Reducing Garbage Collection Overhead in SSD Based on Workload Prediction

Pan Yang, Ni Xue, Yuqi Zhang, Yangxu Zhou, Li Sun, Wenwen Chen,
Zhonggang Chen, Wei Xia, Junke Li, Kihyoun Kwon
Samsung R&D Institute China Xi'an, Samsung Electronics

Abstract

In solid-state drives (SSDs), garbage collection (GC) plays a key role in making free NAND blocks for newly coming data. The data copied from one block to another by GC affects both the performance and lifetime of SSD significantly. Placing the data with different "temperature" into different NAND blocks can reduce data copy overhead in GC. This paper proposes a scheme to place data according to its predicted future temperature. A neural network called LSTM is applied to increase the accuracy of temperature prediction in both temporal and spatial dimensions. And it also uses K-Means to do clustering and automatically dispatch similar "future temperature" data to the same NAND blocks. The results obtained show that performance and write amplification factor (WAF) are improved in various applications. In the best case, the WAF and 99.99% of the write latency are reduced by up to 43.5% and 79.3% respectively.

1 Introduction

NAND flash-based solid-state drives (SSDs) have found wide use recently. But due to the side effect of garbage collection, the performance of SSD may be degraded and the lifetime may be shortened.

In SSD, the programmed page cannot be reprogrammed before erasure. If the data in NAND page is no longer needed, the page will be marked as "invalid". However, flash memory can only be erased in the unit of block which is composed of multiple pages. As more and more invalid page occupies storage space, to reclaim free space, SSD needs to find a block, called the victim block, usually with the largest number of invalid pages. Then valid data in victim block are copied to an available, previously-erased block, making the victim block ready for erasure and subsequent reuse. This process is called garbage collection (GC).

GC is the most efficient if the victim block contains no valid page. However, as SSD is continuously written, the valid and invalid pages will exist in the same block. Thus, valid pages must be copied by GC, and the actual amount of data written into flash memory can be larger than the amount the host system issues. The ratio of them is write amplification factor (WAF). High WAF means that there are a large number of copies of valid pages in GC, which brings big trouble for SSD. For one thing, moving the valid pages consumes internal computing and bandwidth resources and decreases the I/O response performance. For another, the more data are migrated, the quicker NAND flash will be worn out, and consequently, the lifetime of SSD is decreased as well.

The main method for improving GC is to optimize the valid pages' copy to reduce WAF. Wu et al. [1] chose the block with the least number of valid pages as a victim block. Zhang et al. [2] used the idle time to copy some valid pages in order to improve the copy efficiency in GC. Kim et al. [3] predicted the data lifetimes by using write program contexts in RocksDB. This can reduce the valid page of the victim block. However, the prediction is highly depended on applications. Another idea is using data temperature as data with the same temperature may get invalid simultaneously. Jang et al. [4] classified the data into three types with different temperature (hot, cold, and warm); and different types of data were stored in different blocks. Rho et.al [5] distinguished the temperature based on file type. This is a coarse-grained division and requires researchers to be familiar with the host application.

In this paper, we adopt the idea of placing data according to data temperature to optimize GC. In addition, we introduce machine learning technology to predict the future temperature of data. The main contributions are as follows:

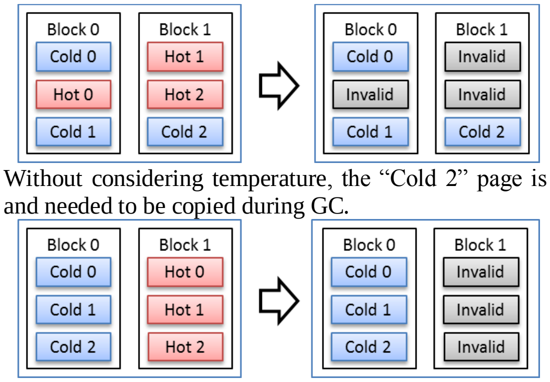
- This paper predicts data temperature in temporal and spatial dimensions according to the previous temperatures and I/O profile features by using a neural network called long short-term memory (LSTM).
- This paper uses K-Means to cluster the data with different temperature and dispatches them to different blocks accordingly.
- This paper also tests the proposed scheme in real SSD with four different applications. The evaluations show that, in the ideal scenario, it can reduce the WAF by 44% and 99.99% of the write latency by 79.3%.

2 Background and related work

2.1 Data placement based on temperature

Since the host continues to perform I/O operations, the same logical address may be written multiple times, known as the update operation. The temperature is measured by the update frequency of a logical sector, i.e., the write count of a logical address over a period of time. A high frequency means a hot temperature, which indicates the data in flash will be invalid in a short period of time.

Placing data into different block according to different temperature can reduce the copy of valid pages. As Fig. 1 shows in (a), without considering temperature difference, all data is placed into the flash in time sequence. Each block consists of valid data and invalid data after a while and leads to the increase of valid page copy in GC. In (b), data is placed based on their temperature, i.e., the cold data and hot data are stored separately. After a short period of time, all of the hot data become invalid and there is no valid page copy before erasure. Therefore, placing data based on temperature is an effective method to reduce WAF.



(a) Without considering temperature, the "Cold 2" page is valid and needed to be copied during GC.

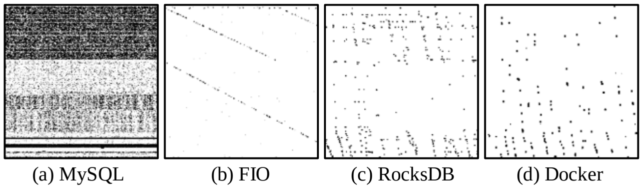
(b) Considering temperature, no copy is needed for block 1.

Fig. 1. Data placement based on GC

2.2 Temperature detection and prediction

Numerous researches focused on detecting data temperature to diminish the influence of GC. For example, Park et al. [6] identified hot/cold data and stored them in different blocks. Stoica et al. [7] estimated data temperature according to the data update frequency. Luo et al. [8] proposed an effective, window-based on-line algorithm to identify frequently-written page. Yang et al. [9] combined some other attributes to detect data temperature and separated them into multiple levels (more than just three levels).

In all these papers, the temperature is obtained by detecting its present state. Fig. 2 shows different workloads have different temperature change rules. For MySQL, different chunks have different temperatures which change slightly over time. For FIO, the temperature translates from one chunk to another. For RocksDB and Docker, the change is in both temporal and spatial dimensions. The temperature detection may be suitable for the case wherein the temperature changes slightly (e.g., MySQL). For the case in which the temperature changes sharply (e.g., RocksDB), the future temperature is very different from the present. If the identified hot data is put into a hot block, it may turn cold and still be valid in the future. This increases the data copy of GC on the hot blocks. Thus, future temperature should be taken into account in data placement. Based on this, the research intends to predict the future data temperature to guarantee dispatch accuracy.



(a) MySQL (b) FIO (c) RocksDB (d) Docker

Fig. 2. Different gray levels in vertical means different temperature in different regions, while different gray levels in horizontal means temperature is changing over time.

3 Scheme Design

GC can be improved by placing the data with different future temperature into different blocks. Our scheme predicts the future temperature and dispatches data into different blocks according to the prediction. The scheme mainly includes three modules: the workload features capture (WFC), the temperature prediction (TP), and the block dispatch (BD). The architecture of our scheme is shown in Fig. 3.

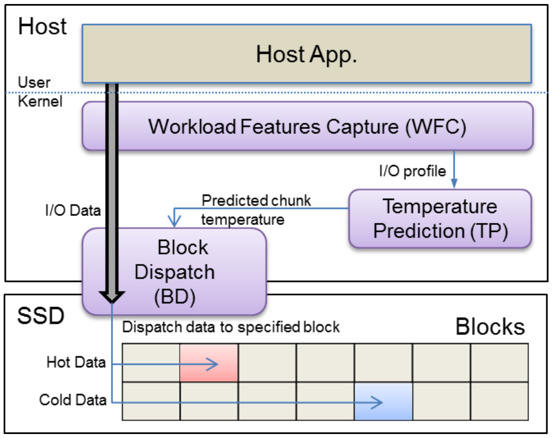

Host
Host App.
User
Kernel
Workload Features Capture (WFC)
I/O profile
Predicted chunk temperature
Temperature Prediction (TP)
I/O Data
Block Dispatch (BD)
SSD
Dispatch data to specified block
Blocks
Hot Data
Cold Data

Fig. 3. Scheme architecture

3.1 Workload features capture

To get the I/O profile, WFC module is deployed on the path where data is written by host to device. Workload features are captured in the help of our self-developed tool called StoneNeedle. StoneNeedle can capture more than 30 types of features such as the throughput, bandwidth, and I/O size/count of each logical sector. We can choose features to be captured according to the actual situation.

Workload features are continuously changing. StoneNeedle outputs the features captured in each period of time (1s, 5s or 10s).

A system normally has a large number of logical sectors to store data. It is resource-consuming and unnecessary to process the information for each sector. In WFC, the entire address space is divided equally into chunks, with each chunk containing a fixed size of sectors. For different applications, the larger the temperature changes, the smaller the size of the chunk should be. All the I/O requests falling in the same chunk will be treated as the feature of this chunk.

Only a part of the features captured by StoneNeedle have a great impact on temperature. Therefore, using all these features to predict the future temperature is time-consuming and may even interfere with the prediction. We choose features that are strongly correlated with temperature using Pearson correlation coefficient (PCC) [10]. The closer the value is to 1 or -1, the more relevant it is to temperature. Fig. 4 shows the PCC between temperature and other features in RocksDB.

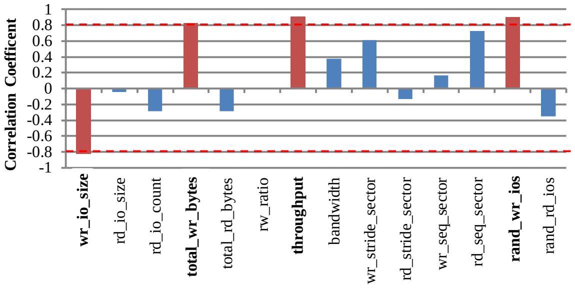


Fig. 4. Correlation coefficient in RocksDB

As shown above, there are four features (red bars) correlated to temperature. So these 4 features are selected as correlated features to predict future temperature in dockerized application. Note that the correlated features may be different in different applications.

StoneNeedle outputs temperature and correlated features collected during each fixed period of time and makes up a record:

$$\boldsymbol{rec}_t\ (f_{1_t}, f_{2_t}, f_{3_t}, \dots, T_t)$$

where T_t represents the temperature of each chunk in the t^{th} period of time, and f_{i_t} represents a correlated feature value (e.g., throughput) during the t^{th} period of time.

The records of all periods assemble into a serial data set which is the sequence of $\boldsymbol{rec}_1$, $\boldsymbol{rec}_2$, …, $\boldsymbol{rec}_t$, denoted as $\boldsymbol{R}$.

3.2 Temperature prediction

There are some challenges in the prediction of future temperature. Firstly, the temperature may change sharply in different periods. Secondly, a chunk's temperature may be related to other chunks. Thirdly, as mentioned before, the temperature may also be impacted by other features. Take RocksDB as an example. Firstly, many hot chunks often become cold in the next period. This is a change in the temporal dimension. Secondly, the accesses of some chunks are often followed by other chunks. Thus, the temperature will translate from some chunks to other chunks. This is a change in the spatial dimension. Thirdly, as is shown in Fig. 4, the temperature is highly related to the feature of write IO size, throughput, etc.

Considering the above three issues, we think LSTM [11] is the most suitable solution. Many prediction algorithms can be used for time series, but most can only use the historical temperature to predict future temperature. However, LSTM can comprehensively consider multiple factors for prediction and is thus deemed as an ideal model to solve the issues in our research.

For the given time step k, LSTM uses the k historical records (from t-k+1th to t^{th}) to output the predicted temperature T_{t+1}. As shown in Fig. 5, the input is the sequence of $\boldsymbol{rec}_{t-k+1}$, …, $\boldsymbol{rec}_{t-1}$, $\boldsymbol{rec}_t$ in k time periods, with each dimension of input corresponding to a feature in $\boldsymbol{rec}$. The output is the predicted temperature T'_{t+1} of the next time period.

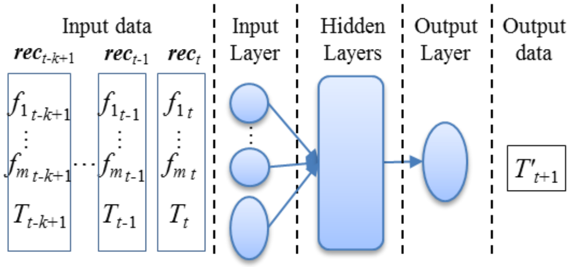


Fig. 5. Temperature prediction by using LSTM

This can be seen as a function:

$$T'_{t+1} = \text{LSTM}\big(\boldsymbol{R}_{t-k+1,k}\big) \tag{1}$$

where $\boldsymbol{R}_{t-k+1,k}$ represents the subset of $\boldsymbol{R}$ that has k sequent records starting with the t-k+1th record. Apparently, since the input data includes the correlated features, the temperature of all chunks, as well as the k time periods records, LSTM can predict the future temperature more accurately.

For a large-volume SSD with a huge number of chunks, it is unrealistic to input all the chunks to LSTM simultaneously, which requires large computing resources and training sets. To handle this problem, for one thing, we treat the chunks with similar I/O patterns as one chunk to reduce the number of chunks, referring to the workload pattern similarity analysis [12]; for another, we put chunks in batches (one batch at a time) and output the predicted temperature of chunks of the current batch.

3.3 Physical NAND block dispatch

When the host issues a write request, we need to choose a NAND block in SSD to store data. The block is chosen based on the predicted temperature T' of the coming data. In this paper, the temperature is predicted in the unit of chunk, so the temperature of data is determined by that of chunk in which the data falls. All chunks will be divided into several temperature ranges. Each range corresponds to a physical block. Thus, chunks with similar temperature will be mapped to the same block, and chunks with large temperature differences are mapped to different blocks.

In each period of time, the next period's temperature of all the chunks has been predicted by LSTM model. In order to gather the chunks with similar temperature, K-Means is adopted to cluster all the chunks according to their predicted temperature. Note that, there are also many algorithms used for clustering. We choose K-Means because of its relatively high efficiency and the ability to output a specific number of clusters. By inputting the predicted temperature of all the chunks and the number of the required cluster *c*, K-Means clusters all the chunks into *c* clusters. As a result, the chunks in the same cluster have the similar temperature, and data falling in these chunks will be dispatched to the same block. For example, chunk *a* is in cluster *A* which is mapped to physical block *I*; then all the write requests falling in chunk *a* will be dispatched to block *I*.

3.4 Implementation of GC improvement scheme

Before prediction, training, whether online or offline, is required to improve the prediction accuracy of LSTM model. In the prediction process, when a time period ends, WFC module outputs $\boldsymbol{rec}_t$, LSTM predicts T'_{t+1} using formula (1), and BD module clusters the chunks according to T'_{t+1}. From then on, the new write requests will be dispatched to blocks according to the new cluster result till the next prediction and clustering.

LSTM and K-Means can be easily implemented by using Python. BD module needs the support of SSD controller. Fortunately, the “Directives and Streams” feature in NVMe 1.3 [13] provides the technical support for dispatch. In this paper, Multi-stream SSD (MS SSD) [14] which satisfies this feature is adopted to implement the dispatch. By giving MS SSD a stream ID along with write request, data will be dispatched to a specified block automatically. In this way, the data with same stream ID will be dispatched to the same block (unless it is full), and those with different stream ID will be dispatched to different blocks. Therefore, after clustering, BD outputs a mapping table where chunks are mapping to stream IDs according to their clusters. Then, the data falling in the chunk will be dispatched to the corresponding stream ID. Since the data stored in a block have similar temperature, the data in the same block has a great probability of being invalid simultaneously. GC will obviously be improved then. The prediction process is shown in Fig. 6.

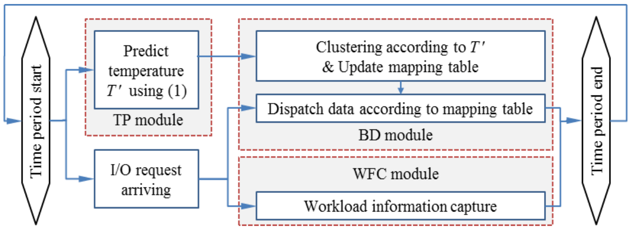

Time period start
Predict temperature T' using (1)
TP module
I/O request arriving
Clustering according to T' & Update mapping table
Dispatch data according to mapping table
BD module
WFC module
Workload information capture
Time period end

Fig. 6. Prediction process

4 Evaluations

In this section, we evaluate the benefit and cost of our proposed scheme in different environments on real SSD. The detailed configuration about the evaluation system is shown in Table 1.

Table 1. Evaluation system configuration

Processor /Memory	**Processor Dual Socket:** Intel (R) Xeon (R) CPU E5-2620 v4 @ 2.10GHz/16 cores **Total Logical CPU:** 32 **Total Memory:** 64 GB **GPU:** NVIDIA GTX 1080, 8G
Operating System	**Distro:** CentOS Linux release 7.5.1804 (Core) **Kernel:** 4.4.2, patched for multi-stream support **Arch :** x86_64
SSD	**SSD:** Samsung NVMe PM963 2.5”, 960GB (Support both “normal” and “multi-stream” mode with 8 streams, with NAND write and host write values in additional S.M.A.R.T)

Since MS SSD has 8 streams, we set the number of the required cluster *c* as 8. Each time period lasts for 5 seconds. The time step *k* of LSTM is 5. LSTM predicts the temperature of 100 chunks each batch. The entire address is divided into 10,000 chunks. Thus, it will take 100 batches for LSTM to predict the temperature of all chunks. Our machine learning algorithm runs on the GPU, and its efficiency will be analyzed in Section 4.2.

4.1 Scheme effectiveness

Our scheme aims to reduce GC overhead. WAF is used to measure the effect of improvement. It is computed as follows:

$$\mathrm{WAF} = \frac{total\_size\_of\_data\_committed\_to\_flash}{total\_size\_of\_data\_\mathrm{received}\_from\_host} \qquad (2)$$

For a fixed total size of data received from the host, a lower WAF means less copy of valid data in GC.

We design four workloads for the test: FIO [15], RocksDB, MySQL and dockerized environment. Besides, we compare our scheme (LSTM+KM) with the other two schemes. The “baseline” is the original scheme that dispatches data without considering temperature. Furthermore, we also compare our scheme with a state-of-art technology called AutoStream [9]. It uses the current temperature as the prediction of the next time period, i.e., $T'_{t+1} = T_t$.

The WAF of each scheme with different applications is shown in Fig. 7. As GC is improved, the performance will also be increased. Different applications have different performance measuring metrics. MySQL returns tpmC (the higher the better). FIO and RocksDB return 99.99% write latency (the lower the better). The performance comparison result is shown in Fig. 8. Note that, baseline is assigned the value 1 to get the normalized results.

FIO is an I/O tool used for benchmark and stress/hardware verification. In this test, several FIOs with fixed IOPS operated simultaneously. Each FIO had its own access area, write queue depth and request size. It can simulate a workload in which different chunks have different temperatures and have a close relationship with its neighbor chunks. And the temperature changes regularly. Fig. 7 shows that, alt-

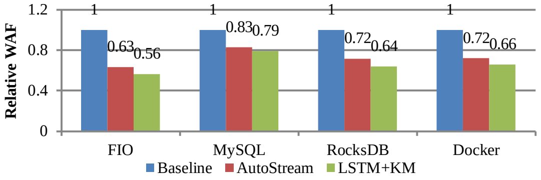


Fig. 7. WAF in different applications

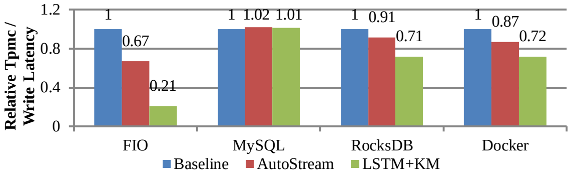


Fig. 8. Performance in different applications

hough the chunk temperature changes frequently, our scheme can predict the change, so WAF is reduced by 43.5% and 10.7% compared with baseline and AutoStream respectively. Furthermore, as shown in Fig. 8, under the high depth write queue pressure, the latency is high in baseline and AutoStream scheme. However, with less data to copy during GC in our scheme, the write latency is improved by 79.3%. These indicate that our scheme makes a significant improvement in the ideal environment.

RocksDB is a widely used open-source distributed NoSQL database. RocksDB generated large sequential I/O requests, and therefore only a few chunks ware accessed at the same time. Fig. 7 shows that the WAF of our scheme is reduced by 35.9% and 10.7% compared with baseline and AutoStream, respectively. This indicates that our scheme can obviously reduce the number of valid page copy in SSD and thus promote GC efficiency. Successively, write latency is also improved by 29% as shown in Fig. 8.

MySQL is a widely used relational database. Most write requests in MySQL are random and distributed through the whole address. As shown in Fig. 7, WAF of our scheme is improved by 21% compared with baseline. But it only achieves 4% improvement compared with AutoStream. The reason is that the temperature of MySQL in each chunk is different but it changes slightly. Thus, dispatching according to temperature can benefit GC, but for the application in which temperature changes slightly, the effectiveness of LSTM will be discounted.

Docker is a computer program that performs operating-system-level virtualization, also known as "containerization". In order to measure the performance of our scheme in complex scenario with heavy workload, we ran 2 RocksDB instances simultaneously with different configurations in Docker. In Docker, applications run independently with each having their own performance. But the WAF of SSD is impacted by all of the applications. As shown in Fig. 7, our proposed scheme still works well under the multi-application environment. Compared with baseline, in this complex scenario with a heavy workload, the WAF of our scheme is still reduced by 34%, along with an average of 28% performance improvement for each application in Docker.

Above all, our scheme achieves an obvious improvement for GC in SSD. Compare with AutoStream, LSTM can predict the future temperature and improve the optimization effect especially for workloads with a great change in temperature.

4.2 Resource consumption

Our scheme will consume some system resources at runtime. It takes less than 80 ms to generate new mapping table from captured features. This mapping table will be used in the next 5 s. Compared with 5 s, the runtime of machining learning is acceptable. The GPU utilization of LSTM is less than 17%. We also test the resource consumption of our scheme by comparing the CPU utilization and memory usage of our scheme to baseline.

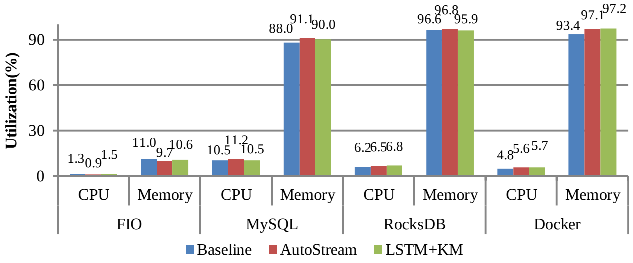


Fig. 9. Resource consumption of CPU & memory

Fig. 9 shows the CPU utilization and memory usage of our scheme in the three cases during the test, which indicates that our scheme consumes almost as many resources as legacy SSD with the difference within 5%.

5 Conclusion

To better adapt to the application scenarios with temperature changes, a scheme to predict the future temperature by using LSTM is proposed in this paper. The scheme employs K-Means to dispatch chunks to different blocks according to the predicted temperature. LSTM increases the prediction accuracy, and K-Means clusters chunks in a more reasonable way. As a result, in different types of workload scenarios, the reduction of WAF illustrates the improvement of GC, and I/O performance is also improved meanwhile. In the future, we will focus on: 1) improving the efficiency and accuracy of our proposed machine learning models, and 2) optimizing the scheme of managing the NAND blocks with the help of host FTL technology [16].

Acknowledgments

We sincerely thanks our project members, Chunchao Ye, Pengli Ju, Lei Geng for their awesome support and the reviewers, Jungmin Seo and Chunyu Guo, for their helpful advice. We also thank for the help of Avani Wildani.

Discussion

This paper tries to reduce GC overhead by dispatching data according to their temperatures. This is also an attempt to enhance SSD using machine learning. Training an ideal model with high efficiency and accuracy is always an open problem. We have done some efforts on that. For example, as we have mentioned in this paper, we selected features correlated to temperature, tried to combine chunks with same IO pattern, and inputted chunks of temperature in batches. These can benefit the test results, but they still require in-depth research. We also tried different LSTM models (with different number of hidden layers and different number of nodes in a layer), different model parameters (e.g., time step, learning rate), and different algorithms of normalization and gradient descent. Due to the limitations of the paper, the details are not shown. And we believe that there are many details that can be optimized to get a better machine learning model.

Another problem is about the block dispatch. There is no commercial SSD supporting host to choose a block for writing currently. We use multi-stream SSD instead of simulators to test in orter to get a more convincing result. With this SSD, host can choose a group of blocks for writing. SSD will choose the specific block for host. At least, it ensures that different data with different Stream ID be dispatched to different blocks. But the number of stream is limited. And how to choose block is a black box. So we are researching on dispatch block using host FTL technology, such as Open Channel SSD. The host FTL has been added to the Linux kernel with an exquisite design (including block dispatch) to achieve high parallelism. Thus, how to realize block dispatch for our scheme without loss parallelism is still a difficult task.

At present, our program still has a lot of room for improvement. We expect to share our experiences with researchers and look forward to gaining more advice about how to improve our scheme.

References

[1] M. Wu, and W. Zwanepoel, "eNvy: a non-volatile main memory storage system," In *Proceedings of the 6th International Conference on Architecture Support for Programming Languages and Operating Systems*, pp. 86-97, 1994

[2] Zhang, Q. et al. Lazy-RTGC: A real-time lazy garbage collection mechanism with jointly optimizing average and worst performance for NAND flash memory storage systems. *ACM Trans. Des. Autom. Electron. Syst.* **20**, 1-32 (2015)

[3] Kim, T. et al. PCStream: Automatic Stream Allocation Using Program Contexts. HotStorage (2018)

[4] Jang, K. H. & Han, T. H. Efficient garbage collection policy and block management method for NAND flash memory. In International Conference on Mechanical and Electronics Engineering (2010)

[5] E. Rho, K. Joshi, S. Shin, N. Shetty, J. Hwang, S. Cho. and D. Lee. FStream: Managing Flash Streams in the File System. In *Proceedings of the 16th USENIX Conference on File and Storage Technologies* (*FAST*'18), 2018.

[6] Park, D. & Du, D. H. C. Hot data identification for flash-based storage systems using multiple bloom filters. In *MSST* (2011)

[7] Stoica, R. & Ailamaki, A. Improving flash write performance by using update frequency. *Proc. VLDB Endowment* **6**, 733-744 (2013)

[8] Luo, Y. et al. Warm: Improving NAND flash memory lifetime with write-hotness aware retention management. In *MSST* (2015)

[9] Yang, J. et al. AutoStream: Automatic Stream Management for Multi-streamed SSDs. In *SYSTOR* (2017)

[10] Wikipedia, Pearson correlation coefficient, https://en.wikipedia.org/wiki/Pearson_correlation_coefficient (2018)

[11] Hochreiter, S. & Schmidhuber, J. Long short-term memory. *Neural Computation* **9**, 1735–1780 (1997)

[12] V. Tarasov. et al. Extracting Flexible, Replayable Models from Large Block Traces. In *FAST*(2018)

[13] http://nvmexpress.org/wpcontent/uploads/NVM_Express_Revision_1.3.pdf

[14] Kang, J. U. et al. The multi-streamed solid-state drive. In *HotStorage* (2014)

[15] https://github.com/axboe/fio

[16] Matias Bjørling, Javier Gonzalez, and Philippe Bonnet. Lightnvm: the linux open-channel SSD subsystem. In *15th USENIX Conference on File and Storage Technologies (FAST 17)*, 359–374. Santa Clara, CA, 2017. USENIX Association.