A NEW STORAGE ARCHITECTURE FOR A FLASH-MEMORY VIDEO SERVER

Sungchae Lim

Dept. of Computer Science, Dongduk Women's University, Seoul, South Korea
sclim@dongduk.ac.kr

Abstract

During the past decade, there has been drastic growth in network traffics that are made from online views on video content in Internet. Owing to current high popularities of video streaming services such as YouTube, Netflix and portals' UCC, such a trend is likely to be continuing in the next decade as well. For the successful proliferation of streaming services in the future, it is important to develop high-performance video servers that can fulfil bandwidth requirements of concurrently served video streams in a cost-effective way. To this end, we propose a video storage architecture using the SSD (solid state drive) as a cache layer. Since the SSD storage provides fast and uniform speed of random reads, it can work well with the EDF algorithm. To employ the EDF algorithm, the system time is divided into cycles and the bandwidth allocation is done by considering the scheduling periods and the number of flash blocks retrieved for serviced video streams. Since the bandwidth allocations of serviced video streams are managed within a full capacity of the scheduled SSD bandwidth, we can efficiently prevent undesirable hiccups of serviced video streams.

KeyWords: *Video streaming service, flash memory, real-time scheduling, data caching, admission control*

----------***----------

1. INTRODUCTION

During the past decade, there has been drastic growth in network traffics that are made from online views on video content in Internet [1-5]. Owing to current high popularities of video streaming services such as YouTube, Netflix and portals' UCC (user created content), such a trend is likely to be continuing in the next decade as well [4]. For the successful proliferation of streaming service in the future, it is important to develop high-performance video servers that can fulfil disk bandwidth requirements of concurrently served video streams in a cost-effective way [10-20]. This has been regarded as a major technical challenge to be tacked, together with the other challenge of cheaply paving digital superhighways with multiple gigabits bandwidth. In the case of database communities, the former one seems to be more important [11, 14, 16].

As earlier works on video servers, there were many studies [14-18] concerning storage architectures that can effectively meet soft deadlines of video segments being played back. The earlier studies aim to pump up real-time data streams from hard disk drives (HDDs), which is characteristic of its relatively poor bandwidth, compared to data consuming rates of video streams [14, 19]. To this end, they mainly focused on how to reduce the seeking overheads of HDDs via a grouped disk scheduling of video segments [19, 12, 14] and sophisticated data placement using RAID technologies [15, 16, 18]. With those schemes, it is possible to reduce lightly the investment cost paid for establishing HDD-based storage systems.

Although such HDD-based algorithms contribute to slightly diminishing cost for the video streaming services, they cannot guarantee satisfactory performance improvements because of the inherent drawback of the HDD, that is, its slow random read speed. To overcome that problem, we need to develop a new video storage systems that are constructed based on the new storage media such as flash memory. Since flash memory does not require any mechanical part for data accesses, differently from the HDD, it provides a very fast and uniform speed while processing random reads. This hardware characteristic is highly advantageous for the use of flash memory in the enterprise-scale video servers. For this reason, many studies have been done for the purpose of incorporating flash memory into the video server [6-9, 12].

In this context, we propose a new flash-aware scheduling algorithm tailored to video streaming servers equipped with flash storage of the NAND-type SSD (solid state disk) [12]. The proposed algorithm is based on a well-known real-time scheduling algorithm, that is, the EDF (earliest deadline first) algorithm [19]. Since the EDF algorithm has features such as the low computation cost and the high utilization of scheduling time, it has been widely accepted as a cost-effective scheduling algorithm. Despite such desirable features, the EDF algorithm has some problems when it is applied for scheduling the I/O requests of HDD storage. In real-time scheduling from the EDF algorithm, the processing orders of tasks are dynamically arranged in the accordance with tasks' deadlineurgencies. In other words, the scheduling orders of tasks coincide with their deadline urgency [12, 17].

When it comes to the disk scheduling, such a coincidence between task's urgency and scheduling orders easily deteriorates the level of disk bandwidth utilization. That is

because preemptive services of urgent data requests can yield enlarged I/O overheads. Since an HDD has overheads of the disk arm seeking and rotational delay while processing data requests, the service order of data requests in HDD needs to be adjusted depending on their cylinder distances, if the reduction of I/O overheads is only one performance factor being considered [19, 13]. For instance, if any of two data requests with highest deadlines are located at the leftmost and rightmost cylinders, respectively, processing of those two distant requests leads to a significant amount of I/O overhead.

Since data requests should be scheduled by considering the worst-case I/O overhead in order to meet their deadlines, the EDF algorithm cannot fully use the disk bandwidth while meeting timing constraints of video streams. To compensate for such a shortcoming of the EDF algorithm in the use of disk scheduling, some hybrid-style EDF algorithms [12, 19, 20] have been proposed. In these schemes, the deadlines of data requests are adjusted by considering the end time of bulk-SCAN in progress. By serving a group of requests through a bulk-SCAN, they can reduce the amount of I/O overheads embedded usually in disk schedule of the EDF algorithm.

In this paper, we propose an EDF style admission scheme that can avoid the above problems of the EDF algorithm. To this end, we employ the SSD storage in order to store the video segments that are delivered to remote clients via the HTTP protocol. Since the SSD storage does not have mechanical parts in it, it is the case that the I/O times for reading in-SSD video segments are independent of their location in the storage. This is very different from the case of HDD scheduling. Using the proposed scheme, we can efficiently eliminate both of deadline misses of data requests and an excessive waste of storage bandwidth in the schedules made from the EDF algorithm.

The rest of this paper is organized as follows. In section 2, we present the technical backgrounds including the concept of video servers and the hardware features of flash memory storage. In section 3, we address the proposed scheduling algorithm and its mechanism for admission control. In section 4, we conclude this paper.

2. BACKGROUNDS

2.1 Video Server

When it comes to alphanumerical data or image files, there do not exist hard timing constraints for reading those data [5]. I/O requests for such types of data have only to be processed in a tolerable time, i.e., the response time. A best-effort policy for processing data requests is enough in the case of such types of traditional data processing. However, if we are aimed at servicing video streams in online mode, then we have different system requirements to be fulfilled in the video servers [13, 15, 16]. Those requirements include admission control of streams within given resource capacity. Since the video stream needs almost constant size of data with a fixed length of time periods, many studies were done based on the algorithms being devised for scheduling periodic tasks [10, 14-19].

Because the earlier scheduling scheme for servicing video streams are designed based on HDD storage, they cannot evade a significant amount of disk bandwidth wastes. This is because they need to take into account the worst-case I/O time from random data placements across disk cylinders. To reduce the wasted I/O bandwidth, the grouped SCAN algorithm was proposed to obtain both of timely service and optimal seek-time overhead [12, 19]. Those style algorithms select a maximum group of I/O requests that can be served in a bulk SCAN, while meeting their deadlines. Since the I/O time using the SCAN algorithm is an optimal time, they can provide better disk bandwidth utilization, compared to scheduling working only depending on requests' deadlines. The proposed algorithm was reported to provide better performance, while video streams are being serviced with the similar playback rates.

As the cost per bit of DRAM is going down continuously, the video server begins to build a large size of a memory buffer used for caching hot segments of videos being played back [13, 5, 3]. The view pattern of video content is reported to be highly skewed toward a popular collection of videos [10]. Views of the popular video collections make up for most of workload of the video server. Therefore, by caching hot video content on a memory buffer, we can remarkably reduce the number of disk I/Os.

This caching scheme works more efficiently when the data size of cached video content is not big. In the case of video data with high resolutions and long playback times, the memory caching may not be effective. This is because the size of cached video collections is too small to reduce disk I/Os [14]. By contrast, the use of buffer memory may adversely affect the performance of video servers because of frequent switching of cached videos for reclaiming buffer space. Instead, other technique caching concurrently shared playback intervals of hot streams is accepted in recent time [10, 21].

2.2 Flash Memory Storage

Over the last decade, flash memory has much attention from the database community. This is mainly because its salient performance features such as low power consumption, fast speeds of random reads, and strong shock resistance. Because of the low cost per gigabit, additionally, NAND-type flash memory have been widely used as storage media. The NAND-type flash storage usually partitions its memory space into equal-size blocks and each block contains 64 or 128 pages of size 0.5KB within it. With multiple of NAND flash, SSD storage is built together with an SDRAM module and a small processing unit. The SSD is superseding the hard disk drives (HDDs) of laptop computers due to its performance advantages. Recently, there have been diverse studies for utilizing the SSD as storage media of enterprise-scale computing systems. In this light, there exist some researches that capitalize on the SSD to reduce the investment cost of video streaming services [2-5]

In the case of flash SSD, the conventional in-place-update is prohibited because of its H/W feature. Instead, FTL (Flash Translation Layer) is responsible forperforming hidden actions for out-of-place updates [7]. The use of out-of-place updates requires the action of garbage collection. Since the I/O costs for garbage collection are high, there is a need to reduce the number of random writes in SSD storage [8]. This is because I/O overheads for garbage collection deeply impact the performance of flash-based storages. For this reason, many studies proposed the FTL algorithms that can reduce the garbage collection costs. Additionally, some researches presented how to reduce the number of random writes in flash [8, 9].

3. PROPOSED METHOD

3.1Architecture of Video Storage

In Internet, a large number of video streaming services have employed the adaptive HTTP streaming (AHS) for flexible data delivery. Dynamic Adaptive Streaming over HTTP (DASH) is an international standard for delivering MPEG-coded video segments [1]. This DASH-based streaming system does not require any use of dedicated video servers and its service is possible with usual off-the-shelf Web servers, that is, Apache web server or IE web server. For the system using the off-the-shelf web servers and DASH data delivery, the networking technology of CDN (Content Delivery Network) is usually deployed for dynamic workload balancing [3]. In this network architecture, the data of each video object are saved on the storage of distributed web servers. Then, each copy of the duplicated videos will be partitioned into a fixed size of segments for delivery [2].

In this paper we mainly focus on the storage architecture and scheduling algorithm for the web server that is pumping video segments to client through the DASH protocol. To capitalize on the fast random reads of flash storage, we incorporate the SSD into the storage system for its use as a cache pool. Fig. 1 depicts the architecture that is assumed in our study.

We dedicate some part of main memory to the web server that serves video streams to remote clients over Internet.The dedicated memory space has two parts of a cache pool and a temporary area. The former is used to cache video segments that are retrieved from the disks; the latter is for reading in-SSD video segments. We do not cache the data that are reading from SSD. That is, from memory we delete the video segments that are read from SSD, immediately after sending them to Internet. The part of main memory dedicated to the web server that serves video streams to client.

The cache pool in Fig. 1 is for caching a portion of hot video segment that are stored in HDDs. For efficient caching of them, many algorithms including LRU (least recently used) can be applied for the reduction of cache miss rate. Since those algorithms are not our main interests, we do not describe detail about them. SSD storage serves as a type of cache area for hot video segments. To prevent performance degradation caused by small sizes of random writes, the minimum size of date writes to SSD storage from HDDs is set to any value in the rage of 1 MB to 8MB. With an enough large size of writes, we can avoid excessive overhead from garbage collections [8, 9]. In contrary, the size of reads is the same that of a video segment. Here, the sizes of video segments have different values in accordance with the different types of video. In this paper we propose the scheduling algorithm for reading video segments in SSD storage in real-time. The scheduling algorithm is presented in the next section.

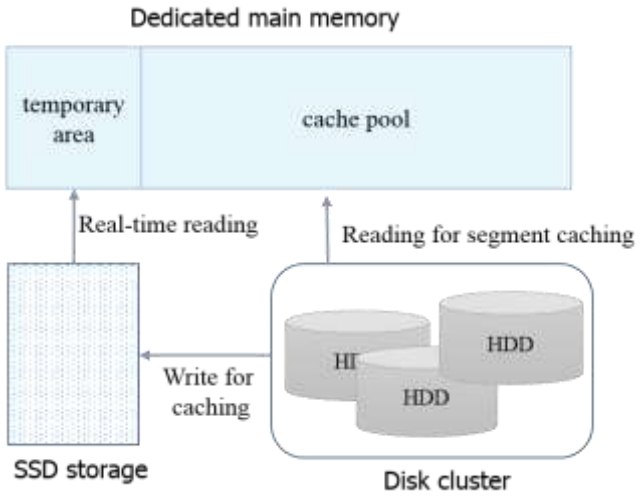

Fig. 1. The storage architecture of the proposed video server storage using SSD storage as a cache pool.

3.2 Scheduling Algorithm

To adopt the EDF algorithm, we save video segments on SSD storage in the unit of a flash block. That is, a single video segment is cached across multiple flash blocks and a single flash block can save data of the same video stream. By reading such video blocks of a video stream sequentially, we can fulfill the bandwidth requirement of that video's streaming service. While serving concurrent video streams, we determine which flash block needs to be scheduled for I/O in the next I/O time. For determining the retrieval orders of flash blocks of video streams, we adopt the EDF algorithm.

More particularly, we divide the system time into cycles, which have a fixed size of length and are used as the smallest unit of storage bandwidth allocation. To serve a video stream requiring bandwidth x, we retrieve b number of flash blocks with a period of p cycles. In other words, by reading b blocks during every c cycles, we can provide the disk bandwidth of $(b*SB)/(p*CYCLE.)$ Here, SB is the size of a flash block and $CYCLE$ is the length of a cycle. In this paper, we refer to the pair of (c, p) as the *bandwidth allocation token* (BAT). By appropriately choosing a BAT for a video stream to be admitted, we can provide a video streaming service.

Algorithm 1: Procedure *GetTheBAT()*

input : $r_{requested}$ = the size of disk bandwidth requested by a stream;
$CYCLE$ = the size of a cycle;
SB = the size of a flash block;
output: (b, p) = the bandwidth allocation token to be assigned;

1 **for** $i \leftarrow 2$ **to** $MaxCycle$ **do** // p in range of 2 to $MaxCycle$
2 Find integer N_i such that

$$\frac{SB \times (N_i - 1)}{i \times CYCLE} < r_{requested} \leq \frac{SB \times N_i}{i \times CYCLE}$$

3 Calculate the size of expected bandwidth fragmentation (GAP_i) such that

$$GAP_i = \frac{SB \times N_i}{i \times CYCLE} - r_{requested}$$

4 Select integer min so that GAP_{min} is the smallest one in the set of $\{GAP_2, GAP_3, \ldots GAP_{MaxCycle}\}$;
5 $(b, p) \leftarrow (N_{min}, min)$; // return the BAT of (b, p)

Fig.2. The algorithm for determining a proper BAT with respect to a video stream requesting its bandwidth requirement of $r_{reqested}$

Fig. 2 shows the algorithm for the procedure *GetBAT()*, which determines a BAT for a streaming request asking for disk bandwidth of $r_{requested}$. In Fig. 2, the value ofGAP_i is the difference between $r_{requested}$ and the bandwidth guaranteed by a considered BAT. By choosing the smallest GAP_i , we can reduce the amount of bandwidth fragmentation, thereby increasing the utilization of SSD bandwidth. The larger size of *MaxCycle* may diminish the total amount of bandwidth fragmentations, while it increases the memory requirements needed to temporarily cache the data read from SSD storage. The value of *MaxCycle*is chosen in the range of 6 to 10.

Suppose that a video stream *s* is admitted with BAT of (*c*, *p*). After admission of streams *s*, we read the first c flash blocks of *s* as soon as possible and begin its playback immediately after the cycle wherein retrieval of *c* flash blocks have been completed. From the playback starting time, the data of admitted video streams are read in accordance with the EDF priorities. For example, if a video streams with (*c*, *p*) begins its playback from a $Cycle_i$, then the next deadline of p flash blocks being read is the same as the end of $Cycle_{i+p}$. The proposed scheduling algorithm can prevent undesirable hiccups of serviced video streams, because the EDF algorithm can serve all the admitted tasks until its utilization does not exceed 1. To admit a stream, we use the algorithm of Fig. 3.

Algorithm 2: Procedure *CheckAdmission()*

input : $r_{requested}$ = the size of disk bandwidth requested by a stream;
n_{max_blocks} = the maximum number of blocks read in a cycle;
R_{used} = the current value of bandwidth utilization;
output: **YES** *or* **NO**;

1 Execute the procedure $GetBAT(r_{requested})$ to get a BAT of (b, p);
2 $U = c/(p \times n_{max\_blocks})$; // compute the bandwidth usage rate of this stream with BAT (b, p) ;
3 **if** $(R_{used} + U) > 1$ **then** // admission checking
4 Return **NO**;
5 **else**
6 $R_{used} \leftarrow R_{used} + U$;
7 Return **YES**;

Figure.3. Algorithm for checking the possibility of the new admission of a stream.

The procedure *CheckAdmission()* is used to check if a new stream can be permitted for playback. In the produce, the playback rate of the stream being admitted is presented by $r_{requested}$. In line 1, *GetBAT()* is called to get a BAT for the stream. Then, its bandwidth utilization of the stream is computed in line 2. With this bandwidth utilization, the procedure computes total bandwidth utilization. If the total utilization is not greater than 1, the stream is accepted for playback as in lines 6-7. When computing the bandwidth utilization in line 2, the value of n is the maximum number of flash blocks that are read within the time interval of a cycle. Since the data retrieval in SSD storage can be without the support of mechanical components, we can obtain constant number of blocks retrieved in a cycle.

4. CONCLUSION AND FUTURE WORK

In this paper, we propose a real-time scheduling method used for the video streaming storage that is aimed at caching hot-video segments in it. Since SSD storage has some advantages of fast and uniform times of random reads, it is easy to apply an EDF-style algorithm for delivering video segments in accordance with real-time requirements. That is different from the case where we perform real-time scheduling the HDD storage. Because of the I/O overheads of disk-arm seeking and rotational delay in HDDs, the EDF-style algorithm is not feasible for scheduling I/O requests for the disk-resident data.

To exploit the fast and uniform speed of random reads, our proposed method divides time into a fixed size of cycles and allocates disk bandwidth based on the size of a cycle. More specifically, for a certain stream *s* we pick a scheduling pattern that is made up of a scheduling period and the number of flash blocks read in *p* cycles. By reading data saved in SSD storage according to a given bandwidth allocation tokenof (b, p), we can support the data delivery capacity of $(b \times bloc\ size)/(p \times cycle\ size)$. From the bandwidth allocation tokenof (b, p) , we compute its bandwidth utilization value of *u*. Then, we add up the values of bandwidth utilization with respect to all the already serviced streams. By admitting a new stream *s* only while the total bandwidth utilization is not greater than 1, the proposed scheduling method easily prevents hiccups of video streams. From this, we can grantee a quality video streaming service.

Until now, we have not developed a well-designed algorithm that can select hot video segments suitable for being cached in SSD storage. In addition, we also need to elaborate the proposed method so that it can efficiently work with situations where not-periodic requests are frequently issued. As future work, we need to solve those two challenging things.

REFERENCES

[1] HTTP (DASH) - Part 1: Media presentation description and segment formats.
[2] T. Stockhammer. “Dynamic Adaptive Streaming over HTTP: Standards and Design Principles", In

Proceedings of ACM Conf. on Multimedia Systems, 2011.
[3] Liao, X., Jin, H., Liu, Y., Ni, L. M., and Deng, D. “Anysee: Scalable live streaming service based on inter-overlay optimization”, In Proceedings of IEEE INFOCOM, 2006.
[4] Liu, C., Bouazizi, I., and Gabbouj, M. “Rate adaptation for adaptive http streaming", In Proceedings of the ACM Conference on Multimedia Systems, 2011.
[5] Li, B., Qu, Y., Keung, Y., Xie, S., Lin, C., Liu, J., and Zhang, X. “Inside the New Coolstreaming: Principles, measurements and performance implications”, In Proceedings of IEEE INFOCOM, 2008.
[6] D. Bae et al, “An Efficient Method for Record Management in Flash Memory Environment,” Journal of Systems Architecture, pp. 221-232, 2012.
[7] S. Lee and B. Moon, “Design of Flash-based DBMS: An In-page Logging Approach“, In Proc. of ACM SIGMOD, pp. 55-66, 2007.
[8] Jaeyoung Do, Donghui Zhang, Jignesh M. Patel, David J. DeWitt, Jeffrey F. Naughton, and Alan Halverson, “Turbo-charging DBMS Buffer Pool Using SSDs”, In Proc. of ACM SIGMOD, 2011.
[9] Mustafa Ganim, George A. Mihaila, BishwaranjanBhattacharjee, Kenneth A. Ross, and Christian A. Lan, “SSD Bufferpool Extensions for Database Systems”, In Proc. of VLDB, pp. 1435-1446, 2010.
[10] Acharya, S. and Smith, B. “Characterizing User Access to Videos on the World Wide Web," In Proc. of MMCN, 2000.
[11] E. Balafoutis, M. Paterkakis, and P. Triantallou, “Clustered Scheduling Algorithms for Mixed-Media Disk Workloads in a Multimedia Server". Cluster Computing Journal, 6(1):75-86, 2003.
[12] AsyushGupta,Youngjae Kim, and BhuvanUrgaonkar, “DFTL: A Flash Translation Layer Employing Demand-based Selective Caching of Page-level Address Mappings”, In Proceedings of ASPLOS, 2009.
[13] Edward Y. Chang and Hector Garcia-Molina. “Effective Memory Use in a Media Server". In Proc. of the Intl. Conference on Very Large Databases, 1997.
[14] Huang-Jen Chen and Thomas D.C. Little. “Storage Allocation Policies for Time-dependent multimedia data”, IEEE Trans. on Knowledge and Data Engineering, 8(5):855-864, 1996.
[15] Ming-Syan Chen, Dilip D. Kandlur, and Philip S. Yu. “Support for Fully Interactive Playout in a Disk-Array-Based Video Server". In Proc. of the ACM Multimedia, October 1994.
[16] Asit Dan and DinkarSitaram. “An Online Video Placement Policy based on Bandwidth to Space Ratio (BSR)". In Proc. of ACM SIGMOD, 1995.
[17] A. Ermedahl, H. Hansson, and M. Sjaodin. “Response-Time Guarantees for ATM Networks". In Proc. of the IEEE Real-Time Systems Symposium, 1997.
[18] O. Ertug, M. Kallahalla, and P. J. Varman. “Real-Time Parallel Disk Scheduling for VBR Video Servers". In Proc. of the Fifth Intl. Conference on Computer Science and Informatics, February 2000.
[19] Sungchae Lim and Myoung Ho Kim. “Real-time Disk Scanning for Timely Retrieval of Continuous Media Objects". Information and Software Technology, 45(9):547-558, June 2003.
[20] R. Wijayaratne and N. Reddy. “Integrated QoS Management for Disk I/O". In Proc. of the IEEE Multimedia Systems, pages 487-492, June 1999.

BIOGRAPHIES

Sungchae Lim

He received the M.S. and Ph.D. degrees in KAIST. He is currently an Assistant Professor at Dongduk Women’s University.