感測節點上的混合式任務排程機制

郭錦福[1]　盧永豐[2]
[1]國立高雄大學 資訊工程學系
[2]國立臺中科技大學 資訊工程學系

摘要

物聯網(Internet of things)中可能具備了許多感測器，這些感測節點的系統中會需要週期性的執行某些任務，與非週期性的執行使用者查詢或是緊急狀況反應，如何以節約能源方式執行系統中的工作，且能夠儘早完成非週期性工作，是一個重要的研究議題。由於系統需要有即時性的需求，因此原本在可排程測試上是利用最大執行時間來做分析。但是當工作實際在執行時，可會提早結束而留下未執行時間，因此本研究將提出一個排程機制讓未來即將執行的工作能有效利用這些未執行時間，使得系統耗能與非週期性工作的平均反應時間能夠都被最小化。由於最小化系統耗能與非週期性工作的平均反應時間有互斥，因此方法的效能評估準則為這兩個值的乘積，不應偏頗任何一方。本篇論文中提出一基於使用者自訂回收比例的演算法(Ratio Reclaim Algorithm)，將未被使用的執行時間依此比例分配給週期性任務的工作來使用，進而減速來減少電源的消耗並改善非週期性工作的反應時間。為了能方便並有效地管理未被使用的執行時間，也提出了一個佇列結構(earliness-queue)，來減少維護未被使用的執行時間之記憶體空間。最後以實驗的方式，來證實我們所提出的演算法，以不同觀點來討論使用者自訂比例對於效能的影響，可以藉此實驗結果來推測在給定的系統負載下，挑選出較合適的比例值，使系統執行任務時能有較佳的效能。

關鍵詞：感測節點、混合式任務、電耗、反應時間、未使用之執行時間、比例。

Abstract

Internet of things consists of many components, including sensor nodes. In a sensor node, there could be a mixed task set where periodic tasks are responsible to do regular operations and aperiodic jobs send warning messages to the sink node when something urgent occurs. This paper aims to investigate the scheduling problem of a mixed task set in such a sensor node. When a mixed task set is executed, a scheduling algorithm has to consider not only the system energy consumption but also the response time of the aperiodic jobs which do not want to be prolonged seriously. This paper proposes the Ratio Reclaim Algorithm (RRA) that reclaims the unused execution time for periodic jobs based on a user-defined ratio. For the convenience of maintaining effectively unused execution time left by early completed jobs, the earliness-queue is presented to reduce the memory space that the earliness-queue occupies. A series of experiments are conducted to evaluate the proposed algorithm. The experimental results demonstrate that the RRA scheme with a suitable value of the user-defined variable has a better performance compared with the Mutual Reclaiming Algorithm.

Keywords: Sensor Nodes, Mixed Task Set, Energy, Response Time, Unused Execution Time, Ratio.

1. 前言

隨著資訊技術的演進，許多各式各樣的新資訊設備不斷被開發出來。利用先進的硬體電路技術整合了處理器計算能力與無線通訊能力，更使得各種新型態的感測設備與控制結構得以發揮更廣闊的影響力創造出更多的應用。在物聯網(Internet of things)中，原本分散在各處的嵌入式系統更可透過無線網路相連並共同去完成一些任務。感測節點(sensor node)扮演著非常重的角色，要將環境與系統相連，並進一步利用運作元件(actuators)讓系統能與環境互動。分散在各處的感測節點大多是利用電池來提供電能執行或處在可能不是時時刻刻都插在插座上的環境，系統中除了週期性任務(如：資料收集與回傳)要週期執行外，可能還要執行有不定時的非週期性工作(如：使用者欲查詢目前節點狀態料)。因此系統如何以較省電方式來執行工作，進而延長感測節點的運作時間(operation time)，還要能盡快完成非週期性工作，是一個已被重視的研究議題。

隨著 VLSI 技術的重要進步，嵌入式系統領域已經趨向利用新的整合性電晶體來節省電能之消耗。由於嵌入式裝置大部分是以電池來供應執行的電能，為了能夠延長電池使用壽命，動態電壓調變技術(Dynamic Voltage Scaling：DVS)被用在現代化的處理器上，用來依據負載(workload)狀況來以不同的頻率(/電壓) 執行[4]。感應節點上的工作很多是為混合式任務模式，不僅有週期性任務，亦有非週期性工作。週期性任務週期性的收集週遭環境的資訊，而非週期性工作則如回應系統使用者不定時的查詢需求。所以本研究計劃要探討的問題是，針對感應節點上的週期性任務與非週期性工作提出排程機制以 DVS 來改善系統總耗能與非週期性工作

的平均反應時間。

和本研究相關的研究主要是 Aydin 等人[1]所提出的 *Mutual Reclaim Algorithm* (MRA)方法，此方法讓即將執行的工作獲得所有可使用的未使用之執行時間。此外，並提出以侵佔式方式將非週期性工作未使用之能力先給週期性工作使用，該研究式利用[2]文獻中所提出的 α-queue 資料結構來紀錄未使用之執行時間。和該研究不同的是，本研究不讓即將執行的工作拿取所有的未執行時間，希望能預留部分的未執行時間給系統中正在等待執行的非週期性工作，此外，我們也將提出一個更有效能維護成本的資料結構來維護工作提早結束時所遺留下來的未使用之執行時間。

需要考慮到非週期性工作的時間行為特性，使得本研究議題的有其困難度。我們先考慮非週期性工作利用 Total Bandwidth Sever (TB Server) [3]的機制來處理，利用此機制來設定非週期性工作的截限時間，然後和週期性工作以 Earliest Deadline First [7]來排程。當系統執行時，由於週期性或是非週期性工作皆有可能提早結束，不會完全使用完原先所分配到的時間，在工作提早結束後，如何去有效應用這些剩餘的執行時間在之後的週期性或是非週期性工作上，是一個需要取得平衡的問題，因為如果偏頗週期性工作，將導致非週期性工作的反應變差。相對的，如果偏向給予非週期性工作，則週期性工作的耗電量會增加。我們針對未使用時間並考量到非週期性工作的特性，提出 Ratio Reclaim Algorithm，此演算法可以依據使用者參數來給予週期性與非週期工作未使用的執行時間的比例，此方法。另外，為能有效率地維護提早完成工作所遺留下可用來降速的未使用執行時間，也提出了新的結構 *earliness*-queue 與相對應的維護規則。我們所提出的方法讓系統設計者保留了欲給週期性工作的未使用之執行時間比例，藉此來減少對於非週期性工作的影響。

本論文將在後續的章節，進行以下安排：第二章將定義系統模式與問題，第三章說明我們所提出的機制，第四章呈現與討論方法的效能，最後於第五章進行總結。

2. 系統模式與問題定義

許多感測節點中的處理器具有DVS機制，處理器並不是任意速度皆能運作，換句話說，處理器只能在某些允許的速度下運作[2]，我們假設處理器的可運作速度$\{S_{min}, S_2, .., S_{max}\}$，另外也假設處理器在切換速度時的時間與電耗成本可以被忽略。在速度S下的每單位時間的耗電量函式以$p(S)$表示，這是一個strictly convex函式，且與速度成最少二次方關係[5, 6]。如果一個工作在時間區間$[t_1, t_2]$內使用這個處理器，則總電耗E為$\int_{t_1}^{t_2} p(S(t))dt$。在本研究中，我們只考慮處理器的耗能。

本研究中所要探討感測節點中有一組動態混合模式的任務集合，包含N個週期性任務(Task)$\{T_1, T_2, ..., T_n\}$和一個非週期性工作集合$\{J_1, J_2, ...\}$。$T_{i,j}$為週期性任務T_i的第j個工作(Job)。此外，與一組在執行期間隨時都有可能提出執行需求的非週期性任務。每個週期性任務T_i以(p_i,c_i)代表，參數說明如下：

(1) 週期(Period) p_i (在本研究中，我們假設截限時間 (Deadline) $d_i = p_i$)
(2) 在最高速度下的最大執行時間(worst-case execution time: WCET) e_i

至於每個非週期性工作 J_k 以(r_k, e_k)代表，參數說明如下：

(1) 抵達時間(Arrival time) r_k
(2) 最大執行時間 e_k

需強調一點，由於工作在實際執行時，可能不會真的使用最大執行時間，所以實際執行時間可能會比最大執行時間小。

問題的定義：

本研究中所要探討的問題，是在一個感測節點上，除有週期性任務外，還有不定時出現的非週期性工作的執行需求，執行過程中需要的處理時間且系統需要消耗電能，此外，兩種工作在執行時都可能提早結束。如何在這種動態變動的系統上，提出一個排程機制兼具考慮到系統消耗電能與非週期性工作的反應性，是本研究要探討的問題。我們將提出一個方法能夠有效使用未使用的執行時間，使得系統耗能與平均反應時間的乘積最小化。

3. 考慮耗能與反應時間平衡的混合式任務排程機制

在本節中，我們將說明所提出的 Ratio Reclaim Algorithm (RRA)方法的原理，此外，並且提出一個維護未使用執行時間的資料結構 *earliness*-queue。和相關文獻中的 α-queue[2]比較，我們所提出的 *earliness*-queue 的維護成本較低，因為我們紀錄較少的資訊。利用 *earliness*-queue 與使用者定義參數可以有效為下一個即將執行的工作取得未使用執行時間。為了說明 *earliness*-queue，我們先定義何謂系統的 ready queue，當一個工作 $T_{i,j}$ ($/J_k$)抵達系統時[1]，會有一個相對應的紀錄(record)被加入ready-queue 中。而工作完成後，這個紀錄會被移除。這個紀錄中包含了以下欄位：編號(identity)、絕對(/虛擬)截線時間(deadline)、在速度 $S'_{i,j}$ ($/S'_k$) 下的剩餘最大執行時間(remaining worst-case execution time) $w_{i,j}^{S'_{i,j}}$ ($/w_k^{S'_k}$)、和剩餘虛擬執行時間(remaining virtual execution time) $rem_{i,j}^a$ ($/rem_k^a$)。當工作抵達時，$rem_{i,j}^a$ ($/rem_k^a$)被設定成 $w_{i,j}^{S'_{i,j}}$ ($/w_k^{S'k}$)，而

[1] 在後面章節中，我們將使用符號$X_{i,j}$ 與 Y_k來表示週期性工作$T_{i,j}$與非週期性工作J_k的屬性。

$w_{i,j}^{S'_{i,j}}$ ($/w_k^{S'k}$)將隨著未執行時間 earliness 而改變。當工作 $T_{i,j}$ ($/J_k$)被執行時，剩餘最大執行時間與剩餘虛擬執行時間的值將會隨著工作的執行而遞減。此外，當即將執行的工作獲得之前工作未使用的執行時間時，這兩個欄位也會被調整，細節將在後面說明。*earliness*-queue 為每個提早完成的工作 $T_{i,j}$($/J_k$)維護了一個相對應的紀錄，紀錄中有以下欄位：編號(identity)、絕對(/虛擬)截線時間(deadline)、和未使用執行時間(earliness) $ear_{i,j}$ ($/ear_k$)。為了方便辨識，我們以 $ear_{i,j}$ ($/ear_k$)來表示 *earliness*-queue 中紀錄的未使用執行時間。

接著，我們將介紹 *earliness*-queue 的規則：

1. 將*earliness*-queue初始化呈一個空的連結串列(linked list)
2. 當週期性工作$T_{i,j}$或是非週期性工作J_k 提早完成時，新的紀錄將被加到*earliness*-queue中，且$ear_{i,j}$ ($/ear_k$) 欄位的值被設定成ready queue中該工作的 $rem_{i,j}^a$ ($/rem_k^a$) 欄位值，*earliness*-queue中的紀錄是根據絕對(/虛擬)截線時間(absolute (/virtual) deadlines)排序。
3. 當下一個工作$T_{i,j}$ ($/J_k$)即將被執行時:

3-1. *earliness*-queue將被參考來為具有最近截限時間的工作$T_{i,j}$ ($/J_k$)計算可以給予它的之前工作未使用的執行時間。 之前提早完成工作的未使用執行時間將依據其截限時間來給下一個即將執行之工作使用，可以以下式子來表示：

$$\varepsilon = \sum_{h|d_h \le d_{i,j}(/d_k)} ear_h \qquad (1)$$

換言之，*earliness*-queue中每一個紀錄中的截限時間d_h 小於等於即將執行工作$T_{i,j}$($/J_k$)的截限時間 $d_{i,j}$ ($/d_k$)即可以提供它所記載的未使用執行時間(earliness)。這些記載的earliness $ear_{x,y}$ ($/ear_z$)將被累加到$T_{i,j}$($/J_k$)的$rem_{i,j}^a$ ($/rem_k^a$)欄位值中。如此一來，就可讓 $T_{i,j}$ ($/J_k$)使用過去提早執行結束工作$T_{x,y}$ ($/J_z$)的earliness。當*earliness*-queue中的紀錄提供工作$T_{x,y}$ ($/J_z$)的earliness後，此紀錄將會被移除。

3-2. 如果即將執行的工作屬於某個週期性任務的工作，且ready queue中存在著非週期性工作，則我們提出以使用者參數R來控制最後實際給予該週期性工作的earliness，earliness將被設定成原計算出的earliness與R的乘積。若ready queue無非週期性工作，則原計算出的所有earliness將給予此即將執行的週期性工作。

4. 當系統無任何工作可以執行時，*earliness*-queue中的第一個紀錄中的 $ear_{i,j}$ (=ear_k) 值將隨著時間的流失而遞減。當 $ear_{i,j}$ ($/ear_k$)值變為 0 時，該筆紀錄則將被移除。

接下來，我們將說明我們所提出的排程演算法 Ratio Reclaim Algorithm (RRA)，此演算法考量到系統耗能與非週期性工作的平均反應時間間的平衡。我們利用之前所介紹的 *earliness*-queue 來完成此演算法，如圖一所示。RRA 能夠回收提早完成工作未使用的執行時間來分配給隨後即將執行的週期性或是非週期性工作。因此，對於一個混合式任務而言，RRA 可以最小化系統耗能與非週期性工作的平均反應時間的乘積。

RRA的輸入為一個週期性任務集合T = {T_1, T_2, …, T_n}、一個非週期性工作集合{J_1, J_2, …}、與使用者參數R。演算法的第1、2行為初始化變數，與將*earliness*-queue和ready queue初始化成空的連結串列。演算法執行時間clockmax為週期性任務週期的最小公倍數。

```
Algorithm 1 Ratio Reclaim Algorithm (mixed task set,R)
Input: {T1, T2, T3, ,Tn}, {J1, J2, …}, R
Output:E * average response time
1: 初始化 E ,rtot, clock, 和numberOfAp為 0;
2: 初始化earliness-queue和ready queue為空的連結串列 ;
3: while clock < clockmax do
4:    if 工作J抵達且為非週期性工作 then
5:        將 J的訊息加入ready queue;
6:        依據 TB server的Rule 2來設定J的截限時間 [3];
7:       設定預設速度Sk = 1;
8:   else if 工作J抵達且為週期性任務Ti工作 then
9:       將 J的訊息加入ready queue;
10:      設定截限時間為clock + Pi ;
11:      設定預設速度Si,j = 1;
12:  end if
13: // 當工作J即將被執行:
14:  if ready queue != NULL then
15:      設定J為ready queue中有最近截限時間的工作 ;
16:      if J是週期性工作 then
17:              z= Reclaim(J,R) ;
18:              if z > 0 then
19:                  利用z來降低速度，將速度Si,j =max(w^Si,j_i,j/(z+w^Si,j) * Si,j ,Smin);
20:                  將剩餘最大執行時間調整為J.w^Si,j_i,j =J.w^Si,j_i,j + earliness;
21:              end if
22:              以速度Si,j 執行J一個單位時間;
23:              J.w^Si,j_i,j --;
24:              J.rem^a_i,j--;
25:              E = E + e(Si,j);
26:              if J完成且J.rem^a_i,j> 0 then
27:                  在earliness-queue中加入關於J的新紀錄(將J.rem^a_i,j 設定成J.eari,j )，並刪除ready queue中的J的紀錄;
29:              end if
30:      else
31:          // J is an aperiodic job to be executed :
32:          Reclaim(J,R) ;
33:          以速度Sk 執行J一個單位時間;
34:              J.w^Sk_k --;
35:              J.rem^a_k--;
36:              E = E + emax;
37:          if J完成且J.rem^a_k> 0 then
38:              在earliness-queue中加入關於J的新紀錄(將J.rem^a_k 設定成J.eark )，並刪除ready queue中的J的紀錄;
39:              rtot = rtot + (clock-J.ready_time);
40:              numberOfAp++;
41:              依據 TB server的Rule 3來設定ready-queue中有最小抵達時間非週期性工作的截限時間 [3];
42:          end if
43:      end if
44:  else
45:      由於系統中無工作執行，所以處理器以最低速度Smin執行一個單位時間 ;
46:      E = E + emin;
47:      if earliness-queue != NULL then
48:          將earliness-queue中有最小截限時間的紀錄之ear_i,j (/eark)欄位減1;
49:      end if
50:  end if
51:      clock++;
52: end while
53:  average_ response = rtot/numberOfAp;
54: return E*average_response;
```

圖 1: Ratio Reclaim Algorithm

第3-52行為主要的while迴圈，每單位時間執行一個迴圈循環，內部可以分成兩個主要區塊：第一個區塊在第 4-12行，主要是在計算與記錄每個抵達工作的絕對(/虛擬)截限時間，然後以Earliest Deadline First 排程機制[7]來排程工作。每個工作抵達系統後的欲設速度皆為1.0 (S_{max})，而非週期性工

作的虛擬截限時間則依據Total Bandwidth (TB) server 規則來設定[3]，某工作J_k的虛擬截限時間d_k的訂定要符合下面說明：當第k的非週期性工作r_k時間抵達時，虛擬截限時間$d_k = \max(r_k, d_{k-1})+c_k/u_s$，$c_k$為$J_k$的最大執行時間，而$u_s$為TB server的能力(Size)。此外，$d_0$ 應被設成0。

而第二個區塊位於第14-43行，主要是控制週期性與非週期性工作的執行，如果下一個有最小截限時間即將執行的工作J為週期性工作，圖2中的Reclaim函式將被呼叫來計算出此工作可用的earliness 數量，然後計算出新的速度$S_{i,j}$，並修改剩餘對大執行時間(remaining worst-case execution time) $w_{i,j}^{S_{i,j}}$。我們以Martin等人[8]所提出論文中的每單位時間的耗能函式$p(S) = S^3$來計算能量消耗，所以在某個時間區間$[t,t+1]$ 的耗能為$e(S_{i,j})$，$e(S_{i,j}) = \int_t^{t+1} p(S_{i,j})dt$。如果$J$為非週期性工作，RRA演算法也會呼叫Reclaim函式，但為了要改善非週期性工作的反應時間，所以會以 S_{max} 速度執行。而單位時間耗能e_{max}為1.0。當工作J完成時，非週期性工作的反應時間將被累加到r_{tot}，之後將除以所完成的非週期性工作個數，以求得平均反應時間。如果該非週期姓工作完成後，系統ready queue中還有其他的非週期性工作，則具有最小抵達時間的非週期性工作的虛擬截限時間將依據TB server的規則來設定[3]。當工作完成且有提早結束時，相對紀錄將被加到*earliness*-queue中，新紀錄中的欄位$ear_{i,j}$ ($/ear_k$)數值將以工作的$rem^a_{i,j}$ ($/rem^a_k$)值設定。

```
函式 Reclaim(J,R)
Input:J, R
Output:e
1:  e = 0;
2:  limit= J.deadline -clock-J.w^Si,j_i,j (/J.w^Sk_k);
3:  if the earliness-queue != NULL then
4:        設定X為earliness-queue中的第一個紀錄;
5:        while ( X!= NULL) and (e <limit) do
6:              if X.deadline<=J.deadline then
7:                    z= z + X.ear;
8:                    if e >limit then
9:                          X.ear = z-limit;
10:                         z =z-X.ear;
11:                         break;
12:                   end if
13:                   刪除X，將earliness-queue的第一個設定為X;
14:             end if
15:       end while
16:       J.rem^a_i,j = e + J.w^Si,j_i,j (/J.rem^a_k = e + J.w^Sk_k) ;
17:       if J為週期性工作且ready queue中有其他非週期性工作 then
18:             z= z* R;
19:       end if
20: end if
21: return e;
```

圖 2: Reclaim 函式

當系統閒置時，處理器將以最低速度S_{min}執行，耗能為e_{min}。此外，如果此時*earliness*-queue中有紀錄，其中的第一個紀錄的$ear_{i,j}$ ($/ear_k$)欄位將減少一個單位時間。我們將在下一節中展示所提出的RRA演算法如何改善系統耗能與平均反應時間的乘積。圖2中的Reclaim函式被RRA演算法用來為即將執行的工作計算earliness，此函式是式子(1)的實作，由於當回收使用的earliness過多時將會導致將執行或是未來的週期性工作會無法在截限時間前完成所有執行，因此需要做限制。此外，在該函式中會將下一個即將執行的工作$T_{i,j}$ ($/J_k$)的剩餘虛擬執行時間(remaining virtual execution time) $rem^a_{i,j}$($/rem^a_k$)將以剩餘最大執行時間(remaining worst-case execution time)與所計算出的earliness z的總和做修改。需要強調的是，如果系統的ready queue中還有其他非週期性工作時，實際上回傳的earliness將會是$z*R$。由於剩餘虛擬執行時間(remaining virtual execution time) 已更新，而剩餘最大執行時間(remaining worst-case execution time)是在圖1中的RRA演算法中更新，因此有可能在某個時間，同一個工作的這兩個欄位的值不同。

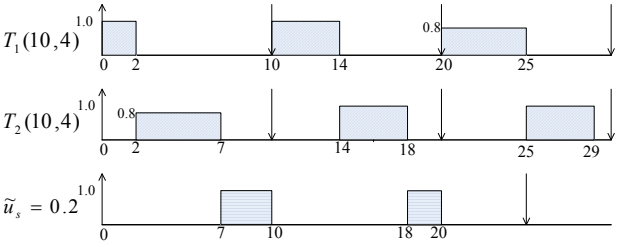

圖 3:任務的執行甘特圖

為了能夠更具體說明我們所提出的RRA演算法，我們將以一個實例來說明，*earliness*-queue與α-queue的狀況將被呈現以比較兩者間的維護成本差異。此混合模式任務中有二個週期性任務$T_1 = (10, 4)$與$T_2 = (10, 4)$，週期與最大執行時間分別是10和4，抵達時間皆為0。此外，還有一個非週期性工作$J_1 = (0,5)$，抵達時間與最大執行時間分別為0與5。我們假設TB server的能力(Size)u_s為0.2而使用者定義變數R值為0.5。執行甘特圖如圖3所示，而*earliness*-queue與α-queue的狀況如表1所示，α-queue相關的實作方式請參考[2]。當$T_{1,1}$時間2提早結束執行時，一個相對應的紀錄將被加到*earliness*-queue中，記錄中$ear_{1,1}$欄位值將被設定成$T_{1,1}$的目前$rem^a_{1,1}$的值，也就是2。同時，因為$T_{2,1}$為系統中有最高優先權的工作，為下一個執行的工作，因此，$T_{2,1}$的$rem^a_{2,1}$欄位值將依據*earliness*-queue規則3-1被設定成6 ($=ear_{1,1}+ w_{2,1}^{S_{2,1}}$ =2+4)。但是由於規則3-2的關係，最後實際給予$T_{2,1}$的earliness只有1 (=2*0.5)，之後此紀錄將由*earliness*-queue 被移除，而$T_{2,1}$的速度與 $w_{2,1}^{S_{2,1}}$ 將分別被設定成0.8 (=4/4+1) 和5 (=4+1)。在時間點7時，$T_{2,1}$完成，而其$rem^a_{2,1}$不為0，因此新的紀錄被加到*earliness*-queue，$rem^a_{2,1}$的值1將被用來設定$ear_{2,1}$。同時，J_1將被執行，所以它的rem^a_1欄位值將被設為6($=ear_{2,1}+ w_1^{S_1}$ =1+5)，相對紀錄將從*earliness*-queue中被移除。之後，由於*earliness*-queue沒有任何紀錄，所以$T_{1,2}$與$T_{2,1}$直接執行。J_1在時間20完成執行時，J_1的rem^a_1值為1，因此

又有相對的紀錄被加到*earliness*-queue，ear_1被設定成1。當$T_{1,3}$即將執行時，因為系統的ready queue中已無非週期性工作，所以從*earliness*-queue中獲得1單位的earliness，而該筆J_1的紀錄也隨之被刪除。從表1中，我們可以發現*earliness*-queue可以以較少的維護資訊完成α-queue[2]要達成的目的。

時間	事件	*earliness*-queue= {(ID,$d_{i,j}$(/d_k), $ear_{i,j}$(/ear_k))}	α-queue= {(ID,$r_{i,j}$(/r_k),$d_{i,j}$(/d_k),$rem_{i,j}$(/rem_k))}
0	$T_{1,1}$, $T_{2,1}$和 J_1抵達	∅	{($T_{1,1}$,0,10,4),($T_{2,1}$,0,10,4),(J_1,0,25,5)}
2	$T_{1,1}$ 提早完成	{($T_{1,1}$,10,2)}	{($T_{1,1}$,0,10,2),($T_{2,1}$,0,10,4),(J_1,0,25,5)}
7	$T_{2,1}$ 完成	{($T_{2,1}$,10,1)}	{($T_{2,1}$,0,10,1),(J_1,0,25,5)}
10	$T_{1,2}$和 $T_{2,2}$抵達	∅	{($T_{1,2}$,10,20,4),($T_{2,2}$,10,20,4),(J_1,0,25,3)}
14	$T_{1,2}$完成	∅	{($T_{2,2}$,10,20,4),(J_1,0,25,3)}
18	$T_{2,2}$完成	∅	{(J_1,0,25,3)}
20	J_1完成	{(J_1,25,1)}	{(J_1,0,25,1)}
20	$T_{1,3}$和 $T_{2,3}$抵達	∅	{(J_1,0,25,1),($T_{1,3}$,20,30,4),($T_{2,3}$,20,30,4)}
25	$T_{1,3}$完成	∅	{($T_{2,3}$,20,30,4)}
29	$T_{2,3}$完成	∅	∅

表 1: *earliness*-queue 與 α-queue 的狀況

4. 效能分析

4.1 實驗設定

我們以模擬實驗驗證所提出的RRA方法在不同*R*值情況下的效能，*R*值設定為10%到100%之間，效能比較對象是MRA[1]，MRA為RRA的使用者參數*R*為0%的特例。我們設計了一個任務產生器來產生任務，實驗的最大系統負載U_e為週期性任務的總使用率U_p與非週期性工作負載U_a的總和。在實驗中，我們將最大系統負載U_e設定為1.3，而U_p則在0.1與0.9之間變化，因此，U_a則被設定(1.3-U_p)，為不影響週期性工作的即時特性需求，Total Bandwidth Server的能力u_s被設定為(1.0-U_p)。週期性任務個數設定為10，任務週期則為[50,400]之間的隨機變數。處理器的最小速度S_{min}設為0.1。任務T_i在最大速度S_{max}下的最大執行時間則為使用率u_i乘以週期P_i，即為u_i *P_i。而相鄰的兩個非週期性工作的抵達時間間格符合平均值為I_a的卜瓦松變數(Poisson probability distribution)[1]。I_a被設定為所有週期性任務的週期平均值，非週期性工作在最大速度S_{max}下的最大執行時間則為U_a*I_a。換言之，非週期性工作的抵達時間的間格會變動但是執行時間皆不變。

為了能夠讓實際的執行時間有變異度，我們採用改變*BCET/WCET*比例的方式，*BCET*為最小執行時間(best-case executiontime)，而*WCET*為最大執行時間(worst-case execution time)。*BCET/WCET*比例值被設定在0.1到1.0之間，工作的實際執行時間符合常態分布(normal distribution)，當給於一個*BCET/WCET*值時，我們將常態分布的平均數(mean)與標準差(standard deviation)分別設定為(*WCET*+*BCET*)/2與(*WCET*-*BCET*)/6。因此，可以讓實際執行時間的長度被設定在[BCET, WCET]區間的機率接近99.7%[9]。由於*BCET/WCET*的關係，當系統如果是超載時，實際負載有可能還未超載，例如當*BCET/WCET*被設定為0.5時，儘管最大系統負載為 1.3，但是由於*BCET/WCET*關係，將使實際負載只有0.65(= 0.5*1.3)。 不過，當*BCET/WCET*為1.0時，相對的此時系統就會是超載，此時系統負載為1.3 (= 1.0*1.3)。每一個數據是20組同樣參數的任務集合的實驗平均值，而模擬時間則為該任務集合中的所有週期性任務的週期的最小公倍數。處理器的耗能函式被設定與速度成3次方關係，即為$p(S) = S^3$。我們將以正規化後的系統耗能與正規化後的非週期性工作的平均反應時間的成績來當做效能評估準則，而正規化的動作是某一測試機制的實驗結果除以利用MRA機制在*BCET/WCET* = 1.0下的實驗結果。

4.2 實驗結果

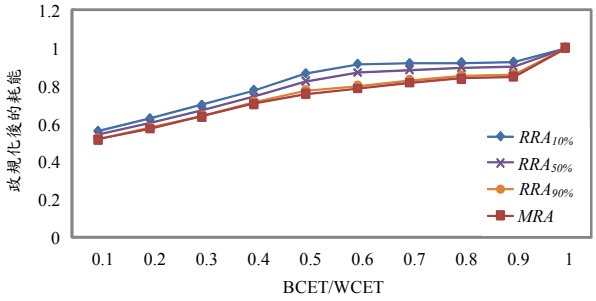

圖 4：當 U_p=0.5 時的正規化後耗能

圖4為在MRA機制(即為$RRA_{100\%}$)與RRA機制在*R值為* 10%、50%、和90%下的正規化後耗能。當*BCET/WCET* = 1.0時，所有的機制都有相同且最大的耗能，因為此時的實際負載都一樣且為1.3，沒有任何工作會提早結束，所以沒有任何未使用的執行時間(earliness)可用來降速。另一方面，如果*BCET/WCET*值漸減時，系統實際負載將隨之減少，工作將會提早結束，未使用之執行時間變多，因此，週期性工作可以用來降速，導致耗能降低。和$RRA_{10\%}$、$RRA_{50\%}$、和$RRA_{90\%}$相較之下，我們可以發現MRA機制有較低的耗能，這是因為它給即將執行的週期性工作所有可使用的未使用的執行時間，未考慮到系統中有非週期性工作在等待執行。

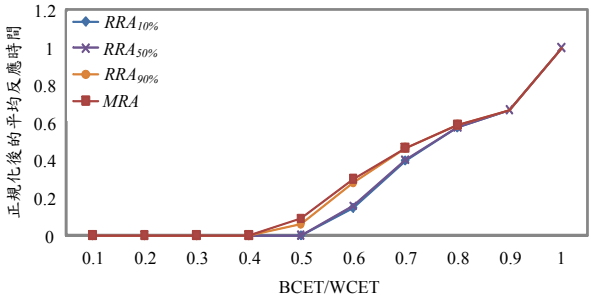

圖 5：當 U_p=0.5 時的正規化後非週期性工作的平均反應時間

圖5為在MRA機制(即為$RRA_{100\%}$)與RRA機制在*R值為* 10%、50%、和90%下的正規化後非週期性工作平均反應時間。我們可以觀察到當

BCET/WCET <=0.4時，所有機制的數據都接近於0，這是因為非週期性工作的實際負載為0.32(=(1.3-0.5)*0.4)，比TB server的能力0.5(=1.0-0.5)小，且由於週期性工作大多提早結束，提供能足夠的未使用執行時間，因此讓非週期性工作可以儘早完成。當*BCET/WCET*介於0.5與0.7之間時，我們可以發現有較低*R*值的$RRA_{10\%}$與$RRA_{50\%}$有較低的反應時間，因為沒有將較多的未執行使用時間給週期性工作使用。最後當*BCET/WCET*>= 0.8時，因為系統中的非週期性工作的負載已經大於等於0.64(=(1.3-0.5)* 0.8)，超過TB server的能力0.5(=1.0-0.5)，因此大量的非週期性工作要執行，且甚至有需多非週期性工作在模擬時間結束後，還未執行完畢。

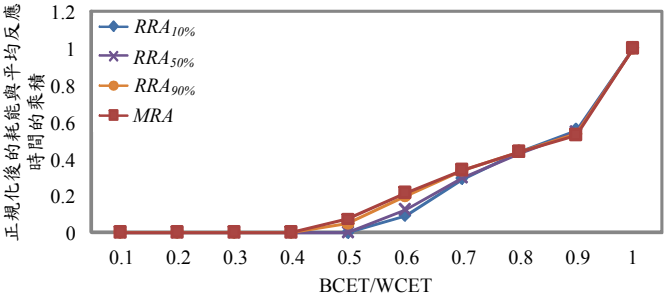

圖 6：當 U_p=0.5 時的正規化後的耗能與非週期性工作的平均反應時間乘積

圖 6 為當 U_p=0.5 時的正規化後的耗能與非週期性工作的平均反應時間乘積。$RRA_{50\%}$ 和 $RRA_{10\%}$ 有相似的效能，且比 MRA 與 $RRA_{90\%}$好，和圖 4 與 5 比較，我們可以發現乘積值主要是受圖 5 的影響。當我們在考慮使用哪個 *R* 值，我們可以再從圖 4 發現，$RRA_{50\%}$是比 $RRA_{10\%}$佳，因為 $RRA_{50\%}$系統耗能較低。

圖 7 為 U_p=0.9 時的正規化後的耗能與非週期性工作的平均反應時間乘積，比較圖 6 與圖 7，我們可以發現圖 7 中的斜率較陡，因為當 U_p=0.9 時，TB server 能力 u_s 只有 0.1(=1.0-0.9)，但是 U_a=0.4(=1.3-0.9)，相差 4 倍。而當 U_p=0.5 時，U_a=0.8，而 u_s 為 0.5(=1.0-0.5)，雖然 u_s 還是比 U_a 小，但是只差 1.6 倍，由於還是主要受到平均反應時間的影響，因此呈現較陡的情況。

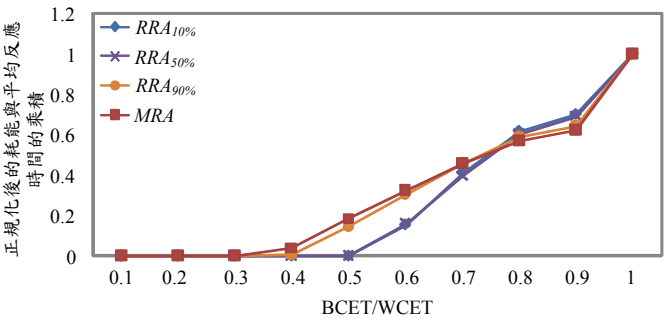

圖 7：當 U_p=0.9 時的正規化後的耗能與非週期性工作的平均反應時間乘積

5. 結論

本研究在探討在感測網路中的感測節點內的混合式任務的省電排程，因為工作有可能不會完全使用完安排給它的所有執行時間，因此我們提出一個名為 *Ratio Reclaim Algorithm* 的演算法，考量到系統耗能與非週期性工作的反應時間之間的平衡，提出利用使用者定義參數來設定要給週期性工作的未使用時間比例，來最小化系統耗能與非週期性工作的反應時間，我們的評估準則是這兩個值的乘積。此外，為能夠減少維護未使用執行時間的成本，我們也提出了新的維護資料結構*earliness*-queue來有效維護未使用執行時間。從實驗結果，我們證實了所提出的方法比之前研究所提的方法有較好的效能。

致謝

本文作者希望向國科會表達感謝。本論文在國科會計畫編號 NSC 101-2221-E-390-007, NSC 102-2221-E-390-009, NSC 101-2218-E-025-001, 102-2221-E-025-002-MY2 的支持下完成。

參考文獻

1 H. Aydin and Q. Yang. Energy-responsiveness tradeoffs for real-time systems with mixed workload. In *IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 74-83, 2004.

2 H. Aydin, R. Melhem, D. Mosse, and P. Alvarez. Power-aware scheduling for periodic real-time tasks. *IEEE TRANSACTIONS ON COMPUTERS*, 53(5):584-600, 2004.

3 M. Spuri and G. Buttazzo. Scheduling aperiodic tasks in dynamic priority systems. *Real-Time Systems*, 10(2):179-210, 1996.

4 J. Chen and C. Kuo. Energy-efficient scheduling for real-time systems on dynamic voltage scaling (dvs) platforms. In *IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 28-38, 2007.

5 I. Hong, G. Qu, M. Potkonjak, and M. Srivastava. Synthesis techniques for low-power hard real-time systems on variable voltage processors. In *The 19th IEEE Real-Time Systems Symposium*, pages 178{187, 1998.

6 T. Li and C. Ding. Instruction balance and its relation to program energy consumption. In *Proceeding of International Workshop on Languages and Compilers for Parallel Computing*, pages 71-85, 2001.

7 C.L. Liu and J.W. Layland. Scheduling algorithms for multiprogramming in a hard-realtime environment. *Journal of the ACM*, 20(1):46–61, January 1973.

8 T. Martin and D. Siewiorek. The impact of battery capacity and memory bandwidth on cpu speed-setting: a case study. In *Proceedings of the 1999 International Symposium Low Power Electronics and Design*, pages 200-205 Aug. 1999.

9 Y. Shin and K. Choi. Power conscious fixed priority scheduling for hard real-time systems. In *Proceedings of 36th Design Automation Conference*, pages 134-139, 1999.