Abstract

Proportional-share algorithms are designed to allocate an available resource, such as a network, processor, or disk, for a set of competing applications in proportion to the resource weight allotted to each. While a myriad of proportional-share algorithms were made for network and processor resources, little research work has been conducted on disk resources, which exhibit non-linear performance characteristics attributed to disk head movements. This paper proposes a new proportional-share disk-scheduling algorithm, which accounts for overhead caused by disk head movements and QoS guarantees in an integrated manner. Performance evaluations via simulations reveal that the proposed algorithm improves I/O throughput by 11–19% with only 1–2% QoS deterioration.

1. Introduction

The prevalence of streaming services increases the chances for disk resource sharing. As a result, the traffic control on the disk resource called storage Quality of Service (QoS) is gaining in significance in order to satisfy the requirements of different applications [1–3]. It is known that partitioning such disk resources as bandwidth helps to satisfy the different QoS requirements of various types of applications, such as best-effort applications and real-time applications [3]. Despite its importance, research on storage QoS is still in its infancy, having mainly focused on underlying disk-scheduling algorithms.

A disk resource exhibits different characteristics from other resource types, such as processors and networks, because high overhead is typically involved in processing I/O requests. Few disk scheduling algorithms proportionally share disk resources, such as YFQ [2] and Cello framework [3]. Among them, the YFQ algorithm is based on packet-based fair queuing.
algorithms in the network, i.e., Weighted-Fair Queuing [4] to choose a subsequent I/O request to be scheduled and Start-time Fair Queuing [5] to maintain a global virtual time. Considering that achieving the ultimate QoS guarantee requires integrated scheduling and management for various resources in the underlying system, such as a processor, network, or disk, the YFQ-like approach with a packet-based fair queuing algorithm is preferable. Unfortunately, proportional-share disk-scheduling algorithms, while preserving a given QoS feature, inevitably suffer from performance degradation in order to improve their disk I/O performance.

For example, the YFQ algorithm first selects a batch of I/O requests mainly based on a QoS guarantee, and then attempts to reduce disk head movement overhead by reordering the batched requests. After investigating the operations of this scheduling algorithm, we determined that the effectiveness of disk-overhead reduction is restricted by separating the operation of I/O request selection from the operation of reducing disk overhead. Moreover, while we can achieve a better I/O performance with a larger batch size, the size of the batch cannot be arbitrarily increased in actual systems.

This paper proposes a new proportional-share disk-scheduling algorithm to consider the issues of disk I/O overhead reduction and I/O bandwidth provisioning in an integrated manner, thus overcoming the shortcomings of limited batch sizes found in the YFQ algorithm. Our basic idea derives from the fact that combining the operation of I/O request selection with the operation of reducing disk overhead will increase the odds of reducing disk overhead while maintaining a certain level of QoS guarantee. The remainder of this paper is organized as follows. Section 2 gives a description of the proposed algorithm. Section 3 provides various simulation results under various synthetic workloads and their analysis. Finally, this paper concludes in Section 4.

2. The proposed algorithm

Fig. 1 presents a set of components of the proposed algorithm to generate an I/O sequence that not only enhances disk I/O throughput, but also preserves a given level of QoS feature. This section will provide a detailed description of each component. We begin by providing a few notations.

2.1. Nomenclature

Assume that \( N \) different I/O workloads exist. An I/O request from the \( i \)th I/O workload arrives at the \( k \)th reservation queue. The \( k \)th reservation queue is denoted as \( RQ_k = \{ r_1^k, r_2^k, \ldots \} \), where \( r_j^k \) refers to the \( j \)th I/O request in the \( k \)th reservation queue. The notation of \( r_j^k \) represents the \( j \)th I/O request from a backlogged reservation queue. The notation of \( r_j^k \) represents the request size of \( r_j^k \). \( RQ_k \) requires a different QoS requirement denoted by \( \phi_k \), i.e., a different amount of disk bandwidth. An actual bandwidth allotted to the \( i \)th I/O workload at time \( t \) is \( \phi_i r_j^k W \), where \( W \) is the underlying disk bandwidth and \( B(t) \) is a set of indexes of the backlogged reservation queues at time \( t \). As with packet-based fair queuing algorithms [4,5], each reservation queue \( RQ_k \) maintains a virtual start time and a virtual finish time, denoted by \( St \) and \( Fs \), respectively. Besides, a global virtual time denoted by \( v(t) \) is also maintained.

2.2. Overall architecture

The proposed algorithm consists of a set of reservation queues (RQ), the base QoS sequence generation (BQS) module, and the disk overhead reduction (DOR) module, as shown in Fig. 1. I/O requests from different I/O workloads are backlogged at their associate reservation queues. Next, the proposed algorithm aims to produce an expanded sequence of I/O requests denoted by \( S_{\text{expanded}} \), which not only preserves a given QoS feature, but also achieves a better disk I/O performance based on a base sequence of I/O requests denoted by \( S_{\text{base}} \), which is generated by the strict QoS-enforcing module (the BQS module). Each I/O workload has its own reservation queue, called \( RQ_k \), to which a corresponding disk bandwidth \( \phi_k \) is allocated.

2.2.1. BQS module: generating a base I/O sequence

The BQS module generates \( S_{\text{base}} \) from I/O requests in \( RQ \) us with other fair queuing based algorithms. Our current design employs the fair queuing scheme of the YFQ algorithm. It first removes an I/O request from the reservation queue of the minimum virtual finish time, say \( v(t) \) of \( RQ_i \). Second, it advances the global virtual time, such that \( v(t) = St \). Third, it updates \( S_{t} = Fs \) and
Fig. 1. Architecture of the proposed algorithm.

\[ F_s = S_s + \frac{l^2_s}{\phi_s} \text{ if } RQ_s \text{ is still backlogged, where } l^2_s \text{ is the size of the next I/O request of } r^1_s. \] Otherwise, \( S_s \) and \( F_s \) remain unchanged. When a new request \( r^1_i \) arrives at the empty \( RQ_i \), then \( S_i = \max(v(t), F_i) \) and \( F_i = S_i + \frac{l^2_i}{\phi_i} \). The maximum number of I/O requests selected by the BQS module is tunable. Note that the maximum number of I/O requests in the BQS module is set to four in our experiments, because the outstanding I/O requests of a single disk is typically set to four. After having chosen a number of I/O requests, the BQS module reorders them to reduce the overhead of disk head movements by using one of existing disk scheduling algorithms, such as SATF [6], C-SCAN, SSTF [7], etc. Of these, our current design employs the C-SCAN algorithm. Consequently, it produces a base I/O sequence, denoted by \( S_{base} \). Algorithm 1 summarizes the behavior of the BQS module.

Obviously, all I/O requests of the BQS_IO can commit a given QoS guarantee.

**Algorithm 1.** The base sequence generation module

**Data:** \( S_{base} = \emptyset \)

**Result:** \( S_{base} \)

**\( N \) — the length of the BQS sequence;**

**begin**

while \( |S_{base}| \neq N \text{ and } \forall RQ_k \neq \emptyset \) do

remove \( r^1_i \) from \( RQ_i \) of \( F_i = \min_i(F_i) \); 

\[ v(t) = S_i; \quad S_i = F_i; \quad F_i = S_i + \frac{l^2_i}{\phi_i} \] if \( RQ_k \neq \emptyset; \quad S_{base} = S_{base} \cup r^1_i; \)

**end**

reorders the I/O requests in \( S_{base} \) according to the C-SCAN algorithm;

**end**
2.2.2. DOR module: generating an expanded I/O sequence

The DOR module consists of the disk head movement overhead time estimator (DME) and the dynamic QoS enforcer (DQE). It generates an expanded I/O sequence with the insertion of I/O requests into the given sequence with the insertion of I/O requests into the QoS enforcer (DQE). It generates an expanded I/O sequence with the insertion of I/O requests into the available disk head movement overhead and a given level of QoS guarantee. Denote the expanded I/O sequence with \( S_{\text{expanded}} \). The two properties depend largely on the two controlling parameters, \( M_{\text{dmo}} \) and \( M_{\text{qos}} \). The succeeding paragraph provides a detailed explanation of each parameter.

### 2.2.2.1. DME module: estimating overhead times for disk head movements

The DME module determines whether an extra I/O request in the backlogged reservation queues can be inserted into \( S_{\text{current}} \) by estimating the disk overhead time that exists in the sequence. The notation of \( S_{\text{current}} \) represents an intermediate I/O sequence in transit from \( S_{\text{base}} \) to \( S_{\text{expanded}} \). An overhead of a disk head movement can be represented by a seek time or an access time that includes both a seek time and a rotational delay. As for seek-time-based estimations, Shindler and Ganger [8] introduced a technique to automatically extract the exact information of a seek time curve of an underlying disk. However, the access-time-based estimation requires the additional prediction of a rotational delay. Although recent research work [9,10] has introduced techniques to estimate an additional virtual time displacement among the current additional virtual times of the backlogged reservation queues, as with \( F_i \) in the backlogged reservation queues, the schemes are too complicated to be used compared with those that predict the seek time. Thus, our current design uses a seek-time-based estimation. However, since this estimation ignores overhead caused by a rotational delay, we devise a marginal overhead time denoted by \( M_{\text{dmo}} \) in order to reduce this type of estimation error. The value of \( M_{\text{dmo}} \) will be added to the estimated disk head movement overhead time between two I/O requests in \( S_{\text{expanded}} \). Note that the odds increase that an extra I/O request can be squeezed into the sequence, as \( M_{\text{dmo}} \) becomes larger. A proper value of \( M_{\text{dmo}} \) will be empirically obtained in the subsequent section. Property 1 clarifies the meaning of the marginal overhead time \( M_{\text{dmo}} \). To begin, denote with \( \{R_1', R_2', \ldots, R_M'\} \) a set of I/O requests in the current \( S_{\text{current}} \), where \( M \) is the number of I/O requests in the current \( S_{\text{current}} \). \( T_{\text{dmo}}(R', R^{+1}) \) means the disk head movement overhead time of each disk head movement represented by \( R' \) in \( R^{+1} \). The two properties depend largely on the two controlling parameters, \( M_{\text{dmo}} \) and \( M_{\text{qos}} \). The succeeding paragraph provides a detailed explanation of each parameter.

#### Property 1:

The request of \( R_i' \) is insertable between \( R^t \) and \( R^{t+1} \) if it satisfies the following inequality:

\[
T_{\text{dmo}}(R', R_i') + T_{\text{dmo}}(R_i', R^{+1}) \leq T_{\text{dmo}}(R', R^{+1}) + M_{\text{dmo}}
\]

The overhead margin of each disk head movement of \( M_{\text{dmo}} \) represents a percentage of the full seek time of the underlying disk. According to Property 1, we expect the length of \( S_{\text{expanded}} \) to be longer in proportion to the value of \( M_{\text{dmo}} \). However, a large \( M_{\text{dmo}} \) is not always desirable due to an improper estimation of a rotational delay. However, recall that a final length of \( S_{\text{expanded}} \) is also affected by \( M_{\text{qos}} \).

### 2.2.2.2. DQE module: preserving a QoS feature with DOR/JO requests

The DQE module controls the degree of QoS enforcement for given I/O requests, which have successfully passed the DME module. For this purpose, it keeps an additional QoS enforcement scheme for the insertion of the additional I/O requests to the base I/O sequence of \( S_{\text{base}} \). It maintains an additional virtual time for each \( R_Q \) and the maximum displacement among such additional virtual times. Denote with \( f_j \) the additional virtual finish time of \( R_Q \). Initially, \( f_i = 0 \). If an I/O request arrives at an empty \( R_Q \), \( f_i = \max\{f_i, \min_{j,j<i}(f_j)\} \) for \( R_Q \neq \emptyset \). Denote with \( h_j \) the maximum displacement among the current additional virtual times of the backlogged reservation queues. It is defined as \( h_j = \max\{|f_i - f_j|\} \) for the backlogged reservation queues, where \( 1 \leq i, j \leq N \). The additional virtual times are based on the virtual finish times, as with \( F_i \) in the BQS module. However, the DQE schedules any of the I/O requests that have passed the DME module unless the maximum virtual time displacement goes beyond the QoS marginal value, \( M_{\text{qos}} \).

#### Property 2:

The request of \( R_i' \) preserves a given QoS feature of \( M_{\text{qos}} \) if it meets the following inequality:

\[
h_f \leq M_{\text{qos}} \text{ when } f_i = f_k + \frac{1}{B}
\]

\( M_{\text{qos}} = \beta \) implies that a set of virtual times will advance with an amount of QoS deterioration, such that the resulting I/O throughput of each \( R_Q \) will deviate from a given QoS feature on the average by \( \frac{\beta}{B} \) I/O requests, where \( B \) is an average block size of an I/O request from \( R_Q \). According to Property 2, no I/O requests can pass the DOR module with \( M_{\text{qos}} = 0 \).
cause no QoS unfairness is allowed. This implies that a given QoS feature is strictly preserved with the BQS module. Denote with \(DOR_{IO}\) a group of additional I/O requests squeezed into the given base I/O sequence \(S_{base}\) by the DOR module. Algorithm 2 summarizes the behavior of the DOR module with the two properties.

**Algorithm 2.** The disk overhead reduction module

**Data:**
\[S_{interim} = S_{base}\]

**Result:**
\[S_{expanded} = S_{interim} + DOR_{IO}\] requests

begin
   \[// S_{interim} = \{R_1, R_2, \ldots, R_M\};\]
   \[R_{head} = R_1; R_{tail} = R_N; R_{next} = R_{head} + 1;\]
   \[\text{while } R_{cur} \neq R_{tail} \text{ do}\]
   for each \(r_i\) in \(R_{Qk} \in R_{Q}\) do
      if \(r_i\) meets Property 1 and Property 2 then
         insert \(r_i\) between \(R_{cur}\) and \(R_{next}\) in \(S_{interim}; R_{cur} + 1 = R_{next}^j; f_j = f_j + \frac{\phi_k}{\phi_1}; R_{next} = r_i;\)
      break;
   end
   \[R_{cur} = R_{next}; R_{next} = R_{next} + 1;\]
end

3. Performance evaluations

This section evaluates the performance of the proposed algorithm by discovering two desirable values of \(M_{dmo}\) and \(M_{qos}\) that not only improve disk I/O performance, but also preserve a given QoS feature with negligible deterioration. We begin by describing the simulation environment for our performance evaluations.

3.1. Simulation environment

We implemented the proposed algorithm as a driver-specific disk scheduling scheduler within the DiskSim simulator [11]. This simulator employs the following parameters for I/O workloads, the underlying disk, and the proposed algorithm itself.

3.1.1. I/O workloads

We generate two competing I/O workloads synthetically. The request size of the I/O requests are distributed normally with a mean of eight blocks of 512 bytes. The ratio of reads to writes is set to two, as used in other QoS work [1]. An I/O workload level becomes heavier by increasing the number of outstanding I/O requests denoted with \(|IO|\), not by reducing the inter-arrival time, in order to control the increase of the queue depth. A start block address of each I/O request is distributed randomly over the entire space of an IBM DINES 309170W SCSI disk which serves arriving I/O requests in a FIFO manner with a maximum of four outstanding concurrent I/O requests. The full seek time of the disk is 17.742 ms and its seek-time curve is available at [12].

3.1.2. Parameters for the proposed algorithm

We have two reservation queues that are \(R_{Q1}\) and \(R_{Q2}\). In addition, the reservation queues are initially configured as \(\phi_1 = 80\) and \(\phi_2 = 20\). With the QoS requirements, the BQS module selects four I/O requests from the head of the reservation queues and then generates \(S_{base}\) where the I/O requests are reordered in a C-SCAN order, as described in Algorithm 1. Next, the DOR module generates \(S_{expanded}\) by expanding the given \(S_{base}\) with a group of extra I/O requests, as given in Algorithm 2. As noted, two controlling parameters exist, which are an overhead margin of \(M_{dmo}\) for the DME module and a QoS margin of \(M_{qos}\) for the DQE module. A combination of these two parameters determines the length of the \(S_{expanded}\), which will eventually characterize the behavior of the DOR module.

3.2. Performance results

To locate two desirable controlling parameters for a given disk, we take the following two steps. First, while strictly maintaining a given QoS feature, we seek the \(M_{best}^{dmo}\) that provides the most proper adjustment to the disk overhead in order to maximize disk I/O performance. Second, by relaxing the degree of QoS enforcement with \(M_{best}^{dmo}\), we attempt to obtain \(M_{best}^{qos}\) which provides a better disk I/O performance with acceptable QoS deterioration. Finally, the proposed algorithm with
M_{dmo}^{best} and M_{qos}^{best} will be compared with the YFQ algorithm in terms of the disk I/O performance and QoS guarantee.

3.2.1. Finding M_{dmo}^{best} controlling parameter

Recall that M_{dmo} was invented to control the slack time for the overhead associated with a disk head movement between two I/O requests. Given a fixed M_{qos}, a larger M_{dmo} will increase the number of I/O requests squeezed into S_{base} to exploit the estimated overhead of the disk head movements. Conversely, no I/O requests are allowed to pass through the DOR module with M_{dmo} = 0. Fig. 2 shows the I/O throughput of the proposed algorithm with different M_{dmo} under various I/O workload levels where M_{qos} = 0.5. We expect that the given QoS feature will be preserved throughout all I/O workload levels because of M_{qos} = 0.5, which corresponds to a strong QoS enforcement. By definition in Property 2, M_{qos} = 0.5 corresponds to a strict QoS enforcement equal to that of the BQS module, i.e., five I/O requests for RQ_1 and an I/O request for RQ_2 on the average. The performance results in Fig. 2 reveal that the best I/O throughput is obtained with M_{dmo} = 20.

Fig. 3 validates the obtained M_{dmo} by examining the variation of I/O throughputs as a function of M_{dmo} under fixed I/O workload levels, |IO| = 11 and |IO| = 61. As with the previous results, the maximum I/O throughput was achieved when M_{dmo} = 20. Observe that large M_{dmo} close to 100 cannot improve the I/O throughput due to improper overhead estimations. Conversely, a small M_{dmo} decreases the probability that the remaining I/O requests within RQ can utilize the existing disk head movement overhead.

Fig. 4 shows the variations of a length of S_{expanded} as a function of M_{dmo}. A length of S_{expanded} is saturated to about 8 when M_{dmo} becomes 20. Recall that the main idea of the proposed algorithm is to exploit the overhead times caused by disk head movements. Thus, the given S_{base} will be expanded by inserting a group of I/O requests which can use the available overhead time of disk head movements. However, overestimating the disk overhead time may cause performance degradation. Finally, we conclude the first performance
Fig. 4. Variations of |Sexpanded| with different I/O workload levels and \( M_{dmo} \) values with \( M_{qos} = 0.5 \).

Fig. 5. Variations of I/O throughputs as a function of an I/O workload level under different \( M_{qos} \) values.

Evaluation with the observation that \( M_{dmo} \) of 20 maximizes the I/O throughput under the condition of a strict QoS enforcement which is the same as that of the BQS module.

3.2.2. Finding \( M_{qos} \) controlling parameter

A further performance enhancement can be achievable by relaxing the degree of QoS enforcement, i.e., by increasing the \( M_{qos} \) value. However, we can expect that the larger \( M_{qos} \) will generate an I/O sequence having a higher QoS deterioration, even if it can provide a higher I/O throughput. Fig. 5 depicts the variations of I/O throughputs under different \( M_{qos} \) values, where the \( M_{dmo} \) is always set to 20. The resulting performance of \( M_{qos} \geq 5 \) is higher than that of \( M_{qos} = 0.5 \) by over 10% in terms of the aggregate I/O throughputs of \( RQ_1 \) and \( RQ_2 \).

Figs. 6 and 7 examine the variation of I/O throughput at each reservation queue as a function of \( M_{qos} \) under fixed I/O workload levels, \(|I/O| = 11\) and \(|I/O| = 61\). Fig. 6 shows that the I/O throughput of the \( RQ_2 \) decreases, whereas the I/O throughput of its competing I/O workload \( RQ_1 \) improves by increasing \( M_{qos} \). Observe that opportunities for further performance enhancements exist with the increase of \( M_{qos} \). Fig. 7(a) and (b) present the BQS_I/O and DOR_I/O throughputs, which constitute the aggregate I/O throughputs in Fig. 6. As \( M_{qos} \) increases with \( M_{dmo} = 20 \), the resulting I/O throughput of each reservation queue becomes dominated by the DOR_I/O, not the BQS_I/O. Note that the BQS_I/O consistently preserves the given QoS feature at any level of the I/O workload. However, the result-

Fig. 6. Variations of I/O throughputs as a function of \( M_{qos} \) under fixed I/O workload levels, \(|I/O| = 11\) and \(|I/O| = 61\), where \( M_{dmo} = 20 \).
Fig. 7. Variations of BQS\_IO and DOR\_IO throughput as a function of $M_{\text{qos}}$ under fixed I/O workloads, $|I0| = 11$ and $|I0| = 61$, where $M_{\text{dmo}} = 20$.

Fig. 8. Variations of a given QoS feature as a function of $M_{\text{qos}}$ with $M_{\text{dmo}} = 20$.

ing I/O throughput of the BQS\_IO is diminished as the DOR\_IO becomes dominant.

Fig. 8 presents that $M_{\text{qos}}$ of 0.5 strongly enforces the given QoS feature to its DOR\_IO requests, so that it can provision disk bandwidth with a given ratio of 4:1 between $RQ_1$ and $RQ_2$. However, as $M_{\text{qos}}$ becomes larger, the desirable ratio deteriorates, i.e., the I/O throughput of $RQ_1$ is mainly affected by $RQ_2$, which uses more disk bandwidth with a relaxed QoS enforcement. As a result, the given QoS feature deteriorates by 1% with $M_{\text{qos}} = 0.5$, by 2% with $M_{\text{qos}} = 1$, and by 5% with $M_{\text{qos}} = 5$. It is difficult to devise a QoS metric to precisely measure the level of QoS satisfaction. In this paper, however, we simply define a ratio of reservation weights as our QoS metric. Using this QoS metric, we have to establish an acceptable range of the QoS satisfaction for a given QoS feature. We believe that satisfying a given QoS feature with 98–100% accuracy is reasonable enough. Thus, we can say that the QoS margin of $0.5 \leq M_{\text{best\_qos}} \leq 1$ falls into such a reasonable range.

3.2.3. Analyzing the degree of QoS guarantee and I/O throughput with the $M_{\text{best\_dmo}}$ and $M_{\text{best\_qos}}$

We have found $M_{\text{best\_dmo}} = 20$ and $0.5 \leq M_{\text{best\_qos}} \leq 1$ to maximize the disk I/O performance with negligible

Fig. 9. Variations of QoS guarantees and the maximum I/O throughput according to $M_{\text{qos}}$ with $M_{\text{dmo}} = 20$. $M_{\text{qos}} = (0, YFQ)$ and $M_{\text{dmo}} = (0.5, 1.0)$.
The two operational parameters, $M_{\text{base}}$ and $M_{\text{dmo}}$, are related to each of the two properties. Through extensive simulations, we have discovered two desirable controlling parameters for a given QoS feature. First, given a strong QoS enforcement with $M_{\text{dmo}} = 0.5$, we observed that the proposed algorithm with $M_{\text{dmo}} = 20$ achieved the best performance. Second, given the $M_{\text{dmo}} = 20$, we learned that a better disk I/O performance is achievable with less than 2% QoS deterioration by increasing $M_{\text{dmo}} \leq 1$. Obviously, a serious QoS deterioration is observed as $M_{\text{dmo}}$ increases continuously. Finally, performance comparisons with the baseline algorithm (YFQ) revealed that the proposed algorithm with $M_{\text{dmo}} = 20\%$ of the disk full seek time and $0.5 \leq M_{\text{dmo}} \leq 1$ improves the I/O throughputs by 11–19% with only 1–2% QoS deterioration.

In future work, we plan to devise an efficient scheme that will automatically find the two desirable controlling parameters in order to maximize underlying disk I/O throughputs with tolerable QoS deterioration for any given I/O workloads, including traced workloads.

Acknowledgements

This research was supported by the Daegu University Research Grant. The authors also would like to thank the Ministry of Education of Korea for its financial support through its BK21 program. This research was also supported in part by grant no. R01-2003-000-10739-01 from the Basic Research Program of the Korea Science and Engineering Foundation and by HY-SDR IT Research Center.

4. Conclusion and future work

This paper proposed a new proportional-share disk scheduling algorithm that considers both disk characteristics and QoS guarantees in an integrated manner. It consists of a BQS module and a DOR module. The former generates a base I/O sequence based on a typical fair-queueing scheme and the latter inserts extra I/O requests to the base I/O sequence if they meet the two given properties associated with the use of the available overhead in disk head movements and a limited relaxation of QoS enforcement.

The two operational parameters, $M_{\text{base}}$ and $M_{\text{dmo}}$, are related to each of the two properties. Through extensive simulations, we have discovered two desirable controlling parameters for a given QoS feature. First, given a strong QoS enforcement with $M_{\text{dmo}} = 0.5$, we observed that the proposed algorithm with $M_{\text{dmo}} = 20$ achieved the best performance. Second, given the $M_{\text{dmo}} = 20$, we learned that a better disk I/O performance is achievable with less than 2% QoS deterioration by increasing $M_{\text{dmo}} \leq 1$. Obviously, a serious QoS deterioration is observed as $M_{\text{dmo}}$ increases continuously. Finally, performance comparisons with the baseline algorithm (YFQ) revealed that the proposed algorithm with $M_{\text{dmo}} = 20\%$ of the disk full seek time and $0.5 \leq M_{\text{dmo}} \leq 1$ improves the I/O throughputs by 11–19% with only 1–2% QoS deterioration.

References


Young Jin Nam received a BE degree in 1992 from Kyungpook National University, Korea, an MS degree in 1994, and a PhD degree in 2004 from Pohang University of Science and Technology, Korea. He worked with Electronics and Telecommunications Research Institute, Korea from 1994 to 1998, where he engaged in the development of a microkernel-based operating system for a massively parallel computer. He was a visiting researcher at Novell Inc. (USA) in 1995, and at IBM Almaden Research Center in 2001. Since 2004, he has been a professor at the School of Computer and Information Technology in Daegu University, Korea. His current research interests include storage architectures with QoS guarantees, object-based storage, and embedded systems.

Chank Park earned a BE degree in 1983 from Seoul National University, Seoul, Korea, an MS degree in 1985, and a PhD degree in 1988, both from Korea Advanced Institute of Science and Technology, Korea. Since 1999, he has been working for Pohang University of Science and Technology, where he is currently a Full Professor in the department of computer science and engineering. He was a Visiting Scholar with Parallel Systems group in the IBM Thomas J. Watson Research Center in 1991, and a Visiting Professor with Storage Systems group in the IBM Almaden Research Center in 1999. He has served a number of international conferences as a member of Program Committee and he is a member of technical committee in the SIG-Storage Systems in Korea Information Processing Society. His research interests include storage systems, embedded systems, and pervasive computing.