1International Journal of Internet, Broadcasting and Communication Vol.10 No.3 26-34 (2018)
http://dx.doi.org/10.7236/IJIBC.2018.10.3.26

IJIBC 18-3-3

FlashEDF: An EDF-style Scheduling Scheme for Serving Real-time I/O Requests in Flash Storage

Seong-Chae Lim

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

Abstract

In this paper, we propose a scheduling scheme that can efficiently serve I/O requests having deadlines in flash storage. The I/O requests with deadlines, namely, real-time requests, are assumed to be issued for streaming services of continuous media. Since a Web-based streaming server commonly supports downloads of HTMLs or images, we also aim to quickly process non-real-time I/O requests, together with real-time ones. For this purpose, we adopt the well-known rate-reservation EDF (RR-EDF) algorithm for determining scheduling priorities among mixed I/O requests. In fact, for the use of an EDF-style algorithm, overhead of task's switching should be low and predictable, as with its application of CPU scheduling. In other words, the EDF algorithm is inherently unsuitable for scheduling I/O requests in HDD storage because of highly varying latency times of HDD. Unlike HDD, time for reading a block in flash storage is almost uniform with respect to its physical location. This is because flash storage has no mechanical component, differently from HDD. By capitalizing on this uniform block read time, we compute bandwidth utilization rates of real-time requests from streams. Then, the RR-EDF algorithm is applied for determining how much storage bandwidth can be assigned to non-real-time requests, while meeting deadlines of real-time requests. From this, we can improve the service times of non-real-time requests, which are issued for downloads of static files. Because the proposed scheme can expand flexibly the scheduling periods of streams, it can provide a full usage of slack times, thereby improving the overall throughput of flash storage significantly

Keywords: *Flash memory, Real-time scheduling, EDF algorithm, Streaming service, Flash storage*

1. Introduction

In the long history of the research of disk scheduling algorithms, latency time in HDD (hard disk drive) has been a major issue being tackled [2, 3, 7, 9, 10, 13, 15-18]. HDD is comprised of a multiple of disk plates that constantly revolve with the same spindle of them. Each side of the disk plates has disk tracks, and a group of disk tracks having the same distance from the spindle is called a disk cylinder. By moving the disk arm across disk cylinders forwards or backwards, HDD reads target data blocks requested. Because the movement of the disk arm relies on mechanical mechanisms, latency time for a block read is dependent on

Manuscript Received: May. 20, 2018 / Revised: May. 26, 2018 / Accepted: June. 2, 2018
Corresponding Author: sclim@dongduk.ac.kr
Tel: +82-2-940-4589, Fax: +82-2-940-4170
Dept. of Computer Science, Dongduk Women's University, Korea

cylinder distances of disk arm movement. Such latency time is referred to the *seek time*. Besides the seek time latency, HDD needs rotational delay during reading a block [14].

To diminish such latency time in HDD scheduling, many researchers proposed a large number of algorithms such as FCFS, SCAN, SSTF (Shortest Seek Time First), CSCAN (Circular SCAN) [10, 3, 14]. For the improved latency time in HDD, those algorithms are devoted to reducing the overall movements of a disk arm. By adjusting the service orders of blocks in retrieval, the algorithms reduce total cylinder distances of arm movements. If the scheduling algorithm has to consider deadlines of I/O requests, then its computational complexity surges highly. Since a real-time scheduler is required to provide an optimal service time, while preventing deadline misses, it cannot be implemented as a polynomial-time algorithm [2, 5, 17]. For this reason, it takes a compromise between robustness of deadline guaranteeing and I/O performance [3, 17].

In this paper, we propose a scheduling scheme for serving I/O requests with deadlines in flash storage. The requests with deadliness, namely **real-time** requests, are issued for streaming service of continuous media. Here, since data such as MPEG-coded video files and sound files are continuously played back at a specific rate, they are called continuous media [7, 16, 18]. Since a stream of continuous media imposes deadlines on its data blocks, I/O requests for streaming service are issued with deadlines. Besides such real-time requests for continuous media, our proposed scheme is aimed at supporting a mixture service of non-real-time requests. For this purpose, we adopt the well-known rate-reservation EDF (**RR-EDF**) algorithm [8]. In fact, for the use of that algorithm, it is required to decide a minimum I/O bandwidth of storage to be assigned for real-time requests. In the case of HDD storage, such computation is not realistic because of HDD's latency times affected by varying service orders of I/O requests.

Unlike HDD, the time for reading flash blocks is almost uniform independently with the service order of blocks. This is because the I/O mechanism of flash storage has no mechanical component [1, 5, 6]. Due to the uniform read time in flash, it is possible to estimate the amount of I/O bandwidth available within a fixed length of time, that is, **round**. Recall that storage bandwidth in HDD severely varies with service orders of I/O requests because of different sets of latency times. By using the RR-EDF algorithm based on the round bandwidth, our proposed scheme can easily estimate I/O utilization of real-time requests in service. Together with real-time requests, our proposed scheme schedules non-real-time requests by using idle storage bandwidth, which is also calculated through the RR-EDF algorithm. Since the proposed scheduling scheme can use up idle bandwidth remaining in each round, it can serve non-real-time requests quickly, while satisfying deadlines of real-time requests. This advantage can be maximized by properly expanding the scheduling period of real-time requests. To this end, our proposed scheme allows served streams to have different scheduling periods.

The organization of this paper is as follows. In Section 2, we give some preliminary knowledge about HDD scheduling algorithms and their main issues. In Section 3, we propose a flash-based scheduling scheme, and study its performance advantages in Section 4. Lastly, we conclude this paper in Section 5.

2. Preliminaries

The time needed to locate a disk arm over a target disk track is called the seek time, which accounts for a major portion of disk service times. This seek time is modeled as in equation (1), where the seek time function *seek_time()* is combination of a square root function and a linear function of cylinder distances D of

disk arm movement [3, 9].

$$seek\_time(D) = \begin{cases} C1 + C2\sqrt{D}, & D < X \\ C3 + C4 \times D, & otherwise \end{cases} \quad (1)$$

, where *Cx* is disk parameters specific to a disk drive, and *X* is a boundary from which the disk arm moves at a constant speed.

Due to the seek time overhead, a disk scheduling algorithm has a need for deciding a service order of I/O requests that results in a smaller size of total seek time overhead. Theoretically, the problem of deciding an optimal disk schedule with the smallest total seek time has the computational complexity of the well-known NP-complete problem of Traveling Salesman Problem (TSP) [11]. If we regard a traveling cost between two cities in TSP to be a seek time between two I/O requests, TSP is reducible to a disk scheduling problem. Moreover, if we assume that deadlines are given to the I/O requests, an optimal scheduling algorithm considering deadlines is NP-hard [11, 17].

To evade such intolerable computational costs, therefore, many heuristic algorithms have been proposed in reality. For example, The SCAN-EDF algorithm was devised as a hybrid algorithm that incorporates the real-time aspect of the EDF algorithm and seek time optimization aspect of the SCAN algorithm [3, 17]. Note that we do not differentiate between the SCAN and Elevator SCAN algorithms, if there is no special need. The main idea of SCAN-EDF is very simple. When this algorithm is applied for serving requests pending in an I/O request queue, it aims to have seek time optimization. That is, it serves only those that have the same earliest deadline. Such scheduling is repeatedly executed. This algorithm has a limitation in that it does not guarantee deadlines of every case of I/O requests as well as optimal I/O throughput.

The FD-SCAN algorithm was proposed devised to improve the performance of the SCAN-EDF algorithm [15]. To this end, FD-SCAN follows the EDF algorithm to determine the direction of the disk head movement. In FD-SCAN, the track location of a data request with the earliest feasible deadline is used to determine the scan direction. Determining the feasibility of a data request is based on the information of the targeted track location and the disk head position. Request's deadline is feasible if the deadline is greater than the current time plus the request's service time. To determine such earliest feasible request, all the data requests are examined at every point of scheduling time. Then, the FD-SCAN scheduler moves the disk arm towards the determined request, while servicing other requests on the way to that request, i.e., the SCAN policy is applied in the range of the current disk position to the current feasible request.

In [15], FD-SCAN is tested against other algorithms, including FCFS, CSCAN, SSTF, and EDF. Simulation results show that FD-SCAN runs best among the compared other algorithms in terms of an ability satisfying deadlines of requests. Even though it can run better over the SCAN-EDF, it does not support completely guaranteeing of deadlines. Moreover, no one of the algorithms above schedules a mixture of real-time I/O requests and non-real-time requests in an efficient way.

3. Proposed Scheduling Scheme

3.1 Rate-Reservation EDF

In the domain of real-time scheduling, the EDF algorithm is one of the most popular algorithms. The EDF algorithm was originally developed to schedule real-time tasks that compete for CPU time. The popularity of the EDF algorithm is due to its low execution cost and easy implementation [2, 4]. In this algorithm, it is

assumed that costs of scheduling switching between two tasks are negligibly tiny. However, if the EDF algorithm is directly adopted for scheduling I/O tasks in HDD storage, such an underlying assumption becomes unacceptable. This is because switching times, which can be regarded as latency times in HDD, vary largely depending on cylinder distances of disk arm's movement [9, 13]. However, in the case of flash storage, such latency time can be negligible and constant in size. For this reason, we employ an EDF-style algorithm for scheduling real-time I/O requests. Specifically, we use the rate-reservation EDF algorithm to serve I/O requests in flash storage.

The RR-EDF scheduler was designed to efficiently handle mixed workloads of periodic and sporadic tasks. In this scheduler, time is divided by a unit scheduling time, called a **unit cycle**, and CPU time is assigned to tasks in the unit of that time. The size of a unit cycle is chosen as the greatest common divisor of all the periods of tasks being scheduled. Since task switching is possible only at the beginning point, the release point of every periodic task is also made to coincide with a beginning point. The schedulability test of period tasks can be done in a very simple way. For each periodic task that requests CPU time of size C for every P unit cycles, its total utilization rate is computed by $C/P \times T$. Here, T is the size of a unit cycle. By keeping the total utilization rate of periodic tasks less than 1, an RR-EDF scheduler can prevent deadline misses by scheduling periodic tasks with the earliest deadline at each task switching point.

An interesting point of the RR-EDF scheduler is on the fact that it can serve sporadic tasks as well as periodic tasks. When the total utilization rate is denoted by U_{res}, the RR-EDF scheduler assigns the ratio of $(1–U_{res})$ to sporadic tasks in each unit cycle. In the study of [8], it has been proven that such CPU time assignment does not harm schedulability of other periodic tasks. According to this scheduling policy, the RR-EDF scheduler can support fast response times of sporadic tasks, while processing periodic tasks without deadline misses. In this paper, we adopt this RR-EDF algorithm to process real-time and non-real-time I/O requests together. The detail of that is presented in the next subsection.

3.2 Use of BRT (Bandwidth Request Type)

For the use of the RR-EDF algorithm, we also choose a fixed unit of scheduling time, which is referred to as a **round** from now on. Let T be the size of a round, and let the end point of the i-th round be denoted by R_i, that is, $R_i = T \times i$. Then, we compute the number of flash blocks that can be read in time T, and let that number be C. Then, capacity C is used as the minimum storage bandwidth available in a single round. If any stream S requires k number of blocks in every round for its playback, then stream S needs to consume storage bandwidth rate of k/C with respect to total storage bandwidth $C \times B/T$, where B is the block size. While the sum of streams' bandwidth utilizations is below 1, our RR-EDF based scheme (called **FlashEDF**) satisfies all the deadlines of real-time requests issued from continuous media streams. When any stream requires k blocks in every round, the pair of $< 1, k >$ is called the **bandwidth requesting type** (BRT).

To make the concept of BRT more general, the BRT of our scheme is made to have the form of $< p, k >$. In the case of stream S with $< p, k >$, stream S requires k blocks for every p rounds. If that stream starts issuing its data requests from time point R_i, the deadlines of S's real-time requests will be R_{i+pn} $(n \geq 1)$. When a stream has a BRT $< p, k >$, its storage bandwidth utilization is computed as fraction of $k/C \times p$.

The schedulability testing and choice of a BRT is performed using procedure *TestAndCalcBRT()* of Figure 1. In Figure 1, the procedure gets two input parameters, b and U_{res}. Parameter b is the size of storage

bandwidth required by a new stream to be serviced. During the loop in lines 1-3, the procedure calculates a set of possible BRTs and the values of bandwidth fragmentations for each of those BRTs. Then, a BRT having the smallest bandwidth fragmentation is selected as in lines 4. If the stream is acceptable within remaining storage capacity, that BRT is assigned for that stream in line 9. Note that the bandwidth utilization for BRT $< p, k >$ is computed with $k/C \times p$. After the testing, the new stream is made to issue real-time data requests with a cycle of length $C \times T$.

Algorithm 1: Procedure *TestAndCalcBRT()*

Input : b = storage bandwidth requested by a stream;
U_{res} = current bandwidth utilization of real-time requests;
Output: $\langle p, k \rangle$ = BRT satisfying bandwidth requirement of b;

1 **for** $i = 1$ **upto** max_p **do**
2 With block size B, find an integer n_i such that

$$\frac{B \times (n_i - 1)}{i \times T} < b \leq \frac{B \times n_i}{i \times T};$$

3 Calculate bandwidth fragmentation $diff_i$ such that

$$diff_i = \frac{B * n_i}{i * T} - b;$$

4 Choose the smallest fragmentation $diff_{min} \in \{diff_1, diff_2, \ldots, diff_{max\_p}\}$;
5 Use the two integers, min and n_{min} as p and k, respectively;
6 **if** $(k/(C \times p) + U_{res}) > 1$ **then** `// schedulability test`
7 Notify that testing is failed because of avaiable bandwidht shortage;
8 **else**
9 Return $\langle p, k \rangle$ as a BRT satisfying schedulability testing;

Figure 1. Procedure for determining a bandwidth request type

3.3 Scheme FlashEDF

As mentioned before, the purpose of scheme FlashEDF is to process a mixture of real-time requests and non-real-time requests at the same time. The non-real-time request is one that is issued for processing downloads of static files of HTMLs or images. In the context of the original RR-EDF algorithm, the non-real-time requests can be regarded as CPU time needed for sporadic tasks. To process real-time I/O requests together with non-real-time I/O requests, our scheduler computes the storage bandwidth that can be assigned for them. That is, the rate of $(1–U_{res})$ is assigned to non-real-time requests with respect to the minimum storage bandwidth that is available until the nearest end point of the rounds. From the theoretical basis of the original RR-EDF algorithm, it is the case that deadline-guaranteeing scheduling is possible, if an EDF algorithm is used to pick up real-time requests in U_{res} rate of total storage bandwidth.

To show how the algorithm works, we present the procedure *ExecuteSchemeFlashEDF()* in Figure 2. The

procedure accepts two sets of requests and the current bandwidth utilization of served streams. In line 2, the procedure first computes the maximum number of blocks that can be retrieved before the endpoint of the current round. Here, *FTtime(k)* is a function returning the I/O time for reading *k* blocks.

In lines 4-6, the procedure selects the real-time requests to be served. For this, the storage capacity assigned for real-time requests is computed in line 3. In lines 7-9, the procedure chooses non-real-time requests within the remaining bandwidth capacity of the current round. After deciding the set of requests being served, that is, S_{serve}, the procedure processes them in line 10 through I/O calls.

Algorithm 2: Procedure *ExecuteSchemeFlashEDF()*

Input : $\mathcal{S}_p$ = Set of periodic requests with deadlines;
$\mathcal{S}_d$ = Set of discrete requests without deadlines;
U_{res} = Bandwidth utilization rate of real-time requests;

1 Let t be the current time, and set $\mathcal{S}_{serve}$ to $\emptyset$ for initialization;
2 Let R_j be the end point of the current round, and compute the service capacity of integer N_j such that

$$FTime(N_j) \leq R_j - t < FTime(N_j + 1);$$

3 Compute service capacity used for serving requests in $\mathcal{S}_p$ such that $N_j^p = \lfloor U_{res} \times N_j \rfloor$, where N_j is the total bandwidth capacity available;
4 **while** $|\mathcal{S}_p| > 0$ *and* $|\mathcal{S}_{serve}| \leq N_j^p$ **do**
5 Select a requerst r that has the earliest arrival time in $\mathcal{S}_p$;
6 Extract r from $\mathcal{S}_p$ and input it into $\mathcal{S}_{serve}$;
7 **while** $|\mathcal{S}_d| > 0$ *and* $|\mathcal{S}_{serve}| \leq N_j$ **do**
8 Select a requerst r that has the earliest arrival time in $\mathcal{S}_d$;
9 Extract r from $\mathcal{S}_d$ and input it into $\mathcal{S}_{serve}$;
10 Process the requests of $\mathcal{S}_{serve}$ through I/O calls.

Figure 2. Procedure for choosing requests to be served in each round

4. Performance Analysis

The idea of the proposed scheduling scheme is based on low volatility of block read times in flash storage, which is different from HDD storage. To utilize this property of flash storage, we choose a fixed length of scheduling round and estimate storage bandwidth made during that time. Then, I/O scheduling is performed by properly assigning the storage bandwidth to I/O requests in each round, by following the EDF policy. To use the EDF algorithm, in addition, we make the deadlines of real-time requests coincide with the ends of the rounds. In this scheduling environment, it was already proven that an EDF algorithm can meet all the deadlines of real-time requests, only if bandwidth utilization rate is not greater than 1 [2, 8].

On the top of the basic idea of EDF scheduling, we considered the mixed service of non-real-time requests, that is, discrete requests. The discrete requests come from downloads of static files, which have no their own deadlines. To provide faster response times of discrete requests, we employ a variation of the EDF algorithm in order to compute the maximum bandwidth allocable to discrete requests. That is, the rate-reservation EDF algorithm was applied for the calculation of such storage bandwidth at a cheap CPU cost. By assigning slack storage bandwidth to discrete requests as much as possible, the proposed scheme can improve the overall I/O

performance without missing deadlines of real-time requests.

The advantage of faster service of discrete requests in our scheme, in fact, deeply depends on the size of a round and the level of bandwidth utilization rate of real-time requests. To look into the I/O advantages, we use a modern SSD package shown in Table 1 [19]. The V-NAND storage has 4KB size of blocs and supports up to 97K reads of blocks per second. This read speed should be downgraded for random reads.

Table 1. Samsung SSD 850 Evo Specification

Parameters	Data in detail
Storage Capacity	500GB
Flash Type	TLC V-NAND
Interface Speed	SATA 6Gb/sec.
Size of a Block	4KB
Block Read Speed	97000/sec.

With the device of Table 1, we can theoretically compute the mean improvement of response times of discrete requests, compared with any scheme using a naive EDF algorithm, that is, any algorithm without rate-the reservation concept. If we set the round length to one second, and the utilization 90% of real-time requests, then we can obtain almost 10% of improvement of the response times of discrete requests, compared with a naive EDF scheme. Here, we assume that the naïve EDF scheme just schedules real-time requests in the order of their earliest deadlines, and its scheduling period is equal to a single round. Such performance gains grow with the increases of the maximum period for BRTs allowed [8, 17]. By expending it, we can expect additional improvement up to about 30% by holding the storage utilization of 90%. However, since the improvement is affected by the storage utilization as well, there is a need to study various features of the proposed scheme as future works.

5. Conclusion

In this paper, we proposed a new way that the rate-reservation EDF algorithm can be applied for serving I/O requests with deadlines. This algorithm is not acceptable for scheduling such real-time requests in HDD storage because of disk latency time varying with cylinder distances between requests being processed. Being free from mechanical movements of an HDD arm, flash storage has almost constant time for a block read, regardless of block's physical locations. By capitalizing on the uniform access time, we compute the bandwidth utilization rate of real-time requests issued by continuous media streams. Then, the rate-reservation EDF algorithm is applied for assigning storage bandwidth to both of real-time requests and non-real-time requests, respectively. From this, the proposed scheme can considerably advance the response times of non-real-time requests, while meeting others' deadlines. In addition, the storage throughput can be enhanced by flexibly expanding the scheduling period of real-time requests. From those advantages, it is likely that our proposed scheme can be used as a scheduling scheme of Web sites that are running on flash storage, where video clips and image files are stored together.

Acknowledgement

This work was supported by the Dongduk Women's University grant in 2017.

References

[1] Seong-Chae Lim, "A Flash-based B+-Tree using Sibling-Leaf Blocks for Efficient Node Updates and Range Searches," *The Journal of the Institute of Internet, Broadcasting and Communication (JIIBC)*, Vol. 8, No. 3, pp. 12-24, August 2016.
DOI: http://dx.doi.org/10.7236/IJIBC.2016.8.3.12

[2] Houssine Chetto and Maryline Chetto, "Some Results of the Earliest Deadline Scheduling Algorithm,". *IEEE Transactions on Software Engineering*, Vol. 15, No. 10, pp. 1261-1269, 1989.
DOI: https://doi.org/10.1109/TSE.1989.559777

[3] R. Geist and S. Daniel, "A Continuum of Disk Scheduling Algorithms," *ACM Transactions on Computer Systems*, Vol. 5, No. 2, pp. 77-92, 1987.
DOI: https://doi.org/10.1145/7351.8929

[4] Byung Joo Kim, "Real-time Fault Detection in Semiconductor Manufacturing Process: Research with Jade S olution Company," *The Journal of the Institute of Internet, Broadcasting and Communication (JIIBC)*, Vol.9, No.2, pp.20-26, 2017.
DOI: https://doi.org/10.7236/IJIBC.2017.9.2.20

[5] Stan Park and Kai Shen "FIOS: A Fair, Efficient Flash I/O Scheduler," in *Proc. 10th USENIX Conference on File and Storage Technologies*, February. 14-17, 2012.

[6] Jaechun No and Sung-soon Park, "Exploiting the Effect of NAND Flash-Memory SSD on File System Design," *in Proc. of International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery*, October 10-12. 2012.
DOI: https://doi.org/10.1109/CyberC.2012.93

[7] Mon-Song Chen, Dilip D. Kandlur, and P. Yu, "Optimization of Grouped Sweeping Scheduling (GSS) with Heterogeneous Multimedia Systems," in *Proc. of the ACM Multimedia*, August 2-6, 1993.
DOI: 10.1145/166266.166293

[8] Kang G. Shin and Yi-Chieh Chang, "A Reservation-Based Algorithm for Scheduling Both Periodic and Aperiodic Real-Time Tasks," *IEEE Transactions on Computers*, Vol. 44, No. 12, pp. 1405-1419, 1995.
DOI: https://doi.org/10.1109/12.477246

[9] C. Ruemmler and J. Wilkes, "An Introduction to Disk Modeling," *IEEE Computer*, Vol. 27, No. 3, pp. 17-28, March 1994.
DOI: https://doi.org/10.1109/2.268881

[10] Hofri Micha, "Disk Scheduling: FCFS vs SSTF Revisited," *Communications of the ACM*, Vol. 23, No 11, pp. 645-653, 1980.
DOI: https://doi.org/10.1145/359024.359034

[11] E.L. Lawler, J.K. Lenstra, A H.G. Rinnooy Kan, and D. B. Shmoys, "The Traveling Salesman Problem: A Guided Tour of Combinatorial," John Wiley & Sons Publications, 1985.
DOI: https://doi.org/10.1057/jors.1986.117

[12] T. H. Lin and W. Tarng, "Scheduling Periodic and Aperiodic Tasks in Hard Real-time Computing systems," *ACM SIGMETRICS*, Vol. 19, No 1, pp. 31-38, 1991.
DOI: https://doi.org/10.1145/107972.107976

[13] M. Seltzer, P. Chen, and J. Ousterhout, "Disk Scheduling Revisited," in *Proc. of the USENIX Winter 1990*, pp. 313-324, December 1990.
DOI: https://doi.org/10.1.1.152.5459

[14] Abraham Silberschatz, Peter B. Galvin, and Greg Gagne, *Operating System Concepts*, 9th edition, Wiley, 2016.
[15] R. K. Abbott and H. Garcia-Molina, "Scheduling I/O Requests with Deadlines: A Performance Evaluation," in *Proc. of the Real-Time Systems Symposium*, pp. 113-125, December 5-7, 1990.
DOI: https://doi.org/10.1109/REAL.1990.128736
[16] E. Balafoutis, M. Paterkakis, and P. Triantallou, "Clustered Scheduling Algorithms for Mixed-Media Disk Workloads in a Multimedia Server," *Cluster Computing Journal*, Vol. 6, No. 1, pp. 75-86, 2003.
DOI: https://doi.org/10.1023/A:1020923202104
[17] Sungchae Lim, "The Dynamic Sweep Scheme using Slack Time in the Zoned Disk," *in Proc. 11st DASFAA,* pp. 404-419, April 12-15, 2006.
DOI: https://doi.org/10.1007/11733836_29
[18] A.L. Narasimha Reddy, "Improving Latency in an Interactive Video Server," in *Proc. of the Intl. Conference on Multimedia Computing and Networking*, pp. 108-112, 1997.
DOI: http://dx.doi.org/10.1117/12.264287
[19] Samsung SSD 850 Evo Specification, *Web URL:https://www.cnet.com/products/samsung-ssd-850-evo/specs/*