Real-Time Disk Scheduling Using Deadline Sensitive SCAN

Kartik Gopalan

Computer Science Department
State University of New York at Stony Brook
Stony Brook, NY 11794-4400

kartik@cs.sunysb.edu

Abstract

This report describes the design and implementation of Deadline Sensitive SCAN (DSSCAN) algorithm for scheduling of real-time disk I/O requests. DSSCAN is a simple, yet powerful, hybrid of traditional EDF and SCAN algorithms. The main feature of DSSCAN is that, whenever possible, it closely imitates the behaviour of *SCAN* algorithm in order to increase the effective throughput of disk, subject to the constraint that deadlines of real-time requests are not missed. Unlike many other algorithms, DSSCAN does not have the notion of a fixed service cycle. This provides DSSCAN with the flexibility to schedule real-time requests with fine-grained deadlines. DSSCAN's natural framework supports scheduling of a mix of periodic real-time, aperiodic real-time, and best-effort requests. A simple variant of DSSCAN can easily support special class of interactive requests. In addition, DSSCAN has a desirable property that the disk throughput dynamically tracks how tight the request deadlines are - i.e. with sparse deadlines, DSSCAN essentially follows SCAN order, and with dense deadlines, it follows EDF order.

1 Introduction

Real-time applications such as real-time databases, multimedia applications, and real-time storage servers require that their disk I/O requests be serviced within a bounded time. In general, disk I/O requests can be classified into three categories based on their timing attributes - (1) periodic real-time (such as multimedia applications), (2) aperiodic real-time (such as real-time databases) and (3) best-effort (any non-real-time disk accesses).

Traditional non-real-time disk schedulers use the SCAN algorithm, or one of its variants [5], to process disk I/O requests. SCAN sorts requests according to their track positions and services them in the sorted order to reduce unnecessary seeks. SCAN is designed to maximize the disk throughput by minimizing seek time and does not take into account any deadline constraints on the I/O being performed.

The simplest algorithm for deadline based scheduling is EDF [1]. However it does not take into account the relative positions of requested data on the disk. This results in low disk resource utilization even when successive deadlines of requests are far apart.

A hybrid of SCAN and EDF algorithms (SCAN-EDF) is described in [6]. Requests with earliest deadline are served first. Several requests having the same deadline are served in SCAN order within a service cycle. Interactive requests, requiring low response times are merged into the SCAN order of the current service cycle. First drawback of this algorithm is that deadlines for real-time periodic requests are required to be multiples of basic service cycle. Second drawback of such a scheme is that the service cycle needs to be large enough if the advantages of SCAN ordering have to be realised. Large service cycle implies that only request streams with coarse-granularity of deadlines can be scheduled. Other algorithms for deadline based I/O are described in [7, 8, 9]. All of them use EDF schedule as the basic scheme and reorder requests so as to reduce seek and rotational latency overhead.

To achieve a performance level as close to the SCAN algorithm as possible while meeting all disk requests' deadlines, we propose a *Deadline Sensitive SCAN Algorithm* (*DS-SCAN*). The main features of DSSCAN are as follows

- It attempts to schedule disk I/O requests in essentially SCAN order without compromising the deadlines of real-time requests.
- It does not have the notion of a fixed service cycle. This provides DSSCAN with the flexibility to schedule real-time request streams with fine-grained deadlines.
- It's natural framework supports scheduling of a mix of periodic real-time, aperiodic real-time, and best-effort requests.
- With DSSCAN, the disk throughput dynamically tracks how tight the request deadlines are - i.e. with sparse deadlines, DSSCAN essentially follows SCAN order, and with dense deadlines, it follows EDF order.

A simple variant of DSSCAN can easily support special class of interactive requests. Interactive requests are those which require low response times, but have no specific deadlines associated with them. This makes interactive requests different from both real-time and best-effort requests.

The rest of the paper is organized as follows. Section 2 explains the DSSCAN algorithm and presents its variant for handling interactive requests. Section 3 describes the implementation of DSSCAN in the Integrated Resource Scheduling (IRS) system developed at SUNY, Stony Brook and section 4 presents the performance of DSSCAN algorithm.

2 The DSSCAN Algorithm

DSSCAN classifies requests into two classes - real-time and best-effort Real-time requests have a completion deadline associated with them. They could be either from periodic of aperiodic streams. Based on completion deadline and the size of the I/O request, a *start-deadline* can be computed. *Start-deadline* represents the latest time the real-time request has to be scheduled if it is to meet its completion deadline, assuming worst-case service time. Best-effort requests have no deadlines associated with them.

In the following subsections, we describe the the DSSCAN scheduler and start-deadline calculation algorithm in detail. Following that, we present a brief description of a DSSCAN variant for supporting the special class of interactive requests.

2.1 Scheduler

DSSCAN schedules disk I/O requests by maintaining two request queues - one ordered by start-deadlines and the other by SCAN order (i.e., track positions), as shown in Figure **??**. Each real-time request is queued in both start-deadline queue and SCAN queue. Best-effort requests are queued only in SCAN queue.

The *DS-SCAN* scheduler services the next request in the SCAN queue if this would not cause the request with the earliest start-deadline to miss its deadline. Otherwise, the scheduler services the disk request with the earliest start-deadline and then re-arranges the SCAN queue accordingly. Specifically, given the two request queues, DSSCAN uses the following rules to make a scheduling decision:

- Pick a request from the start-deadline queue according to the Earliest Deadline First (EDF) policy.
- Pick the first request in SCAN queue.
- If the two requests thus selected are different, schedule the request picked from the SCAN queue if its execution will not cause the request picked from the start-deadline queue to miss its deadline. Otherwise schedule the request picked from the start-deadline queue.
- Delete the chosen request from all queues it resides in, and rearrange the SCAN queue (if necessary) to begin from the location of chosen request.

$Current = ED_{N_{disk}}$;

$for \ \ (i = N_{disk}; i \geq 1; i--) \ \ \{$
$\quad SD_i = min(Current, ED_i) - X_i;$
$\quad Current = SD_i;$
$\}$

Figure 1: *The start deadline calculation algorithm computes the latest time a disk request should be started to meet its completion deadline, assuming the request's allocated delay budget is the worst-case service time.*

The last step is necessary for the following reason. Consider the situation when the request chosen to be scheduled the was not at the head of SCAN queue, i.e., a request r_d at the head of deadline ordered queue was chosen. Once r_d is serviced, the scheduler would like to attempt SCAN order scheduling from the disk head position where r_d was serviced. This makes it necessary to re-arrange the SCAN queue to begin from the location where r_d was serviced. This rearrangement can be performed in constant time if queues are maintained as doubly linked lists.

This scheduling algorithm has the desirable property that the system throughput tracks dynamically how tight the deadlines of real-time disk requests are. When there is much latitude for meeting disk requests' deadlines, *DS-SCAN* schedules disk requests mostly according to the scan order, and the overall throughput increases. On the other hand, when the resource requirements of real-time disk requests are close to the full system capacity, *DS-SCAN* follows mostly the deadline order, and the overall throughput goes down.

2.2 Calculating Start-Deadlines

Each real-time request is originally specified with a delay budget and a completion deadline. Assume the requests are sorted according to their completion deadlines in the ascending order, and their delay budgets and completion deadlines are X_i's and ED_i's, $i = 1, N_{disk}$. The algorithm shown in Figure 1 is used to calculate the start deadline of each request. It starts with the disk request with the largest completion deadline.

SD_i is the start deadline of the i-th disk request, which represents the latest time at which the i-th disk request should be started, assuming the request's delay budget is its worst-case service time. Whenever a new disk request is inserted into the start deadline queue, the start deadlines all the disk requests whose completion deadline is smaller than the new request's completion deadline need to be re-calculated according to the algorithm in Figure 1, starting with the new disk request, rather than with the request with the largest completion deadline. Similarly, when an existing disk request is serviced, the start deadlines all the disk requests whose completion deadline is smaller than the serviced request need to be re-calculated. Using the start deadlines instead of the completion deadlines significantly simplifies efficiency-conscious real-time disk scheduling such as *DS-SCAN*, because the start deadline of the disk request succinctly summarizes the total resource usage of *all* disk requests. That is, at any instant, if the start deadline of the first request in the queue can be satisfied, then the start deadline of all the subsequent requests in the queue can also be satisfied. As a result, *DS-SCAN* only needs to check the start deadline of the head of the deadline queue.

2.3 Supporting Interactive Requests

As defined earlier, *interactive* I/O requests are those which require low response times, for instance, I/O request generated by change in video sequence, or responses to mouse click events in window-based applications. The utility of such requests can be maximized by serving them quickly, but unlike real-time requests, there is no deadline associated with these requests. DSSCAN algorithm can be modified in the following manner to accomodate the class of interactive requests.

In addition to the two queues already described in section 2.1, DSSCAN maintains a third queue for interactive requests in First Come First Served (FCFS) order. Requests marked as interactive are queued *only* in the interactive queue and in no other queue. Steps for making the scheduling decision are modified as follows:

- Pick a request from the start-deadline queue according to the Earliest Deadline First (EDF) policy.

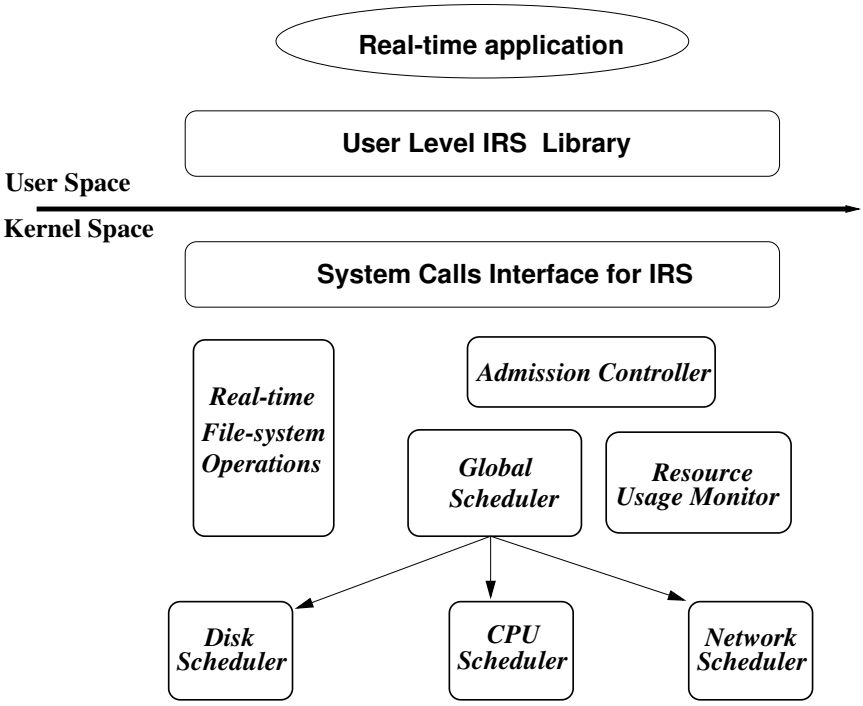

Figure 2: *The software architecture of the DSSCAN scheduler*

- If the interactive queue is not empty
  - Pick first request from interactive queue.
  - If execution of the selected interactive request will not cause the request from start-deadline queue to miss its deadline, schedule the interactive request. Otherwise, schedule the request from start-deadline queue.
- Else, interactive queue is empty, hence pick the first request in SCAN queue.
  - If the two requests thus selected are different, schedule the request picked from the SCAN queue if its execution will not cause the request picked from the start-deadline queue to miss its deadline. Otherwise schedule the request picked from the start-deadline queue.
- Delete the chosen request from all queues it resides in, and rearrange the SCAN queue (if necessary) to begin from the location of chosen request.

3 Implementation

DS-SCAN algorithm has been implemented as part of the IDE disk driver under LINUX operating system. The prototype runs on Intel Pentium-based machines running LINUX operating system. Figure 2 shows the software architecture of disk scheduler. A one-time admission controller manages the resource reservations and a usage monitor continuously keeps track of the resource consumption of individual tasks.

A single real-time read/write task may involve requests for I/O on multiple blocks of data on disk. In traditional OS such as LINUX, a process that issues multiple synchronous disk accesses in succession must go to sleep and get woken up once for every such access, thus incurring significant context switching and process scheduling overheads. Hence "blocking on one I/O event at a time" approach is inappropriate in real-time contexts. As an optimization, we modify the file-system read/write routines to simultaneously dispatching *all* requests associated with single read/write request before blocking the process. However, this optimization is not strictly necessary for correct funtioning of DSSCAN algorithm itself.

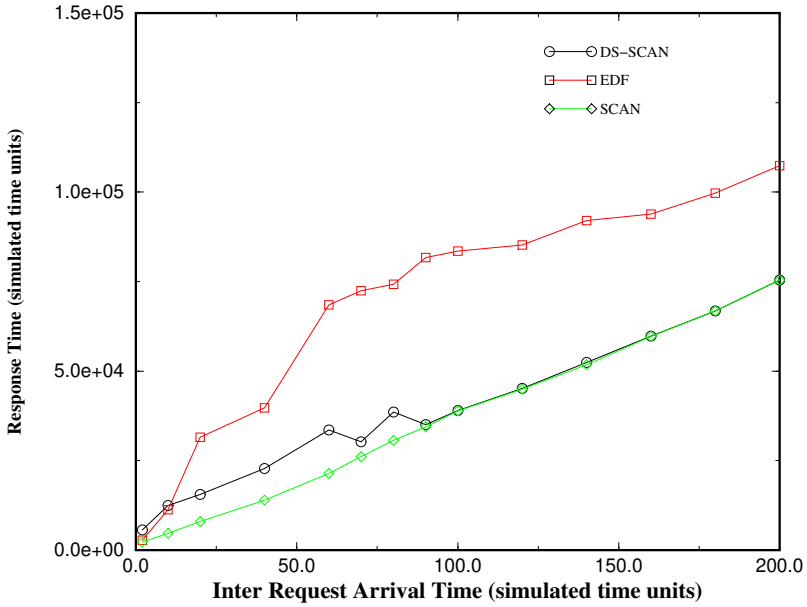

Figure 3: *Comparison of request service times (for servicing* 250 *I/O requests from* 5 *periodic streams) using* DS-SCAN, SCAN *and* EDF *disk scheduling policies with varying inter request arrival times. All times are in units of simulated time, where* 1 *unit of simulated time is equivalent to the time for transferring one block of data from disk.*

4 Performance

In this section we present the performance analysis of the *DS-SCAN* algorithm for scheduling real-time disk I/O requests. Apart from a working implementation of the algorithm on Linux, we also developed a simulation to compare the effective throughput of the three scheduling schemes, namely *SCAN*, *EDF*, and *DS-SCAN*. The program simulates five sources generating real-time disk I/O requests in a periodic fashion and these requests get serviced according to the scheduling policy being simulated. The rationale for having multiple sources is that each source accesses data from different files which are located at physically distant tracks or blocks on the disk. Thus an interleaved pattern of requests from all the sources excercises the movement of disk head across tracks, thus influencing the throughput observed. We varied the inter request arrival time at the disk scheduler by varying the period with which each stream issues I/O requests and measured the time taken for scheduling a total of 250 requests from all the five sources taken together. The lower the time taken, the better the throughput. In our simulation, *SCAN* policy completely ignores the deadlines associated with tasks, thus acting as a benchmark for the throughput that can be compared with other scheduling policies.

Figure 3 shows the result of our simulation. When requests arrive at the scheduler with smaller inter request arrival times, the deadlines are close to each other, resulting in the response time of *DS-SCAN* being closer to that of *EDF* policy. This is because, with closer deadlines, *DS-SCAN* has less latitude in scheduling requests by scan order and essentially follows the deadline order. With increasing inter request arrival times, deadlines of requests are spaced further apart and *DS-SCAN* uses this latitude to schedule requests in essentially scan order. This ability of *DS-SCAN* to dynamically adjust its behaviour depending on amount of real-time load is the key to getting higher I/O throughput and shorter response times.

5 Conclusions and Future Work

In this paper, we presented the design and implementation of a new real-time disk scheduling algorithm, DS-SCAN, that satisfies the deadline requirements of each disk I/O request while serving the requests by scan order whenever possible in order to achieve higher disk throughput. DSSCAN's simple design enables flexible fine-grained deadline specification for periodic and aperiodic real-time requests in addition to supporting best-effort requests. A simple variant of DSSCAN also supports low response times for special class of interactive requests.

While implementing DSSCAN we realized that one of the important factors in ensuring effectiveness of DSSCAN is accurate estimation of disk service time. Currently, the start-deadline calculation algorithm assumes worst case service time for each real-time I/O request. The next step in this work is to incorporate accurate request service-time estimation into DSSCAN algorithm.

References

[1] C.L. Liu and J.W. Layland, "Scheduling algorithms for multiprogramming environment in a hard real-time environment," Journal of the ACM, 20(1), 47-61, (1973).

[2] J.Neih and M.S.Lam, "The design, implementation and evaluation of SMART: A scheduler for multimedia applications", In Proc. ACM Symposium on Operating Systems Principles, St.Malo, France, Oct. 1997.

[3] Tzi-cker Chiueh, Chitra Venkatramani, Michael Vernick, "Design and Implementation of the Stony Brook Video Server", in Software – Practice and Experience, January 1997.

[4] Tzi-cker Chiueh, Kartik Gopalan, "Design and Implementation of Integrated Real-Time Resource Scheduling", Technical Report TR-56, Experimental Computer Systems Labs, Deptt. of Computer Science, SUNY at Stony Brook, May 1999.

[5] Brad Barclay, "A Survey of Disk Scheduling and Seek Optimization Techniques", http://yaztromo.idirect.com/seekopt.html

[6] A.L. Narasimha Reddy and J. Wyllie, "Disk Scheduling in Multimedia I/O System". In Proceedings of ACM Multimedia'93, Anaheim, CA, 225-234, August 1993.

[7] R.K. Abbot and H. Gracia-Molina, "Scheduling I/O Requests with Deadlines: A Performance Evaluation", In Proceedings of RTSS, 113-124, December 1990.

[8] M.J. Carey, R. Jauhari, and M. Linvy, "Priority in DBMS Resource Scheduling", In Proceedings of the 15th VLDB Conference, 1989.

[9] S.Chen, J.A.Stankovic, J.F.Kurose, and D.Towsley, "Performance Evaluation of Two New Disk Scheduling Algorithms for Real-Time Systems", Journal of Real-Time Systems, Vol. 3, 307-336, 1991.