多處理器資料庫系統中利用連續查取特性之多重查詢排程
C-R Property Approach for Multiple-query Scheduling in Multi-processor Database Systems

羅有隆 *Yu-lung Lo*　　**范美琴** *Mei-chin Fan*　　**游合成** *Ho-cheng Yu*

朝陽科技大學 資訊管理系
台中縣霧峰鄉吉峰東路 168 號
Department of Information Management
Chaoyang University of Technology
168, GiFeng E. Rd., WuFeng TaiChung County, Taiwan 413, ROC

yllo@cyut.edu.tw　*s8854608@mail.cyut.edu.tw*　*s8854618@mail.cyut.edu.tw*

摘要

多處理器系統架構被公認為最具擴充性且符合超大型資料庫的高產能需求，而查詢排程是影響多處理器系統效能的重要因素之一。 到目前為止，絕大部分有關查詢排程的研究，僅注重在每次單一查詢的有效處理，只有少數的研究關注在多重查詢的排程。 在理想中的多重查詢排程，很有機會可以找出最佳的查詢組合群，嘗試使整組查詢執行時間的最小化。 另外，連續查取特性是個有名的資料組合特性，應用在資料庫裡，使有效的將資料存放到儲存媒體。 在本研究報告中，我們利用連續查取的組合特性，設計了新的多重查詢排程技術。 經由實驗，我們發現新的排程技術有著令人印象深刻的效能改進。

關鍵詞：查詢排程、連續查取特性、多處理器系統、平行處理

Abstract

It has been recognized that multiprocessor system architecture is most scalable to support very large database. Query scheduling is an important performance issue in multiprocessor database systems. Most researches on query scheduling have so far concentrated mainly on efficient processing of one query at a time. However, only few researches have paid attention on this issue of multiple-query scheduling. The ideal multiple-query scheduling method would optimize a set of queries together and try to minimize the total execution time for the entire set of queries. Furthermore, the consecutive retrieval property (C-R property) is a well known data set property and most used as for data allocation in database systems. In this paper, we investigate the C-R property approach for multiple-query scheduling techniques. Our studies show that the new proposed scheduling techniques provide impressive performances.

Keyword : query scheduling, C-R property, multiprocessor system, parallel processing

一、 簡介

隨著資料庫系統大小不斷地擴大，使得資料查詢變得更為複雜。 特別是在一些超大型資料庫(Very Large DataBase, VLDB)**應用系統中**(**如全國戶政系統、地理資訊系統、決策資訊系統等等**)**的查詢，經常是相當耗時的。 因此，如何加速資料庫查詢的處理速度，是一重要且急需解決的問題。 平行處理是可用來加速查詢處理的重要方法之一，由於微處理機的技術成熟且價格越來越便宜，使用多個處理器於資料庫系統中變得更加經濟可行。 近年來，許多利用平行處理原理的多處理器資料庫系統原型**(prototypes)**已被發展出來，典型的例子包括了東京大學的** Grace Machine[17]、Teradata **的** DBC/1012 **資料庫電腦**[25]、MCC **的** Bubba **系統**[2]、**以及威斯康辛大學的** GAMMA **資料庫系統**[9]**等。**

多處理器系統架構被公認為最具擴充

性，且符合超大型資料庫的高產能需求[9,10,17,25]。 在此架構中，每個處理結點(Processing Node, PN)含有一或數個處理器，以及擁有專屬的記憶體與硬碟，而這些處理結點靠著系統內高頻寬的網路連結。 系統效能在此多處理器架構中，將取決於有效資源的利用與查詢負載的排程。 特別是查詢排程(query scheduling)，是影響多處理器系統效能的最重要因素之一。

而到目前為止，絕大部分有關查詢排程的研究報告，僅注重在每次單一查詢(single-query)的有效處理[6,7,24]。 然而，每次單一查詢的處理，在多使用者(multi-user)的系統環境中，並非最適當的。 特別是大部分的單一查詢排程研究，嘗試著利用單一查詢的平行處理(parallel processing)最佳化，以使回應時間(response time)的最小化，卻忽略了同時平行處理多個查詢的多重查詢排程(multiple-query scheduling)，有著可能潛在優勢。也就是說，目前以先做查詢最佳化(Query Optimization)[5,13,14,15,16,17,18,19,20]，然後再一次一個查詢的執行[6,24]，無法保證達到最有效率的排程。 例如不同的查詢有可能是對某一相同資料的查詢，若將不同查詢分開來執行，則此一資料可能需要做兩次的讀取。而且未必每個查詢的資料剛好分佈在所有的處理結點(PNs)裡，而為了達到最大的平行度(maximum degree of parallism)與負載平衡(workload balance)，可能還必須做資料的重分配。 也就是，有部分資料必須傳送至其他處理節點處理，否則查詢的執行過程中，可能有部分處理節點因無資料可處理而遭到閒置。理想中的多重查詢排程，很有希望可以找出最佳的查詢組合群組，而一次一組查詢同時執行，儘量避免資料不必要的重分配，嘗試使整組查詢執行時間的最小化，而此時間當然希望能小於各別執行一個個查詢所需時間的總合。 然而，目前還只有很少的研究關注在多重查詢的排程。

連續查取特性(Consecutive Retrieval Property, C-R property)[11]的應用，是對於查詢的資料可以重組出某些連續(sequential)關係使之具有 C-R property(將於文中第三節說明)，再利用此連續特性將資料連續置放於儲存媒體中，此置放方式可以節省查詢的執行時間[1,3,4,8,11,12]，對系統效能的提升有顯著的助益。 在本研究報告裡，我們將利用 C-R property 的特性，設計新的多重排程查詢技術，以促進多處理器資料庫系統之效能與實用可行。 經過實驗的證明，我們發現此新的技術確實優於現有的多重查詢排程演算法。

這篇報告的其他章節安排如下，我們在第二節將現有的多重排程技術做了擇要的介紹；在第三節，將針對於 C-R property 的應用做簡單的說明；而於第四節，提出了我們利用 CR-property 所設計的多重查詢排程新技術；接著在第五節，我們設計實驗以證實所提出的新技術實用可行；在最後一節，我們將做研究報告的總結，以及提出未來可能的研究方向。

二、 現有之研究概況

於多處理器資料庫系統應用中，相較於為數眾多的一次執行一個查詢之單一查詢最佳化研究報告，有關多重查詢排程技術之研究相對稀少，代表性的研究有 Manish Metha 等的 Batch Scheduling[22]，與 Lo 等的 CB Scheduling & PB Schedulings[21]。 分別詳述如下：

- Mehta、Soloviev 與 DeWitt 發表的 Batch Scheduling[22]：此技術在查詢排程中，考量到記憶體(memory)與 I/O 的有效運用，凡需讀取相同關連之查詢，都儘量以批次(batch)一起執行，如此可以節省不少重複讀取之不必要 I/O。 例如有許多的查詢在等待被執行，查詢 Q_i 是 R_1 與 R_2 兩關連的連結(Join)，而查詢 Q_k 是 R_1 與 R_3 兩關連的連結，在記憶體的容量允許下，可以將 Q_i 與 Q_k 同時一起執行，則每一個關連，不管是 R_1、R_2 或 R_3，都只需要做一次的讀取。 反之如果將 Q_i 與 Q_k 分開來執行，一次只做一個查詢，若 Q_k 無法於 Q_i 執行後馬上接著排進去執行以利用記憶體裡對 R_1 的殘留，則 R_1 可能無法避免的需要多一次的讀取而為兩次。 所以多重查詢的效益在此情況下得以顯現。 此技術之缺點是未把最大平行度與平衡負載列入考量，因此有可能所選取的查詢局限於使用部分之處理節點來執行，而閒置了部分資源與使處理節點有不平衡負載。

- Lo、Hua 與 Tavanapong 於[21]所提出的 Competition-Base (CB) Scheduling 與 Planning-Based (PB) Scheduling：

  — CB scheduling：此術技是放任查詢以先來後到的方式自由爭取執行所需的處理節點，一旦都爭取到了處理節點的查詢，即可立即被執行。 例如：查詢 Q_1、Q_2、Q_3 依序進入系統，查詢 Q_1 需要在處理結點 PN_1、PN_2 與 PN_3 中執行，查詢 Q_2 需要在 PN_3 與 PN_4，另外查詢 Q_3 則須在 PN_4、PN_5 與 PN_6 中執

行。 依先來先搶的規則，查詢 Q_1 因可以馬上爭取到所需的處理節點，將馬上被執行，而查詢 Q_2 雖可占有 PN_4，但因 PN_3 為 Q_1 所占用，需要等 Q_1 完成後才能被執行。 至於 Q_3，則因 PN_4 被 Q_2 佔有，而必須排在 Q_2 之後執行。 而如果 Q_3 能在 Q_2 前，優先的搶到 PN_4 的話，則 Q_1 與 Q_3 將有機會可以同時被執行。

— PB scheduling：此技術將現有空閒的的處理節點，有計畫的做查詢排程。同樣的被安排到足夠處理節點的查詢，就可以馬上被執行。 如果多個查詢同時擁有所需的處理節點，則這些查詢可以被同時執行。利用 PB scheduling 的理論發展出 Largest-Fit-First 及 First-Fit-First 兩種演法，其說明如下：

Largest-Fit-First 演算法：

主要的精神在於先將等待執行的查詢依其所需使用到處理節點數量由大至小排序，將必須使用較多處理節點的查詢優先處理。

First-Fit-First 演算法：

依查詢依查詢進入系統的時間由先至後排序，將較先進入系統的查詢優先處理。

在這個研究中我們發現，PB 的多重查詢排程優於 CB 的多重查詢排程，而在 PB 的排程中 Largest-Fit-First 又略優於 First-Fit-First 模式。

由以上兩個研究的驗證，證實有計畫的安排多重查詢排程是有其必要而且可行的。 然對多重查詢排程的設計仍有研究發展的空間，因不管是使用 Large-Fit-First 或 First-Fit-First，來進行排程仍然可能如[22]一樣，閒置部分系統資源。

三． 連續查取特性(C-R Property)

C-R property 其基本的應用是將資料重新整理過後，把與一個查詢相關的記錄，存放在線性儲存媒體(linear storage)裡的連續區域中，以減少查詢時對資料的讀取時間，相關研究報告有[4,8,11,12]。 其中 Chang 等，於[4]中，更將其應用於圖片資料庫的查詢。 而另一種常見的應用方式，是將資料重組後使其具有 C-R property，再依序儲存於多重磁碟系統(multiple-disk system)中，以加速資料的平行存取，相關研究有[3]。 這兩種資料存放應用方式，已被研究證實效果比傳統的方式良好。 接著我們以一簡單的範例來說明 C-R property 的形成方式：

假設一資料表格有六筆記錄(R_1 ..R_6)，而有三個查詢(Q_1、Q_2、Q_3)對此資料表格中記錄的讀取需求如圖 1：

		Q_1	Q_2	Q_3
Page1	R_1:	1	0	0
	R_2:	0	1	1
	R_3:	1	0	0
Page2	R_4:	0	1	1
	R_5:	1	1	0
	R_6:	0	0	1

圖 1、 查詢讀取需求範例

在圖 1 裡，表示查詢 Q_1 需讀取記錄 R_1、R_3 與 R_5，查詢 Q_2 需讀取記錄 R_2、R_4 與 R_5，而 Q_3 依此類推。 如果資料是以 I/O 區塊來存放，如：頁(pages)，存於儲存媒體中，而又假設每一頁只能放置三筆的記錄，則所有資料共需放置於兩頁之中，假設 R_1、R_2 與 R_3 置於同一頁 Page1，R_4、R_5 與 R_6 置於另一頁 Page2。則處理 Q_1 需要兩次的 I/O，同樣的 Q_2 與 Q_3 也需要兩次的 I/O，而平均查詢所需 I/O 次數為 2。 但如果重組每一頁裡的記錄安排，使每個查詢所需資料剛好都能連續排列在一起，假設 R_1、R_3 與 R_5 被分配到 Page1 中,而 R_2、R_4 與 R_6 分配於 Page2 中，結果如圖 2 所示，則稱此具有 C-R property：

		Q_1	Q_2	Q_3
Page1	R_1:	1	0	0
	R_3:	1	0	0
	R_5:	1	1	0
Page2	R_2:	0	1	1
	R_4:	0	1	1
	R_6:	0	0	1

圖 2 、 具 C-R property 之資料安排

符合 C-R property 的此種資料分配裡，處理查詢 Q_1 與 Q_3 只需一次的 I/O，而 Q_2 雖然仍然需要兩次 I/O，但總平均 I/O 需求則為 1.33 次，比之前未具有 C-R property 之資料分配來得有效率。

四、應用 C-R property 之查詢排程

C-R property 雖然至今仍大多被應用於資料的置放，但在這篇報告裡，於多處理器資料庫系統中，我們率先將 C-R property 應用在多重查詢排程的處理中。 在介紹所提出新的多重查詢排程技術之前，我們將先說明適用於查詢排程的兩種符合 C-R property 的重組策略。

（一）C-R property 的重組策略

同樣的先以一範例來說明 C-R property 的兩種重組策略，假設一多處理器資料庫系統中有七個等待被執行的查詢，在表 1 中列出此七個查詢對系統處理結點的相關需求。 如查詢 Q_1，其可能因資料存放位置的關係，必須使用到處理結點 PN_3、PN_2 與 PN_6；而查詢 Q_2，其處理資料則必須使用到處理結點 PN_1 與 PN_4，等等。

表 1. 查詢對系統處理結點相關需求範例

	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7
PN_1	1	1					1
PN_2	1					1	1
PN_3				1		1	
PN_4		1			1		1
PN_5			1		1		1
PN_6	1					1	1
PN_7			1		1		
PN_8				1		1	

在不改變查詢對處理結點的需求原則下，重組表 1 中欄(column)與列(row)的順序，使重組過後具 C-R property 的排列方式。 基本上我們可以有兩種重組策略：處理結點需求數目最小的查詢優先(CRP_Smallest_First)，以及處理結點需求數目最大的查詢優先(CRP_Largest_First)。

CRP_Smallest_First — 處理結點需求數目最小的查詢優先的策略，其重組結果如表 2 所示，重組過後的查詢與處理結點，具有 C-R property 的排列方式，其特性說明如下。

於表 2 中，如果我們將查詢按照其所需的第一個處理結點出現的列(row)來分層級，例如：Q_4 與 Q_6 所需的第一個處理結點，皆為表中第一列的 PN_3，則 Q_4 與 Q_6 同屬第 1 層；而又 Q_1 與 Q_7 所需的第一個處理結點，皆為表中第三列的 PN_6，則 Q_1 與 Q_7 亦同屬第 3 層；於此類推，Q_2 為第 5 層，Q_5 為第 6 層，Q_3 為第 7 層。 如此 CRP_Smallest_First 的特性在於同一層的查詢中，其排列方式是將查詢依處理結點的需求數目，按遞增方式排序。 例如於表 2 的第 1 層中，Q_4 需兩個處理結點，而 Q_6 需四個處理結點，如此 Q_4 排在 Q_6 的前面；又如第 3 層中，Q_1 需三個處理結點，而 Q_7 需四個處理結點，則 Q_1 排在 Q_7 的前面。

表 2. 查詢對系統處理結點需求以 C-R property 重組後範例

層級	1		3		5	6	7
	Q_4	Q_6	Q_1	Q_7	Q_2	Q_5	Q_3
PN_3	1	1					
PN_8	1	1					
PN_6		1	1	1			
PN_2		1	1	1			
PN_1			1	1	1		
PN_4				1	1	1	
PN_5				1		1	1
PN_7						1	1

CRP_Largest_First — 處理結點需求數目最大的查詢優先的策略恰好與 CRP_Smallest_First 相反，顧名思義是將屬同一層級的查詢，依所需處理結點的數目，按遞減的方式排序。 將表 1 使用 CRP_Largest_First 策略重組，結果如表 3 所示。 而這次第 1 層中的 Q_6 排在 Q_4 的前面，而第 3 層中的 Q_7 則排在 Q_1 的前面。

表 3. 查詢對系統處理結點需求以 C-R property 重組後範例

層級	1		3		5	6	7
	Q_6	Q_4	Q_7	Q_1	Q_2	Q_5	Q_3
PN_3	1	1					
PN_8	1	1					
PN_6	1		1	1			
PN_2	1		1	1			
PN_1			1	1	1		
PN_4			1		1	1	
PN_5			1			1	1
PN_7						1	1

（二）應用 C-R property 之查詢排程演算法

在此研究中，我們發展出應用 C-R property 的多重排程演算法(CRP_Scheduling)，此排程技術也是利用批次做查詢執行，原則上有以下四個步驟：

(1) 將查詢佇列(query queue)中的查詢重組，使重組過後具 C-R property 的排列方式，使用 CRP_Smallest_First 或 CRP_Largest_First 策略。

(2) 在同一層的查詢中，同時間最多只能取一個查詢的原則下，找出一個查詢

組合能使用全部處理結點，或使用最多數目的處理結點。

(3) 執行所找到的查詢組合，並將查詢從佇列中移除。

(4) 如果查詢佇列中已無查詢存在，則排程結束。 如不然，且若又有新的查詢進入佇列，則重覆步驟(1)~(4)安排下一批次的查詢；而若未有新查詢進入佇列，則重覆步驟(2)~(4)安排下一批次的查詢。

以下以一 C 程式的虛擬碼來表示我們的演算法：

```
CRP_Scheduling：
WHILE (Query_Queue NOT EMPTY)
{
   IF (there is a new query)
      CRP_Matrix = Qry_CRP(Query_Queue);
      /* 步驟(1)，結果存於 CRP_Matrix */

      Query_Set = Cmp_Comb(CRP_Matrix);
      /* 步驟(2)，結果存於 Query_set */

      Run_Queries(Query_Set, Query_Queue);
      /* 步驟(3)，執行並從 queue 中
      移除查詢 */

} /* 步驟(4)，下一輪 */
```

（三）範例說明

在這節裡，我們以重組過後具 C-R property 的表 3 做為範例，簡單說明 CRP_Scheduling 演算法的執行過程：

1. 考量表 3 中第一個查詢 Q_6，所需的處理結點有 PN_3、PN_8、PN_6與 PN_2，於表中佔用第 1 列至第 4 列的處理結點。
2. 從第 5 層(列)起，找尋可以與 Q_6同時執行的查詢，而發現 Q_2 可以，且其佔用第 5 列至第 6 列的處理結點。
3. 再來從第 7 層(列)起，找尋可以與 Q_6及 Q_2同時執行的查詢。 Q_3可以，且其佔用第 7 列至第 8 列的處理結點。
4. 此時 Q_6、Q_2與 Q_3已可佔用全部的八個處理結點，無須再尋找，這一批次就安排此三個查詢共同被執行，且將它們從佇列中移除。
5. 假設沒有新的查詢進入佇列，進入下一批次的排程。
6. 考量表中剩下的第一個查詢會是 Q_4，所需的處理結點有 PN_3與 PN_8，於表中佔用第 1 列與第 2 列。
7. 從第 3 層(列)起，找尋可以與 Q_4同時執行的查詢，發現 Q_7可以，且其佔用第 3 列至第 7 列的處理結點。
8. 從第 8 層(列)起，找尋可以與 Q_4及 Q_7同時執行的查詢，但找不到了。
9. 由於所找到的 Q_4及 Q_7僅佔用七個處理結點，未佔用全部，我們必須嘗試找尋是否有佔用更多處理結點的查詢組合。
10. 回溯至上一層，我們發現於第 3 層(列)尚有一查詢 Q_1 可與 Q_4 組合，兩查詢共佔用第 1 列至第 5 列的處理結點。
11. 從第 6 層(列)起，找尋可以與 Q_4及 Q_1同時執行的查詢。 Q_5中選。
12. 此時 Q_4、Q_1與 Q_5已可佔用全部的八個處理結點，執行此三查詢，且將它們從佇列中移除。
13. 再下一批次的排程，假設仍無新的查詢加入，則佇列中只剩查詢 Q7。
14. 執行 Q7。
15. 排程結束，綜合排程情形於表 4。

七個查詢共分為三個批次執行，而系統執行狀況及處理節點使用情況如下：

表 4. 排程結果

批次	查詢	使用的處理節點
1	Q2,Q6,Q3	PN1, PN2, PN3, PN4, PN5, PN6, PN7, PN8
2	Q1,Q4,Q5	PN1, PN2, PN3, PN4, PN5, PN6, PN7, PN8
3	Q7	PN1, PN2, PN4, PN5, PN6

五、 實驗

為了檢驗所提之新的多重查詢排程演算法 CRP_Scheduling 的效能，接下來我們設計了排程實驗系統，模擬 CRP_Scheduling 的排程，並與現有的多重查詢排程技術做比較，以確定新排程技術的可行性。

(一) 實驗模型

在圖 3 中展示出我們的實驗模型。首先由查詢產生器(Query Generator)利用亂數產生查詢，以及其所需使用的處理節點，接著將所產生的查詢送進查詢佇列(Query Queue)。 為避免查詢佇列中有過多的查詢，使得決定排程過於耗時，我們另外設計了一個排程視窗(Scheduling Window)於查詢佇列的前面，唯有進入排

程視窗範圍內的查詢，得以被考慮於下一批次的排程。 再來排程器(Scheduler)利用所提供的各種排程演算法，從排程視窗中找尋可共同執行的適當查詢組合，最後將所選定的查詢送入服務單元(Serving Unit)進行執行。 為了使排程與執行兩階段工作能銜接順利，我們建議系統能有一專屬的協調處理器(coordinator)，讓它只負責排程，而不參與查詢執行，如此當有查詢正在執行時，協調處理器也利用了這個時間決定下一批次可以被執行的查詢組合。

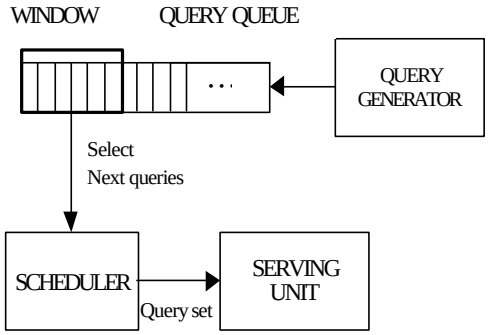

圖 3．實驗模型

於實驗中所使用的各項參數範圍值如表 5，其中處理節點數(Number of PNs)與排程視窗的大小(Scheduling window size)，是主要的變動參數，我們將利用不同的處理節點數與排程視窗的大小，來評估對各排程的影響。 而每次實驗所測試由亂數產生的查詢個數(Number of queries)必定大於 10,000 個，目的是為了使實驗結果的誤差值能在 ±1%以內。此外，查詢所需使用的處理結點數，亦由亂數產生，可以是 1 個到使用全部的處理結點，而全部查詢的平均約是使用了三分之一的處理結點數。

表 5. 各項參數值

PARAMETER	VALUES
Number of PNs(N)	8,16,32 or 64,128
Number of Queries	over 10,000
Query Size(no. of PNs used)	1~N, average (N/3)
Scheduling Window Size (queries)	varied, 16~128

除此之外，我們還使用了系統使用率 ***U*** (System Utilization)，亦即系統中處理結點的平均使用率；以及系統產能 ***T*** (System Throughput)，系統平均單位時間處理查詢的數量。 以兩統計值來評估各排程的效能。

$$U = \frac{\sum_{i=1}^{N} U_i}{N}$$

U_i：Utilization of the ith PN(每處理節點使用率)

N：Total Number of PNs(處理節點總數)

$$T = \frac{TQ}{TT}$$

TQ：Total Number of Queries(處理完的查詢總數)

TT：Total Execution of Time(執行總時間)

接下來的實驗，對於 CRP_Scheduling 演算法，我們同時採用了 CRP_Smallest_First 與 CRP_Largest_First，兩種 C-R property 的重組策略，以瞭解兩者對多重查詢排程的影響，是否有差異。 在往後的說明，我們將簡單的以 CRP_Smallest_First 與 CRP_Largest_First 來代表 CRP_Scheduling 的兩種演算法。 另外，為了評估 CRP_Scheduling 的效能，我們將它與應用性質較接近的 PB scheduling[21]中的 First_Fit_First 以及 Largest_Fit_First 兩演算法做比較。 因在[21]中已證實了此兩演算法是較有效率的多重排程技術。

最後附帶一提，在下面的實驗數據中有所謂的相對改善率 (Relative Improvement, *RI*)，是指各演算法與 First_Fit_First 比較的結果。 再下來的是，針對不同參數變化所做的實驗所得。

(二) 處理結點數的影響(Effect of Number of PNs)

在這個實驗裡我們將排程視窗大小固定為 32，亦即每一輪排程，必須是在查詢佇列前 32 個查詢，才有機會被安排去執行。 分別測試處理節點數為 16、32、64 及 128 的情況下，對各排程演算法的影響。

首先探討系統使用率，其實驗結果於表 6 與圖 4 中。 不幸的，我們發現，所有的系統使用率，幾乎是隨著處理結點數的增加而遞減，究其原因，是因為排程視窗被固定為 32。如此，當處理結點數增加時，卻無足夠數目的查詢，可供各排程演算法安排出最理想的查詢組合。(我們另外有排程視窗大小隨著處理結點數增加而增加的實驗，將說明於專節五之四中說明。) 但此實驗的主要目的為比較各排程演算法，在相同條件設定下的優劣差異。 由圖 4，我們發現 CRP_Largest_First 有著最佳的系統使用率，而 CRP_Small_First 只略遜於 CRP_Largest_First，兩者是很接近的。 再來 Largest_Fit_First 與 CRP_Largest_First 已有明顯的差距，而系統表現最沒有效率的是 First_Fit_First。 另外，CRP_Largest_First 與

CRP_Small_First 相較於 First_Fit_First，最好時有接近 12%左右的改善率(*RI*)，而反觀 Largest_Fit_First 最多約只有約 7.2%或以下的改善率。

再由系統產能來討論，相似於系統產能系統使用率，表現最佳的還是 CRP_Largest_First，而與其很接近的仍然有 CRP_Smallest_First，再來是 Largest_Fit_First，表現最差的仍舊是 First_Fit_First。系統產能是系統單位時間處理查詢的能力，實驗結果表現於表 7 與圖 5。由實驗結果顯示，所有演算法的系統產能皆隨著處理結點數的增多，而往下降，然後趨於緩和，理由同樣是受限於排程視窗固定為 32。而 CRP_Largest_First 與 CRP_Small_First 相較於 First_Fit_First，仍有著約 10%左右或高達 11.6%的改善率(*RI*)，而 Largest_Fit_First 卻約只有 5.9%或以下的改善率。

表 6. 處理結點數 vs.系統使用率

No.of PNs	First_Fit_First	Largest_Fit_First		CRP_Smallest_First		CRP_Largest_First	
	U	*U*	*RI*	*U*	*RI*	*U*	*RI*
16	89.19	95.6	7.187%	99.13	11.145%	99.57	11.638%
32	86.33	92.18	6.776%	96.04	11.248%	96.68	11.989%
64	86.05	90.19	4.811%	94.41	9.715%	95.21	10.645%
128	86.27	88.97	3.130%	93.66	8.566%	94.55	9.598%

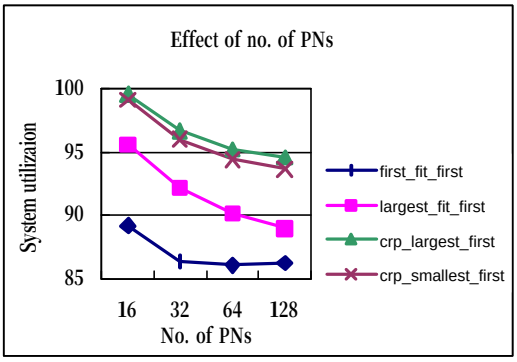

圖 4. 處理結點數 vs.系統使用率

表 7. 處理結點數 vs.系統平均產能

No. of PNs	First_Fit_First	Largest_Fit_First		CRP_Smallest_First		CRP_Largest_First	
	T	*T*	*RI*	*T*	*RI*	*T*	*RI*
16	2.66	2.76	3.759%	2.95	10.902%	2.97	11.654%
32	2.53	2.68	5.929%	2.81	11.067%	2.82	11.462%
64	2.54	2.66	4.724%	2.78	9.449%	2.80	10.236%
128	2.56	2.64	3.125%	2.78	8.594%	2.81	9.766%

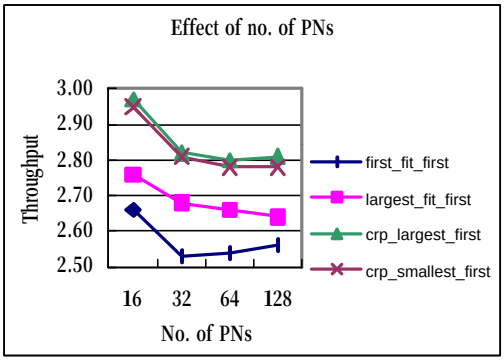

圖 5. 系統平均產能

（三） 排程視窗大小的影響(Effect of Size of Scheduling Window)

我們設計排程視窗(Scheduling Window)的主要目的，是為了避免如果查詢佇列中有著太多的查詢待排程，則排程演算法於每一批次的排程，得花很多的時間去決定該有那些查詢可以被執行，如此可能影響了排程銜接不上執行的缺點。而為了解排程視窗大小對系統效率的影響，在這個實驗裡，我們將處理節點數目固定為32，而變化排程視窗大小所得的實驗結果，表示於表8、表9、圖6、以及圖7。

再次先從系統使用率來討論，由表8與圖6的實驗結果可以看出，四個排程方法的系統使用率，皆隨著視窗大小的增大，而呈現上升的趨勢。這是因為隨視窗大小的增加，排程有了越來越多的查詢選擇空間，可以找到較佳的查詢組合所致。而比較四個排程演算法的優劣，我們再度的發現，CRP_Largest_First仍然表現的最有效率，系統使用率甚至可高達99%以上，也就是在多數的情況下可以使用到所有的處理結點，而甚少有閒置的處理結點。其次CRP_Smallest_First依舊緊跟著CRP_Largest_First之後，表現相差不遠。而第三名的還是Largest_Fit_First，系統使用率最差的還是First_Fit_First。另外，在相對比較於First_Fit_First的改善率(*RI*)，其他三個排程技術皆呈遞減的趨勢，這是因為他們的效能已經相當的高，往上再成長的空間有限，反觀First_Fit_First卻因為原本效能較差，而有較大的成長空間。不過令人欣慰的，CRP_Largest_First與CRP_Smallest_First相較於First_Fit_First，在多數的情況下大多能有10%左右或甚至高達12%的改善率。而Largest_Fit_First則介於5%至7%之間。

接下來探討系統產能，於表9與圖7中，所呈現的實驗結果與之前的系統使用率極為相似，因其受到了相同因素的影響，讀者可自行參閱，將不再贅述。

表8 系統使用率

Size of Scheduling Windows	First_Fit_First	Largest_Fit_First		CRP_Smallest_First		CRP_Largest_First	
	U	*U*	*RI*	*U*	*RI*	*U*	*RI*
16	82.27	87.87	6.81%	91.54	11.27%	92.17	12.03%
24	84.77	90.53	6.79%	94.31	11.25%	94.96	12.02%
32	86.33	92.18	6.78%	96.04	11.25%	96.68	11.99%
40	87.77	93.5	6.53%	97.54	11.13%	98.2	11.88%
48	88.78	94.26	6.17%	98.25	10.67%	98.83	11.32%
56	89.92	94.99	5.64%	98.75	9.82%	99.32	10.45%
64	90.69	95.32	5.11%	99.06	9.23%	99.57	9.79%

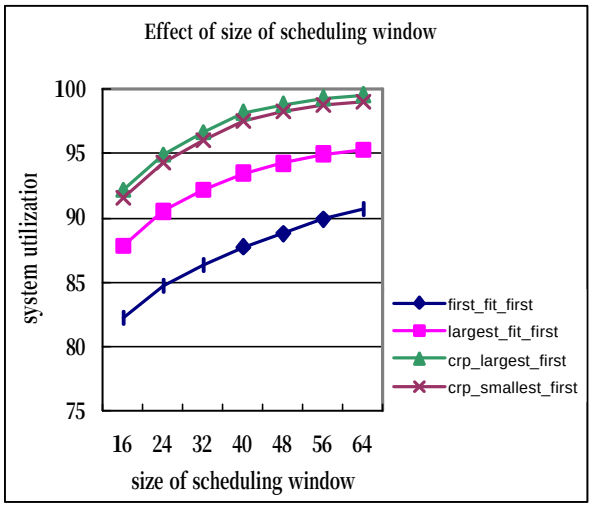

圖6 系統使用率

表 9 系統平均產能

Size of Scheduling Windows	First_Fit_First	Largest_Fit_First		CRP_Smallest_First		CRP_Largest_First	
	T	*T*	*RI*	*T*	*RI*	*T*	*RI*
16	2.39	2.54	6.276%	2.65	10.879%	2.67	11.715%
24	2.47	2.63	6.478%	2.74	10.931%	2.75	11.336%
32	2.53	2.68	5.929%	2.81	11.067%	2.82	11.462%
40	2.58	2.73	5.814%	2.86	10.853%	2.87	11.240%
48	2.63	2.76	4.943%	2.89	9.886%	2.9	10.266%
56	2.67	2.79	4.494%	2.91	8.989%	2.92	9.363%
64	2.71	2.8	3.321%	2.92	7.749%	2.93	8.118%

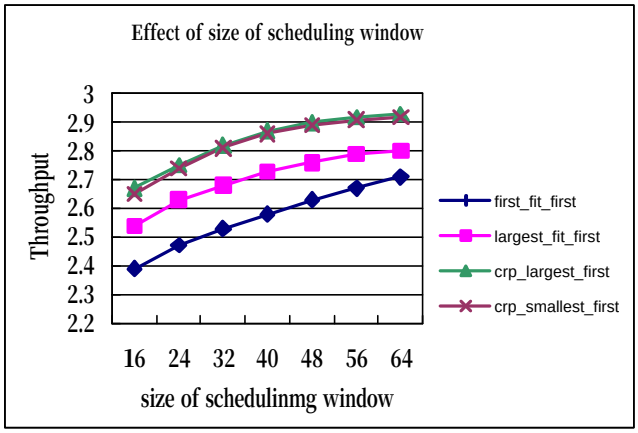

圖 7 系統平均產能

（四） 處理結點數與排程視窗大小同步變化的影響

由於前兩節報告中的實驗探討，每次固定一個參數(處理結點數或排程視窗的大小)，而只變化另一個參數。 在這節的實驗，我們讓該兩參數以 1:1 的比例同步的變化，以探討處理結點數及排程視窗的大小對四個排程演算法的影響。 我們將系統的處理節點數目與排程視窗的大小從 16 變化到 128，實驗結果列於表 10、表 11、圖 8、以及圖 9。

首先探討系統使用率，由表 10 與圖 8 的實驗結果可以看出，四個排程方法隨著處理節點數目與排程視窗大小的增加，使用率呈上揚的趨勢。 由於在五之一節實驗模型的介紹中，我們有提及，查詢所需的平均處理結點數，約為全部處理結點數的三分之一。 換個角度來說，也就是每一批次的排程，不論處理結點數的多少，平均約只有三個查詢可以被執行。 如此當處理節點數目與排程視窗的大小等比例成長，而平均每批次的查詢數不變，這將使得所有的排程方法都有很大的查詢選擇空間，系統使用率自然不斷的往上提升。 而於圖 8 中 First_Fit_First 曲線的上揚角度較大，理由與在五之三節的解釋相同，亦即因其較低的系統使用率，使得有較大的成長空間。 不過實驗參數若繼續增大，First_Fit_First 的曲線會漸趨緩和，逐漸接近其他三條曲線，但不至於超越他們。 而對於四個排程方法的比較，沒有意外的還是以 CRP_Largest_First 表現出的系統使用效能最佳，CRP_Smallest_First 的效能其次，再來才是 Largest_Fit_First，以及系統使用率最差的 First_Fit_First。

對於系統產能的討論，也因實驗所呈現結果與系統使用率相似，理由相同，不再著墨。 有關處理節點數目與排程視窗的大小，除 1:1 比例的實驗外，我們也做了 1:1/2、1:3/4、1:1.5 以及 1:2 等的實驗，結果與 1:1 所能呈現的意義與比較，差別不大，而限於篇幅，不再詳細討論。 然縱觀所有的實驗結果，隨著處理結點數的增加，適當的提高排程視窗的大小，是有其必要性，能增加查詢排程的效能。 但太大的排程視窗，則只會浪費在決定排程所需的時間。

表 10 系統使用率

No. of PNs	First_Fit_First	Largest_Fit_First		CRP_Smallest_First		CRP_Largest_First	
	U	*U*	*RI*	*U*	*U*	*U*	*RI*
16	84.21	91.18	8.277%	94.92	12.718%	95.33	13.205%
32	86.33	92.18	5.929%	96.04	11.067%	96.68	11.462%
64	88.74	92.9	4.688%	97.01	9.319%	97.95	10.379%
128	91.15	94	3.127%	97.96	7.471%	99.02	8.634%

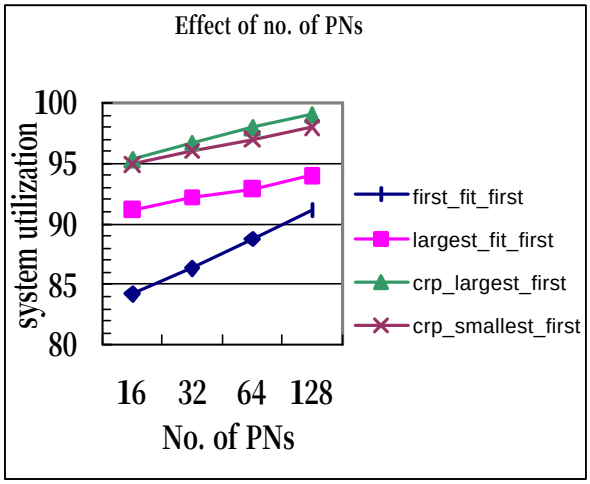

圖 8 系統使用率

表 11 系統平均產能

No. of PNs	First_Fit_First	Largest_Fit_First		CRP_Smallest_First		CRP_Largest_First	
	T	*T*	*RI*	*T*	*RI*	*T*	*RI*
16	2.41	2.57	6.639%	2.71	12.448%	2.72	12.863%
32	2.53	2.68	5.929%	2.81	11.067%	2.82	11.462%
64	2.63	2.74	4.183%	2.87	9.125%	2.9	10.266%
128	2.72	2.8	2.941%	2.92	7.353%	2.95	8.456%

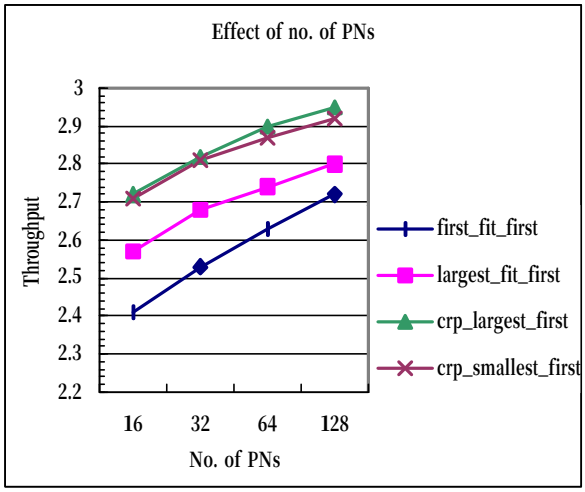

圖 9 系統平均產能

六、結論及未來工作

在這篇研究報告中，針對多處理器資料庫系統的查詢排程，我們應用了 C-R property 的的兩種重組策略—CRP_Largest_First 與 CRP_Smallest_First，提出了新的多重排程演算法。此演算法能從重組過的查詢佇列中，快速的找到佔用全部或最多處理結點的查詢組合。我們期望新的排程技術，能充份發揮多處理器系統的效能，以加速資料庫查詢的處理。我們也建構了一個實驗模型，利用不同參數的變化，做了一系列的實驗，以評估所提出多重查詢排程新技術的效能。在實驗結果裡，我們發現，應用 C-R property 的多重排程技術，無論在平均系統的使用率，或所達成的平均系統產能，皆能明顯的優於現有的多重查詢排程技術，也證實了新排程技術的可行性。而兩種 C-R property 重組策略中，又以 CRP_Largest_First 略優於 CRP_Smallest_First，但差異並不大。

我們的實驗除了考量處理器數目的變化外，也多了一個排程視窗(Scheduling Window)以規範被考慮排程的查詢數目，其主要目的，是為了避免查詢佇列中有過多的查詢，使得決定排程過於耗時。而實驗結果也證實了，當排程視窗增加到一定大小以後，對系統效能的增進是有限的，如此排程若考量所有在佇列裡的查詢，實質效益是不大的。不過根據實驗，我們也建議，隨著處理結點數目的增加，適度的增加排程視窗的大小，也有其必要性，可增進排程的效能。

本研究報告著重於，如何讓每一批次的排程盡量佔用全部或最多的處理結點數，以發揮最高的系統使用率。而已被提出的多重排程技術，如：Mehta 等[22]，則是考量如何使記憶體與 I/O 的有效運用。兩種排程技術考量的角度雖然不同，但對增進資料庫系統效能，與加速查詢處理的目標卻是一致的。未來的研究方向，將可嘗試著如何把多種考量的因素結合在一起，以發展出考量更為完整，使系統效率更高的多重排程技術。

七、參考文獻

[1] S. S. Al-fedaghi and Y. H. Chin, "Algorithmic Approach to the Consecutive Retrieval Property," *International Journal of Computer and Information Sciences*, Vol. 8, No. 4, Pages 279 - 301, 1979.

[2] H. Boral, W. Alexander, L. Clay, G. Copeland, S. Danforth, M. Franklin, B. Hart, M. Smith, and P. Valduriez, "Prototyping Bubba, A Highly Parallel Database System," *IEEE Transaction on Knowledge and Data Engineering*, 2(1):4-24, 1990.

[3] Chin-Chen Chang and Jaw-Ji Shen, "Consecutive Retrieval Organization as A File Allocation Scheme on Multiple Disk Systems," *Proceedings of the International Conference on Foundations of Data Organization*, Pages 74-80, Kyoto, Japan, May 21-24, 1985.

[4] Chin-Chen Chang, Ji-Han Jiang, and Jau-Ji Shen, "Organization of Pictorial Databases for Spatial Match Retrieval," *Proceedings on International Symposium on Next Generation Database Systems and Their Applications*, Pages 205-210, Fukuoka, Japan, 1993.

[5] Chandra Chekuri, Waqar Hasan, and Rajeev MotWani, "Scheduling Problems in Parallel Query Optimization, " *Proceedings of the 14th ACM Symposium on Principles of Database Systems*, Pages 255-265, San Jose, California, May 1995.

[6] M. S. Chen, M. Lo, P. S. Yu, and H. C. Young, "Using Segmented Right-deep Trees for the Execution of Pipelined Hash Joins," *Proceedings of International Conference on VLDB*, Pages 15-26, Vancouver, Canada, August 1992.

[7] S. Dandamudi and C-Y Chow, "Performance of Transaction Scheduling Policies for Parallel Database Systems," *Proceedings of the 11th International Conference on Distributed Computing Systems*, Pages 116 –124, 1991.

[8] Jitender S. Deogun, Vijay V. Raghavan and Thomas K. W. Tsou, "Organization of Clustered Files for Consecutive Retrieval," *ACM Trans. On Database Systems*, 9(4), Pages 646-671, 1984.

[9] D. DeWitt, R. Gerber, G. Graefe, M. Heytens, K. Kumar, and M. Muralikrishna, "GAMMA – A High Performance Dataflow Database Machine," *Proceedings of the Twelfth International Conference on Very Large DataBase*, pages 228-237, Kyoto, Japan, August 1986.

[10] D. DeWitt, S. Ghandeharizadeh, D. Schneider, A. Bricker, H. Hsiao, and R. Rasmussen, "The GAMMA Database Machine Project," *IEEE Transactions on*

Knowledge and Data Engineering, 2(1):44-62, 1990.

[11] Sakti P. Ghosh, "File Organization: The Consecutive Retrieval Property," *Communications of the ACM (CACM)*, Volume 15, Pages 802-808, 1972.

[12] Sakti P. Ghosh, "Consecutive Retrieval Property," *Data Base Organization for Data Management*, Chapter 6, Pages 212-268, 1977.

[13] W. Hong and M. Stonebraker, "Optimization of Parallel Query Execution Plans in XPRS, " *Proceedings of International Conference on Parallel and Distributed Information Systems*, Pages 218-225, Dec. 1991.

[14] Kien A. Hua and Chiang Lee, "Handling Data Skew in Multiprocessor Database Computers Using Partition Tuning," *Proceedings of the International Conference on VLDB*, pages 525-535, Barcelona, Spain, 1991.

[15] Kien A. Hua, Yu-lung Lo, and Honesty C. Young, "Considering Data Skew Factor in Multi-way Join Query Optimization for Parallel Execution," *VLDB Journal*, 2(3):303-330, 1993.

[16] Kien A. Hua, Yu-lung Lo, and Honesty C. Young, "Optimizer-Assisted Load Balancing Techniques for Multicomputer Database Menegement Systems," *Journal of Parallel and Distributed Computing*, 25(1):42-57, February 1995.

[17] M. Kitsuregawa, H. Tanaka, and T. Moto-oka, "Application of Hash to Database Machine and Its Architecture," *New Generation Computing*, 1(1):66-74, 1983.

[18] Chiang Lee and Zue-An Chang, "Workload Balance and Page Access Scheduling for Parallel Joins in Shared-Nothing Systems," *Proceedings of International Conference on Data Engineering*, pages 411-418, 1993.

[19] Chiang Lee and Kien A. Hua, "A Self-adjusting Data Distribution Mechanism for Multidimensional Load Balancing in Multiprocessor-based Database Systems," *Information Systems*, 19(7), Pages 549-567, July 1994.

[20] Chiang Lee and Zue-An Chang,"Utilizing Page-Level Join Index for Optimization in Parallel Join Execution," *in the IEEE Transactions on Knowledge and Data Engineering*, Vol.7, No.6, Pages 900-914, December, 1995.

[21] Yu-lung Lo, Kien A. Hua, and Wallapak Tavanapong, "Scheduling Queries for Parallel Execution on Multicomputer Database Management System," *Lecture Notes in Computer Science-Database and Expert Systems Applications*, Vol. 1134, pp.698-707, September 1996.

[22] M. Mehta, V. Soloviev, and D. J. DeWitt, "Batch Scheduling in Parallel Database Systems," *Proceedings of the 9th International Conference on Data Engineering*, Pages 400–410, 1993.

[23] T. H. Merrett and Yahiko Kambayashi, "Join Scheduling in a Paging Environment Using the Consecutive Retrieval Property," *Proceedings of International Conference on Foundations of Data Organization and Algorithms (FODO)*, Pages 323-347, Kobe, Japan, 1998.

[24] S. Takkar and S. P. Dandamudi, "Performance of Hard Real-time Transaction Scheduling Policies in Parallel Database Systems," *Proceedings of the 6th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, Pages 176 – 184, 1998.

[25] Teradata Corporation, Los Angeles, California, *Teradata DBC/1012 Data Base Computer Concepts and Facilities*, release 3.1 edition, 1988, Teradata Document C02-0001-05.