Dynamic Sporadic Server

Moiz Zaheer Malik
B.Eng. Electronic Engineering
Hochschule Hamm-Lippstadt
Lippstadt, Germany
moiz-zaheer.malik@stud.hshl.de

***Abstract*—In real-time systems, where tasks have different levels of critical importance, it is essential to serve aperiodic (irregular, event-driven) tasks while ensuring that the deadlines of high-priority periodic tasks are not violated. The Dynamic Sporadic Server (DSS) is a scheduling method designed for Earliest Deadline First (EDF) systems that addresses this problem.**

DSS is defined by a period T_s and a budget C_s, but unlike traditional sporadic servers, it does not restore the full budget at every period. Instead, when an aperiodic task arrives, DSS assigns it a deadline and restores only the amount of budget that was actually used. This approach allows the processor to reach full (100 percent) utilization while ensuring that all deadlines are still met. DSS improves the response time of aperiodic tasks without compromising the guarantees of periodic tasks.

Originally introduced by Spuri and Buttazzo[5], DSS has been widely studied for its efficiency in managing mixed task sets under EDF scheduling. This report reviews the theory behind DSS, describes its operation (including budget management), highlights its advantages over traditional methods, and discusses potential application areas.

I. Introduction

Real-time systems often mix periodic tasks (recurring jobs with fixed periods and deadlines) and aperiodic/sporadic tasks (jobs which arrive unpredictably, e.g. interrupts or user requests). Ensuring that aperiodic tasks receive timely service without causing any periodic task to miss its deadline is a major challeng[2].

A common solution is to use a server abstraction a dedicated "server task" is given a share of the processor to handle aperiodic jobs. Examples include Polling Servers, Deferrable Servers, and Sporadic Servers. The Sporadic Server (SS), introduced by Sprunt et al[4], assigns a fixed priority (typically via Rate Monotonic) to the server and provides a budget C_s every period T_s. This allows aperiodic tasks to be executed with limited interference to hard real-time tasks. However, under static-priority scheduling the processor often cannot be fully utilized, unused server budget might be wasted or cause complex analysis.

With dynamic scheduling (EDF) one can achieve higher utilization. Dynamic priority servers extend the SS concept to EDF. In particular, Spuri and Buttazzo proposed the Dynamic Sporadic Server (DSS)[5]. DSS is characterized by a server period T_s and capacity C_s (the maximum aperiodic budget)[1]. Unlike SS, the DSS does not replenish C_s fully at each period start. Instead, the server's deadline and replenishment events are set dynamically whenever aperiodic work arrives or is consumed[5]. This allows DSS to adaptively use idle time for aperiodic tasks and to achieve 100 Percent CPU utilization if needed. The key property is that under the EDF schedule, the combined utilization of periodic tasks (U_p) plus the utilization of DSS ($U_s = C_s/T_s$) must satisfy

$$U_p + U_s \leq 1$$

In order to meet all deadlines[5].This EDF schedulability bound (100 Percent utilization) for DSS is analogous to the standard EDF limit for purely periodic tasks, which means that DSS does not reduce the processor's capacity for periodic jobs. In contrast, a fixed-priority SS can only guarantee less than full utilization without more complex analysis[5].

II. Background: Periodic and Aperiodic Tasks; EDF

periodic task: A periodic task τ_i generates an infinite sequence of jobs, each job requires execution time C_i and must finish by a relative deadline D_i after its arrival. Commonly, $D_i = T_i$ (deadline equals period). The total utilization of periodic tasks is

$$U_p = \sum_i \left(\frac{C_i}{T_i}\right)$$

Based on the study by Liu and Layland [3], EDF schedules any set of periodic tasks (with deadlines equal to periods) if and only if

$$U_p \leq 1$$

sporadic task: A sporadic task also has a minimum inter-arrival time (period) T_s and execution time C_s, but its jobs arrive irregularly (only the spacing is bounded). Sporadic tasks are often treated like a subset of periodic tasks where job activations are constrained by a minimum interval [1].

Aperiodic tasks: More general than sporadic, aperiodic tasks have no fixed period; they arrive unpredictably. Often aperiodic jobs have soft deadlines (misses are tolerable) or only request best effort service. To quantify their demand, one may consider the average (or maximum) aperiodic load U_a. The challenge is to service aperiodic jobs promptly without compromising the hard deadlines of periodic tasks [1].

EDF scheduling: Under EDF, jobs are assigned dynamic priorities equal to their absolute deadlines (earliest deadline = highest priority). The EDF schedulability criterion for mixed periodic and aperiodic tasks (when aperiodic jobs are treated like sporadic tasks with deadlines) remains

$$U_p + U_a \leq 1$$

(assuming job deadlines do not exceed their periods) [5]. However, simply queueing all aperiodic jobs under EDF can cause spikes in demand that violate periodic tasks' guarantees. Therefore, server algorithms are used to police how much CPU the aperiodic workload can use, effectively reserving bandwidth

$$U_s = \frac{C_s}{T_s}$$

for aperiodic service [1]. DSS is one such server, optimized for EDF-based systems [1].

III. Sporadic and Dynamic Servers

A. The Sporadic Server (SS)

Before DSS, the classic Sporadic Server (SS) was devised, initially for fixed-priority scheduling [1]. A sporadic server has parameters (T_s, C_s) and acts like a periodic task under fixed priority (typically, a rate-monotonic priority based on T_s). Its rule is: when an aperiodic job arrives and the server has available capacity, the server executes that job at high priority (as long as budget remains). After consuming an amount of execution u, the server schedules a replenishment of u after one period T_s. The Sporadic Server ensures that the server uses at most C_s time in any window of length T_s, preserving periodic task guarantees as long as

$$U_p + \frac{C_s}{T_s} \leq 1$$

where U_p is the utilization of periodic tasks. However, in SS the server's priority is fixed, making it not directly suitable for an EDF system [1].

B. The Dynamic Sporadic Server (DSS)

The Dynamic Sporadic Server (DSS) modifies SS for EDF scheduling [5]. Like SS, DSS has a server period T_s and capacity C_s, but crucially, DSS uses dynamic deadlines rather than a fixed priority. The server behaves like an EDF task whose deadline is recalculated at runtime whenever it begins servicing an aperiodic job. Its budget is not simply reset every T_s; instead, only consumed portions are "recharged" after T_s units.

The precise behavior of DSS is as follows [5, 1]:

Initialization: The server starts with full budget C_s. No deadline d_s is set until the first aperiodic job arrives.

Activation: When an aperiodic job arrives at time t_A and the server is idle with available budget > 0, the server sets its current deadline $d_s = t_A + T_s$ and schedules its next replenishment at $R_T = d_s$. The server immediately becomes the highest-priority EDF task (ties resolved in its favor) and begins executing aperiodic jobs. All pending aperiodic jobs in that busy interval share the same deadline d_s.

Execution: The server executes, deducting from its remaining capacity as it runs aperiodic jobs. If the aperiodic workload finishes before the budget is exhausted, the server waits with leftover budget. If more jobs arrive while budget remains, they are queued but served under the same deadline.

Budget Exhaustion or Job Completion: When capacity is exhausted or the last pending job finishes at time t_I, the server computes how much budget u was consumed since t_A and schedules a replenishment of amount u at time R_T. If pending jobs remain after exhaustion, they are paused until the replenishment.

Next Activation: After replenishment at time R_T, if pending jobs exist, the server repeats the activation step: it sets a new deadline $d_s = R_T + T_s$ and executes again.

These rules ensure that the server never consumes more than C_s time in any sliding window of length T_s [5, 1]. Thus, DSS behaves like a periodic task under EDF. The schedulability condition for mixing DSS with periodic tasks is:

$$U_p + U_s = \sum_i \frac{C_i}{T_i} + \frac{C_s}{T_s} \leq 1$$

where U_p is the utilization of periodic tasks and $U_s = C_s/T_s$ is the server's utilization.

Mathematical Formulation: Formally, it can be shown (Lemma and Theorem in [5]) that in any busy interval of length Δ, the server executes at most C_s time per T_s window. Thus, DSS achieves full utilization in EDF-based systems, as long as $U_p + U_s \leq 1$.

IV.

V. Dynamic Sporadic Server Simulation

In this example, we simulate two periodic tasks τ_1 and τ_2 along with a Dynamic Sporadic Server (DSS) that serves incoming aperiodic requests. Task τ_1 has period $P_1 = 8$ and execution time $C_1 = 2$, while τ_2 has $P_2 = 12$ and $C_2 = 3$. The DSS is configured with period $T_s = 6$ and capacity $C_s = 3$. Aperiodic tasks arrive at specific times (for example at $t = 3, 6, 9, \ldots$ with varying execution demands) and are queued for service by the DSS. The scheduler uses earliest-deadline-first (EDF) policy, breaking ties in favor of the server[1]. The DSS logic follows the standard rules: its budget is initialized to C_s and is replenished one period after each activation. Whenever a new aperiodic request arrives and the server is idle with remaining budget, the server becomes *active*, and its deadline is set to (current time + T_s). While active, the server executes as long as there is pending aperiodic work and budget remains; each execution unit decrements both the server's remaining capacity and the active aperiodic job's remaining work. If the server exhausts its budget or completes all queued aperiodic tasks, it becomes inactive and schedules its consumed budget to be replenished at the previously assigned deadline[1].

VI. DSS Simulation Overview

This section presents a simple C++ simulation of a Dynamic Sporadic Server (DSS) interacting with two periodic tasks and a queue of aperiodic requests. The DSS behavior follows the logic described in Section V, with a server period $T_s = 6$ and capacity $C_s = 3$. The goal of the simulation is to demonstrate how the DSS schedules aperiodic jobs without disrupting the periodic task execution, while managing its budget and deadline updates as described earlier.

Listing 1: Simplified DSS Simulation in C++

```
struct Task {
  int period, exec_time, next_release, remaining;
};

struct DSS {
  int Ts = 6, Cs = 3, rem = 3, deadline = -1;
  bool active = false;

  void activate(int t) {
    if (!active && rem > 0) {
      active = true;
      deadline = t + Ts;
    }
  }

  void consume() {
    if (rem > 0) rem--;
    if (rem == 0) active = false;
  }
};
```

In this simplified simulation, a main event loop updates each time unit. Periodic tasks are released based on their period, and the DSS becomes active when an aperiodic request arrives. The server's deadline is updated on activation, and each execution unit decrements its remaining capacity. When the capacity is exhausted, the server becomes inactive and schedules budget replenishment after T_s units.

TABLE I: Sample DSS Timeline Execution

Time	Task/Event
0–1	Periodic τ_1 executes
2	Periodic τ_2 executes
3–4	DSS handles aperiodic task (2 units)
5	Periodic τ_2 continues
6	DSS serves another job (1 unit)
8	Periodic τ_1 resumes
9	DSS replenished (2 units)
10	DSS serves job (1 unit from earlier queue)
11	Idle
12–13	DSS handles new aperiodic request (2 units)
14	Periodic τ_2 executes
15	DSS replenished (3 units)

VII.

VIII.

IX.

X. Conclusion

References

[1] Giorgio C. Buttazzo. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Springer Science & Business Media, 2011.

[2] Philip A. Laplante. *Real-Time Systems Design and Analysis: Tools for the Practitioner*. 4th. Wiley-IEEE Press, 2011.

[3] C. L. Liu and J. W. Layland. "Scheduling algorithms for multiprogramming in a hard-real-time environment". In: *Journal of the ACM (JACM)* 20.1 (1973), pp. 46–61.

[4] Barry Sprunt, Lui Sha, and John P. Lehoczky. "Aperiodic Task Scheduling for Hard-Real-Time Systems". In: *The Journal of Real-Time Systems* 1.1 (1989), pp. 27–60.

[5] Marco Spuri and Giorgio C. Buttazzo. "Efficient Aperiodic Service under Earliest Deadline Scheduling". In: *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 1994, pp. 2–11.