Effectively Use Solid State Drives in Multi-tiered Storage Systems

A Thesis Presented
by
Cai Wei
to
The Department of Department of Electrical and Computer Engineering
in partial fulfillment of the requirements for the degree of
Master of Science
in
Electrical Engineering
in the field of
Computer Engineering

Northeastern University
Boston, Massachusetts

December, 2012
Contents

Abstract .................................................................................................................................................. 2
1 Introduction ......................................................................................................................................... 4
2 Performance Degradation of SSD ...................................................................................................... 10
  2.1 Background .................................................................................................................................. 10
  2.2 Write amplification, garbage collection and performance degradation ....................................... 12
  2.3 Real measurements of SSD’s performance .................................................................................. 17
    2.3.1 Experimental setup .................................................................................................................. 17
    2.3.2 Experiments and results .......................................................................................................... 19
    2.3.3 Quantitative analysis and theoretical model ............................................................................ 26
3 The algorithms of our multi-tiered storage system .......................................................................... 29
  3.1 Background of the multi-tiered storage system .......................................................................... 29
  3.2 The migration algorithm of our multi-tiered storage system ...................................................... 30
4 Simulation model of multi-tiered storage system ........................................................................... 33
  4.1 Overview of the experiment .......................................................................................................... 33
  4.2 The implementation of our multi-tiered storage system .............................................................. 35
5 Simulation results and analysis ......................................................................................................... 40
  5.1 Simulation results on different migration policies ........................................................................ 40
  5.2 Workload pattern versus $p_0$ ..................................................................................................... 44
  5.3 Design an adaptive management of multi-tiered systems .......................................................... 48
    5.3.1 The significance of the previous experiment ......................................................................... 48
    5.3.2 Endurance degradation and lifetime wearing ....................................................................... 48
    5.3.3 The design of the dynamic multi-tiered system ..................................................................... 50
6 Conclusion and future work .............................................................................................................. 52
7 Acknowledgements .......................................................................................................................... 53
8 Citation ............................................................................................................................................... 54
Abstract

With the fast development of the semiconductor industry, solid-state disk (SSD) has been more and more widely used in storage systems. How to take better advantage of SSD devices in storage systems gradually becomes a hot topic nowadays. Although the price for SSDs decreases in a fast pace and becomes acceptable in a variety of applications, the gap in price per Gigabyte between SSDs and traditional hard drive devices is still tremendous. As a result, multi-tiered storage systems which combine both SSDs and hard drive disks (HDDs) as their storage devices are developed for finding the best balance between cost and performance, i.e., achieving the performance as good as SSDs and meanwhile maintaining comparatively low cost. In order to develop such an efficient multi-tiered storage system, a thorough understanding of both SSDs and HDDs is highly needed. Comparing to traditional hard disks, solid-state disk drives are more complicated with a large number of features and mechanisms that are still unclear. Therefore, in this paper we first study the performance degradation under different I/O access patterns. After observing the phenomenon of non-negligible performance degradation when we have continuous random writes on the SSD, we develop two models: a theoretical model called Pump-Pool model to capture the performance degradation as a function of the SSD occupancy rate, and a simulation model of a multi-tiered storage system in which performance degradation in the SSD tier is precisely described. We finally design a new data migration algorithm which dynamically monitors the application workloads and changes the system’s configuration to achieve better I/O bandwidth in an adaptive way. In additional, we take endurance degrading rate into consideration in our
algorithm, and then we conduct a set of trace-driven simulations to evaluate the new algorithm. The simulation results show that good performance improvement in terms of both speed and endurance is achieved under our new algorithm.
1 Introduction

In the traditional computer architecture, a storage system is the one that develops slowest among all the computer components. This is mainly due to the limitation of mechanical structure and the rotation part in hard disk drive (HDD). Since IBM introduced the IBM 3340 "Winchester" style disk drive which has low mass and low load heads with lubricated platters in 1973, this structure and technology of HDD still remains through nowadays [1]. Low speed and high latency caused by the mechanical moving parts have made disk-based storage systems to be the bottle neck of the whole system. The advent of the NAND-flash based solid-state storage device (SSD) is certain to represent a sea change in the architecture of computer storage subsystems[2]. SSDs brings tremendous I/O bandwidths, not only providing excellent performance of continuous reads and writes, but also allowing random access several orders of magnitude faster than HDDs which is more crucial to transactional database systems and batch processing, query or decision support analysis [1, 3, 4].

Though the SSD brought tons of advantages as a storage media, two defects have become the main reasons impeding the widely usage of SSDs in storage systems: cost and reliability. Flash costs are falling fast, mostly due to the reduction in semiconductor feature sizes. Price per Megabyte fell 56 percent from 2004 to 2005, and 47 percent more from 2005 to 2007 [5]. Today, despite the fast falling speed, the price of SSD is still 18 times more than that of HDDs[6]. On the other hand, with the respect of reliability hard drives can stand for millions of read and write operations while an SLC flash chip's endurance level is 100,000 write/erase cycles per cell and
the endurance level for an MLC chip is even worse which could be as low as 10,000[5].

![Figure 1.1 Average HDD and SSD prices in USD per gigabyte.](image)

As a compromise solution for the trade-off between cost and performance, the multi-tiered system is developed and this system uses both SSDs and HDDs as storage medias. In general, there are two methods to integrate flash devices and hard drives into one storage system. The first method employs flash drive as the cache of the entire storage system. Developers of NetApp’s Flash Cache (FC), which has this kind of implementation, claim that they gain better performance with FC drives and better capacity with SATA drives with no additional budget. The other method chooses to integrate SSDs into tiered storage architectures as the fast persistent storage. In this mode system, controller uses sophisticated algorithms to migrate data between layers based on access demand relative to other chunks in the same system [8]. The efficiency of this algorithm decides the performance of the entire system considerably. In this paper, we focus on the second way for a multi-tiered system implementation.

Different from the usual usage of SSD storage devices in personal desktops and laptops, database and other complicated systems which might adopt a multi-tiered hybrid storage structure, have higher requirements on random IOPS (I/O per second)
access [9]. Performance evaluation on solid-state drive’s random access has been well discussed. Most suppliers of SSD products provide information on how much IOPS their products could achieve. However, this information is insufficient for users to get the full picture of random write performance in SSD. Limited by its own complex organization and inherent defects, flash-based solid device cannot sustain high speed under heavy random write load [10, 23]. This problem of random write performance degradation in SSD is barely mentioned and discussed both in research papers and manuals of commercial products. Comparing with magnetic hard drive which just refreshes each unit and records new data, SSD is bothered with the issue of sophisticated erasure. As a result, it might take much longer time for SSD to execute simple rewrite tasks which leads to latency increase and performance degradation. In this paper, we use a set of tools and benchmarks to stress and test the performance of SSDs under various situations in a real system. The experimental results show non-negligible performance degradation if there are continuing random write operations. Therefore, we propose a Pump-Pool model to describe such performance degradation phenomena. In this model we use the capacity of the pool to represent the duration time length of high-speed random writes before degradation.

In the second section of this paper, we present simulation model of a multi-tiered storage system based on our Pump-Pool model. Given the limited capacity of SSD tier, this multi-tiered system keeps track of the address and time for each access, and periodically calculates the temperature (i.e. the possibility one particular location will be accessed at the next moment) for each location. To save storage space in the SSD tier, cold data (i.e., frequently accessed) in SSD tier would be moved to HDD while hot data that stored in HDD would be moved to the SSD tier. In our multi-tiered
storage system there are four tunable parameters: preset SSD occupancy rate $\rho_0$, migration size, frequency of migration and threshold temperature for migration. Once the system decides to trigger a migration job, it automatically scans and collects the information such as the present disk occupancy, the location and temperature of the hot and cold data in both of the two tiers. Then these information and values are compared with the preset parameters of SSD occupancy rate $\rho_0$ and threshold temperature, then the system decides what data need to be transferred to the other tier.

Traces with different parameters including average arrival rate, coefficient of variation (CV) and autocorrelation (ACF) are generated as input for the multi-tiered system simulating the I/O workloads in the real environment. The system uses these traces to generate a sequence with arrival and service time for each job including reading, writing and migration. In another program, a single server single queue model uses this sequence as input and calculate the delay and processing time for each job.

In the third part, we analyze all the simulation results of the experiment and compare the performance of different migration configurations in diverse workloads. Under different types of workloads, systems with different configurations may perform differently. Through these results, we conclude that the performance degradation due to random writes in SSD may affect the performance of a migration policy, and hence it should be considered in the algorithm design and the parameter setting. Despite of performance in bandwidth and latency, we also discuss the issue of endurance and wearing at the SSD tier. We attempt to find the optimal configuration under different patterns of workloads, aiming at providing the fastest IO processing speed without increasing write amplification ratio and costing too much lifetime of the flash memory.
in the SSD. In our final model, the parameters of the configuration are not fixed, and they are dynamically changed based on the workload this system is processing. With the module that tracks and analyzes the parameters of IO requests, the system recognizes the type of the current workload and adaptively updates its own parameters. At the end of this section, we give the performance of this final model and compare it with the systems that have fixed configurations. From this comparison we conclude that the adaptive system achieves both performance and endurance in the changing environment.

The key contributions of this work are the following. First we propose the problem of performance degradation in the SSD devices and use the Pump-Pool model to precisely describe it. Second, we introduce the migration algorithms in our multi-tiered storage system which dynamically keeps the occupancy rate in the SSD tier in fixed level. Based on this migration algorithm, the performance of systems with different configurations and workloads is given. Third, with the previous experiment results, we set up the storage system in an adaptive approach which adaptively changes its parameters according to the current workload pattern. Good performance in both speed and endurance is achieved under changing workload.

The remainder of the paper is organized as follows. Section 2 provides a survey of the performance degradation of the solid-state drive. In section 3 the migration algorithm that is used in our multi-tiered storage system is given. Section 4 presents the simulation model for our multi-tiered storage using the migration algorithm described in section 3. As demonstrated in Section 5 the system with different configurations
and workloads has different performance, and a dynamic version of this system and its performance is shown in this section. Finally, we conclude in Section 6.
2 Performance Degradation of SSD

2.1 Background

A solid-state drive (SSD) is a data storage device that uses integrated circuit assemblies as memory to store data persistently [11]. SSD plays the role exactly the same as traditional rotational hard drive disk and shares the same input and output interface with it. Distinguished from hard drive disk which has rapidly rotating discs/platters and magnetic heads arranged on a moving actuator arm, solid-state drive does not employ any mechanical components and moving parts. Benefited from that SSD does not bothered by the limitation of mechanical structure and has better performance in many ways. As of nowadays there are three types of SSD classified by storage media, NAND-based flash memory, NOR-based flash memory and random-access memory (RAM). Most SSDs use the former one and in narrow terms the phrase SSD refers to NAND-based solid drive disk and in this paper we only discuss NAND-based SSD. The NAND flash architecture was introduced by Toshiba in 1989 and now has become the mainstream of non-volatile memory market.

![Figure 2.1 Section-view of a flash memory cell structure on silicon](image)
Physically, the flash memory stores information in an array of memory cells made from floating-gate transistors and each of the memory cells has the similar structure like MOSFET except that the NAND flash has one more gate than MOSFET (shown in Figure 2.1 and 2.2) [13, 14]. This extra gate, called Float Gate, is the place where data are stored [12]. Traditionally each basic unit of flash memory shown in Figure 2.1 stores only one bit of information. Thus, this kind of structure is called SLC. By contrast, multi-level cell (known as MLC) devices stored more data (typically 2 bit) in each cell by the technology of precise charge placement, precise charge sensing, and precise charge retention [15, 27]. In each flash memory chip these basic units and memory cells are organized in the structure called pages and blocks at different levels. Figure 2.3 shows an example of an array organization for memory chip MT29F2G08AxB by MICRON [16]. For normal reading and writing (usually called programming), each I/O request is operated in page level, i.e. a page is the smallest size for reading and programming. Although it can be read or programmed one page
at a time in a random access fashion, it can only be erased a "block" at a time. With limited erase endurance, this mismatch between read size and erase size becomes the major defect of flash memory.

![Array Organization for MT29F2G08AxB by Micron](image)

Figure 2.3 Array Organization for MT29F2G08AxB by Micron [16].

2.2 Write amplification, garbage collection and performance degradation

As we mentioned in Section 2.1, the basic unit for rewrite operation on SSD is a block. Although one can read or program the data of one byte through a random access, the data for erasing or reprogramming cannot be smaller than one block which may contain 64, 128 or more pages (depends on the flash memory model). This huge gap leads to critical issues including write amplification, garbage collection and performance degradation.
Given that the basic rewrite unit should be much larger than a page, additional process is introduced when we need to modify a particular page. In Figure 2.4, we provide an example of such case where we have one block that is composed of five pages. The second and the third pages contain the saved data while the fourth and the fifth pages keep empty. The data which was stored in the first page is deleted and marked as invalid with a red cross. When we have a new write request which requires to store the new data the with the size of three pages, instead of directly modifying the data stored in the first page, we first need to read and copy the information in all pages to the cache which is integrated in the SSD itself. Once in the cache, those invalid data will be thrown away while new data set will be merged with the valid data stored in the third and the fourth pages. Following that, all data in that block can be erased safely and all data in the cache will then be written to a new empty block. In this example, three pages of new data that need to be added to the storage block cause five page writes on this block and a long read-to-cache and write-to-block process, which
leads to an actual amount of writes about twice of what the user requests. This ratio of data written to the flash memory over data written by the host is called write amplification [17].

Figure 2.5 Page remapping to delay the erase-before-write cycle

To delay the appearance of such a long erase-before-write cycle, SSD employs wear-leveling technique which can prolong the service life of its NAND storage media. Similar to the write request in Figure 2.4, Figure 2.5 shows that instead of erasing the whole block the controller of SSD remaps the first page to another empty page. In SSDs the mapping information is recorded in a mapping table which is managed by the flash translation layer (FTL). The FLT translates the logical address to physical one and vice versa [18]. There are three ways to design the FTL, i.e., page-level and block-level and hybrid FTL. In a page-level FTL, each logical page is mapped to a corresponding physical page and all these precise and complicated information is stored and modified in a build-in SRAM at the SSD. Comparing with that a block-level FTL uses an approach similar to the set-associative style design of CPU cache, where the mapping relationship only exists at the block level which requires storage
space much less than the page-level scheme [19]. For example, given a 4GB flash
memory with 4KB page size and 256 pages per block the page-level scheme requires
about 1MB to store the mapping table whereas the block-level FTL only needs 4KB.

In summary, the block-level scheme has the following two disadvantages. First, the
issue related to erase-before-write cycle has not been solved, thus write access would
be slowed down easily. Second, this management scheme leads to low space
utilization. Hybrid-level FTL combines the characteristics of these two FTL schemes
to achieve high management productivity at comparatively low cost in space [21].

Hybrid-level FTL adds another structure, called log blocks, to the block-level style
scheme to attain similar performance as the page-level FTL. Figure 2.6 illustrates a
block-level mapping table which maps each local block address (LBN) to a physical
block address (PBN). Initially, two data sets (i.e., pages 4 and 5) are written in data
block 10. Then, the incoming rewritten requests impel the page-level mapping table to
find an empty log block to record rewritten data. For each read or write operation, the
hybrid-level FTL first checks the page-level mapping table (shown in Figure2.6). If
the LBN address for this access operation is recorded in this table, the FLT accesses
the data in the log block area. Otherwise it directly accesses the data in the log block
area.
With limited size of space, the empty pages in the log block area are written and marked as invalid (also called stale pages) in no time. Thus the log block area would quickly run out of clean pages. These invalid pages can be recycled and cleaned by the process of garbage collection. In this process, the controller of SSD reads and rewrites only valid pages in the old block into another previously erased empty block, then the old block left by previous operation is erased [20][21]. In this way FTL removes invalid data and provide new clean block for log-block area.

The garbage collection process could be much slower than the device’s maximum throughput. Since some SSDs implement garbage collection only in idle-time (called background garbage collection or idle garbage collection), these devices are more likely bothered with running out of clean pages when the system load is heavy. The situation would be probably worse in some systems that use RAID disk array or have
operation systems other than Windows 7 or Linux since 2.6.33 [28]. In these two cases the TRIM command, which enables the operating system to tell an SSD what blocks of previously saved data are no longer needed, is not supported. In these situations, the long read-modify-write cycle can no longer be avoided and the write (especially small size random write) performance would be degraded significantly.

2.3 Real measurements of SSD's performance

2.3.1 Experimental setup

Currently, the storage market is experiencing an enormous expansion and surge leaded by solid state device. Based on the constantly evolving SSD techniques, hundreds of large well-known storage vendors and small unknown start-ups provide a variety of products and systems ranging from consumer electrons to the high-end server applications. The performance of these products varies in a large scale because of their different memory chips, hardware controllers and management policies that are integrated in the firmware.

In our experiments, a 120GB Intel-520 solid-state drive is used to evaluate the performance. In details, this SSD uses a SF-2281 chip which is provided by SandForce as the controller. On the PCB board of this SSD, there are 8 NAND memory chips with the model of 29F16B08CCME2. Each chip provides 16GB of memory space, so the capacity of this SSD is 128GB plus 8GB that is allocated as an over-provisioning space and is invisible to regular users and the operation system [22]. The specifications for the NAND chip 29F16B08CCME2 is given in Table 2.1.
We run our experiments in a Dell OptiPlex 760 desktop which is equipped with an Intel Core 2 Duo E7400 processor (2.8GHz, 3M L2Cache, 1066FSB) and 4GB DDR2 Non-ECC SDRAM running at 800MHz. To avoid being a performance bottleneck, we use another Intel 520 SSD to hold the operation system which is Ubuntu 12.04.1 LTS with Linux v3.2.14 kernel and an Ext4 file system. Since SSD has a complicated management policy, we choose the NOOP I/O scheduler to eliminate the impact of some environmental variables. The NOOP scheduler bypasses the buffering whereas in most cases buffering is undesirable due to two reasons. First, the SSD devices benefit little from buffering as it has much lower latency than HDDs. Second, the read buffering step consumes more CPU cycles for memory copy operations whereas read I/Os in the SSD drive are more sensitive to CPU resource consumption than those in the HDD drive[23].

In our experiments, we use Iometer as the benchmark to measure the performance of our SSD drive. The Iometer benchmark provides a variety of options in order to make it suitable for testing drives in the web, mail and database servers which require more random read/write IOPs rather than sequential I/Os. In Linux, an executable file, named Dynamo reports the results in command lines. However, Dynamo has no GUI
interface. Therefore, we use another Windows host (shown in Figure 2.7) where the GUI interface for Iometer is provided. Commands from this Windows host are sent to test our SSD device on the Linux host via the wired LAN (Local Area Network). In the same way, the testing results are sent back to the host in Windows.

![Figure 2.7 Iometer GUI on Windows](image)

### 2.3.2 Experiments and results

Before running the benchmark on our SSD drive, we first fill it with different types of files to simulate the usage of SSD in a server storage array. Part of the drive is filled with large media files, and the remaining part is filled with files which contain random content and have size much smaller than the block size (2MB).

<table>
<thead>
<tr>
<th>Time</th>
<th>0</th>
<th>5</th>
<th>10</th>
<th>15</th>
<th>20</th>
<th>25</th>
<th>30</th>
<th>35</th>
<th>40</th>
<th>45</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bandwidth MB/s</td>
<td>115.2</td>
<td>80.1</td>
<td>79.00</td>
<td>80.10</td>
<td>79.76</td>
<td>79.76</td>
<td>79.82</td>
<td>78.33</td>
<td>78.57</td>
<td>78.49</td>
</tr>
<tr>
<td>IOPS</td>
<td>28800</td>
<td>20025</td>
<td>19750</td>
<td>20025</td>
<td>19940</td>
<td>19940</td>
<td>19955</td>
<td>19582</td>
<td>19642</td>
<td>19622</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>50</th>
<th>55</th>
<th>60</th>
<th>65</th>
<th>70</th>
<th>75</th>
<th>80</th>
<th>85</th>
<th>90</th>
<th>3600</th>
<th>4800</th>
</tr>
</thead>
<tbody>
<tr>
<td>78.30</td>
<td>78.59</td>
<td>78.52</td>
<td>78.48</td>
<td>78.35</td>
<td>78.39</td>
<td>78.15</td>
<td>78.20</td>
<td>78.24</td>
<td>78.31</td>
<td>78.25</td>
</tr>
<tr>
<td>19575</td>
<td>19647</td>
<td>19630</td>
<td>19620</td>
<td>19587</td>
<td>19597</td>
<td>19537</td>
<td>19550</td>
<td>19560</td>
<td>19578</td>
<td>19563</td>
</tr>
</tbody>
</table>
The first experiment is to test the performance of the SSD drive in the situation when its occupancy is almost 1. From the results shown in Figure 2.8 we can clearly observe that the random write performance drops significantly in quite short period of time. Another noteworthy fact is that after the quick drop at the beginning the performance keeps stable during the next one and a half hour. When the occupancy of a SSD drive is high, the spare space which is related to the log block area and garbage collection mentioned in section 2.2 is so limited that the performance for the SSD drive is approaching its minimum value. The low efficiency also means higher write amplification ratio and faster wear speed on its endurance [24]. Even in this worst case, the random write speed of the drive still maintains its value around 78 MB/s for a considerable long period of time, and this implies the fact the erase-before-write cycle is fast enough to provide new clean blocks at this rate.

<table>
<thead>
<tr>
<th>Time(sec)</th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
<th>50</th>
<th>60</th>
<th>70</th>
<th>80</th>
<th>90</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bandwidth MB/s</td>
<td>115.4</td>
<td>120</td>
<td>84.28</td>
<td>81.13</td>
<td>74.49</td>
<td>74.55</td>
<td>78.35</td>
<td>74.88</td>
<td>75.7</td>
<td>74.7</td>
<td>73.84</td>
</tr>
</tbody>
</table>
IOPS | 2885 | 30000 | 21070 | 20282 | 18622 | 18637 | 19587 | 18720 | 18925 | 18675 | 18460

Table 2.3 Random write performance with 80% drive occupancy (20% free)

Figure 2.9 Random write performance with 80% drive occupancy (20% free)

<table>
<thead>
<tr>
<th>Time(sec)</th>
<th>0</th>
<th>5</th>
<th>10</th>
<th>15</th>
<th>20</th>
<th>25</th>
<th>30</th>
<th>35</th>
<th>40</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bandwidth MB/s</td>
<td>99.98</td>
<td>102.25</td>
<td>97.51</td>
<td>94.85</td>
<td>96.83</td>
<td>83.01</td>
<td>82.02</td>
<td>81.8</td>
<td>81.78</td>
</tr>
<tr>
<td>IOPS</td>
<td>24745</td>
<td>22062</td>
<td>19377</td>
<td>21212</td>
<td>20707</td>
<td>20502</td>
<td>20505</td>
<td>20450</td>
<td>20445</td>
</tr>
</tbody>
</table>

Table 2.4: Random write performance with 60% drive occupancy (40% free)
Figure 2.10 Random write performance with 60% drive occupancy (40% free)

<table>
<thead>
<tr>
<th>Time (sec)</th>
<th>5</th>
<th>10</th>
<th>15</th>
<th>20</th>
<th>25</th>
<th>30</th>
<th>35</th>
<th>40</th>
<th>45</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bandwidth MB/s</td>
<td>112</td>
<td>110.71</td>
<td>118.49</td>
<td>126.18</td>
<td>125.72</td>
<td>117.48</td>
<td>115.87</td>
<td>113.15</td>
<td>101.7</td>
</tr>
<tr>
<td>IOPS</td>
<td>28000</td>
<td>27677.5</td>
<td>29623</td>
<td>31545</td>
<td>32680</td>
<td>29370</td>
<td>28967</td>
<td>28287</td>
<td>25425</td>
</tr>
</tbody>
</table>

Table 2.5 Random write performance with 40% drive occupancy (60% free)

<table>
<thead>
<tr>
<th>50</th>
<th>55</th>
<th>60</th>
<th>65</th>
<th>70</th>
<th>75</th>
<th>80</th>
<th>85</th>
</tr>
</thead>
<tbody>
<tr>
<td>79.96</td>
<td>78.27</td>
<td>80.55</td>
<td>79.25</td>
<td>80.31</td>
<td>81.54</td>
<td>80.28</td>
<td>79.73</td>
</tr>
<tr>
<td>19990</td>
<td>19567.5</td>
<td>20137.5</td>
<td>19812.5</td>
<td>20077.5</td>
<td>20385</td>
<td>20070</td>
<td>19932.5</td>
</tr>
</tbody>
</table>

Figure 2.11 Random write performance with 40% drive occupancy (60% free)

Similar to the first test, Figure 2.9, 2.10 and 2.11 show the different results of random write performance as the SSD occupancy rate varies. Thirty minutes interval time between different tests is given to ensure that the result of one test would not be interfered by the others. In general, all these four groups of the test have the same trend: in the beginning the I/O bandwidth is as high as 110MB/s or 27500 IOPs (transfer request size equals to 4KB), and then at some point drops to around 80MB/s or 20000 IOPs. The difference is that, as idle space ratio increases (from less than 1%
to 60%), the drop point delays and the SSD can maintain in high speed for a longer time. In addition to that we also use the ATA Security Erase Unit command to set the SSD drive to its cleanest “out-of-the-box” state by wiping out all fragmentation and erase and physically erasing all NAND blocks [23]. The Figure 2.12 shows the test result (note that the unit for time label is minute). Comparing with the previous experiment, the “completely clean state” provides much longer time in high-bandwidth period which is approximately 7 minutes/350 seconds.

<table>
<thead>
<tr>
<th>Time (min)</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bandwidth MB/s</td>
<td>130.86</td>
<td>131.46</td>
<td>132.3</td>
<td>128.32</td>
<td>132.84</td>
<td>132.64</td>
<td>130.57</td>
<td>89.25</td>
</tr>
<tr>
<td>IOPS</td>
<td>32715</td>
<td>32865</td>
<td>33075</td>
<td>32080</td>
<td>33210</td>
<td>33160</td>
<td>32642</td>
<td>22312</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>82.85</td>
<td>82.31</td>
<td>81.8</td>
<td>83.24</td>
<td>81.54</td>
<td>80.02</td>
<td>82.35</td>
<td>82.48</td>
</tr>
<tr>
<td>20712.5</td>
<td>20577.5</td>
<td>20450</td>
<td>20810</td>
<td>20385</td>
<td>20005</td>
<td>20587.5</td>
<td>20620</td>
</tr>
</tbody>
</table>

Table 2.6 Random write performance after implementing security erase (completely clean state).

Comparing with random write access, in theory sequence write does not have much influence on garbage collection and the erase-before-write activities, which refers that it is not bothered with the performance degradation issue. To prove that we test the
sequential write performance of our SSD drive over a long period of time. At the beginnig 5 minutes of the test, it seems that the SSD drive has not fully recognized the sequential access pattern and data locality, and thus leads to performance fluctuation. Later in the next one hour the SSD drive entered the stationary phase with stable speed around 200MB/s. From this long span test we can derive the conclusion that the sequential write access is not affected by the performance degradation.

The previous experiments provide a general view of performance degradation problem. Even in the stable period after degradation, we still captures some abnormal phenomena in a microscopic view. Figure 2.13 shows a period of waveform of a random write test on a fully occupied SSD after performance degradation. In this experiment the sampling frequency is set to 1 Hz per second to acquire more detailed information. In figure 2.13 we can easily observe troughs that periodical appear (labeled with read arrows). Since the drive is in the worst status where garbage collection is frequently triggered and such behavior can strongly affect the I/O bandwidth of SSD, these troughs are very likely the symbol of garbage collection activity. To explore this issue further, here we use MATLAB to make a Fast Fourier Transform (FFT) analysis on the waveform in Figure 2.13, and Figure 2.14 shows the
FFT result. In this figure the first peak occurs at 0.04572 Hz indicates the 21.87 second repeat period of the waveform in Figure 2.13. Considering the average bandwidth is about 80MB/s, in each cycle the total traffic of I/O is 80 * 21.87 = 1744MB. We are not clear with the meaning of this value and further research in low level is needed to reveal the truth behind it.

![Figure 2.13 Result of a long running random write test on an almost full SSD drive (ρ > 99%). Troughs that periodical appear are labeled with read arrows.](image1)

![Figure 2.14 The FFT spectrum of Figure 2.13 with first peak value of 0.04572 Hz. A higher pick around 0.3309 Hz is the 3 sec period which is the most significant one.](image2)
2.3.3 Quantitative analysis and theoretical model

The result shown in the previous section indicates that the free space ratio of the SSD is the key parameter for how long it can sustain before performance degradation. Though we did not examine the precise activities inside the SSD drive as they are invisible to host and operation system, existing information on this drive is enough for explaining the experiment result and establishing simulation model. Multiple sources show that the SandForce SF-2281 controller used in our Intel 520 SSD drive employs aggressive foreground garbage collection policy. In this algorithm SSD does not only do garbage collection during idle time but also clears and moves data as needed (even when this drive is processing I/O requests including random write access) [25].

Based on all information above we set up our simulation model called “Pump-Pool Model” to describe the random write performance degradation. Figure 2.15 presents the abstract parameters used in our model. In this model water is continuously pumped into the pool at a fixed rate $a_1$ and the water in this pool leaks out from the valve located at the bottom at the rate of $a_2$. When the valve is closed (i.e. $a_2$ equals to zero), the water in this pool keeps rising before reaching its maximum capacity. This upper bound of the pool capacity $f(\rho)$ is variable and it is a function of the SSD occupancy rate $\rho$. 

In this model the water in this pool represents the clean and empty blocks in the SSD. The garbage collection activity collects invalid pages and turns them into new clean blocks at rate $a_1$. The random write access with request size much smaller than the block size can produce dirty pages and blocks at a rate faster than $a_1$ (denoted by $a_2$). Depending on the garbage collection type, the pump may keep working constantly or sometimes be shut down in the valve at the bottom is not closed. In the previous paragraph we have mentioned that our testing device uses foreground garbage collection mechanism which collects and cleans dirty blocks even when the drive is busy. This indicates that the pump in this model still pumps water to the pool when the valve at the bottom is open. Given that at the time point $t_d$ the I/O bandwidth decreases to minimal value, the calculation formula for the up bound of the pool capacity is:

$$f(\rho) = \int_{t_0}^{t_d} (a_2 - a_1) \, dt \quad (2.1)$$

In Figure 2.16 all curves of previous results is split into two parts by the time the performance decreases to 80 MB/s (the purple vertical line). Then we calculate the green area which indicates the maximal amount of clean blocks and acquire the result.
shown in Figure 2.17. By this way we get a rough estimation of functional relations between disk occupancy $\rho$ and the maximum capacity $f(\rho)$. In the next section we use this model and the function to establish the simulation mode of multi-tiered storage system.
3 The algorithms of our multi-tiered storage system

In section 2 we discuss the performance degradation in random write access on solid-state drive in a quantitative approach. In this section we present our new algorithms of a multi-tiered storage system. In this model the previous discussion on performance degradation is taken into consideration.

3.1 Background of the multi-tiered storage system

As data continues to grow at an exponential rate, managing data growth has become one of the biggest challenges for IT. The demands for capacity continue to grow exponentially while the requirement on high the performance applies extra pressure on the budget of the system. The rise of solid-state drives in enterprise data storage arrays in recent years provides a better solution on performance. On the other hand, the fact behind the explosive increase of data is that quite few chance these data would be accessed or modified and only few of them are frequently needed. In fact, the vast majority of the data become inactive and are accessed as little as one month a time. Some research based on the theory of temporal locality and Pareto principle implies that in server 70-80% of all stored data is inactive, and these data rarely accessed after 39-90 days [26].

A number of leading storage vendors are addressing these issues with tiered storage integrated using different classes of storage to match different performance needs. The traditional way of designing the multi-tiered storage system which using data inventory and manual time scheduler suffers from several limits including stiff and closed structure implementation, lacking intelligent management and unable to adapt
itself to the fluctuating workload. The automated storage tiering (AST) system is introduced aiming at solving the balance problem between cost and performance in a more intuitive way. In this section we present a multi-tiered storage system in which data are automatically allocated and the performance degradation is not ignored in the SSD tier.

### 3.2 The migration algorithm of our multi-tiered storage system

In the conventional view the larger space of the faster tier in a multi-tiered structure always means better performance. For example, the high-end CPU products of Intel normally have larger Level 2 cache than its cheap products, and this is a strong indicator of their performance. Similarly, increasing the number of SSD devices is a common approach to improve the performance of a multi-tiered storage system.

When given the fact that the hardware and capacity of the storage system is fixed, one thing the manager of the system should concern is that taking fully use of the fast SSD tier helps to achieve better performance. Keeping the most of the space in the expensive SSD device in empty status obviously is not a wise choice for good performance.

In the meantime, keeping the SSD tier full does not directly leads to the optimal performance of the system. In the previous sections of this thesis we have discussed and analyzed the performance degradation issue, which is closely related to the occupancy rate of the SSD drive (denoted by $\rho$). The high occupancy rate $\rho$ leads to earlier occurrence of the performance degradation problem. In our migration algorithm the occupancy rate of the SSD tier is the key parameter and the system keeps the occupancy rate $\rho$ around a predefined threshold $\rho_0$. 
When the present SSD occupancy rate $\rho$ is close to the threshold $\rho_0$, once the system decides to trigger a migration job, it will execute the following operations (shown in ). First, the data in the SSD and HDD tiers are sorted by their temperature. Then the coldest data in the SSD tier are moved to the HDD tier and the same amount of data are migrated from the HDD to the SSD. The same amount of migration in two direction results in stable space occupancy in SSD.

![Diagram](image)

**Figure 3.1** The migration algorithm of our multi-tiered storage system when $\rho = \rho_0$

In some other cases, it happens that a large amount of new data are written into the SSD tier which causes the space occupancy rate $\rho$ is higher than the threshold value $\rho_0$. In Figure 3.2 the migration policy for this situation is provided. Unlike the migration in Figure 3.1, here the data moved to HDD is more than the data moved in the opposite direction. In contrast, when the data in the SSD tier is deleted and $\rho$ is lower than $\rho_0$, the system will migrate more data from the HDD to the SSD. By implementing these operations the system dynamically changes its migration policy to keep the SSD’s space occupancy rate close to the predefined value.
Figure 3.2 The migration algorithm of our multi-tiered storage system when $\rho > \rho_0$ and $\rho < \rho_0$. 
4 Simulation model of multi-tiered storage system

In the previous section we introduce the basic algorithm of our multi-tiered storage system. In this section, we will give more details which mainly focuses on how we implement our migration algorithms and establish the simulation model of this multi-tiered storage system. The notation used for the purpose of our analysis is given in Table 4.1. At the end of this section we also provide the pseudo code of this simulation model in Table 4.2.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>A fixed value to indicate whether the system is idle</td>
</tr>
<tr>
<td>t_n</td>
<td>The inter-arrival time of the nth incoming job</td>
</tr>
<tr>
<td>add_n</td>
<td>The logical address of the nth incoming job</td>
</tr>
<tr>
<td>N</td>
<td>For every N incoming jobs the system forces to insert migration job</td>
</tr>
<tr>
<td>p</td>
<td>Current SSD occupancy rate</td>
</tr>
<tr>
<td>p_0</td>
<td>Preset SSD occupancy utilization rate</td>
</tr>
<tr>
<td>C</td>
<td>Number of clean blocks in SSD drive</td>
</tr>
<tr>
<td>C_max</td>
<td>Maximal value of clean blocks which is a function of p</td>
</tr>
<tr>
<td>SSD_WH, SSD_WL</td>
<td>Processing time for a write access on SSD before/after performance degradation</td>
</tr>
<tr>
<td>HDD_W, SSD_R, HDD_R</td>
<td>Processing time for jobs that are: writes on HDD, reads on SSD, and read on HDD</td>
</tr>
<tr>
<td>Migration_size</td>
<td>Number of blocks transferred in each migration job</td>
</tr>
<tr>
<td>Migration_time</td>
<td>Processing time for a migration job triggered inside storage system. It is a function of Migration_size</td>
</tr>
</tbody>
</table>

Table 4.1 Notation of simulation model parameters

4.1 Overview of the experiment

Our experiment is composed of three parts: a trace generator, simulation model of multi-tiered system and a results analysis program. In trace generator part, we use Maple software to generate groups of traces with tunable statistic parameters including mean, coefficient of variation (CV) and autocorrelation (ACF). In each trace the numbers are exponentially distributed to represent the inter-arrival time of I/O requests. The local block number (LBN) address of each request sent from the operation system (OS) is attached to the original trace. For different periods of time
the LBN distribution is different which simulates the situation that the hot spot in storage system is changing over time. I.e., at different moments the host in OS reads and writes different files and locations.

In the next step, the traces generated in the first part are used as input in the core simulation model of our multi-tiered storage system. This system uses metadata structure to keep track of the access history for each block. When a migration job is triggered, the system finds the proper data and migrates them to another tier (since this is a simple model we only employ two tiers – an HDD tier and an SSD tier). Then based on the address of input requests and the preset configuration, this core model converts the input trace to a sequence of events, each with corresponding arrival and processing time. A detailed description of this system is given in section 4.3.

In the last part, the output sequence is delivered to a single server single queue model to calculate the waiting time/latency of each task. The analysis program in this part also provides detailed statistic data such as the hit rate of I/O requests on SSD tier and how often the SSD tier is stuck in performance degradation status.
4.2 The implementation of our multi-tiered storage system

The architecture of our multi-tiered storage system is given in Figure 4.1. First, a sequence of arrival events is generated and delivered to our multi-tiered storage system. This input trace records the information including the sequence number \( n \) (i.e. this is the \( n \)th arrival job), the interval time between job \( n \) and the last job \( n - 1 \), the logical block number (LBN) of this job \( \text{addr}_n \) and the read/write access pattern. Since this model does not acquire and return real data, and the latency between issuing and finishing of each request is the only thing we concern in this experiment, the real data transferred between host and storage system is ignored.
The biggest challenge for the multi-tiered storage system is placing the most frequently access data on the fastest SSD tier to maximize the benefits achieved by integrating SSD drives in the storage array [4]. How and when to implement the two-way data migration between SSD and HDD tiers is the key of the whole system’s efficiency and performance. In our model, two conditions will trigger the data migration in the storage system. First, our system takes use of the idle time to implement data migration aiming at avoiding extra workload when the traffic is heavy. This should remind the reader of the background garbage collection which is also triggered once the system is idle [9]. Here the inter-arrival time for each arrival job \( t_a \) is compared with the threshold value \( T \) to inform the system whether it is idle. However, the idle-time data migration is not enough especially in the situation when system keeps busy for long time. In this situation it is beneficial and necessary to insert forcible data migration job. Our solution is periodically inserting migration jobs even when the system is busy. Here for each incoming I/O request, the system first checks if the number of arrival job \( n \) is divisible by a fixed parameter \( N \) (i.e. the migration is triggered at a frequency of \( N^{-1} \)) and then decides whether to trigger and insert a data migration job on two tiers.

When the system is ready for a migration job, the first thing is to figure out the size of migration job. Several factors determine this size including the inter-arrival time, locality of hot data and the disk occupancy \( \rho \). The time gap between two arrival jobs describes the utilization and how busy the system is. With longer inter-arrival time which indicates the system is in a more idle status, larger size of migration job is allocated by the system. The occupancy of SSD tier \( \rho \) is introduced based on the fact that it is a crucial parameter when the performance degradation and write endurance is
concerned. The guideline here for the role of $\rho$ is that it adjusts the migration size of two directions: when $\rho$ is higher than a prefixed value $\rho_0$, the system would migrate more data from SSD to HDD to draw down the disk occupancy of SSD tier, and vice versa. This step guarantees the occupancy of SSD layer stay in the preset range $\rho_0$ and we will explain its significance in the experiment analysis section.

The main goal of data migration operation is to keep hot data in SSD and cold or inactive data in HDD layer. Here we introduce the notion of ‘temperature’ which describes the possibility that a particular block of data will be accessed soon. According to the temporal locality, if at one point in time a particular memory location is referenced, then it is likely that the same location will be referenced again in the near future. In our system we use a weighted algorithm to calculate the temperature for each location in the storage system based on its metadata which records how frequently it was accessed and when each access occurred. In this algorithm, each access on that particular location within the time window is assigned with a different value of weight based on how early that access occurred. Then the weighted sum is saved and updated in the temperature table. Once the system decides to trigger a migration job, system checks the temperature table and moves cold and infrequently accessed data in SSD tier to any empty address in HDD tier. The extra step for the migration in the opposite direction is that the hot data in HDD is moved to the coldest place in SSD, not a random address. By doing that we achieve a similar effect of wear-leveling and avoid frequently erasing one particular block in SSD which is significant to prolonging its lifespan.
After determining the type (read/write/migration) and the location (SSD/HDD) of each job, the “get process time” module assigns corresponding process time to it. In particular, the processing time for the write access on SSD has two values: SSD_WH for before performance degradation, and SSD_WL for after performance degradation. Recalling our Pump-Pool model, the volume of the water in the pool indicates the fact that how long it can sustain before performance degradation. Here in our module we use C (clean area/blocks) to represent this volume. By checking whether it is higher than zero we estimate the status of SSD drive in our system and assign high or low processing time to a write request on SSD. After that, the system adjusts the value of C based on the type and inter-arrival time for the request job. If the inter-arrival time is short and the request is to write on SSD, the value of C decreases. Otherwise, C raises and the increment value is decided by the idle time of the system and its upper bound C_max which is a function of the current SSD occupancy rate ρ.

```plaintext
1 update the temperature table and calculate current SSD occupancy rate ρ
2 if t_n > T or n mod N = 0
3 /*trigger migration job*/
4 if system is idle
5 trigger large migration job
6 else
7 trigger small migration job
8 if current SSD occupancy ρ < ρ_c
9 migrate more data to SSD
10 if ρ > ρ_c
11 migrate more data to HDD
12 if ρ = ρ_c
13 transfer the same amount of data
14 sort all physical addresses in all drives by their temperature
15 do migration
16 output a migration event with process time = Migration_time <case 1>
17 if the nth job is a write access on SSD
18 if t_n < T
19 if C > 0
20 output a write access event with process time = SSD_WH <case 2>
21 C --
22 else
23 output a write access event with process time = SSD_WL <case 3>
24 else
25 output a write access event with process time = SSD_WH
26 increase C based on the value of t_n
27 if the nth job is a write access on HDD
```
output a write access event with process time = HDD_W <case 4>
increase C based on the value of t_n

If the nth job is a read access on SSD
output a write access event with process time = SSD_R <case 5>

If the nth job is a read access on HDD
output a write access event with process time = HDD_R <case 6>

Table 4.2 Pseudo code of the core module with six output branches.
5 Simulation results and analysis

In our experiment, each trace issues 20000 I/O requests to the core module of our multi-tiered storage system. This system processes these incoming jobs dynamically and outputs a sequence of events, each of which has different processing time based on their corresponding I/O requests and the status of the system at the moment. The output sequence has more than 20000 events and the extra events are the migration jobs triggered by the system itself. Then the output sequence is delivered to a single server single queue model, and this model calculates and provides information including average service and delay time of the original 20000 jobs. The value of waiting time, which is the sum of service and delay time, is used to describe the total latency between one I/O request is sent and finished. In our experiment we use this parameter to evaluate the performance of the storage system.

5.1 Simulation results on different migration policies

The purpose of the first experiment is to explore and compare the system performance of different migration policies. In the previous paragraphs we introduce two tunable parameters in this system: migration size and migration interval N. The results of the experiment are given in Figure 5.1. As migration interval decreases from 500 to 100, the migration frequency grows by 5 times which implies more migration jobs are triggered and inserted into the sequence of original tasks. In that way data are better arranged and the possibility for one I/O request to access the data in the SSD tier (i.e., the hit ratio in SSD tier) is higher. Thus, the performance improvement is captured in the right part of Figure 5.1. When the migration interval continues to decrease, we can observe that the waiting time gets longer and performance becomes worse in the left
part of Figure 5.1 which is due to redundant migration. The excessive migration jobs bring not as much benefit as it when the interval is high, meanwhile the cost is so expensive that the whole system’s performance is dragged down. We can also acquire the optimal value for migration size in this storage system.

![Figure 5.1 The I/O latency of the system as a function of migration intervals (inverse of migration triggering frequency) for migration size = 2, 4, and 6](image)

The second experiment focuses on the influence of different space utilization ratio $\rho_0$ in SSD. As shown in Figure 5.2, the impact of the performance degradation in the SSD drive is remarkable. In the conventional view multi-tiered system (for example, the cache in CPU), the larger space in fast tier always means better hit ratio and performance. But in this experiment the optimal performance is not achieved at the point where $\rho_0$ equals to 1, and the reason is shown in Figure 5.3. As more space is used in the SSD tier, the number of requests in the HDD tier is reduced as expectation. But for those requests in the SSD tier, the high occupancy ratio $\rho_0$ also introduces
higher possibility that the SSD is in performance degradation status (highlighted by red dots).

![Figure 5.2](image1)

Figure 5.2 The I/O latency of the system as a function of preset SSD occupancy rate $\rho_0$

![Figure 5.3](image2)

Figure 5.3 The access types of the original 20000 jobs

In Figure 5.1 we present the impact of excessive migration. Here we modify some part of migration algorithm and compare the performance with the normal module.
The reason for the impact of excessive migration is, as the frequency and the size of the migration jobs increase, the system begins to migrate mild data that are neither too hot nor too cold. These excessive migration jobs are barely beneficial while they introduce considerable stress on the whole system. To avoid such situation, before the decision of migration is made, the system automatically checks the temperature gap between the coldest data in the SSD tier and the hottest in the HDD tier. Only if this temperature gap is larger than a preset threshold value $T_{\text{gap}}$ the system will trigger the migration job. Here in this section, the modified system is called the system with Intelligent Swap, distinguished from the previous “Normal Swap” system. The results shown in Figure 5.4 and 5.5 indicate that the benefit brought by this modification is more obvious in the following two cases: the system with high migration frequency, and with high disk occupancy rate. This is reasonable because in both of these two situations, the migrated data has higher opportunity that they neither too hot nor too cold. Such kind of migration transfers only mild data between SSD and HDD tiers and brings little benefit to the system.
5.2 Workload pattern versus $\rho_0$

In this part we study the performance of our multi-tiered system at different workloads and search for the optimal configuration in each situation. In the first group of tests the inter-arrival time of the input traces has different value of autocorrelation function (ACF). In Figure 5.6, as the ACF of the inter-arrival time increases, the average latency for each job gets higher which indicates worse overall performance. In this situation, higher space occupancy $\rho_0$ of SSD drive brings better performance. On the contrary when the value of ACF is small, the optimal performance is achieved not at the point where $\rho_0 = 1$ but somewhere around 0.6. An extreme example may explain the result here. Suppose all jobs with small inter-arrival time is issued at the very beginning which leads to heavy traffic in the system, then during the rest of time the workload is light and the inter-arrival time is high which keeps the system in the
almost idle status. In this situation different value of $\rho_0$ makes little difference on the
how often the system is in degradation status. That means the pool in our model is
either full (at the beginning) or empty (in the remain time) and the value of the
maximal capacity is no longer significant. In this extreme case, the ACF value of the
inter-arrival time is high, and the system with high SSD occupancy rate is barely
bothered and punished by performance degradation just like the purple line in Figure 5.6 implies.

In the second group, the changing variable is the coefficient of variation (CV) of the
inter-arrival time of the I/O requests. As is shown in Figure 5.7 when the inter-arrival
time has larger value of CV (the blue curve), the variation or dispersion exists from
the average is larger which means the workload is more fluctuating. In this situation,
the performance of the system becomes worse and the impact of performance
degradation in SSD is more obvious. In contrast, the workload with low value of CV
is more stable. In this situation performance (the red curve) is better and the optimal
performance is achieved at high SSD occupancy rate ($\rho_0 = 1$). This refers that the
system with high SSD occupancy rate is not affected by the performance degradation issue when the workload is smooth and stable.

![Graph](image)

Figure 5.7 I/O latency of the system as a function of SSD occupancy rate for CV = 0.73, 1.27, and 1.47

In the third group of tests we change the arrival rate of the workload. In Figure 5.8 when the average inter-arrival time of the incoming requests is low, which indicates high arrival rate and heavy workload, the latency increases as expectation and the performance degradation on the SSD tier harms overall performance as the space occupancy ratio $\rho_0$ increases (the blue line). When the pressure of workload on the system is low and the system is in the status somewhere next to idle, the high value of $\rho_0$ is more beneficial to the performance of the whole system (the purple line).
All these three groups of tests provide us the guideline of how to choose the optimal value of $\rho_0$ (Figure 5.9 shows the general view of it). In the next section we will use the conclusion of these tests to design a multi-tiered storage system which dynamically changes its configuration as the type of the workload changes.
5.3 Design an adaptive management of multi-tiered systems

5.3.1 The significance of the previous experiment

In the previous section we try to explore all corners of our simulation model of the two-tier storage system. Through these tests we validate our previous guess that the performance degradation phenomena, which is related to the underlying mechanism of SSD, may have influence on the performance of the whole system. In different types of workload, the optimal configuration of this system also change when the performance degradation is taken into consideration. Based on these experiment results, beneficial hints for how to establish a multi-tiered storage system are provided and these guidelines help us to set and adjust the parameters of the multi-tiered system.

5.3.2 Endurance degradation and lifetime wearing

Before this section, all the tests and discussion only concern the performance of speed and bandwidth through the I/O interface. Another issue that cannot be ignored is the endurance degradation and lifetime wearing, which is introduced by the nature of NAND flash memory. Given limited finite number of program-erase cycles (typically written as P/E cycles), the frequent write and erase requests can wear down the lifespan of the flash memory tremendously. The existence of the internal management operations including garbage collection and wear-leveling make the situation even worse. The concept of write amplification and its connection to garbage collection is presented in section 2.2. In this section we focus on the endurance degradation speed of the SSD drives with different occupancy rates. Several papers have provided the delicate and quantitative analysis on the write amplification (Figure 5.10, 5.11) and we use these results to estimate the wear rate in the SSD tier. Previous work by
Werner Bux gives the result on the write amplification over different drive occupancy and number of pages per block by developing two complementary theoretical models of the SSD operation for uniformly-distributed random small user writes [24]. Simona Boboila and Peter Desnoyers also provide similar result in an experimental approach. Figure 5.11 presents the degradation in relative endurance for several different combinations of device size $N$ (in erase blocks) and erase block size $k$, plotted against the fraction of free space in the device [20].
5.3.3 The design of the dynamic multi-tiered system

The results of the experiment in the last section show that in different types of workload, the optimal setting of parameters for the system to achieve best performance is different. In this part we establish a simple adaptive model to achieve two goals: better performance (higher bandwidth and lower latency), and lower endurance rate (longer service life for the SSD drive). In this system we dynamically monitor the parameters of the workload trace and adaptively change the predefined occupancy $\rho_0$ of SSD tier to achieve better performance, and at the same time avoid high endurance wear rate especially when the workload is heavy. The framework of this dynamic data migration system is given in Figure 5.12.

![Figure 5.12 Framework of our dynamic data migration system](image)

The workload trace in this experiment has 100,000 jobs and in different periods of time it has different parameters. The performance of this adaptive system is shown in Figure 5.13. Here we can observe that the performance of this adaptive system is as
good as the system where $\rho$ is fixed at 0.9 or 1. On the other hand, as is shown in Figure 5.14, it keeps endurance wear rate at very low level which is much lower than the situation where SSD space occupancy is high. From this experiment we can conclude that this adaptive multi-tiered system achieves both high speed performance and low endurance degradation rate, which are the two main purposes of the project.

Figure 5.13 Performance comparison between the adaptive multi-tiered system and the system with fixed SSD occupancy rate $\rho_0$.

Figure 5.14 Relative endurance wear rate comparison between the adaptive multi-tiered system and the system with fixed SSD occupancy rate.
6 Conclusion and future work

In this paper we discuss the working mechanism of the NAND-based solid state drive and propose the problem of performance degradation under small random write workload. By running the benchmark in the real environment, we present the relation between the occupancy rate of SSD and the performance degradation. Based on that we present the Pump-Pool model to explain the performance degradation in a quantitative way. Then a simulation model of multi-tiered storage system is given, and we explore its performance with different configurations under different types of workloads. With these results and previous research on write amplification, an adaptive system is established and the simulation result validates that it achieves good performance in both speed and endurance.

Through trace-driven simulation, this paper provides theoretical view of our multi-tiered storage system and the impact introduced by the performance degradation in the SSD drive. Real environmental implementation for this system is needed to validate the simulation results presented in this paper. In addition to that, the implementation of the final adaptive system can be improved through other effective ways including machine learning, and this is crucial for establishing a multi-tiered system in the real environment.
7 Acknowledgements

I would like to express the deepest appreciation to my advisor, Prof. Ningfang Mi. It is in her class I got the initial idea of this thesis. Without her guidance and persistent help this dissertation would not have been possible.

I would like to thank my committee members, Prof. Yunsì Fei and Prof. Kaushik Chowdhury, who spend time in reviewing my work and offer me valuable suggestion.

In addition, a thank you to three PHDs in my lab, Jianzhe Tai, Yi Yao and Zhengyu Yang, who give me a lot of beneficial discussion on my research topic. I thank Jun Li for his scripts that is used in my experiment. I also thank Dr. Simona Boboila for her previous research and she also gives me plenty of suggestions on my experiment. At last I would give my appreciation to my families and Angelina for their spiritual support.
8 Citation


[28] "Linux 2.6.33 Features". kernelnewbies.org. 2010-02-04. http://kernelnewbies.org/Linux_2_6_33#head-b9b8a40358aeef60a61fcf12e9055900709a1cfb.