A heuristic for selecting the cycle length for FAIRIO

S. Arunagiri[1], Y. Kwok[1], P. J. Teller[1], R. A. Portillo[1], and S. Seelam[2]
[1]*Department of Computer Science, The University of Texas at El Paso, El Paso, Texas 79968, USA*
{sarunagiri@, ykwok2@miners., pteller@, raportil@miners.}utep.edu
[2]*IBM T.J. Watson Research Center, USA*
sseelam@us.ibm.com

Abstract

FAIRIO is a cycle-based I/O scheduling algorithm that provides differentiated service to workloads concurrently accessing a consolidated RAID storage system. FAIRIO enforces proportional sharing of I/O service through fair scheduling of disk time. During each cycle of the algorithm, I/O requests are scheduled according to workload weights and disk-time utilization history. There are several parameters in the FAIRIO scheduler that can affect its behavior. One parameter in particular, the length of a scheduling cycle, can affect the scheduler's ability to react to and compensate for changes in workload access characteristics. Unfortunately, there is no cycle length that is optimal for all workloads and, thus, it must be chosen according to the workload environment. This technical report describes a heuristic that can be used to choose a favorable cycle length that promotes performance differentiation given a priori knowledge of workload access behavior. The heuristic is validated using simulations driven by several real and synthetic workload scenarios that represent a broad range of request types, sizes, and access characteristics. We show that when workload weights properly map to workload requirements, cycle lengths deduced by our heuristic promote differentiated disk-time sharing within 3.9% of perfect proportionality to workload weights.

I. Introduction

FAIRIO is designed for RAID storage systems such as the one schematically depicted in Figure 1, and is assumed to be implemented at the I/O driver in order to provide proportional sharing of the RAID's bottleneck device, the disks. It is a cycle-based algorithm that provides to each concurrently active workload, one expected to generate I/O requests, a proportional share of the total available disk time, i.e., an allocation that is relatively proportional to the workload's weight (a w_i/W share of total disk time, where w_i is $workload_i$'s weight and W is the sum of the weights of all workloads). This is accomplished over a sequence of scheduling cycles via coordinated disk-time share allocations to active workloads.

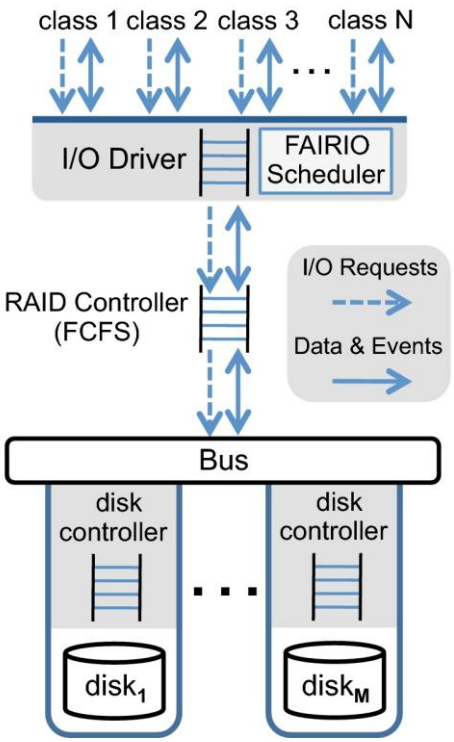

Fig. 1. RAID Storage System: FAIRIO is assumed to operate at the I/O driver level of the RAID storage hierarchy.

At the beginning of each scheduling cycle, FAIRIO calculates the amount of disk time to be allocated to each active class, $Class_i$. However, variability in request characteristics, request streams, and storage state across cycles may cause a difference between the amount of disk time allocated to a class and the amount that is actually used to service its requests. Therefore, in order to maintain differentiated service, the allocation of disk time in a cycle factors-in not only workload weights but also observed deviations in the history of disk-time utilization during a set of previous cycles, called a *window of observation*. Specifically, this moving window of time, where cycles are dynamically added and discarded from the window, approximates a user-defined number of full cycles. At the end of a cycle, the disk time utilized by each class during the cycle is recorded into the window of observation. This utilization information is stored in the window as context-dependent values that ignore service deviations that did not degrade the service of other classes.

Since FAIRIO adjusts disk-time shares at the beginning of every cycle, the length of a scheduling cycle has a significant impact on the performance of FAIRIO, and, thus, is an important FAIRIO parameter. The length of a scheduling cycle involves tradeoffs between FAIRIO responsiveness, disk-time availability, and scheduling overhead. For example, short cycle lengths increase the frequency of disk-time share adjustments based on observed utilization and, thus, enable FAIRIO to more quickly react to changes in workload access characteristics. In contrast, long cycle lengths result in more available disk-time share per cycle to meet minimum allocation thresholds for workloads, and incur less control and computation overheads in the I/O system. Thus, the length of a scheduling cycle should be short enough to allow disk-time share adjustments at the rate with which workload access varies, yet not so short as to incur unnecessary overhead and allocation inflexibility.

This technical report demonstrates how to select a cycle length for FAIRIO that offers a favorable balance between the above-mentioned tradeoffs. This report first illustrates in Section II the impact of various

cycle lengths on FAIRIO performance. Next, Section III discusses a heuristic for selecting a suitable cycle length, i.e., a cycle length that leads to small errors in terms of disk-time differentiation. Finally, the effectiveness of the heuristic is demonstrated in Section IV. We have included necessary details of FAIRIO required for understanding the heuristics into this technical report. Complete details of FAIRIO can be found in [1].

II. The Performance Impact of Cycle Length

To demonstrate the performance impact of cycle length, we evaluated 14 different values of this parameter for two sets of experiments, Illustration 1 and Illustration 2. Each set of experiments was driven by an artificial I/O trace consisting of two request classes with weights 1 and 2, and with 10,000 requests each. The request inter-arrival time of each class was 10 ms, which led to an average overall request inter-arrival time of 5 ms. For all experiments, we selected a window of observation 20 times the size of a full cycle length, i.e., we selected a value of 20 for the FAIRIO parameter *r*. FAIRIO defines two thresholds as throughput-optimization parameters. The first threshold, $Class_{threshold}$, specifies a minimum number of requests of a class that can be dispatched in a cycle, while the second threshold, $StorageLoad_{threshold}$, specifies the minimum number of in-flight requests for the storage to not be considered under-loaded. Once the storage becomes under-loaded, FAIRIO attempts to increase the number of in-flight requests to reach the threshold. Though the first throughput optimization can potentially improve performance, in the absence of a starvation-avoidance scheme we chose to disable it, by setting $Class_{threshold}$ to 1, i.e., in order to prevent requests from being upheld at the I/O driver indefinitely. In order to ensure high bandwidth utilization, $StorageLoad_{threshold}$ is set to match the request capacity of the disk controller queue, i.e., 16.

To determine how well a particular cycle length promotes performance differentiation, we measured the error between a class' disk-time utilization ($ActualRatio_i$) and its rightful share of disk time ($IdealRatio_i$). We compute this error, $Error_i$, $i \in \{1 \ldots n\}$, where n is the number of active classes, in terms of DT_i, the disk-time utilized by $Class_i$ over time, and w_i, the weight of $Class_i$, i.e.,

$$Error_i = (ActualRatio_i - IdealRatio_i) * 100\%, \text{ where}$$
$$IdealRatio_i = w_i / \sum(w_i), \text{ and}$$
$$ActualRatio_i = DT_i / \sum(DT_i).$$

Tables I and II show the results for each set of experiments, respectively. These tables present three types of information: (1) the cycle length used in the experiment, (2) the error in performance differentiation for each class ($Error_i$), and (3) the aggregate throughput of the two classes combined, averaged over time. We now describe each set of experiments in detail and present the associated results.

Illustration 1: For this set of experiments, both classes accessed the same storage utility with 128-block requests; $Class_1$ and $Class_2$ issued random and sequential read access streams, respectively. As shown in Table I, the choice of cycle length did not significantly affect the delivered aggregate throughput; it ranged from 7.26 to 7.32 MB/s. In contrast, in terms of disk-time differentiation, the errors mostly decreased as the cycle length increased, reaching a minimum of 0.18% for a cycle length of 2,000 ms, before beginning to rise for the last two cycle lengths, i.e., 2,500 and 3,000.

Table I: The effect of cycle length for Illustration 1
where both request classes have fixed access characteristics.

Cycle length (ms)	$Error_1$ (%)	$Error_2$ (%)	Aggregate throughput (MB/s)
100	12.20	-12.20	7.28
200	13.69	-13.69	7.26
300	8.90	-8.90	7.28
400	11.49	-11.49	7.26

500	6.48	-6.48	7.27
600	9.00	-9.00	7.27
700	6.20	-6.20	7.28
800	2.16	-2.16	7.29
900	9.02	-9.02	7.27
1,000	1.33	-1.33	7.29
1,500	1.17	-1.17	7.29
2,000	-0.18	0.18	7.31
2,500	0.36	-0.36	7.30
3,000	0.50	-0.50	7.32

Illustration 2: In this set of experiments, the request trace of $Class_2$ remained the same as in Illustration 1, while $Class_1$ issued read requests at the same rate as in Illustration 1 but with monotonically increasing sizes and distances between consecutive requests. Below, Equations (i) and (ii) describe how the request sizes and address distances between consecutive requests increased for $Class_1$, respectively.

For j=0, 1, 2, ...

$$\text{size of request}_j = 16 + 10 * \text{Floor}(j / 100) \quad \text{(i)}$$

$$\text{jump distance between request}_j \text{ and request}_{(j+1)} = \text{request size} + 10 * (1 + \text{Floor}(j / 100)) \quad \text{(ii)}$$

For example, the first size of the first hundred $Class_1$ requests was 16 and the request address distance between consecutive requests was 26.

As shown in Table II, the choice of cycle length had an observable effect on the delivered aggregate throughput of the experiments of Illustration 2; it ranged from 8.07 to 9.32 MB/s, which is a difference of 15.49%. In terms of disk-time differentiation, the errors were above 11% for cycle lengths at or under 800 ms; and below 6% for the rest of the experiments; cycle length 900 ms yielded the smallest error with 1.40%.

Table II: The effect of cycle length for Illustration 2
where $Class_1$ and $Class_2$ have varying and fixed access characteristics, respectively.

Cycle length (ms)	**$Error_1$ (%)**	**$Error_2$ (%)**	**Aggregate throughput (MB/s)**
100	14.49	-14.49	8.94
200	15.54	-15.54	9.24
300	16.08	-16.08	9.32
400	11.27	-11.27	8.85
500	11.79	-11.79	8.87
600	17.76	-17.76	8.90
700	15.29	-15.29	8.88
800	14.49	-14.49	8.94
900	-1.40	1.40	8.07
1,000	5.14	-5.14	8.19
1,500	2.41	-2.41	8.12
2,000	4.57	-4.57	8.21
2,500	4.81	-4.81	8.23
3,000	4.44	-4.44	8.22

The results in Illustration 1 and Illustration 2 show that, in general, there is a downward trend in the performance differentiation error as the cycle length is increased. However, this trend eventually hits a minimum and after this point the error begins to increase. In the case of Illustration 1, where both classes have non-varying access characteristics, the minimum error is reached with a relatively large cycle length of 2,000 ms. On the other hand, in the case of Illustration 2, where one of the classes has a highly varying access behavior, the minimum error is reached sooner, i.e., with a relatively smaller cycle length of 900 ms. These results support our argument that larger cycle lengths are suitable for workloads with non-varying access characteristics, while shorter cycle lengths are suitable for workloads with more varying access behaviors.

III. A Heuristic for Selecting FAIRIO Cycle Length

As shown by the two sets of experiments in Section II, a cycle length that results in low performance differentiation error for one workload may result in a relatively large error for another. Accordingly, we explored a heuristic for choosing a suitable cycle length given a workload of classes sharing a storage system. Below, we present the heuristic and then illustrate its effectiveness. Although it is possible to estimate cycle lengths dynamically during runtime, we chose to use static values in our experiments as an initial step to evaluate this heuristic.

Our heuristic has the following input parameters for each class, $Class_i$:

- $ClassType_i$: a classification of $Class_i$ based on the stability of its access characteristics. These characteristics include request arrival rate, access pattern, request size, and request type (read or write). If any of these characteristics is unstable, we classify $Class_i$ as $Type_1$. Otherwise, if these characteristics are all stable, $Class_i$ is considered to be of $Type_2$.
- $ServiceRequest_i$: an estimation of average service time for a $Class_i$ request. In most cases, the average service time per request changes only negligibly with FAIRIO, relative to what it is when FAIRIO is disabled. Therefore, when using our heuristic for the experiments discussed in Section IV, we used the average service time that we observed with FAIRIO disabled.
- $NumRequests_i$: the number of requests that $Class_i$ is expected to dispatch during a scheduling cycle, which depends on $ClassType_i$. Below, we discuss how to choose the value of $NumRequests_i$ based on $ClassType_i$ to achieve the goal of low expected error of differentiated service; note that this value must be greater than or equal to $Class_{threshold}$, which is a FAIRIO parameter related to performance optimization [1].

Given these parameters, we can derive the values for (1) the minimum service allocation for each class in a cycle ($ServiceShare_i$), then (2) the minimum amount of total service in a cycle ($Service_{cycle}$), and, finally, (3) the minimum wall-clock time in a full scheduling cycle ($Cycle_{length}$). These terms are summarized and placed in the context of a FAIRIO environment in Table III. We now show the derivation of these values to arrive at a favorable value for $Cycle_{length}$.

Table III: FAIRIO terminology relevant to the $Cycle_{length}$ heuristic (disk-time is the service metric).

Scheduler Environment	
$Cycle_{length}$	Amount of wall-clock time in a full scheduling cycle
$Service_{cycle}$	Amount of service in a full scheduling cycle
Service Allocation	
$ServiceShare_i$	Amount of service allocated to $Class_i$ in a cycle
Storage System	

NumDisks	Number of disks in the RAID storage system

First we must determine $ServiceShare_i$. As a lower bound, $ServiceShare_i$ must be large enough to accommodate $NumRequests_i$ requests, i.e.,

$$ServiceShare_i >= (ServiceRequest_i * NumRequests_i). \quad (1)$$

Independently of this lower bound, during workload execution FAIRIO computes $ServiceShare_i$ in two ways depending on whether or not $Class_i$'s service utilization has deviated from its rightful share. Specifically, if a deviation from its rightful share occurs, $ServiceShare_i$ is computed *relative to the amount of deviation*, a value that we cannot easily predict. Otherwise, FAIRIO computes it *relative to the static weights of the classes*, which we know *a priori*. Since we can only predict the computed value of $ServiceShare_i$ in the scenario where no class deviates from its rightful share, we only consider this case to estimate the initial value of cycle length. Thus, we assume that $ServiceShare_i$ is computed as follows:

$$ServiceShare_i = (w_i / W) * Service_{cycle}, \text{ where } W = \sum(w_i).$$

Thus,

$$Service_{cycle} = (W / w_i) * ServiceShare_i. \quad (2)$$

If we combine the lower bound on $ServiceShare_i$ in Equation (1) with Equation (2), $Class_i$ imposes the following constraint on $Service_{cycle}$:

$$Service_{cycle} >= (W / w_i) * ServiceRequest_i * NumRequests_i.$$

To satisfy all classes, we pick the maximum of

$$(W / w_i) * ServiceRequest_i * NumRequests_i$$

over all i as the value of $Service_{cycle}$. Finally, given the following formula for calculating $Service_{cycle}$ for a RAID device [1],

$$Service_{cycle} = (Cycle_{length} * NumDisks),$$

we can derive the lower bound for $Cycle_{length}$ as:

$$Cycle_{length} > = \max((W / w_i) * ServiceRequest_i * NumRequests_i) / NumDisks. \quad (3)$$

Given Equation (3), our heuristic for selecting a suitable cycle length, which is guided by class types, is:

- If $ClassType_i = Type_1$, select a small $NumRequests_i$.
- If $ClassType_i = Type_2$, select a larger $NumRequests_i$.

In conjunction with Equation (3), the above guidance suggests a lower bound of cycle length based on the properties of each class. For the sake of convenience, note that in our experiments we rounded up the cycle lengths suggested by Equation (3) to the nearest hundredth millisecond.

According to the two sets of experiments in Section II, FAIRIO achieves performance differentiation with small errors when $NumRequests_i$ equals 17 and 85, for $ClassType_i$ of $Type_1$ and $Type_2$, respectively. Since Illustration 1 and Illustration 2 represent workloads of the two extremes in terms of the stability of access characteristics, we can use them as references when deducing the suitable cycle length for real and synthetic benchmarks. Section IV illustrates this.

IV. Performance of the Heuristic for Real and Synthetic Benchmarks

FAIRIO's performance is evaluated using simulations conducted on an enhanced version of DiskSim [2]. We enhanced DiskSim 3.0 to support request classes and implement the FAIRIO algorithm in the I/O driver. The simulated I/O system is similar to the one depicted in Figure 1. As shown, I/O requests from request classes are input to a unified arrival-ordered list of pending requests (tagged with Class IDs) that

is used by FAIRIO in determining dispatch order. Each request class is of a specific length and is assigned a particular weight. The I/O driver feeds requests to the RAID controller, which has an FCFS-scheduled queue of "infinite" length – the queue can hold more than the total number of scheduled requests of any experiment. The RAID controller and disks (each modeled as a closed I/O subsystem) are connected via an I/O bus that is used to pass requests, data, and events. An 8-disk RAID-0 configuration is used with IBM model 18es disk drives, each with an SSTF-scheduled request queue that can hold up to 16 requests and a cache of minimal size. We used a RAID-0 configuration because of DiskSim 3.0 limitations.

DiskSim is a trace-driven simulator, thus, suitable I/O traces are needed to drive experiments that can be used to evaluate FAIRIO performance. We produced six such traces of 4 classes each, three real and three synthetic, using four I/O benchmarks. The four I/O benchmarks used to produce the traces are: (1) varmail executed using Filebench as a workload personality, a non-scientific benchmark; (2) NAS BTIO and MADbench2, two scientific benchmarks, and (3) IOR, a synthetic benchmark from Lawrence Livermore National Laboratory, parameterized in three different ways to generate a trace with a random access pattern (for IOR1 experiments); a trace with a sequential access pattern (for IOR2 experiments); and a trace with a mixed access pattern in which Classes 1 and 3 have random access patterns and Classes 2 and 4 have sequential access patterns (for IOR3 experiments). Table IV summarizes the workload represented by each class of each I/O trace used in this study. As in Section II, we selected the FAIRIO parameters r, $Class_{threshold}$, and $StorageLoad_{threshold}$ to 20, 1, and 16, respectively, for all experiments.

Table IV: Experimental Details

Benchmark	Description of the trace for each class
varmail	10-minute execution of Filebench with 12 concurrent threads; emulates the storage accesses of a /var/mail NFS mail server through a specific loadable workload personality
BTIO	BTIO with problem size Class A and four MPI tasks accessing the same output file; configured to issue a write every three (instead of the default five) time steps
MADbench2	MADbench2 with 4, 000 x 4, 000 pixel problem size, four MPI tasks, all of which can issue read and write requests concurrently, and 16 component matrices; gang scheduling disabled and 32-byte file block size
IOR1, IOR2, and IOR3	IOR with 12 MPI tasks; writes 64KB requests to a 3GB file; configured to use the POSIX I/O API; performs fsync after each write, which transfers modified data from the system buffer to storage in order to minimize the effect of I/O buffering, and upon each write close

In order to decide a suitable value for the parameter $NumRequests_i$ used in our heuristic for each class, we first need to classify the class type. We generated the artificial traces used in Illustration 1 and Illustration 2 using pre-defined I/O access characteristics. Therefore, there is little ambiguity when deciding their class types. However, traces used in our Section IV experiments are based on real and synthetic benchmarks, which are not artificial. Thus, the stability of their access characteristics are likely between those of the two artificial traces (representing two extremes) in Illustration 1 and Illustration 2. In order to decide suitable values for $NumRequests_i$ for these benchmarks, we used the access characteristics of the artificial traces as references when we evaluated the stability of our benchmark traces. For workloads with fairly stable access characteristics, we chose a value for $NumRequests_i$ close to 85, the value that led to the most suitable cycle length in Illustration 1. In contrast, for workloads that demonstrate unstable access characteristics, we chose a small value for $NumRequests_i$ that is close to 17, which led to the most suitable

cycle length in Illustration 2. However, note that if the access characteristics of a workload are more unstable than those of Illustration 2, a suitable value for $NumRequests_i$ may actually be smaller than 17.

We now show the performance of our heuristic for the workload experiments described above, i.e., six experiments, each running four differently weighted classes of the same workload, BTIO, MADbench2, varmail, IOR1, IOR2, or IOR3. For each experiment, Table V shows the values of $ServiceRequest_i$, $NumRequests_i$, the cycle length suggested by the heuristic, and the error-bound among classes in terms of disk-time differentiation. Overall, Table V shows that the errors were bounded by 6.5% and that the varmail and IOR1 workloads experienced negligible errors.

The relatively large error for MADbench2 was due to the fact that the two classes with higher weights did not have request rates high enough to consume their disk-time allocations. In general, given a set of request classes, FAIRIO can provide differentiated disk-time sharing only under the condition that the request arrival rate (B/s), *Ai*, of each $Class_i$ is evenly distributed across cycles and satisfies the following inequality:

$$A_i \geq (w_i * T)/\sum(w_i), \text{ for all } i,$$

where w_i is the weight of $Class_i$ and T is the estimated throughput of the RAID system achievable by the given set of request classes or, as an approximation, the highest throughput achieved by any set of classes. Hypothetically, under these conditions FAIRIO could still enforce differentiation by not allowing the other two classes to supplement their allocations with surplus service. However, such strict enforcement of performance differentiation would hinder work conservation. Thus, in this scenario FAIRIO chooses to forgo perfect performance differentiation in exchange for the utilization of otherwise unused I/O service. Specifically, FAIRIO allowed the two classes with lower weights to consume extra disk time on top of their allocations. In short, the high errors for MADbench2 were due to classes not being able to take full advantage of their high weights rather than due to the selection of cycle length.

With respect to IOR2, we believe that the errors were because we over-evaluated the stability of their access characteristics. Specifically, the value of $NumRequests_i$ is possibly too large. To address the possible inaccuracy of our method for defining a value for $NumRequests_i$, we are currently developing a revised heuristic that measures workload access characteristics to quantify stability and, thus, more accurately estimate the value of $NumRequests_i$. We expect that this revised heuristic will be able to suggest a more suitable cycle length given the fact that it takes quantitative measurements as inputs.

Table V: Derived values and respective performance differentiation when our heuristic is applied to a set of real and synthetic benchmarks.

Workload	**ServiceRequest$_i$ (ms)**	**NumRequests$_i$**	**Cycle length based on heuristic (rounded up to nearest 100 ms)**	**Error bound (%)**
BTIO	73.42	39	2,200.00	1.62
MADbench2	73.57	56	3,100.00	6.47
varmail	69.47	19	1,000.00	0.86
IOR1	70.25	39	2,100.00	0.72
IOR2	74.01	64	3,600.00	3.81
IOR3	69.44	32	1,700.00	1.46

V. Conclusions and Future Work

In this technical report, we demonstrated how the choice of cycle length impacts the performance of FAIRIO in providing differentiated disk-time sharing. We further empirically demonstrated that favorable cycle lengths, which lead to relatively small errors in terms of disk-time differentiation, differ from one

workload to another. In doing so, we introduced a heuristic that can be used to deduce cycle lengths for FAIRIO based on workload access characteristics. Experiments using a set of widely used workloads show the performance of our heuristics. It is able to deduce favorable cycle lengths for the FAIRIO scheduler to achieve differentiated disk-time sharing within 3.9% of perfect proportionality to workload with weights properly mapped to workload requirements. As future work, we are revising the heuristic to quantify stabilities of workload access characteristics in order to more accurately estimate values of the parameter $NumRequests_i$, an important parameter in our heuristic.

References

[1] S. Arunagiri, Y. Kwok, P. J. Teller, R. A. Portillo, and S. Seelam, "FAIRIO: an algorithm for differentiated I/O performance," to be published in *SBAC-PAD '11: 23rd International Symposium on Computer Architecture and High Performance Computing*, Oct 26-29. 2011.

[2] J. S. Bucy, G. R. Ganger, and Contributors, "The DiskSim simulation environment version 3.0 reference manual," School of Computer Science, Carnegie Mellon University, Tech. Rep. CMU-CS-03-102, January 2003.