國 立 中 正 大 學

資訊工程學系研究所

碩 士 論 文

可應用於改善固態硬碟磁碟陣列之存取效能的同位元檢查碼及資料的快取管理機制

A Parity Check and Data Cache Management Method to Improve the Performance of a Solid-State Disk-Based RAID

研 究 生 ： 許皓翔

指導教授 ： 鍾菁哲 博士

中華民國 一百零一 年 七 月

國 立 中 正 大 學 碩 士 班 研 究 生

學 位 考 試 同 意 書

本人所指導 資訊工程學系

研究生 許皓翔 所提之論文

(可應用於改善固態硬碟磁碟陣列之存取效能的同位元檢查碼及資料的快取管理機制)A Parity Check and Data Cache Management Method to Improve the Performance of a Solid-State Disk-Based RAID.

同意其提付 碩 士學位論文考試

指導教授 [illegible] 簽章

101 年 6 月 11 日

國立中正大學碩士學位論文考試審定書

資訊工程學系

研究生許皓翔 所提之論文

(可應用於改善固態硬碟磁碟陣列之存取效能的同位元檢查碼及資料的快取管理機制)A Parity Check and Data Cache Management Method to Improve the Performance of a Solid-State Disk-Based RAID.

經本委員會審查，符合碩士學位論文標準。

學位考試委員會
召　集　人 張錫嘉 簽章

委　　　員

林泰吉　　[illegible]

鍾菁哲

指　導　教　授 鍾菁哲 簽章

中華民國 101 年 7 月 2 日

博碩士論文授權書

（本聯請裝訂於論文紙本書名頁前空白處，供學校圖書館做為授權管理用）

ID:100CCU00392063

本授權書所授權之論文為授權人在 國立中正 大學(學院) 資訊工程研究所 系所 ______ 組 100 學年度第 二 學期取得 碩 士學位之論文。

論文題目： 可應用於改善固態硬碟磁碟陣列之存取效能的同位元檢查碼及資料的快取管理機制

指導教授： 鍾菁哲,Ching-Che Chung

茲同意將授權人擁有著作權之上列論文全文(含摘要)，提供讀者基於個人非營利性質之線上檢索、閱覽、下載或列印，此項授權係非專屬、無償授權國家圖書館及本人畢業學校之圖書館，不限地域、時間與次數，以微縮、光碟或數位化方式將上列論文進行重製，並同意公開傳輸數位檔案。

紙本論文：茲同意將授權人擁有著作權之上列論文全文(含摘要)，提供讀者基於個人非營利性質之閱覽或列印，此項授權系非專屬、無償授權國立中正大學圖書館做為編目上架及公開陳列閱覽使用。

☐ 校內外立即開放
☐ 校內立即開放，校外於　　年　　月　　日後開放
☑ 校內於 2017 年 08 月 15 日；校外於 2017 年 08 月 15 日後開放
☐ 其他 ______

授權人：許皓翔

親筆簽名或蓋章： 許皓翔　　　　民國 101 年 8 月 15 日

摘要

在這篇論文中，我們探討了固態硬碟在磁碟陣列中所遇到的問題並且提出新的快取管理機制和適合的同位元檢查碼。固態硬碟主要元件有控制器、平行架構的快閃記憶體和輸出輸入的介面所組成。固態硬碟有出色的特點像是讀取速度快、抵抗震動的能力很好和較低的耗電量，但是固態硬碟也有些缺點像是有限的寫入次數和寫入的最小單位是分頁但是抹除卻是一整個區塊，這些缺點主因是快閃記憶體的特性造成。

由於固態硬碟的缺點是傳統硬碟所沒有的，如果直接把固態硬碟直接用在磁碟陣列架構中，會導致固態硬碟的效能無法發揮且可能造成使用壽命減短。在磁碟陣列中含有同位元檢查碼，同位元檢查碼是由每一筆資料彼此透過互斥運算所得到的值，同位元檢查碼涉及到讀取固態硬碟和寫入固態硬碟的動作，在一般傳統的磁碟陣列架構中，每一筆新的資料寫入就會伴隨新的同位元檢查碼產生和寫入，頻繁的同位元檢查碼寫入會導致磁碟陣列系統效能下降和固態硬碟使用壽命下降。

為了解決上述問題，使得固態硬碟能在磁碟陣列中發揮應有的效能，我們參考了前人的方法並與本論文的快取管理機制做結合，在本論文所提出的快取管理機制可以同時減少產生同位元檢查碼所需的讀取次數和同位元檢查碼寫回固態硬碟中的次數，此外我們整理了近年來相關的論文研究，並且討論先前研究的優點和缺點。

我們自己建立一個磁碟陣列的模擬平台，在這平台上我們實踐先前研究的方法並且和本方法做比較，另外我們也透過可程式化閘陣列(FPGA)平台來做驗證。

Abstract

In this thesis, we discuss the problems of Solid-State Disks (SSDs) with RAID scheme and propose a parity check and data cache management method. The major components of a SSD are a controller, flash chips and I/O interfaces. SSDs have more advantages than HDD, such as, shake resistance, small power consumption and faster performance. However, SSDs have some drawbacks such as limited write times and the unit for write operation is a page, but the unit for erase operation is a block. These drawbacks contribute to flash chip's characteristics.

SSDs also have some problems that never happened in the HDD. It is not suitable to directly dispose SSDs to the RAID-5, or the performance of SSDs and life cycle time of SSDs will be decreased. In the RAID scheme, the parity is generated by every data exclusive-or with the other data. The generation of parity includes read operations and write operations to the SSDs. Whenever there is a new write request in the RAID scheme, the parity must be generated or updated and written it back to SSDs. The frequent parity update results in the poor performance of SSDs and shortens the life time of SSDs.

This thesis combines the earlier methods and the proposed efficient buffer management method with a cache. This proposed efficient buffer management method reduces both of read operations and write operations for generating parities in RAID system. We also summarize previous researches and RAID scheme in recent years. Moreover, we also compare advantages and disadvantages in each RAID scheme.

In addition, we implement a RAID simulator and compare prior RAID architectures with the proposed method. We also verify the proposed scheme in FPGA.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction to solid-state disks in the market

In the personal computer (PC) filed, the Central Processing Unit (CPU) has made a great progress. The core number of a CPU is increased from single core to multi-core and the performance is improved obviously. The improvement of other component in PC such as, DRAM and video card are also obvious besides the hard disk drive (HDD). The HDD is composed of platters and needs a head actuator mechanism. Because the physical limitations, the HDD improvement is not evident. In the recent years, the solid-state disks (SSDs) becomes more popular in the PC market [1] due to the price of the flash chips are acceptable.

Now SSDs have already become the storage system in the ultrabook [2] which is a thin and light weight notebook, we can find that lots of famous companies such as HP, Dell, and Asus adopt SSDs in theirs products. With improvement of manufacturing process, the high density and large capacity flash chips are developed so that the size of a SSD become smaller as shown Fig. 1.1.

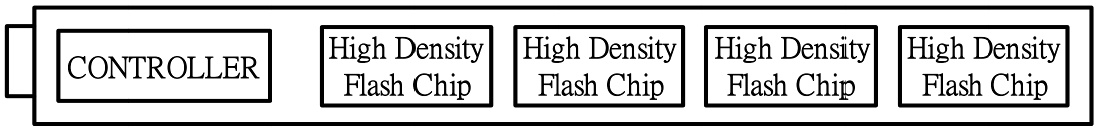


Fig. 1.1 The extreme small SSDs with high density NAND flash chips.

1.2 Introduction to solid-state disk in the datacenter

Some companies such as Hitachi and Samsung propose solutions of SSDs for the datacenter [3], [4]. The Hitachi [3] constructs the datacenter by Serial-Attached SCSI (SAS) interface and Fibre Channel (FC) with SSDs. The [4] analyses the market of SSDs in the datacenter in the future and indicates the challenges. The [4] mentions that challenges are write reliability and price of SSDs. The power consumption is a big issue for the datacenter [5]. The [5] explains and analyses that SSDs save the energy and enhance the performance.

In the big datacenter as shown in Fig. 1.2, they use traditional HDDs with redundant array of independent disks (RAID) as the storage system due to the cheaper price. However, the power consumption and heat dissipation are critical problems. To cool down the datacenter, it requires to spend 40 percent of the whole power consumption [6]. SSDs cannot replace the HDDs in the datacenter even though SSDs have lower power consumptions and have lower temperature. Because SSDs price are too expensive as compared to HDDs with the same capacity. Therefore, there only a few companies develop the RAID controller for SSDs. In the future, when the flash chips price decrease substantially, the datacenter can use the SSDs as the storage system. The SSDs not only reduce the costs of power consumptions but also increase the speed performance. In addition, the datacenter always uses RAID technique to enhance performance and ensure the data integrity.

Fig. 1.2 The picture of a datacenter

1.3 Introduction to components of solid state disk

SSDs are made of many parallel flash chips and don't have the head actuator mechanism. All components of SSDs are all electronic, so SSDs have more advantages, for example, shake resistance, lower power consumption and faster performance. However, SSDs also have some issues that never happened on the HDD, such as limited write times and write a page but erase a whole block.

The controller and flash chips are SSDs major components as shown in Fig. 1.3. The controller is responsible to handle the data access and parity generation, wear-leveling, garbage collection, and Error Checking and Correcting (ECC). Fig. 1.4 is the teardown of a real product SSD.

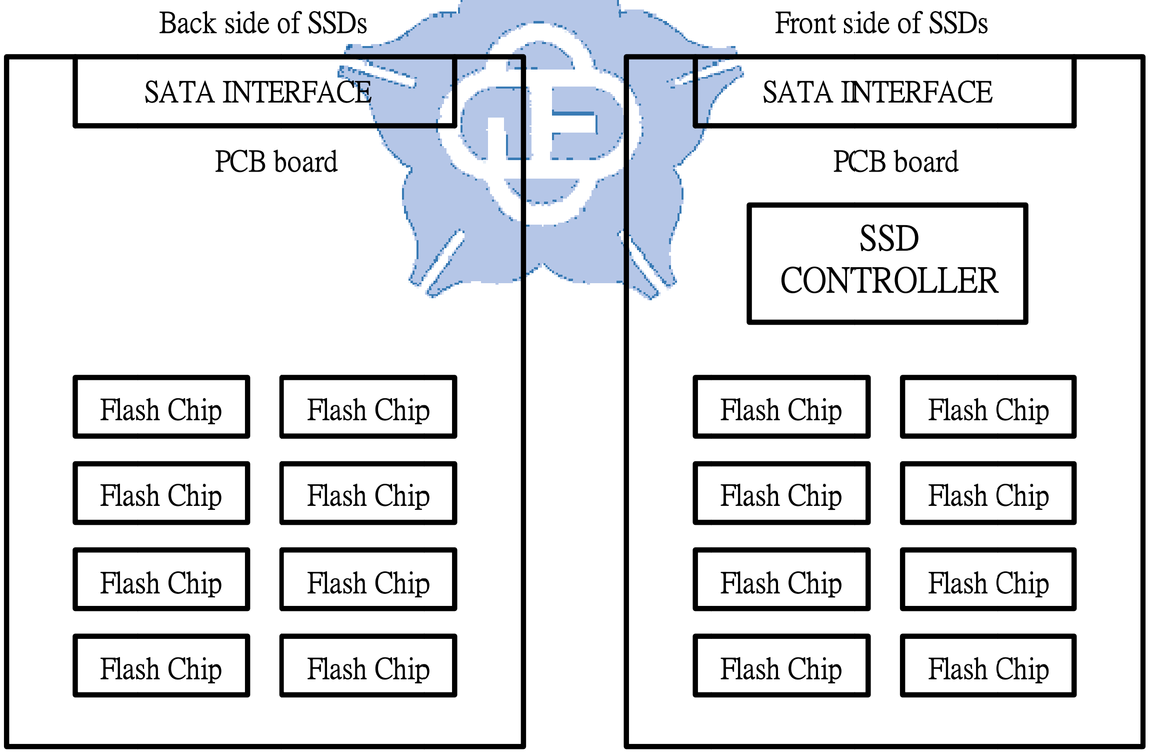


Fig. 1.3 The front side and back side of SSDs.

1.4 Thesis overview

The rest of the thesis is organized as follows: Chapter 2 discusses the characteristic of SSDs, flash memory, RAID architecture and related works. Chapter 3 describes a parity check and data cache management method and operations. Chapter 4 discusses the experimental results with a RAID-5 simulator. Chapter 5 shows the proposed design verified by FGPA. Chapter 6 gives conclusions and future work for this thesis.

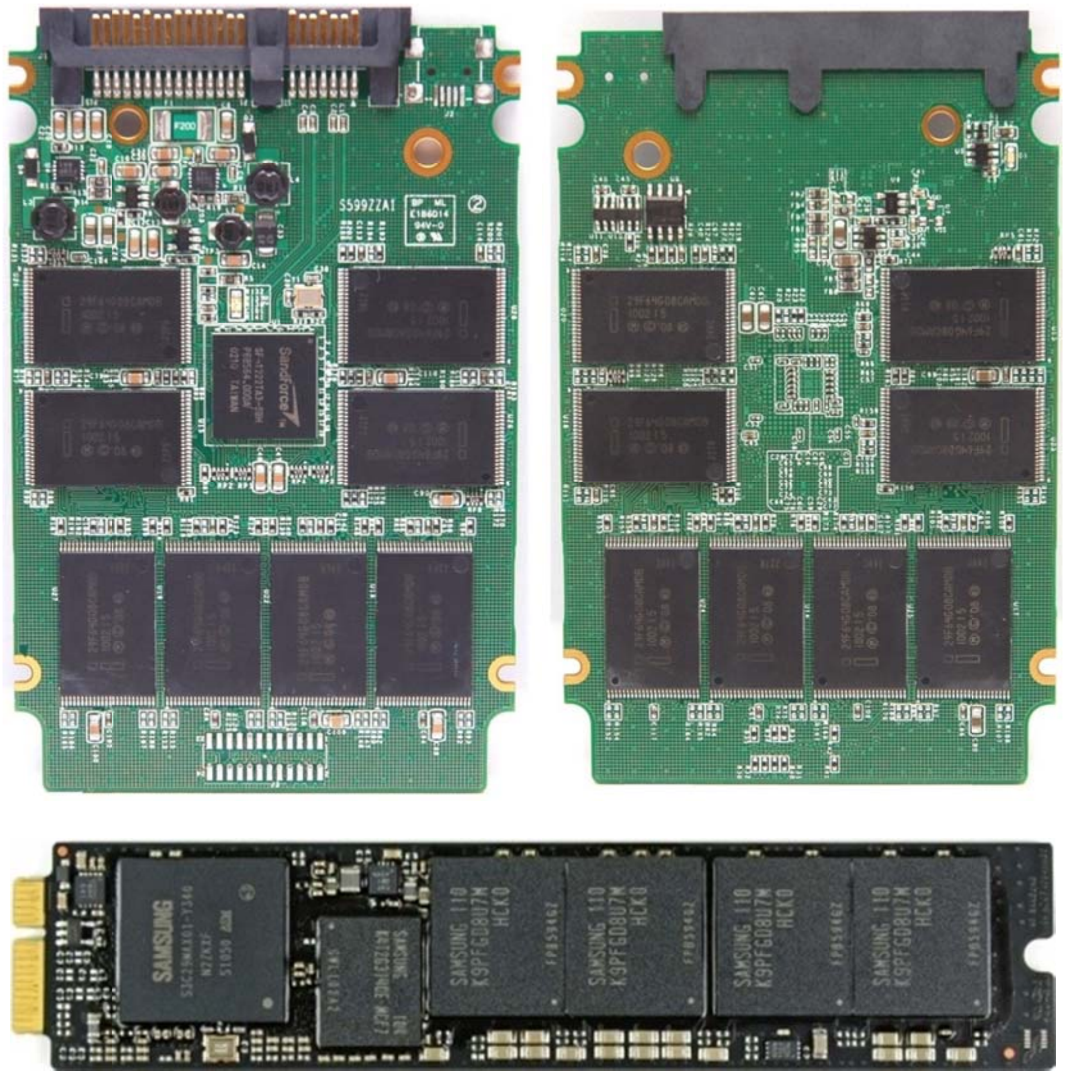


Fig. 1.4 The teardown of real products of SSDs [29], [30].

Chapter 2

Background & Related works

2.1 Background

2.1.1 Background of flash

The flash chips are divided into NOR type and NAND type. The NAND type flash chips characteristic are very suitable for SSDs [7], because the price of NAND type is cheaper than NOR type flash. There are many kinds of NAND type flash chips, such as Single- Level Cell (SLC), Muti-Level Cell (MLC), and Triple-Level Cell (TLC) etc. Their difference is how many bits can be stored in a cell. However, the performance of both read and write operations in the MLC are worse than SLC [8], but the MLC has higher density and cheaper price than SLC [8]. So many SSDs adopt MLC NAND flash chips to store data.

The flash memory has characteristics as follows:

- Small write and large erase

  The page is a unit for write operation but a block is a unit for erase operation.

- Write once

  When we write data to the address which has been written before, this address must be erased firstly.

- Limited erase times

  Each block of the flash have limited erase times.

For SSDs, the flash memory is basic components to store data. Now in the market, the mobile devices like smartphone and tablet computer all adopt the flash memory as the storage system rather than traditional HDD. The flash memory has some unfavorable restrictions such as limited write times and erase before write. The minimum unit of write operation is a page, but when there is no free page, the flash controller must erase a block to recycle free pages. Before the controller erases a block, the controller must copy the valid data in the block to another free block, this operation is called garbage collection (GC).

Table 2.1 shows that block erase time is ten times longer than page program time. From above terrible restrictions, we can figure out that the total cycles of write operation, erase block operation and garbage collection are very terrible. As a result, how to decrease the number of write operations is very important. When the write operations are decreased, the life time of flash memory is extended and the performance is improved. The same situations in SSDs, we can add cache and efficient buffer management method to decrease the write operations and erase operations.

Table 2.1 The simple specifications of NAND flash memories [9].

Hynix 32Gb NAND FLASH	
Data integrity	100,000 Program/Erase cycles
Page read	25 us = 0.025 ms
Page program time	200 us = 0.2 ms
Block erase time	2 ms

2.1.2 **Background of SSDs**

To overcome MLC flash chips worse performance as compared to SLC, the controller plays an important role. The SSDs controller faces the upper operating system layer and lower flash chips layer. For the upper layer, the SSDs controller must handle read or write requests from the operating system. Besides, the SSDs controller must process the lower flash chips layer such as wear-leveling, parity handler, and Flash Translation Layer (FTL) simultaneously. The SSDs controller often adopts the ARM processor and maybe a dual-core processor [10] so that it can handle the heavy loading.

Fig. 2.1 shows the block diagram of the SSD controller. This controller supports SLC type and MLC type flash chips. The RAISE controller shown in Fig. 2.1 is a RAID-5 architecture.

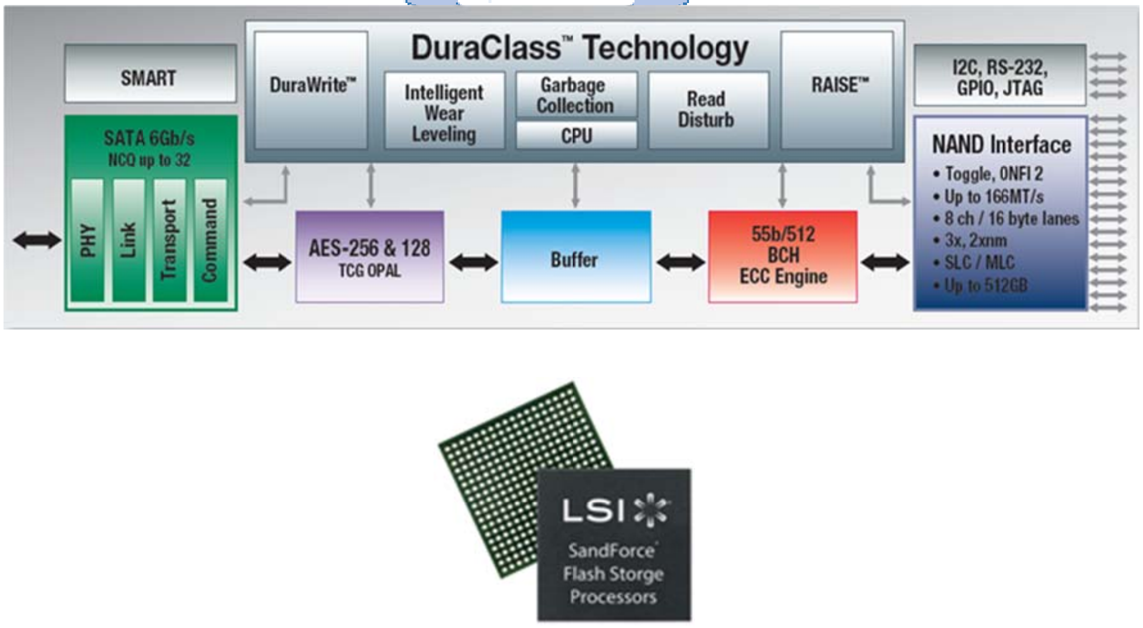


Fig. 2.1 The SSD controller of SandForce [11]

SSDs have exceptional characteristics such that sequential write speed is faster than small random write. The speed gap difference between sequential write and small random write is very huge [12], [13] as shown in Fig 2.2. Besides, frequently small random write will attrit limited write times so that SSDs will be damaged very soon [12]. The operating system or user softwares often have large amounts of small random write. As a result, small random write can dominate the SSDs performance. We can add hardware buffers and provide a software-based buffer management to extend the life cycle time of SSDs. A SSD is consisted of parallel flash chips in a RAID-0 configuration. If we don't have schemes to maintain data integrity, if one flash chip failures, the whole SSDs will crash.

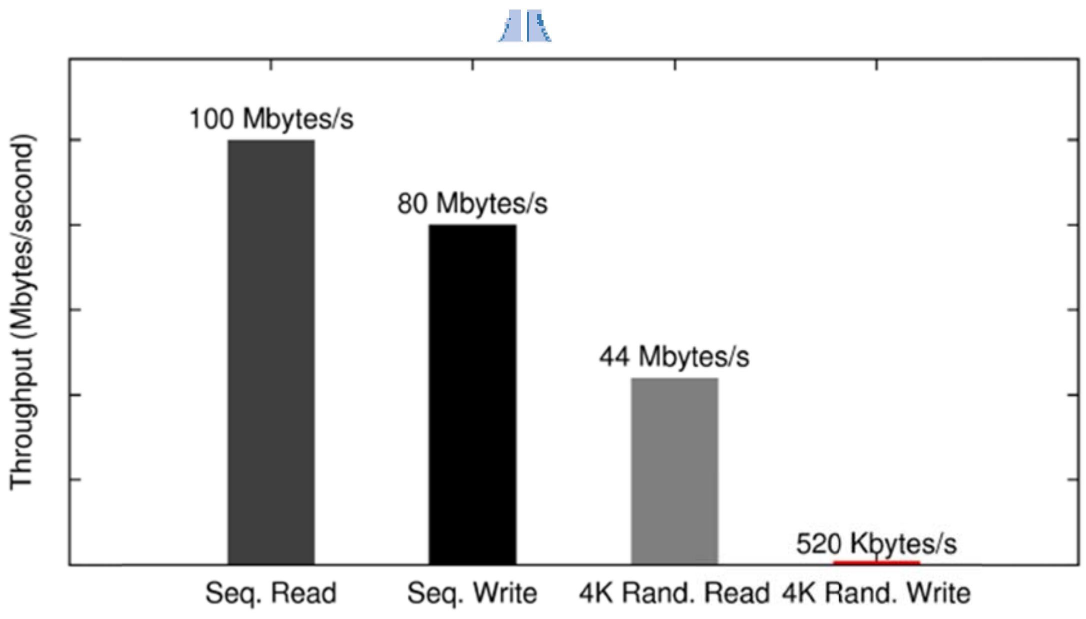


Fig. 2.2 The performance between sequential write and rand write [12]

2.1.3 **Background of RAID**

The RAID technique is commonly used in workstation, datacenter, and SSDs. RAID technique not only raises the performance by parallel data access scheme but also makes sure the data integrity by adding parity data. Fig. 2.3 shows the RAID-0 and RAID-1 architecture.

RAID-0 uses a block-level stripping, and data are written to different storage devices at the same time. The RAID-0 performance is very fast, and it can make use of full capacity. However, the reliability is awful due to that no redundant data to recover the whole storage system when one disk is crashed. RAID-1 copies the data to two different devices in the same time. If we use two storage devices, the available capacity is only 50 percent of the total capacity. The cost of RAID-1 is too expensive.

RAID-2 uses a bit-level stripping and error collection by hamming code. If one bit is wrong, that bit can be recovered. However, the hardware logic scheme of error collection is complicated.

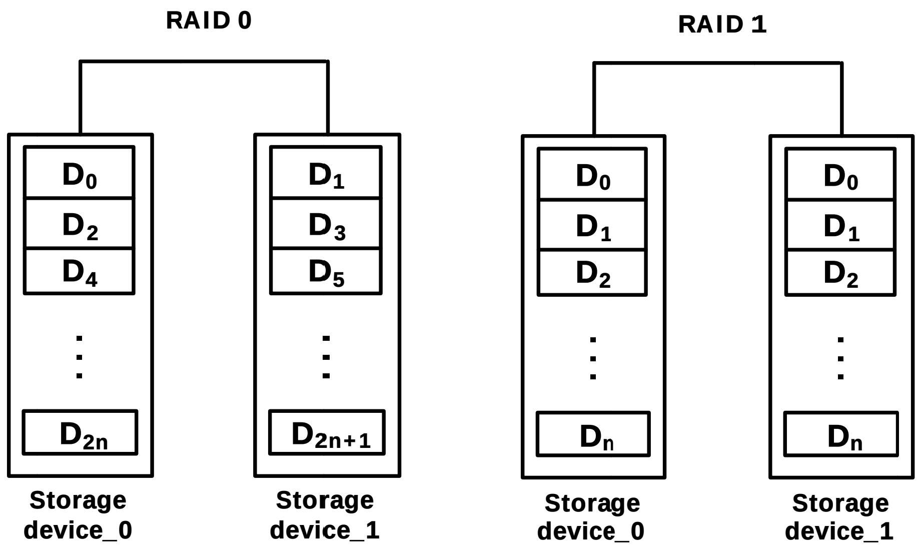


Fig. 2.3 The RAID 0 and RAID 1 architecture

RAID-3, RAID-4, and RAID-5 all use extra storage devices to store parity so that they can tolerate one storage device failed. RAID-3 uses a byte-level stripping and parity data. Both RAID-4 and RAID-5 use block-level stripping and parity data. The minimum number of the storage devices is three. Among these three storage devices, two of them are responsible for storing data and another one is for storing parity data in RAID-3 and RAID-4.

The RAID-3 and RAID-4 dispose the parity to the fixed storage devices. Dispose the parity to the fixed storage device causes the parity storage device becomes very busy all the time. Because whenever data are updated or written to the storage device, the parity must be calculated and updated. That means the storage device which stored the parity has a lot of write operations as shown in Fig. 2.4. RAID-5 disposes the parity to every storage devices so that parity write operations are separated into each storage devices, as shown in Fig. 2.5.

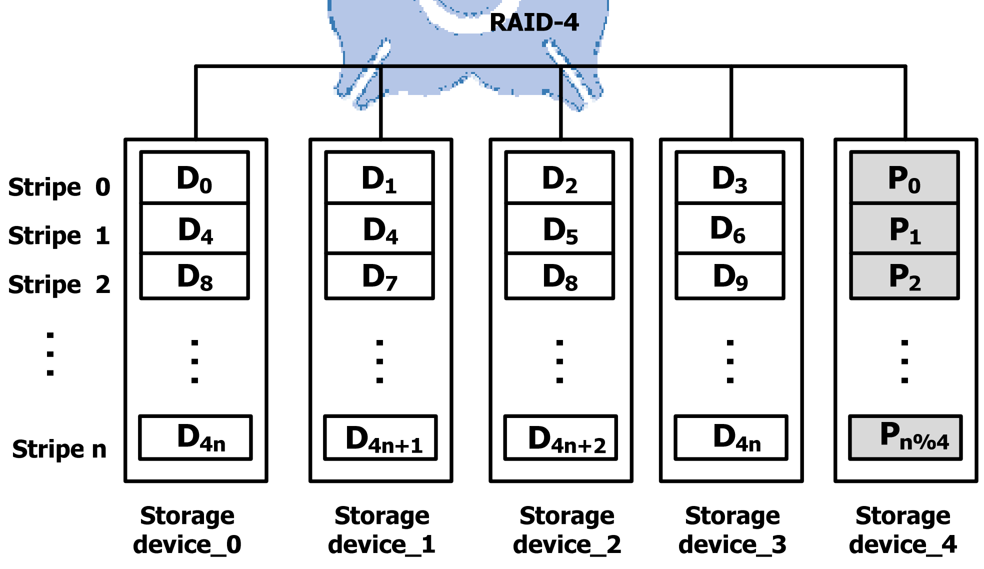


Fig. 2.4 The RAID-4 architecture

This thesis uses the RAID-5 architecture. However, RAID-5 has a critical drawback which is frequently parity writes. In Fig. 2.5, the parity P_0 is generated by $D_0 \oplus D_1 \oplus D_2 \oplus D_3$. The $\oplus$ represents the exclusive-or operator. When we update a data such as D_1, the parity P_0 must be updated. Therefore no matter how large or small amount of data are updated, the parity must be generated again and written to the storage device. The cost of parity generation includes read operations which read data from the storage device and a write operation.

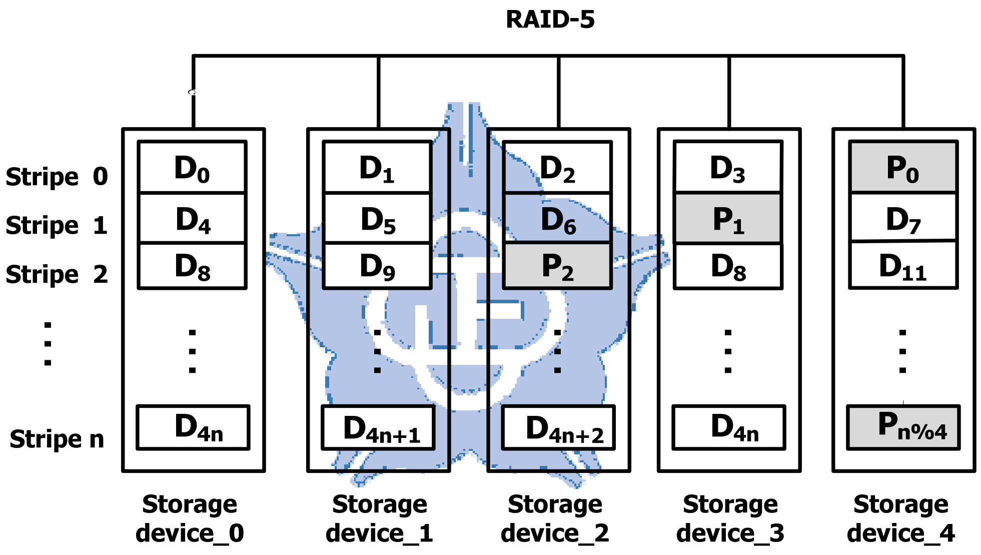


Fig. 2.5 The RAID-5 architecture

RAID-6 enhances the reliability from RAID-5. RAID-6 also uses block-level stripping and two parities across each storage devices as shown in Fig. 2.6. RAID-6 needs one more storage device to store more parity data, so the minimum number of storage devices is four. When RAID-6 updates or writes a data, two parities must be calculated and updated. As a result, the management of parity is more complex and spends more time.

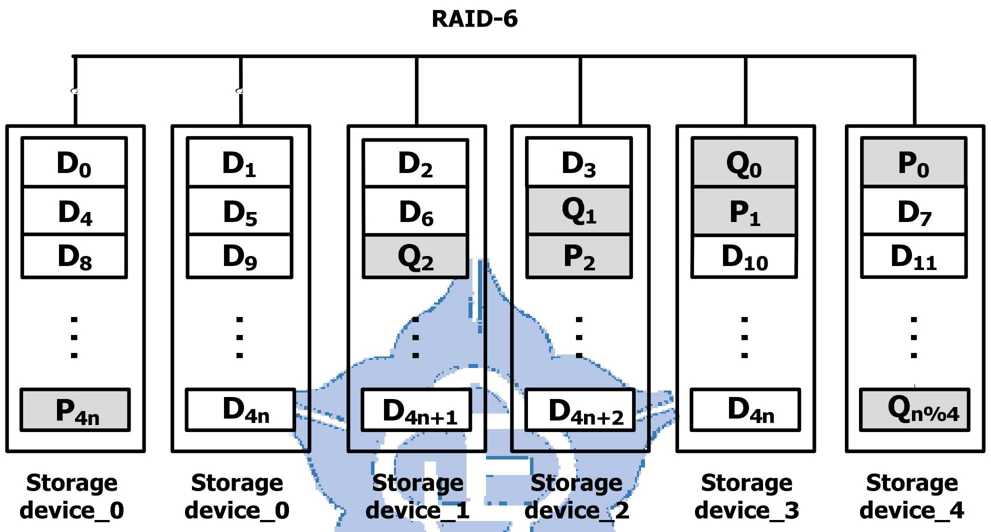


Fig. 2.6 The RAID-6 architecture

RAID 10 also named RAID 1+0, the lower layer is RAID-1 and the upper layer is RAID-0 as shown in Fig 2.7. Oppositely, the lower layer is RAID-0 and the upper is RAID 1 with RAID 01. Both RAID 01 and RAID 10 need at least four storage devices. Both of theirs capacity utilization is 50 percent as shown in Fig. 2.7, so the costs are also very expensive.

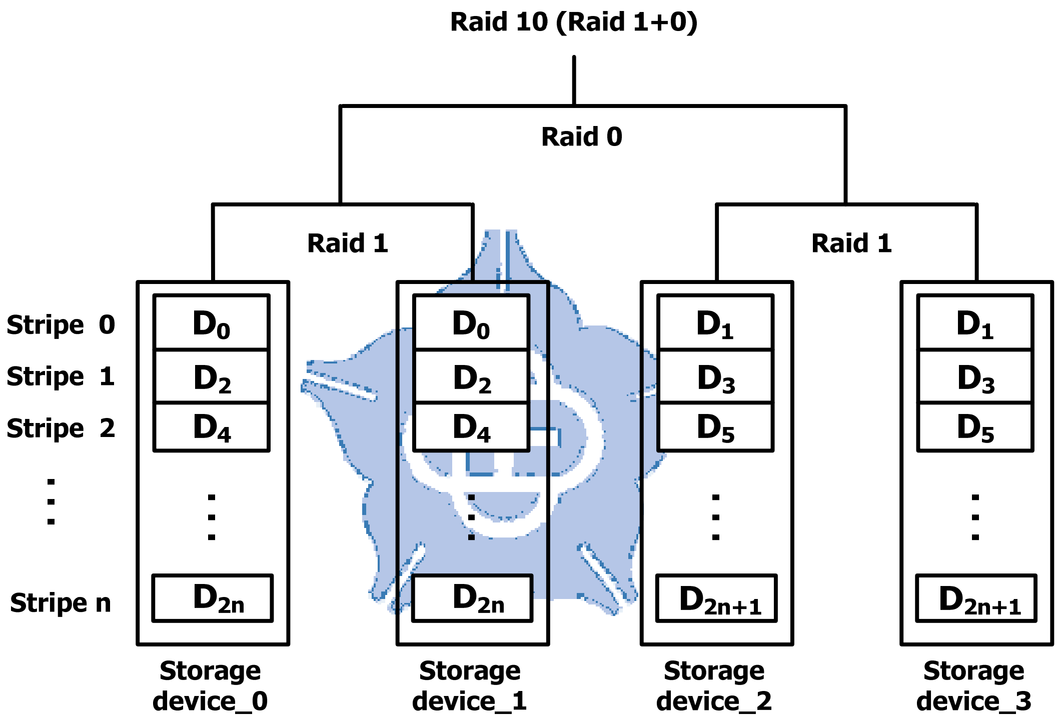


Fig. 2.7 RAID 10 schemes has 50 percent utilization of storage device.

The storage capacity utilization and performance in RAID-5 are acceptable. We adopt the RAID-5 architecture due to above considerations. This thesis represents RAID-5 as n+1, n represents the number of data storage devices and 1 is for parity store. For example, Fig. 2.5 is a 4+1 RAID-5 architecture.

2.2 The problem of SSDs-based RAID architecture

There are products with RAID-0 SSDs in the market. The speed performance of RAID-0 is very fast with no doubt, but the RAID-0 does not have the reliability. When one of the RAID-0 SSDs is damaged, the entire storage system also crashes. For the reliability consideration, the RAID-4 and RAID-5 are more suitable solutions, they have the parity to recover data when one of the storage devices damages.

When we use RAID-4 or RAID-5 with SSDs, we must considerate the flash chip features as shown in Table 2.1. Table 2.1 shows that the flash program speed is very slow. The RAID-4 and RAID-5 has a critical issue that a large number of parity write operations. That issue encumbers the entire storage system speed.

As long as RAID-4 and RAID-5 write new data, the parity must be updated with read operations to generate the new parity. The new parity will write to the storage system. The storage system must decrease the parity generation overheads and parity write times. Hence, we need an efficient SSDs parity cache management scheme to increase the speed of performance with reliability.

2.3 The purpose of the thesis

This thesis combines the earlier methods and the proposed an efficient buffer management method with a new cache. We implement a RAID-5 simulator, the simulation results show that both read and write operations are decreased. The earlier method is partial parity cache (PPC) architecture [14], we also use the PPC which represents the parity of many data.

Therefore, we can reduce the chance of parity write by merging the parity data. Besides, we add a special data buffer which keeps the old data. This special data buffer is for partial parity generation so that we don’t need to read data from storage system during parity update.

An efficient buffer management method can reduce the times of parity write and the proposed data cache can reduce the times of read operations during updating the parity data. Simulations results show that the write performance is improved by 76 percent as compared to traditional RAID-5 architecture and is improved by 19 percent as compared to previous partial parity architecture [14].

Section 2.4 and section 2.5 discuss the advantages and disadvantages of the previous research.

2.4 SSDs RAID architecture with redundant parity

When we directly dispose the SSDs to the RAID architecture, there will be lots of problems [15] - [17]. The previous research [15] proposed the heterogeneous SSD-based RAID-4, the wear leveling scheme for the SSD-based RAID-5, and the efficiency of erasure codes schemes for SSD-based RAID-6.

For the RAID-4, [15] finds that parity disks write times are three times larger than the average write times of the other disks. The experimental results show that the write operations concentrate on the parity disk, so their solution uses a HDD to replace the SSD parity disk as shown in Fig. 2.8. We can figure out that the SSD-based RAID-4 is not suitable, because speed of SSDs write is too slow and SSDs has limited write times.

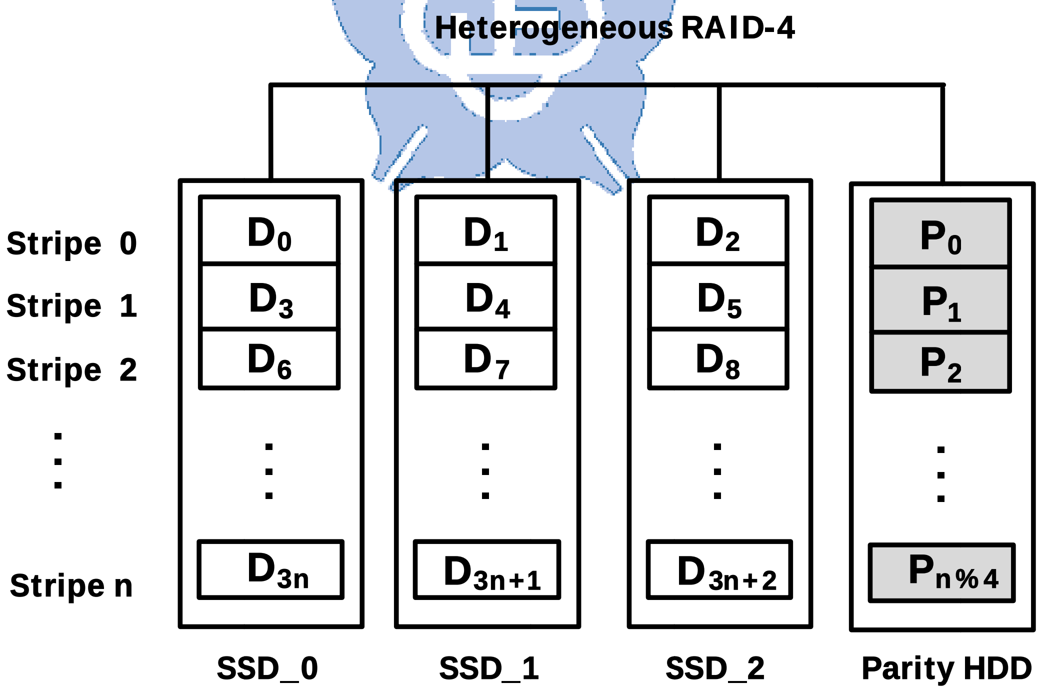


Fig. 2.8 The heterogeneous RAID-4 architecture

For the RAID-5, [15] proposes wear leveling scheme which places the parity data dynamically and creates a k-bit map table for recording the number of parity write times as shown in Fig. 2.9. When one disk's write time is larger than the specific value, the wear leveling scheme will exchange that parity data with other parity data which has lower write times. The wear leveling scheme only balance the SSDs write times, the root cause of the frequently parity update problem that is not solved.

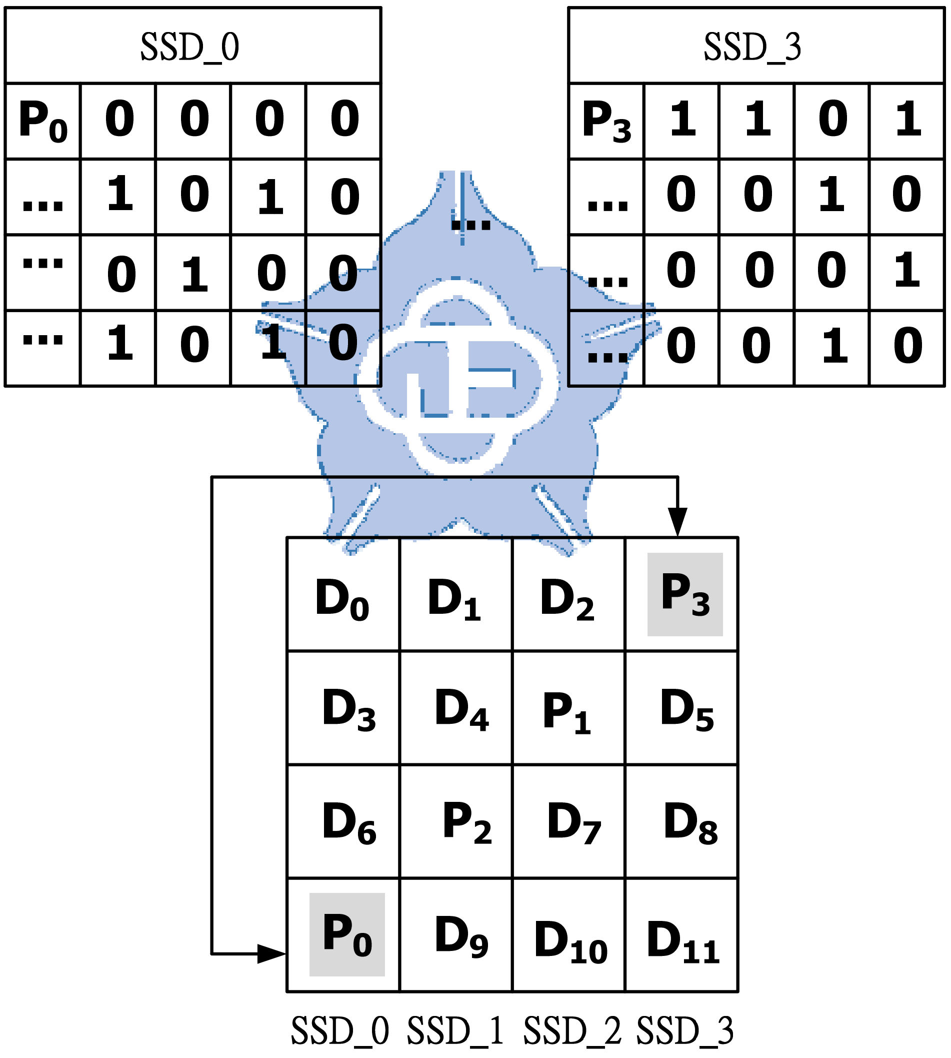


Fig. 2.9 The bitmap table of RAID-5 architecture

For the RAID-6, [15] uses Reed-Solomon (RS) codes and EVENODD codes to analysis which codes are suitable for SSDs. The experimental results show that RS codes has better reliability, but the parity data calculations for RS codes are complex and have a critical drawback in large amounts of write operations. By contrast, EVENODD codes is more simple, because it only needs the exclusive-or to encode the parity and has small amount of write operations. We can learn that concerns of the SSDs-based RAID architecture are different from the HDD-based RAID.

To solve the frequently parity data write problem in RAID architecture, the prior research add a parity buffer to reduce parity data write times. This way is called the delay parity update scheme. The parity data is kept in the parity buffer until specific conditions met. The delay parity update scheme can definitely reduce the parity write times.

Both [18] and FRA [19] use the delay parity update scheme. They reduce the parity data write times but the parity generation overhead is heavy. They must re-calculate the parity data as long as the write requests coming. Table 2.2 is a comparison table of prior approach.

The Partial Parity Cache PPC [14] also adopts the delay update scheme and the partial parity to encode the parity. The PPC [14] generates a partial parity and stores it into the cache. When the cache is full, the partial parity must be rebuilt to the full parity, and written to the SSD.

The experimental results indicate that read operations for generating a parity are decreased obviously. However, the parity write times are almost the same with other approaches. When PPC updates data which already exists in the storage device, PPC scheme often read old data and old parity to update parity.

Table 2.2 The comparison of prior research.

	PPC [14]	[18]	FRA [19]
Raid architecture	Raid-5	Raid-4	Raid-5
Encode parity scheme	Partial parity	MDS	General parity
Parity cache type	Nvram	Nvram	RAM
Flash mapping method	Page-level	Page-level	Dual-mapping table

2.5 Both of HDD and SSD RAID architecture

Some researches use SSDs and HDDs to construct the storage system [20] - [22]. Fig. 2.10 is the Hybrid Parity-based Disk Array (HPDA) architecture. The HPDA [20] use SSDs to store data and two HDDs to store parity and to be a write buffer as shown in Fig. 2.10. The SSDs and a HDD are constructed by RAID-4. The remainder space of the parity disk HDD and another HDD are constructed by RAID-1 as a write buffer. If write requests are sequential, these requests are distributed to the RAID-4. Oppositely, if the requests are random access, the requests are written to the write buffer. When the I/O is idle, the requests which are in the write buffer are written back to the RAID-4.

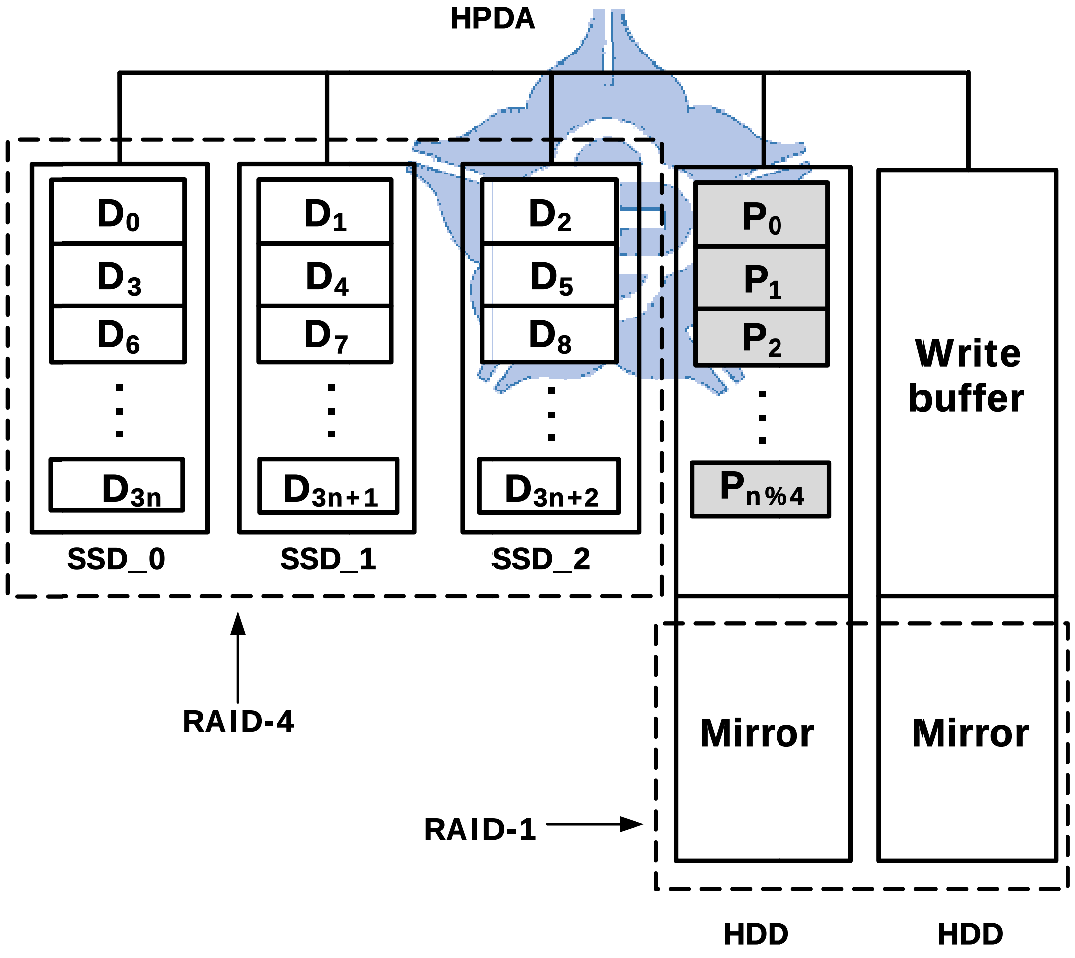


Fig. 2.10 The HPDA architecture

The I-CASH [21] uses the advantages of SSDS to build the hybrid storage architecture as shown in Fig. 2.11. The read speed of SSDs is very fast, so the I-CASH usually reads by SSDs and writes data by HDDs. The SSDs store the data's reference information. The HDD stores the data which combines reference information. When I-CASH writes data, the I-CASH must read reference information from the SSDs and encode it with data to the HDDs. When I-CASH read data, the I-CASH must decode the data from the HDDs with references information from SSDs.

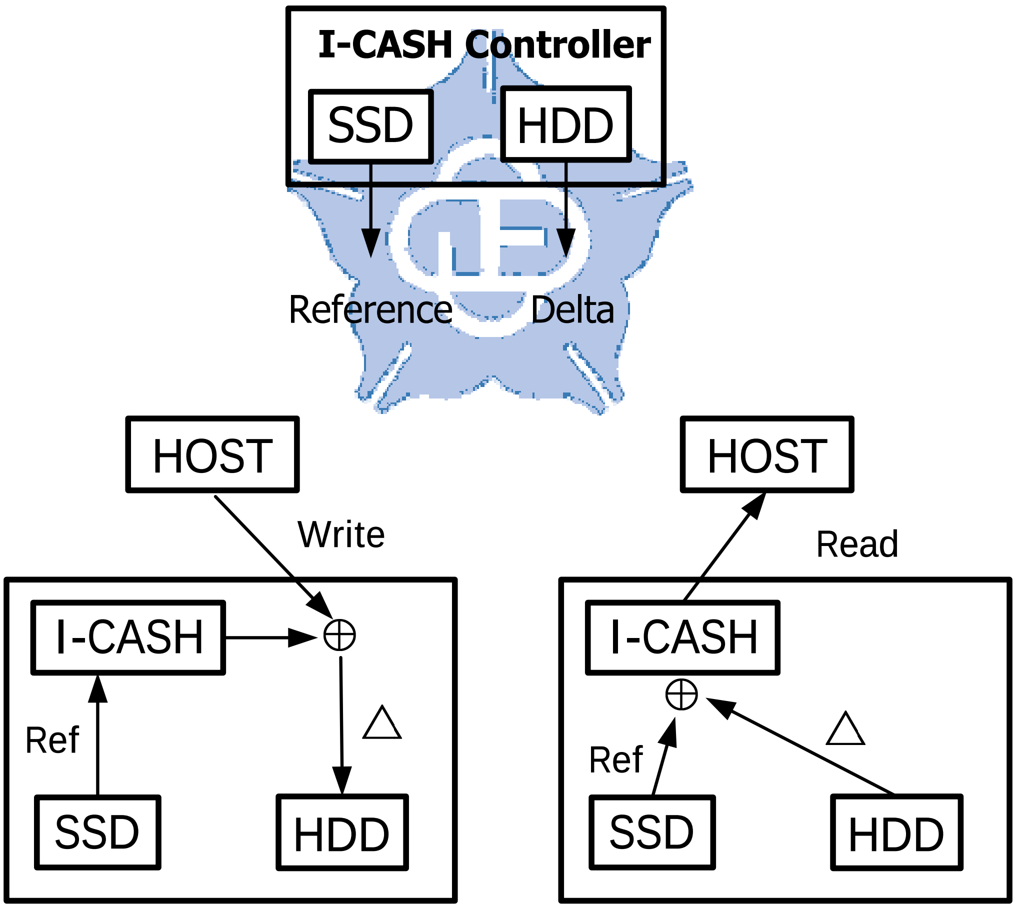


Fig. 2.11 The I-CASH architecture.

The RAF [22] also uses fast read speed characteristic of SSDs. The RAF uses one SSD as a cache and four HDDs in the storage system as shown in Fig. 2.12. The RAF divides the SSD into a write cache and a read cache. Both of read and write cache only store random data, this is because that random access speed in the SSD is faster than the traditional HDD. The sequential data are handled by HDD, since the sequential read or write speed of HDDs are close to SSDs. Whatever read or write operations, the random data is cached by a SSD and the sequential data is performed by HDDs.

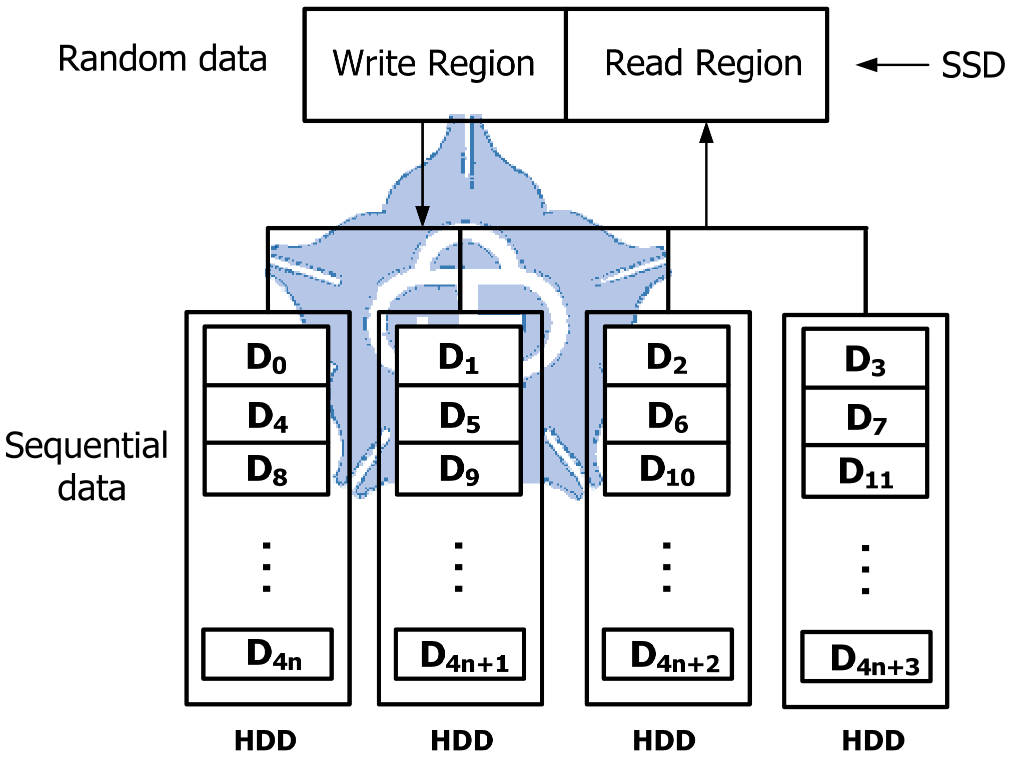


Fig. 2.12 The RAF architecture.

Chapter 3

Efficient buffer method with data buffer scheme

The overall system architecture is shown in Fig. 3.1. The operation system layer sends the read or write requests to the storage interface. The RAID controller handles buffer and distributes data to the each SSDs. The RAID controller also manages the buffer and cache. When the buffer or cache is full, the RAID controller must handle this situation. We cannot reduce the read or write requests from the operation system layer, but we can hold the data in the buffer or the cache to reduce the actual read/write times to the SSDs.

The “partial” parity [14] can represent the data stripe even the partial parity contains only a part of parity. For example, there are P_0, P_1, P_2~P_{16} partial parities in the cache. These partial parities can represent the stripes S_0~S_{16}. Each stripe has four data in 4+1 RAID-5 architecture, for example the data stripe contains S_0 can accept D_0~D_4. It means that this partial parity cache can merge the D_0~D_{64} data. In [14], there are 16 partial parities in the cache, it can merge 64 data.

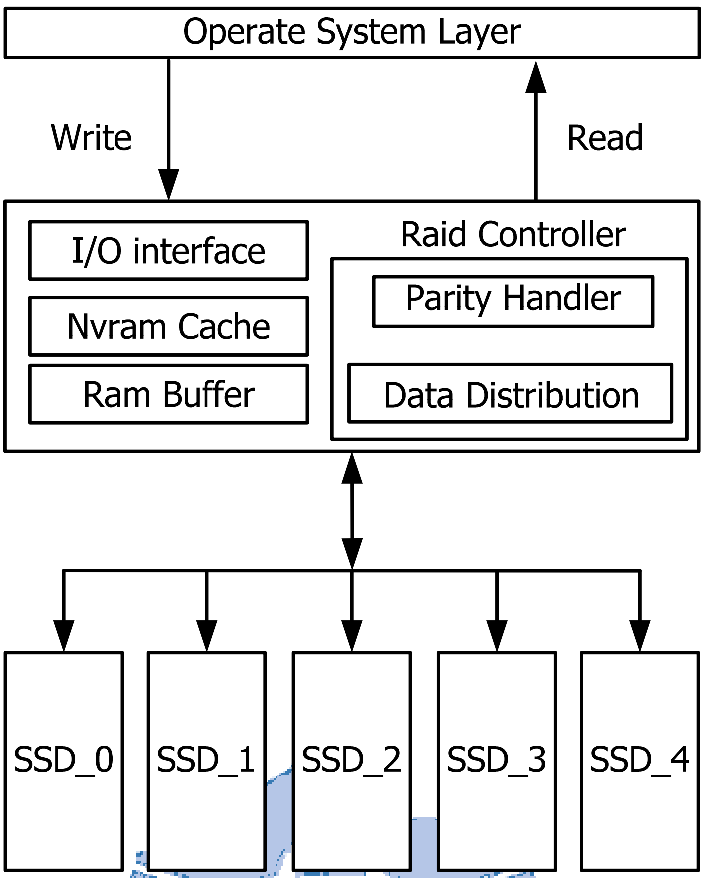


Fig. 3.1 The overall of system architecture

We use this characteristic to select the data which existing in the partial parity cache. This method is called an efficient buffer method. The efficient buffer method tries to avoid that the partial parity cache being full. When the partial parity cache is full, the partial parity must be rebuilt to the full parity. To build the full parity, the RAID controller must read data from storage devices and write back the full parity to the storage device. Therefore the proposed efficient buffer method can greatly reduce the cost in the RAID system.

3.1 Comparison with PPC [14]

We introduce the PPC [14] flowchart in Fig. 3.5 and it shows PPC operations from the write cache to the PPC. The host transmits the data to the storage devices. The RAID controller determines whether the write cache is free or not. If yes, the data are stored in the write cache, and for the host system the data transmission is complete. If not, the controller must select the victim stripe and writes back them to the storage device.

The selection rule of the victim stripe is to find out which stripe in the write cache has most data. When the stripe contains more data, these data can be written to the SSDs in parallel. These victim data in the selected stripe are also used to generate the partial parity.

After generating the partial parity, the controller checks whether the PPC is free or not. If yes, the partial parity will be written to the PPC and the data in the selected stripe will be written to the storage devices. When data are removed from the write cache, the write cache will have free space. If the PPC is not free, the RAID controller will select the victim partial parity by Least Recently Used (LRU) algorithm. After selecting the victim parity, the controller must rebuild the full parity. This operation may read the data which is not in the partial parity ingredient. When the partial parity is removed from PPC, the PPC will have free space.

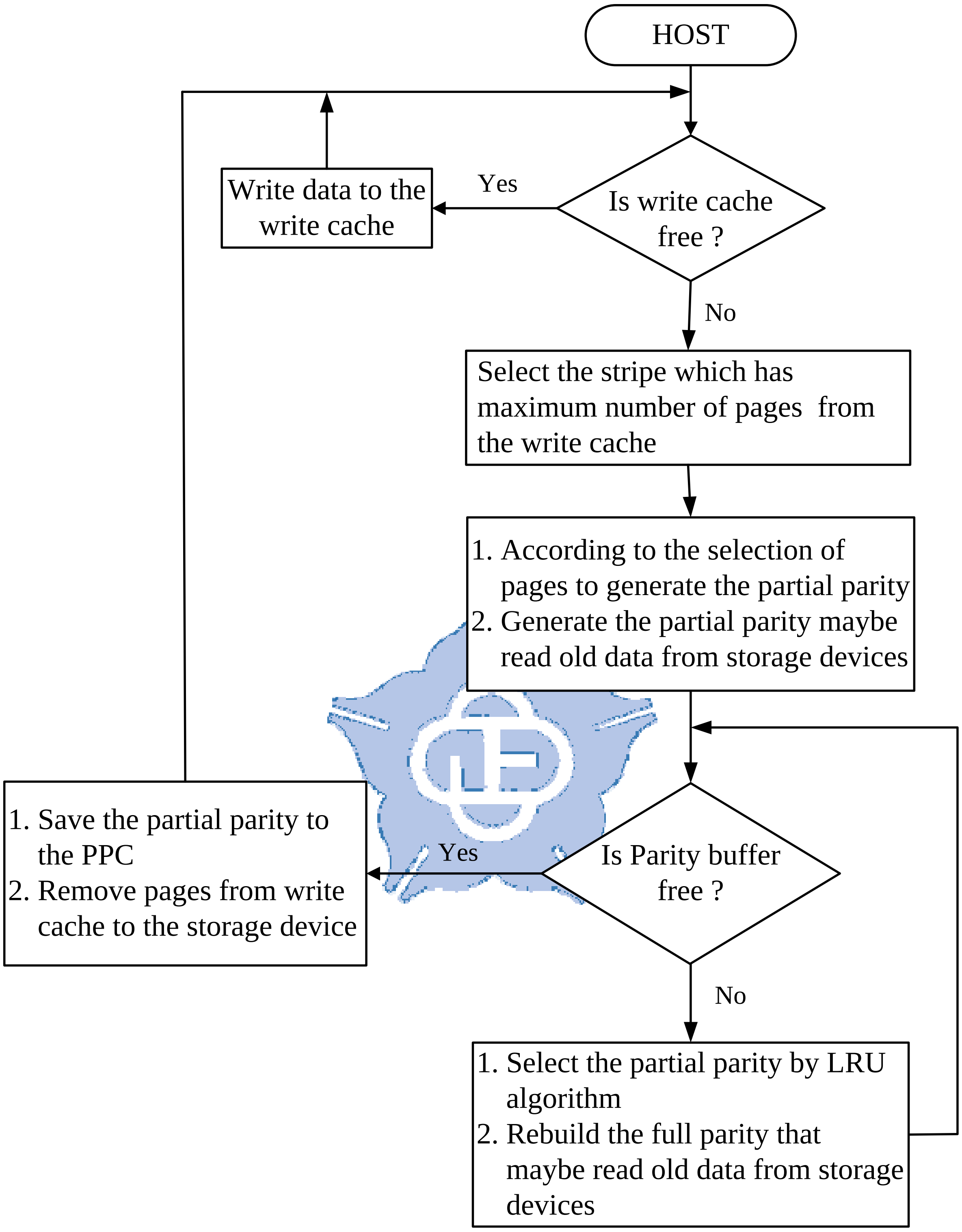

HOST
Is write cache free ?
Yes
Write data to the write cache
No
Select the stripe which has maximum number of pages from the write cache
1. According to the selection of pages to generate the partial parity
2. Generate the partial parity maybe read old data from storage devices
Is Parity buffer free ?
Yes
1. Save the partial parity to the PPC
2. Remove pages from write cache to the storage device
No
1. Select the partial parity by LRU algorithm
2. Rebuild the full parity that maybe read old data from storage devices

Fig. 3.5 The flowchart of PPC [14].

Fig. 3.6 shows the flowchart of the proposed efficient buffer method. The major difference between PPC [14] and our method is the data stripe selection in the write cache. The victim stripe selection rule is that the stripe is in both PPC and the write cache will be chosen firstly. This stripe must contain maximum number of pages. If the related partial parity of the stripe exists in the PPC, we can merge the existing partial parity and don't need to request another for PPC space. When PPC space is full, it will need to compute the full parity and writes back parity data to the storage device, thus the overhead is very huge. We also use a data buffer to reduce the cost of the partial parity update, because the controller maybe read data from the storage devices for updating the partial parity.

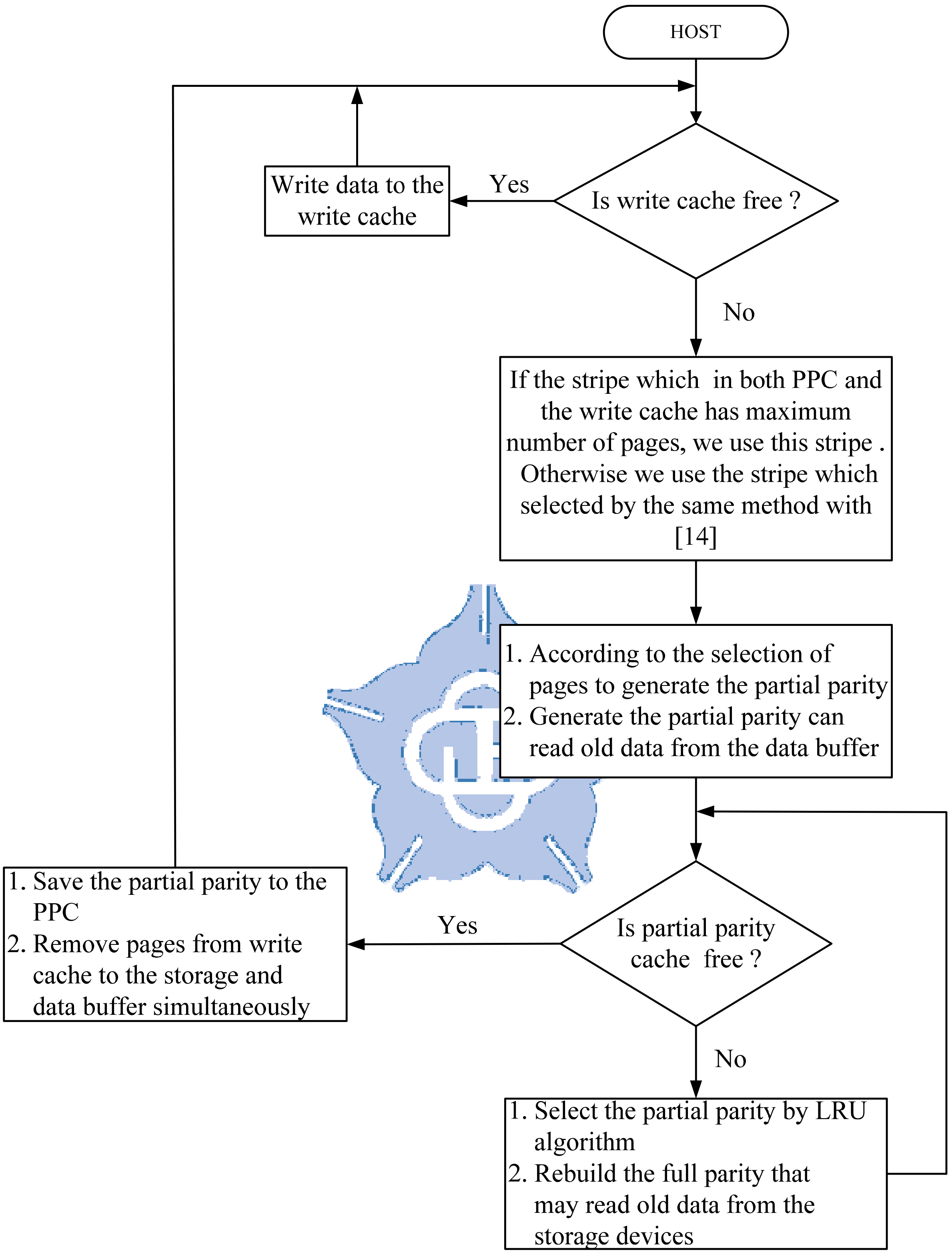

HOST
Is write cache free ?
Yes
Write data to the write cache
No
If the stripe which in both PPC and the write cache has maximum number of pages, we use this stripe . Otherwise we use the stripe which selected by the same method with [14]
1. According to the selection of pages to generate the partial parity
2. Generate the partial parity can read old data from the data buffer
Is partial parity cache free ?
Yes
1. Save the partial parity to the PPC
2. Remove pages from write cache to the storage and data buffer simultaneously
No
1. Select the partial parity by LRU algorithm
2. Rebuild the full parity that may read old data from the storage devices

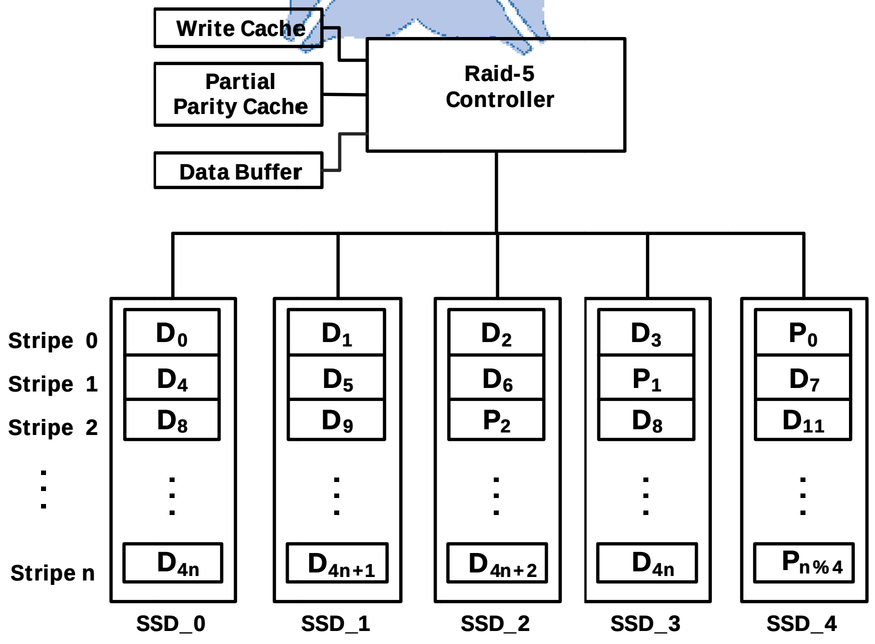
Fig. 3.6 The operation flowchart of proposed scheme.

3.2 Overall Architecture

The proposed 4+1 RAID-5 architecture is shown in Fig. 3.2. We use SSDs to construct the RAID-5 architecture. We propose an efficient buffer management scheme in the RAID-5 controller and a data buffer to reduce the cost of full parity generations.

The write cache holds the data from operating system until the write cache is full, and Non-Volatile RAM (NVRAM) is used to be a write cache. The partial parity cache stores the partial parity until the partial parity cache is full. We also use the NVRAM to be a partial parity cache. The data buffer stores data from the write cache evictions. The data stored in the data buffer can be reloaded from the SSDs, and therefore we use the SRAM or DRAM to be a data buffer.

Both the write cache and partial parity cache are NVRAM. When the system shuts down unexpectedly, the data which is in the write cache or the partial parity cache will be used to recover the data.



Write Cache
Partial Parity Cache
Data Buffer
Raid-5 Controller
Stripe 0
Stripe 1
Stripe 2
Stripe n
D_0 D_4 D_8 D_{4n}
D_1 D_5 D_9 D_{4n+1}
D_2 D_6 P_2 D_{4n+2}
D_3 P_1 D_8 D_{4n}
P_0 D_7 D_{11} $P_{n\%4}$
SSD_0
SSD_1
SSD_2
SSD_3
SSD_4

Fig. 3.2 The proposed RAID-5 architecture

3.3 The structure and size of each buffer and cache

Fig. 3.3 shows the data structure of two caches in the proposed RAID-5 controller. The m represents the number of entries in the cache. Both partial parity cache and data buffer has the same number of entries. T represents total data stripe number in the RAID-5, and n represents the number of storage devices in the RAID. The size of one partial parity is one page size. The n-bit field is used to represent the associate data with that partial parity. For example, a 4-bit binary value “1100” for the stripe number (S_0) means that the partial parity is generated with D_0 and D_1. In addition, the D_0 and D_1 are in the data buffer.

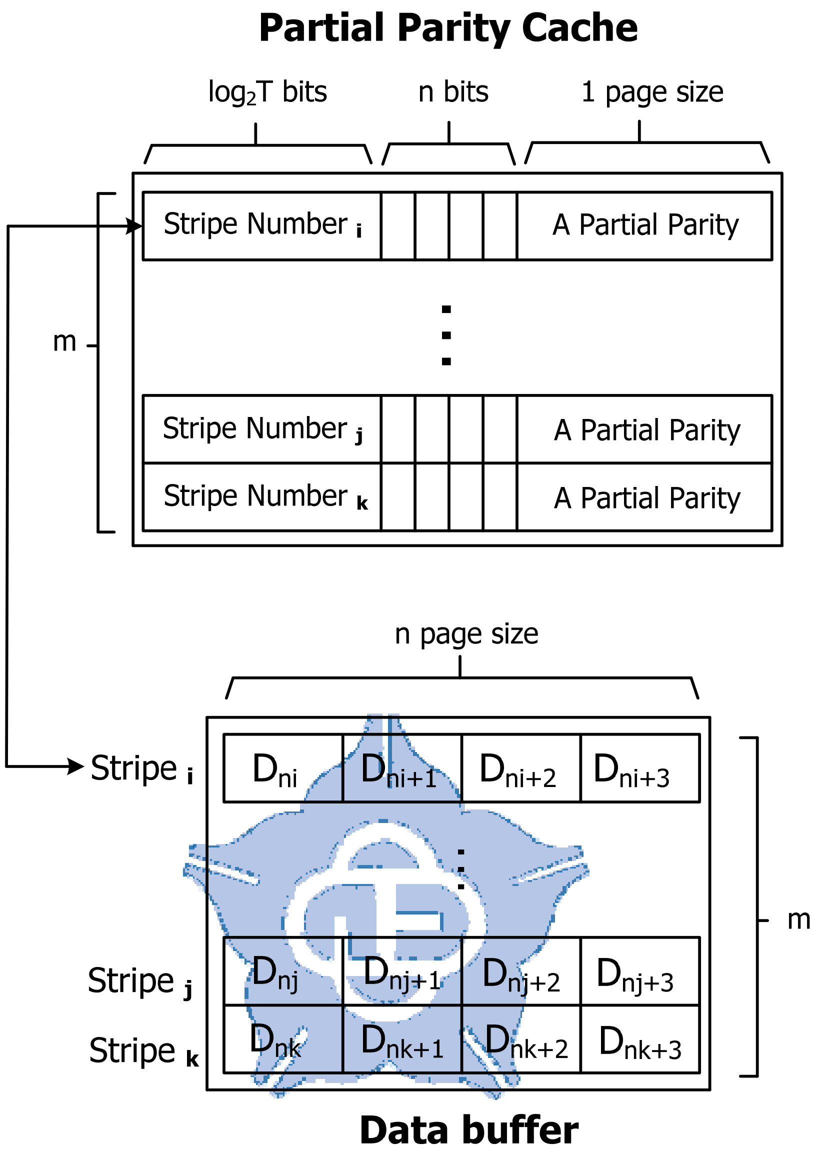


Fig. 3.3 The structure of the PPC and the data buffer

The data structure of the write cache is shown in Fig. 3.4. The data structure of the write cache is similar to the partial parity cache. The major difference is that the write cache contains data values in each entry. The n represents the number of storage devices in the RAID. The size of data area is that n * page size.

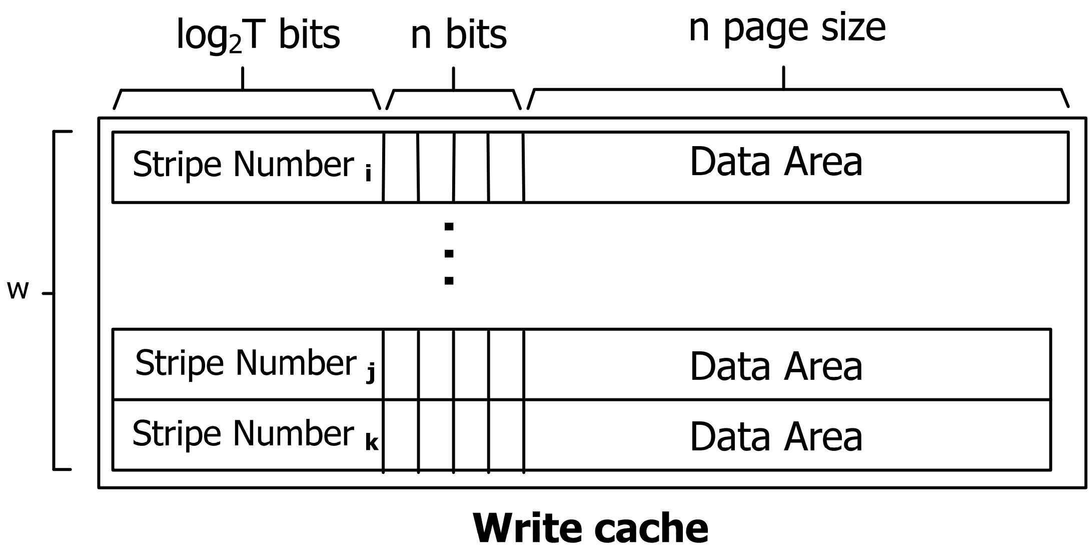


Fig. 3.4 The structure of the write cache.

3.4 The operation of proposed buffer management

We discuss the operations with each cache in this section. The write cache stores the data from the host system. When the write cache is full, the RAID controller will select the victim stripe from the write cache. For example, Fig. 3.7 shows that we select the stripe S_0 to be a victim stripe, because the S_0 contains the most number of data i.e. (three data). The remainder stripe S_5 and S_8 are still stayed in the write cache to incorporate more data.

When the stripe S_0 is selected, the controller performs the following operations. The steps are as follows:

1. Generate the partial parity P_0 by D_0, D_2, and D_3 and store ingredient bit to the PPC.
2. Write D_0, D_2, and D_3 to the corresponding locations in data buffer.
3. Write D_0, D_2, and D_3 to the corresponding locations in SSDs.

Fig. 3.7 shows the condition when system is reset, thus there is no partial parity in the PPC, so the RAID controller directly creates a partial parity. The partial parity P_0 is held in the PPC and doesn't write to the SSD. The partial parity P_0 has a great characteristic, the P_0 can incorporate the D_0, D_2, $D_{3,}$ and even D_4. When P_0 is still in the PPC, there is no full parity update cost for D_0 to D_4.

We don't have any write full parity operations to the SSDs so far. The data buffer always stores the newest data. No matter how the data exists in the data buffer, the new data directly overwrite the data in the data buffer.

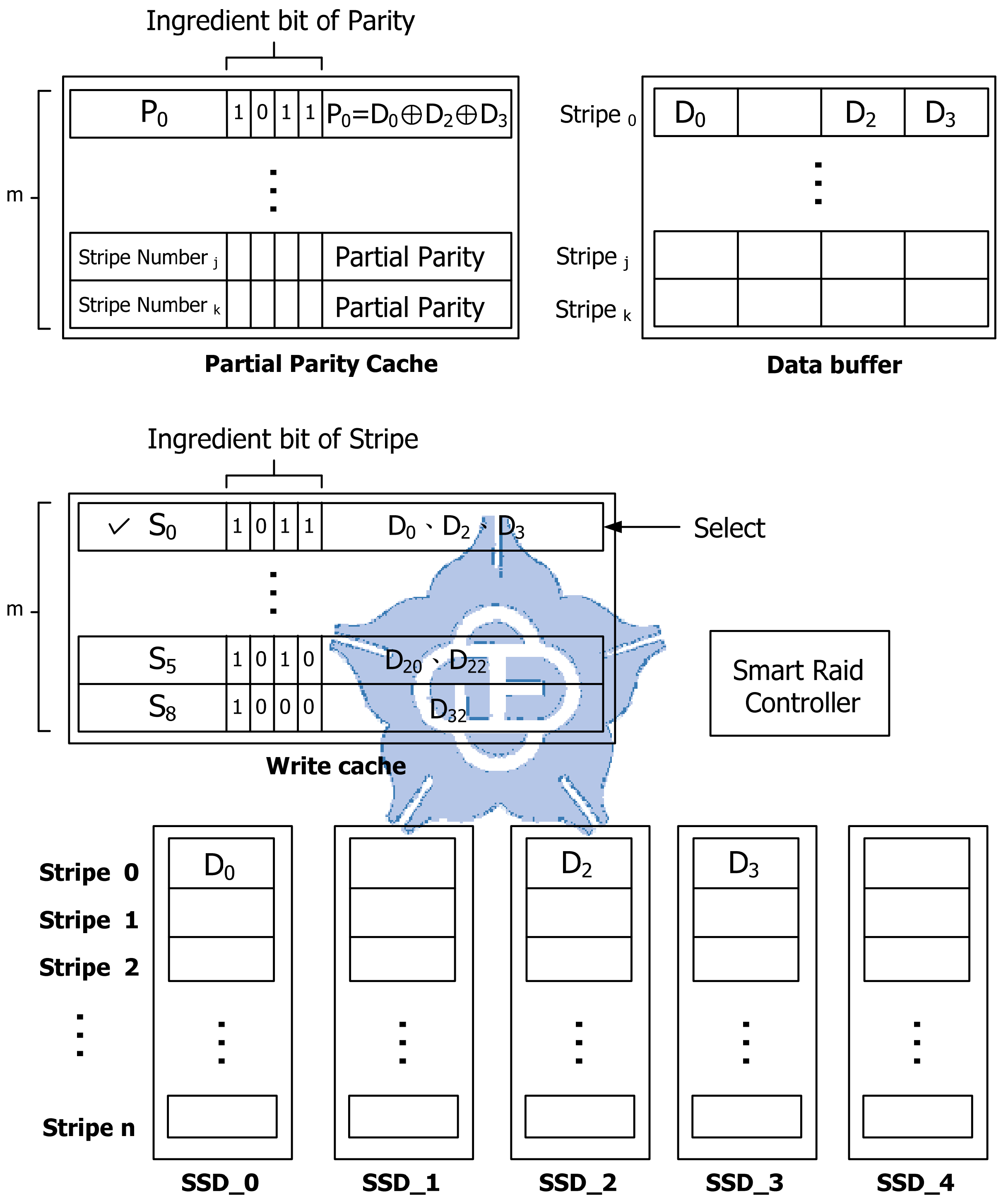


Fig. 3.7 The operation of proposed scheme.

Both PPC and write cache will not have enough free space as host system keeps access the RAID, they are encountered frequently selections of victim stripes or partial parity generations. Fig. 3.8 shows that how to use the partial parity to merge data. There are many entries in the write buffer which contain three data. In addition, we assume there is only one free space in the PPC.

In PPC [14], they will select any stripe which contains most of data to be a victim stripe. If the S_1 are selected to be a victim stripe, the PPC will be full. When the PPC is full, the controller will select the victim partial parity in the PPC by the LRU algorithm to generate and write back full parity to the SSDs. Thus the proposed method will select S_7 as a victim stripe since it both exists in the PPC and write buffer. If the controller selects partial parity P_7 to be a victim stripe, the parity generation costs are two read and one write operation to the SSDs. The two read operation is that the controller must read D_{29} and D_{31} from the SSDs to build the full parity, because the current partial parity P_7 is only related to D_{28} and D_{30}. The one write operation is that the generated full parity must be written to the SSDs.

The proposed efficient buffer method will check whether the data stripe in the write buffer is also in the PPC or not. We will select the S_7 rather than S_1 to be a victim stripe. The stripe S_7 in the write cache can be merged with the P_7 in the PPC, so the partial parity P_7 can update D'_{28}, D_{29}, and D_{31} to be a new partial parity P_7. The steps are as follows:

1. Update the new data to be a new P'_7

   $P'_7 = P_7 \oplus D_{28} \oplus D'_{28} \oplus D_{29} \oplus D_{31}$ (D_{28} is from the data buffer)

2. Write D'_{28}, D_{29}, and D_{31} to the data buffer (D_{28} will be replaced by D'_{28})

3. Write D'_{28}, D_{29}, and D_{31} to SSDs (in SSD_0, D_{28} are overwritten by D'_{28})

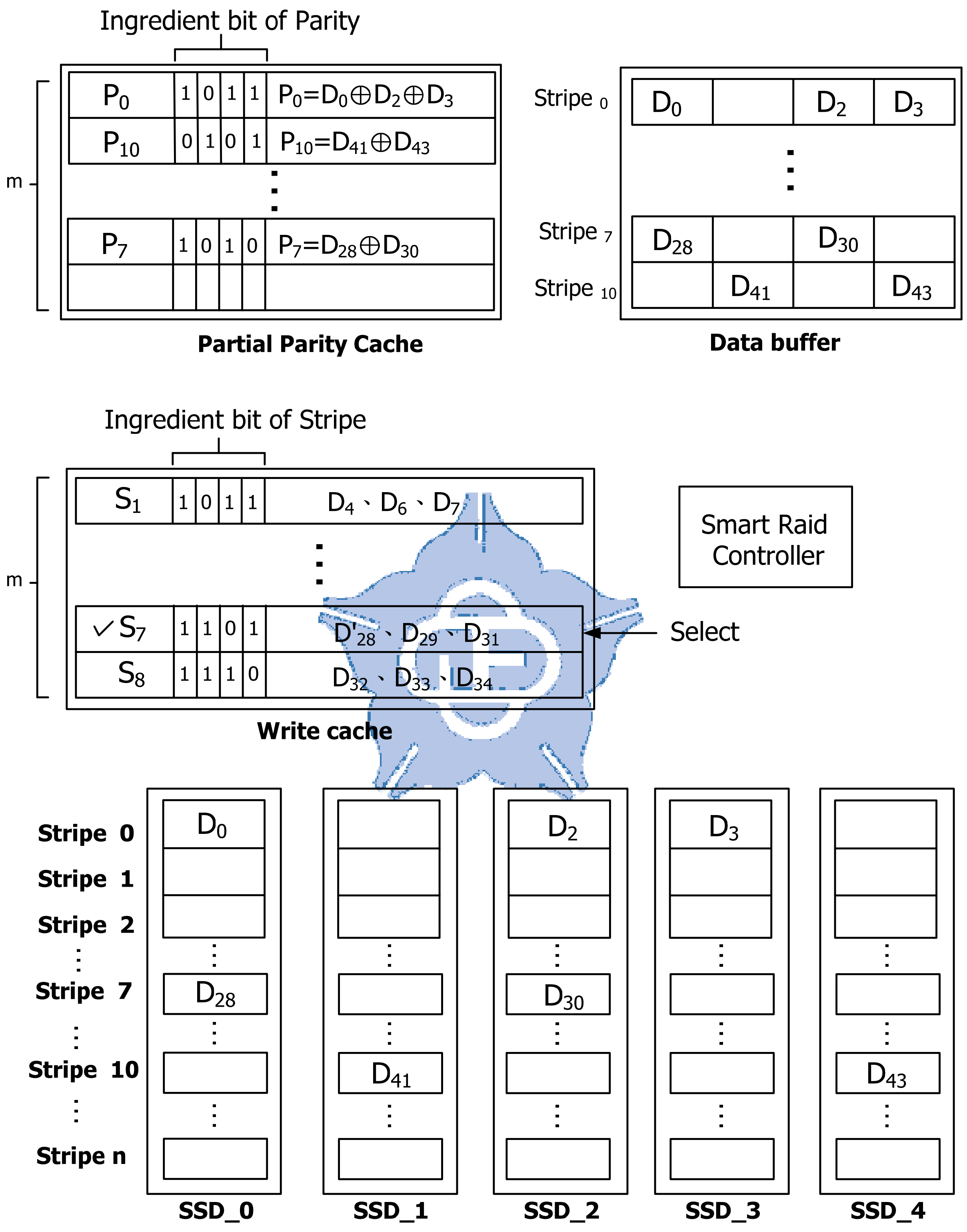


Fig. 3.8 The operation of the proposed parity merge.

The new partial parity P’$_7$ replaces the old partial parity P$_7$ and does not occupy a new space in the PPC. Therefore, the PPC will not be full in this example and we can reduce the number of full parity generation and write back parity data to the SSDs.

We compare the PPC [14] partial parity generation costs with the proposed efficient buffer method as follows according to this example.

- PPC [14] costs : (Either S$_1$ or S$_7$ is selected)

  Two read operations from SSDs and one write operation to SSDs

- The proposed method costs : (only S$_7$ will be selected)

  No costs

Our method can reduce the parity data write times to the SSDs, and we use the PPC characteristics and merge the data with existing partial parity as many as possible.

3.5 The overhead of write back full parity

Fig. 3.9 shows that all situations of the full parity write back cost. There are two methods to rebuild the full parity as follows:

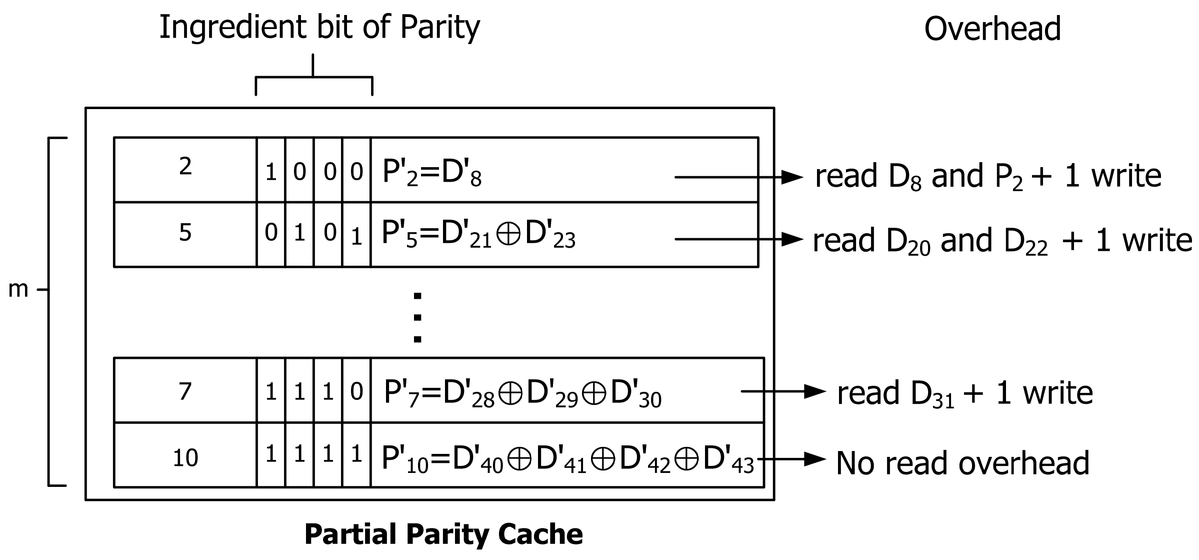
Method 1: Read the corresponding old data and old parity to rebuild the full parity.

Method 2: Read the other data which is not ingredient of the stripe.

When P'_2 writes back to the SDDs, the controller adopts the method 1 to rebuild the full parity rather than method 2 in Fig. 3.9. The reason is that the method 2 has three read costs (read D_9, D_{10}, and D_{11}) are more than method 1 cost. The full parity of P'_2 is $P'_2 \oplus D_8 \oplus P_2$.

The controller will determine which method is efficient. However, the method 1 must record the old data and old parity position so that the controller can read the old data or old parity successfully. When P'_5 and P'_7 must write back to SSDs, the controller will adopt the method 2 to rebuild the full parity. The full parity of P'_5 is $P'_7 \oplus D_{20} \oplus D_{22}$ and The full parity of P'_7 is $P'_7 \oplus D_{31}$.

The partial parity P_{10} is the best case. There are no overheads to write P_{10} back to the SSDs, because P_{10} has all ingredients data, the partial parity is also the full parity.



Ingredient bit of Parity
Overhead
m
2
1 0 0 0
$P'_2=D'_8$
read D_8 and P_2 + 1 write
5
0 1 0 1
$P'_5=D'_{21}\oplus D'_{23}$
read D_{20} and D_{22} + 1 write
7
1 1 1 0
$P'_7=D'_{28}\oplus D'_{29}\oplus D'_{30}$
read D_{31} + 1 write
10
1 1 1 1
$P'_{10}=D'_{40}\oplus D'_{41}\oplus D'_{42}\oplus D'_{43}$
No read overhead
Partial Parity Cache

Fig. 3.9 The overhead of write full parity back to SSDs.

3.6 The recovery when SSDs failure

The SSDs may fail due to the limited erase times of the SSDs, and thus the life time of SSDs is finite. When a SSD fails in the storage system, we can remove the failure SSD and replace a new SSD with the same capacity, brand, and model.

The controller has two ways to recover the lost data. If the stored data which are part of the partial parity, the controller must use the partial parity to recover the SSD, as shown in Fig. 3.10. If not, the controller uses the original RAID-5 scheme to recover the data.

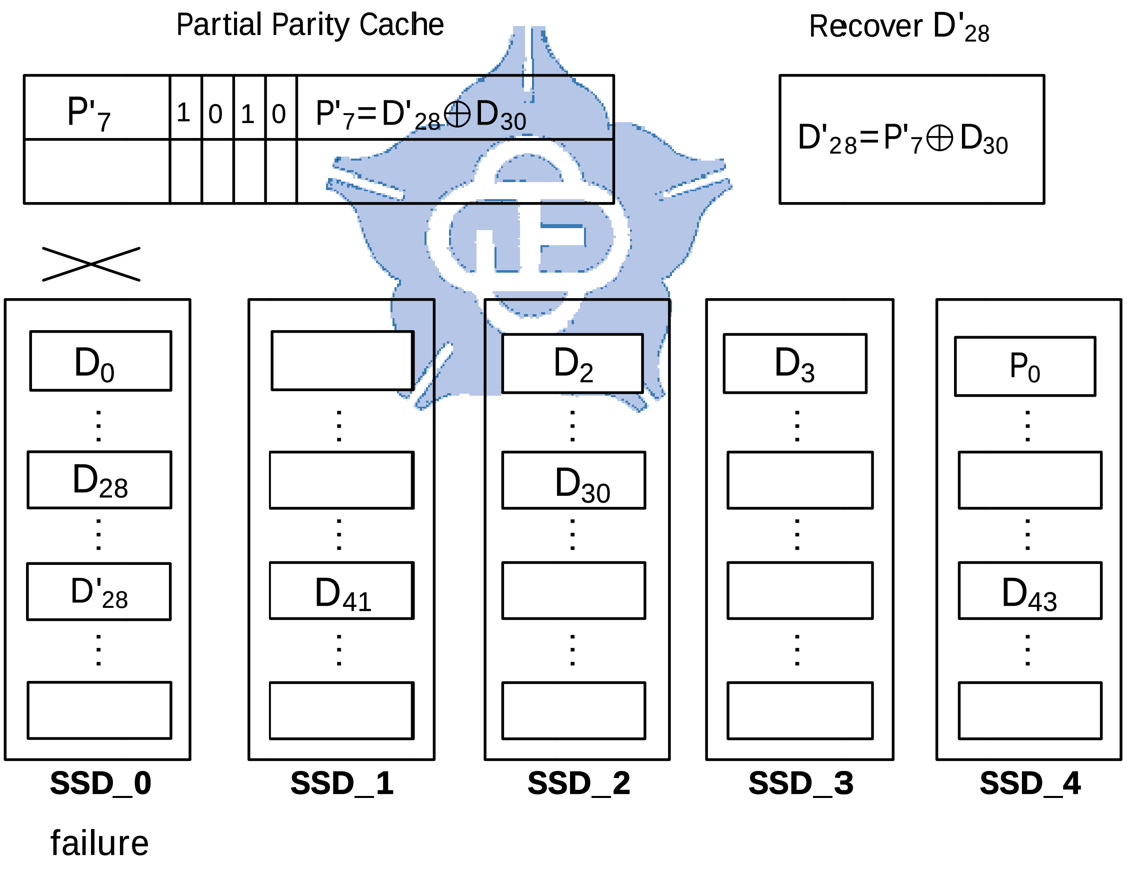


Fig 3.10 Recovery data by partial parity cache

Chapter 4

Experimental Results

4.1 Experiment preparation

We implement a RAID-5 simulator to evaluate the performance of the proposed RAID controller, as shown in Fig. 4.1. The entire system consists of a RAID-5 simulator and a SSD model. The RAID-5 simulator accepts the inputs which from the benchmark profiling results. The RAID-5 controller determines the stripe number, SSD number, and computes parities.

The stripe number and SSD number is generated by the logical address divided by total number of data storage devices, and then the quotient and reminder is stripe number and SSDs number, respectively. For example, if the RAID system is RAID-5 (4+1) and the logical address is 2045. The 2045 is divided by 4, and then the quotient and reminder is 511 and 1, respectively. Therefore, the data (logical address is 2045) is written to the SSD_1 at stripe S_{511}. The RAID controller also manages the write cache, partial parity cache and the data buffer.

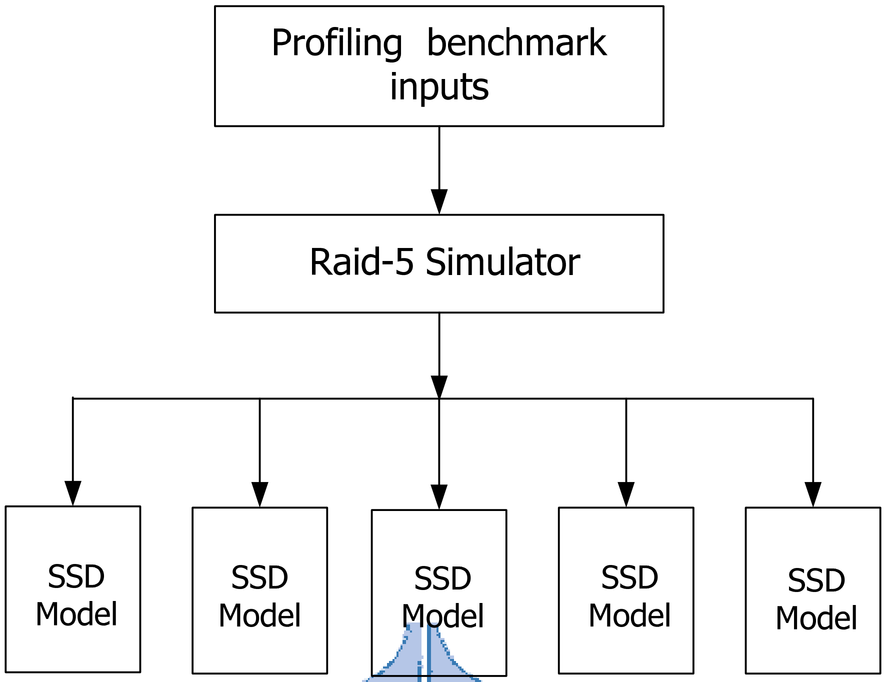


Fig. 4.1 The RAID-5 simulator.

We profile two benchmarks: iozone [27] and postmark [28] as inputs. From the profiling results, we use iozone and postmark as a sequential writing test and random writing test, respectively.

We assume the specification of our RAID system as follows:

1. SSD’s page size is 2KB.
2. The write cache is 16KB.
3. The data buffer and the PPC are 16KB.
4. All the comparisons are based on RAID-5 (4+1).

We use a small size of the write cache and the PPC, because we want to emphasis on that the efficient buffer method can reduce the overhead without too much hardware costs. We build four types of RAID storage systems as follows:

1. RAID-5	(only has one write cache)
2. FPC	(RAID-5 and full parity cache)
3. PPC	(RAID-5 and Partial Parity Cache [14])
4. The proposed	(RAID-5, PPC, and a data buffer)

4.2 The overhead of parity generation

The RAID-5 is the common scheme. The FPC uses a full parity scheme with a parity cache. In Fig. 4.2 to Fig. 4.9, we normalize the number of read and write times to the SSDs by the input total write requests (i.e. 4000).

Fig. 4.2 shows that the average write request overhead for generating the parities. We analyze four types RAID schemes: RAID-5, FPC, PPC, and the proposed method.

The original RAID-5 scheme doesn't have a parity cache so that the number of read times and for generating the parities write times is highest. When the write cache in the RAID-5 scheme is full, the controller will select the victim data to write back to the SSDs. According to the selection data, the controller will build the full parity and write back the full parity to the SSDs directly. When the controller generates the full parity, it needs to read the remainder data of the victim stripe, and these read operations are the overhead for full parity generation, so the number of read times is also highest in the RAID-5 scheme. Therefore, it is not suitable to dispose SSDs to the RAID-5 directly.

The FPC scheme has a full parity cache, the write times for the parities can be decreased obviously in the FPC scheme, but the read times for full parity generation are almost the same as the RAID-5 scheme. The reason is that the full parity generation in FPC is the same as the RAID-5. The main difference between the FPC and RAID-5 is that the full parity stays in the parity cache until the parity cache is full. The parity cache help to reduce the write times of the parities, but read times are almost the same.

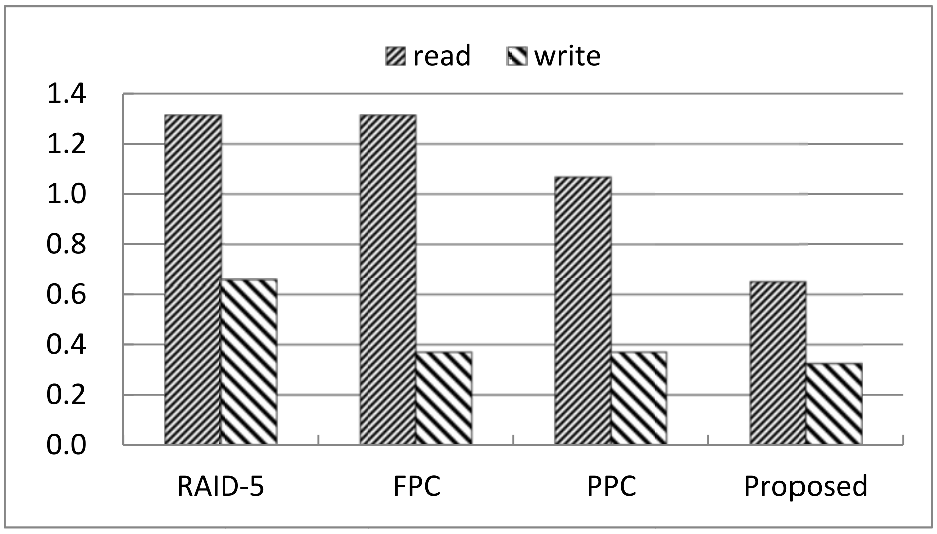


Fig. 4.2 The write request overhead for generating the parities (postmark)

(Normalized to 4000 write requests)

The PPC scheme adopts the partial parity scheme, and it can reduce the read times for parity generation. The write times of the parities are almost the same as the FPC scheme. We can give two conclusions from above experimental results.

1. The parity cache can reduce the write times of parity data.
2. The partial parity scheme can decrease the read times for generating the full parity.

The proposed scheme use an efficient buffer method with a data buffer to reduce both read times and write times as shown in Fig. 4.2. The normalized write times for the parities of RAID-5, FPC, PPC, and the proposed scheme is 0.66, 0.37, 0.37, and 0.32, respectively. The proposed scheme decreases the number of write times for the parities by 13 percent and 51 percent respectively as compared to PPC and RAID-5. The normalized read times for parity generation of RAID-5, FPC, PPC, and the proposed scheme is 1.31, 1.31, 1.068, and 0.65 respectively. The proposed scheme decreases the number of read times for parity generation by 39 percent and 50 percent respectively, as compared to PPC and RAID-5.

Fig. 4.3 shows the simulation results for iozone benchmark. The normalized write times for the parity generation of RAID-5, FPC, PPC, and the proposed scheme is 0.3, 0.19, 0.19, and 0.18 respectively. The proposed scheme decreases the number of write times by 5 percent and 40 percent respectively as compared to PPC and RAID-5. The write times can be decreased due to parity cache and the efficient buffer method.

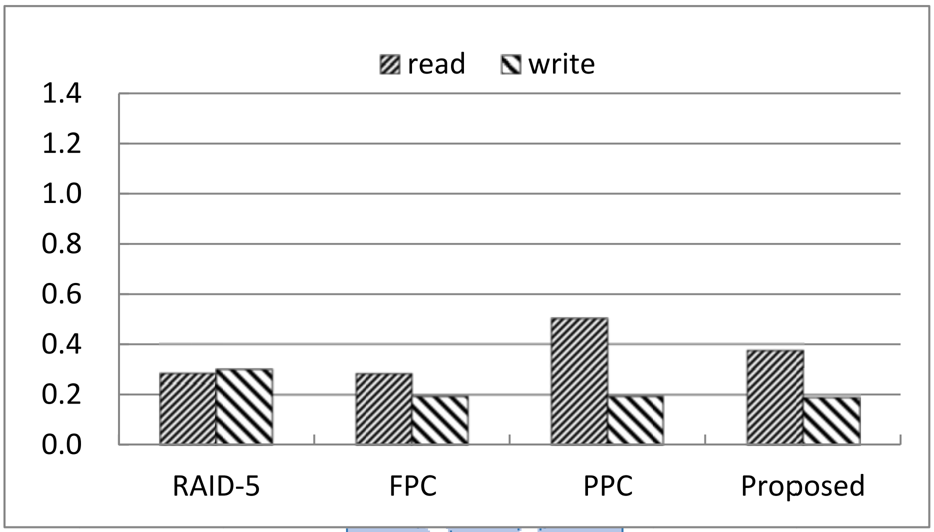


Fig. 4.3 The average write request overhead for generating the parities (iozone)

(Normalized to 4000 write requests)

The normalized read times for parity generation of RAID-5, FPC, PPC, and the proposed scheme is 0.28, 0.28, 0.5, and 0.32 respectively. The proposed scheme can effectively decrease the number of read times for parity generation by 36 percent as compared to PPC.

However, the read times for parity generation of both PPC and the proposed scheme is larger than FPC and RAID-5. The reason is that we use iozone benchmark as sequential inputs. There are many re-write requests which are the files are already existed in the storage system, so there are many partial parity update. The read times ingredients as follows:

- The read times of PPC [14] :

  partial parity update + full parity write back
- The read times of the proposed method :

  full parity write back

The PPC's read times are divided into two parts: partial parity update and full parity write back. The PPC's read times is 0.5 which is the sum of 0.32 (partial parity update) and 0.18 (full parity write back). This is why we need to add a data buffer. The data buffer removes the read times of partial parity update and the hardware cost is not too expensive.

4.3 The entire performance

Fig. 4.4 shows that the I/O performance in each RAID type. The performance is normalized to the original RAID-5 scheme. The performance of RAID-5, FPC, PPC, and the proposed scheme is 1.0, 1.13, 1.47, and 1.76, respectively with postmark benchmark. The I/O performance of the proposed scheme is improved by 76 percent and 19 percent as compared to RAID-5 and PPC, respectively.

The detail analysis of the entire I/O performance are listed as follows:

●	Wcycle	: The cycle spent in the write cache
●	Dcycle	: The cycle spent in the write back data and data buffer.
●	Rcycle	: The cycle spent in the read operation for generating full parity or updating partial parity.
●	WPcycle	: The cycle spent in the write back the full parity.
●	EBF	: The cycle spent in the Efficient Buffer Method.
●	LRU	: The cycle spent in the Least Recently Used method.
●	Tcycle	: Total cycle of the operation.

In the proposed method, the total cycle can be expressed as

- Tcycle = Wcycle + Dcycle + Rcycle + WPcycle + EBF + LRU

In the PPC [14], the formula for the total cycle of operation is

- Tcycle = Wcycle + Dcycle + Rcycle + WPcycle + LRU

In addition, the total cycle of operation for RAID-5 and FPC:

- Tcycle = Wcycle + Dcycle + Rcycle + WPcycle

Our scheme handles the random requests very well, because the parity merge will often happen in the partial parity cache with random data input. The Rcycle and WPcycle of the proposed approach are smaller than the PPC [14]. The random access request is more important than sequential request.

The performance of the iozone benchmark (sequential access requests) in each RAID type is nearly the same, because the controller almost selects whole ingredients data of the stripe to write back to SSDs. For example, the controller selects 4 data: D_0, D_1, D_2, and D_3 to write back to SSDs. The 4 data: D_0, D_1, D_2, and D_3 are written back to SSD_0, SSD_1, SSD_2, and SSD_3 simultaneously. When controller selects the full data (4 data), both of PPC and proposed scheme don’t need to rebuild the full parity, the overhead of generating partial parity and full parity is the same. The Rcycle and WPcycle in the each scheme are almost the same in this case. Both partial parity and the full parity is $D_0 \oplus D_1 \oplus D_2 \oplus D_3$ in this example. Besides, the EBF is very smaller, because the Efficient Buffer Method only spends 1 or 2 cycles to manage the buffer.

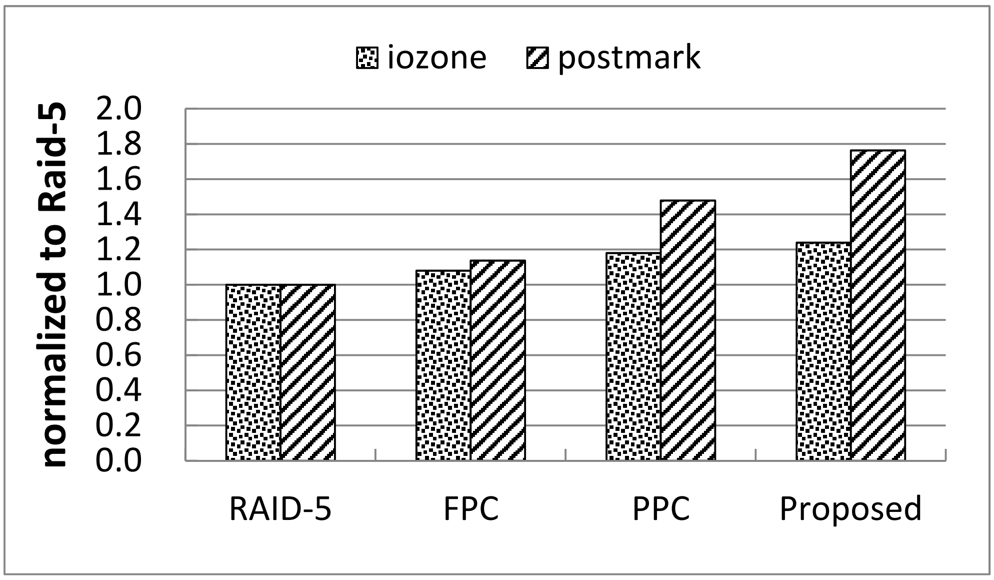


Fig. 4.4 The overall I/O performance

4.4 Different parameters with proposed scheme

4.4.1 The stripe size

Fig. 4.5, Fig. 4.6, and Fig. 4.7 show the simulation with different stripe size. Both write cache and partial parity cache are all 16 KB. The size of data buffer in the proposed management method is the same with the stripe size.

Fig. 4.5 shows the comparisons of the average read times for generating the parity with different stripe size. The read count of proposed scheme is smaller than the PPC scheme. There are two reasons explain why the read count is smaller in the proposed scheme as follows:

1. The write times of the proposed scheme is smaller than PPC scheme as shown in Fig. 4.6.
2. The proposed scheme adds a data buffer for reducing the number of read times.

Each write full parity back to SSDs operation means that the controller must build the full parity. Then it may accompany with read data from the SSDs, so the read times are affected by the number of write times. The data buffer avoids the read operations from SSDs overhead for full parity generation.

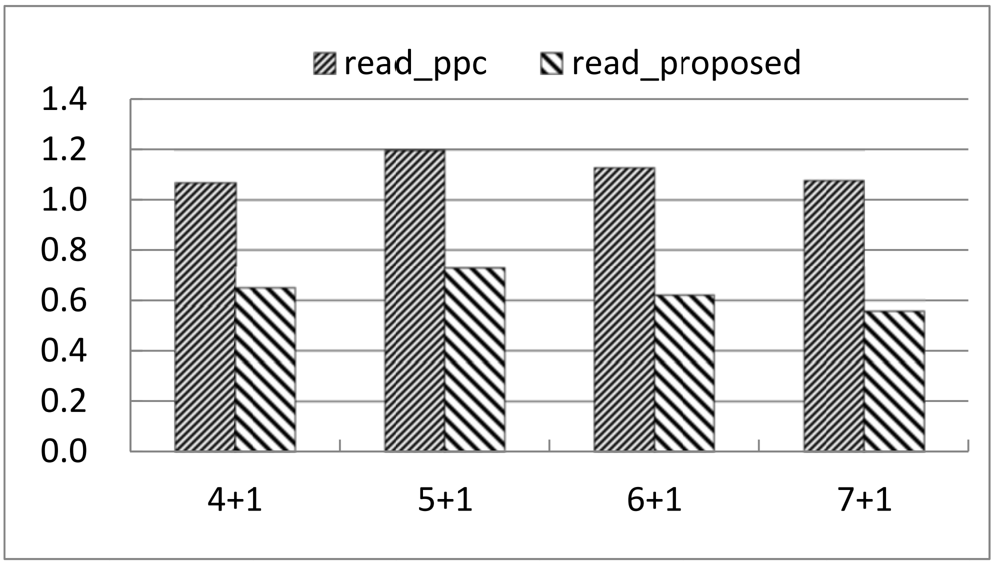


Fig. 4.5 The average read operation overhead for generating the parities generation (postmark) (Normalized to 4000 write requests)

Fig. 4.6 shows the comparisons of average write times for parity write back to SSDs with different stripe size. When the stripe size is larger, the write times are decreased in PPC and the proposed scheme. The reason is that the partial parity can merge more data. For example, the partial parity P_0 in 4+1 scheme represents D_0 to D_3. However, the partial parity P_0 in 7+1 scheme represents D_0 to D_7. It means that possibilities of merging partial parities are increased.

Write times of the proposed scheme are smaller than the PPC. This is due to the proposed efficient buffer method as shown in Fig 3.6.

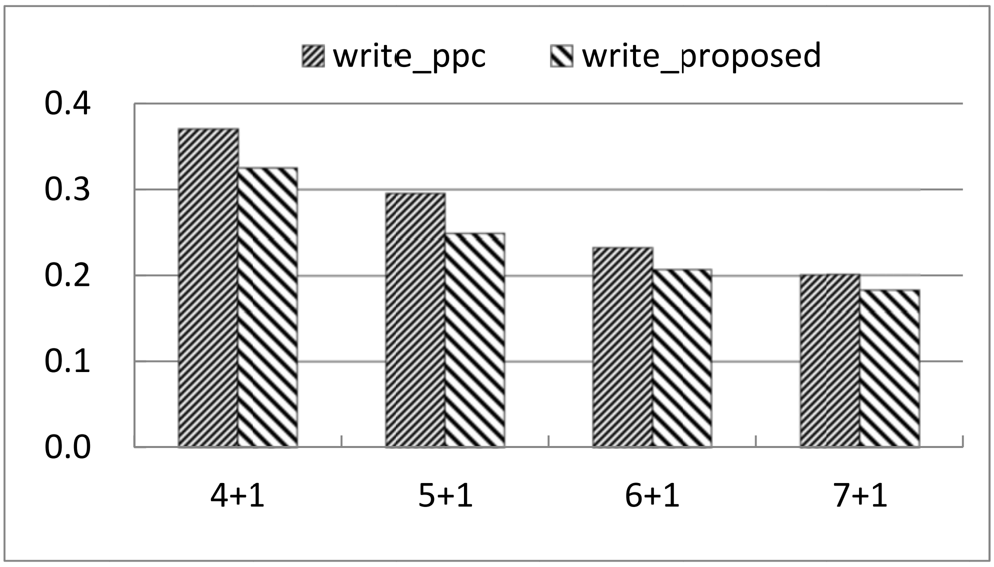


Fig. 4.6 The average write operations overhead for generating the parities (postmark)

(Normalized to 4000 write requests)

Fig. 4.7 shows I/O performance of the proposed scheme with different stripe size. The performance is normalized to 4+1. The I/O performance of 4+1, 5+1, 6+1, and 7+1 is 1, 1.15, 1.29, and 1.47 respectively. When the stripe size becomes larger, the I/O performance is improved. The reasons are as follows:

1. Both read times and write times are decreased when the stripe size becomes larger.
2. When stripe size becomes larger in the storage system, it means that the I/O parallelism is also better.

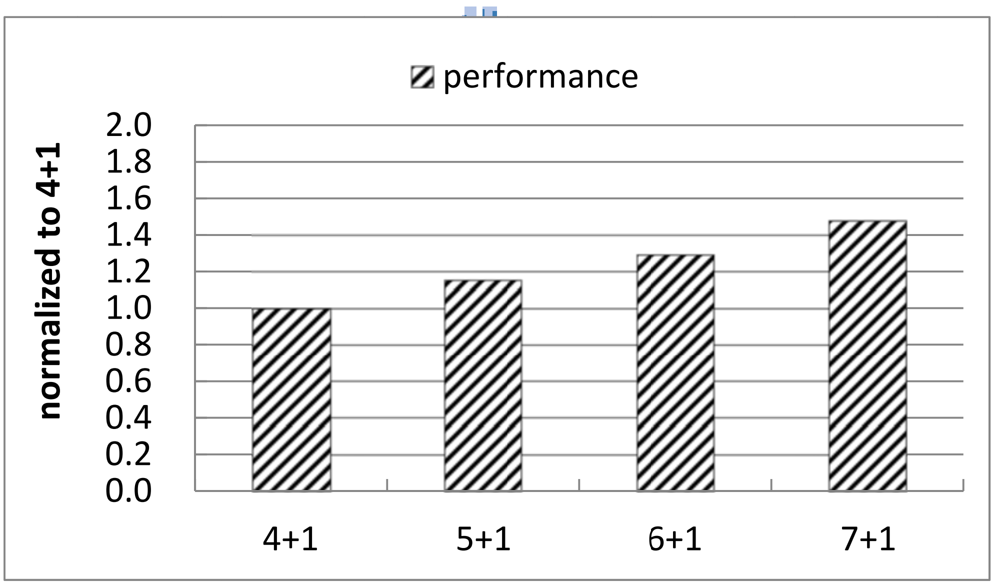


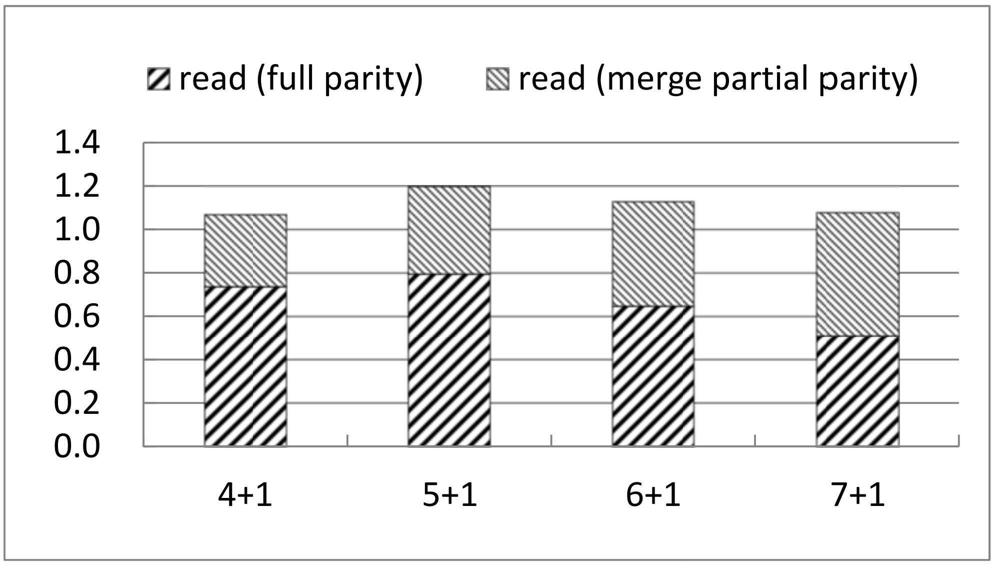
Fig. 4.7 The overall I/O performance with different stripe size (postmark)

Fig. 4.8(a) shows that the ratio of read operations of the PPC in postmark benchmark (random writing requests). The numbers of read times which is for full parity write back to SSDs is 0.73, 0.79, 0.64, and 0.51, respectively. The numbers of read times which is for partial parity update to SSDs is 0.33, 0.40, 0.48, and 0.56 respectively. We can find that the number of read times for partial parity update is increased gradually with a larger stripe size, because the partial parity associates with more data with a wider data stripe. The partial parity associate with more data means that there are more opportunities to merge data by partial parity.

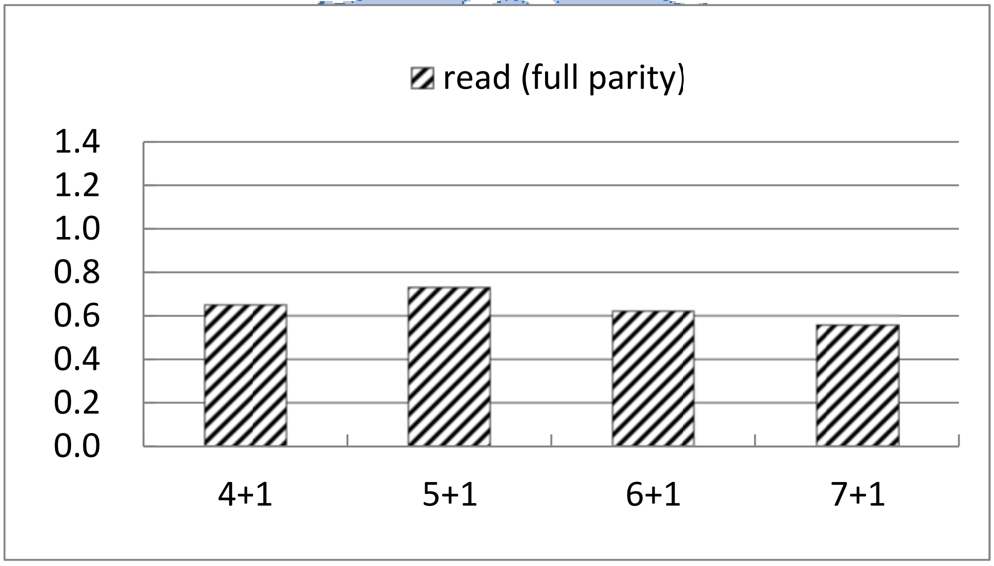
Fig. 4.8(b) shows that the ratio of read operations of the proposed scheme in postmark benchmark. The proposed scheme adds a data buffer; therefore, there is no read operation from the SSDs for parity update.

Fig. 4.9(a) shows the ratio of read operations of the PPC in iozone benchmark (sequential writing requests). The read times for full parity write back to SSDs with different stripe size are 0.18, 0.64, 0.53, and 0.34 respectively. The number of read times which is for partial parity update to SSDs is 0.32, 0.39, 0.47, and 0.54 respectively. We find that the read times for full parity generation in random writing requests are higher in sequential writing requests.

In Fig 4.9(b), the proposed scheme uses a data buffer to avoid the read operation from the SSD, and thus the total read times are reduced accordingly.



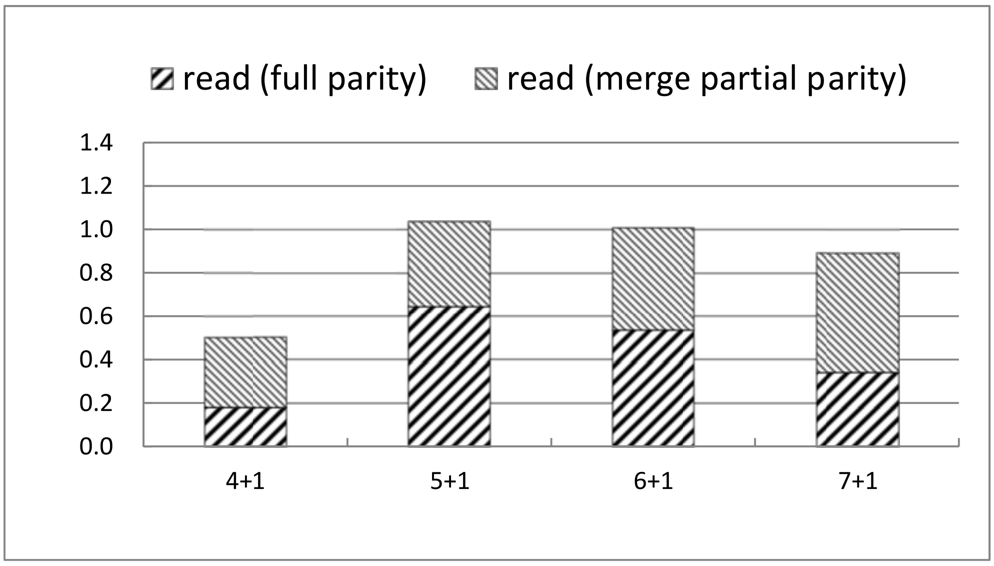
(a)



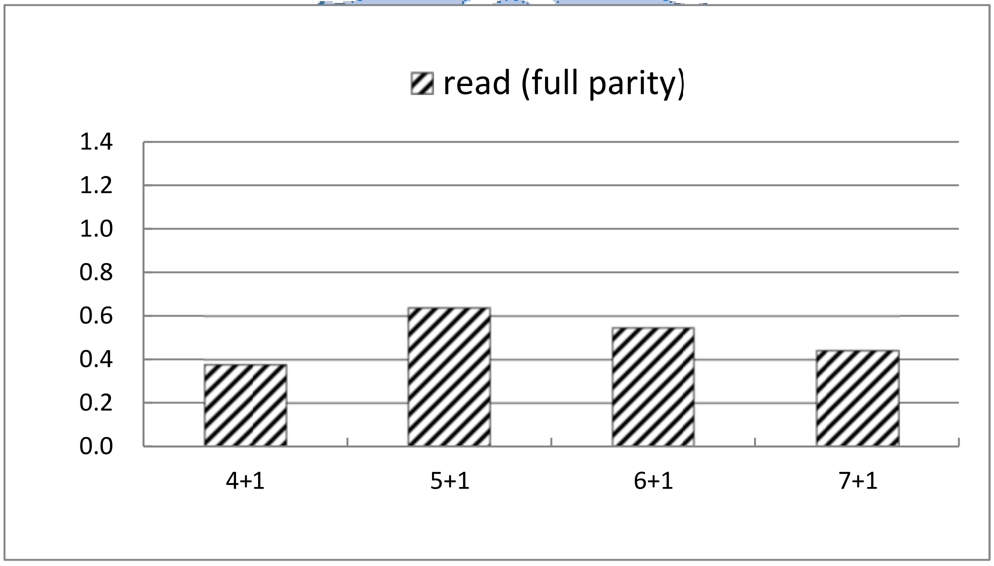
(b)

Fig. 4.8. The analysis of read operations for generating the parities (postmark).

(a) PPC scheme. (b) Proposed scheme. (Normalized to 4000 write requests)



(a)



(b)

Fig. 4.9. The analysis of read operations for generating the parities (iozone).

(a) PPC scheme. (b) Proposed scheme. (Normalized to 4000 write requests)

4.4.2 **Parity Cache size**

Fig. 4.10 shows that average size of victim pages in the write cache with two benchmark inputs. The different victim pages have the same stripe size. The ideal number of victim page is four with 4+1 RAID-5 scheme, so the ideal number of victim pages with n+1 RAID scheme is n.

The RAID scheme uses the parallelism I/O architecture well when the number of selection victim pages is ideal number. Besides, there is no read operation for generating full parity or partial parity in this case, because these victim pages only need to do exclusive-or with each other.

The average size of victim pages with iozone benchmark and Postmark benchmark are 3 pages and 1.3 pages, respectively. In iozone benchmark, the controller maybe need 1 read operation for generating the full parity. In Postmark benchmark, the controller maybe need 2.7 read operations for generating the full parity. Thus the number of read operations of the Postmark is larger than the iozone as shown in Fig 4.8 and Fig 4.9.

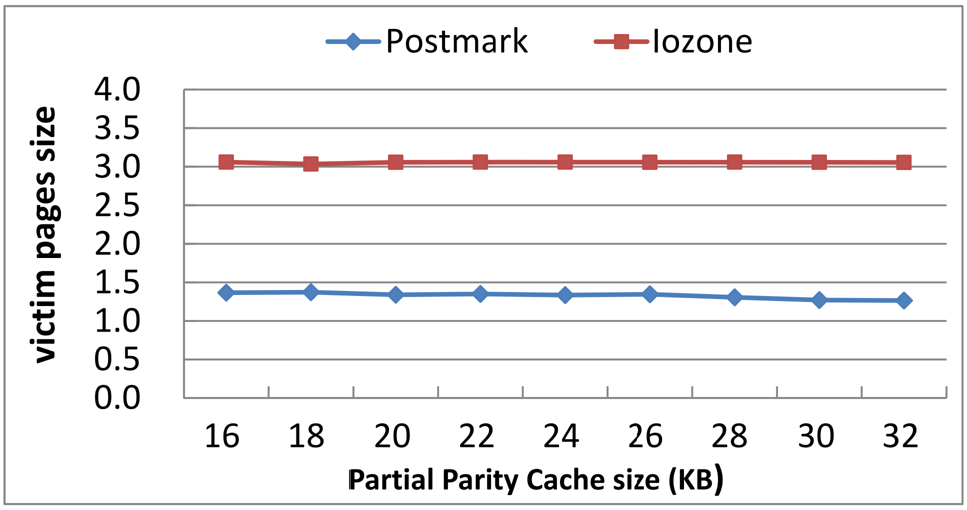
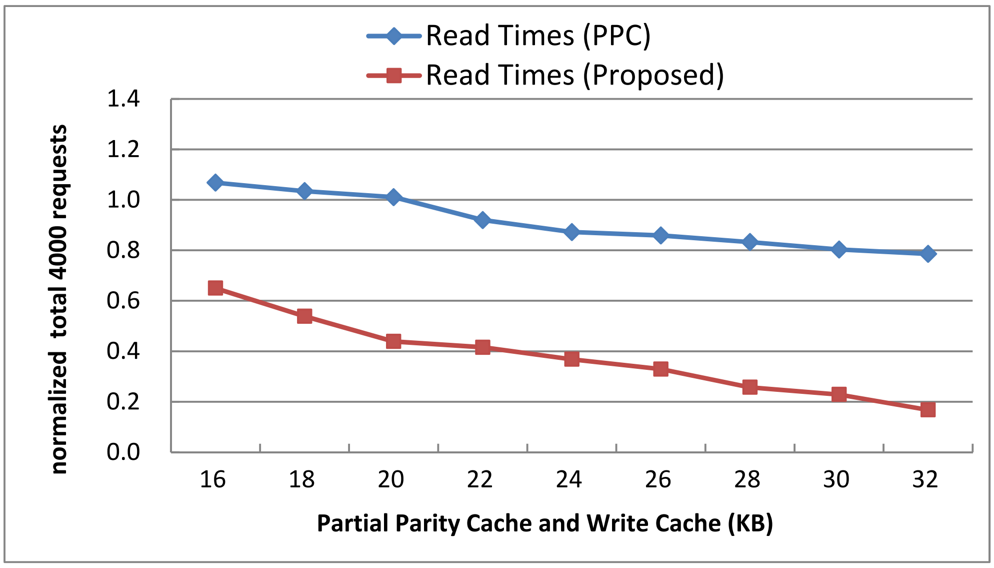


Fig. 4.10 Numbers of victim pages with proposed 4+1 RAID-5 scheme

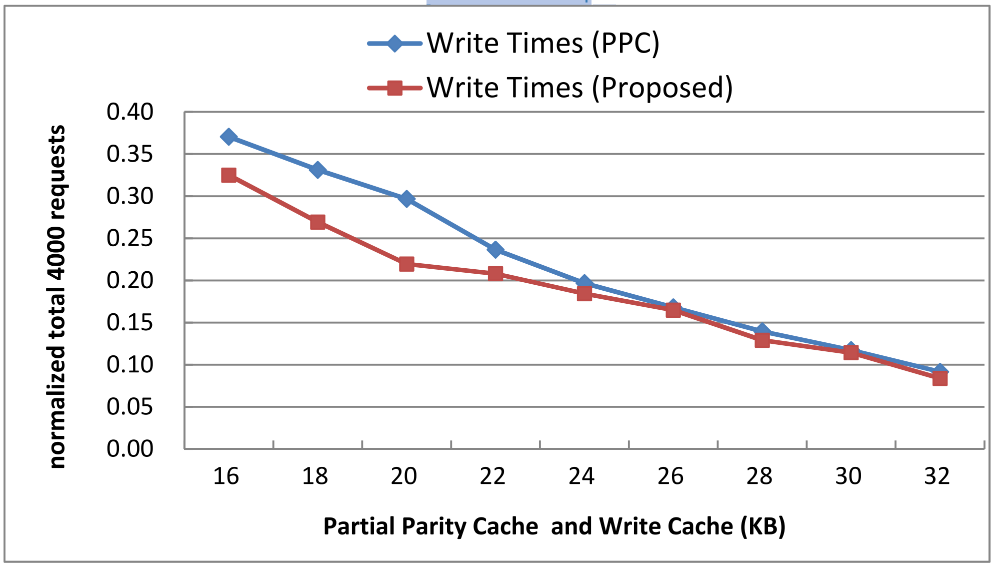
4.4.3 Parity cache size and write cache size

Fig. 4.11 shows that read times and write times for parity generation with different cache size (Postmark). Fig. 4.11(a) shows that the read times for parity generation of the proposed scheme is always smaller than the PPC. The read times of the proposed scheme are 0.65 and 0.16 with partial parity cache and write cache size at 16 KB and 32 KB respectively. The read times of PPC scheme are 1.06 and 0.90 with partial parity cache and write cache size at 16 KB and 32 KB respectively. The reason is that larger partial parity cache can store more partial parity data. When there are more partial data, we can merge partial parity more efficient. In addition, the data buffer in the proposed scheme also helps to reduce the read times for parity generation.

Fig. 4.11(b) shows that write times for full parity generation and write back to the SSDs with different cache size. The proposed scheme can reduce writes time even with a smaller cache. As a result, we can process the write operations well even when there are limited hardware resources. With a larger cache size, the write times can be reduced in the PPC scheme and proposed scheme. However, with the proposed efficient buffer scheme, the write times of the proposed design can be always smaller than the PPC scheme.



(a)



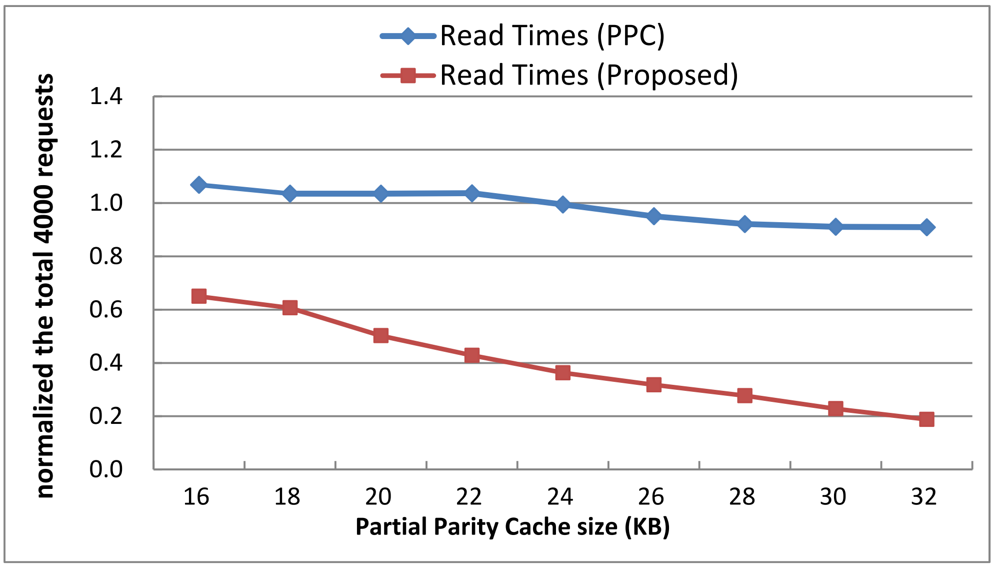
(b)

Fig. 4.11 The different of data buffer size and parity cache size with Postmark

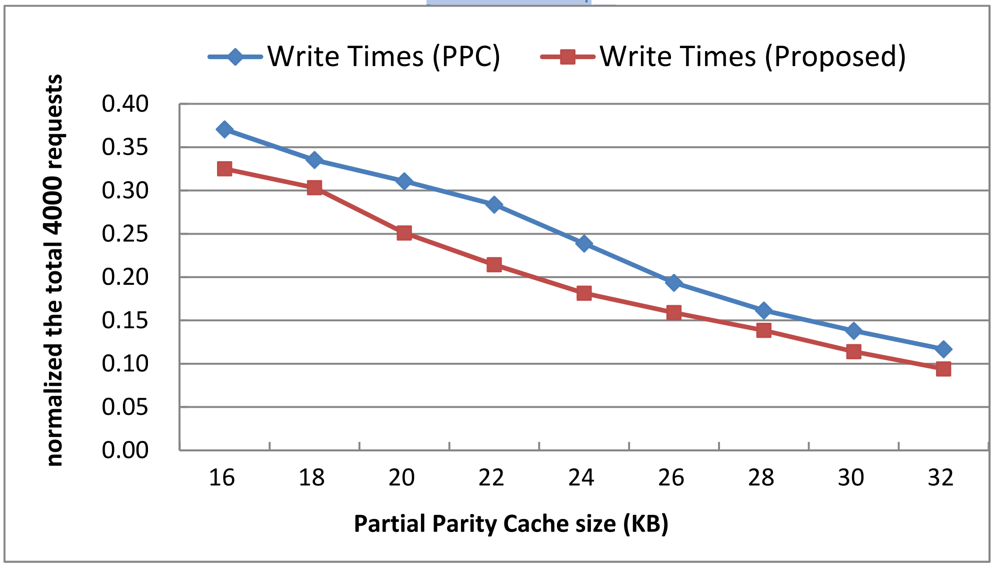
(a) Read Times (b) Write Times (Normalized to 4000 write requests)

Fig. 4.12 shows the simulation results with different partial parity cache size and a fixed 16 KB write cache size. The times is decreased as the size of partial parity cache is increased. Thus, we can only increase the size of partial parity cache to obtain better performance.

Fig. 4.12(b) shows that the write times for parity generation of the proposed scheme are all smaller than PPC scheme. We only need to add a larger partial parity cache as compared the simulation results shown in Fig 4.11. The reason is with a larger write cache we can only store more data when there are more data in the cache, the I/O performance is not improved obviously. In addition, the efficient buffer method can manage the partial parity very well.



(a)



(b)

Fig. 4.12 The different of partial parity cache size with Postmark

(a) Read Times (b) Write Times (Normalized to 4000 write requests)

4.4.4 Verification with FPGA

We use the Socle Technology Corporation MDK-3D development board to verify the proposed efficient buffer management scheme. The specifications as follows:

- The CPU is ARM1176JZF and the frequency is up to 1GHz.
- The AHB frequency is up to 200MHz.
- Support the NOR-flash/NAND-flash/DDR2
- The ROM size is 4096x32 bytes.

We implement the RAID-5 and the proposed method in the FPGA. The measurement results shows that both read and write times are reduced by the proposed method. The write buffer, write cache, and data buffer are all 8 KB due to the size limitations in the FPGA. The test pattern are 4096 random write requests.

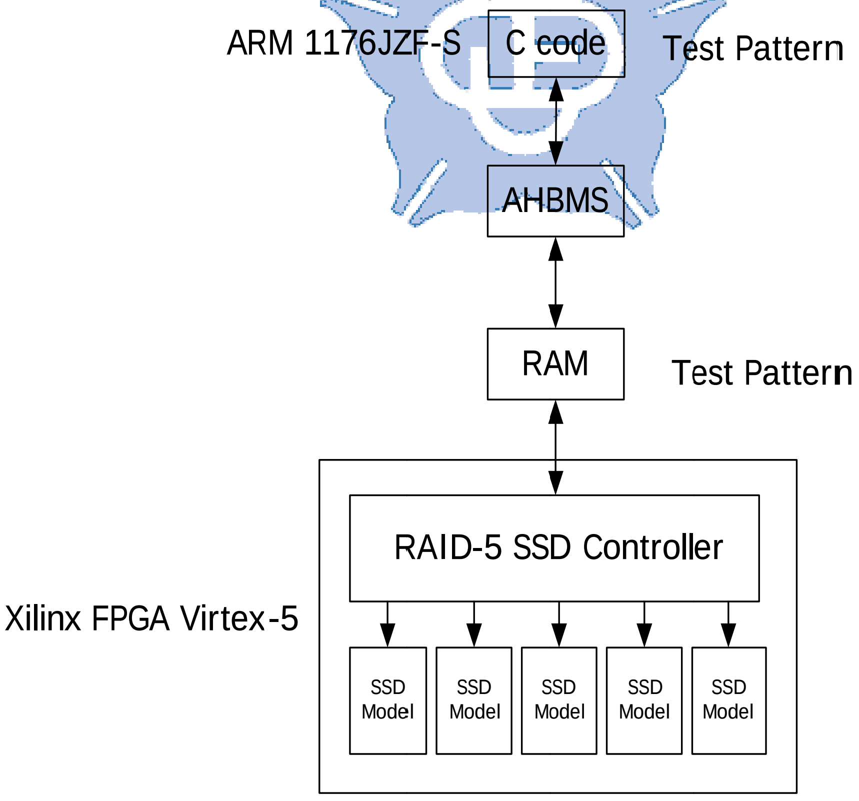


Fig. 4.13 The RAID architecture.

4.4.5 The block diagram of the proposed method

Fig 4.14 shows the block diagram of the proposed efficient buffer management method. The proposed efficient buffer management method is composed of a memory controller, a system controller, a SRAM, a RAID-5 SSD controller, a write cache, a data buffer, and a parity cache. The memory controller handles SRAM address, data, and read/write operation of SRAM. If the value of the wen is 0, it represents that the RAID-5 SSD controller must read data from the SRAM. If the value of the wen is 1, it represents that the RAID-5 SSD controller must write data to the SRAM. Firstly, the addresses of the SRAM (Addr) and data (Din) are written to the SRAM by AHB interface. When the write operation is finished, the RAID-5 SSD controller starts to read data from the SRAM. The RAID-5 SSD controller processes the data to the write cache, data buffer, or parity cache.

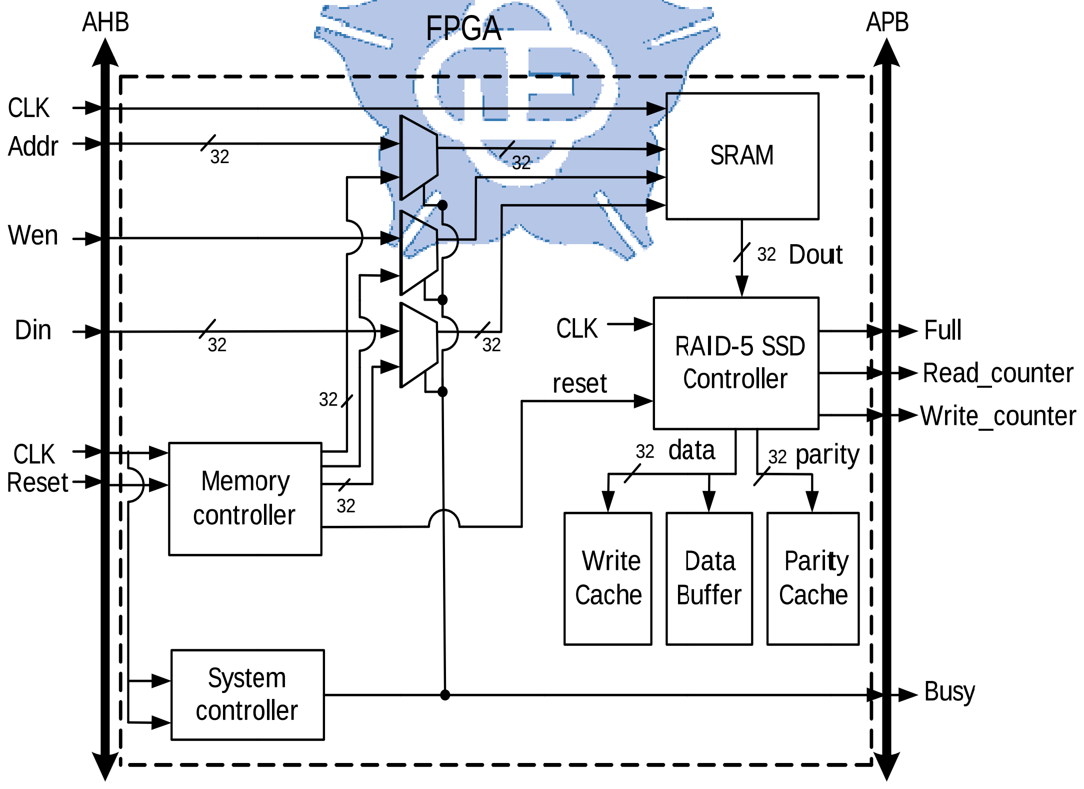


Fig 4.14 The block diagram of the proposed method

4.4.6 **The synthesis report and experimental results with FPGA**

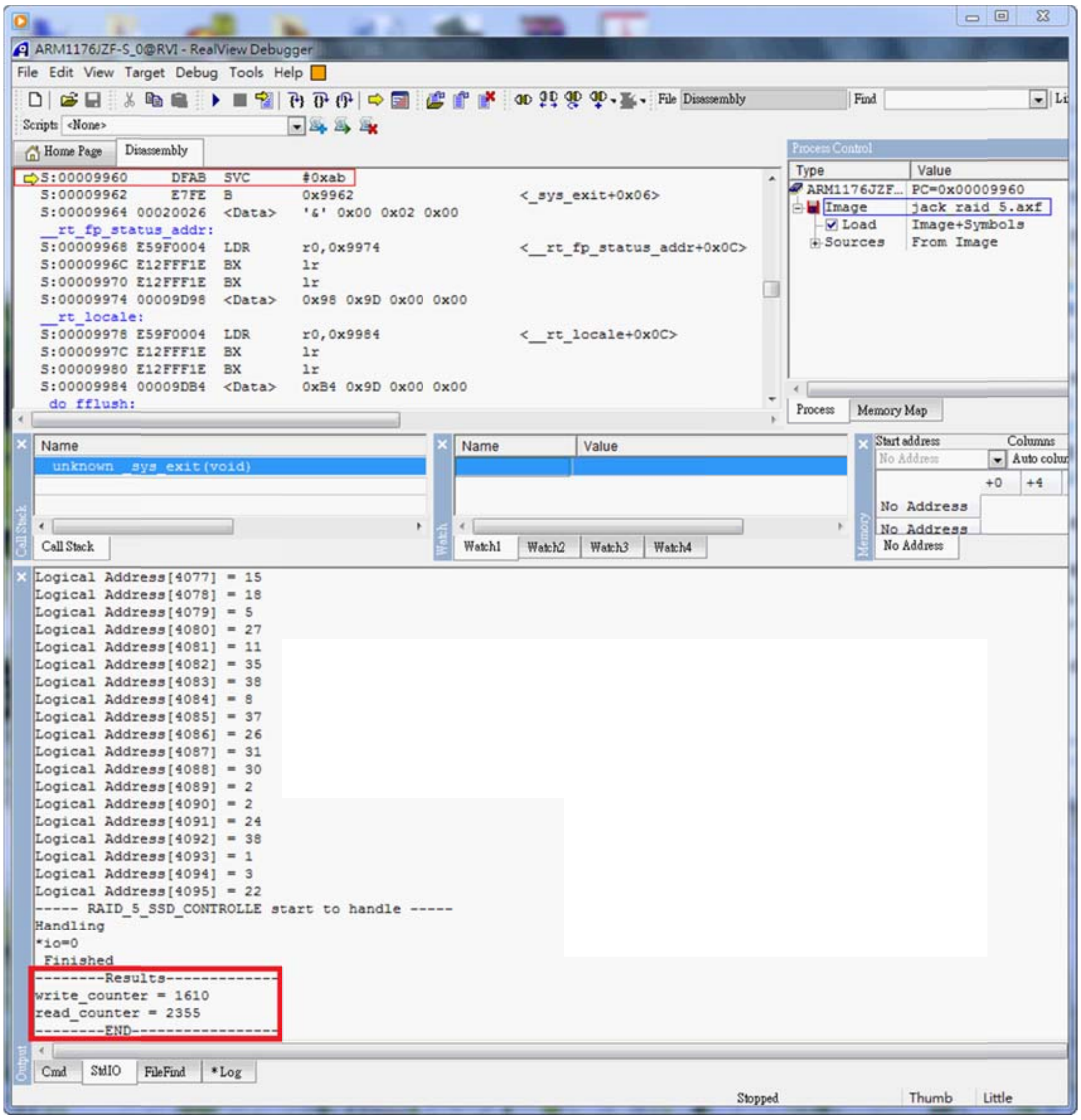
We synthesize the proposed scheme by Xilinx ISE 10.1 [23]. Table 2.3 shows the number of slice registers and lookup table (LUTs) reported by ISE 10.1.

Table 2.3 Number of slice registers and LUTs.

	RAID-5	The proposed method
Number of slice registers	211	390
Number of slice LUTs	588	912

Fig 4.15 The picture of the FPGA

We input the 4096 logical addresses to the FPGA. Fig 4.16 shows the results of RAID-5 with FPGA. The write_counter and read_counter are used to record the write times and read times respectively for full parity write back , partial parity update ,and full parity generation. The number of write_counter and read_counter is 1610 and 2355, respectively.

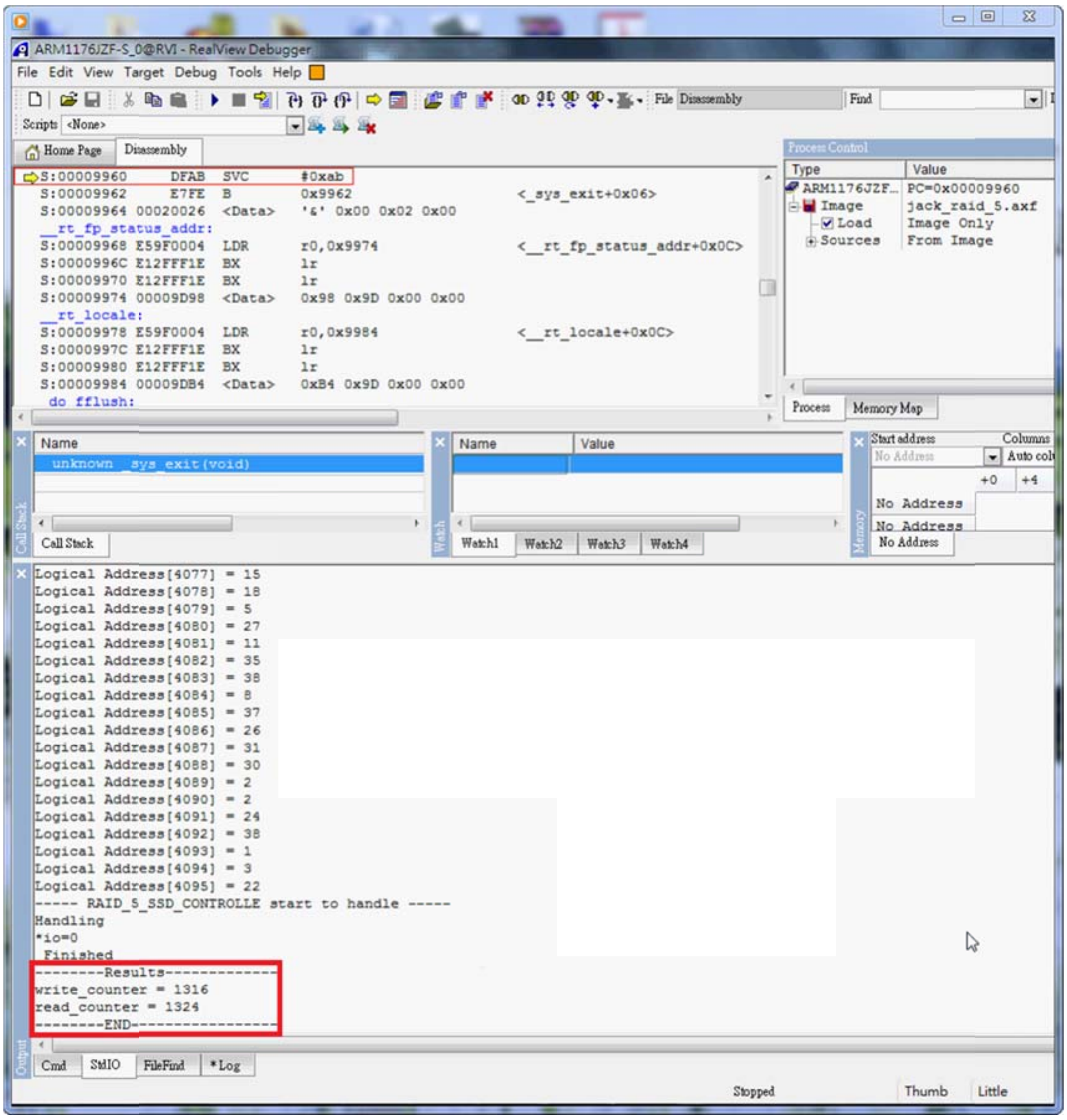


```
S:00009960      DFAB  SVC     #0xab
S:00009962      E7FE  B       0x9962                  <_sys_exit+0x06>
S:00009964 00020026 <Data>  '&' 0x00 0x02 0x00
__rt_fp_status_addr:
S:00009968 E59F0004 LDR     r0,0x9974               <__rt_fp_status_addr+0x0C>
S:0000996C E12FFF1E BX      lr
S:00009970 E12FFF1E BX      lr
S:00009974 00009D98 <Data>  0x98 0x9D 0x00 0x00
__rt_locale:
S:00009978 E59F0004 LDR     r0,0x9984               <__rt_locale+0x0C>
S:0000997C E12FFF1E BX      lr
S:00009980 E12FFF1E BX      lr
S:00009984 00009DB4 <Data>  0xB4 0x9D 0x00 0x00
 do fflush:
```

```
Logical Address[4077] = 15
Logical Address[4078] = 18
Logical Address[4079] = 5
Logical Address[4080] = 27
Logical Address[4081] = 11
Logical Address[4082] = 35
Logical Address[4083] = 38
Logical Address[4084] = 8
Logical Address[4085] = 37
Logical Address[4086] = 26
Logical Address[4087] = 31
Logical Address[4088] = 30
Logical Address[4089] = 2
Logical Address[4090] = 2
Logical Address[4091] = 24
Logical Address[4092] = 38
Logical Address[4093] = 1
Logical Address[4094] = 3
Logical Address[4095] = 22
----- RAID_5_SSD_CONTROLLE start to handle -----
Handling
*io=0
 Finished
--------Results------------
write_counter = 1610
read_counter = 2355
--------END----------------
```

Fig 4.16 The experimental results of RAID-5 with FPGA

Fig 4.17 show the results of the proposed management method with FPGA. The number of write_counter and read_counter is 1316 and 1324, respectively. Both write_counter and read_counter in the proposed management method are smaller than RAID-5.


ARM1176JZF-S_0@RVI - RealView Debugger
File Edit View Target Debug Tools Help
Scripts <None>
Home Page
Disassembly
Process Control
Call Stack
Watch1 Watch2 Watch3 Watch4
Cmd StdIO FileFind *Log
Stopped Thumb Little

```
S:00009960      DFAB  SVC     #0xab
S:00009962      E7FE  B       0x9962                  <_sys_exit+0x06>
S:00009964 00020026  <Data>  '&' 0x00 0x02 0x00
__rt_fp_status_addr:
S:00009968 E59F0004  LDR     r0,0x9974               <__rt_fp_status_addr+0x0C>
S:0000996C E12FFF1E  BX      lr
S:00009970 E12FFF1E  BX      lr
S:00009974 00009D98  <Data>  0x98 0x9D 0x00 0x00
__rt_locale:
S:00009978 E59F0004  LDR     r0,0x9984               <__rt_locale+0x0C>
S:0000997C E12FFF1E  BX      lr
S:00009980 E12FFF1E  BX      lr
S:00009984 00009DB4  <Data>  0xB4 0x9D 0x00 0x00
 do fflush:
```

Type	Value
ARM1176JZF...	PC=0x00009960
Image	jack_raid_5.axf
Load	Image Only
Sources	From Image

```
unknown _sys_exit(void)
```

```
Logical Address[4077] = 15
Logical Address[4078] = 18
Logical Address[4079] = 5
Logical Address[4080] = 27
Logical Address[4081] = 11
Logical Address[4082] = 35
Logical Address[4083] = 38
Logical Address[4084] = 8
Logical Address[4085] = 37
Logical Address[4086] = 26
Logical Address[4087] = 31
Logical Address[4088] = 30
Logical Address[4089] = 2
Logical Address[4090] = 2
Logical Address[4091] = 24
Logical Address[4092] = 38
Logical Address[4093] = 1
Logical Address[4094] = 3
Logical Address[4095] = 22
----- RAID_5_SSD_CONTROLLE start to handle -----
Handling
*io=0
 Finished
--------Results-------------
write_counter = 1316
read_counter = 1324
--------END----------------
```

Fig 4.17 The experimental results of the proposed management method with FPGA

Chapter 5

Conclusion and Future Works

In this thesis, we propose an efficient buffer management method with a data buffer to improve the performance of RAID with SSDs. There are many considerations for disposing RAID with SSDs, because the characteristics of SSDs are different from traditional HDDs. The partial parity cache can merge parity data. The efficient buffer method uses this characteristic sufficiently, so the controller select the suitable victim data to prevent the partial parity cache being full. When the partial parity cache is full, the controller selects the victim partial parity to write back and with read operations for rebuilding the full parity. We also add a data buffer to reduce the full parity generation overhead. The cost of a data buffer is acceptable and the data buffer can reduce the read operations from our experimental results. This proposed scheme decreases both of read operations and write operations for generating the parity data in the RAID system.

The RAID-5 scheme only tolerates one storage device failure. When we need one more tolerances to storage devices failure, we must use more hardware cost in each cache. For example, there are two parities in the RAID-6 scheme, the proposed scheme needs double size of buffers and caches. The hardware cost is very expensive. Thus how to reduce the hardware costs and maintain the performance well are difficult. That can be a further work for a RAID-6 system.

Reference

[1] Ron Ho, Kenneth W. Mai, and Mark A. Horowitz, “The Performance of PC Solid-State Disks (SSDs) as a Function of Bandwidth, Concurrency, Device Architecture, and System Organization,” in *Proc. 36th Int'l Symp. on Computer Architecture (ISCA)*, 2009, pp. 279-289.

[2] Wikipedia, “Ultrabook”. Available: http://en.wikipedia.org/wiki/Ultrabook

[3] Hitachi Global Storage Technologies (GST), “Solid State Drives for Enterprise Data Center Environments,” http://www.hgst.com/tech/techlib.nsf/techdocs/F81A37DF296938BF8625763B00048D41/$file/SSD_techbrief.pdf, 2010.

[4] David Reinsel and Jeff Janukowicz, “Datacenter SSDs : Solid Footing for Growth,” http://www.samsung.com/us/business/semiconductor/news/downloads/210290.pdf, 2008.

[5] Greg Schulz, “Achieving Energy Efficiency Using FLASH SSD,” http://www.cristie.co.uk/uploads/media/Achieving_Energy_Efficiency_Using_SSD.pdf, 2007.

[6] Ahuja, Nishi “Datacenter power savings through high ambient datacenter operation: CFD modeling study,” in *Proc. IEEE Semiconductor Thermal Measurement and Management Symposium (SEMI-THERM)*, March 2012, pp. 104-107.

[7] Chul Lee, Sung Hoon Baek, and Kyo Ho Park “A hybrid flash file system based on NOR and NAND flash memories for embedded devices,” *IEEE Transactions on Computers*, vol. 57, no. 7, pp. 1002-1008, July 2008.

[8] Li-Pin Chang “A hybrid approach to NAND-fash-based solid-state disks,” *IEEE Transactions on Computers*, vol. 59, no. 10, pp. 1337-1349, October 2010.

[9] Hynix Corporation. HY27UK08BGFM NAND flash memories. http://www.hynix.com/inc/pdfDownload.jsp?path=/datasheet/pdf/flash/HY27UK08BGFM%20(Rev0.0).pdf, 2007.

[10] OCZ, "SSD controller," http://www.ocztechnology.com/aboutocz/press/2012/491, 2012.

[11] SandForce, "SF-2000 Family of SSD Processors," http://www.storagesearch.com/sandforce-sf2000-1.pdf, 2010.

[12] Hyojum Kim and Umarkishore Ramachandran "FlashLite: A user-level library to enhance durability of SSD for P2P file sharing," in *Proc. Distributed Computing Systems (ICDCS'09),* June 2009, pp. 534-541.

[13] Nitin Agrawal, Vijayan Prabhakaran, Ted Wobber, John D. Dacis, Mark Manasse and Rina Panigrahy, "Design tradeoffs for SSD performance," in *Proc. USENIX 2008 Annual Technical Confer-ence on Annual conference (ATC'08).* Berkeley, CA, USA: USENIX Association, 2008, pp. 57-70.

[14] Soojun Im and Dongkun Shin "Flash-aware RAID Techniques for dependable and high-performance flash memory SSD," *IEEE Transactions on Computers,* vol. 60, no. 1, pp. 80-92, January 2011.

[15] Kwanghee Park, Dong-Hwan Lee, Youngjoo Woo, Geunhyung Lee, Ju-Hong Lee, and Deok-Hwan Kim "Reliability and performance enhancement technique for SSD array storage system using RAID mechanism," in *Proc. Communications and Information Technology (ISCIT),* September 2009, pp. 140-145.

[16] Asim Kadav, Mahesh Balakrishnan, Vijayan Prabhakaran and Dahlia Malkhi, "Differential RAID: Rethinking RAID for SSD Reliability," in *Proc. Workshop on Hot Topics in Storage and File Systems (HotStorage'09)*, October 2009.

[17] Milo Polte, Jiri Simsa and Garth Gibson, "Enabling enterprise solid state disks

performance," in *Proc. Workshop on Integrating Solid-state Memory into the Storage Hierarchy (WISH'09),* March 2009.

[18] Keven M. Greenan, Darrell D.E. Long, Ethan L. Miller, Thomas J.E. Schwarz, and Avani Wildani "Building flexible, fault-tolerant flash-based storage systems," in *Proc. Hot Topics in System Dependability (HotDep'09),* 2009.

[19] Yangsup Lee, Sanghyuk Jung, and Yong Ho Song "FRA: A flash-aware redundancy Array of flash storage devices," in *Proc. Hardware/Software Codesigns and System Synthesis (CODES+ISSS),* 2009, pp. 163-172.

[20] Bo Mao, Hong Jiang, Dan Feng, Suzhen Wu, Jianxi Chen, Lingfang Zeng, and Lei Tian "HPDA: A hybrid parity-based disk array for enhanced performance and reliability," in *Proc. Parallel & Distributed Processing (IPDPS),* April 2010, pp. 1-12.

[21] Jin Gen and Qing Yang "I-CASH: Intelligently Coupled Array of SSD and HDD," in *Proc. High Performance Computer Architecture (HPCA),* February 2011, pp. 278-289.

[22] Yang Liu, Jianzhong Huang, Changsheng Xie, and Qiang Cao "RAF: A random access first cache management to improve SSD-based disk cache," in *Proc. Networking, Architecture and storage (NAS),* July 2010, pp. 492-500.

[23] Xilinx®, "ISE 10.0". Available: http://www.xilinx.com/itp/xilinx10/books/docs/qst/qst.pdf

[24] ARM Limited., "Application Note 119 - Implementing AHB Peripherals in Logic Tiles". Available: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dai0119e/index.html

[25] Inc. Updated Circuit Systems, "ICS307 Serially Programmable Clock Source". Available: http://www.datasheetcatalog.org/datasheet/icst/ICS307M-02T.pdf

[26] Iozone, IOzone File system Benchmark. http://iozone.org.

[27] J. Katcher. Postmark: a New File System Benchmark. Technical Report TR3022, Network Appliance, October 1997.

[28] Datacenter, . Available: http://www.engineersonline.nl/nieuws/id19922-vijf-valkuilen-voor-datacenters.html

[29] SSD teardown, . Available: http://www.xfastest.com/thread-37656-1-1.html

[30] The small size of a SSD, . Available: http://www.mobile01.com/topicdetail.php?f=481&t=2409231