Bounding Disk I/O Response Time for Real-Time Systems

V. Brocal, M. Masmano, I. Ripoll, and A. Crespo
Industrial Informatics and Real-Time Systems Group
Universidad Politecnica de Valencia
Cami de Vera s/n, Valencia (Spain)
{vibrotor, mmasmano, iripoll, acrespo}@ai2.upv.es

Abstract

Real-time I/O scheduling, specifically disk scheduling, has frequently received far less attention than other aspects of real-time theory.

Nevertheless, several proposals have addressed this area meaning to merge the real-time CPU scheduling along with traditional I/O scheduling to provide real-time disk response. These proposals are all based on simplistic and unrealistic disk models that only consider mechanical parameters, leaving apart capabilities such as prefetching or write-caching.

In our opinion, real-time I/O scheduling algorithms should take advantage of these additional mechanisms to increase sensitively I/O performance.

In this paper, we present our first approach to a new model of a modern hard-disk (taken advantage of its features, specifically, its capability to identify streaming access patterns and prefetching a number of sectors belonging this stream).

Besides, we also describe how we have implemented our experiments in RTLinux.

1 Introduction

In the past, most efforts in the real-time theory have been devoted to CPU and network scheduling, which are essential for a large range of application involving real-time constraints. Nonetheless, with the emergence of new, more complex soft real-time applications which demand more system features such as I/O scheduling are being required to provide such systems with persistent mass storage capabilities.

Within this category, disk media has shown to be a reliable and cost-effective choice, though traditionally disk I/O has been considered as undeterministic. To cope with this unwanted undeterministic behaviour, several I/O scheduling algorithms have been proposed.

The schedulers proposed in [8] and [10] use the proximity of deadlines along with the seek distance between requests to establish the scheduling plan. In [3] an algorithm which is aware of the disk response time is presented along with others that follow the *earliest deadline* approach. More recent proposals, [7] and [1], take advantage of more realistic disk models to choose schedule order.

Nevertheless, most of these algorithms do not consider disk response time, or use simplistic models which over-estimates the time needed to serve each I/O request.

As shown in [9], a more complicated disk model, which includes the *read-ahead* feature, can bring us a more accurate service time estimation. In this paper, we follow this approach to provide a suitable disk model for real-time I/O cost.

The remainder of this paper is organised as follows: in the next section we propose a new model of hard-disk drive. In section 3, the environment and the loads that are being applied to test the model are described. The test scenarios and their results are shown in section 4. To wrap up, the advantages and drawbacks of the model are discussed in the conclusions section.

2 Disk I/O response time model

2.1 Response time model

The proposed model is the union of a mechanical model, as suggested in [4], and a linear time function

that models behaviour of the disk cache. At first instance the model checks for the availability of the requested blocks in the cache, and when not available, the mechanical model is used. Then, the response time $T_{r_i}(s_i, l_i)$ of an I/O request $P_i = (s_i, l_i)$, being s_i and l_i the first sector to transfer and the length of the resquested block respectively, is expressed as:

$$T_r(s,l) = \begin{cases} T_{ovh} + C(l) & l \leq C_n(s) \\ T_{ovh} + C(l - C_n(s)) + \\ +M(s + C_n(s), l - C_n(s)) & l > C_n(s) \end{cases} \quad (1)$$

Where:

$\mathbf{T_{ovh}}$ $\equiv$ Controller overhead.

Command processing time for each I/O request. In modern devices is almost nonexistent.

$\mathbf{C_n(s)}$ $\equiv$ Number of cached sectors starting from s.

As will be seen in section 2.2, $C_n(s)$ depends on the sequence of requests and issue instant, and the belonging of s to a data stream. This model does not provide means to estimate analytically this value. However, there is no problem to obtain it empirically, since all the required information is available.

$\mathbf{C(l)}$ $\equiv$ Cache transfer time for a block of l sectors.

Cache access response time. Even when a partial hit occurs, this term has a significant impact on the overall response time, since represents an access to a semiconductor memory, which is strongly faster than a physical access to a mechanical medium. A linear behaviour is expected:

$$C(l) = l\ C_r + C_{ovh} \quad (2)$$

Where C_r is the transfer rate (sectors/time) and C_{ovh} is the access overhead to the cache memory.

$\mathbf{M(s,l)}$ $\equiv$ Required time to access the physical media to transfer a data block of length l starting at sector s.

$M(s,l)$ represents the physical operations needed to access the magnetic medium[1]. These three operation correspond each one with a term of equation 3, as explained below:

$$M(s,l) = T_{seek}(s) + T_{rot} + T_{trans}(s,l) \quad (3)$$

s

[1] a platter

Seek time. $T_{seek}(s)$ Time needed to position the head arm in the cylinder $F_c(s)$ which the sector s belongs to. The current head arm position M_c should be known since seek time is a function of distance:

$$T_{seek}(s) = seek(|F_c(s) - M_c|)$$

Being P_{i-1} the previous request, M_c can be calculated as $M_c = F_c(s_{i-1} + l_{i-1})$. It could be thought that *read-ahead* operations may meddle this value, however, constraints imposed to these operations (see section 2.2) prevent such interferences. [9] and [4] agree to define $seek(d)$ function as:

$$seek(d) = \begin{cases} a_1 \sqrt{d} + b_1 & \text{if } d < c \\ a_2\ d + b_2 & \text{if } d \geq c \end{cases}$$

Where a_1, b_1, a_2, b_2 and c are model parameters.

Latency. T_{rot} Once the head arm is in the correct cylinder, the drive must wait until sector s has rotated under the head. As the rotation position of the platter is unknown at the beginning of each request, the total rotation time is used as upper bound.

Transfer time. T_{trans} Time that takes transferring l sectors from the magnetic surface. Modern disks are formatted using ZBR, so inner tracks have less sectors than outer ones. Then, the transfer time depends on the number of sectors of each track $F_n(s)$:

$$T_{trans}(s,l) = \frac{T_{rot} + T_{skew}}{F_n(s)}\ l + T_{skew}$$

The T_{skew} in the second coefficient covers the transfers involving the last sector of a track and the first of next track. T_{skew} is the fraction of rotation time needed to perform a head or cylinder skew.

2.2 Cache operation model

Along with the *response time model*, a *cache operation model* is needed to establish the *read-ahead* operation behaviour. Pending questions such as calculating the value for $C_n(s)$ must be addressed. As written forward, this value is derived from the block of sectors saved in cache when last *read-ahead* operation $RA(s, T_{ra})$ occurred, which depends on the request sequences. The following rules control the behaviour of *read-ahead* mechanism:

1. If s belongs to a data stream, the mechanism is always capable of detecting it.

2. If the number of data streams is larger than the number of segments in the cache C_n, then $\forall s : C(s) = 0$.

3. The maximum number of cached sectors for each stream is limited by C_{max}.

4. The *read-ahead* operation is interrupted *immediately* by any I/O request.

5. The *read-ahead* operation can not cross track boundaries, i.e. neither seek nor head switch operation can be performed.

The constraint 4 guarantees that an I/O operation can not be delayed by an in-progress cache preload operation. From 5 no request is overloaded by a seek in-progress operation[2], nor the value of M_c be altered influencing T_{seek}. Constraints 3 and 5 lead to the expression of the number of sectors retrieved by a *read-ahead* operation, $RA(s, T_{ra})$, which starts at sector s and has a duration of T_{ra}:

$$RA\left(s, T_{ra}\right) = \min\left\{C_{max}, F_e\left(s\right) - s, \left\lfloor \frac{T_{ra}}{T_{rot}} F_n\left(s\right) \right\rfloor\right\}$$

Where, $F_e(s)$ is the last sector of the track of s, and T_{ra} is obtained empirically, as the time elapsed since the last I/O request ended and the moment where the *read-ahead* was interrupted (constraint number 4), i.e. next request arrived. Modern hard-disk drives use segmented caches to support *read-ahead* for multiple streams. An algorithm adjusts the size of each segment to fit data rate, avoiding minimising the allocated cache buffer. However, these algorithms, along with other cache related issues, are subjected to patent restrictions, so there exist a large range of alternatives. For these reasons, we have decided to assume a static segment size allocation schema for the definition of the model, with C_{max} as the segment size obtained empirically.

Assuming P_i as the last I/O request, then a *read-ahead* operation is triggered when:

i. The disk is idle. There are not any remaining I/O requests to server after P_i (constraint number 4).

ii. P_i belongs to a data stream as required by constraint number 1.

iii. The next sector of the stream is not cached, i.e. $C(s + l + 1) = 0$

[2]typically, seek operations can not be aborted
[3]not tested

3 Experimental set up

According to our experience, RTLinux [11] has shown to be a stable and mature platform to implement applications with hard real-time constraints. It has encourage us to use RTLinux to implement this new hard-disk model and to test it. However, RTLinux lacks of its own hard-disk driver. Currently, when a real-time application requires of these capabilities Linux's drivers are used instead. This is a good solution for the most of the existing RTLinux applications but not for us. Therefore, our first challenger before implementing our model was to implement a RTLinux hard-disk driver.

Beyond being RTLinux native, our driver is capable of performing probe operations to detect and configure present hardware, and it has been designed to introduce minimum overhead. For this purpose, DMA transfers are being used along with IRQ data transfer completion notification. The code associated to the IRQ handling has been optimised in this sense: when an interrupt is received, the only operations performed in IRQ context are its acknowledge, and its occurrence notification to the thread which was waiting for. Status check and further operations are performed in normal CPU-scheduled context. The exclusive I/O access is achieved through a mutex, thus the requests are served in CPU scheduler order. In addition, modern LBA sector addressing schema is supported, as well as large disk 48-bit addressing feature set[3].

By using this driver, the parameters required by our model have been measured through the techniques proposed in [2] and in [4].

Table 1 shows the parameters obtained from the target disk WDC-WD136BA and the table 2 shows disk geometry.

To test the model, a constant bandwidth I/O generator feeds the model implementation and the WDC-WD136BA, through the referred driver, using an RT-task. As the model is aimed to take advantage of locality reference, the generated load has to follow a stream pattern with enough time within requests to allow the *read-ahead* mechanism to pre-fetch the maximum number of sectors. Varying the request length in the range [0, C_{max}], allows us to evaluate the profit of the proposed cached model versus the mechanical model. We have used the following I/O synthetic loads:

Period	300 ms
Length	1, 50, 200, 350, 385 sector/request

General parameters

T_{ovh}	0 ns
T_{rot}	8,312,032 ns
T_{skew}	2,401,344 ns

Cache parameters

C_r	14,616 ns/sector
C_{ovh}	106,843 ns
C_{max}	385 sectors
C_n	2 segments

Seek function

$$seek(d) = \begin{cases} 124770\sqrt{d} + 702938 & \text{if } d < 1834 \\ 122\ d + 6500000 & \text{if } d \geq 1834 \end{cases}$$

TABLE 1: *Model parameters for WDC-WD136BA*

Initial cylinder	Number of cylinders	Sectors
0	13131	450
13131	7033	432
20164	6989	420
27153	7644	405
34797	4540	390
39337	3201	378
42538	7800	360
50338	6742	330
57080	4064	315
61144	2921	300
64065	5793	270

TABLE 2: *WDC-WD136BA disk format*

4 Model validation: early outcomes

In our first experiments to validate this new hard-disk model, we have compared it with a real hard-disk and with other previously existing model: a disk model usually referred in bibliography which only regards the mechanical operation of a disk. The results are expressed in absolute and relative error respect the outcomes obtained from the real disk.

Cache-enabled disk - Mechanical model This scenario shows how the classic mechanical model does not take advantage of caching capabilities. As can be seen in figure 1 increasing the request length will decrease the error very fast, since the *read-ahead* mechanism offer less benefit. This shows the importance of the model presented.

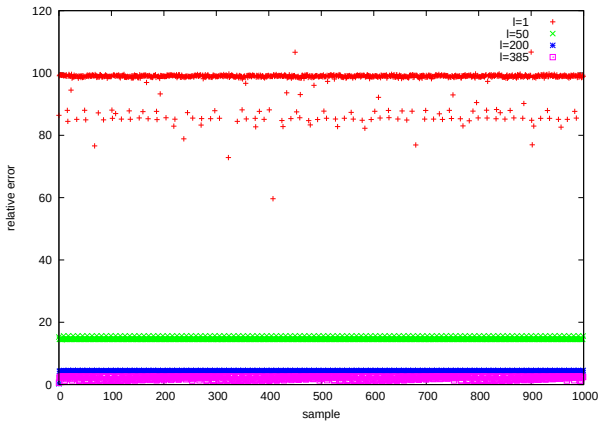

FIGURE 1: *Relative error for scenario 2*

Cache-enabled disk - Caching-capable disk model The benefit of using a caching-aware disk model is illustrated. As can be seen, relative error for the lowest request lengths has dropped drastically. Although cache failures still produce very high relative errors[4], the number of cache failures are proportional to request length, recall the C_{max} and track boundary constraints (figure 3).

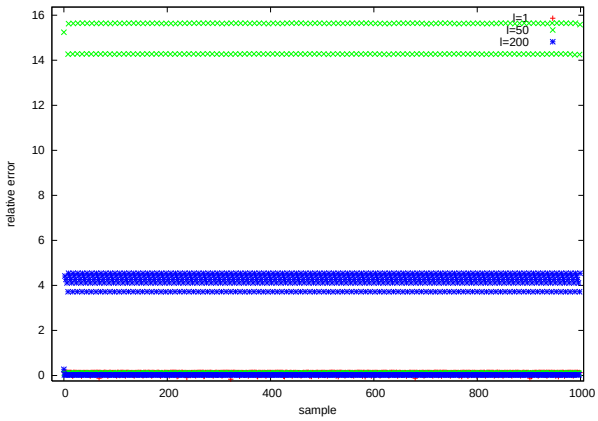

FIGURE 2: *Relative error for scenario 3*

[4]Peak errors are not shown for $l = 1$, in means for figure cleanness.

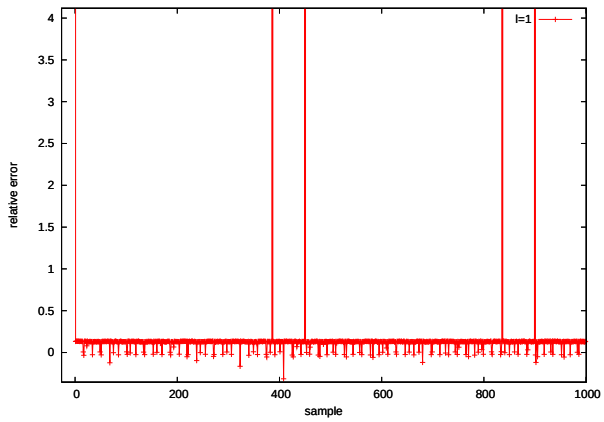

FIGURE 3: *Detailed view for $l = 1$. Scenario 3.*

Cached disk model - Mechanical model When comparing the error frequencies for the two models depending on the request length (figure 4), we believe that it becomes clear that using the proposed model more accurate service time predictions are possible.

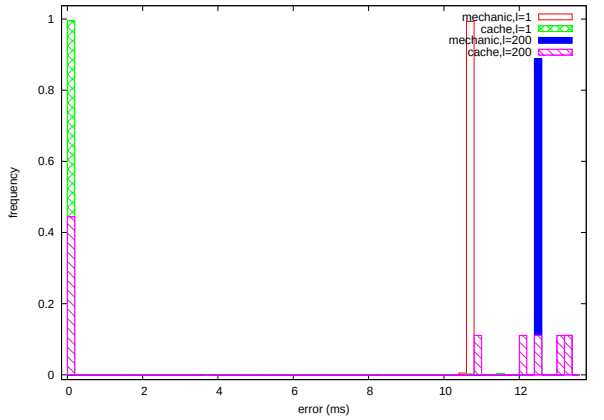

FIGURE 4: *Absolute error frequency for complete cached and mechanical-only disk models*

5 Conclusions and future work

A new hard-disk model has been proposed, and evaluation tests show off that an important benefit can be achieved using such a model, since more accurate response time are provided. Although these benefits only take place when data streams are presented, it is a common data pattern in real-time systems where I/O is needed. Furthermore, this model could be also applied to I/O scheduling with minimal modifications to detect I/O stream as required by cache operation constraint 1. Moreover, new scheduling algorithms could take advantage of pre-fetching techniques by reordering the requests to maximise *read-ahead* and then cache hits. A serious drawback of the model is revealed here since the lack of an analytic expression for obtaining the value of $C_n(s)$ makes impossible to use this model in schedulability tests.

As explained, the necessity of a formula for the calculation of $C(s)$ needs to be addressed to achieve a fully usable model. As well, more test with multiple stream and different I/O load patterns should be performed in means of evaluating it's benefit and usability in a largest range of applications. In the end, the proposal of a new I/O scheduling algorithm aware of *read-ahead* mechanism could be contemplated.

References

[1] Tai-YiHuang Chu, Edward T.-H. and Cheng-HanTsai. *EMSOFT 2004 - Fourth ACM International Conference on Embedded Software*. Association for Computing Machinery, 2004.

[2] G.R. Ganger and J.Schindler. Automated disk drive characterization. *Performance evaluation review*, 28(1):112–113, 2000.

[3] H. Garcia-Molina and R.K.Abbott. *[1990] Proceedings 11th Real-Time Systems Symposium*. 1990.

[4] N Lambert and OMesut. Hdd characterization for a/v streaming applications. *IEEE transactions on consumer electronics*, 48(3):802–807, 2002.

[5] Bradley G.Wherry Love, J. Spencer and RamakrishnaKaredla. Caching strategies to improve disk system performance. *Computer*, 27(3):38–46, 1994.

[6] J.Wilkes Merchant, A. and E.Shriver. An analytic behavior model for disk drives with readahead caches and request reordering. *Performance evaluation review*, 26(1):182–91, 1998.

[7] RCChang Shih, WK and RIChang. Real-time disk scheduling for multimedia applications with deadline-modification-scan scheme. *Real-time systems*, 19(2):149–168, 2000.

[8] et al Stankovic, John A. Performance evaluation of two new disk scheduling algorithms for real-time systems. *Real-time systems*, 3(3):307–336, 1991.

[9] J. Wilkes and C.Ruemmler. An introduction to disk drive modeling. *Computer*, 27(3):17–28, 1994.

[10] J.1 Wyllie and A.L.Narasimha Reddy. *Proceedings ACM Multimedia 93*. ACM, New York, NY, USA, 1993.

[11] V. Yodaiken. The RTLinux manifesto. In *Proc. of The 5th Linux Expo, Raleigh, NC*, March 1999.