利用動態資料群集技術提升檔案系統效能之設計與研究

Using Dynamic Data Clustering to Improve File System Performance

高振元　　陳永埕　　姜美玲　　黃家信
國立暨南國際大學資訊管理學系
joanna@ncnu.edu.tw

摘要

電腦科技日新月異，然而由於機械的特性，使得磁碟存取速度遠不及 CPU 與 RAM 速度的提升，使得以磁碟為基礎的儲存系統一直是系統效能的瓶頸。雖然速度的提升不易，而磁碟容量卻是日遽驟增，在儲存的資料量愈大的狀況下，儲存資料的配置將更嚴重的影響存取速度。因此，本論文提出一種動態資料群集技術，並將之應用於磁碟的資料重整中，我們採用 log-based 的資料存取方式，藉由使用對檔案系統作分區的動態群集的技術，將常使用的磁碟資料動態地集中在磁碟中央的 cylinders，以減少資料在存取時磁碟的搜尋時間（seek time），希望能大幅地提升磁碟儲存系統的效能。而此種資料重整方法是不需要使用額外的硬碟儲存空間，且是動態的進行資料重整，而非只能在離峰時間才能進行重整，不然會嚴重地影響到其他使用者的正常使用。

關鍵詞：檔案系統，資料群集，作業系統，搜尋時間，LFS

Abstract

Disk performance has been the bottleneck of a computer system. By reducing the disk seek time, the performance of a disk-based storage system can be improved. This paper bases on the log-based approach to store and access data, and proposes a new dynamic data clustering technique to actively reorganize data in the storage system. Applying this technique, the storage system is partitioned into several areas, and frequently accessed data will be dynamically migrated to the centric of disks. In this way, seek times of data access can be largely reduced. Besides, this dynamic data clustering technique does not need extra disk storage space and large time overhead for data reorganization.

Keywords：File Systems, Data Clustering, Operating Systems, Seek Time, LFS

一、 簡介

電腦各種硬體技術的逐漸進步，使得 CPU 時脈快速提昇，主記憶體存取速度的增加，區域網路的速度也漸漸愈來愈快，相對來說，次級儲存媒體，以目前使用最普遍的硬碟來說，由於硬碟本身機械的特性，無法像其他硬體有著大幅度的速度增加。因此以磁碟為主要的儲存系統的電腦而言，檔案系統一直是電腦系統的瓶頸之一。

有許多的研究致力於檔案系統效能的提升，如 clustering 、logging、journaling、 prefetching、read-ahead、write-behind、、、等等都是常用的技術，探討的重點不外乎 data placement （資料應該如何配置儲存空間）以使相關的資料能盡量的連續配置，大量的 I/O 的方式，high availability，fast backup 及 recovery 的考量，提升 buffer cache 的 hit ratio 策略以減少 I/O，使用資料重整 (data

reorganization) 或 data migration 技術達到最佳的 data layout。

而硬碟在資料讀取或寫入的時候，讀寫頭要先移到目的 cylinder，接著 platters 旋轉到目的 sector，最後讀取或寫入資料；所以讀寫資料的時間（disk access time）可分為三個部分：

(a) seek time：讀寫頭移到目的 cylinder 的時間。
(b) rotational latency：platters 旋轉到目的 sector 的時間。
(c) data transmission time：硬碟實際傳輸資料的時間。

其中 data transmission time 是因為硬體的限制而無法在軟體上增強其效能，所以相關研究的重點皆是放在減少 seek time 和 rotational latency 上。

而且，由於硬碟在容量進展的相當快速，從以前的幾十、幾百 MB，在幾年之間就增進至幾十、甚至上百 GB，cylinders 數目也隨之大增，但是硬碟本身的機械特性，使得在讀寫頭移動的 seek time 及 rotational latency 的速度在提升上就相對慢了許多，因此，對於容量不斷增加的大容量硬碟來說，檔案系統如何配置空間給儲存的資料、磁碟驅動程式讀寫到實際 sectors 的順序，對系統整體效能的影響將會愈來愈大。

而檔案系統和磁碟驅動程式的實作時的不同方式即對 seek time 和 rotational latency 有著相當大的影響；檔案系統如何存放檔案？又如何配置硬碟空間給檔案？Contiguous allocation 的速度最快，可是卻有一些先天的限制，例如外部破碎（external fragmentation），還有如何決定給該檔案多大的空間以備之後的寫入等等；雖然 linked allocation 彌補了 contiguous allocation 的眾多問題，但需要額外的儲存空間來存放一些 meta-data、且多了萬一 meta-data 遺失或損毀等其他問題存在；至於 indexed allocation 雖然沒有以上的問題，但卻又更浪費儲存空間了（因其需要儲存及使用更多的 meta-data）；而可用空間的管理和目錄的不同實作方法，同樣地亦對於效能有所影響。

另一方面，磁碟驅動程式實際上讀寫資料到各個 sectors 的順序，也有許多的 disk scheduling 演算法[9]，像是 FCFS、SSTF、SCAN、C-SCAN、LOOK、C-LOOK、、、等等，研究上[9]指出，不同的 disk scheduling algorithms 對於資料存取時的 seek time 有極大的影響。

過去的研究[1,2,7,10,12]指出，利用動態資料重整（active data reorganization）的技術，在系統使用率較低的時段內，來收集使用者常用的資料，將這些資料搬移到事先保留起來的相鄰磁區，能夠有效的減少存取時所需的 seek time。然而他們所提出的方法，有些需要浪費額外的硬碟儲存空間，或者在資料重整時只能在系統閒置時或使用率較低時來進行，不然會嚴重地影響到其他使用者的正常使用。

本研究的基本想法是把磁碟中常用的資料集中放置在中央的 cylinders 上，藉由減少讀寫頭移動的距離和次數，有效地降低讀寫資料時所需的 seek time。而將相關資料群集配置在實體的硬碟空間，亦可以有效地降低所需的 rotational latency。因此，本研究的目的即是發展高效率的動態資料群集（active data reorganization）的技術，藉者收集哪些是硬碟裡常用的資料，將其動態地搬移至磁碟中央的 cylinders，以利減少存取時的 seek time，以提昇整體的效能；而執行此動態資料群集技術是低成本的，亦即此動作不應該影響到一般使用者的正常使用及反應時間（response time），且不應該降低系統的效能，同時亦不需浪費額外的硬碟空間來做資料重整。

二、 相關文獻探討

在檔案系統中有很多相關的研究，在 'Adaptive Block Rearrangement Under UNIX' [2] 的研究中，在硬碟中保留了一個固定的區域，專門放置較頻繁被讀寫的資料，他們的方法即是在系統閒置時或使用率較低的時候，將較頻繁被讀寫的資料 copy 至此保留區域內，再透過 block-remapping

table 將之後的讀寫，先去找尋保留區域內的資料，以此方法減少在讀取較頻繁被讀寫的資料時的 seek times。然而此種方法卻需保留一個固定的磁碟區域來備份較頻繁被讀寫的資料，且 block rearrangement 的動作只能在系統的使用率較低的時候進行，不然會影響到一般使用者正常使用。

在 'A System for Adaptive Disk Rearrangement'[12] 的研究中，最常被使用的 cylinders 會被移動到 disk 中間。

在 'Disk Shuffling'[7] 是 Hewlett-Packard Laboratories' Datamesh project 的一部份，它考慮 block shuffling 和 cylinder shuffling，block shuffling 和 block rearrangement 很相像，其不同在於 shuffling 簡單地交換熱門（hot）和冷門（cold）的 blocks，而 rearrangement 把熱門的資料移動到保留區域，而這也因此增加了一些空間的花費。

'Smart Filesystems'[11]則是利用一個被讀寫的頻率除以檔案大小的比率，值愈大的檔案往磁碟中間移動，值愈小的檔案往磁碟邊邊移動。

三、 利用動態資料重整技術提升檔案系統效能的研究

在本研究中，基本的想法如下：為了達到將常用的資料盡量放到磁碟中央的 cylinders 的目的，將整個硬碟劃分為 *n* 個區域，例如：當 n=3 時，其中 A 區是最內圈的，E 區是最外圍，B、C、D 區則介在兩者之間，區域大小則是（C=B+D=A+E），如圖 1。

本研究採用 Log-based 的方式（如 LFS[6]、Logical Disk[5]、LinLogFS[4]），也就是說，這個檔案系統為 append-only，當資料區塊（data blocks）需要修改時是直接加在後面的可用空間，此時並建立新舊存放空間的對映，以後需要該資料區塊時即到修改後的存放空間存取。

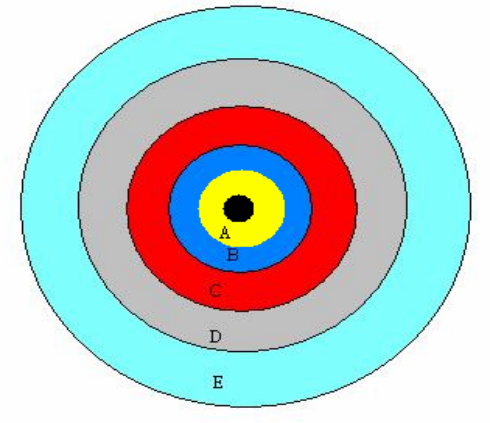

圖 1：分區的磁碟儲存空間

為達到動態地將常用的資料群集在硬碟中央的 cylinders 的目的，在本研究的方法中，以圖 1 為例，當有修改到 A、E 區的資料區塊的情況時，則將該資料區塊往 C 區的方向搬移到 B、D 區，若往後再有寫入的動作時，則再搬到 C 區，如此一來，愈常用的資料區塊就會集中到 C 區（亦即磁碟中央的 cylinders），至於較不常用的資料區塊，則逐漸由 C 區愈遠移到 B、D 區或 A、E 區，藉由此方式達到動態調整資料群集的目的。

另外在寫入資料的時候，本研究採用和 LinLogFS 相似的方法，盡量收集多次的寫入需求至 segment buffer 內，寫入的時候直接一次整個寫入在該區段資料區的後面，這樣可以減少原本多次寫入不同 sectors 必須移動讀取頭的次數以及 rotational latency，而減少所需的時間。

由於是採用 Log-based 的方式，資料區塊在經過修改後，會被移到新的區域去，上層檔案系統對磁碟驅動程式所發出的讀寫資料區塊的需求必需能夠照樣地順利完成。為了加速找尋到資料區塊目前的正確儲存位置，因此根據目前檔案系統的使用情形，在 RAM 中建立一個 block-mapping table 的資料結構，將檔案系統發出的讀寫資料區塊的需求透過此對照表轉換到我們存放該資料的正確區域，也就是說，原本要求讀寫哪些 blocks，透過對照表的轉換，而能夠知道資料實際上是存放在哪些實體硬碟 blocks 才對。

而在分區的想法方面，若將整個硬碟分成 n 個一樣大小的空間，在各區是有固定大小的情況下，可能會有某個區域堆滿了資料且無法再放進資料而產生錯誤，但其他還有區域實際上還有可用空間的情形；因此目前採用的方法是，當分區內的可用空間小於一設定值（threshold），則執行垃圾回收（garbage collection）動作。

此垃圾回收的動作基本上是採用 Log-based 的系統都必需做的動作，因為 Log-based 的系統是 append-only，當資料區塊需要修改時是直接寫在磁碟其他的可用儲存空間，是 non-update-in-place 的方式，而舊的資料區塊所佔用的磁碟儲存空間就必需靠此垃圾回收的程序去回收。然而，若垃圾回收的成本太高或效益太低，亦即回收的空間太少時，如回收空間小於該分區大小的 5%，則將資料存放到相鄰的區域內，以滿足寫入的需求和避免明明別的區域還有空間卻無法寫入的問題。

至於垃圾回收方法，在許多研究[6]都指出，如何挑選節段(segment)來做垃圾回收會嚴重地影響清除成本及回收效益，目前本系統實作的是 greedy algorithm[6]，不過由於不同的垃圾回收方法，也會影響到可回收的 blocks 數目等，因此以後也應該再加上例如 Cost Benefit [6]等其他垃圾回收方法的試驗和評估。

在實作方面則可分為兩種方法，第一種是更改磁碟驅動程式（disk device driver）的部份，優點是比較簡單，不需要修改到檔案系統本身，而且可以用原來 Linux 的 Ext2 或 Ext3 檔案系統，但缺點是這樣可能比更動檔案系統本身的效率來得差；第二種則是修改檔案系統，優點是系統的執行會比較有效率，且比較能夠跨不同的儲存裝置，但是缺點是修改檔案系統的複雜度相對地較高。因此，本研究實作的方法是採用折衷的方法，在檔案系統和磁碟驅動程式中間設計一個轉換層（Pseudo Log Disk Driver），來達到目的，如圖 2。

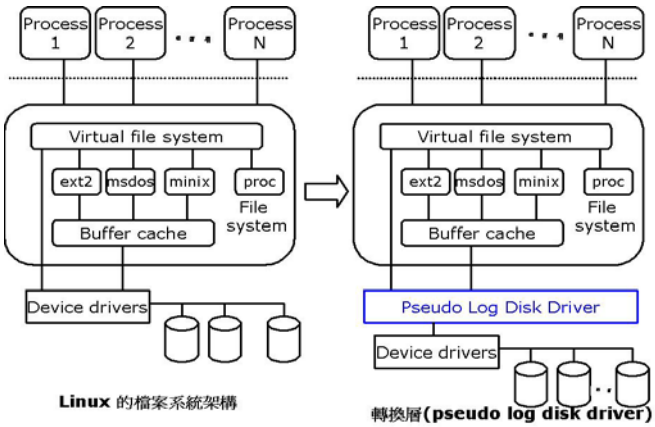

圖 2：轉換層（Pseudo Log Disk Driver）

四、 實驗方法

本研究以實作與模擬實驗 (simulation) 等方式來驗證我們所提的利用動態資料群集技術來提升系統的效能，而本論文著重在使用 trace-driven simulation 的方式來做系統的效能評估；有兩項主要的工作，即模擬器的製作與 trace 資料的收集。

在模擬（simulation）實驗中，我們使用 C++ 語言來實作一個模擬器，此模擬器提供一個測試環境，在其上先行試驗與測試應用動態資料群集的技術進行資料重整後，系統效能提升的情況，且此模擬環境亦提供許多的控制參數，可供進一步的相關研究。

4.1 模擬器的製作

此模擬器提供許多控制的參數，例如：提供兩種模式，應用我們所研發的資料重整技術的使用與否，另一方面也提供各種不同的 disk scheduling algorithms，如： FCFS、SSTF、SCAN、C-SCAN、LOOK、C-LOOK 等等，希望能夠測試我們發展的資料重整技術與各個 disk scheduling algorithms 對整體系統效能的提升，再加上 segment sizes 的測試，看系統所用的緩衝區需要多少可以達到理想的效能而且不會浪費系統資源。

此模擬器測量效能的指針是讀寫頭的移動距離，我們可以依著各種 disk scheduling algorithms

和參數，來模擬移動執行每個 request 來測試，然後計算讀寫頭移動的總距離，來當作評估效能的根據。

本模擬器的運作方式如圖 3，以一個 input trace 作為測試的資料來源，將它輸入模擬器中，首先，先建立硬碟的原始資料分配狀態，透過 input trace，將裡面的資料搜尋過一遍，將裡面有用到的硬碟磁區設定成已經使用，並且將它所需的相關資訊儲存來，在之後實際模擬的時候，來避免發生要讀某一磁區的資料而發生該磁區無資料存在的情況發生。

建立完初始資料後，然後再重新的從 input trace 將每一筆的資料讀入，判斷這個 request 是要讀取還是寫入；若是 request 是讀取的話，它會先進入 Write Buffer 中查詢看看要的資料是否在 Write Buffer 中，若沒有發現，則計算此資料在磁碟的實體位置，然後將此 request 放入 queue 中等待執行；若是這個 request 是要寫入的話，它會計算此資料是該放入哪一區，然後將這 request 放入該區的 queue 中等待執行，然後再檢查該 queue 是否已經是滿足一個 segment size，若沒滿足一個 segment size 的話，則繼續讀下一個 request，否則就去找該區域的空間是否有空的，若有的話就直接寫入整個 Write Buffer 的 segment 資料，若沒有的話則進入 garbage collection 的過程，清出空的空間，再寫入該空間，完成一次寫入的動作。

在進入 garbage collection 的過程中，若在第一次的 garbage collection 並沒有發現有多的空間或回收效率太差的話，則進行第二次的 garbage collection，但此次是找下一個相鄰的磁碟區域，若有空的磁碟空間則寫入，完成此次的寫入動作，若此次並不能寫入完成則代表磁碟空間以滿，已經無法寫入，而停止寫入的動作，然後藉由這些動作來完成模擬器的模擬。

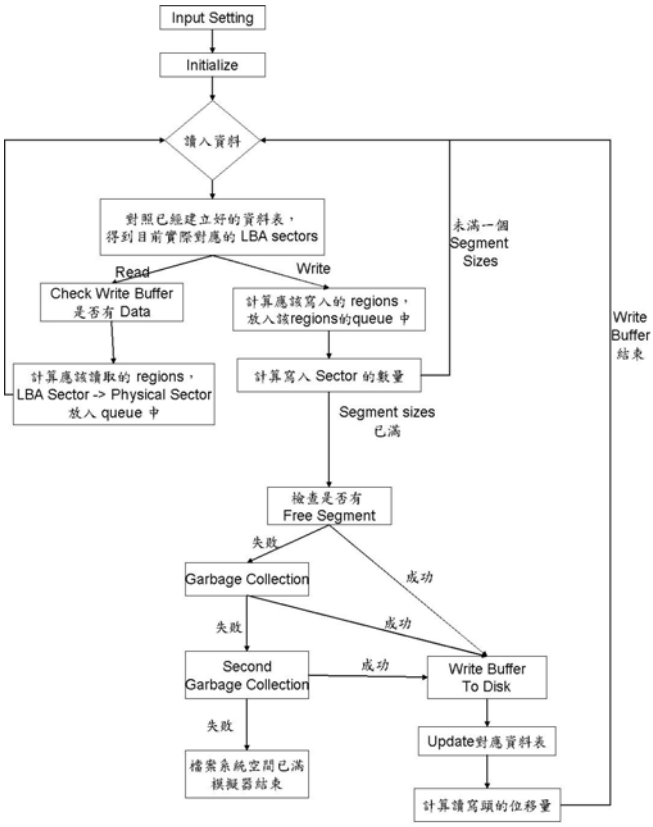

圖 3：模擬器的流程圖

4.2 trace 資料的收集

在 trace 資料的收集方面，我們在 Linux 上收集硬碟的 I/O 動作，做來模擬器的測試資料。在此處實作的方法是在 Linux 核心內加入程式碼，在原始碼 “init/main.c” 中，加入一個 collection_function ()，如表 1。

在這個 function 中，一開始是跟核心配置了一塊記憶體，用來暫存將來要收集的資料，然後將這些暫存的資料透過檔案系統的 system call 的方式，暫存的資料經過轉換然後寫入磁碟中。而這個 function 是利用 kernel thread 的方式啓動，如表 2。在系統開機的時候，把這個 function 啟動，執行收集 I/O 的動作。

然而要將此 function 從一開機就啟動，有個問題需要解決，就是當系統開機的過程，檔案系統是在只能讀取而不能寫入的模式下，所以必須將此

function 延遲個幾秒，等檔案系統變成讀取跟寫入都可以的時候再繼續執行此 function；在之後此 kernel thread 會定期地被叫起來執行此 function，寫入磁碟中實際的 I/O 的動作，這些 I/O 的動作包括讀寫要求發生的時間、讀或寫、block number、讀寫長度、資料型態、、、等等。

表 1： main.c 中新增的程式碼

init/main.c

```
void collection_function ()
{//在核心內配置一塊記憶體
change_temp = (char *) kmalloc (sizeof (char)*35,GFP_KERNEL);
timeout=10*HZ;
current->state=TASK_INTERRUPTIBLE;
timeout=schedule_timeout(timeout);
//因為在開機的模式只能讀不能寫，所以讓此 thread 暫停
for(;; ){
   sprintf(file_name,"/trace/%d",i);
   fd = open(file_name,O_RDWR | O_CREAT,0);
   for(j=0;j<1000;j++)
   {
      timeout=5*HZ;
      current->state=TASK_INTERRUPTIBLE;
      schedule_timeout(timeout);
      collection_end=index_collection;
      if(collection_start < collection_end)
      {
         for(k=collection_start;k<collection_end;k++)
         {
            sprintf(change_temp,"\n%lu %d %lu %lu %u
                  %c",collectionwrite[k].time,
                  collectionwrite[k].diskno,
                  collectionwrite[k].cyl,
                  collectionwrite[k].sector,
                  collectionwrite[k].rl,
                  collectionwrite[k].rw);
            write(fd,change_temp,35);
            memset(change_temp, '\0', 35);
         }
      ……
}
//將暫存記憶體存到檔案，而且每作 1000 次儲存記憶體的動作後，重開開檔繼續儲存
```

表 2：收集資料的 function 啟動方式

init/main.c

```
collection_thread_id=kernel_thread(collection_function, NULL, CLONE_FS | CLONE_FILES | CLONE_SIGNAL);
//將 collection_function 以 kernel thread 的方式啟動
```

在原始碼" drivers/block/ll_rw_blk.c"中，此檔案是負責磁碟 I/O 的所有動作，所以在此加入所需的程式碼，來收集 trace 的資料，如表 3；此檔案中有一個 function 是 ll_rw_block()，此函式可以知道每個 I/O request 的詳細資料，就在這邊我們將它存入我們所預先配置的記憶體內，然後等待以 kernel thread 方式啟動的 function 定期的將它從記憶體中寫到磁碟中；此處又會發生另一個問題需要解決，就是我們記錄 I/O 的那個檔案，也會進入我們收集的記錄中，因此，我們要將此情況排除，我們在將資料存到記憶體之前，先利用 process id，判斷每個 request 進來時，若非是此 kernel thread 的 I/O 才記錄，以達到收集資料的完整性與正確性。

表 3： ll_rw_blk.c 收集資料

drivers/block/ll_rw_blk.c

```
if(collection_thread_id != current->pid){
    collectionwrite[index_
collection].time=jiffies / HZ ;
    collectionwrite[index_
collection].sector=bh->b_rsector;
    collectionwrite[index_
collection].diskno=bh->b_rdev;
}
```

五、 實驗結果與討論

本模擬器所執行的作業系統是 Windows XP，使用的開發工具是 Visual Studio 6.0，硬體是 CPU 1.2G，Memory 1G；Trace 資料收集執行環境作業系統是 Red-Hat 7.2 with WD-BBS，Linux 核心版本是 2.4.17，硬體是 CPU 1.6G，

Memory 256MB，硬碟 40G。

此模擬器是以 HP I/O trace[8]和我們在系上的BBS Server上所收集的磁碟I/O動作當作此模擬器的 input trace，在透過模擬器相關參數的設定，例如是否要執行設計的演算法，各種 disk scheduling 方式，還有 segment sizes 的參數設定，測試結果如圖 4 至 6 所示。

我們實驗了在不同的分區(1 區及 3 區)下，以及不同的 segment size 的設定(512 KB 至 2048 KB)之下，實驗結果顯示當 Log-based 系統採取有我們所提的動態資料重整技術的功能時，不論是在何種的 disk scheduling algorithm 之下，都可以省下相當多的讀寫頭移動距離，因此，對檔案系統的效能可以有大幅地提昇。

我們預計在加入了其他 policies 的探討至 simulator，如 segment cleaning policy 與 active data clustering policy 等不同的 policies 的調校後，效能會有更大幅的提昇，而整體電腦系統的效能也會因檔案系統效能的提昇而有更顯著的效能提昇。

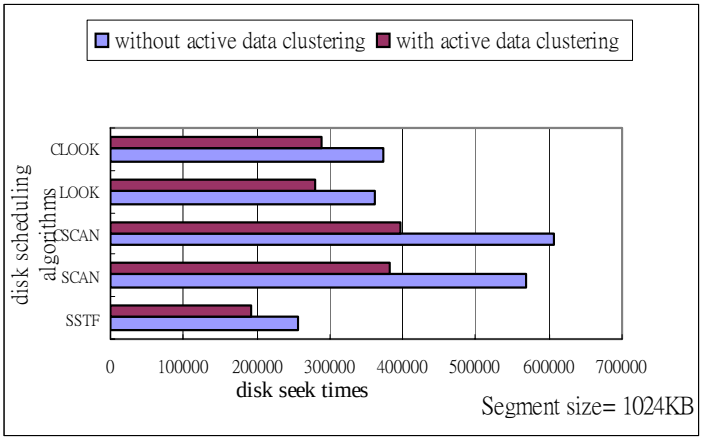

圖 4: 效能評估 (HP I/O Trace, segment size 2048 KB)

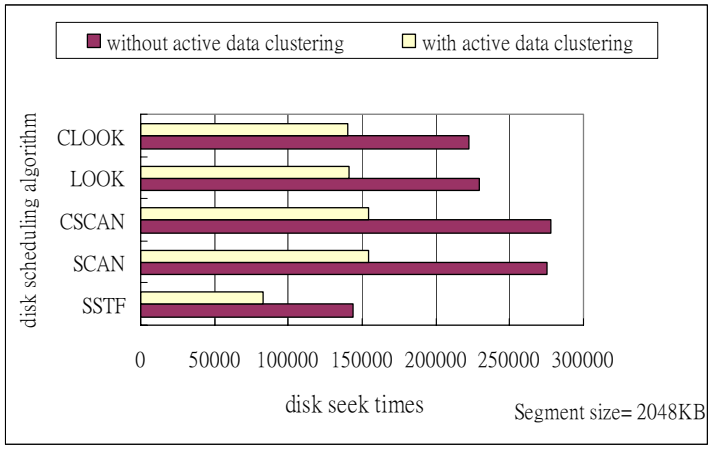

圖 5: 效能評估 (HP I/O Trace, segment size 1024 KB)

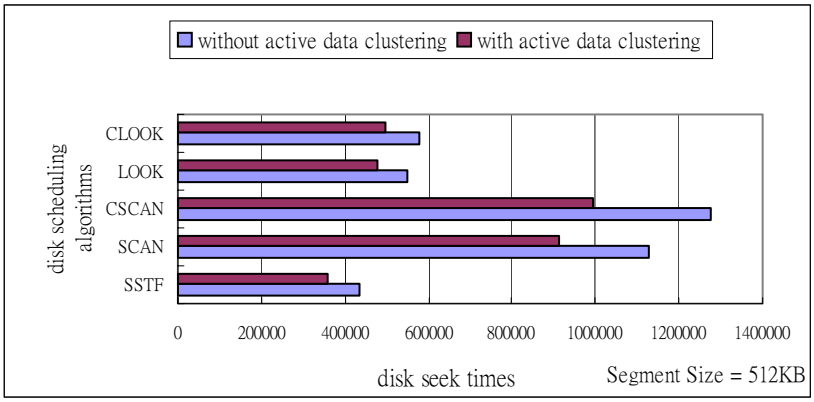

圖 6: 效能評估 (HP I/O Trace, segment size 512 KB)

六、 結論

我們採用 log-based 的資料存取方式，提出一個新的動態資料群集技術，並將之應用於動態磁碟資料重整中，目的是藉由使用此對檔案系統作分區的動態群集技術，將常用的資料動態地集中在磁碟中央的 cylinders，以減少資料在存取時磁碟的 seek

time，希望能大幅地提升磁碟儲存系統的效能。而此動態資料重整與傳統的磁碟資料重整方式不同的是在寫入資料的同時，即已依資料的 access history 而被群集，並不需如傳統的磁碟資料重整的運作方式，需要浪費額外的硬碟儲存空間，或需是在離峰時段來執行，不然會嚴重影響使用者正常運作的反應時間。

透過模擬實驗的測試結果，可以發現使用動態資料群集技術於動態磁碟資料重整中可以減少大約 35%的讀寫頭移動的距離，減少了很多的讀寫頭移動時間，若是將此動態資料群集技術應用到伺服器的主機，將可以提高磁碟儲存系統的效能，減少不必要的讀寫頭移動等待的時間，使伺服器的磁碟機壽命變長，整體系統提昇效能。

誌謝

本研究由國科會大專學生參與專題研究計畫編號 NSC90-2815-C-260-004-E-及國科會專題研究計畫編號 NSC91-2213-E-260-022- 所支持。

參考文獻

1. S. Akyurek and K. Salem, “Adaptive Block Rearrangement”, ACM Transactions on Computer Systems, 13, (2), 89-121, 1995.
2. S. Akyurek and K. Salem, “Adaptive Block Rearrangement Under UNIX”, Software-Practice and Experience, 27, (1), 1-23, January 1997.
3. D. P. Bovet, M. Cesati, Understanding the Linux Kernel – 2nd edition, O’Reilly & Associates, Inc., December, 2002.
4. C. Czezatke, dtfs, A Log-Structured Filesystem For Linux, Ph. D. Thesis, August 8, 1998, http://www.complang.tuwien.ac.at/czezatke/linlogpapers.html.
5. W. de Jonge, M. F. Kaashoek, and W. C. Hsieh, “The Logical Disk: A New Approach to Improving File Systems”, Proceedings of 14th Symposium on Operating Systems Principles, pp.15-28, 1993.
6. M. Rosenblum and J. K. Ousterhout, “The Design and Implementation of a Log-structured File System”, ACM Trans. Computer Systems, 10(1), 26-52, 1992.
7. C. Ruemmler and J. Wilkes, “Disk Shuffling”, Technical Report HPL-91-156, Hewlett-Packard Laboratories, Palo Alto, CA, October 28, 1991.
8. C. Ruemmler and J. Wilkes, “UNIX Disk Access Patterns”, Proceedings of the 1993 Winter USENIX, San Diego, CA, Jan. 1993.
9. A. Silberschatz, P. B. Galvin, and G. Gagne, Operating System Concepts – 6th ed, John Wiley & Sons, Inc., ISBN 0-471-36414-2, 2001.
10. C. Staelin and H. Garcia-Molina, “Clustering Active Disk Data to Improve Disk Performance”, Technical Report CS-TR-283-90, Department of Computer Science, Princeton University, September 1990.
11. C. Staelin and H. Garcia-Molina, “Smart Filesystems”, Proceedings of the 1991 Winter USENIX, pp. 45-51, Dallas, TX, 1991.
12. P. Vongsathorn and S. D. Carson, “A System for Adaptive Disk Rearrangement”, Software-Practice and Experience, 20, (3), 225-242, 1990.