Research Journal of Applied Sciences, Engineering and Technology 7(21): 4574-4583, 2014
DOI:10.19026/rjaset.7.836
ISSN: 2040-7459; e-ISSN: 2040-7467

Submitted: January 13, 2014 Accepted: February 25, 2014 Published: June 05, 2014

Research Article

SSD Aware File System Page Cache Algorithms Designs and Implementation to Increase Switch Merge and Reduce Full Merge

[1]Arul Selvan Ramasamy and [2]Porkumaran Karantharaj
[1]Computer Science and Engineering, Anna University, Chennai, India
[2]NGP Institute of Technology, Coimbatore, India

Abstract: The NAND based SSD is a tremendous challenge for file system page cache management design as it possess inbuilt imbalance in reading and writing speeds. The present disk-based page cache management policies adopt strategies to maximize the page hit ratio and treat the cost of page read and writes are equal rather than fully exploiting the characteristics of the SSD. Therefore there exists slowness in SSD write which eventually affects the SSD endurance. In special cases like in random writes there exists a problem of augmented write amplification resulting in penalty of write performance. Therefore, a new page replacement algorithm is needed for storage systems based on NAND flash memory. The aim of this study is to propose a new page replacement algorithm for operating systems which focuses on reducing the replacement cost and I/O execution time. Trace-driven simulations show that the proposed algorithm performs better than existing algorithms in terms of the replacement cost and I/O execution time.

Keywords: Data clusters, enlarged write, full merge, random write, switch merge, write amplification, write back cache

INTRODUCTION

Flash memory has evolved into cost-effective and potent solid-state storage technology as an alternative to disk based storage solution. Commercially flash is of two types NOR and NAND. NOR-this type of flash memory supports the byte unit I/O and shows longer write time and shorter read time in comparison with NAND. NAND-this flash memory helps in supporting the page unit input/output and this furnishes the block mode interface. NAND provides faster write time and slower read time compared to NOR and is prominently used for the purpose of data storage which acts as an alternative for hard disk drives. In this chapter the characteristics of NAND flash based SSD and an overview of FTL and its related work is presented.

Flash memory characteristics: NAND based SSD consists of set of blocks which are fixed in number and each block comprises of a fixed set of pages or whole pages set makes up a block. Typical page size is 4 KB and block size is 512 KB. In SSD the tiny unit which is addressable is known as sector or page. Generally NAND flash memory comprises of three operations they include read, program or write and erase. The operation read conveys the data from a page, program operation helps in writing the data on a page and erase operation readjusts the target block values to one. Execution of operations read and write takes place per page level. Once after writing on the page it should be erased before the next writing operation continues on the same page. On the other hand the data is erased on block level by using erase operation and this is termed as erase block which is smallest erasable part in SSD. The erase operation should be executed before rewrite. Since the write and erase operations accompany each other the access time is high.

In write procedure, Flash memory must first be erased before it can be rewritten with new data. In order to write a page of data, the SSD's NAND Controller will first ensure that the destination block is erased, before it writes the new data. So the other pages in the block have to copied and rewritten with the new page data. This erase/write cycle of the entire block occurs even if only a single bit changes in the block. This is known as write amplification (Gal and Toledo, 2005). The definition of write amplification is as follows:

> "Write Amplification = Actual no. of page writes/No. of user page write"

The following operations should be followed in SSDs to write a single page or sector. The erasure block should be read which stores a requested sector and changing the block with requested sector and write the modified block. This sequential operations are termed as read modify write operations. This is commonly seen

Corresponding Author: Arul Selvan Ramasamy, Computer Science and Engineering, Anna University, Chennai, India

in random write requests. Write amplification is observed during the garbage collection process.

practiced on SSD by understanding the characteristics. However the dependency of a particular file system becomes mandatory and in this approach the performance and endurance of SSD is not addressed completely.

But the updated high performance SSD's, comprises of a program known as Flash Translations Layer (FTL) which helps in managing the flash memory. The FTL possess following characteristics, It abstracts the hardware features of device therefore it is called as pseudo file system. It presents block device (Intel Corporation, 1998) interface by hiding the hardware characteristics of NAND device and the file system consumes the interface. FTL maps the logical address space into physical locations of the flash memory is done by FTL. It also helps in operations like garbage collection and wear leveling. For device endurance and high performance read and write cycles diverse algorithms are required and FTL contains these algorithms. The basic function of FTL algorithm is to map the page number from logical to physical and these mapping entry data is preserved in random access memory. The FTL address mapping procedures are classified into three categories namely, block level mapping, page level mapping and hybrid mapping.

Hybrid mapping is used in the recent FTL's which uses both page and block mapping techniques to create a hybrid FTL. By doing this the random access memory usage of page mapping can be minimized and there will reduction in erase counts related to block mapping. The example for this is log block based FTL. This executes through a principle of writing a page into a log block by using page mapping and allows the mapping of the page anywhere inside the block. The action of garbage collection enters the role when the log block is completely filled. The victim block is selected by the FTL and it merges with corresponding data blocks. Depending upon the situation the merge operation is generally of three types such as switch, partial and full merge.

When complete pages in the data block are invalidated, the log block moves to the new data block and erases the old one then switch merge arises (Fig. 1). Partial merge arises (Fig. 2) if every valid page is to be copied to the remaining empty block places in the log block. Full merge (Fig. 3) needs 2 block erasures like 'n' page writes and 'n' page reads. The synonym for full merge is normal merge. This merge assigns a free block which is erased before and copies the recent updated valid pages into this free space. These valid pages are collected either from log block or data block. Once after completion of copying the free block is replaced by data block and the log blocks and previous data blocks are erased.

Flash translation layer: The device hardware features should be understood by the file system for data storage and retrieval from SSD. File systems such as logFS was

Though hybrid FTL is advanced compared with block based or pure page FT, still the performance is poor due to read, write and erases merge operations for random writes (Min *et al*., 2012). For better scale up of flash potential this poor performance should be addressed. Advanced research has mainly focused on the improvement of random write performances of flash by the addition of DRAM-backed buffers and also on implementation of buffering write requests by employing page cache algorithms.

Garbage collection and wear leveling: In the flash data delete operation, data is obsoleted not deleted. Obsolete data still occupies storage and cannot be deleted alone in the same erase block. Therefore a garbage collector is required to clean an erase block by moving all valid data into a free erase block, obsoleting old erase block. For the preparation of new incoming data, the data which is valid is separated from the invalid data and is moved to an unoccupied storage area. As a result the invalid data containing block is erased. This process of collecting, moving of valid data and erasing the invalid data is called as garbage collection. Through SSD firmware command TRIM (Shu and Obr, 2007) the garbage collection is triggered for deleted file blocks by the file system. To communicate that a range of logical pages usage is no longer an ATA inter face standard called TRIM is used. This garbage collection used for flash file systems is similar to that of segment cleaning method in log structured file systems (Rosenblum and Ousterhou, 1992). Although garbage collection enhances the performance but it involves an additional cost of computation and live data migrations.

Commonly used erase blocks puts off quickly, slows down access times and finally burning out. Therefore the erase count of each erase block should be monitored. There are wide number of wear-leveling techniques used in FTL to address this problem.

Motivation and contribution: SSD write performance and device endurance is affected by the write amplification. Most of the file systems perform sub-optimally when running on top of SSDs with FTL technology. This suboptimal performance is attributed by the poor random write (Qiu and Reddy, 2013) performance in SSDs. Particularly SSD performance gets affected on random write workloads on overwrite file systems. Random writes also shortens the lifespan of SSDs as these writes include more block erases for every write. So designing the operating system's page replacement algorithm is very crucial for SSD writes. Especially for over write file systems, the host based page replacement algorithm for SSD must be designed factoring the write amplification scenario. However,

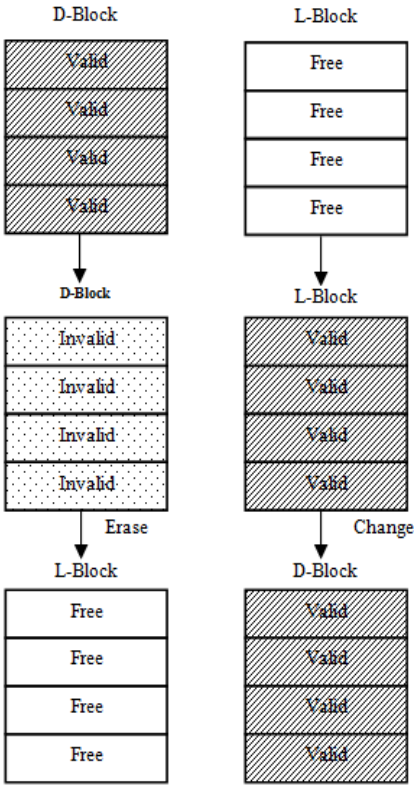

Fig. 1: FTL switch merge-one erase operation

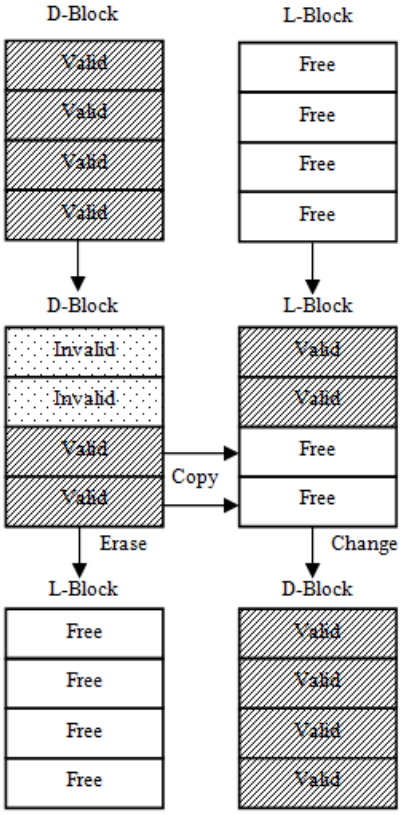

Fig. 2: FTL partial merge-one erase operation+live data copy

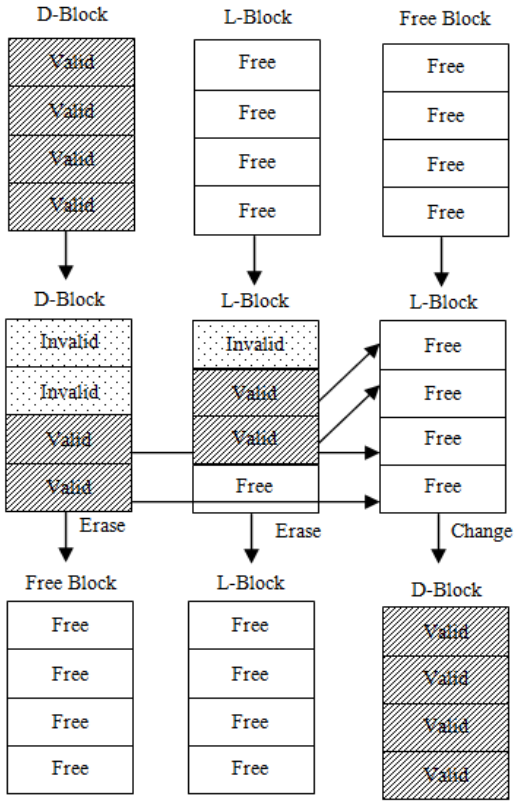

Fig. 3: FTL full merge-two erase operations+live data copy

this new cache replacement method should not affect the host cache expulsion rate which leads to over or under use of file system page cache. Hence, the flash-aware host based page cache replacement algorithm should drive the FTL for the probability of reduced partial, full merges and increased switch merges.

We summarize the basic principles of the proposed flash-aware host based page cache replacement algorithm as follows. The algorithm drives the FTL for the increased probability converting full and partial merge to switch merge; this is the design construct for our algorithm. The following are the design principles of the algorithm:

- Cluster the dirty pages for write efficiency
- Convert the random work load into sequential workload through write enlargement
- Clean page eviction policies to enhance the spatial locality and to support the write enlargement

This algorithm maximizes the present LRU (Least Recently Used) algorithm's efficiency. Depending upon the design principles various cache management strategies are developed and implemented on the LRU algorithm. K-means (Lloyd, 1982) clustering algorithm is used for dirty page clustering to form write clusters. Generally, in regular application SSD's need sequential work load for high write throughput and it is difficult to assure in real time. Hence a set of cache management

policies are developed to make sure that the writes are sequential in nature. The probability of page writes targeting on a specific SSD erasure block is increased by applying these policies. The proposal is termed as CSWLRU, which means Clustering dirty pages and Sequential Write formation on Least Recently Used. The results were promising on implementation of this CSWLRU in a trace driven simulation environment compared to the previously proposed algorithms.

LITERATURE REVIEW

CFLRU Park *et al*. (2006) and LRUWSR (Jung *et al*., 2008) are based on modified form of LRU algorithm and is a flash aware page replacement strategy for operating systems. In CFLRU algorithm, the LRU list is divided into two parts. In clean-first region, the clean pages are selected as victims over dirty pages. Only in the absence of clean pages in clean first region the dirty pages are selected as victim pages. The buffer place for dirty pages is effectively increased by evicting the clean pages. Hence the flash write number reduces. LRUWSR uses only a single list as auxiliary data structure. The scheme of this LRUWSR is to maintain hot-dirty pages as buffer and eject the cold-dirty pages. If there is a necessary for victim page, the search begins from LRU end of the list. If a clean page is visited, it will be returned immediately (LRU and clean-first strategy). If a dirty page is visited and is marked "cold", it will be returned; otherwise, it will be marked "cold" and the search continues. This strategy increases the page read count but reduces the number of page writes. As a result this algorithm improves the overall performance of SSD. CFLRU and LRU-WSR do not address the write patterns.

The order of write is important to reduce the average cost of each write. WOW (Gill and Modha, 2005) groups the pages in write groups and sorts them in logical order based on the physical proximity on the disk. WOW solves the write order and STOW (Gill *et al*., 2009) solves the write pace problem. STOW algorithm exploits the spatial and temporal locality by partitioning the dirty pages into sequential and random queue. By doing so it provides better control on page eviction and minimizes the one time sequential write cache pollution. Both WOW and STOW is proposed for the write performance in disk based storage system.

In Btrfs (http://btrfs.wiki.kernel.org.) and WAFL (Hitz *et al*., 1994) file systems, always the dirty pages are written into new pages. Btrfs uses TRIM command for reporting the free blocks to the firmware and it is a SSD aware file system. Btrfs has effective optimizations like omitting the unnecessary seeks and writing group of dirty pages, even if they are from unrelated files. The outcome is faster write throughput and larger write operations.

JFFS2 Woodhouse (2001) is a log-structured file system designed and optimized for raw flash devices and not FTL aware. JFFS2 is aware of the restrictions imposed by flash technology and which operates directly on the flash chips, thereby avoiding the inefficiency of having two journaling file systems on top of each other. IotaFS (Cook *et al*., year) is SSD optimized file system which places the random write into convenient write locations to lessen the impact of erase block copying overhead. In addition to this allocation is done on erasure block units to achieve spatial locality of writes for high throughput. FAB (Jo *et al*., 2006) is one more buffer cache management policy used for flash memory. In this the buffers of the same erase blocks are grouped together on LRU sequence and victim groups are larger which are based on the page count on every group. In sequential writes FAB plays an effective role.

CDFC (Ou *et al*., 2010) and CASA (Ou and Harder, 2010) are improvements of the CFLRU algorithm. Clean-First Dirty-Clustered (CFDC) is a flash aware buffer management algorithm which gives first priority for clean buffer pages in replacement process. The dirty pages are grouped for better spatial locality of the page flushes. This algorithm was incorporated with conventional page replacement algorithm and evaluation of performance is done in data bases. CASA improves CFLRU by automatically adjusting the size of buffer portions allocated for clean and dirty pages according to the storage device's read/write cost ratio.

DESIGN OF CUSTERING AND COMPOSING SEQUENTIAL WRITE ON LEAST RECENTLY USED

CSWLRU is a host based page cache management scheme, host consists of CPU, a file system and the system page cache. Figure 4 shows the general system configuration considered in this study.

CSWLRU is combination of three techniques namely, dirty page clustering, enlarged write through read ahead write cache and frequency and distance based clean page eviction.

CSWLRU K-means page clustering: Clustering is an important tool for automated analysis of data. CSWLRU utilizes K-means clustering technique on dirty pages. Clustering dirty pages with large granularity will help to convert the random work load into sequential workload. For a given set of 'n' data points, the algorithm partitions the points into 'k' clusters. In k means clustering algorithm, initially the number of clusters known as 'k' value should be specified. After the 'k' value is determined, these 'k' points are selected as cluster centers. According to Euclidean (Aloise *et al*., 2009) distance metric all instances which are needed to be divided are assigned to their nearest cluster centers.

Lloyd's algorithm (Lloyd, 1982) is the most accepted method used for calculating k-means. One dimensional dirty page of length 'n' acts as input for this algorithm. The initial 'k' value or partition boundary is calculated based on the erasure block size.

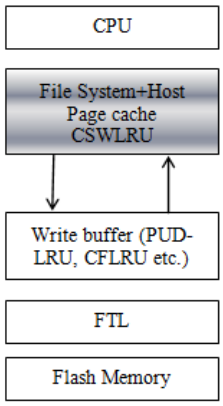

Fig. 4: System configuration CSWLRU is applied to the host file system

This indicates the number of cluster means production which translates to the number of clustered writes on SSD. In each and every iteration, the continuous k-mean algorithm examines only a random sample of data points where as CSWLRU examines all data points in sequence. The algorithm is in following flow:

- One dimensional pages of length 'n' is the input and 'k' indicates the number of cluster means produced.
- Choose 'k' values from the 'n' dirty pages with the block boundary heuristic mentioned above. Ordering and selecting the initial k values shorten the conversion time.
- As these values are potential means they are named as k-means.
- Further, using square distance method each block is grouped with the nearest mean value.
- By the end of this step, all the data points belong to potential mean group.
- At this iteration, the new mean is derived by calculating the average of all data points.
- The process is repeated until no change is observed in the centroid.

After a few number of iterations, the squared error stops to decrease prominently or is unchanged. Below is the example:

Input values {1, 3, 5, 12, 52, 25, 13, 571, 834, 432, 489, 8, 45, 60, 2, 11}
k = 2→ {432, 489, 571, 834}, {1, 2, 3, 5, 8, 11, 12, 13, 25, 45, 52, 60}
k = 3→ {834}, {432, 489, 571}, {1, 2, 3, 5, 8, 11, 12, 13, 25, 45, 52, 60}

Clustering the dirty pages increases the probability of switch merges. CSWLRU uses the k-means algorithm by examining all dirty pages and produces the dirty page clusters.

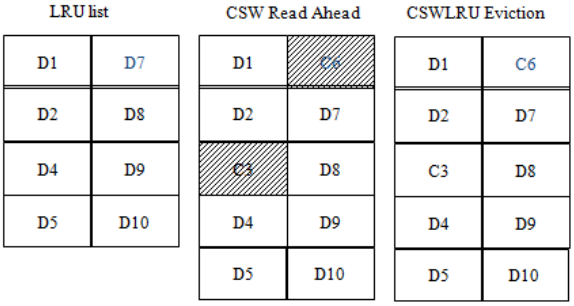

Fig. 5: An example of read ahead cache. The clean pages C3 and C6 are read from SSD to form a sequential write. C-clean page D-dirty page

CSWLRU-read ahead write cache: K-means clustering produces clusters of dirty pages and most of the time the clustered pages are not continuous. For accelerating the switch merges count, the pages should be adjoining and arrayed with SSD erasure block borders. For the formation of continuous writes, the clean pages are brought into cache after reading from SSD. Or else, relatively expensive full merge or partial occurs instead of a switch merge. As the pages are read and cached for write it is named as read ahead write cache or enlarged writes. Page cache algorithm is also implemented by FTL for detecting the scope of erasure block cache based on few set of rules for example last page write in the erasure block or sequential write operations. These set of rules helps in determining the life time of erasure pages cache. Based on cluster density the decision of reading the pages from SSD is determined for a cluster sequential write formation.

In the Fig. 5, since the clustering of dirty pages does not contain the clean pages therefore the c3 and c6 pages are read from SSD to form a sequential write. Though pages c3 and c6 are clean, these pages are once again written into SSD along with neighboring dirty pages. The write time is optimized to best by bringing the clean pages from SSD, resulting in switch merge increase. As the switch merge count increases there would be increase in endurance, performance and reliability of the device. Few FTL algorithms decide the switch merge by comparing the clean to dirty page ratio along with the weal level information such as erase latency or erase count which is stored on a different erasure block. If the clean page falls on the adjacent erase block, FTL can choose to compare the hash code of the existing page and the incoming page (Chen *et al*., 2011) to decide on the write.

Similarly, there is another method called as block padding (Kim and Ahn, 2008) for improving the block utilization was proposed. All the pages in buffer cache are managed by block padding by using block level LRU policy in FTL. The block which was not accessed since long time was chosen as victim block. Through block padding BPLRU invokes switch merge. On the other hand, page recency is the problem of BPLRU. In this instance if one of the pages in a block has high

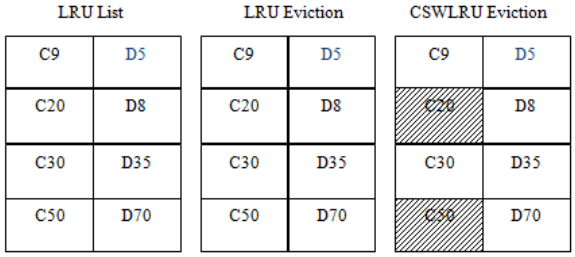

Fig. 6: An example of distance based eviction. For the clean page eviction, the distance is calculated on both directions to find the victim page

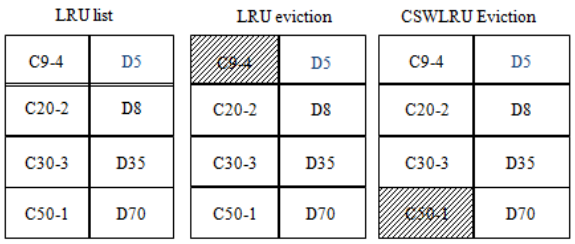

Fig. 7: CSWLRU*:* frequency based clean page eviction. In LRU based eviction clean page C9 is evicted. In CSWLRU based eviction clean page C50 is evicted because the write count is less compared to other pages

percentage of recency the other pages tend to stay in the buffer cache even though the pages were not used recently. This occupies the buffer cache space. Though BPLRU increases the switch merge, the scope is determined by the FTL buffer size. The FTL buffer management are influenced by the host file system during frequent flush command. In enlarged write the cache buffer size is huge compared to the cache buffer inside the SSD. And the clean page padding is done on individual clusters. On individual clusters the clean page padding is performed. Only on high density clusters the enlarged write through read ahead write cache gets activated. This assures for less read count although the price of random read is low on SSD.

CSWLRU-distance based clean page eviction: LRU policy is not very effective for sequential writes. Adaptive replacement cache algorithms (Megiddo and Modha, 2003) respond dynamically to the changing access patterns based on frequency and recency. These replacement algorithms replace the clean pages based on the usage pattern. For instance MRU based algorithm replace the new page while LRU based algorithm replaces the older one. CSWLRU avoids certain clean page eviction to compensate for the enlarged writes. Depending on the distance from the dirty page the clean pages are evicted. If a clean page stays within the erasure block from any dirty page, then the clean page is retained or else it is evicted. For decision making on page eviction, the basic SSD erasure block size is necessary.

In the above example (Fig. 6), it is considered that SSD erasure block comprises of ten pages. Although the clean page c9 is on top of the eviction list, it is retained because the dirty page d4 lies under the erasure block by keeping the clean page as centroid. In the same way page c30 is also retained as the dirty page d35 falls under the erasure block boundary for page c30 as a centroid. Clean pages c20 and c50 are considered as victim pages and they are evicted. Through the property of avoiding clean page eviction, the SSD random writes are converted into sequential writes by writing the clean pages along with dirty pages. This result in conversion of partial merges into switch merges in FTL.

CSWLRU-frequency based clean page eviction: The write frequency of the clean page is monitored at times when write enlargement writes clean pages along with dirty pages. More frequently written clean pages are retained in contrast to the distance based eviction. It is expected that the adjacent pages would become dirty soon. This retention property addresses the clean page management for slow sequential writes.

In the Fig. 7, LRU based eviction the clean page c9 is evicted whereas in CSWLRU the clean page c50 is evicted because it has less write count when compared with other clean pages.

PROTOTYPE IMPLEMENTATION AND EXPERIMENT RESULTS

Thus far, we have discussed the CSWLRU algorithm in abstract terms. This section evaluates the performance of the proposed CSWLRU algorithm in a trace driven simulation environment. In this section the performance of proposed CSWLRU algorithm is evaluated in a trace driven simulation environment. For the comparison, we have evaluated the LRU, CFLRU, CFDC, FAB with the proposed CSWLRU algorithm.

Experimental setup: For evaluating the performance characteristics of CSWLRU algorithms, a trace driven simulation environment is constructed. The goal of this stimulator is to evaluate the performance of CSWLRU write back caching algorithm on SSD for several random workload traces. In C language on the Linux platform the page cache simulator is implemented as a separate tool. This stimulator accepts the trace file as input and produces series of write/read requests. The I/O requests are represented in pair of values by trace file. The configuration value for total number of pages is taken up by the page cache simulator. On arrival of new page write, the LRU procedure removes the least recently used frame when cache is filled. For the implementation of LRU cache two data structures are used. Utilizing a doubly linked list a queue is

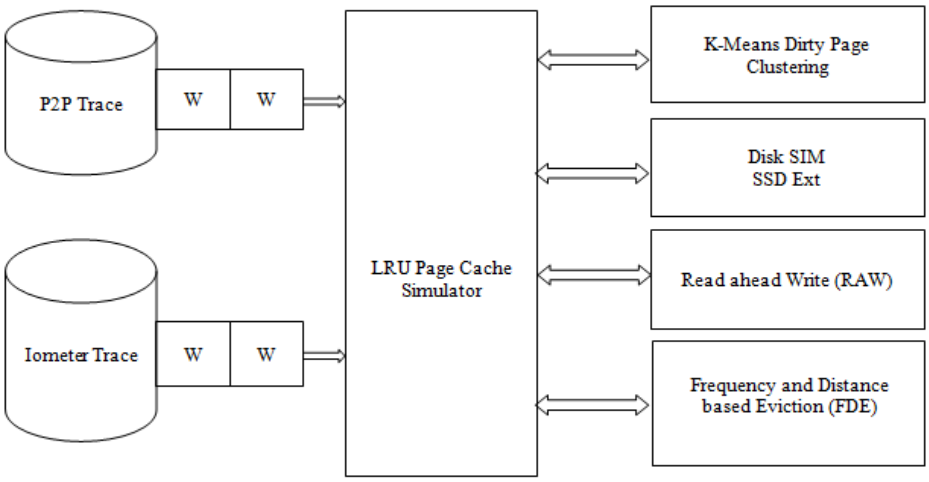

Fig. 8: Simulation framework

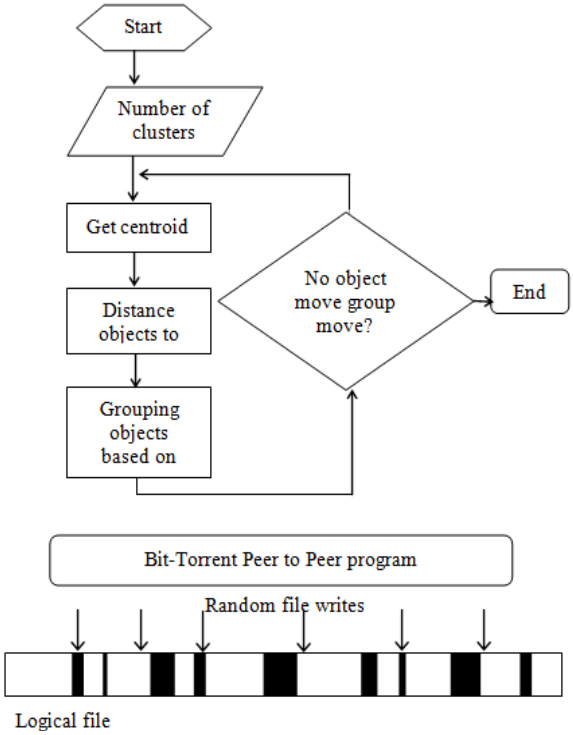

Fig. 9: P2P file sharing trace collection

implemented. The utmost size of the queue would be equal to the total number of available frames (Fig. 8).

For SSD simulation environment, the DiskSim (Microsoft, 2010) Microsoft® Research SSD extension was used. DiskSim is well modularized and establishes the major components of FTL such as indirect mapping, garbage collection and wear leveling strategies. In order to provide accurate timings for the handling of storage I/O requests, the DiskSim-SSD is incorporated into a full system level simulator. As the DiskSim extension does not contain inherent cache support, on receiving the write request it is immediately written into SSD.

Write trace collection: The first trace is acquired using DiskMon (Russinovich, 2006). The trace file obtained from DiskMon is not applicable directly to CSWLRU algorithm because the program's output has several features which are not necessary for the simulator. The raw outcome of DiskMon gives offset of a file but not actual sector address of the file. Current application consults Master File Table in NTFS for locating the sector address of the file. Cache manager reads the IO trace file and optimizes several cache algorithms in order to provide insights in designing cache policies for SSD bases storage. Peer-to-peer file sharing generates random write traffic to storage due to the nature of swarming. We collected disk accesses on Windows 7 with Diskmon while downloading a 739 MB movie file using µTorrent, a P2P file sharing program. Small parts of the file are written randomly to the storage because the peer-to-peer program downloads different parts concurrently from numerous peers (Fig. 9). We used an empty E drive while Windows 7 was installed on C drive to filter out unrelated disk accesses to our test. Before the movie file download, we formatted the 'E' drive with NTFS to get rid of disk aging effect.

Second trace is acquired using Iometer Project (Year) (http://www.iometer.org.) on Linux Ext4file system for producing homogeneously distributed random access. It creates large file with complete partition size and then overwrites the sectors randomly. 100% 4K write and 60% random workloads were created for this test. The Iometer access specification ensures the alignment of IO's on a 4 K boundary.

Work flow and performance evaluation: The performance results of CSWLRU algorithms were presented in this part. Initially, the workflow in the simulation environment and secondly evaluation results for identified workload traces were explained.

The LRU cache manger was implemented using double linked list. Double linked list represents the

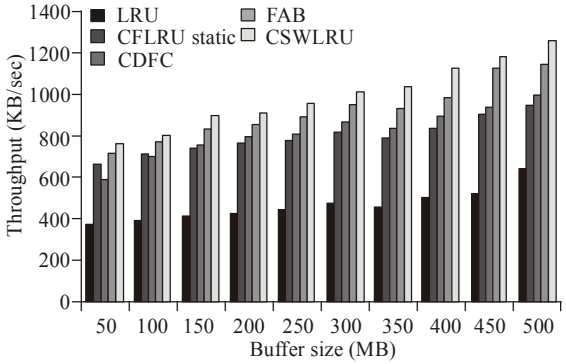

Fig. 10: Iometer mixed workload

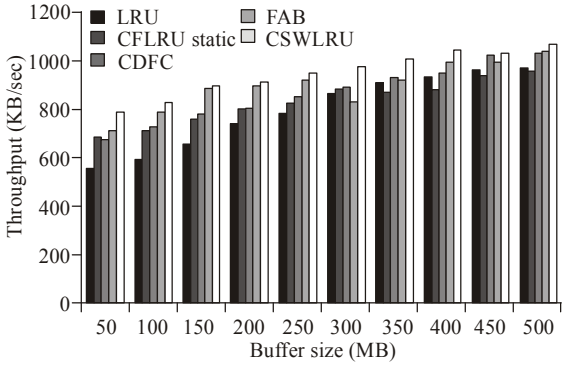

Fig. 11: Iometer sequential workload

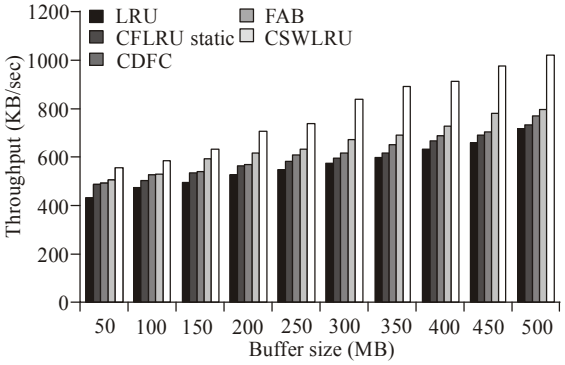

Fig. 12: Iometer random workload

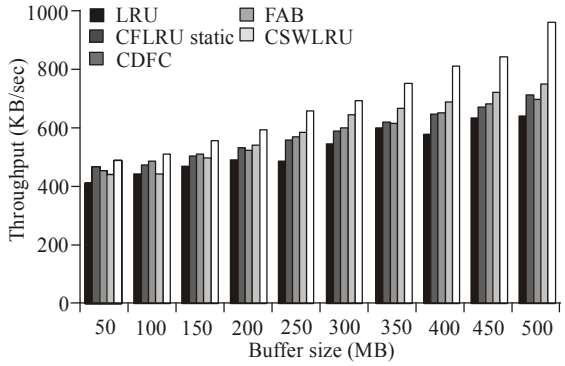

Fig. 13: P2P random workload

page cache lines sequence in an ascending order of their most recent access times. The head node represents the least recently accessed dirty cache lines while the tail node represents the most recent one. Each cache line comprises of an additional field for saving the pointer to corresponding node in the list to support the double linked list. Finally, the implementations such as clustering algorithm, Read Ahead Write (RAW) policy, Frequency and Distance based Eviction (FDE) took place. These clustered pages are inputted to the read ahead write policy to pad the clean pages from the SSD.

The clean page padding from the SSD is decided by the cluster density. For padding the clean pages clusters of high density with multiples erasure block size are given preference. Calculation of cluster density is as follows:

- Order the cluster elements
- Computer the density using (last element-first element) /total elements
- Clusters are ordered based on the density factors and average wait time

We kept the page size of 4096 bytes of maximum cache size of 512 MB and simulated a 9.1 GB SSD. Parameter for the Seagate Cheetah9LP disk, which is of size 9.1 GB, is used as this is the largest disk that can currently be simulated by DiskSim. DiskSimSSD was driven by write request traces by the cache manager (Fig. 10 to 13).

DISCUSSION

The result of simulation performance shows the improved writes on the selected workload traces. From the performance it is obvious that the sequential workload performs good in majority of the previously proposed algorithms, however CSWLRU outperforms for mixed and random workload. In today application the disk scheduler and FTL reorders the request of random writes for write optimization. But the prediction of workload knowledge is limited and optimizations such as enlarged writes are not feasible. Hence the file system layer is more appropriate for random SSD write optimization to drive FTL away from full merge. At the time of FTL full merge the logical page offset would not be same as physical page offset. In partial and switch merge cases, the logical page offset is equal to that of physical page offset. The reason behind this is switch and partial merge does in place update. Generally in Meta data updates the page level mapping operation dominates. If the logical page number is not equal to that of physical, then proposed algorithm could not yield required result. Although the speed of random read is better in SSD, it is suggested for allocation of

user data pages in sequential manner in multiples of erasure blocks. This is done to deflect the full merge which is caused by random updates. Usually, the cluster page selection should be done based on the high priority pages, but for the sake of simplicity the dirty page priority was not factored.

CONCLUSION

Flash based storage devices were considered as an alternative to hard drives. Enhancing the performance of SSD is a challenging task with traditional file system which is not designed for SSD characteristics. For complete exploitation of performance benefits in SSD based storage system, there is necessity for a host based buffer replacement policy for asymmetric write workload. The buffer replacement algorithms like LIRS, ARC and LRU cannot deal effectively with the SSD random write requests. In SSD the read operation is faster when compared with write and erase operation. For the improvement of overall performance of a flash memory system, the random write workload should be reduced by the buffer replacement algorithms.

This paper studies on cache policies for assisting the discussed write back design values. CSWLRU, a new host based page cache replacement algorithm was proposed for SSD. This algorithm assures the conversion of random writes into sequential writes for SSD. Using two kinds of traces, the trace-driven simulation was performed which represents the random write patterns. It was demonstrated that the SSD write performance is greatly enhanced by the proposed write cache policies in host file system. Based on the results of trace-driven simulation experiments, it was proved that CSWLRU algorithm improves the overall performance prominently when compared to the previously proposed algorithms. This was achieved by reducing the number of physical SSD writes and erases operations.

REFERENCES

Aloise, D., A. Deshpande, P. Hansen and P. Popat, 2009. NP hardness of Euclidean sum-of-squares clustering. Mach. Learn., 75(2): 245-248.

Chen, F., T. Luo and X. Zhang, 2011. CAFTL: A content-aware flash translation layer enhancing the lifespan of flash memory based solid state drives. Proceedings of the 9th USENIX Conference on File and Storage Technologies. San Joes, USA.

Cook, H., J. Ellithorpe, L. Keys and A. Waterman, Year. Iotafs: Exploring File System Optimizations for SSDs. University of California at Berkeley.

Gal, E. and S. Toledo, 2005. Mapping structures for flash memories: Techniques and open problems. Proceedings of the IEEE International Conference on Software Science, Technology and Engineering, pp: 83-92.

Gill, B.S. and D.S. Modha, 2005. WOW: Wise ordering for writes-combining spatial and temporal locality in non-volatile caches. Proceeding of the 4th USENIX Conference on File and Storage Technologies (FAST'05), pp: 129-142.

Gill, B.S., M. Ko, B. Debnath and W. Belluomini, 2009. STOW: A spatially and temporally optimized write caching algorithm. Proceedings of the USENIX Annual Technical Conference (USENIX'09), pp: 26-26.

Hitz, D., J. Lau and M. Malcolm, 1994. File system design for an NFS file server appliance. Technical Report 3002, Proceeding of USENIX Winter Technical Conference. USENIX Association, Berkeley, CA, USA, pp: 19-19.

Intel Corporation, 1998. Understanding the Flash Translation Layer (FTL) Specification. Application Note AP-684.

Iometer Project, Year. iometer-[user-devel] @ lists. sourceforge.net. Iometer Users Guide. Retreieved form: http://www.iometer.org.

Jo, H., J.U. Kang, S.Y. Park, J.S. Kim and J. Lee, 2006. FAB: Flash-aware buffer management policy for portable media players. IEEE T. Consum. Electr., 52(2): 485-493.

Jung, H., H. Shim, S. Park, S. Kang and J. Cha, 2008. LRU-WSR: Integration of LRU and writes sequence reordering for flash memory. IEEE T. Consum. Electr., 54(3): 1215-1223.

Kim, H. and S. Ahn, 2008. BPLRU: A buffer management scheme for improving random writes in flash storage. Proceedings of the 6th USENIX Conference on File and Storage Technologies (FAST'08), pp: 239-252.

Lloyd, S.P., 1982. Least squares quantization in PCM. IEEE T. Inform. Theory, IT-28(2).

Megiddo, N. and D.S. Modha, 2003. ARC: A self-tuning, low overhead replacement cache. Proceeding of 2nd USENIX Conference File and Storage Technologies (FAST). San Francisco, CA, USA.

Microsoft, 2010. SSD Extension for Disksim Simulation Environment. Retrieved form: http:// research. microsoft.com/en-us/downloads/b41019e2-1d2b-44d8-b512-ba35ab814cd4.

Min, C., K. Kim, H. Cho, S. Lee and Y. Eom, 2012. SFS: Random write considered harmful in solid state drives. Proceedings of the 10th Conference on File and Storage Technologies in FAST'12.

Ou, Y. and T. Harder, 2010. Clean first or dirty first?: A cost-aware self-adaptive buffer replacement policy. Proceeding of the 14th International Database Engineering and Applications Symposium, ACM, pp: 7-14.

Ou, Y., T. Härder and G.P. Jin, 2010. CFDC: A flash-aware buffer management algorithm for database systems. Proceedings of the 14th East European Conference on Advances in Databases and Information Systems (ADBIS'10).

Park, S., D. Jung, J. Kang, J. Kim and J. Lee, 2006. CFLRU: A Replacement Algorithm for Flash Memory. Proceeding of International Conference on Compilers, Architecture and Synthesis for Embedded System (CASES, 2006), pp: 234-241.

Qiu, S. and A.L.N. Reddy, 2013. NVMFS: A hybrid file system for improving random write in NAND-flash SSD. Proceeding of the IEEE 29th Symposium on Mass Storage Systems and Technologies (MSST, 2013), pp: 1-6.

Rosenblum, M. and J.K. Ousterhou, 1992. The design and implementation of a log-structured file system. ACM T. Comput. Syst., 10(1): 26-52.

Russinovich, M., 2006. DiskMon for Windows v2.01. Retrieved form: http://www.microsoft.com/technet/sysinternals/utilities/diskmon.mspx.

Shu, F. and N. Obr, 2007. Data Set Management Commands Proposal for ATA8-ACS2. Retrieved form: http://www.t13.org.

Woodhouse, D., 2001. JFFS: The Journaling Flash File System. Proceeding of the Ottawa Linux Symposium 2001.