Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2017) 25: 3367 – 3380
© TÜBİTAK
doi:10.3906/elk-1610-271

Disk scheduling with shortest cumulative access time first algorithms

Nail AKAR[1,*], **Çağlar TUNÇ**[2], **Mark GAERTNER**[3], **Fatih ERDEN**[3]
[1]Department of Electrical and Electronics Engineering, Faculty of Engineering, Bilkent University, Ankara, Turkey
[2]Department of Electronics and Computer Engineering, Tandon School of Engineering, New York University, NY, USA
[3]Seagate Technology, Shakopee, MN, USA

Received: 21.10.2016 • **Accepted/Published Online:** 08.01.2017 • **Final Version:** 30.07.2017

Abstract: A new class of scheduling algorithms is proposed for disk drive scheduling. As opposed to choosing the request with the shortest access time in conventional shortest access time first (SATF) algorithms, we choose an ordered sequence of pending I/O requests at the scheduling instant with the shortest cumulative access time. Additionally, we introduce flexibility for forthcoming requests to alter the chosen sequence. Simulation results are provided to validate the effectiveness of the proposed disk scheduler. Throughput gains of 3% and above are shown to be attainable, although this occurs at the expense of increased computational complexity.

Key words: Disk scheduling, shortest access time first, sequential request processing

1. Introduction

Magnetic hard disk drive (HDD) is the principal data storage technology stemming from large capacities, low costs, and high reliability [1]. An HDD consists of one or more platters that rotate at a fixed speed. Each platter has two sides, each of which is referred to as a surface. Data are stored on concentric circles of sectors on each surface, called tracks. The disk head that is attached to the disk arm moves across the surface to position the head over the desired track for I/O read/write operations. The disk service time is the sum of two components: positioning time and transfer time. The positioning time is, again, the sum of two components: seek time and rotational latency. The seek time is the time required for the disk head to travel to the track of the disk where the data will be read from or written to. The seek comprises the following phases: acceleration, during which the arm starts moving; coasting, when the arm moves at full speed; deceleration, when the arm slows down; and settling, when the head is fine-positioned over the correct track [2]. Once the disk head is positioned on the right track, we need to wait longer, i.e. rotational latency, until the rotation of the disk brings the required disk sector under the read/write head. The sojourn time (or response time) of a request is the amount of time that a request needs to wait in the system until it is fully served. Sojourn time is the sum of two components: queue waiting time and service time. Typically, disk schedulers attempt to reduce the service times, which, in turn, reduces the sojourn times. A lower sojourn time is indicative of a higher throughput, which is defined as the mean number of I/O requests completed in unit time.

Disk performance, characterized in terms of I/O request latency or I/O throughput, is generally known to be poor compared to that of other storage media such as solid state disks (SSDs). This is due to the mechanical

*Correspondence: akar@ee.bilkent.edu.tr

nature of disk access time, i.e. the time required to position the disk head over the requested sector. In order to enhance disk performance, various disk scheduling algorithms have been proposed, which choose to serve one of the pending requests based on a certain policy. A disk scheduling algorithm may opt to maximize the overall disk throughput; or be fair to different requests or classes of requests; or prioritize a certain class of requests over others; or it may seek to minimize the computational effort placed for scheduling. In this study, we focus on overall disk performance enhancement, and fair disk bandwidth-sharing by different I/O requests is omitted from the scope of this paper.

In this article, we propose a new class of algorithms, called shortest cumulative access time first (SCATF) algorithms, which serve a sequence of pending requests at a scheduling instant, as opposed to serving one request only. Once the service of a sequence is started, new requests may or may not be allowed to change the service sequence, leading to different versions of the SCATF algorithm.

The paper is organized as follows. In Section 2, a brief survey of existing disk scheduling algorithms is presented. We describe the proposed SCATF algorithms in detail in Section 3. Section 4 presents the simulation results, and the final section provides the conclusions.

2. Related work

In this section, we review the existing disk-scheduling algorithms proposed in the literature. For a more recent and elaborate survey of disk schedulers, we refer the reader to [3].

2.1. First-come-first-serve scheduler

The first-come-first-serve (FCFS) scheduler serves the request that joined the system earliest, irrespective of how the disk head is positioned with respect to the sectors containing data for pending requests [1]. Although FCFS scheduling is commonly used elsewhere, it leads to poor performance in hard disk drives [3].

2.2. Shortest seek time first scheduler

The shortest seek time first (SSTF) scheduler is a greedy scheduler that first serves the requests on the nearest track to minimize seek time [4,5]. There are several drawbacks to using SSTF: i) it does not take into account the rotational delay, which may be (at least) equally important to the seek time in modern disk drives; ii) since the middle disk tracks are more likely to be chosen by the SSTF algorithm, the pending requests located close to the inner and outermost disk tracks may potentially starve, leading to unfairness among tracks; iii) the disk's internal details may not be available to the host OS, which may then only approximate SSTF, for example by a nearest block first (NBF) algorithm [1].

2.3. SCAN scheduler

The SCAN algorithm (also known as the elevator algorithm) sweeps the disk arm from the outermost cylinder towards the innermost cylinder and back, serving the pending disk requests along the way [6]. The SCAN scheduler has many variations. The LOOK policy reverses the direction of a scan, once there are no more outstanding requests in the current scan direction [7]. SCAN and LOOK schedulers visit the middle disk tracks twice as often than the inner and outermost disk tracks, again leading to unfairness among tracks. Cyclical variants of SCAN or LOOK, known as C-SCAN and C-LOOK, respectively, sweep the tracks in one direction only. However, once the sweep is complete, the disk arm returns to the starting track and continues to sweep in the same direction, eliminating the preferential treatment feature of the middle disk tracks of the original policies. SCAN-based policies are subject to starvation, for which starvation-reducing algorithms have been proposed. For example, the VSCAN(R) algorithm, proposed in [8], forms a continuum of algorithms between SSTF and LOOK, where the algorithm parameter R may be swept from $R = 0$ (pure SSTF) to $R = 1$ (pure LOOK), trading off overall performance and starvation. A time-complexity analysis of several variants of SCAN and SSTF is performed in [9].

2.4. Shortest access time first scheduler

The shortest access time first (SATF) policy is a greedy policy that serves the pending request with the shortest access time, i.e. service time with respect to the current head position; see [10,11]. SATF is slightly different from the shortest positioning time first (SPTF) policy, which includes the positioning time but not the data transfer time, whereas SATF includes both [3]. For fixed-size data requests, these two policies are equivalent, whereas SATF presents a slight preferential treatment to smaller data blocks in the case of variable-sized data blocks. The SATF algorithm has been studied extensively and has been shown to outperform other existing disk scheduling algorithms, such as SCAN, in terms of overall throughput [10–13]. Similarl to SSTF, SATF is known to have vulnerabilities in terms of I/O request starvation. To reduce starvation, enhancements have been proposed for SATF. For example, the aged shortest access time first (ASATF(ω)) algorithm, proposed in [10], forms a continuum between FCFS and SATF, where the algorithm parameter ω may be swept from $\omega = 0$ (pure SATF) to $\omega \to \infty$ (pure FCFS), thus trading off overall performance and starvation.

2.5. Other proposed disk schedulers

The authors in [14] present hard disk scheduling algorithms by defining a reachability function, which uses the radial distance to the location of a request as the input in order to increase the throughput. It does this by minimizing the number of rotations during the service of waiting requests. In other words, the set with the highest number of requests that can be visited in a single rotation is served first, and then the algorithm is applied to the remaining ones. The study in [15] modifies the algorithm in [14] to obtain the longest increasing subsequence in a permutation, assuming a probability distribution on the locations of I/O requests on the disk. As these studies show, increasing speed and memory in modern processors give rise to the question of how to improve the throughput performance of I/O scheduling without increasing its complexity. In [16], a new disk scheduling algorithm is proposed to reduce the number of head movements, thereby reducing seek time and improving the disk bandwidth for modern storage devices. The reference [17] employs fuzzy logic to optimize the overall performance of disk drives, considering both seek time and rotational latency.

3. Shortest cumulative access time first scheduling algorithms

As opposed to the studies described in the previous section, we focus on developing algorithms that minimize the cumulative access times of waiting requests, by considering all possible scheduling decisions whose complexity strongly depends on the algorithm parameters. The proposed algorithm, namely the shortest cumulative access time first (SCATF), has two variations, Version 1 ($SCATFv1$) and Version 2 ($SCATFv2$). $SCATFv1$ is introduced first. Algorithm $SCATFv1$ is characterized by a pair of algorithm parameters (J, L), represented by $SCATFv1(J, L)$. Parameter J determines the maximum number of steps for which the cumulative access time is to be calculated, whereas L is the maximum number of requests to be selected in each step of the calculation of the cumulative access time. Let us assume W pending I/O requests (or requests in short) at a scheduling instant at which $SCATFv1(J,L)$ chooses a sequence of pending requests of length $\bar{J} = \min(J,W)$. In this paper, a sequence is defined as an ordered list of distinct I/O requests. For the sake of simplicity, we first assume $W \geq J$. For the purpose of choosing an ordered sequence, in the first step of the algorithm, $SCATFv1(J,L)$ scans and orders all the pending requests in terms of access times from the current disk head, and finds $L_1 = \min(L,W)$ distinct requests denoted by $a_1^{(1)}, a_2^{(1)}, \ldots, a_{L_1}^{(1)}$, with shortest access times from the current disk head. Let us denote the set of requests obtained in the first step as $P_1 = \{a_1^{(1)}, a_2^{(1)}, \ldots, a_{L_1}^{(1)}\}$. In the second step, for each of the L_1 requests $a_{j_1}^{(1)}, 1 \leq j_1 \leq L_1$ obtained in the first step, we find $L_2 = \min(L, W-1)$ requests denoted by $a_{j_1,1}^{(2)}, a_{j_1,2}^{(2)}, \ldots a_{j_1,L_2}^{(2)}$, with shortest access times from the disk head, assuming that the disk head has just served the request $a_{j_1}^{(1)}, 1 \leq j_1 \leq L_1$. This construction gives rise to a set of L_1L_2 two-hop sequences, i.e. sequences of length two, denoted by $P_2 = \{(a_{j_1}^{(1)}, a_{j_1,j_2}^{(2)}), 1 \leq j_k \leq L_k, 1 \leq k \leq 2\}$. The cumulative access time of a two-hop sequence $(a_{j_1}^{(1)}, a_{j_1,j_2}^{(2)})$ is defined as the sum of the access time from the current disk head to request $a_{j_1}^{(1)}$, and the access time required by the disk head to move from $a_{j_1}^{(1)}$ to request $a_{j_1,j_2}^{(2)}$. We then scan and sort the sequences in P_2, so as to pick $\hat{L}_2 = \min(L, L_1L_2)$ of these sequences with the shortest cumulative access times, denoted by the set of sequences $C_2 = \{(a_{j_1}^{(1)}, a_{j_1,j_2}^{(2)})\}$ of cardinality $\hat{L}_2$. In the third step, for each of the $\hat{L}_2$ requests $a_{j_1,j_2}^{(2)}$ obtained as the last element of a sequence in C_2, we find $L_3 = \min(L, W-2)$ requests, denoted by $a_{j_1,j_2,1}^{(3)}, a_{j_1,j_2,2}^{(3)}, \ldots, a_{j_1,j_2,L_3}^{(3)}$ with minimum access times from the disk head, assuming that the disk head has just served the request $a_{j_1,j_2}^{(2)}$. This construction subsequently gives rise to the following set of three-hop sequences:

$$P_3 = \{(a_{j_1}^{(1)}, a_{j_1,j_2}^{(2)}, a_{j_1,j_2,j_3}^{(3)}), (a_{j_1}^{(1)}, a_{j_1,j_2}^{(2)}) \in C_2, 1 \leq j_3 \leq L_3\},$$

of cardinality $\hat{L}_2L_3$. At the end of the third step, we scan and sort the sequences in P_3 in terms of the cumulative access times in ascending order, so as to pick the highest $\hat{L}_3 = \min(L, \hat{L}_2L_3)$ of the sequences with minimum cumulative access times. These subsequently constitute the chosen set of sequences at the end of the third step, denoted by $C_3 = \{(a_{j_1}^{(1)}, a_{j_1,j_2}^{(2)}, a_{j_1,j_2,j_3}^{(3)})\}$ of cardinality $\hat{L}_3$. Here the cumulative access time of the sequence $(a_{j_1}^{(1)}, a_{j_1,j_2}^{(2)}, a_{j_1,j_2,j_3}^{(3)})$ is similarly defined as the sum of the cumulative access time of the two-hop subsequence $(a_{j_1}^{(1)}, a_{j_1,j_2}^{(2)})$ and the access time required for the disk head to move to request $a_{j_1,j_2,j_3}^{(3)}$, provided that the disk head has just served request $a_{j_1,j_2}^{(2)}$. This process then repeats for $J-1$ steps to obtain the following set of candidate sequences:

$$P_{J-1} = \{(a_{j_1}^{(1)}, a_{j_1,j_2}^{(2)}, \ldots, a_{j_1,j_2,\ldots,j_{J-1}}^{(J-1)}), (a_{j_1}^{(1)}, a_{j_1,j_2}^{(2)}, \ldots, a_{j_1,j_2,\ldots,j_{J-2}}^{(J-2)}) \in C_{J-2}, 1 \leq j_{J-1} \leq L_{J-1}\},$$

of cardinality $\hat{L}_{J-2}L_{J-1}$, where $L_k = \min(L, W-k+1), k \geq 1$. There are two versions of the algorithm $SCATFv1$, depending on how the final Jth step is executed. Version A of the proposed algorithm, named $SCATFv1A$, decides to serve one of the subsequences in P_{J-1} with the shortest cumulative access time, which is defined as the sum of the cumulative access time of the subsequence $(a_{j_1}^{(1)}, a_{j_1,j_2}^{(2)}, \ldots, a_{j_1,j_2,\ldots,j_{J-1}}^{(J-1)}) \in P_{J-1}$ and the shortest access time to some other request $a^{(J)}_{j_1,j_2,\ldots,j_J}$, given that the disk head has just served the request $a^{(J-1)}_{j_1,j_2,\ldots,j_{J-1}}$. Once the scheduling decision has been made, the disk serves the chosen J requests in the designated order. A slightly modified Version 2 of the same algorithm, called $SCATFv1B$, first picks $\hat{L}_{J-1}$ of the sequences in P_{J-1} with the shortest cumulative access times. Then we define the set of chosen sequences $C_{J-1} = \{(a^{(1)}_{j_1}, a^{(2)}_{j_1,j_2}, \ldots, a^{(J-1)}_{j_1,j_2,\ldots,j_{J-1}})\}$ with cardinality $\hat{L}_{J-1}$. This step is then followed by the very final step of choosing the sequence $(a^{(1)}_{j_1}, a^{(2)}_{j_1,j_2}, \ldots, a^{(J)}_{j_1,j_2,\ldots,j_J}) \in C_{J-1}$ with the shortest cumulative access time, which is the sum of the cumulative access times of the subsequence $(a^{(1)}_{j_1}, a^{(2)}_{j_1,j_2}, \ldots, a^{(J-1)}_{j_1,j_2,\ldots,j_{J-1}}) \in C_{J-1}$and the shortest access time to another request $a^{(J)}_{j_1,j_2,\ldots,j_J}$, given that the disk head has just served request $a^{(J-1)}_{j_1,j_2,\ldots,j_{J-1}}$. Newcoming requests need to wait until the next scheduling instant while another request in the sequence is being served, both in $SCATFv1A(J,L)$ and in$SCATFv1B(J,L)$. For the case of $W < J$, $SCATFv1A(J,L)(SCATFv1B(J,L))$ should be reduced to $SCATFv1A(W,L)(SCATFv1B(W,L))$, since it is not possible to serve a J-hop sequence in this case. However, at most, W-hop sequences are allowed.

Although cumulative access time-based algorithms may prove to be beneficial in certain scenarios, newcoming I/O requests are ignored until the next scheduling instant for the two variations of $SCATFv1(J,L)$. Algorithm $SCATFv2A(J,L)$ is the same as $SCATFv1A(J,L)$; when there are no request arrivals during the time the chosen sequence is served. However, when a new request arrival takes place after the service of the $(i-1)$st request of the J-hop sequence, but before the ith request, the service ending of the ith request becomes a new scheduling instant. In this case, $SCATFv2A(J,L)$ reruns algorithm$SCATFv1A(J-i,L)$, to opt for a new subsequence of length $(J-i)$ to serve after the disk head completes the service of the ith request, while taking into consideration the new arrival(s). Algorithm $SCATFv1B(J,L)$ is similarly extended to $SCATFv2B(J,L)$. For convenience, we describe $SCATFv1A(J,L)$ in Algorithm 1, since the extension of the algorithm to the remaining three algorithms, namely $SCATFv2A(J,L)$, $SCATFv1B(J,L)$, and$SCATFv2B(J,L)$, is straightforward.

In order to compare the computational complexities and storage requirements of the four proposed algorithms, we assume that the computation of access time from a fixed request to each of the remaining $W-1$ requests requires one CPU operation, whereas the remaining operations (comparison within an array of numbers, selecting and storing the minimum, etc.) are assumed to be negligible in terms of computational complexity. For convenience, we also assume that W is much larger than parameters J and L. At the first step of algorithm *SCATFv1A*, access times from the head to all the W requests are computed, which requires W CPU operations. At step i for $2 \le i \le J$, access times for each L^{i-1} request to the remaining $W-i+1 \approx W$ requests are computed. Overall, using algorithm *SCATFv1A* results in $W \sum\limits_{i=0}^{J-1} L^i$CPU operations to process J requests. Moreover, access times for roughly JL^{J-1} I/O requests should be stored in the memory. The only difference of *SCATFv1B* is that at the final step, access times are computed for$L$ requests instead of L^{J-1}. This reduces the number of CPU operations required to process J requests to $WL + W\left(\sum\limits_{i=0}^{J-2} L^i\right) \approx W\left(\sum\limits_{i=0}^{J-2} L^i\right)$ for large . Similarly, access times for roughly $(J-1)L^{J-2}$ requests should be stored in the memory instead of JL^{J-1}. On the other hand, each version of the *SCATFv2* algorithm executes the corresponding version of

Algorithm 1 $SCATFv1A(J, L)$ disk scheduling algorithm.

```
**Input**: Parameters $J, L,$ pending requests $W$, and the current position of the disk head
**Output:** Sequence of requests to serve
$\bar{J} \leftarrow \min(J, W)$
$L_1 \leftarrow \min(L, W)$
$P_1 \leftarrow$ the set of $L_1$ distinct requests with shortest access times from the current disk head
$i \leftarrow 2$
**while** $i \leq \overline{J} - 1$ **do**
$L_i \leftarrow \min(L, W - i + 1)$
$P_i \leftarrow$ the set of sequences, including $L_i$ distinct requests with shortest access times for each request in set $P_{i-1}$
$i \leftarrow i + 1$
end
$P_{\overline{J}} \leftarrow$ the set of sequences, including the distinct request with shortest access time for each request in set $P_{\overline{J}-1}$
Serve the sequence of requests with shortest cumulative access time in set $P_{\overline{J}}$
```

the *SCATFv1* algorithm J times, by processing one request at a time and reducing parameter J by 1 at each step. Resulting CPU per request (total required CPU operations divided by J) and storage requirements are provided in the Table, along with their values evaluated for the sample parameter set $(W, J, L) = (128, 8, 4)$.

Table. CPU and storage requirement expressions for the four proposed algorithms.

	SCATFv1A	*SCATFv1B*	*SCATFv2A*	*SCATFv2B*
CPU operations	$\frac{W}{J}\sum_{i=0}^{J-1} L^i$	$\frac{W}{J}\sum_{i=0}^{J-2} L^i$	$\frac{W}{J}\sum_{k=1}^{J}\sum_{i=0}^{J-k} L^i$	$\frac{W}{J}\sum_{k=1}^{J}\sum_{i=0}^{\max(J-k-1,0)} L^i$
$(W, J, L) = (128, 8, 4)$	87,376	21,840	465,984	116,480
Storage requirement	JL^{J-1}	$(J-1)L^{J-2}$	$(J-1)L^{J-2}$	$(J-1)L^{J-2}$
$(W, J, L) = (128, 8, 4)$	131,072	28,672	131,072	28,672

We now provide an illustrative example to describe the *SCATF* algorithms of interest. For this purpose, consider a hypothetical disk drive, illustrated in Figure 1, with one surface, 6 tracks, and 8 sectors per track, totaling 48 sectors on the single surface. For illustrative purposes, we assume the following simplistic disk model: one full disk rotation requires 16 ms, and the seek time between a source and destination track is assumed to be 4-ms times the distance between the two tracks. We assume all I/O requests require the transfer of one single sector. At time t_0, 4 requests, namely requests 1, 2, 3, and 4, are assumed to be pending, and the disk head resides at the outermost track. A scheduling decision is to be made at time $t_i, i \geq 0$, which will result in the service completion of a request at time t_{i+1}. We assume request $i + 4$ to have just joined the system by time t_{i+1}. The parameter pair (J, L) is assumed to be $(3, 2)$ for this illustrative example. In particular, we are interested in the scheduling decisions made by both versions of $SCATFv1(3, 2)$ and $SCATFv2(3, 2)$ at time $t_i, 0 \leq i \leq 2$.

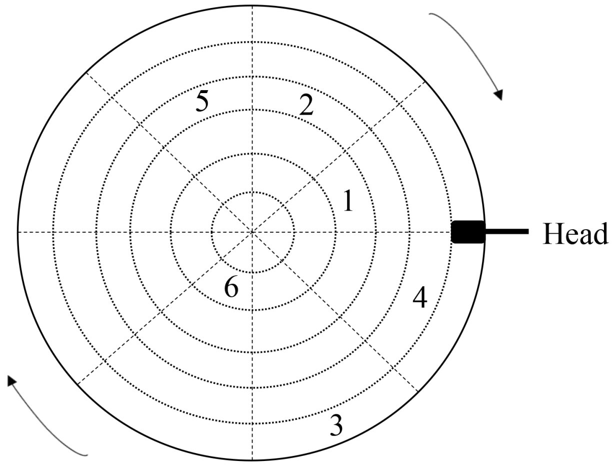

Figure 1. Hypothetical disk drive with one surface, 6 tracks, and 8 sectors per track.

Let us start with $SCATFv1(3,2)$. At time t_0, in the first step, there are $W = 4$ pending requests, and $SCATFv1(3,2)$ finds the set $P_1 = \{3, 4\}$ with cardinality $L_1 = 2$. Note that at time t_0, the positioning from the current disk head, i.e. access times minus the sector traversal times, to requests 3 and 4, are 12 and 14 ms, respectively, which are shorter than those to requests 1 and 2, which are 16 and 18 ms, respectively. Subsequently, in the second step, we obtain the set $P_2 = \{(3,1),(3,4),(4,1),(4,3)\}$. In the final step of the algorithm, $SCATFv1A(3,2)$ finds one single three-hop sequence out of the subsequences in P_2, namely sequence $(3,1,4)$, with the shortest cumulative access time being 42 ms. This amounts to deciding at time t_0 to serve requests 3, 4, and 1 at times t_0, t_1, and t_2, respectively. For algorithm$SCATFv1A(3,2)$,we further construct set $C_2 = \{(3,4),(4,3)\}$, which is a subset of P_2 of cardinality 2. In the final step, $SCATFv1B(3,2)$ finds two three-hop sequences out of the subsequences in C_2, namely $(3,4,1)$ and $(4,3,1)$, with the shortest cumulative access time being 44 ms, one of which will be chosen at random by $SCATFv1B(3,2)$. Evidently, the two versions of $SCATFv1(3,2)$ produced different sequence decisions, for both of which the new-coming requests 5 and 6 need to wait until t_3 to be considered for service.

Next, we describe the operation of $SCATFv2A(3,2)$. As in $SCATFv1A(3,2)$, the decision to serve the three-hop sequence $(3,1,4)$ is made at t_0. However, request 5 arrives just before the service completion of request 3, which occurs at time t_1. Algorithm $SCATFv1A(2,2)$ is then run with request 5 taken into account to find the shortest cumulative access time two-hop sequence, which turns out to be $(4,5)$. Once the service of request 4 is complete, request 6 has just arrived. In the final step, SATF is employed to decide to serve request 5. We observe that the cumulative access times of these three requests are 32 ms, if $SCATFv2A(3,2)$ is used.

Let us now describe the operation of $SCATFv2B(3,2)$. As in $SCATFv1B(3,2)$, the decision to serve one of the two sequences $(3,4,1)$ or $(4,3,1)$ at random, say sequence $(4,3,1)$, is made at t_0. However, request 5 arrives just before the service completion of request 3, which occurs at time t_1. The algorithm $SCATFv2B(2,2)$ is then run at time t_1, with request 5 taken into account to find the shortest cumulative access time two-hop sequence, which turns out to be $(5,1)$. Once the service of request 5 completes, request 6 has just arrived. In the final step, SATF is employed to decide to serve request 1. We observe that if $SCATFv2B(3,2)$ is used, the cumulative access times of these three requests is 28 ms. If SATF were used, requests 3, 4, and 5 would be served at times t_0, t_1, and t_2, respectively, yielding a total service time of 32 ms, which is the same as that of $SCATFv2A(3,2)$.

4. Numerical examples

In all the numerical examples to follow, we employ the HP 97560 SCSI disk drive, based on [10], with 10 platters and 19 data surfaces, 1964 physical cylinders, 72 sectors (512 bytes each) per track, yielding a total capacity of 1.38 GB. The disk speed is 4002 rpm and the settling time for switching between tracks in the same cylinder is ignored in the simulations. The seek time model is again inherited from [10]. Each request reads or writes two sectors of data. We note the differences with this general disk model used in all the numerical examples, and the simple low-capacity disk used in the hypothetical example, given in Figure 1 for illustrative purposes. Similar to the illustrative example of Figure 1, we assume that whenever a request is served, another new request immediately joins the system at a random location on the disk, leading to a fixed number of pending requests, which is denoted by Q. The disk model and scheduling mechanism of interest are implemented in MATLAB, and each simulation is terminated once 2 10^6 requests are served. All four proposed algorithms are

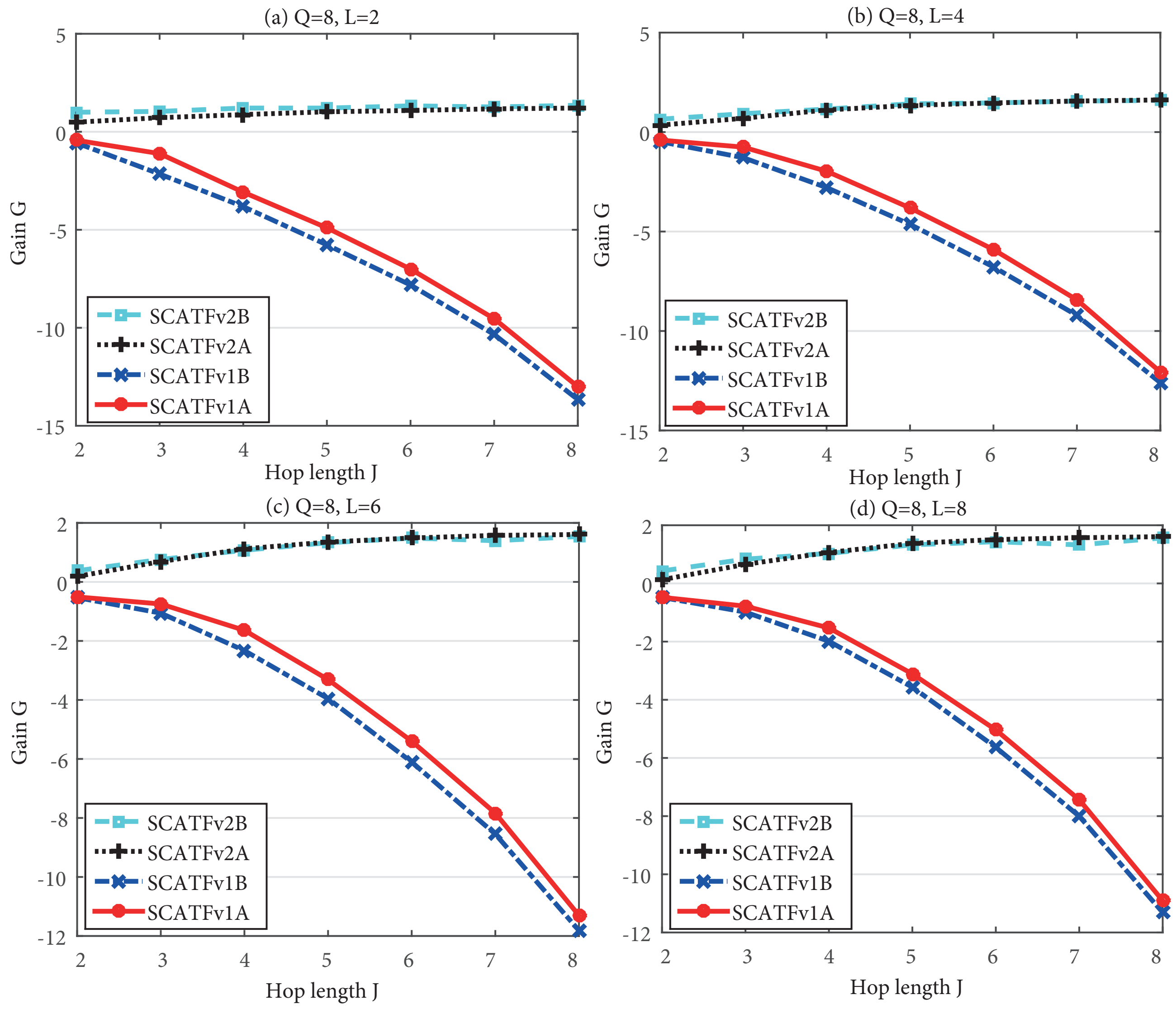

Figure 2. Percentage gain obtained by the proposed algorithms as a function of hop length J for $Q = 8$ and various values of L.

compared and contrasted against the conventional SATF algorithm by using a percentage gain metric G, which is defined as $G = \frac{T - T_{SATF}}{T_{SATF}} \times 100$, where T_{SATF} is the throughput of the system in units of I/O operations per second (IOPS), if SATF is to be used as the scheduler. T is the throughput of the system when one of the four variations of the proposed algorithm is to be deployed. A negative value for the metric G is indicative of a loss in throughput performance with respect to SATF.

Example 1 *In the first example, we plot the percentage gain obtained using the four proposed $SCATF$ algorithms as a function of the hop length J, for various choices the algorithm parameter $L \in \{2, 4, 6, 8\}$ and for three values of the fixed queue size $Q \in \{8, 32, 128\}$, shown in Figures 2, 3, and 4, respectively. For convenience, we also plot the throughput obtained using SATF and the four proposed $SCATF$ algorithms as a function of the hop length J for $Q \in \{8, 32, 128\}$, shown in Figures 5, 6, and 7, respectively. We have encountered sub-*

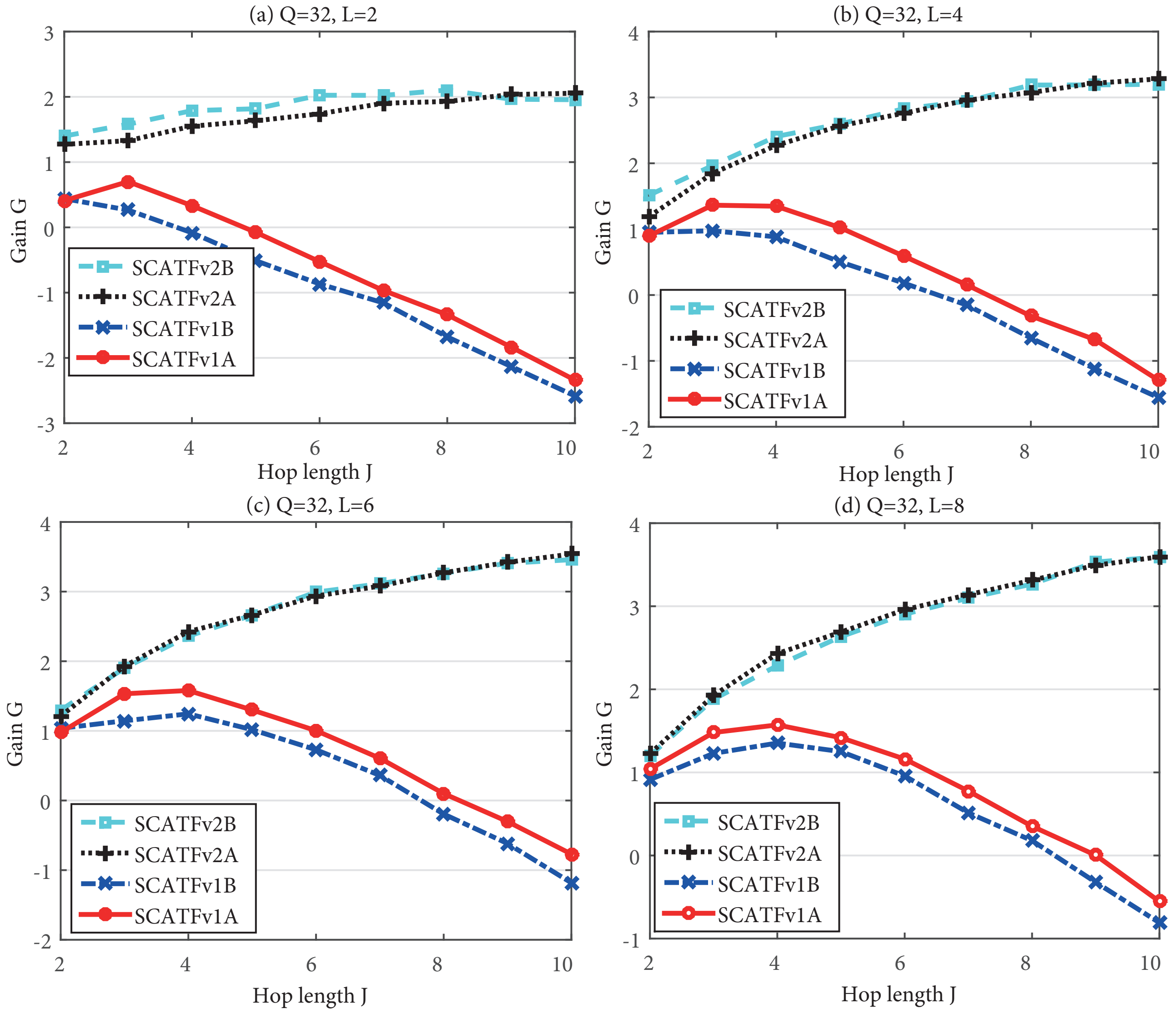

Figure 3. Percentage gain obtained by the proposed algorithms as a function of hop length J for $Q = 32$ and various values of L.

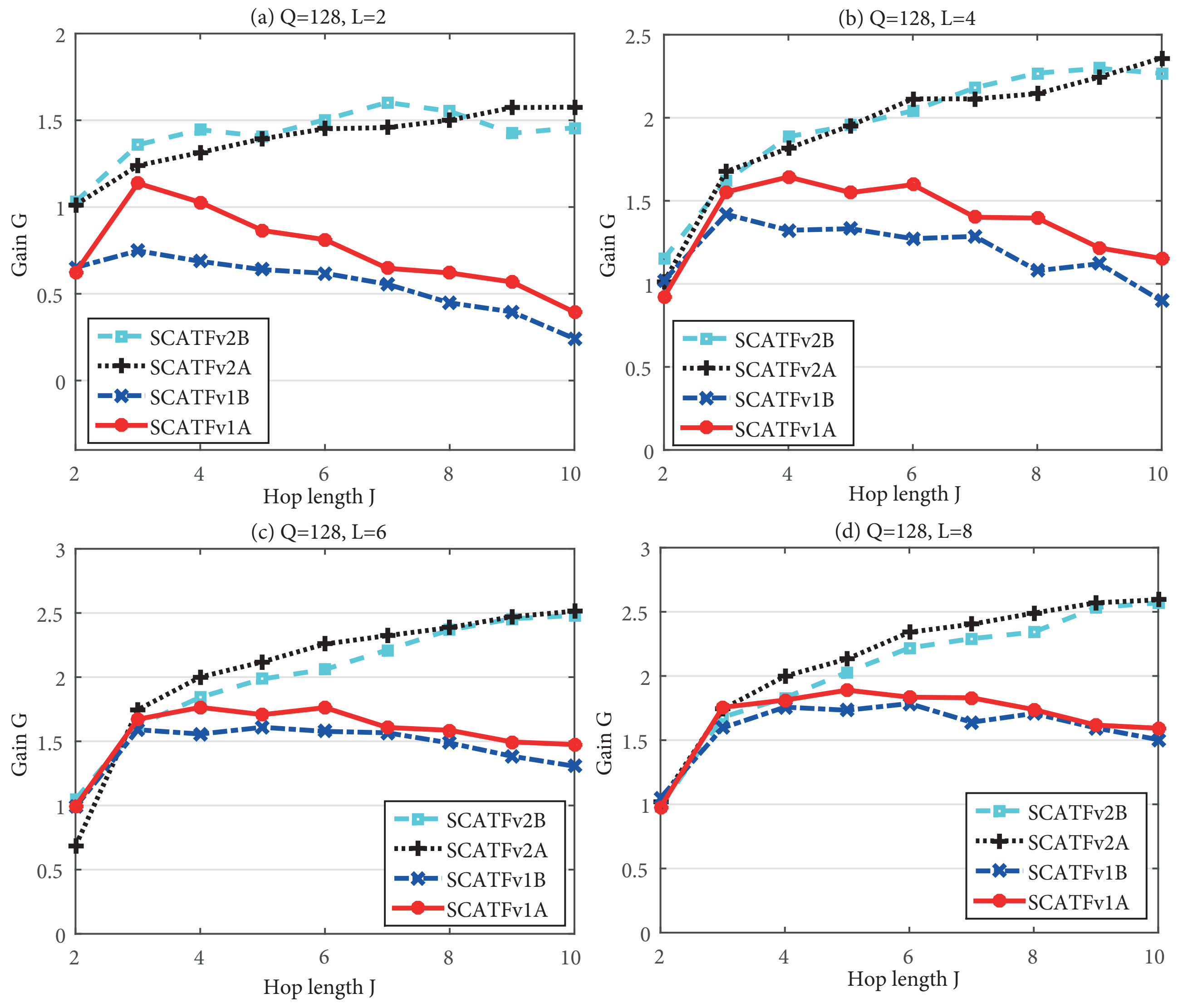

Figure 4. Percentage gain obtained by the proposed algorithms as a function of hop length J for $Q = 128$ and various values of L.

stantial performance losses with A and B variations of the $SCATFv1$ algorithm, with the loss increasing for smaller queue length Q and for larger hop length J. The reason for this observation is that the ignorance of new arrivals for smaller queue lengths and larger hop lengths leads to many wasted opportunities. On the other hand, for larger queue sizes and relatively low values of hop length $J, SCATFv1$ presents an improved throughput performance. Higher throughput values are obtained for both versions of the $SCATFv2$ algorithm, which appear to consistently increase with increased hop length J and, again, for increased algorithm parameter L. However, beyond certain values of the parameter pair (J, L), improvement in throughput is marginal. The A and B variations of the scheduling algorithm $SCATFv2$ provided very close results with $SCATFv2A$, slightly outperforming $SCATFv2B$ for larger queue lengths and larger hop lengths. However, we have observed cases where this ordering is slightly reversed in other scenarios. Based on the results obtained in Example 1 favoring $SCATFv2$, and $SCATFv2A$ in particular, we fix the disk scheduling algorithm to $SCATFv2A$ in the remaining numerical examples.

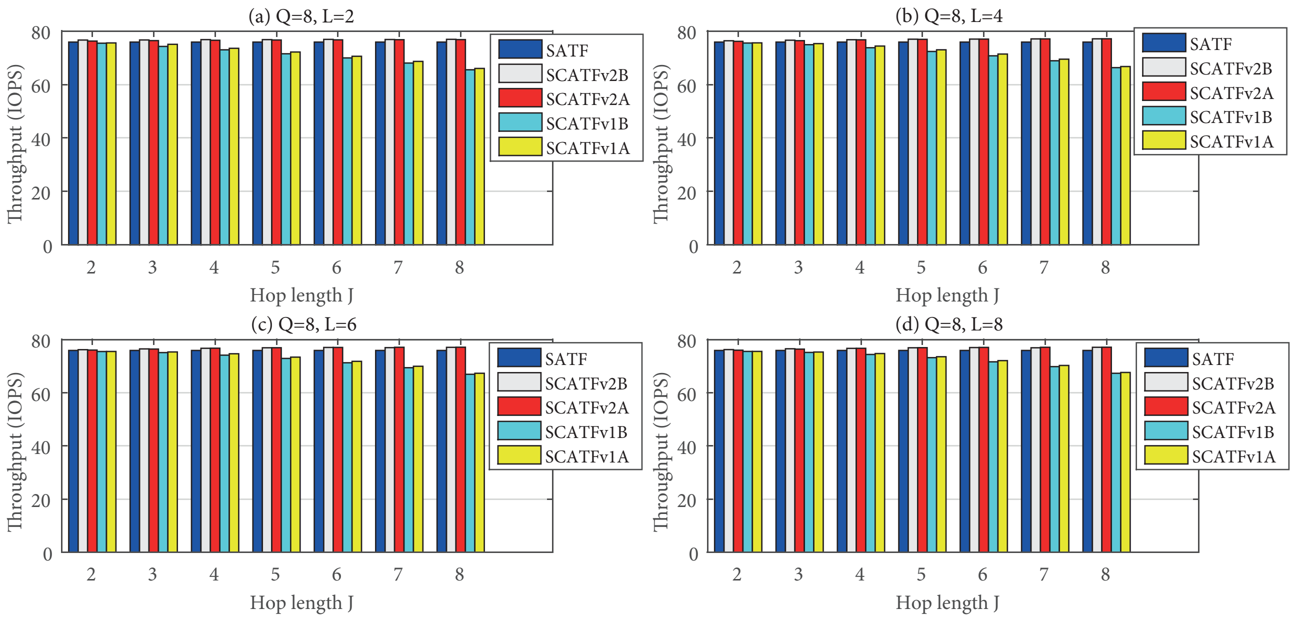

Figure 5. Throughput obtained by SATF and the proposed algorithms as a function of hop length J for $Q = 8$ and various values of L.

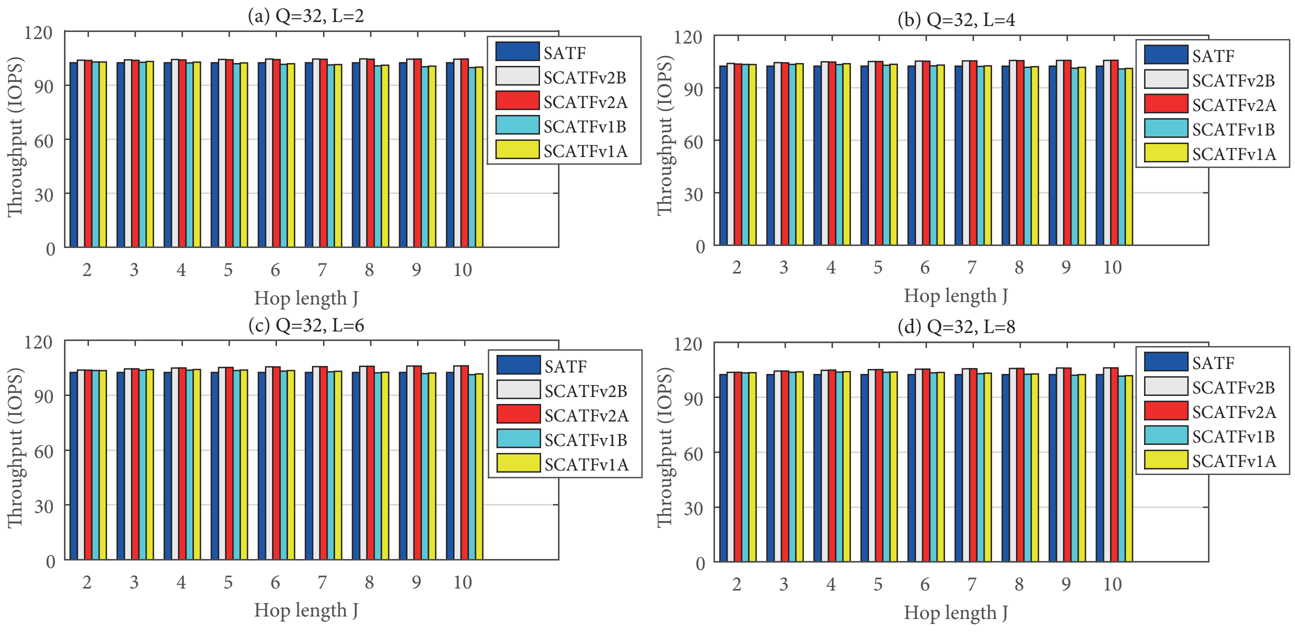

Figure 6. Throughput obtained by SATF and the proposed algorithms as a function of hop length J for $Q = 32$ and various values of L.

Example 2 *In this example, we study the performance gain G of $SCATFv2A$ disk scheduler as a function of hop length J for four values of algorithm parameter L, shown in Figure 8, and for varying values of the queue length Q: (a) $Q = 8$, (b) $Q = 12$, (c) $Q = 16$, (d) $Q = 32$, (e) $Q = 64$, and (f) $Q = 128$. We observe that the performance gain obtained by $SCATFv2A$ is always positive, with the gain monotonically increasing with increased hop length J for all values of queue size Q. We also observed that for relatively large queue lengths*

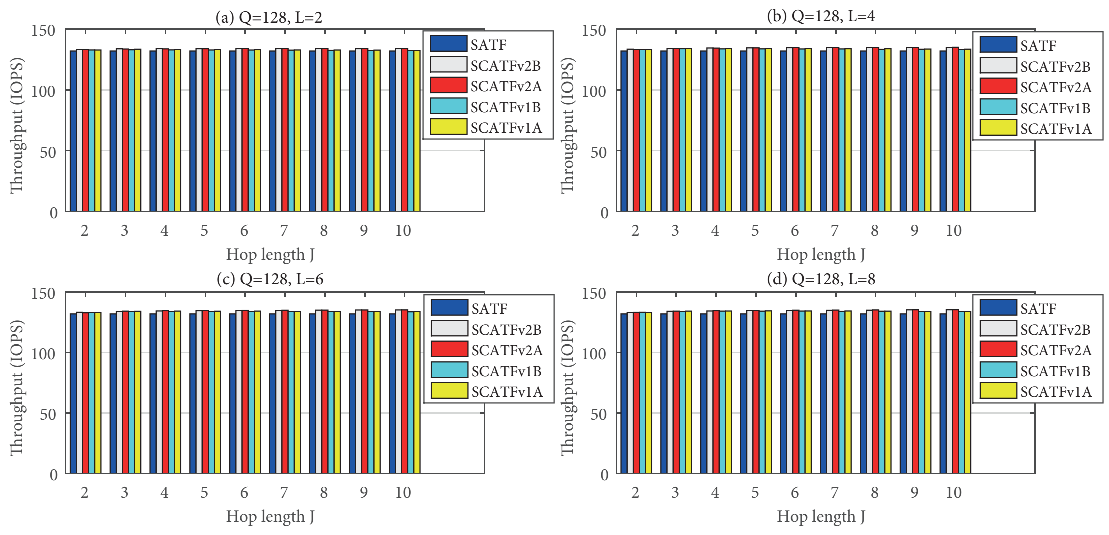

Figure 7. Throughput obtained by SATF and the proposed algorithms as a function of hop length J for $Q = 128$ and various values of L.

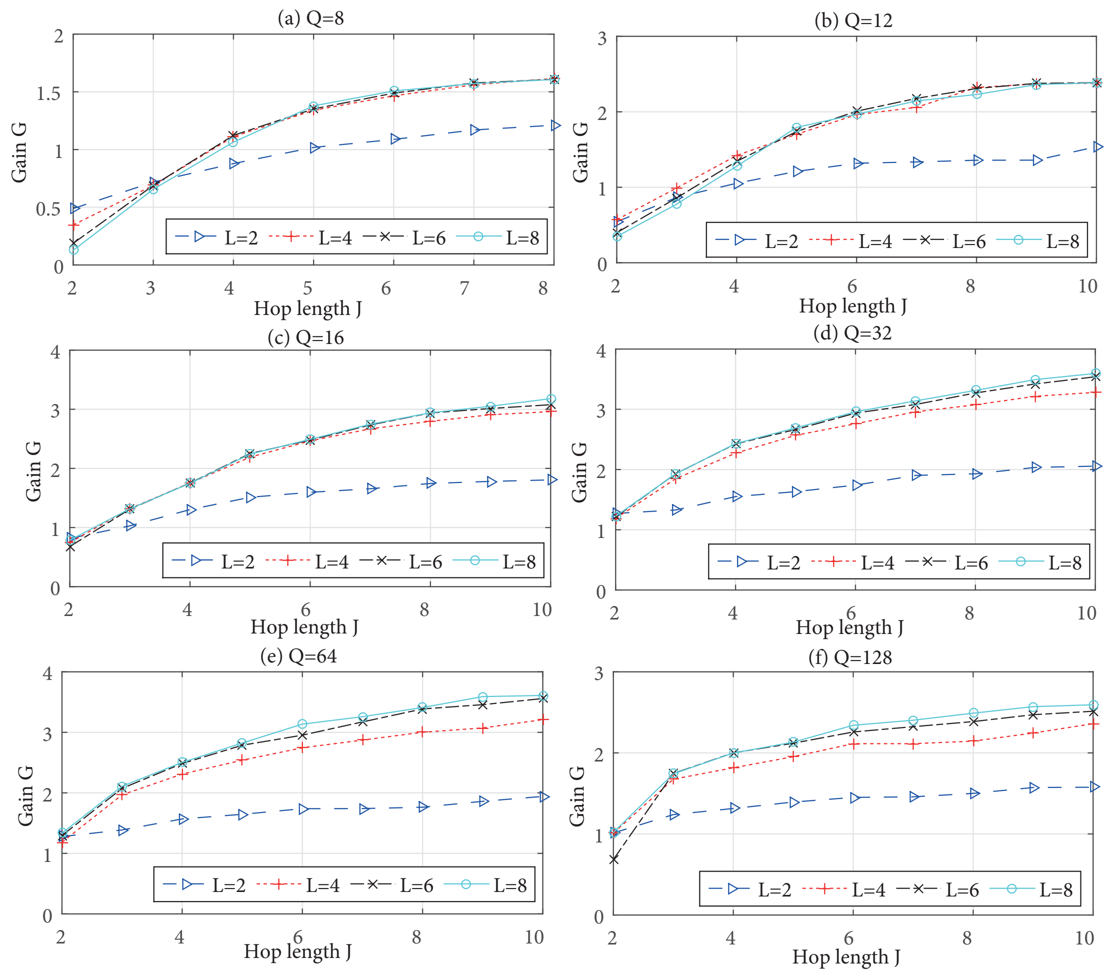

Figure 8. Performance gain G, obtained by using *SCATFv2A* as a function of hop length J for four values of the algorithm parameter L for varying values of queue length Q: (a) $Q = 8$, (b) $Q = 12$, (c) $Q = 16$, (d) $Q= 32$, (e) $Q = 64$, (f) $Q = 128$.

(for example $Q \geq 32$), gain G also increases monotonically with increased algorithm parameter L. However, for much smaller queue lengths, i.e. $Q < 32$, this relationship may not necessarily hold for relatively small values of J. Therefore, it is possible that increasing L may lead to a slight reduction in overall throughput in this particular regime.

Example 3 *In this example, we again employ $SCATFv2A$, although this time we plot the percentage performance gain G, shown in Figure 9 and obtained using the disk scheduler $SCATFv2A$, in comparison with $SATF$ as a function of queue size Q for various values of the algorithm parameter pair (J, L). We observe that the performance gain is not monotonic with respect to queue size Q for a given pair (J, L), and there is a certain value for Q such that this gain is maximum. The maximum-gain attaining queue size appears to be close to $Q = 64$ for most cases we studied. Hence, we observe a performance gain of 3.61% when $J = 10, L = 8$. Moreover, we observed that the maximum-observed gain increases with increased choices of J and L; however, beyond the choice of $L = 6$, the improvement is marginal. A similar statement can be made for parameter choice J by also taking into account the findings of Figure 8.*

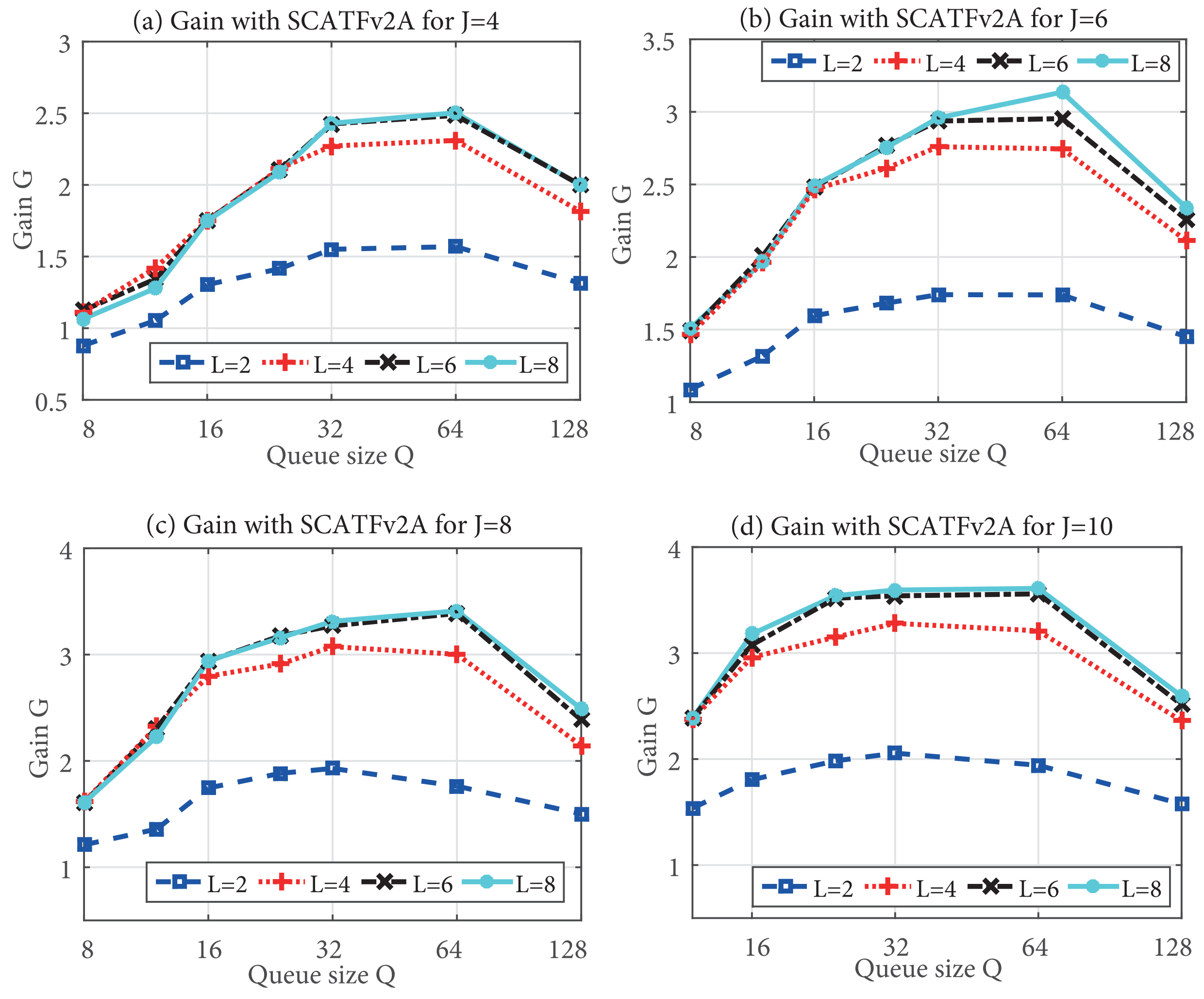

Figure 9. Performance gain G, obtained by using *SCATFv2A* as a function of queue size Q for various values of L and for two different choices of hop length J: (a) $J = 4$, (b) $J = 6$, (c) $J = 8$, (d) $J = 10$.

5. Conclusion

A new class of disk-scheduling algorithms is presented based on the choice of a sequence of pending I/O requests at the scheduling instant with the shortest cumulative access time. This introduces flexibility for forthcoming requests to alter the chosen sequence. The proposed algorithm requires the selection of two algorithm parameters, namely J and L, which refer to the sequence length and number of next-hops computed at each step of the algorithm, respectively. Through extensive simulations, it was shown that the $SCATFv2A$ algorithm can provide robust throughput performance gain in comparison to the conventional SATF algorithm for sufficiently large choices of J and L. The largest performance gain was obtained for moderate queue lengths, and up to 3.61% performance gains were observed. However, this increased performance gain is obtained at the expense of increased computational effort, which needs to be further explored for the efficient implementation of these algorithms.

References

[1] Deng Y. What is the future of disk drives, death or rebirth? ACM Comput Surv 2011; 43: 1-27.

[2] Arpaci-Dusseau RH, Arpaci-Dusseau AC. Operating Systems: Three Easy Pieces. Madison, WI, USA: Arpaci-Dusseau Books, 2015.

[3] Thomasian A. Survey and analysis of disk scheduling methods. ACM Comp Ar 2011; 39: 8-25.

[4] Denning PJ. Effects of scheduling on file memory operations. In: ACM 1967 Spring Joint Computer Conference; 18–20 April 1967; Atlantic City, NJ, USA. New York, NY, USA: ACM. pp 9-21.

[5] Teorey TJ, Pinkerton TB. A comparative analysis of disk scheduling policies. ACM Commun 1972; 15: 177-184.

[6] Coffman EG, Klimko LA, Ryan B. Analysis of scanning policies for reducing disk seek times. SIAM J Comput 1972; 1: 269-279.

[7] Merten AG. Some quantitative techniques for file organization. PhD, University of Wisconsin, Madison, WI, USA, 1970.

[8] Geist R, Daniel S. A continuum of disk scheduling algorithms. ACM T Comput Syst 1987; 5: 77-92.

[9] Chen TS, Yang WP, Lee R. Amortized analysis of some disk scheduling algorithms: SSTF, SCAN, and N-Step SCAN. BIT 1992; 32: 546-558.

[10] Jacobson DM, Wilkes J. Disk Scheduling Algorithms Based On Rotational Position. Palo Alto, CA, USA: Hewlett Packard, 1991.

[11] Seltzer M, Chen P, Ousterhout J. Disk scheduling revisited. In: USENIX 1990 Technical Conference; 22–26 January 1990; Washington, DC, USA. Anaheim, CA, USA: USENIX Association. pp. 313-324.

[12] Thomasian A, Liu C. Some new disk scheduling policies and their performance. In: ACM 2002 Measurement and Modeling of Computer Systems Conferences; 15–19 June 2002; Marina del Rey, CA, USA. New York, NY, USA: ACM. pp. 266-267.

[13] Worthington BL, Ganger GR, Patt YL. Scheduling algorithms for modern disk drives. In: ACM 1994 Measurement and Modeling of Computer Systems Conference; 16–20 May 1994; Nashville, TN, USA. New York, NY, USA: ACM. pp 241-251.

[14] Zhang AB. New algorithms for disk scheduling. Algorithmica 2002; 32: 277-301.

[15] Bachmat E. Average case analysis of disk scheduling, increasing subsequences and spacetime geometry. Algorithmica 2007; 49: 212-231.

[16] Mahesh Kumar MR, Renuka Rajendra B. An improved approach to maximize the performance of disk scheduling algorithm by minimizing the head movement and seek time using sort mid current comparison (SMCC) algorithm. Procedia Comput Sci 2015; 57: 222-231.

[17] Hooda P, Raheja S. A new approach to disk scheduling using fuzzy logic. J Comput Commun 2014; 2: 1-5.