# 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[7], 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[4].

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[6], 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)[7]. DSS is characterized by a server period $T_s$ and capacity $C_s$ (the maximum aperiodic budget)[2]. 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[7]. 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[7].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[7].

## II. Background: Periodic and Aperiodic Tasks; EDF

**periodic task:** A periodic task $\tau_i$ 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 [5], 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 [2].

**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 [2].

**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) [7]. 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 [2]. DSS is one such server, optimized for EDF-based systems [2].

## III. Sporadic and Dynamic Servers

### A. The Sporadic Server (SS)

Before DSS, the classic Sporadic Server (SS) was devised, initially for fixed-priority scheduling [2]. 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 [2].

### B. The Dynamic Sporadic Server (DSS)

The Dynamic Sporadic Server (DSS) modifies the Sporadic Server (SS) for EDF scheduling [7]. 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 [7, 2].

The precise behavior of DSS is as follows [7, 2]:

**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 $d_s$ [7].

**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 [7].

**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 $T_s$ [7, 2]. 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 [2].

**Mathematical Formulation:** Formally, it can be shown (Lemma and Theorem in [7]) that in any busy interval of length $\Delta$, 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. Dynamic Sporadic Server Simulation

In this example, we simulate two periodic tasks $\tau_1$ and $\tau_2$ along with a Dynamic Sporadic Server (DSS) that serves incoming aperiodic requests. Task $\tau_1$ has period $P_1 = 8$ and execution time $C_1 = 2$, while $\tau_2$ 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[2, 8].

The DSS logic follows the standard rules: its budget is initialized to $C_s$ and is replenished one period after each activation[8]. 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 + $T_s$)[8]. 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[2, 8].

## V. 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 IV, with a server period $T_s = 6$ and

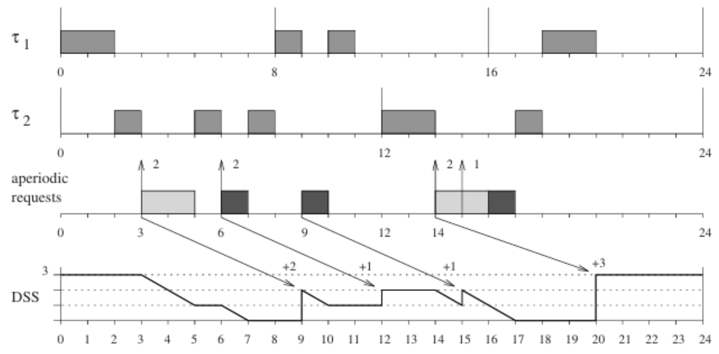

Fig. 1: Example for dynamic sporadic server[2]

| Time | Task/Event |
|---|---|
| 0–1 | Periodic $\tau_1$ executes |
| 2 | Periodic $\tau_2$ executes |
| 3–4 | DSS handles aperiodic task (2 units) |
| 5 | Periodic $\tau_2$ continues |
| 6 | DSS serves another job (1 unit) |
| 8 | Periodic $\tau_1$ 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 $\tau_2$ executes |
| 15 | DSS replenished (3 units) |

TABLE I: DSS Timeline Execution

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: 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 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[2], and each execution unit decrements its remaining capacity[8]. When the capacity is exhausted, the server becomes inactive and schedules budget replenishment after $T_s$ units[2].

Table I illustrates the execution timeline of the example. Initially, $\tau_1$ runs at time 0–1, then $\tau_2$ at $t = 2$. The first aperiodic request arrives at $t = 3$, so the DSS activates (deadline $t + T_s = 9$) and serves the request. The pattern continues, with the DSS using its budget when needed while preserving the schedulability of the periodic tasks[8]

## VI. Comparison with Other Servers

Table II compares key features of Polling, Deferrable, Sporadic, and Dynamic Sporadic servers. Polling servers operate under fixed-priority scheduling and reset their budget at each period regardless of use [6]. Deferrable servers also run at high priority but carry unused capacity to the end of the period [2]. The classic Sporadic Server (for Rate-Monotonic scheduling) replenishes only the portion of its budget that was actually consumed [6].

| **Feature** | **Polling** | **Deferrable** | **Sporadic** | **DSS** |
|---|---|---|---|---|
| Scheduling | Fixed-priority (RM) | Fixed-priority (RM) | Fixed-priority (RM) | Dynamic (EDF) |
| Priority | Highest static | Highest static | Static (RM) | Dynamic (deadline $t + T_s$) |
| Replenishment | Periodic full | Periodic full | On-demand | On-demand |
| Unused Budget | Lost if unused | Carried to period end | Carried to next period | Reused flexibly |
| Utilization | Often suboptimal | Improved response | Moderate | Full (up to 100%) |

TABLE II: Comparison of Aperiodic Servers

DSS stands apart by its EDF-based policy and flexible budget usage. In practice, DSS provides the responsiveness of a high-priority server without leaving CPU time idle when aperiodic demand is present [2, 4].

## VII. Applications and Use Cases

Dynamic Sporadic Servers (DSS) are well-suited to systems that must mix hard real-time periodic tasks with unpredictable aperiodic workloads. Typical examples include automotive ECUs, robotic controllers, avionics systems, embedded Linux platforms, multimedia devices, and medical equipment.

For instance, an automotive braking controller may have a periodic speed-monitoring task and a sporadic braking-request task [1]. In such domains, DSS provides a reserved capacity for aperiodic jobs without violating hard deadlines of periodic jobs.

In robotics, a servo loop may run at a fixed rate while sensor or vision processing tasks arrive unpredictably. DSS can absorb these aperiodic jobs using its time-varying deadline mechanism (e.g., "ties are always resolved in favor of the server") [8], improving responsiveness without jeopardizing safety-critical loops.

Similarly, avionics flight computers often interleave periodic control tasks (e.g., stabilization) with aperiodic ones (e.g., communication, diagnostics). In multimedia systems, the Linux `SCHED_DEADLINE` scheduler uses a variant of DSS (Constant Bandwidth Server) to serve aperiodic workloads (network packets, user interactions) alongside real-time tasks [8].

In medical systems such as implantable monitors or patient alarms, DSS can isolate life-critical periodic sensing from sporadic events like logging or alarms.

- **Automotive ECUs:** DSS allocates a fixed budget $C_s$ per period $T_s$ to event-driven aperiodic tasks without delaying periodic engine control jobs [1].
- **Robotics and Control:** DSS enables mixing real-time servo loops with irregular processing (vision/path planning) while meeting EDF feasibility conditions [8].
- **Avionics:** DSS helps meet deadlines for both periodic sensor/actuator cycles and asynchronous diagnostics.
- **Embedded Linux & Multimedia:** DSS variants like CBS are used in Linux for EDF-based scheduling of sporadic user interactions or audio/video streams [8].
- **Medical Systems:** DSS ensures timely response to alarms while maintaining guaranteed periodic monitoring rates.

DSS offers strong temporal isolation: aperiodic tasks are treated like a periodic task with utilization $U_s = C_s/T_s$. Spuri and Buttazzo showed that if $U_p + U_s \leq 1$, all EDF deadlines will be met [7, 2].

## VIII. Limitations and Practical Considerations

Implementing DSS involves non-trivial complexity. The operating system must manage a dynamic deadline and a replenishable budget. Each aperiodic job arrival or execution requires updating the deadline and tracking consumed capacity [8]. This includes scheduling a replenishment for any consumed portion after $T_s$ time units.

Whenever an aperiodic job executes, it inherits the server's deadline and may preempt periodic jobs. This makes DSS more complex than simple polling or deferrable servers [8, 3].

In systems using fixed-priority scheduling (e.g., Rate Monotonic), DSS cannot be directly implemented. It relies on EDF for dynamic prioritization. While real-time Linux and some RTOSes support CBS/DSS, integration still requires modifying scheduler internals.

There is also an overhead trade-off: smaller $T_s$ reduces latency but increases context switching. A short $T_s$ means smaller budgets $C_s$, which may lead to queue buildup. Conversely, large $T_s$ values reduce overhead but delay service to aperiodic jobs [3].

Heavy aperiodic workloads can overwhelm the server. If job arrivals exceed $U_s$, delays increase sharply. Worst-case latency occurs when jobs arrive just after budget exhaustion. System designers must choose $T_s$ and $C_s$ carefully to balance responsiveness with feasibility.

Corner cases also matter. If a server and a periodic job become ready at the same time, tie-breaking policies must prioritize the server [8]. Priority inheritance protocols are needed to avoid deadlocks when tasks share resources. In multicore systems, sharing the server budget across cores further complicates scheduling.

In summary, DSS offers predictability but requires careful implementation and tuning.

## IX. Conclusion

The Dynamic Sporadic Server is a powerful scheduling mechanism that extends EDF by offering guaranteed bandwidth to aperiodic tasks. By dynamically assigning deadlines and replenishing only the used capacity, DSS achieves efficient processor utilization while respecting hard deadlines for periodic tasks [7, 2].

Its strengths include:

- Strong temporal isolation and bandwidth reservation.
- High responsiveness to aperiodic jobs.
- Full utilization up to 100% under EDF schedulability bounds.

DSS is particularly beneficial in mixed-criticality real-time systems such as automotive, avionics, robotics, and embedded Linux environments. Compared to simpler servers, it provides better control and responsiveness, especially under unpredictable workloads.

Looking forward, DSS can be enhanced through variants like the Constant Bandwidth Server (CBS), integration with priority-inheritance protocols, and adaptations for multicore platforms [4]. Future research may also explore hybrid EDF-fixed priority schemes and virtualization-aware servers.

In conclusion, DSS remains a relevant and effective solution for real-time systems that require dynamic, predictable aperiodic task management.

## References

[1] Magnus Bengtsson and Elias Karlsson. *Real-Time Task Scheduling in Automotive ECUs*. https://www.diva-portal.org/smash/get/diva2:1420547/FULLTEXT01.pdf. Accessed: May 1, 2025. 2020.

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

[3] John Kingston and Ying Wang. *Real-Time Server Mechanisms in Multicore EDF*. https://research.cs.queensu.ca/techreports/2023-EDF-Server.pdf. 2023.

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

[5] 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.

[6] 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.

[7] 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.

[8] University of British Columbia. *CPEN 432 Course Notes: 04 – Scheduling Real-Time Tasks*. https://cpen432.github.io/notes/04-scheduling-real-time-tasks/index.html. Accessed: May 1, 2025. 2025.