒度
電力不足，記憶體充足	以剩餘電力計算新的視窗大小
記憶體與電力皆不足	分別以剩餘記憶體與剩餘電力計算新的視窗大小，取其最小值作為新的視窗大小，並調整動態式摘要格子的細粒度

首先，針對移動視窗大小來做調整，當視窗大小很大時，所要處理的串流資料量也會相對的變多，計算上也會耗費較多的記憶體，且連線處理也須要較多的電力消耗。因此，當記憶體低於門檻值時，我們使用記憶體剩餘率 R_m 乘以視窗大小 w 來得到調整後的視窗大小，當 R_m 越小則視窗大小縮減越多；當電力低於門檻值時，我們使用電力剩餘百分比 R_b 乘以視窗大小 w 來得到調整後的視窗大小，當 R_b 越小則視窗大小縮減越多。當記憶體與電力皆不足時，將其分別計算出的視窗大小取最小值作為調整後的視窗大小。圖 6 為調整移動視窗大小範例，一開始使用者設定的視窗大小 w 為 6。在情境 1 記憶體的使用量 U_m 為 20，計算出來的 R_m 為 0.8。接著透過 $R_m \times w = 0.8 \times 6$ 得到新的視窗大小為 5，因此在情境 1 我們將視窗大小從 6 縮減為 5。而在情境 2 電力的使用量 U_b 為 30，計算出來的 R_b 為 0.7。接著透過 $R_b \times w = 0.7 \times 5$ 得到新的視窗大小為 4，因此在情境 2 我們將視窗大小從 5 縮減為 4。在情境 3 記憶體的使用量 U_m 為 22，計算出來的 R_m 為 0.78。接著透過 $R_m \times w = 0.78 \times 4$ 得到新的視窗大小為 4。電力的使用量 U_b 為 40，計算出來的 R_b 為 0.6。接著透過 $R_b \times w = 0.6 \times 4$ 得到新的視窗大小為 3，因此在情境 3 我們將視窗大小從 4 縮減為 3。

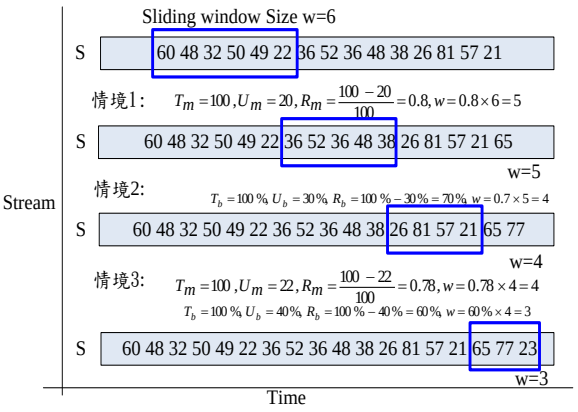

圖 6 動態式摘要格子

當記憶體不足時接著將目前動態式摘要格子的細粒度進行格子合併的處理來調整格子的細粒度大小，由於調整動態式摘要格子的細粒度會影響分群的精確度，所以只有當記憶體剩餘率 $R_m \leqq LB_m$ 時才會執行此程序。另外，當 R_m>80%代表記憶體充足且 R_b>80%電力也充足的狀況下，則會進行格子解析的處理。格子合併與格子解析詳細處理的步驟已在第 3.1.3 節闡述過。

3.2.2分群

分群有二個部份組成，第一個部份是分群程序，我們利用以密度為基礎的程序在格子上進行分群。第二個部份是資源監控程序，當資源不足時調整分群程序的疏稀格子密度門檻值，當電力危急時透過預估可執行的格子個數調整密度值低的格子個數，使探勘工作可以即時輸出分群結果。反之，當資源足夠時會恢復疏稀格子密度門檻值，以提昇探勘工作的準確度。當電力危急發生後，一旦電力恢復時，會將密度值低的格子個數再調整回來，以提昇分群結果的準確度。

在多維度空間，為了形成群集,我們考慮連接相鄰近格子，相關定義如下：

定義 1：相鄰格子(Neighbor Grids)：如果 2 個格子 $g_1 = (j_1^1, j_2^1, ..., j_d^1)$ 、$g_2 = (j_1^2, j_2^2, ..., j_d^2)$ 存在 k (1≦k≦d) 滿足以下條件 $\left| j_k^1 - j_k^2 \right| = 1$ ， $j_i^1 = j_i^2, i = 1, ..., k-1, k+1, ..., d$ ，則表示 g_1 和 g_2 在第 k 層維度是相鄰格子。

定義 2：格子密度 D(g)=n，n 表示對應到此格子內的資料個數。

定義 3：平均密度 D_a=1/G，G 表示格子的個數。格子平均密度的資料數量 N_a=N*D_a，N 為所有的資料總數。

定義 4：密集格子(Dense Grid)，當格子密度 D(g)≧使用者定義的密集格子的密度值 D_d*N_a 時，其為一個密集的格子，D_d≧1。

定義 5：稀疏格子(Sparse Grid)，當格子密度 D(g)＜使用者定義的密集格子的密度值 D_s*N_a 時，其為一個稀疏的格子，0＜D_s＜1。

定義 6：一般格子(normal Grid)，當格子密度介於密集格子與稀疏格子之間，其為一個一般格子。

定義 7：一組相鄰近的格子所形成的格子群集以 c 表示，$c=(g_1,g_2,\ldots,g_m)$。

定義 8：內部格子(Inside Grids)，在一個格子群集 $c=(g_1,g_2,\ldots,g_m)$，若 g 的每個維度的相鄰格子都屬於相同的格子群集 c，則 g 為此格子群組中的內部格子。

定義 9：外部格子(Outside Grid)，在一個格子群組 $c=(g_1,g_2,\ldots,g_m)$ ，若 g 存在一個維度的相鄰格子不屬於此格子群集 c，則 g 為此格子群集的外部格子。

定義 10：格子群集 c 是一群相鄰的格子所組成，初始時其內部格子都是密集格子所組成，並由這些格子往外擴增其群集的範圍。

分群程序的輸入為密集格子的密度值 D_d、稀疏格子的密度值 D_s、記憶體使用量最低邊界 LB_m、電力剩餘最低邊界 LB_b、電力危急剩餘最低邊界 DanLB_b以及目前動態式摘要格子所儲存的格子特徵向量 x_i，如圖 7 所示。分群程序的步驟分為三個部份。第一部分是計算初始時的門檻值(圖 7 的 line 4-8)，首先計算每個格子的平均資料數量 N_a，接下來計算密集格子的資料數量 $N_d=N_a\times$使用者輸入的密集格子的密度值 D_d，稀疏格子的資料數量門檻值 $N_s=N_a\times$使用者輸入的稀疏格子的密度值 D_s。第二部分是群集的產生(圖 7 的 line 10-25)。先將密集格子形成群集(圖 7 的 line 10-15)，分配每個密集格格不同的群集代號，若與密集格子相鄰的格子屬於不同的群集代號，則會將群集合併。接下來將每個群集往外擴散，找出與群集外圍格子相鄰的格子，若其格子的資料數量≦稀疏格子的資料數量門檻值 N_s，則刪除此格子，否則若格子不屬於其它群集則將其加入到此群集。第三部分是資源監控程序(圖 8)，透過計算記憶體剩餘率 R_m 與電力剩餘百分比 R_b，如果 $R_m \leqq LB_m$ 並且 $R_b > LB_b$ 表示記憶體不足但電力充足，則藉由增加 N_s 乘上記憶體剩餘率 R_m 之值來增加 N_s。當電力剩餘百分比 $R_b \leqq LB_b$ 並且 $R_m > LB_m$ 表示電力不足但記憶憶充足，則藉由增加 N_s 乘上電力剩餘百分比 R_m 之值來增加 N_s。如果 $R_m \leqq LB_m$ 並且 $R_b \leqq LB_B$ 表示記憶體與電力皆不足，根據目前記憶體剩餘率與電力剩餘百分比來調整 N_s，首先分別以記憶體剩餘率與電力剩餘率計算出新的 N_s，再取其最大值做為調整後的 N_s，N_s 越大則表示分群程序越容易將格子為離群值並丟棄，以減少記憶體的使用量，並且減少群集往外擴散的處理運算過程以節省電力的消耗。如果 $R_b \leqq DanLB_b$ 表示電力處於危急狀態，則計算目前尚未處理的格子數 nu←nt - np，並且預估剩餘的電力尚可執行的格子個數 n←$R_b \times (np / U_b)$ ，如果未處理的格子數 nu 比預估可執行的格子數多，則將密度值低的最後 nu- n 個移到暫不處理格子區塊。當電力危急發生後，一旦電力恢復至 80%時，系統會檢查暫不處理的格子區塊是否有格子，若有格子則會將暫不處理的格子移回須處理區塊。

分群程序的參數定義如下：

- D_d 為使用者定義的密集格子的密度值，D_s 為使用者定義的稀疏格子的密度值。
- LB_m 為使用者定義的記憶體使用量最低邊界，LB_b 為使用者定義的電力使用量百分比邊界，$DanLB_b$ 為使用者定義的電力使用量百分比危急邊界。
- $x=\{x_i \mid 1 \leq i \leq N\}$ 為目前動態式摘要格子所儲存的格子。Y 為暫不處理的格子。
- nt 為格子總數，np 為已處理完成的格子個數，nu 為尚未處理完成的格子個數，n 為預估可處理的格子個數。
- C 為分群程序最後產生的群集的集合。
- T_m 為記憶體總共的大小，T_b 為電力總共的百分比。
- N_a 為格子的平均資料數量，N_d 為密集格子的資料數量門檻值，N_s 為稀疏格子的資料數量門檻值。
- g 為群集的外部格子，h 為外圍格子的相鄰格子。
- c 為目前處理的格子的群集代號，c’為相鄰格子的群集代號。
- U_m 為記憶體使用量，U_b 為電力使用量百分比。
- R_m 為記憶體剩餘率，R_b 為電力剩餘百分比。

```
1.Procedure clustering(D_d , D_s ,LB_m, LB_b ,DanLB_b, x)
2.   compute  T_m ;
3.   compute  T_b ;
4.   nt←count(x );
5.   D_a←1/ nt;
6.   N_a←sum(x ) × Da;
7.   N_d←N_a × D_d ;
8.   N_s←N_a × D_s;
9.   sort x in descending order;
10.  For each dense grid x_i   do
11.     label x_i in new cluster ck;
12.     For each neighbor grid h of x_i do
13.        If (h is a dense grid ) then
14.           If (x_i belongs to cluster c) and
                 (h belongs to cluster c') then
15.              merge c & c';
16.For each cluster ci do
17.     For each Outside grid of ci do
18.        If ( g is a sparse grid) then delete g;
19.        Else
20.           For each neighbor grid h of g do
21.              If (h is not a sparse grid)
                    and (h.label ="no_class" ) then
22.                 label h.label=ci;
23.                 Outside grids←Outside grids∪h ;
24.                 check_resoure();
25.output C
26.end procedure
```

圖 7 分群程序

```
1.Procedure check_resource()
2.   compute U_m;
3.   compute U_b ;
4.   R_m←(T_m-U_m)/T_m;
5.   R_b←T_b-U_b;
6.   If (R_m ≦ LB_m) and (R_b > LB_b) then
7.      N_s ←N_s  + N_s × R_m ;
8.   If (R_b ≦LB_b ) and (R_m > LB_m) then
9.      N_s ←N_s  + N_s × R_b ;
10.  If (R_b ≦LB_b ) and (R_m≦LB_m) then
11.     N_s1  ←N_s + N_s × R_m ;
12.     N_s2  ←N_s + N_s × R_b ;
13.     N_s  ←Max(N_s1 , N_s2 );
14.  If (R_m > 80%) and (R_b > 80%) then
15.     N_s1  ←N_s  × (1- R_m );
16.     N_s2  ←N_s  × (1- R_b );
17.     N_s  ←Min(N_s1 , N_s2 );
18.  If (R_b ≦DanLB_b )   then
19.     nu  ←Count(x_i .label=" no_class")
20.     np←nt - nu;
21.     n←R_b × ( np/ U_b );
22.     If ( nu > n) then move (x,n+1, nu , Y) ;
23.  If (R_b > 80% ) and ( count(Y)>0) then
24.        move Y to x;
25.end procedure
```

圖 8 資源監控程序

以下是分群程序與資源監控程序詳細步驟說明：

Step 1(line 2-3)：透過系統內建函數來計算記憶體總共大小 T_m 與電力百分比 T_b。

Step 2(line 4-9)：計算密集格子與稀疏格子的資料數量門檻值。依格子密度由大至小排序摘要格子。

Step 3(line 10-15)：處理每個密集格子，分配每個密集格子新的群集代號，但是若密集格子的相鄰格子已有群集代號，則合併群集。

Step 4(line 16-24)：處理每個密集格子所形成的群集。

Step 4.1(line 17-24)：處理每個群集的外部格子，判斷與外部格子相鄰的格子，若相鄰格子的密度＜稀疏格子的資料數量門檻值，表示其為稀疏格子，則刪除此格子。若相鄰格子為非密集格子且尚未標示其群集代號，則將此相鄰的格子加入該群集，並且將此格子加入此群集的外部格子。

Step 4.2(line 24)：check_resoure 函數(圖8)是監控資源的狀態。其運作方式是計算記憶體剩餘率 R_m 與電力剩餘百分比 R_b (line2-3)，如果 $R_m \leqq LB_m$ 並且 $R_b > LB_b$ 表示記憶體不足但電力充足，則藉由增加 N_s 乘上記憶體剩餘率 R_m 之值來增加 N_s (line6-7)。當電力剩餘百分比 $R_b \leqq LB_b$ 並且 $R_m > LB_m$ 表示電力不足但記憶憶充足，則藉由增加 N_s 乘上電力剩餘百分比 R_b 之值來增加 N_s (line8-9)。如果 $R_m \leqq LB_m$ 並且 $R_b \leqq LB_b$ 表示記憶體與電力皆不足，則分別以記憶體剩餘率及電力剩餘百分比計算要增加的 N_s，然後再取其最大值做為 N_s 門檻值。當記憶體剩餘率與電力剩餘百分比越大則減少 N_s (line10-13)。如果 R_m ＞80%並且 R_b ＞80%表示記憶體與電力皆充足，則分別以記憶體剩餘率及電力剩餘百分比計算要減少的 N_s，然後再取其最小值做為 N_s 門檻值(line14-17)。如果 $R_b \leqq DanLB_b$ 表示電力處於危急狀態，則計算目前尚未處理的格子數 nu，並預估剩餘的電力尚可執行的格子數 n←R_b × (np / U_b) ，如果未處理的格子數 nu 比預估可執行的格子數多，則將密度值低的最後 nu- n 個格子移到暫不處

理區塊(line18-22)。如果電力危急發生後，使用者進行充電，一旦電力恢復至 80%時，系統會檢查暫不處理的格子區塊是否有格子，若有則會將暫不處理的格子移回須處理區塊(line23-24)。

Step 5：重覆 Step 4.1-4.2 直到此外部格子的相鄰格子皆處理完成。

Step 6：重覆 Step 4.1-5 直到此群集的外部格子皆處理完成，且沒有新的外部格子產生。

Step 7：重覆 Step 4-6 直到處理完成每個密集格子所形成的群集，結束分群。

舉例說明，首先我們設定 D_d=1.5、D_s=0.6、LB_m=30%、LB_b=40%與 $DanLB_b$=20%，並且目前階層式摘要格子的層級所儲存的格子有 16 個，其所含之資料筆數分別為 G11(2)、G12(5)、G13(5)、G14(5)、G21(4)、G22(5)、G23(8)、G24(5)、G31(2)、G32(2)、G33(1)、G34(2)、G41(5)、G42(7)、G43(5)、G44(3)。Step 1 計算記憶體大小為 40MB、目前電力大小為 90%。Step 2 計算每個格子的平均密度 N_a=(2+5+5+5+4+5+8+5+2+1+2+2+5+7+5+3)/16=4。Step 2.1 計算密集格子的資料筆數門檻值 N_d=N_a*D_d=4*1.5=6。Step 2.2 計算稀少格子的資料筆數門檻值 N_s=N_a*D_s=4*0.6=2。依格子密度由大至小排序分別為 G23、G42、G12、G13、G14、G22、G24、G41、G43、G21、G44、G34、G33、G32、G31、G11。Step 3 分配每個密集格子 G23 、G42 不同的群集代號，分別為 C_1、C_2。Step 4 先處理群集 C_1。Step 4.1 找出群集 C_1 中目前的外部格子 G23，與其相鄰的格子有 G13、G22、G23、G24 ，判斷 G23 為稀疏格子，將其移除，判斷 G13、G22、G24 為非密集格子且無群集標示，將其群集標示為 C_1 並將 G13、G22、G24 加入為 C_1 的外部格子 G23。Step 4.2 計算剩餘記憶體與電力剩餘百分比做為是否調整稀疏格子的門檻值。情節 1 目前記憶體使用量為 30MB，求出 R_m=(40-30)/40=0.25 ，電力使用百分比為 20% ，求出 R_b=0.9-0.2=0.7。由於 R_m < LB_m ，因此透過 2+2*(0.25)，將 N_s 增加為 3。情節 2 記憶體使用量為 18MB ，求出 R_m=(40-14)/40=0.55，電力使用百分比為 60% ，求出 R_b=90%-60%=30%。由於 R_b < LB_b ，因此透過 2+2*(0.3)，將 N_s 增加為 3。Step5 重覆 Step 4.1 處理群集 C_1 中與外圍格子相鄰的格子，有 G12、G14、G21 判斷其為非密集格子且無群集標示，將其群集標示為 C_1。Step 6 重覆 Step 4.1-5 直到外圍格子找不到相鄰的格子，或其相鄰格子已標示群集代號，產生群集 C_1=[G12、G13、G14、G21、G22、G23、G24]。重覆 Step 4 處理群集 C_2。Step 4.1 找出群集 C_2 中的外部格子 G42，與其相鄰的格子有 G41、G43，判斷其為非密集格子且無群集標示，將其群集標示為 C_2。Step5 重覆 Step 4.1 直到外圍格子找不到相鄰的格子，或其相鄰格子已標示群集代號。重覆 Step 4.2 計算剩餘記憶體與電力剩餘百分比做為是否調整稀疏格子的門檻值。若目前記憶體使用量為 35MB ，求出 R_m=(40-33)/40=0.175 ，由於 R_m < LB_m ，因此透過 3+3*(0.125)，得到 N_{s1} 為 4。目前電力使用百分比為 65% ，求出 R_b=0.9-0.65=0.25，由於 R_b < LB_b ，因此透過 3+3*(0.25)，得到 N_{s2} 為 4。將 N_s 取 N_{s1} 與 N_{s2} 的最大值得到 4。Step 7 重覆步驟 4.1-6，產生群集 C_2=[G41、G42、G43] 。處理完所有密集格子形成的群集輸出所產生的分群結果，結束處理程序。圖 9 為分群程序處理的過程，表 2 為產生的群集範例。

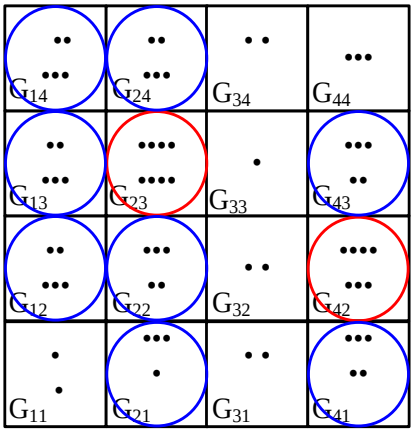

圖 9 分群範例

表 2 產生群集範例

Clusters	Dense Grids	Cluster Grids
C_1	G_{23}	$G_{13}, G_{22}, G_{24}, G_{14}, G_{12}, G_{21}$
C_2	G_{42}	G_{41}, G_{43}

4.效能評估

4.1 實驗環境與資料

由於本論文主要針對行動設備有限的資源下進行分群，為了能夠完整模擬行動設備的環境，因此實驗部分我們是在筆記型電腦上安裝 Android 所提供的手機模擬器(emulator)上來測試與比較效能，並且搭配 Eclipse 開發工具來撰寫程式。硬體部分使用 Pentium P6200 2.13GHz 處理器搭配 1GB 記憶體與 80GB 硬碟。

在真實資料集方面，我們使用美國加州大學爾灣分校(University of California at Irvine)的資訊電腦學院(Donald Brean School of Information and Computer Science)在 UCI Machine Learning Repository 網站上所提供的 Shuttle 資料集，其是與天文學有關的資料集並且其被許多資料串流分群的文獻[13]做為實驗的真實資料集。這個資料集包含 43500 筆資料且有 9 個屬性值。另外，為了使用各種資料點數量與維度的資料來進行實驗，我們利用 Microsoft SQL Server 2005 搭配 Microsoft Visual Studio Team Edition for Database Professional 來產生人造資料集，舉例來說，以 B100kC10D5 的表示方式是指此資料集包含 100k 個資料點、其每個資料點有 5 個維度，並且屬於 10 個不同的群集。

4.2 實驗與分析

由於 RA-Cluster [13]同樣是針對無所不在的資料串流環境的分群演算法，並且可以在有限的記憶體與電力資源下持續地進行資料串流分群，因此本實驗針對 RA-DCluster 與 RA-Cluster 的串流處理效率、精確度、記憶體使用量與電力使用量進行比較。

4.2.1 串流處理效率

首先針對串流資料處理能力進行實驗，資料來源為真實資料集。圖 10 與圖 11 顯示 RA-DCluster 與 RA-Cluster 的串流處理效率的比較，橫軸為資料處理所經過的時間(elapsed time)以秒為單位，而縱軸為每秒可處理的資料點數量。如圖 10 所示，RA-DCluster 的串流處理效率比 RA-Cluster 佳，因為 RA-DCluster 在處理資料串流抵達時只要將資料串流對應到相對應的格子，而 RA-Cluster 須要將資料串流進行微群集處理，因此 RA-DCluster 的處理效率比 RA-Cluster 佳。如圖 11 所示，在較長的執行時間下，RA-DCluster 的處理效率也比 RA-Cluster 佳。

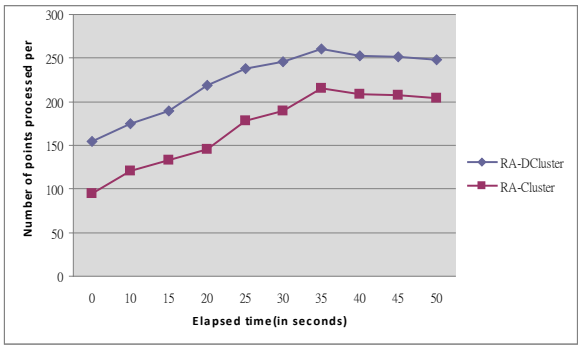

圖 10 串流處理效率比較

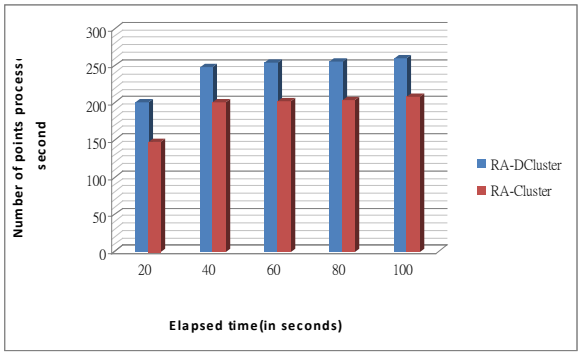

圖 11 在較長的執行時間下串流處理效率比較

4.2.2 精確度

在精確度評估的實驗中，我們使用平均距離變異總和(average of the sum of square distance, Average SSQ)[1]來衡量分群結果

的精確度。假設在目前時刻 Tc 前的期間 h 中，總共有 n 個資料點，對於在 h 的每一個資料點 n_i 找出距離其最近的群集中心 c_j 並計算 n_i 與 c_j 的距離 $d^2(n_i, c_j)$。因此目前時刻 Tc 前的期間 h 所計算出來的平均距離變異總和 Average SSQ(Tc,h)等於 h 中所有 n 個資料點與其群集中心之間的距離總和除以群集個數。Average SSQ 的值越小表示精確度越高。圖 12 使用真實資料集，圖 13 使用人工資料集其有 5% 的離群值(Outlier)。如圖 12 與圖 13 所示，可以看到在不同時間 RA-DCluster 的 SSQ 比 RA-Cluster 低，意味 RA-DCluster 獲得的群集，其群組內的資料比在 RA-Cluster 的更相似。

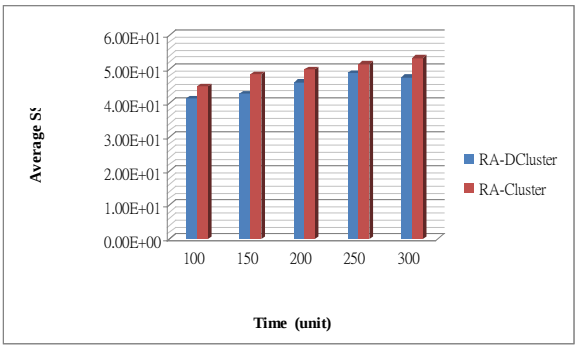

圖 12 探勘精確度比較(真實資料集)

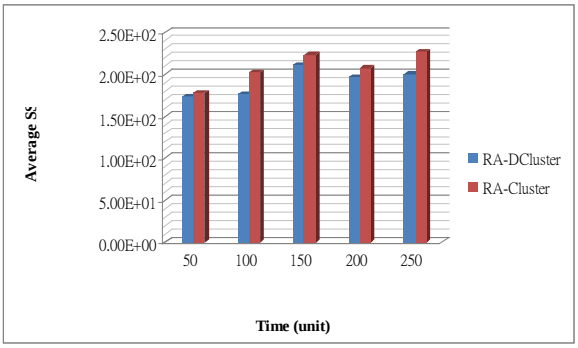

圖 13 探勘精確度比較(人工資料集)

4.2.3記憶體使用量

由於在行動設備中探勘串流資料的困難之一在於行動設備有限的記憶體，當記憶體不足時可能導致探勘中斷或失敗，因此我們藉由分析記憶體使用量來比較演算法持續探勘的能力。圖 14 顯示 RA-DCluster 、 RA-Cluster 與傳統的 K-Means 在執行分群時，記憶體使用量的比較，其橫軸為資料處理所經過的時間(elapsed time)以秒為單位，縱軸為剩餘的記憶體以 megabyte (MB)為單位。資料來源是使用人造資料集 B200kC10D5，其參數設定為 Dd=3、Ds=0.02 以及總共的記憶體大小為 100MB。如圖 14 所示，傳統的 K-Means 無法持續地探勘串流資料，並且當執行到 150 秒之後發生中斷。RA-Cluster 雖然能夠持續地進行探勘，但記憶體使用的起伏較大。相反地，RA-DCluster 能夠保持較少且穩定的記憶體使用量，其原因在於 RA-Clusters 是在固定的時間才會調整距離門檻值以釋放資源，而 RA-DCluster 則是在分群處理中就調整演算法的參數，使得探勘能夠穩定且持續。

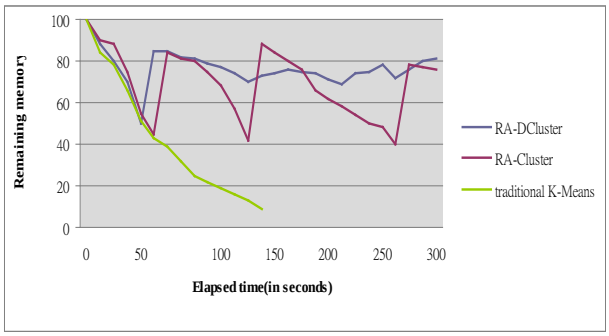

圖 14 記憶體使用量比較

4.2.4電力使用量

由於在行動設備中探勘串流資料的困難在於行動設備有限的電力，當電力不足時可能導致探勘中斷或失敗，因此我們藉由分析電力使用量來比較演算法持續探勘的能力。圖 15 顯示 RA-DCluster、RA-Cluster 與傳統的 K-Means 在執行資料串流分群時，電力使用量的比較，其橫軸為資料處理所經過的時間(elapsed time)以秒為單位，縱軸為剩餘的電力百分比。資料來源是使用人造資料集 B200kC10D5，其參數設定為 Dd=3、Ds=0.02。如圖 15 所示，傳統的 K-Means 無法持續地探勘串流資料，並且當執行到 1800 秒之後電力只剩不到 40%。RA-DCluster 比 RA-Clusters 能

夠使用較少的電力，其原因在於 RA-DCluster 除了調整輸入視窗大小並且在分群處理中調整演算法的參數，使得探勘能夠更減少電力消耗。

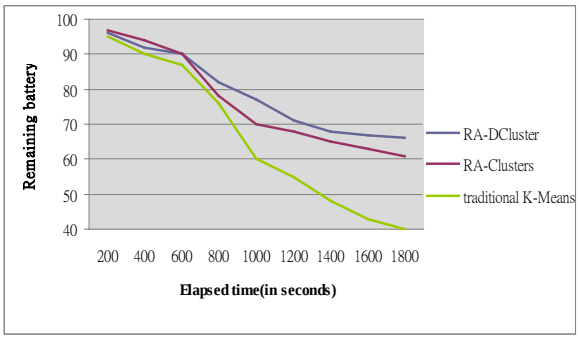

圖 15 電力使用量比較

5.結論

在行動手持設備越來越普及的情況下，如何有效地利用其有限的資源來探勘資料串流是很重要的議題。因此，本論文針對無所不在的資料串流環境提出 RA-DCluster 演算法，其利用資源感知技術調整演算法的參數與動態式摘要格子的細粒度，使行動設備能夠在有限的記憶體與電力下持續地進行分群。實驗結果證明 RA-DCluster 處理資料串流的效率高於 RA-Cluster 而且能夠保持較少且穩定的記憶體使用量，並且更節省電力的消耗。

參考文獻

[1]C.C. Aggarwal, J. Han, J. Wang, and P.S. Yu, "A Framework for Clustering Evolving Data Streams," *Proc. of the 29th International Conference on Very Large Data Bases*, Berlin, Germany, September 2003, pp. 81-92.

[2]B. Babcock, S. Babu, R. Motwani, and J. Widom, "Models and issues in data stream systems," *Proc. of the 21st ACM SIGMOD Symposium on Principles of Database Systems*, Madison, Wisconsin, U.S.A., June 2002, pp.1-16.

[3]F. Cao, M. Ester, W. Qian, and A. Zhou, "Density-Based Clustering over an Evolving Data Stream with Noise," *Proc. of the 6th SIAM International Conference on Data Mining*, Bethesda, 2006, pp. 328-339.

[4]Y. Chen and L. Tu , "Density-Based Clustering for Real-Time Stream Data," *Proc. of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, San Jose, California, USA, August 2007, pp. 133-142.

[5]P. Domingos and G. Hulten, "A General Method for Scaling Up Machine Learning Algorithms and its Application to Clustering," *Proc. of the 18th International Conference on Machine Learning*, Williamstown, M.A., U.S.A., July 2001, pp. 106-113.

[6]M. M. Gaber, Sh. Krishnaswamy, and A. Zaslavsky, "Cost-Efficient Mining Techniques for Data Streams," *Proc. of the 2004 Workshop on Data Mining and Web Intelligence (DMWI2004)*, Dunedin, New Zealand. CRPIT, 32. Purvis, M., Ed. ACS, 2004, pp.109-114.

[7]M. M. Gaber, Sh. Krishnaswamy, and A. Zaslavsky, "Adaptive Mining Techniques for Data Streams using Algorithm Output Granularity," *Proc. of the 2nd Australasian Data Mining Workshop*, Canberra, Australia, December 2003.

[8]M. M. Gaber and P.S. Yu, "A Framework for Resource-aware Knowledge Discovery in Data Streams: A Holistic Approach with Its Application to Clustering," *Proc. of the 2006 ACM Symposium on Applied Computing*, Dijon, France, April 2006, pp.649-656.

[9]M. M. Gaber, A. Zaslavsky, and S. Krishnaswamy, "Towards an Adaptive Approach for Mining Data Streams in Resource Constrained Environments," *Proc. of the International conference on data warehousing and knowledge discovery*, Vol. 3181, Zaragoza, Espagne, September 2004, pp. 189-198.

[10]M. M. Gaber, S. Krishnaswamy, and A. Zaslavsky, "Ubiquitous Data Stream Mining," *Proc. of the 8th Pacific-Asia Conference on Knowledge Discovery and*

Data Mining, Sydney, Australia, May 2004.

[11]O. Horovitz, S. Krishnaswamy, and M.M. Gaber, “A Fuzzy Approach for Interpretation of Ubiquitous Data Stream Clustering and its Application in Road Safety,” *The Journal of Intelligent Data Analysis*, Vo11, No. 1, January 2007, pp.89-108.

[12]H. Kargupta, B.H. Park, S. Pittie, L. Liu, D. Kushraj, and K. Sarkar, “MobiMine: Monitoring the Stock Market from a PDA,” *ACM SIGKDD Explorations Newsletter*, Vol. 3, No. 2, December 2002, pp. 37-46.

[13]N. H. Park and W. S. Lee, “Statistical Grid-based Clustering over data streams,” *ACM SIGMOD Record*, Volume 33, Issue 1, 2004, pp. 32-37.

[14]N. H. Park and W. S. Lee, “Grid-based Subspace Clustering over Data Streams,” *Proc. of the 16th ACM Conference on Conference on information and Knowledge Management*, New York, 2007, pp.801-810.

[15]J. Ren and R. Ma, “Density-Based Data Streams Clustering over Sliding Windows,” *Proc. of the 6th International Conference on Fuzzy Systems and Knowledge Discovery*, Washington, US, 2009, pp.248-252.

[16]R. Shah, S. Krishnaswamy, and M. M. Gaber, “Resource-Aware Very Fast K-Means for Ubiquitous Data Stream Mining,” *Proc. of 2nd International Workshop on Knowledge Discovery in Data Streams*, Porto, Portugal, October 2005, pp.40-50.