Available online at www.sciencedirect.com
SCIENCE DIRECT®

Future Generation Computer Systems 22 (2006) 601–610

FGCS
FUTURE GENERATION COMPUTER SYSTEMS
www.elsevier.com/locate/fgcs

Design and evaluation of an efficient proportional-share disk scheduling algorithm

Young Jin Nam[a], Chanik Park[b,*]

[a] School of Computer and Information Technology, Daegu University, Jinryang, Gyeongsan, Kyungbuk 712-714, South Korea
[b] Division of Electrical and Computer Engineering, Pohang University of Science and Technology (POSTECH), San 31 Hyoja-dong, Pohang 790-784, South Korea

Available online 2 November 2005

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.

Keywords: Proportional share; Storage QoS; Disk resource; Disk head movement overhead

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

* Corresponding author.
E-mail address: cipark@postech.ac.kr (C. Park).

0167-739X/$ – see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2005.09.009

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 shortcoming 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 kth I/O workload arrives at the kth reservation queue. The kth reservation queue is denoted as $RQ_k = \{r_k^1, r_k^2, \ldots\}$, where r_k^i refers to the ith I/O request in the kth reservation queue. The notation of r_*^i represents the ith I/O request from a backlogged reservation queue. The notation of l_k^i represents the request size of r_k^i. RQ_k requires a different QoS requirement denoted by ϕ_k, i.e., a different amount of disk bandwidth. An actual bandwidth alloted to the kth I/O workload at time t is $\phi_k/\sum_{i\in B(t)} \phi_i W$, 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 S_k and F_k, 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_{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_{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 ϕ_k is allocated.

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

The BQS module generates S_{base} from I/O requests in RQ as 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 r_s^1 of RQ_s. Second, it advances the global virtual time, such that $v(t) = S_s$. Third, it updates $S_s = F_s$ and

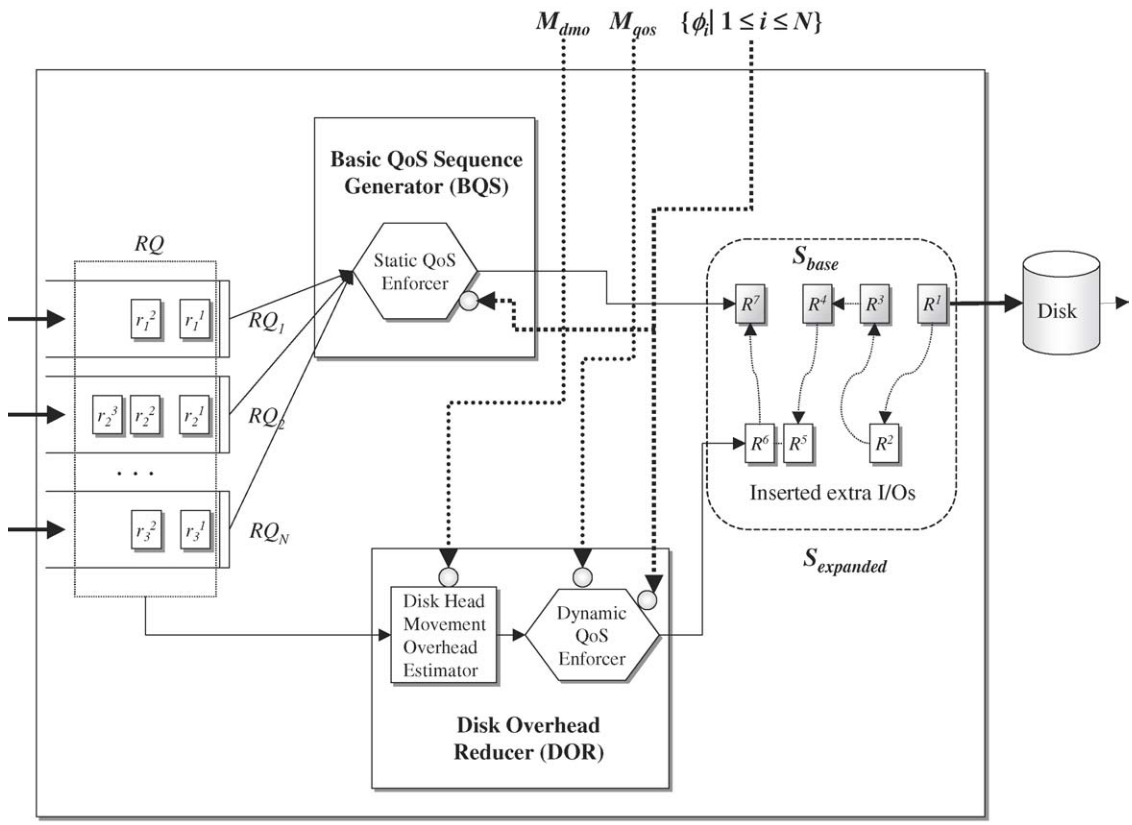

Fig. 1. Architecture of the proposed algorithm.

$F_s = S_s + \frac{l_s^2}{\phi_s}$ if RQ_s is still backlogged, where l_s^2 is the size of the next I/O request of r_s^1. Otherwise, S_s and F_s remain unchanged. When a new request r_i^k arrives at the empty RQ_i, then $S_i = \max\{v(t), F_i\}$ and $F_i = S_i + \frac{l_i^k}{\phi_i}$. 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 IO 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. Finally, denote with BQS_IO a group of I/O requests, which have passed through 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: $S_{\text{base}} = \emptyset$
Result: S_{base}
$N \leftarrow$ the length of the BQS sequence;
begin
  while $|S_{\text{base}}| \neq N$ **and** $\forall RQ_k \neq \emptyset$ **do**
    remove r_k^1 from RQ_k of $F_k = \min_i\{F_i\}$;
    $v(t) = S_k$; $S_k = F_k$; $F_k = S_k + \frac{l_k^2}{\phi_k}$ if $RQ_k \neq \emptyset$; $S_{\text{base}} = S_{\text{base}} \cup r_k^1$;
  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 S_{base} by considering the two properties associated with the available disk head movement overhead and a given level of QoS guarantee. Denote the expanded I/O sequence with S_{expanded}. The two properties depend largely on the two controlling parameters, M_{dmo} and M_{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_{interim} by estimating the disk overhead time that exists in the sequence. The notation of S_{interim} represents an intermediate I/O sequence in transit from S_{base} to S_{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 a rotational delay, 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 value denoted by M_{dmo} in order to reduce this type of estimation error. The value of M_{dmo} will be added to the estimated disk head movement overhead time between two I/O requests in S_{interim}. Note that the odds increase that an extra I/O request can be squeezed into the sequence, as M_{dmo} becomes larger. A proper value of M_{dmo} will be empirically obtained in the subsequent section. Property 1 clarifies the meaning of the marginal overhead time M_{dmo}. To begin, denote with $\{R^1, R^2, \ldots, R^M\}$ a set of I/O requests in the current S_{interim}, where M is the number of I/O requests in the current S_{interim}. $T_{\text{ov}}(R^i, R^{i+1})$ means the disk overhead time from request R^i to R^{i+1}.

Property 1. *The request of r_*^k is insertable between R^i and R^{i+1} if it satisfies the following inequality*: $T_{\text{ov}}(R^i, r_*^k) + T_{\text{ov}}(r_*^k, R^{i+1}) \leq T_{\text{ov}}(R^i, R^{i+1}) + M_{\text{dmo}}$.

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

2.2.2.2. DQE module: preserving a QoS feature with DOR_IO 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_{base}. It maintains an additional virtual time for each RQ_k and the maximum displacement among such additional virtual times. Denote with f_k the additional virtual finish time of RQ_k. Initially, $f_k = 0$. If an I/O request arrives at an empty RQ_k, $f_k = \max\{f_k, \min_{\{j, j \neq k\}}\{f_j\}\}$, for $RQ_j \neq \emptyset$. Denote with δ_f the maximum displacement among the current additional virtual times of the backlogged reservation queues. It is defined as $\delta_f = \max\{|f_i - f_j|\}$ 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_{qos}.

Property 2. *The request of r_k^i preserves a given QoS feature of M_{qos} if it meets the following inequality*: $\delta_f \leq M_{\text{qos}}$ *when* $f_k = f_k + \frac{l_k^i}{\phi_k}$.

$M_{\text{qos}} = \beta$ implies that a set of virtual times will advance with an amount of QoS deterioration, such that the resulting I/O throughput of each RQ_k will deviate from a given QoS feature on the average by $\frac{\phi_k}{l_k^{\text{avg}}}\beta$ I/O requests, where l_k^{avg} is an average block size of an I/O request from RQ_k. According to Property 2, no I/O requests can pass the DOR module with $M_{\text{qos}} = 0$, because 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: $S_{\text{interim}} = S_{\text{base}}$
Result: $S_{\text{expended}} = S_{\text{interim}} +$ DOR_IO requests
begin
  //$S_{\text{interim}} = \{R^1, R^2, \ldots R^M\}$;
  $R^{\text{cur}} = R^{\text{head}} = R^1$; $R^{\text{tail}} = R^N$; $R^{\text{next}} = R^{\text{cur}+1}$;
  while $R^{\text{cur}} \neq R^{\text{tail}}$ **do**
    for *each* r_k^i *in* $RQ_k \in RQ$ **do**
      if r_k^i meets Property 1 **and** Property 2 **then**
        insert r_k^i between R^{cur} & R^{next} in S_{interim}; //$R^{j+1} = R^j$ for $j = M$ to *next*;
        $f_k = f_k + \frac{l_k^i}{\phi_k}$; $R^{\text{next}} = r_k^i$;
        break;
      end
    end
    $R^{\text{cur}} = R^{\text{cur}+1}$ in S_{interim}; $R^{\text{next}} = R^{\text{cur}+1}$;
  end
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 DNES 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 RQ_1 and RQ_2. 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 $M_{\text{dmo}}^{\text{best}}$ 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 $M_{\text{dmo}}^{\text{best}}$, we attempt to obtain $M_{\text{qos}}^{\text{best}}$ which provides a better disk I/O performance with acceptable QoS deterioration. Finally, the proposed algorithm with

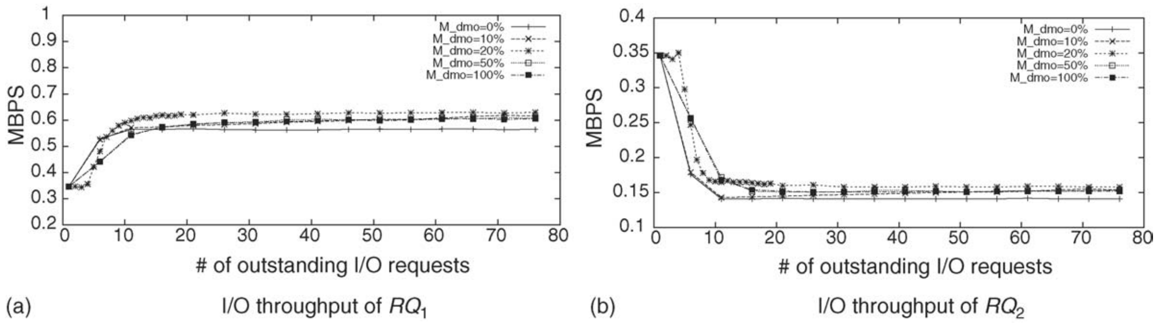

Fig. 2. Variations of I/O throughputs of RQ_1 and RQ_2 as a function of an I/O workload level by varying the number of outstanding I/O requests ($|IO|$) under different values of M_{dmo}, where $M_{\text{qos}} = 0.5$.

$M_{\text{dmo}}^{\text{best}}$ and $M_{\text{qos}}^{\text{best}}$ will be compared with the YFQ algorithm in terms of the disk I/O performance and QoS guarantee.

3.2.1. Finding $M_{\text{dmo}}^{\text{best}}$ 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 $M_{\text{dmo}} = 0$. Fig. 2 shows the I/O throughput of the proposed algorithm with different M_{dmo} under various I/O workload levels where $M_{\text{qos}} = 0.5$. We expect that the given QoS feature will be preserved throughout all I/O workload levels because of $M_{\text{qos}} = 0.5$, which corresponds to a strong QoS enforcement. By definition in Property 2, $M_{\text{qos}} = 0.5$ 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 $M_{\text{dmo}} = 20$.

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 $M_{\text{dmo}} = 20$. 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

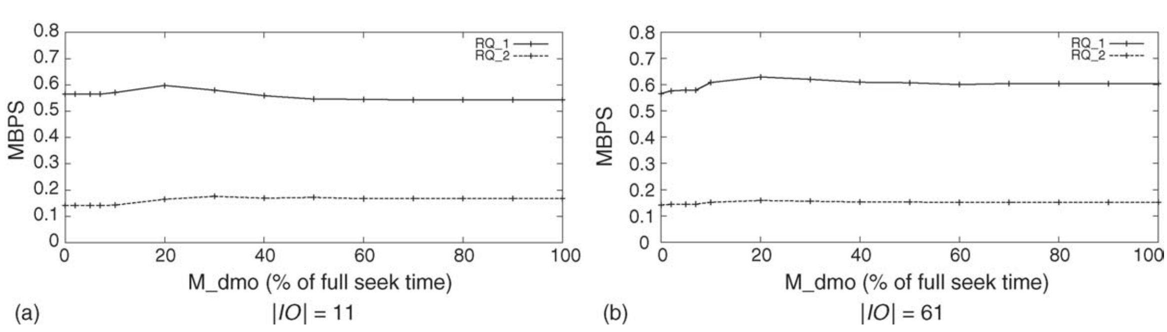

Fig. 3. Variations of I/O throughputs as a function of M_{dmo} when $|IO| = 11$ and $|IO| = 61$, where $M_{\text{qos}} = 0.5$.

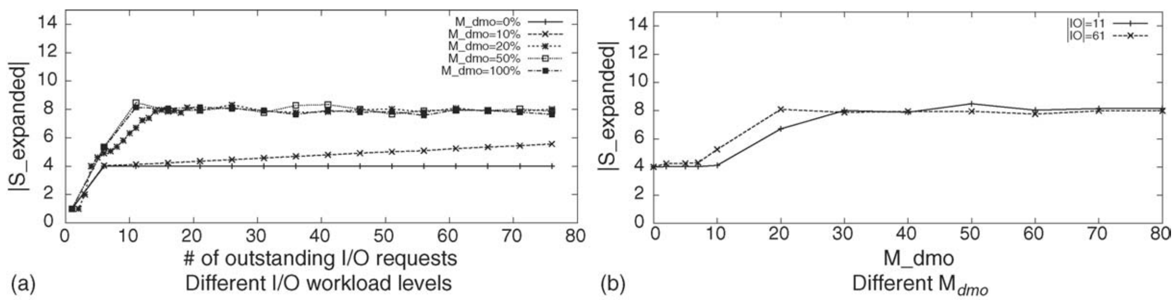

Fig. 4. Variations of $|S_{\text{expanded}}|$ with different I/O workload levels and M_{dmo} values with $M_{\text{qos}} = 0.5$.

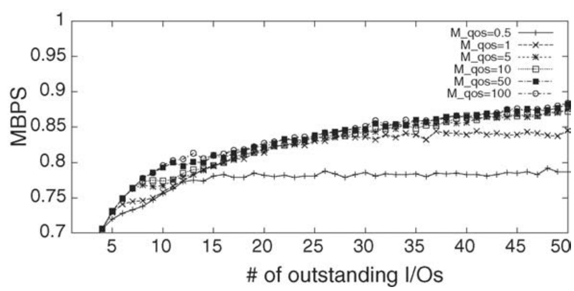

Fig. 5. Variations of I/O throughputs as a function of an I/O workload level under different M_{qos} values.

evaluation with the observation that $M_{\text{dmo}}^{\text{best}}$ 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 $M_{\text{qos}}^{\text{best}}$ 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 $M_{\text{qos}} \geq 5$ is higher than that of $M_{\text{qos}} = 0.5$ 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, $|IO| = 11$ and $|IO| = 61$. Fig. 6 shows that the I/O throughput of the RQ_1 decreases, whereas the I/O throughput of its competing I/O workload RQ_2 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_IO and DOR_IO throughputs, which constitute the aggregate I/O throughputs in Fig. 6. As M_{qos} increases with $M_{\text{dmo}} = 20$, the resulting I/O throughput of each reservation queue becomes dominated by the DOR_IO, not the BQS_IO. Note that the BQS_IO consistently preserves the given QoS feature at any level of the I/O workload. However, the result-

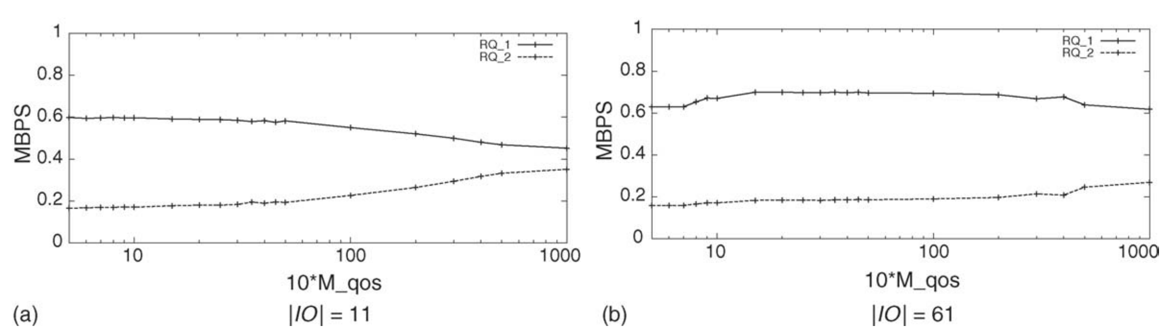

Fig. 6. Variations of I/O throughputs as a function of M_{qos} under fixed I/O workload levels, $|IO| = 11$ and $|IO| = 61$, where $M_{\text{dmo}} = 20$.

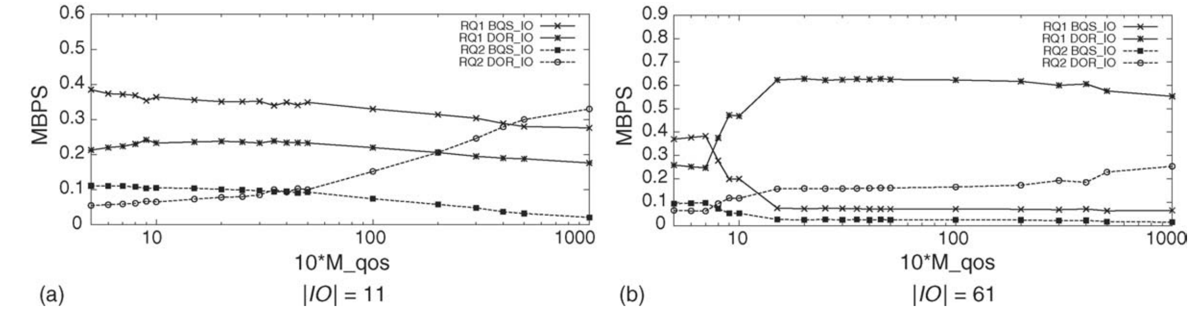

Fig. 7. Variations of BQS_IO and DOR_IO throughput as a function of M_{qos} under fixed I/O workloads, $|IO| = 11$ and $|IO| = 61$, where $M_{\text{dmo}} = 20$.

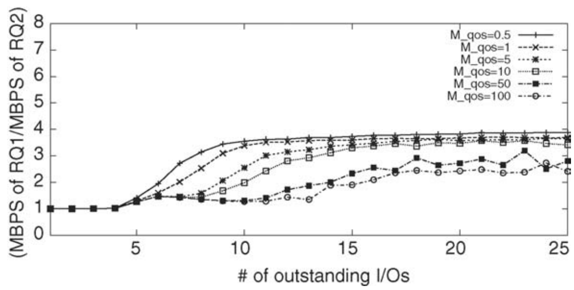

Fig. 8. Variations of a given QoS feature as a function of M_{qos} with $M_{\text{dmo}} = 20$.

ing I/O throughput of the BQS_IO is diminished as the DOR_IO becomes dominant.

Fig. 8 presents that M_{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_{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 $0.5 \le M_{\text{qos}}^{\text{best}} \le 1$ falls into such a reasonable range.

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

We have found $M_{\text{dmo}}^{\text{best}} = 20$ and $0.5 \le M_{\text{qos}}^{\text{best}} \le 1$ maximize the disk I/O performance with negligible

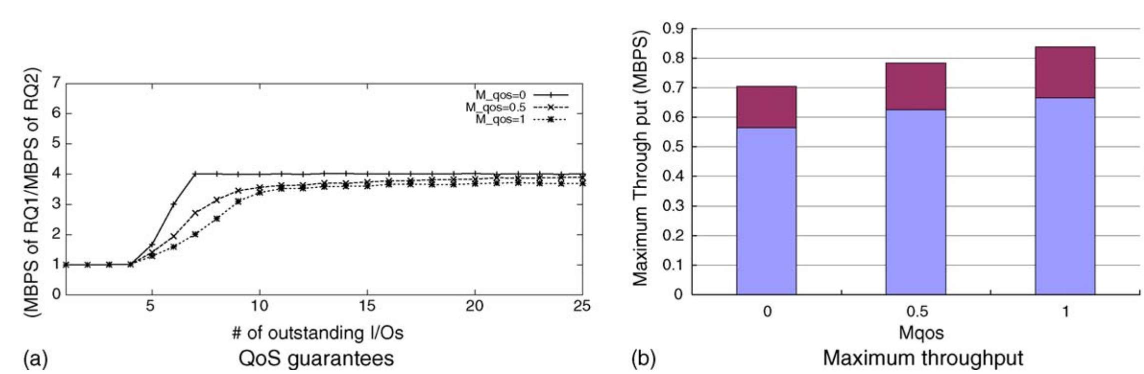

Fig. 9. Variation of QoS guarantees and the maximum I/O throughput according to M_{qos} with $M_{\text{dmo}} = 20$: $M_{\text{qos}} = 0$(YFQ) and $M_{\text{qos}}^{\text{best}} = \{0.5, 1.0\}$.

QoS deterioration. At this point, we will compare the proposed algorithm with the baseline algorithm (YFQ) assuring QoS guarantee in terms of the degree of QoS enforcement and I/O throughput. Fig. 9(a) shows the degree of QoS enforcement for the given QoS ratio of 80:20 as a function of the number of outstanding I/Os. As expected, the proposed algorithm with $M_{\text{qos}} = 0.5$ and $M_{\text{qos}} = 1$ maintained the given QoS ratio with more than 98% accuracy. While the baseline algorithm having $M_{\text{qos}} = 0$ provides the given 4:1 ratio, the $M_{\text{qos}} = 0.5$ and $M_{\text{qos}} = 1$ end up with 3.99:1 and 3.96:1, respectively. Fig. 9(b) compares the maximum I/O throughput of the proposed algorithm with the $M_{\text{dmo}}^{\text{best}}$ and $M_{\text{qos}}^{\text{best}}$ with the case of $M_{\text{qos}} = 0$ (YFQ). Observe that the proposed algorithm with $M_{\text{qos}} = 0.5$ and $M_{\text{qos}} = 1.0$ improves the I/O throughputs by 11–19% with only 1–2% QoS deterioration.

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-queuing 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_{dmo} and M_{qos} 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 $M_{\text{qos}} = 0.5$, 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 $M_{\text{qos}} \leq 1$. Obviously, a serious QoS deterioration is observed as M_{qos} 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 $0.5 \leq M_{\text{qos}} \leq 1$ improved I/O throughput 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. R01-2003-000-10739-0 from the Basic Research Program of the Korea Science and Engineering Foundation and by HY-SDR IT Research Center.

References

[1] C. Lumb, A. Merchant, G. Alvarez, Facade: Virtual storage devices with performance guarantees, Proceedings of Conference on File and Storage Technologies, 2003.

[2] J. Bruno, J. Brustoloni, E. Gabber, B. Ozden, A. Silberschatz, Disk scheduling with quality of service guarantees, Proceedings of the IEEE International Conference on Multimedia Computing and Systems, 1999.

[3] P. Shenoy, H. Vin, Cello: A disk scheduling framework for next-generation operating systems, Proceedings of ACM SIGMETRICS, 1998.

[4] A. Parekh, R. Gallager, A generalized processor sharing approach to flow control in integrated services networks: The single-node case, IEEE/ACM Trans. Network. 1 (3) (1993) 344–357.

[5] P. Goyal, H. Vin, H. Cheng, Start-time fair queueing: A scheduling algorithm for integrated services packet switching networks, IEEE Trans. Network. 5 (5) (1997) 690–704.

[6] D. Jacobson, J. Wilkes, Disk scheduling algorithms based on rotational position, Technical Report HPL-CSP-91–7rev1, Hewlett Packard, 1991.

[7] B. Worthington, G. Ganger, Y. Patt, Scheduling algorithms for modern disk drives, Proceedings of SIGMETRICS Conference, 1994, pp. 241–251.

[8] J. Schindler, G. Ganger, Automated disk drive characterization, Tech. Re CMU-CS-99–176, CMU (1999).

[9] C. Lumb, J. Schindler, G. Ganger, Freeblock scheduling outside of disk firmware, Proceedings of Conference on File and Storage Technologies, USENIX, 2002.

[10] L. Huang, T. Chiueh, Implementation of a rotational-latency-sensitive disk scheduler, Technical Report.

[11] G. Ganger, B. Worthington, Y. Patt, The DiskSim Simulation Environment Version 2. 0 Reference Manual, CMU, December 1999.

[12] Ibm dnes 309170w seek curve., http://www.pdl.cmu.edu/DiskSim/diskspecs.html.

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.

Chanik 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 1989, 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.