Research on Parallel Pipeline Strategy and Controllable Write Allocation of High Speed Solid State Storage Array

Xiaowen Wang[1, a], SongYan Liu*[, 1, b], Huan Liu[1, c], Yanlin Chen[1], and Yifei Niu[1]

[1]School of Electronic Engineering, Heilongjiang University, Harbin 150076, China.

[a]wxw0917@outlook.com, [b]liusongyan@hlju.edu.cn, [c]2171390@s.hlju.edu.cn

*Corresponding author

Abstract. Due to the special write mode and limited lifetime characteristics of NAND Flash, a variety of technologies for the solid state disk flash translation layer (FTL) have been spawned. The existing Flash channel management method does not completely release the bandwidth of the Flash channel. In response to this problem, combined with a Nand Flash channel organization form of a solid-state disk, a multi-level parallel management mode is designed and implemented inside the SSD, which greatly improves the solid state. At the same time, a write distribution mechanism implemented in the upper layer software is designed. This mechanism can cooperate with the internal firmware of the SSD to realize the controllable write allocation of the host, strengthen the management ability of the user to the SSD, and realize the transparency of the data block.

Keywords: Solid state storage, Parallelism, Write allocation, Nand Flash controller.

1. Introduction

Recently, the demand for high-speed information services, mass data storage, and high-speed acquisition has continuously challenged the highest performance of storage devices. The ms-level response time of traditional disks has been unable to meet the ever-increasing data storage requirements. Nand Flash has sprung up in the storage market and has become the mainstream high-speed storage device[1].

To improve the performance of SSDs and extend the life of SSDs, there will be a flash translation layer (FTL) in the SSD. It handles all transactions related to Nand control. Key features include wear leveling[2], cache management[3], address mapping, garbage collection[4], etc. The strategy of the FTL directly affects the performance of the SSD.

At present, FTL of SSDs is mostly implemented in the disk. SSD is a black box to the Host, and the real transmission path of the data cannot be known. Therefore, a host-based FTL implementation is proposed, which establishes a model, but only initially implemented on the actual platform[5].Based on the actual SSD hardware platform, the traditional FTL is split into a driver layer and a firmware layer. The powerful computing power and a large amount of memory resources on the host side can completely release the storage performance of the SSD. On the host side, through a write allocation strategy and command scheduling within the firmware layer, read and write optimization is implemented on the basis of ensuring data integrity. Different I/O policies can be used for application workloads with different read/write ratios.

2. System Resource Analysis

2.1 Hardware Architecture.

A solid-state storage disk usually consists of a main controller, a Nand controller, a RAM controller, an external interface, and other functionally required buses[6]. A SSD has multiple flash channels and uses DMA to transmit data internally. SATA or PCIe interfaces are usually used externally. Since the bandwidth of SATA is limited to 600MB/s or less, in order to meet the high-speed storage requirements of practical applications, using PCIe will not make the interface a bottleneck restricting the performance of the SSD.

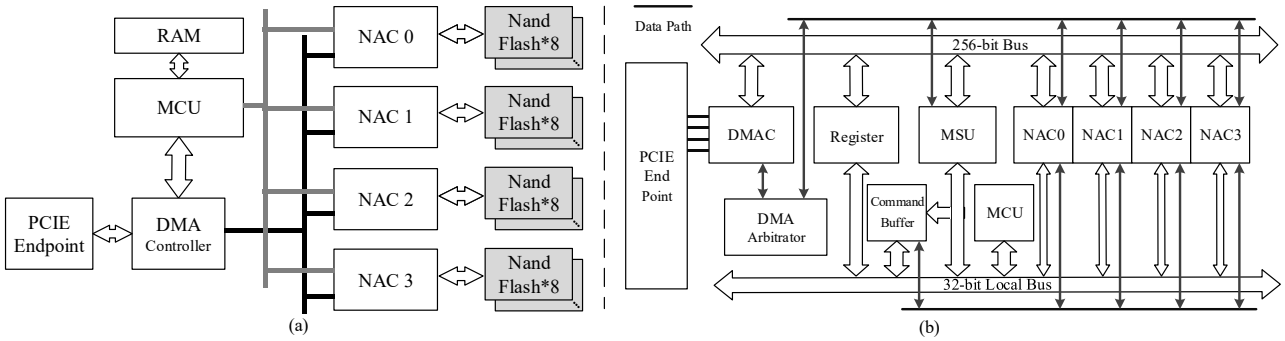

Fig. 1 Solid state disk controller overall structure and Internal module connection

The hardware platform is implemented using Xilinx Virtex FPGA. The main control and the Nand controller are implemented by using logic IP inside the FPGA. The SSD has a total of 32 Nand Flash channels, which are divided into 4 groups and 8 channels as a group. Multiple ARM-Cortex controllers are implemented inside the FPGA. One is responsible for the processing of messages and control commands. It is called the Message and Control Unit (MCU). Each of the four flash channels is adapted to one, called the Flash Array Controller (NAC). The overall architecture of the solid state storage disk is shown in Figure 1-a.

After receiving the scheduling command of the MCU, the NAC goes to the command cache pool to retrieve the corresponding command and converts it into operation on the flash chipset. This operation is generally a read/write command. At this time, the NAC sends a request to the DMA controller to request the DMA operation of the corresponding address and simultaneously reports the status to the MCU. A DMA controller with a 256-bit width performs DMA operations on four sets of flash chips.

For a read request, the DMA controller converts it to a PCIe read request and sends it to the host via the PCIe controller. When the connection establishment completion signal is received, the DMA controller writes data to the data buffer of the DMA source of this operation. For write requests, the DMA controller reads data from the data buffer of the DMA source and writes to the corresponding flash address. The CRC check is performed at the same time as each module transmits data to ensure the correctness of the DMA operation. The connection of each module is shown in Fig. 1(b).

2.2 Host-FTL.

The advantages of implementing the Host-FTL side have gradually been discovered and studied by the industry. The literature [7] further completed the FTL implementation of the OpenSSD platform on the basis of the literature [5], which increased the optimization of the thread and greatly improved the small-grained writing I/O scenario. But in OpenSSD, the dual-core Cortex-A9 chip embedded in the Xilinx ZYNQ7000 has limited performance. So it's not a good host choice. In literature [8], the open-channel SSD subsystem is implemented in the Linux Kernel. All channels are displayed for the Host through the PPA I/O interface, instead of being sealed in the SSD board.

2.3 Multi-Level Parallelism.

SSDs can be divided into multiple levels of parallel resources from the channel to the inside of the chip[9]. Combined with the design of Host-FTL, resources are mainly divided into two aspects: logical channel level and logical unit level. The logical channel level includes command scheduling of the upper layer software and I/O scheduling of the driver layer. The logic unit level includes the flow control of a set of flash chips, and the simultaneous operation of multiple planes inside the chip. During scheduling, the MCU can circulate pipelines to distribute instructions to each logic unit. In this system, a set of flash chips is in the form of a multiplexed connection on the bus, which can be regarded as a flash chip. Although the bandwidth of each flash block is more widely distributed, this solution is not realistic due to the limitation of the number of chip pins and the storage capacity of a single disk.

Using the FPGA resource to construct the data buffer, the MCU will wait for the data buffer then perform the write operation. The physical address to which the data will be written is determined by the mapping table maintained by the FTL layer. The data buffer consists of a dual-port SRAM with a

size of 16 pages, so that the next data write preparation can be performed at the same time as the write operation to ensure the system's responsiveness.

Self-starting garbage collection mechanism is a major factor affecting the performance of SSD[10]. When the FTL is inside the SSD, research will reduce the impact of garbage collection by optimizing the data migration timing of garbage collection and considering the classification and recovery of hot and cold data[11]. However, the point of garbage collection in the SSD is still unpredictable on the Host side. Based on Host-FTL, the channel status can be opened to the upper layer application, and the recovery operation can be performed at the most suitable time. Further, it is possible to perform time-division garbage collection on different flash chipsets without causing the SSD to generate idle bandwidth due to GC operations.

3. Firmware Command Management

The Host sends the command to the MCU and stores it in the command buffer built by the dual port SRAM. The operating speed of the SRAM and the MCU are kept at an order of magnitude and do not affect the system speed. During the use of SSDs, commands have two characteristics: burstiness and concentration. Since the number of commands from the host is unknown to the MCU and is infinite, an appropriate strategy should be used to schedule the command buffer to handle burstiness. If there is intensive access to one channel and the other channel has no command, the SSD will show a significant performance degradation. In order to avoid the waste of channel resources, the system should try to avoid the centralized access of the host to the flash channel.

When a new command revenue from the host is received, the message management unit writes the command entry to the command slot queue, and the firmware obtains the command status information by reading the NAC 0-3 command slot read port register. The firmware loops through to find commands in the command buffer. As shown in Fig. 2 (a), the NAC command slots built using hardware logic are divided into two categories, one with four slots, corresponding to four NAC command slots, each of which is a 192-byte space. Different commands have different command formats, such as command IDs and different lengths.

Depending on the command format, the firmware must send command data to different parts of the NAC 0-3 command pool with special out-of-band data regions: the register portion and the host address portion. After the command data in a specific slot is sent to the NAC 0-3 command pool, the slot can be freed for reuse. The firmware pushes the command state back to the empty command slot queue by writing to the null command slot queue to write to the port's registers. Each write to this register will push an entry back to the queue.

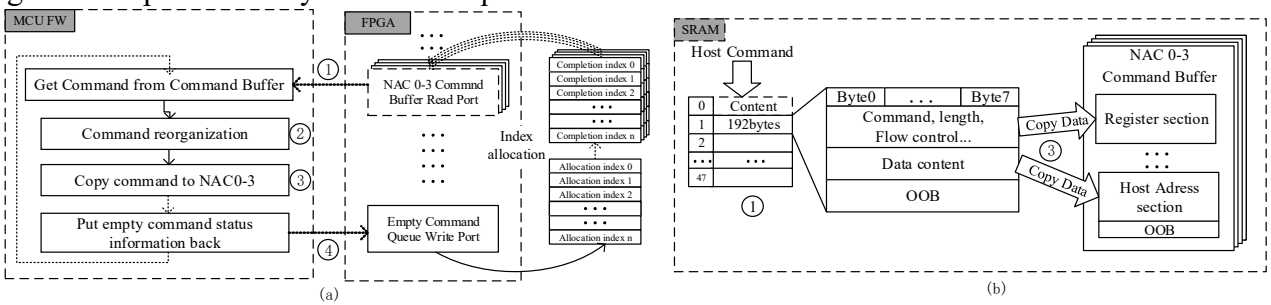

Fig. 2 Command allocation mechanism

4. Host-FTL Read and Write Allocation

4.1 Optimization Feasibility

Although the SSD has high-speed transmission conditions on hardware resources, due to the three special characteristics of the solid-state storage array. First, it cannot cover writes like traditional magnetic media, only can be erased and written first. Second, in terms of space, the granularity of reading, programming, and erasing operations is different. Reading and programming are in units of pages, and erasing is in blocks. The third is time, the read time is much lower than the erase time and

a flash wafer cannot respond to other instructions from the Host while performing programming operations. The details of the three types of flash chips are shown in Table 1.

Table 1 Comparison of typical values of three flash memories

Flash type	SLC	MLC	TLC
Erase life	≈100k	≈5k	≈1.2k
Read (us)	≈25	≈50	≈75
Write(us)	≈260	≈580	≈840
Wipe (us)	≈1500	≈2800	≈4200

Classic FTL algorithms such as NFTL, BAST, and FAST use a more reasonable full coupling and hybrid mapping to maximize the life of the solid state disk, but each channel is still used alone. The parallelism of multi-channel multi-level is not fully utilized.

In the system, channels are managed using a classification aggregation. There are 4 logical channels in the system, each channel has 8 physical channels, and there are two wafers that can be executed in parallel in the physical channel. There are corresponding caches between the resources at each level for buffering.

According to the use of the SSD, the granularity of the operation can be adjusted according to the load condition. When a large file is written, the super page is required to be continuously written. Each channel constitutes a super page, and the write is allocated, so as to occupy the flash bandwidth as much as possible. Ideally, puts all operational wafers in a programmed state. When the system is operating, the small file will be changed frequently, the super page mode will be discarded, and the granularity will be changed to 4 KB. Continuous reading can be read continuously if it reads a large file that was previously written. However, with the use of SSDs, the static wear leveling strategy may change the position of the data block. If the data block holding the fixed data is written less frequently, it will be replaced with other data. Therefore, block-level read scheduling is used for continuous reads, and the four channel requests are separated.

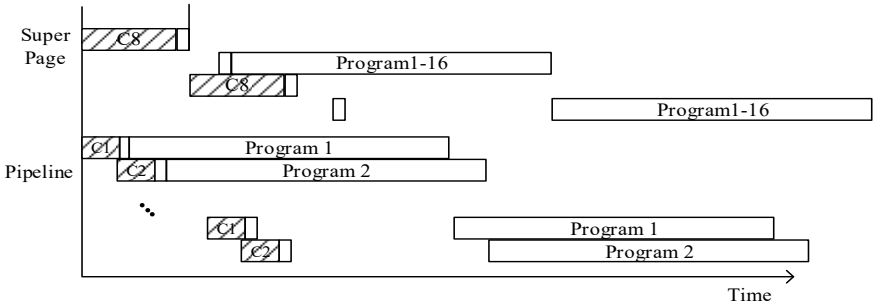

Fig. 3 Super page and pipeline mode

The write operation is shown in Figure 4. The Host sends the data to the cache of each flash chip. One channel executes the programming command uniformly, and writes N times of data in the same programming time. N is the number of super pages. C8 is the I/O time for transmitting all pages, and a time is the cache time for preparing all pages. When the cache is ready, it is written uniformly. If you do not use the super page strategy, for normal write operations, another optimization method is used in the system that is, using the classic pipeline strategy. Controls the order in which the Host commands are transmitted. C1 is the data I/O time of one page. Since the page programming time is much larger than the I/O time, the difference is two orders of magnitude. Therefore, the data can be flooded and the I/O time can be completely drowned during the programming time. However, the pipeline does not guarantee that it can be connected every time, mainly because it takes more time for the MCU, and may be busy wait.

4.2 Write Allocation.

The strategy of writing allocation is for block groups. A write allocation list is configured for each independently programmable flash die. In a Die, the total number of block groups is not high, so a 2byte size index is sufficient for addressing all block groups and page groups. Lists can be saved in a linked list, so you can achieve page-level mapping speed and parallelism of multiple channels. When an exception occurs, the system first stores these lists in the flash memory.

The smallest physical page size is 16KB. Because the logical sector size of the system is defined as 4KB, some consolidation strategy is needed to effectively use space and program bandwidth. Due to the 4KB granular mapping strategy, 4 logical sectors can be aggregated into a single page, even if they are independent of each other. This involves a problem, where to go to 4 pages to write to a physical page, one is to aggregate in the Host memory, and the other is to use the SRAM resources of the FPGA. Although the memory capacity of the Host is large and the number of aggregates is large, it will be transported back and forth in the memory, occupying the memory bandwidth of the Host, and the response time will be affected by the Host. While using SRAM to build a data pool, using the hardware logic of the FPGA to automatically combine, so there is almost no delay, it is a better solution.

The actual load may cause the block group wear in the channel to be unbalanced. To maintain the balance of space utilization, the channel with the least data is written first instead of the usual cycle. Therefore, the channel with the lowest space utilization rate may be the channel with the most cold data, allowing it to carry more thermal data to maintain the overall life of the solid state disk.

5. System Test

5.1 Test Environment Setting.

The test environment is based on the SSD storage system and is connected to the Host use PCIex4 link. The Host configuration is shown in Table 2.

Table 2 Test host configuration

CPU	Intel i7 6700K @2.8Ghz
RAM	2400Mhz DDR4 16GB
OS	CentOS 6.5/Linux kernel 3.10
Interface	PCIe 3.0x4
Host SSD	Samsung PM961 256GB
Tools	IOmeter

Since the performance of the SSD is affected by various internal mechanisms, the simulation and testing of a certain internal module can't reflect the impact on the performance of the SSD. This paper selects the black box test method and tests the entire SSD system. Test data includes sequential read and write speeds, random read and write speeds, IOPS, delays, and more. IOPS (I/O Operations per Second) is an important parameter that reflects the performance of SSDs. The value of IOPS varies greatly depending on the system configuration, including the ratio of read and write, the proportion and configuration of sequential access and random access, the number of threads, and the depth of the access queue.

5.2 Test Data.

Table 3 IOPS performance of different block sizes and different queue depths

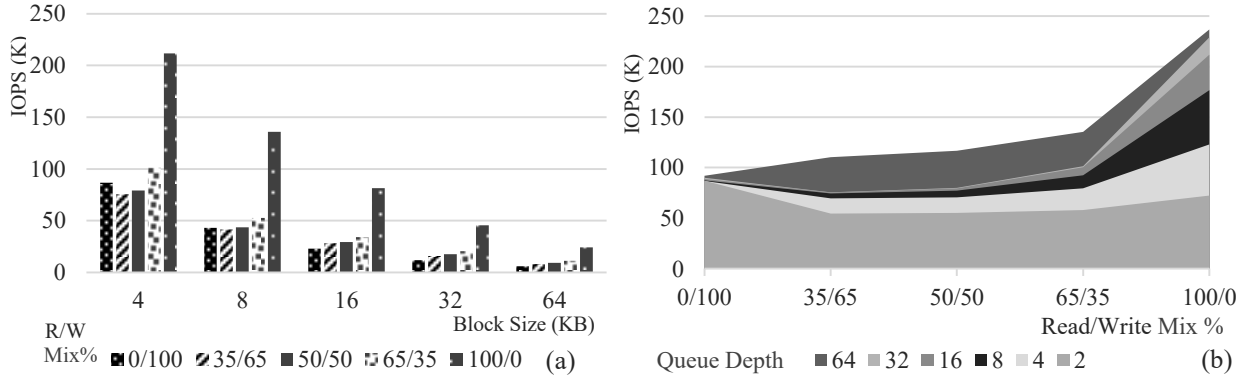

Table 3(a) shows the IOPS for different block sizes and five types read and write request ratios with a queue depth of 16 and a number of threads of 4. The ratio of 5 different read and write requests represents different load conditions. It can be seen that when the block size is 4KB, the IOPS of the 100% read request is much higher than other cases, mainly because the read does not trigger internal garbage collection. In the case of IOPS, the level is stable at 75 to 100K. For larger block sizes,

because the system uses the management of block groups, IOPS is inversely proportional to the block size. Since the total number of bytes in one operation becomes larger, the total bandwidth is still not reduced, which is in line with the system design expectations.

At different queue depths, IOPS will have different manifestations. Table 3(b) shows the IOPS performance from 2 to 64 queue depth for a block size of 4KB and different read-write mix modes. As the depth of the queue deepens, the improvement in IOPS is smaller. In the case of a full read request, a higher queue depth will still improve, mainly because the read request is executed much faster than the write request, and the queue flows to receive feedback more quickly.

Unlike ordinary SSDs, the throughput and rate of Host-FTL SSDs are directly related to queue depth. For write operations, each flash chip has the longest write time, and the system hides other I/O times during the write time by pipelining or Super Pages. The overall performance of the write operation depends on the bandwidth of the internal programming operation and can be expressed as Equation 1.

$$S_{write} = N_{log} \times N_{LUN} \times \frac{D_p \times N_{phy}}{T_p} \quad (1)$$

N_{log} represents the number of logical channels, N_{LUN} represents several Dies inside a chip, D_p represents the data size of a page operation, N_{phy} represents the number of physical channels in the logical channel, and $T_p$is the programming time. Equation 1 does not calculate factors such as internal bandwidth consumption and ECC check time. This can roughly estimate the total write bandwidth $S\_Write = 4 \times 8 \times (15.6K \times 2)/600us \approx 1600MB/S$. For the read operation, it can be seen from Table 2 that the read consumption time is about 50us, and the read bandwidth can be estimated to be about 10 times of the write. The bottleneck of the system lies in the I/O part, and the time of this part is mainly determined by the DMA rate.

$$S_{read} = W_d \times f_d \div 8 \times \frac{D_n}{D_{all}} \quad (2)$$

In Equation 2, W_d is the data width of the DMA, f_d is the DMA frequency, and D_n/D_all is the effective data ratio, so $S_{read} = 256bit \times 120Mhz \div 8 \times 0.9 \approx 3450MB/S$.

Selecting different queue depths and block sizes will affect the continuous read and write. As shown in Table 4(a), when the block size is large, the SuperPage policy takes effect at this time, and the queue depth has little effect on the continuous write speed because The input speed is mainly determined by the programming speed, and the upper layer is more likely to fill the write channel of the entire solid state disk. The problem of slower write speeds occurs only when the queue depth and block size are small.

Table 4 Comparison of continuous write and read speeds at different queue depths

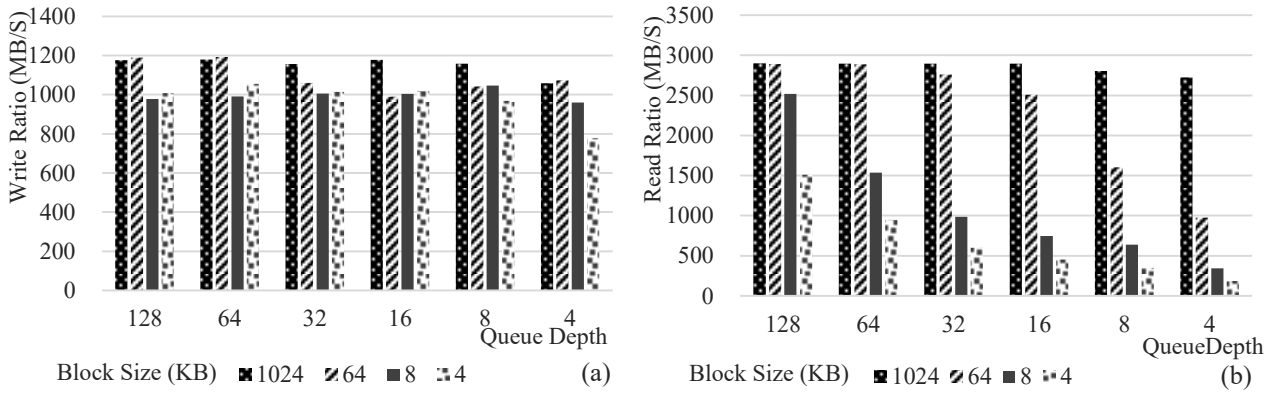

Table 4(b) shows the impact of different queue depths and block sizes on sequential reads. It can be seen that in the case where the block size is large enough, the continuous read speed is maintained at a level of about 2800 MB/s, and the bandwidth is basically run in consideration of the space for DMA operations and instruction operations. As the block size decreases, the rate exhibits a following decrease. When the block size is below 8 and the queue depth is less than 32, there is a large decrease due to insufficient read instructions to form a continuous address.

6. Conclusion

Based on an FPGA as the main control solid-state storage array, considering the current difficulties and difficulties in the development of FTL solid-state storage disks, a Host-FTL-based write distribution method is designed. With the custom hardware structure in the FPGA, the parallelism and flexibility of the solid-state storage array are improved as much as possible, and the host directly manages and deploys the hardware. Tested in the actual hardware system, it is found that the hardware architecture and Host-FTL can effectively improve the parallelism of the solid-state storage system, and have a big breakthrough in IOPS and delay, which can exert the proper hardware performance, especially for large The data block group performance improvement is more obvious. Host-FTL makes it easy to expand and modify SSDs in the future, and is more in line with the future development of storage devices.

References

[1]. Wang Z，Zhang Y，Wu Q，et al. Degradation reliability modeling based on an independent increment process with quadratic variance[J]. Mechanical Systems & Signal Processing, 2015, 70:467-483.

[2]. Wei DB, Deng LB, Zhang P, et al. A page-granularity wear-leveling (PGWL) strategy for NAND flash memory-based sink nodes in wireless sensor networks. Journal of Network and Computer Applications, 2016, (63): 125-139.

[3]. Cui J, Wu W, Zhang X, et al. Exploiting latency variation for access conflict reduction of NAND flash memory. MASS Storage Systems and Technologies. IEEE, 2017.

[4]. Shi L, Wu K, Zhao M, et al. Retention Trimming for Lifetime Improvement of Flash Memory Storage Systems[J]. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, 2016, 35(1):58-71.

[5]. OpenSSD Project. http://www.OpenSSD-project.org/wild/The OpenSSD Project.

[6]. Pua K S, Lee K L. Solid state disk storage system with parallel accesssing architecture and solid state disck controller: U.S. Patent Application 11/874,080[P]. 2009-3-26.

[7]. González J, Bjørling M, Lee S, et al. Application-driven flash translation layers on open-channel ssds[C]//Nonvolatile Memory Workshop (NVMW). 2014.

[8]. Bjørling M, González J, Bonnet P. LightNVM: The Linux Open-Channel SSD Subsystem[C] //FAST. 2017: 359-374.

[9]. Chen F, Lee R, Zhang X. Essential roles of exploiting internal parallelism of flash memory based solid state drives in high-speed data processing[C]//High Performance Computer Architecture (HPCA), 2011 IEEE 17th International Symposium on. IEEE, 2011: 266-277.

[10]. Soga A, Sun C, Takeuchi K. NAND flash aware data management system for high-speed SSDs by garbage collection overhead suppres-sion[C]//Memory Work Shop (IMW), 2014 IEEE 6th International. IEEE, 2014: 1-4.

[11]. Qin Y, Feng D, Liu J, et al. DT-GC: adaptive garbage collection with dynamic thresholds forSSDs[C]//Cloud Computing and Big Data (CCBD), 2014 International Conference on. IEEE, 2014: 182-188.