國立交通大學

網路工程研究所

碩 士 論 文

小型同位元檢查磁碟：
一個改善 RAID 平行度的技術

Small Parity Check Disks:
An Technique to Improve RAID Parallelism

研 究 生：黎光仁

指導教授：張立平　教授

中 華 民 國　九 十 七　年 三 月

小型同位元檢查磁碟：
一個改善 RAID 平行度的技術

Small Parity Check Disks:
An Technique to Improve RAID Parallelism

研 究 生：黎光仁　　Student：Guang-Ren Li

指導教授：張立平　　Advisor：Li-Pin Chang

國 立 交 通 大 學

網 路 工 程 研 究 所

碩 士 論 文

A Thesis

Submitted to Institute of Network Engineering of

College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science

June 2007

Hsinchu, Taiwan, Republic of China

中華民國九十七年三月

小型同位元檢查磁碟：一個改善 RAID 平行度的技術

學生：黎光仁　　指導教授：張立平

國立交通大學 資訊工程 學系（網路工程 研究所）碩士班

摘　　要

在一般的電腦系統的環境裡，磁碟是主要的儲存設備，並且爲了加大儲存裝置的容量，和提升存取速度的效能，所以會使用磁碟陣列的方式來做儲存裝置，雖然磁碟陣列使用 stripe 的方式來提升平行度，但在實際上的工作情形裡，寫入的工作並不會因爲 stripe 的方式而有效的分散在各個磁碟上，造成實際上系統的工作量並不平衡，因此在這裡我們提出了利用 erasure code 編碼方式的特性，把資料進行擴充，之後只要取回少量的資料便可得到原本的資料，藉此來改善系統的平行度，並且經由實驗結果得知，我們只需額外增加2個磁碟，便能使得系統的回應時間得到有效的改善。

關鍵字: **disk array, storage systems, RAID**

Small Parity Check Disks: An Technique to Improve RAID Parallelism

student：**Guang-Ren Li** Advisors：Li-Pin Chang

Department（Institute of Network Engineering）of Computer Science
National Chiao Tung University

ABSTRACT

Disk is the primary storage in modern computer system. In order to provide more storage capacity and better the access speed, we can use disk array as the storage device. Although disk array use the stripe method to enhance the parallelism, the writing jobs don't distribute averagely amount all disks in real usage, it causes the unbalanced loading. Therefore we propose a method to increase the parallelism by using the characteristic of erasure code and extending the jobs to get the data just with fewer accesses. And according to the experiment result, with just two additional disks we make improvement of the system response time.

Key word: disk array, storage systems, RAID

誌　　謝

終於抵達寫致謝詞的這一刻，這意味著研究所生涯即將正式的畫上句點，回顧兩年來的經歷，內心充滿幸福與感謝，首先誠摯的感謝指導教授張立平老師，老師悉心的教導使我得以了解儲存系統的深奧，不時的討論並指點我正確的方向，使我在這些年中獲益匪淺。因為有你的體諒及幫忙，使得本論文能夠更完整而嚴謹。

二年多的求學生涯裡，讓我學習了不少新的知識和待人接物的方法，非常感謝每個老師的敦敦教悔，而且勤而不懈的教導我，讓我能夠一路走來都很順利。

再來，感謝在這段期間，游仕宏、張譽璿、曾士豪、杜俊達同學的幫忙，使得我能順利走過這兩年。實驗室的黃千庭、許辰暉、林松德、鄭家明、許惠茹…等學弟妹們，當然也不能忘記與你們相處的日子。還有各位在新竹一起奮鬥的朋友們，在這非常感謝你/妳們的幫忙、陪伴及搞笑我深刻地銘記在心，由衷感謝各位的相助。謝謝！

研究口試期間，感謝羅習五老師、陳雅淑老師不辭辛勞細心審閱，不僅給予我指導，並且提供寶貴的建議，使我的論文內容可以更完善， 在此由衷的感謝。

感謝系上諸位老師在各學科領域的熱心指導，讓我增進各項知識範疇，在此一併致上最高謝意。

最後感謝我摯愛的雙親和家人，因為他們的支持，才有此篇論文的產生。

目　　錄

圖　目　錄

Small Parity Check Disks: An Technique to Improve RAID Parallelism

Guang-Ren Li and Li-Pin Chang
Department of Computer science
National Chiao-Tung University, Hsin-Chu, Taiwan 300, ROC
lisunman@yahoo.com.tw,lpchang@cs.nctu.edu.tw

Abstract

在一般的電腦系統的環境裡，磁碟是主要的儲存設備，並且爲了加大儲存裝置的容量，和提升存取速度的效能，所以會使用磁碟陣列的方式來做儲存裝置，雖然磁碟陣列使用 stripe 的方式來提升平行度，但在實際上的工作情形裡，寫入的工作並不會因爲 stripe 的方式而有效的分散在各個磁碟上，造成實際上系統的工作量並不平衡，因此在這裡我們提出了利用 erasure code 編碼方式的特性，把資料進行擴充，之後只要取回少量的資料便可得到原本的資料，藉此來改善系統的平行度，並且經由實驗結果得知，我們只需額外增加2個磁碟，便能使得系統的回應時間得到有效的改善。

Keywords: disk array, storage systems, RAID

1. Introduction :

現在在電腦儲存裝置方面的發展很迅速，不過因爲在速度和價格方面上的種種因素考量，在大多數環境裡的應用，主要還是使用磁碟來當主要的儲存設備，因此磁碟的速度對整個電腦系統的運作速度依然占了絕大的因素。並且由於在速度上的提升幅度， CPU 遠比磁碟來的大的許多，因此磁碟的速度變成現在電腦系統在存取速度上的瓶頸。雖然以前有許多人在單一個磁碟上，提出了一些有效提升速度的方法 [12, 13]，但是所能提升的效能也是有限的，因此便開始考慮使用磁碟陣列的方式去加快資料的存取速度。並且由於在一些像是儲存系統方面的應用，常常需要存放大量的資料，因此需要一個大容量的儲存空間，但是對於大容量的磁碟來說，價格會比小容量的磁碟來得貴的許多，所以通常都會透過使用 RAID 這種方式，來建立一個價格合理的大容量儲存系統。

在使用 RAID 這個方法做儲存系統後，存取速度和容錯能力這兩方面已經提升了許多，容量也可任意的擴大，並且可以透過使用 stripe 的方式，把連續的資料分散存放在各個磁碟上，之後可以同時在數個磁碟上存取資料來提升平行度，但是實

際上磁碟陣列的工作量可能並不平衡，原因是因爲實際上寫入的資料通常都是小的且在固定位置上，並不會因爲 stipring 的動作而分散到各個磁碟上，所以有可能會一直寫入到相同的磁碟上，造成各個磁碟的工作量並不平衡，使得這個磁碟變成了系統效能的瓶頸，拖累了整個系統的效率。

這時候一般的解決方法有兩種，首先第一種是會再加入新的磁碟，去把資料分散的更開，好讓工作量能更平衡，但是由於 striping 資料的時候，每一筆資料都會橫越在數個的磁碟上，因此當在增加或減少磁碟的時候，必需重新安排位置，因此需要把系統停下來，將資料重新放置到相對映的位置上，而這樣的動作可能會對使用者造成不便，且全部的資料做重新的安排動作可能需要花費很長的時間。因此便考慮第二種方法，針對原先變成瓶頸的那個磁碟，用 mirror 的方法去增加一個它的複本磁碟，藉此來分擔造成瓶頸的磁碟的工作量，這個方法雖然把這個磁碟所造成效能上的瓶頸的問題解決了，但是相對的空間使用率變的差了許多，並且在另一方面，在 RAID 系統裡的所有磁碟的性能皆需相同，但由於新出產的磁碟和舊有的磁碟在性能上必定有所不同，因此 RAID 在爲了提升效能而去擴充或更新磁碟時，可能並無法完全的充分利用新加入的磁碟的性能。

基於以上的種種原因，我們提出了一個不同於以往一般 RAID 系統所使用的解決方法，來改善原有 RAID 系統的平行度，來提升回應時間(response time)，這個方法是利用 erasure code 的編碼方式，將所需的任何資料任意擴編，之後在想要讀取這些資料時，可以閃避因寫入的工作(request)而造成忙碌的磁碟，任意從較不忙碌的磁碟上取回較少份的片段，即可動態的重組回所需要的資料，圖1就是一個 erasure code 幫助工作量平衡的例子，在這裡要取回{ D_a , D_b , D_c }，原本必須從磁碟1和磁碟2取回的資料，因爲經過 erasure code 編碼後，在磁碟3上有 D_b 的多餘物(redundancy)，而磁碟4上有 D_c 的多餘物，所以可以從磁碟3和磁碟4上取回，便可把這兩個工作移到磁碟3和磁碟4上執行，藉此方法來讓工作量平衡，提升系統效能。

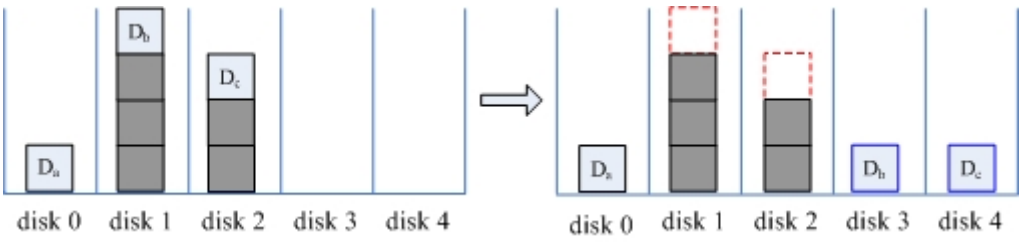


圖 1: Erasure code assist in load balance

我們的方法和 mirror 方法不同的地方是在於 mirror 方法是一份資料就要建立一份複本，而之後在取回的時候，這份資料只能從其原始資料或複本的地方取回，而我們的方法可以利用一份多餘物去減輕數個磁碟上的工作量，且除了原始資料外，還可以透過取回其 xor 後產生的任何多餘物，對它所相對映的資料一起做 xor 後取回資料，回復的方法彈性加大，並且在取回資料的時侯，一個多餘物可以給數個資料一起使用，在空間使用率上大大的提升。雖然我們的方法是以原本 RAID 系統爲基礎來建立的，但是我們新加入的磁碟並不需放入到原本的 RAID 系統裡，所

以當加入新的磁碟的時候，我們的方法不必像磁碟替換一樣需要暫停整個系統，對整個系統進行資料的重新放置，只需把之後所產生的多餘物放至到這磁碟上即可，這些是原本 mirror 方法和磁碟替換所辨不到的，因此我們可以透過這樣的方法，來建立一個較有彈性空間使用度較高且較快的儲存系統。

2. Related work

因爲磁碟造成儲存系統存取速度上的瓶頸，所以在改善磁碟的存取效率這方面的問題，也已經被研究了一段時間，並且因爲磁碟陣列的應用，當新增或移除磁碟的時候，會需要做資料重新放置的動作，因此在資料搬移方面也被提出來許多相關的研究。

Brinkmann et al. 提出 Cut-and Paste strategy 這個方法，可以在一個動態改變的儲存環境裡，把資料區塊有效率且均勻的分散出去 [1]，這個方法首先假設存在一個雜湊函數可以把所有的資料區塊均勻的分散到各個磁碟上，則當增加磁碟的時候，就從原本的 n 個磁碟剪下區域 [1/n+1, 1/n] 的資料，再把這些資料貼到新加入的第 (n+1) 個磁碟的區域 [0, 1/n+1] 上，而當在移除磁碟的時候，則把這個動作反過來做，把在要被搬移的磁碟上的資料搬到其它的磁碟上即可。Khuller et al. 提出 Cloning-based data migration 這個方法，是把原本資料搬移問題的解決方法，進一步的延伸成透過資料的複製來解決，並且另一方面也能提升資料傳輸的平行度 [2]。

Azer Bestavros 提出把 information dispersal algorithm (IDA) 運用在磁碟陣列上，透過 IDA 這個編碼的方法，把資料從 n 份擴充爲 m 份，之後只要從這 m 份中任意取回 n 份即可回復成原本的 n 份資料 [6, 7]，讓資料分散存放在各個磁碟上，來提高容錯能力和效能。Guillermo A. Alvarez et al. 提出 DARUM ，在磁碟陣列上可以容忍多個錯誤發生的方法，並且把 IDA 這個方法加以改進，把原本的資料和增加多餘物的部份加以區分出來 [9]，再把資料透過這個方法來分散到各個磁碟上，來提高容錯能力和效能。

erasure code 是一個編碼的技術，把原本 n 份資料擴充爲 n+k 份資料，接著我們只要取回 n+k 份資料中的任一個子集合 S (|S| ≧ n)，即可回復成原本的 n 份資料，假如 |S| = n 則爲 optimal erasure code ，需要靠大量的計算才能達到，像 IDA [6, 7]和 Reed-Solomon Codes [10]，而 sub-optimal erasure code 則有 Tornado Code ，low-density parity check (LDPC) [5]，small parity check[4]，把 erasure code 使用在儲存系統上面已經不是一個新的想法了，像是把 simple parity check 或是 mirror 應用到 RAID 系統上的 RAID-5, RAID-0+1, RAID-1+0 來進行容錯的磁碟陣列，或者是可以容忍兩個錯誤的 RAID-6, EVENODD [8]，更進一步的把 IDA 和 Reed-Solomon Codes [10] 拿來運用，可以容忍大數量的錯誤發生，雖然這些都是拿 erasure code 來運用到儲存系統上，但是主要的目的是用來容錯之用，和我們的目的要提升儲存系統的平行度來的不同。

3. Problem definition and system model

3.1 Disk-Array Organization

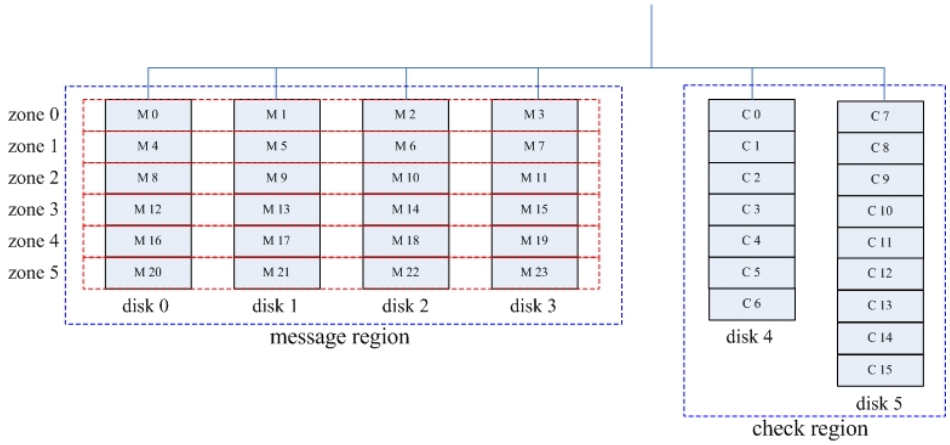


圖 2: System architecture

圖2是我們的系統架構圖，把所有的磁碟分在兩個大區域， message region 和 check region ，每個磁碟只會存在其中一個區域裡，message region 是一個 RAID 5 的系統，我們附加的 parity check 是放在 check region 裡，每當我們需要提升系統平行度的時候，可以動態的把磁碟加入到 check region 裡，而這些新加入的磁碟的特性，可以不必和原本在 check region 裡的磁碟的特性相同。

在我們的系統裡有兩種不同的磁碟，在 message region 裡的磁碟叫做 message disk ，而其它在 check region 裡的磁碟則叫做 check disk ，我們再把所有的磁碟切成數個大小相同的區域，而這些區域叫做 segment ，一個 segment 是由數個 stripe block 所組合而成的，一個 stripe block 是由數個 sector 所組合而成的，而這些 stripe block 也就是 RAID-5 系統把 data stripe 到各個磁碟上的單位，而 segment 則是我們去增加 redundancy 的單位，在 message disk 上的 segment 叫做 message segment ，也就是圖中的 “M”，在 check disk 上的 segment 叫做 check segment ，也就是圖中的 “C” ，並且我們把 message region 分成數個 zone ，每一個 zone 是由連續的 message segment 所組成的，包含各個 message disk 上的一個 message segment 。

3.2 RAID systems and performance issues

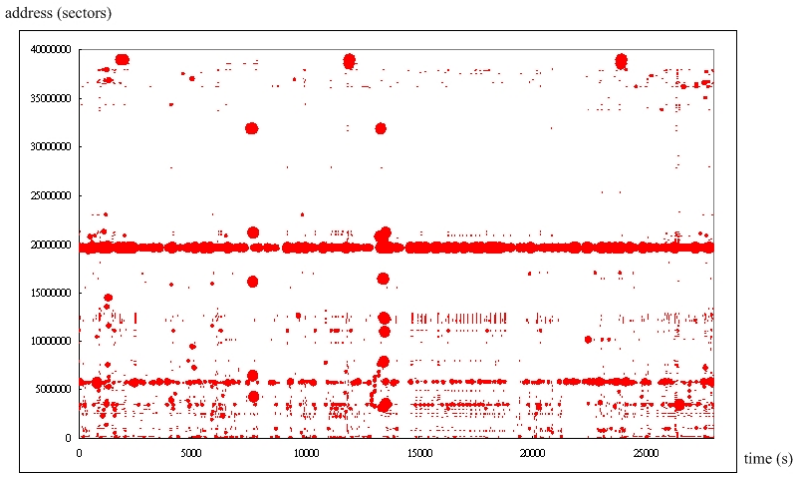


圖 3: Distribution of write request

因爲 metadata 經常需要更新，所以會常常需要寫入的動作，並且 metadata 的尺寸是很小的，且具有區域性的特性，圖3是寫入的工作的分佈情形，紅色的點就是寫入的工作，我們可以看到寫入的工作總是小尺寸的，且常寫入到相同的地方，可能總是落在相同的磁碟上，因此我們發現到寫入的工作是具有時間區域性的特性，儘管有使用 stripe 去把資料分散到各個磁碟上面，但是由於有這些常常被寫入的地方，所以造成實際上工作量仍然不平衡，而這些寫入的工作所在的地方就變成系統效能的瓶頸了。

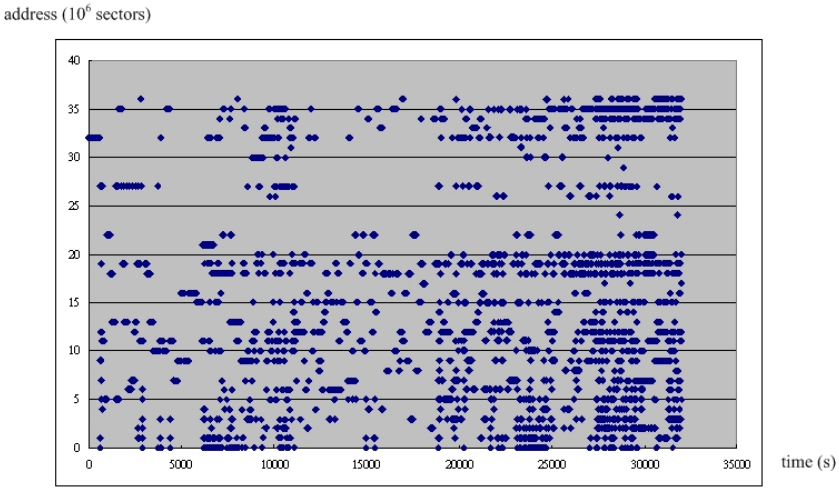


圖 4: Distribution of read request

因爲系統裡面已經有了快取記憶體，所以實際上不存在常常被讀取的資料，但是參考實際上的工作情形，我們可以發現到讀取的工作是具有空間區域性的特性，圖4是讀取的工作的分佈情形，藍色的點就是讀取的工作，這些被讀取的資料幾乎都分佈在特定的幾個 zone 裡，而會造成這種情形發生的原因，可能是因爲檔案系統在放置這些邏輯上相關的資料時，會放在相鄰的區域裡，所以讀取的工作具有空間區域性的特性，舉例來說，像是當在同一時間複製到儲存系統裡的資料，或是被放置

在相同的資料夾裡的資料，當其中一份資料被讀取的時候，其它相鄰的資料也很可能會跟著被讀取。

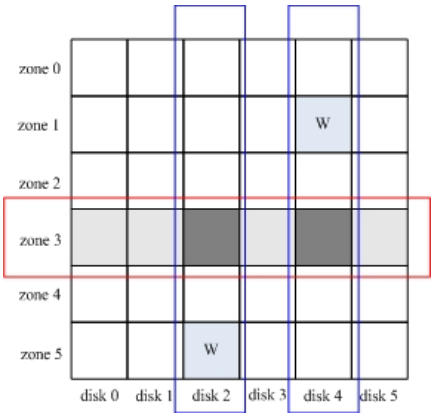


圖 5: System bottleneck

圖5是在說明在一般的 RAID 系統上的性能瓶頸， “w”方塊是常常被寫入的區域，當讀取的工作要從磁碟2和磁碟4上讀取資料的時候，則會常常被這個常被寫入的地方給擋住而造成延遲，而 zone 3 是常常被讀取的區域，灰色的方塊就是這個系統效能的瓶頸所在，基於上述的這些原因，這些在磁碟2和磁碟4上的讀取的工作必須被搬移到別個磁碟上去執行，所以我們將利用 small parity check 去改善工作量平衡，增加儲存系統的平行度，讓讀取的工作可以選擇讀取 parity check 來去閃躲忙碌中的磁碟。

3.3 Erasure resilience vs. load balancing

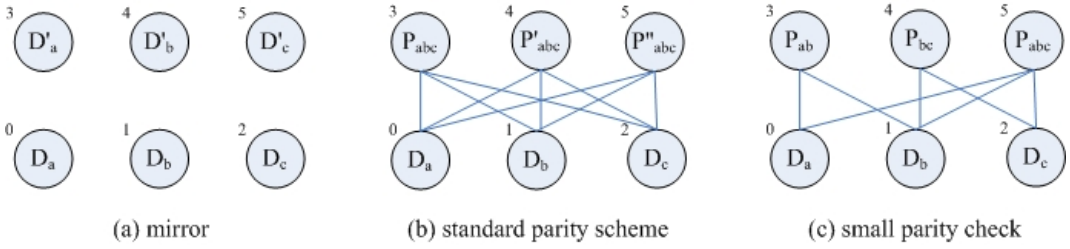


圖 6: Mirror, standard parity scheme, small parity check

多餘物是由原始資料經由運算後所產生的資料，通常是被用來改善錯誤容忍度，但是我們認爲多餘物也可以拿來提升系統的平行度，所以在我們的研究裡焦點是使用多餘物來提升系統的平行度，在這一節裡我們將比較三種方法 : mirror, standard parity scheme, and small parity check [4]。

在圖6裡， D 代表 data，而 P 則代表 parity check ， $P_{abc} = P'_{abc} = P''_{abc} = D_a \oplus D_b \oplus D_c$. $P_{ab} = D_a \oplus D_b$. $P_{bc} = D_b \oplus D_c$ ，在每一個圓的左上角的數字代表著這個 data 或 parity check 所被存放的磁碟的編號。 mirror 是一種把原本的資料整個完全的複製一份複本的方法，利用這樣複製資料後，當要取回資料的時候，可以選擇

從原始的資料或複本來取回所需的資料，例如 :圖6.a就是一個 mirror 的例子，假如 D_a 壞掉的時候，就可以從 D_a‘ 取回。

而 standard parity scheme 是被 RAID-5 所採用的方法， parity check 是一種由每個磁碟上的資料一起經 xor 後所產生出來的資料，之後假如其中一個磁碟上的資料發生遺失，則可以經由其它磁碟上的資料一起經 xor 後來回復，例如 :圖6.b就是一個 standard parity check 的例子，假如 D_a 壞掉的時候，就可以透過取回 $\{D_b, D_c, P_{abc}\}$ 或 $\{D_b, D_c, P'_{abc}\}$ 或 $\{D_b, D_c, P''_{abc}\}$ 來回復。而 small parity check 是一種以 standard parity scheme 爲基礎加以延伸的方法， small parity check 和 standard parity scheme 不同的地方在於 small parity check 可以不規則的去增加 parity check ，相同的是 small parity check 也可以使用 parity check 去回復資料，例如 :圖6.c就是一個 small parity check 的例子，在這裡有 $\{D_a, D_b, D_c\}$ 三份資料和 $\{P_{ab}, P_{abc}\}$ 兩份 parity check ，假如 D_a 壞掉的時候，可以透過取回 $\{D_b, P_{ab}\}$ 或 $\{D_b, D_c, P_{abc}\}$ …等等方法來回復資料。

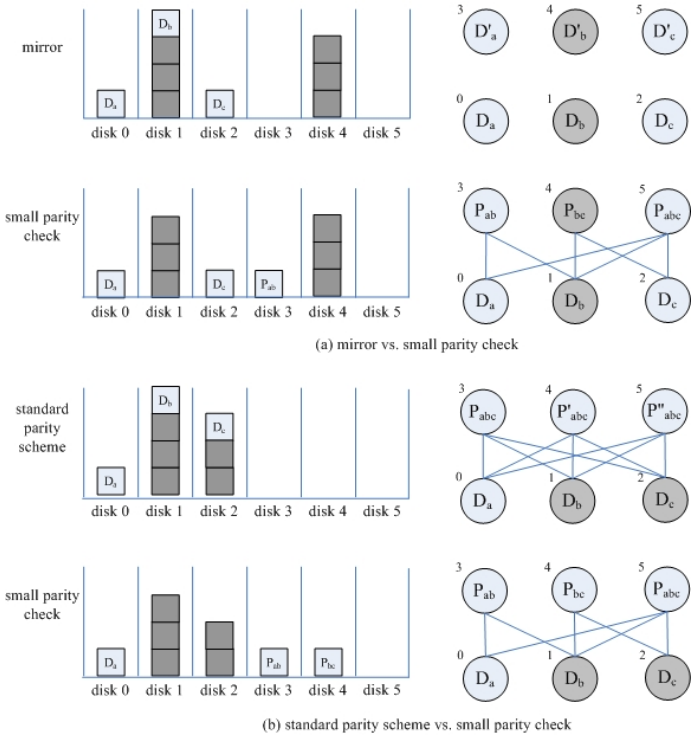


圖 7: Mirror vs. small parity check and standard parity scheme vs. small parity check

圖7是比較 mirror, standard parity scheme 和 small parity check 這三種方法的平行度的一個例子，藍色的框代表著各個磁碟的工作序列，淡藍色的方塊代表著我們所要取回的資料的工作，灰色的方塊代表著已經存在於磁碟的工作序列裡等待執行的工作，在這些例子中，我們要取回 $\{D_a, D_b, D_c\}$ 這三筆資料。圖7.a是 mirror 和 small parity check 的比較，對於 mirror 而言，這個工作將會被擋住，因爲 D_a 只能從磁碟1和磁碟4取回，而對 small parity check 而言，這個工作不會被擋住，因爲可以透過取回 $\{D_a, D_c, P_{ab}\}$ 來回復。圖7.b是 standard parity scheme 和 small parity

check 的比較，對於 standard parity scheme 而言，這個工作將被擋住，因爲 standard parity scheme 只允許原本存放資料的一個磁碟在忙碌，而對 small parity check 而言，這個工作不會被擋住，因爲可以透過取回 $\{D_a, P_{ab}, P_{bc}\}$ 來回復，同樣的例子也可以應用在 P_{abc} 代替 P_{bc} 。

基於上述的原因，可以發現 small parity check 的彈性比 mirror 和 standard parity check 還要好，雖然在錯誤容忍度方面 small parity check 比 standard parity scheme 還差，但是在標準的 RAID-5 系統裡就已經有了錯誤容忍度的能力，所以我們使用 small parity check 來改善 RAID-5 系統。

接下來在我們的研究中，有一些技術上的問題 ：

1. 如何去分配多餘物來充份使用這些多餘物?
2. 如何去放置多餘物到我們的系統裡的 check disk 上，來讓各個 check disk 工作量平衡?
3. 用什麼樣的編碼方式來得到較多的取回資料方式?
4. 如何去排程這些工作來提升系統效能?

這些問題我們將在第四章裡討論。

4. An erasure-code-based load balancing technique over RAID

4.1 Small parity check on parallelism improvement

我們的儲存系統分成兩個階段： 線上(online)和離線(offline)。我們讓系統在線上的情況下去接受工作並且進行服務，爲了避免因爲調整 parity check 讓原本已經在忙碌中的系統變的更加忙碌，所以每隔一段時間，便讓系統進入到離線的階段，去進行 parity check 的調整，等調整完 parity check 之後，我們便讓系統再回到線上的狀態，重新去接受工作進行服務。

我們的方法主要可以分成四個部份 ：1. redundancy allocation 2. coding scheme 3. redundancy placement 4. request scheduling ，首先我們會在 redundancy allocation 決定如何分配這些多餘物，來充份使用這些多餘物，接著在 coding scheme 決定要用什麼樣的 coding scheme ，來得到較多的方式取回資料，再在 redundancy placement 決定這些被增加的多餘物，分別是要被放在哪些 check disk 上面，來達到工作量平衡，最後當我們要取回資料的時候，再透過 request scheduling 來決定是要從哪些磁碟上來取回，來充份提升系統效能。

4.2 Redundancy allocation

在分配 check segment 時，在技術上的問題是由於能分配的 check segment 個數有限，爲了充份使用這些 check segment ，所以必需找到接下來最有可能被讀取的區域，爲它們去增加 check ，讓那些常被寫入的磁碟上的讀取的工作，可以透過 check segment 去減輕工作量，並且由於讀取通常都具有空間區域性，所以在一段時間內常被讀取的區域，其相鄰的區域接下來最有可能被讀取，因此我們是找在這一段時間內最常被讀取的 zone 去增加 parity check 。

首先我們先把每個 message segment 被讀取的次數記錄下來，接著算出一整個 zone 的讀取次數，然後從被讀取最多次的 zone 開始去增加 parity check ，對 zone 裡的相鄰兩個 message segment 即增加一個 check segment ，並且對同一個 zone 的第一個 message segment 和最後一個 message segment 也增加 check segment ，等到這 zone 裡的 message segment 都被分配到 check segment 後，再找下一個被讀取最多次的 zone ，重覆上述這些動作，直到所有的 check disk 的空間都被分配完爲止，圖8即是一個 redundancy allocation 的例子，小方塊代表 message segment ，圓代表 check segment ， zone 1 爲被讀取最多次的 zone ，即爲整個 zone 都增加 check segment 。

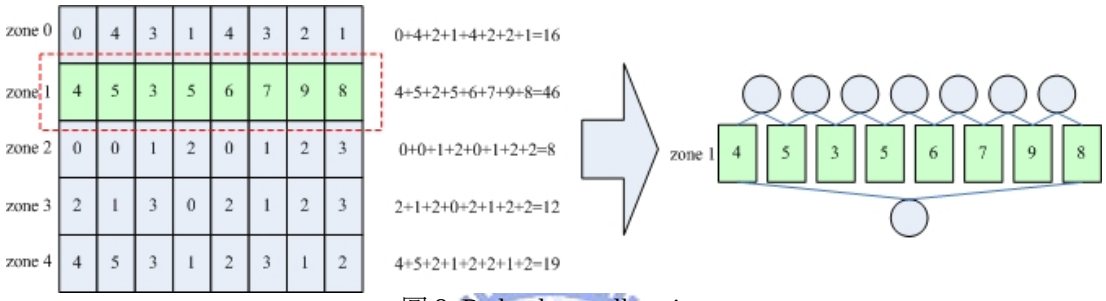


圖 8: Redundancy allocation

4.3 Coding scheme

當我們已經決定好多餘物的分配之後，接下來我們要決定編碼的方式，在技術上的問題是要考慮到如何能得到較好的空間使用率，以及額外產生的花費比較少，首先我們稱這個 message segment 和 check segment 相連接的圖爲 link graph ，並且把這個問題分成三個部份去考慮：1. 是要用多少 level 的圖，2. 是要用 degree 多少，3. 是要用哪種圖形。

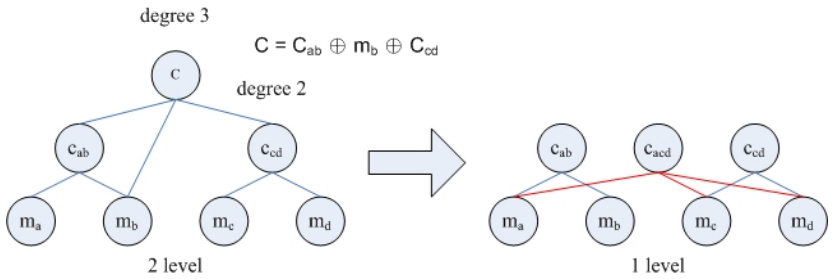


圖 9: Level and degree

首先在考慮 level 的部份，由於 link graph 的 level 是由 message segment 和 check segment 所組合而成的，而 check segment 又是由數個 message segment 互相做 xor 運算所得到的，並且由於任何的 message segment 若是和自己進行 xor 則會被消去，所以任何的 level 的 link graph 都可以化成 one level 的 link graph ，圖9是一個 two level 變成 one level 的例子， m 代表 message segment，而 c 則代表 check segment ，接下來的圖也是如此表示。

	m=1	m=2	m=3	m=4	m=5	m=6	m=7	m=8
n=3	8	16	10					
n=4	16	45	56	17				
n=5	32	121	162	176	26			
n=6	64	320	492	472	512	37		
n=7	128	841	1570	1737	1570	1408	50	
n=8	256	2205	4600	4785	4600	5469	3712	65

圖 10: Degree compare

接著我們使用 one level 去觀察各種不同的 degree ， degree 就是一個 check segment 連接到 message segment 的個數，圖10爲 degree 的比較， n 是 message segment 和 check segment 的個數， m 是 degree ，裡面的數字代表著取回和 message segment 個數相同的資料(message segment 或是 check segment)時，能得到我們所需的資料的方法數，由比較結果發現 mirror(m=1) 和 standard parity scheme(m=n) 時，方法數比較少，而在空間的花費上面， degree 大的比 degree 小的花費來的小，因爲可以較多的 message segment 共用同一個 check segment ，而在寫入時更新 check segment 的花費上面， degree 大的比 degree 小的花費來的多，因爲每當有一個 message segment 要被更新時，和它相連的 check segment 也就要跟著更新，因此我們決定使用 degree 小的來設計我們的 link graph 。

但是假如 message segment 不能透過 check segment 和別的 message segment 相連的話，當我們要取回資料的時候，這些 message segment 就不能利用別的 message segment 和 check segment 來進行回復了，所以我們不使用 degree one 去建立我們的 link graph ，因爲 degree one 無法和別的 message segment 相連，因此我們選用 degree two 來建立我們的 link graph 。並且假如這些 message segment 不能彼此透過 check segment 來相連的話，在取回資料的時候就無法透過那些沒相連的部份來取回，基於上述的理由，我們發現環狀具有較好的回復能力，因此我們使用環狀來建立 link graph ，圖11是一個 coding scheme 的例子。

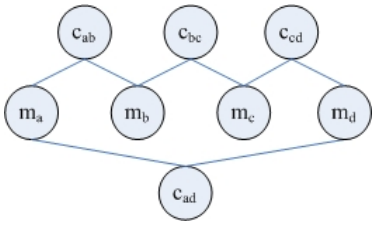


圖 11: Ring

4.4 Redundancy placement

當我們已經決定好如何增加 check segment 之後，接著我們必需決定把這些被增加的 check segment ，分別放至在哪些 check disk 上，在技術上的問題是要讓這些 check disk 能工作量平衡，此時必需考慮到各個 check disk 的性能可能不同，並且還要考慮到對於在同一個 zone 的 check segment 的放置，必需分別放在不同的 check disk 上，否則可能造成這些 check segment 無法同時被利用。

基於上述的理由，我們設計了兩個值， performance rate 和 load value去解決這個問題，每一個 check disk 有一個 performance rate ， performance rate 和 check disk 的性能有關，有較好性能的 check disk 的 performance rate 較小，性能較差的 check disk 的 performance rate 較大，因爲這樣才能讓性能較好的 check disk 分到較多的工作量，在經過增加 check segment 這個動作後，每一個 check segment 有一個 check load ， check load 就是在相鄰兩個 message segment 上全部的讀取工作的回應時間的差值除以二，代表著這個 check segment 可能分擔的工作量，而 load value 則是 check load 和 performance rate 的乘積，被使用來平衡各個 check disk 的工作量，每一個 check disk 有一個 total load value ， total load value 就是存放在這個磁碟上面的 check segment 的 load value 的總和。

這個演算法的動作就是當我們增加 check segment 的時候，找出 total load value 最小的 check disk ，把 check segment 放置到這個 check disk 上面，並且對相鄰的 message segment 所產生出來的 check segment ，放置到不同的 check disk 上，且當我們把 check segment 放置到 check disk 上後，便把 load value 加入到這個 check disk 的 total load value ，這個演算法重覆上述的動作，直到所有增加的 check segment 都放置完成爲止。圖12是一個 redundancy placement 的例子，長方形方塊代表磁碟，在 message segment 下的數字代表這個 message segment 的全部的讀取工作的回應時間，在 check segment 旁的數值代表 check load ，在磁碟上的數值代表 total load value ，而在磁碟下面的數值代表 performance rate，藍色的 c_{bc} 是目前要放置到 check disk 上的 check segment ，而小的藍色方塊是加入到磁碟的 load value ，這個演算法放置 c_{bc} 到磁碟2，因爲磁碟2的 total load value 是最小的，而 check load 爲 (1200-200)/2 = 500 ， load value 爲 0.3 × 500 = 150 把 load value 加入到磁碟2裡，磁碟2的 total load value 變成 400 + 150 = 550。

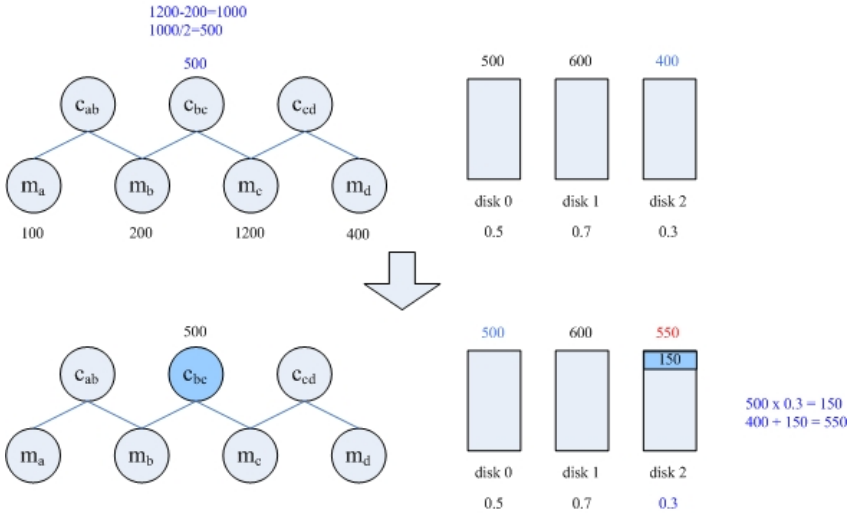


圖 12: Redundancy placement

以上所述之演算方法基本上是一種貪婪演算法的方法。採用這種方法的原因是因爲這個問題的複雜度相當高，爲 NP-hard，因而我們採取這種比較簡單快速的貪婪

演算法。欲證明該問題的 NP-hardness，我們可將 PARTITION [15]問題簡化到本問題的一個特殊例子，也就是假設僅有兩個效能一樣的 check disk。顯見想要將工作量平均，最佳狀況爲兩個 check disk 的 total load value 爲相等。

4.5 Request scheduling

當我們在離線時進行調整多餘物，把所有該增加多餘物的部份都增加了，並且把多餘物放在該放至的 check disk 上之後，便可以再度進入到線上的階段，讓系統重新開始運作，而在這時候我們便需要 request scheduling 來決定當有工作進來的時候，是要從哪些磁碟上來取回資料。

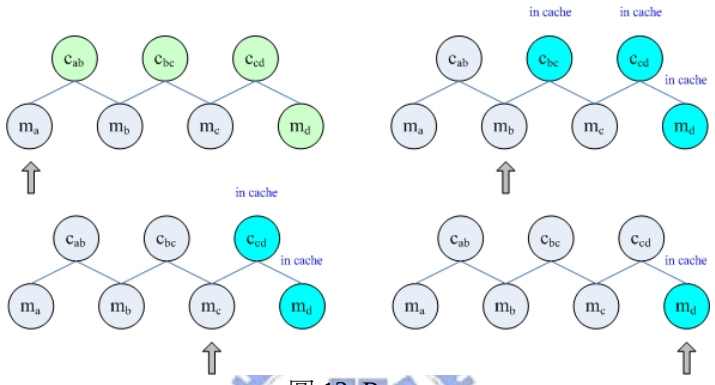


圖 13: Reuse

當我們需要的 message 所在的磁碟正在忙碌時，而我們透過取回目前尚未需要取回的資料部份，來幫助回復這份 message 時，在未來若是要讀取到這些事先被取回的部份，則可以直接從快取記憶體取回，不需再去磁碟上重新讀取一次，我們稱這個動作爲 reuse 。在這裡我們對於可以透過 check 來彼此連接的一群 message 和 check 稱它們爲一個 group ，當同一個 group 的 message 都要取回，因爲整個 group 的 check 都可以加以利用，所以此時的彈性最大，可以任意從 message 或 check 取回。圖13就是一個 reuse 的例子，當我們爲了得到 $\{m_a\}$ 時，而事先取回了 $\{c_{ab}, c_{bc}, c_{cd}, m_d\}$ 這些資料，之後當我們要取回 $\{m_b\}$ 或 $\{m_c\}$ 或 $\{m_d\}$ 時，便可直接從快取記憶體裡取得。

我們的方法主要是每當有一個工作進入到系統，便會把工作送入到這個工作所要取回的 message 和相連的 check 所在的磁碟上，接著每當取回一個資料，便看哪些工作可以透過這些資料做 xor 取回，就把那些工作取消，直到取回整份要求的資料就停止。並且我們有做一個 read-ahead 的功能，藉此來增加取回資料時選擇的彈性。

我們稱原本想要取回的 message 爲 original message ，而能幫助得到 original message 的 message 和 check 的集合爲回復路徑。在選擇從哪些磁碟上取回的時候，若這份資料已經存在快取記憶體，則直接從快取記憶體取回即可，因爲快取記憶體速度遠比磁碟快，所以已經存在於快取記憶體裡的資料，就直接從快取記憶體裡取回。而其它需要從磁碟上取回的資料，則是看哪一個工作先到達磁碟開始做，

就從那個磁碟機取回，因爲只有先到達磁碟的工作，才能確保比別的磁碟上的工作能早完成，而其它可以透過這些已經取回的資料 xor 出來的資料，則就可以直接把這些工作取消，不必再去從磁碟機上取回。

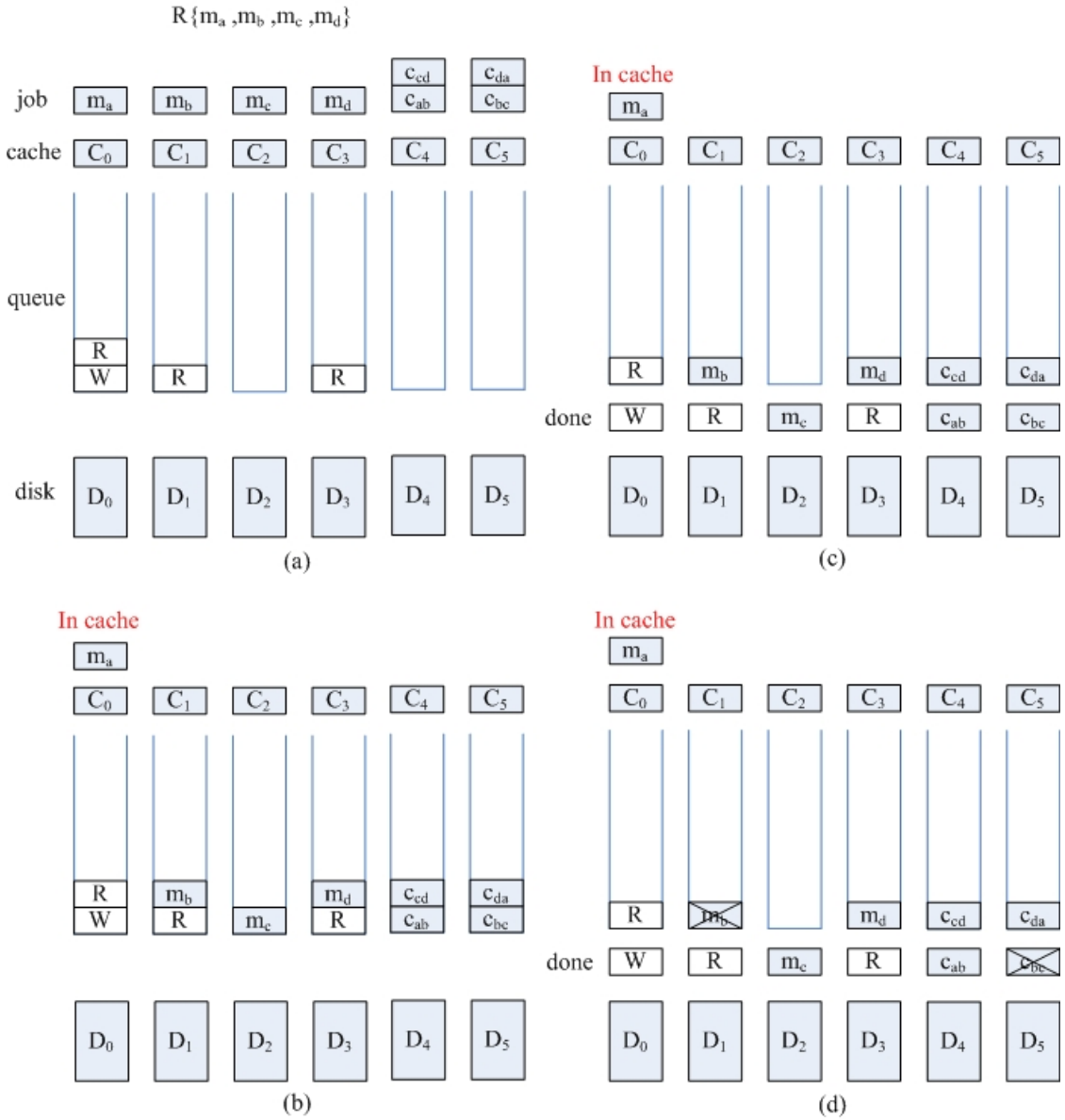


圖 14 : Request scheduling

圖 14是一個 request scheduling 的例子，由圖 14.a可以看到，當要取回{m_a ,m_b ,m_c ,m_d }這四筆資料時，送入{m_a ,m_b ,m_c ,m_d ,c_{ab} ,c_{bc} ,c_{cd} ,c_{da}}這八個工作到 controller 上，接著在圖 14.b 中，判斷可以直接從快取記憶體上取回{m_a}，便直接從快取記憶上取回，其它不能從快記憶體上取回的，則送入到 queue 裡，圖 14.c 從 queue 裡面取出工作送到磁碟上執行，圖 14.d 由於得到 {m_a ,c_{ab}} 即可 xor 得到 {m_b}，得到 {m_b ,m_c} 即可xor 得到 {c_{bc}} ，故可以把 {m_b ,c_{bc}} 這兩個工作取消，重覆這樣的動作直到得到{m_a ,m_b ,m_c ,m_d }資料爲止。

4.6 A read-ahead policy

因爲大的工作能得到較好的彈性去分擔工作量到各個磁碟上，所以爲了能提供較好的彈性，在這裡我們使用了一個 read-ahead 的方法，來讓小的工作可以轉換成大的工作來執行，並且我們透過一些限制，使得這個方法使用起來能更有效的執行 read-ahead 的動作。

經由分析我們所收集的磁碟工作記錄，我們發現到兩個潛在的問題。第一個，一些連續的讀取工作會被切開成許多小的讀取工作，第二個，即使這個讀取工作很大，但並無法和 zone 對齊。因此爲了處理這兩個問題，我們的 read-ahead 有採取一些機制去把小的讀取工作轉換成大的讀取工作，並且對齊 zone ，爲了去提供更好的彈性。

由於 Read-ahead 的主要目的是去利用空間區域性的特性，例如連續數筆資料被使用，則和它相鄰的資料在下一段時間內很有可能會被使用。這個可以透過使用計數器和一個門檻值來做到，假如最近到達的讀取工作是接連前面讀取工作的最後一個位址，則把計數器累加，否則重新計數，而當計數到超過門檻值，接下來的工作就一次取回一個 group ，在這裡我們設的門檻值爲 128 sector 。

因爲系統是 multiprogramming ，所以數個工作可能會混合在一起，因此我們要能從中找到連續的部份，在這裡我們設計一個 thread table ，在我們的設計裡，有兩種 thread table : random thread table , continue thread table ，並且每一種 thread table 都有二十個 entry ，這個值應該不會小於同時發生的 thread ，每一個 entry 記錄著工作的最後一個 block 的位址和連續的 block 數，每當有一個工作到達時，即去判斷是否有連續的，假如有，則把連續的部份合併，更新 entry 的最後一個 block 的位址和連續的 block 數，假如沒有，則取代最早加入的 entry 。但是這個方法仍然會有問題，當有連續的隨機的讀取的動作時，這整個 table 則會被清除掉，因此我們加入了一個 sliding window 的機制，這個 sliding window 保持著最近二十個讀取工作的位址，當有讀取工作到達的時候，假如有連續的話，則去替換 continue thread table 的 entry ，若沒有的話，則把這個讀取工作也加入到這個 window 裡，也就是 random thread table 的 entry 裡。

5. Experimental results

5.1 Experimental setup and performance metrics

在接下來的實驗中，我們將會透過 disksim 這個模擬器 [11]，來模擬一系列的實驗，證明用 small parity check 來提升系統平行度是有效的。圖15左邊是我們實驗的系統規格，磁碟的規格我們是採用 disksim 所附的 HP_C3323A 的磁碟參數檔 [14]，在磁碟的容量方面則是以等比例去作放大，並且由於我們所收集的 trace 是從 ftp server 上收集下來的，所以我們必需使用到磁碟陣列來架構我們的系統。圖15右邊是實際的系統架構，並且我們採取下面的這些假設 : 每個工作可以在瞬間就擴充多餘物和重建， 因爲 xor 動作比 I/O 動作要來的快的許多，所以 xor 動作的花費可以被忽略掉，並且所有在離線時所進行的調整動作我們也把它忽略掉。

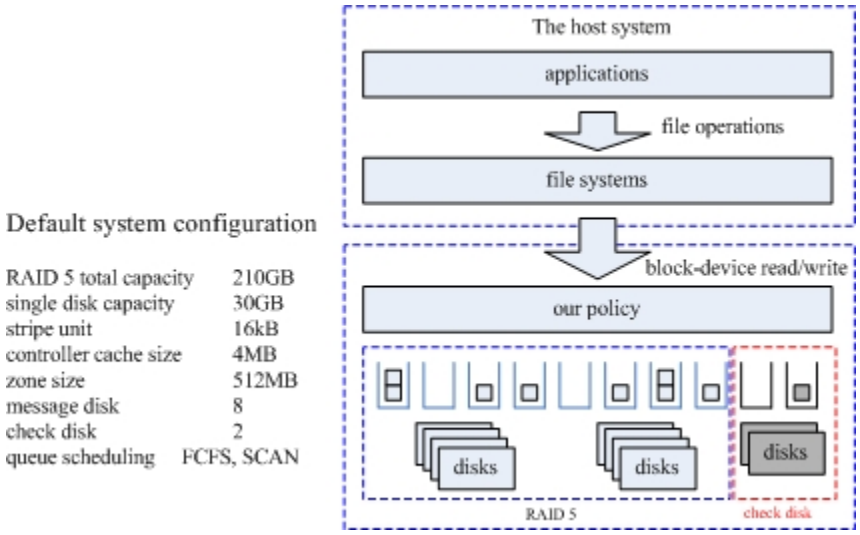


圖 15:The experimental system configuration

實驗的工作是使用 diskmon 去收集的，收集一台桌上型個人電腦所架的FTP server 上的動作的資料， CPU 是 AMD Athlon(tm) 64 Processor 3000+ 2.0G Hz ，記憶體是 2.0GB ， 磁碟的容量是 200GB ，作業系統是使用 windows XP ，檔案系統是使用 NTFS ，桌上型個人電腦是一種常被大眾拿來做一般用途的系統，像是網頁瀏覽器， email 的用戶端，FTP 的用戶端， office 系列，影音播放器，遊戲…等等方面的使用，我們持續收集一段時間的記錄，並且讓這些記錄重新運作在我們的模擬器上面來進行比較，圖16是我們模擬的系統組織圖，ctrl 也就是 controller ，在 controller 上面的 queue 的 scheduling 是使用 FCFS, SCAN ，左邊藍色虛線的方塊裡是原本 RAID 5 的系統，右邊紅色虛線的方塊裡是使用我們的方法後，所增加的 check disk 所放置的區域。

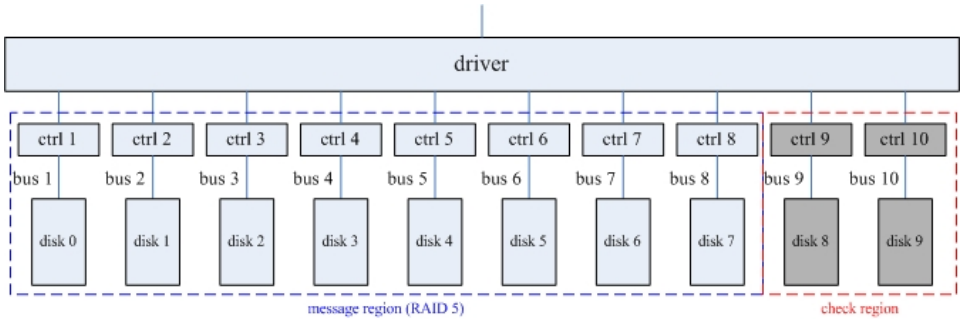


圖 16:Redundancy RAID 5 system configuration

我們稱藍色虛線未使用多餘物的系統爲原本的 RAID 5 儲存系統(original RAID 5 system)，而有使用紅色虛線多餘物的系統爲多餘物的 RAID 5 儲存系統 (redundancy RAID 5 system)，也就是說多餘物的 RAID 5 儲存系統就是有使用我們提出的方法的儲存系統，並且讓 RAID 5 的部份的組織架構都完全相同，唯一不同的是多餘物的 RAID 5 儲存系統比原本的 RAID 5 儲存系統額外增加兩個磁碟來增加多餘物使用，如圖16右邊就是我們實驗時的系統組織圖，灰色的部份就是多餘物

的 RAID5 儲存系統用來增加多餘物的 check disk ，並且針對這兩種系統來進行比較，我們去改變 check disks 的數量， stripe unit 的大小和 zone 的大小來觀察發生什麼變化，因爲我們主要目標是改善回應時間，因此採用下面的評比標準， RT 代表回應時間

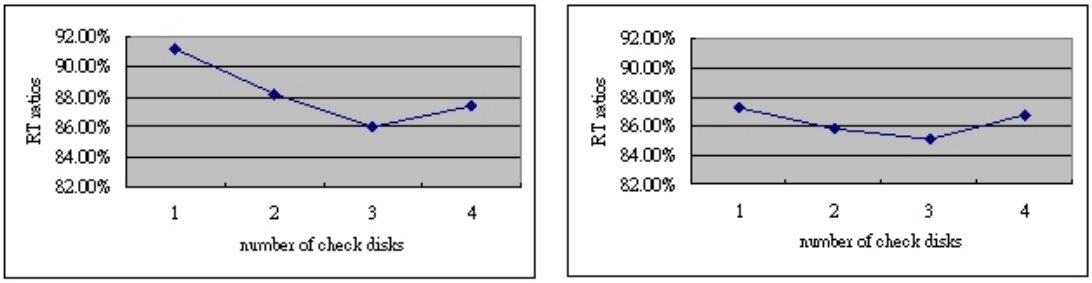
$$\text{RT ratio} = \frac{\text{Average response time in the redundancy RAID 5 system}}{\text{Average response time in the original RAID 5 system}}$$

而回應時間比例愈低代表愈好。

5.2 Numerical Results

5.2.1 Number of check disks

首先我們第一個要考慮的是 check disk 的數量，因爲使用愈多 check disk ，就等於需要愈多的空間花費，而 check disk 數量愈多，能增加多餘物的空間也就愈多，所以這裡針對各種不同數量的 check disk 對提升系統效能的影響來進行比較。



(a) number of check disks(FCFS) (b) number of check disks(SCAN)

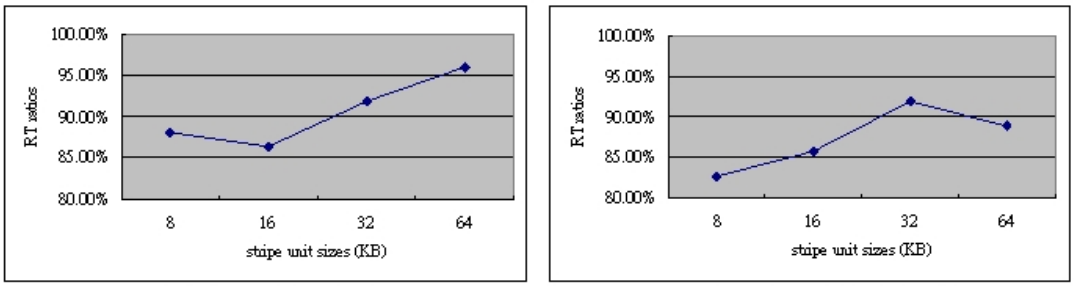
圖 17:Number of check disks

圖17是針對不同數量的 check disk ，使用 scheduling 爲 FCFS 和 SCAN 進行比較所得到的結果，由此圖中可以發現，當 check disk 數量愈多的時候，所能提升的效能愈高，這是因爲 check disk 數量愈多，也就愈多的 check disk 能幫助讀取的工作閃躲，愈有機會能把工作量分擔出去，因此能提升的效能也就愈高。而當 check disk 數量爲 3個的時候形成了一個反折點，主要原因是當 check disk 爲 3個的時候，便已經能充分的把工作分擔到 check disk 上了，而當 check disk 爲 4個的時候，反而因爲太多工作被分擔到 check disk 上了，造成原本可以在同一個 message disk 上連續讀取的工作，變成不連續了，因此造成能提升的效能反而沒有 check disk 爲 3個的時候來的得好。

5.2.2 Stripe unit sizes

接著我們再考慮 stripe unit 的尺寸，因爲當 stripe unit 的尺寸愈大，讀取時所橫跨的磁碟數量也就會相對愈少，系統的平行度相對的也變小，並且一次讀取所取回的 message 的個數變少，所以能選擇的回復路徑也就變少。而當 stripe unit 的尺寸太小的時候，則會造成原本可以在同一個磁碟做連續讀取的動作即可得到的資料，變成要在數個磁碟上做不連續讀取的動作才能取回，而造成不必要的磁碟運作

的時間上的浪費。



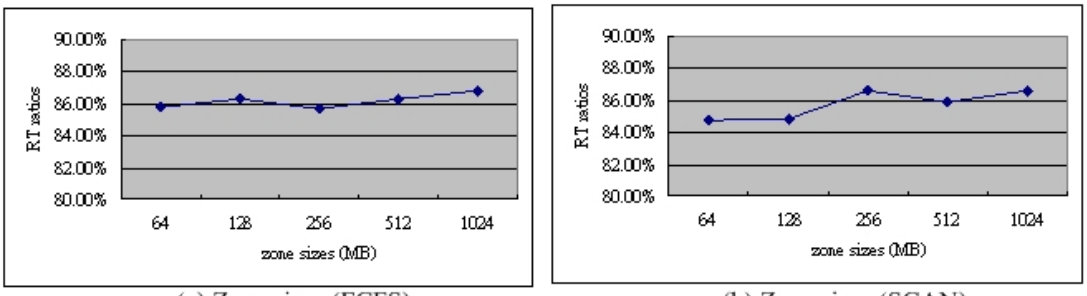
(a) stripe unit sizes(FCFS)　　(b) stripe unit sizes(SCAN)

圖 18:Stripe unit sizes

圖18是對不同的 stripe unit 的尺寸進行比較所得到的結果，由這張圖中可以發現，當 stripe unit 的尺寸愈小的時候，所能提升的效果愈好，這是因爲當 stripe unit 的尺寸愈小的時候，對相同大小的工作而言，能橫跨的磁碟數量愈多，相對的也就是能提供選擇的回復路徑也就愈多，愈能讓系統工作負載愈平衡。在圖18.a 裡，當 stripe unit 的尺寸爲 16KB的時候，形成了一個反折點，主要原因是可能因爲當 stripe unit 的尺寸爲 8KB時，因爲 stripe unit 的尺寸太小，造成原本可以在同一個磁碟上連續讀取的工作變成在數個磁碟上不連續的讀取工作，因此造成效能提升上反而沒有 stripe unit 的尺寸爲 16KB時來得好。在圖18.b裡，則是當 stripe unit 的尺寸爲 32KB的時候，形成了一個反折點，這可能是因爲當 stripe unit 的尺寸爲 64KB 的時候，可以在同一個磁碟上連續的讀取大小比 stripe unit 的尺寸爲 32KB 時來的多，造成 stripe unit 的尺寸爲 64KB的時候得到的效果較好。

5.2.3 Zone sizes

最後我們針對 zone 的大小去進行比較，因爲我們的方法要找到一個接下來最有可能被讀取的區域，爲這塊區域去增加多餘物，利用有限的資源來幫助提升系統的平行度，並且因爲 check disk 的容量不變，所以 zone 的大小愈小能預測的愈準確，愈能充分運用這些資源。



(a) Zone sizes (FCFS)　　(b) Zone sizes (SCAN)

圖 19:Zone sizes

圖19.a是當 scheduling 使用 FCFS 的時候，使用不同的 zone 的大小的比較結果，而圖19.b是當 scheduling 使用 SCAN 的時候，使用不同的 zone 的大小的比較結果。本來應該是 zone 的大小愈小的愈能精確的去把需要增加 check 的部份找

出來，並且爲它們去增加 check ，能提升的效能愈高。但是由於實際的工作量是從一個 FTP server 收集下來的，而在這個 FTP server 上面存放著許多像是影片檔這一類的大檔案，因此若有使用者去進行下載這些檔案，即會使得不管是使用小的 zone size 或是大的 zone size 去進行判斷，所包含的區域都是一樣的，所以造成 zone size 雖然愈小，但所能提升的效能並沒有一定是變的愈多。

6. Conclusion

在我們這一篇論文裡，發現到由於在一般的磁碟陣列上面，那些常常被寫入的區域可能會造成系統效能上的瓶頸，所以希望能透過閃躲掉這些忙碌中的磁碟去把資料取回，來讓工作的回應時間能藉此加快，因此我們想透過把 small parity check 這個編碼技術，拿來運用到 RAID 5 系統上面，藉此來提升系統平行度，並且藉由實驗結果得知透過這樣的編碼方法，只需額外的增加幾個磁碟，便能使得系統的回應時間變的比較好。

在未來的工作方面，由於目前我們只專注在讀取時的 request scheduling ，而未注意到寫入時所產生的額外花費，所以接下來可以進一步研究寫入時所產生的額外花費，並且目前我們所使用的 erasure code 是 small parity check ，我們可以再進一步的去試看看別種的 erasure code ，找到一個可以在編碼方面的花費低，且又能得到許多的回復方法的編碼方式。

reference

1. Brinkmann, A., Salzwedel, K., and Schedideler, C., “Efficient, distributed data placement strategies for storage area networks,” in the ACM Symposium on Parallel Algorithms and Architecture, 119-128, 2000.

2. Khuller, S., Kim, Y.-a., and Wan, Y.-C. J. “Algorithms for data migration with cloning,”in Proceedings of the 22nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, 27-36, 2003.

3. Wilkes, J., Golding, R., Staelin, C. and Sullivan, T. “The HP AutoRAID hierarchical storage system,” in High Performance Mass Storage and Parallel I/O : Technologies and Applications, H. Jin, T. Cortes, and R. Buyya eds. IEEE Computer Society Press and John Wiley, New York, NY, 90-126, 1985.

4. J. S. Plank, A. L. Buchsbaum, R. L. Collins, and M. G. Thomason, “Small Parity-Check Erasure Codes - Exploration and Observations,” in Proceedings of the International Conference on Dependable Systems and Networks, ISBN 978-0780353916, 2005.

5. M. Luby, M. Mitzenmacher, A. Shohrollahi, D. Spielman, and V. Stemann, “Practical Loss-Resilient Codes,” in Proceedings of the 29th ACM Symposium on Theory of Computing, 1997.

6. M. O. Rabin, “Efficient Dispersal of Information for Security, Load Balancing, and Fault Tolerance,” Journal of the ACM, Vol. 36, Issue 2, 1989.

7. A. Bestavros, “IDA-based Redundant Arrays of Inexpensive Disks,” in Proceedings of the 1st International Conference on Parallel and Distributed Information Systems, 1991.

8. M. Blaum, J. Brady, J. Bruck, J. Menon, “EVENODD: An Efficient Scheme for Tolerating Double Disk Failures in RAID Architectures," IEEE Transactions on Computers, vol. 44, no. 2, 1995.

9. W. A. Burkhard, F. Cristian, G. A. Alvarez, “Tolerating Multiple Failures in RAID Architectures with Optimal Storage and Uniform Declustering,” in Proceedings of the 24th Annual International Symposium on Computer Architecture (ISCA), 1997.

10. J. Plank, “A tutorial on reed-solomon coding for fault tolerance in raid-like systems,” Software Practice and Experience, 27(9):995–1012, Sept. 1997.

11. J. S. Bucy and G. R. Ganger, “The disksim simulation environment version 3.0 reference manual,” Technical Report CMU-CS-03-102, Carnegie Mellon University, 2003.

12. LUND, K. AND GOEBEL, V., ”Adaptive disk scheduling in a multimedia DBMS,” In Proceedings of the ACM Multimedia Conference. ACM, New York, 65–74, 2003.

13. Z. Dimitrijevic and R. Rangaswami, “Quality of service support for real-time storage systems,” International IPSI-2003 Conference, October 2003.

14. J. S. Bucy and G. R. Ganger, “HP C3323A,” http://www.pdl.cmu.edu/DiskSim/.

15. M. R. Garey and D. S. Johnson, ``Computers and intractability," 1979