Deferrable Server (DS)

Heansuh Lee
Electronic Engineering
Hochschule Hamm-Lippstadt
Lippstadt, Germany
heansuh.lee@stud.hshl.de

***Abstract*—In this paper, the Deferrable Server (DS) will be covered in details. DS is one of the periodic servers, and it is the simplest bandwidth-preserving server that improves response time of aperiodic tasks as compared to polling server (PS).**

Unlike PS, DS preserves its budget if no aperiodic requests are pending upon its invocation. It schedules aperiodic jobs that arrive later during its period until the budget is exhausted.

This paper also covers the usage of established mathematical proofs and how to implement them properly in practice, which cover (1) Consumption Rule, and (2) Replenishment Rule, for a better understanding of this concept.

DS allows users to have the knowledge to plan and set up a real-time system both on paper and in practice. It is critical to achieve timing correctness in embedded systems, which means to guarantee that the system reacts within the real-time requirements.

***Index Terms*—DS, deferrable server, consumption rule, replenishment rule, polling server, real-time systems, real-time, embedded systems**

I. Introduction

Definition of DS: A deferrable server is the simplest of bandwidth-preserving servers. Like a poller, the execution budget of a deferrable server with period p_s and execution budget e_s is replenished periodically with period p_s. Unlike a poller, however, when a deferrable server finds no aperiodic job ready for execution, it preserves its budget.

The basic idea of this fixed-priority server is to handle aperiodic requests from the beginning of its execution until:

- End of its period (T_S), or
- Its capacity (C_S) gets exhausted.

The capacity is replenished at the beginning of each period.

A. Deferrable Server

The complexity of the implementation of a DS is similar to the one of a polling server (PS). The average response time to aperiodic requests is improved with respect to the PS, since it is possible to use the capacity of the DS during the whole period, provided that its capacity is not exhausted.

However, there is a negative impact on the schedulability of the periodic tasks. The reason for this impact is that the delayed executions increase the load on the future. For example, it is possible having two consecutive executions (back-to-back execution).

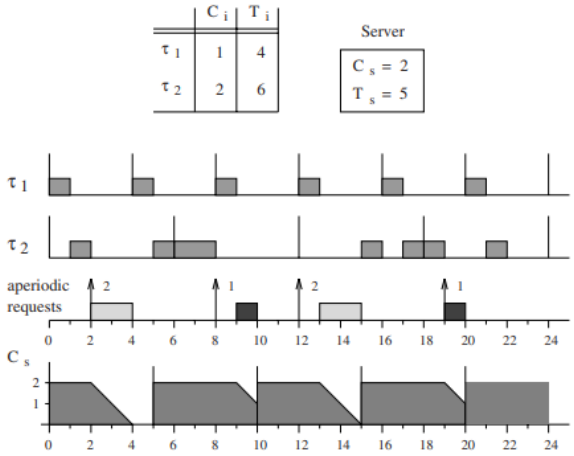

Fig. 1. Example of a Deferrable Server scheduled by RM [1].

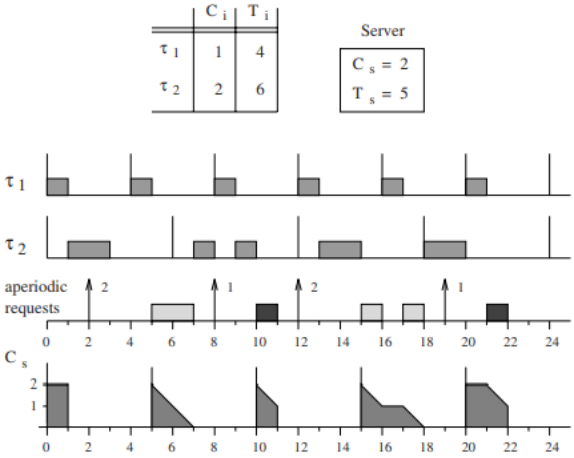

Fig. 2. Example of a Polling Server scheduled by RM [1].

The DS algorithm is illustrated in Fig. 1 using the same task set and the same server parameters (C_s = 2, T_s = 5) considered in Fig. 2. At time t = 1, when τ_1 is completed, no aperiodic requests are pending; hence, the processor is assigned to task τ_2. However, the DS capacity is not used for periodic tasks, but it is preserved for future aperiodic arrivals. Thus, when the first aperiodic request arrives at time t = 2, it receives immediate

service. Since the capacity of the server is exhausted at time t = 4, no other requests can be serviced before the next period. At time t = 5, C s is replenished at its full value and preserved until the next arrival. The second request arrives at time t = 8, but it is not served immediately because τ_1 is active and has a higher priority [1].

II. Schedulability Analysis

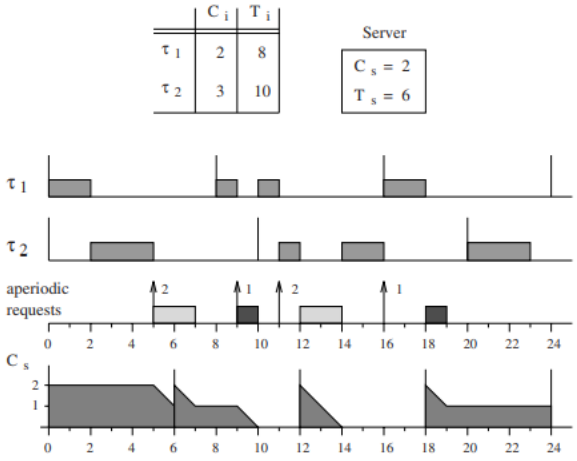

Fig. 3. Example of high-priority Deferrable Server [1].

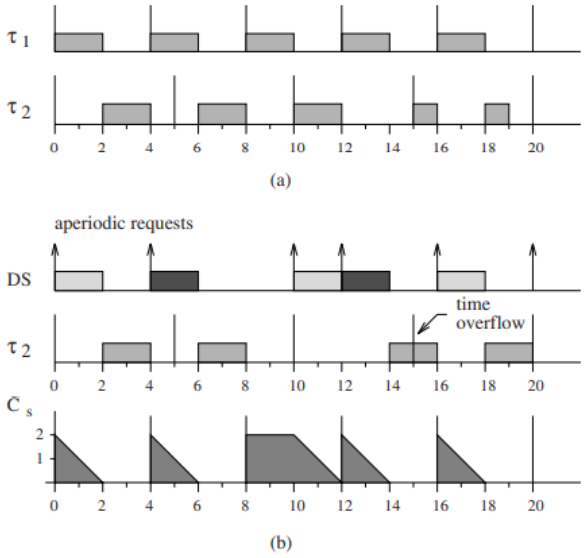

Fig. 4. DS is not equivalent to a periodic task. In fact, the periodic set τ_1, τ_2 is schedulable by RM (a); however, if we replace τ_1 with DS, τ_2 misses its deadline (b) [1].

Any schedulability analysis related to the Rate-Monotonic (RM) algorithm has been done on the implicit assumption that a periodic task cannot suspend itself, but must execute whenever it is the highest-priority task ready to run. It is easy to see that the Deferrable Server violates this basic assumption. In fact, the schedule illustrated in Fig. 3 shows that DS does not execute at time t = 0 although it is the highest-priority task ready to run, but it defers its execution until time t = 5, which is the arrival time of the first aperiodic request.

If a periodic task *defers* its execution when it could execute immediately, then a lower priority task could miss its deadline even if the task set was schedulable. Fig. 4 illustrates this phenomenon by comparing the execution of a periodic task to the one of a Deferrable Server with the same period and execution time.

The periodic task set considered in this example consists of two tasks, τ_1 and τ_2, having the same computation time ($C_1 = C_2 = 2$) and different periods ($T_1 = 4, T_2 = 5$). As shown in Fig. 4(a), the two tasks are schedulable by RM. However, if τ_1 is replaced with a DS having the same period and execution time, the low-priority task τ_2 can miss its deadline depending on the sequence of aperiodic arrivals. Figure 5.9b shows a particular sequence of aperiodic requests that cause τ_2 to miss its deadline at time t = 15. This happens because, at time t = 8, DSdoes not execute (as a normal periodic task would do) but preserves its capacity for future requests. This deferred execution, followed by the servicing of two consecutive aperiodic requests in the interval [10, 14], prevents task τ_2 from executing during this interval, causing its deadline to be missed.

Such an invasive behavior of the DS results in a lower schedulability bound for the periodic task set. The calculation of the least upper bound of the processor utilization factor in the presence of Deferrable Server is shown in the next section.

A. Calculation of U_{LUB} for RM+DS

The schedulability bound for a set of periodic tasks with a Deferrable Server is derived under the same basic assumptions to compute U_{lub} for RM. To simplify the computation of the bound for n tasks, we first determine the worst-case relations among the tasks, and then we derive the lower bound against the worst-case model [2].

Consider a set of n periodic tasks, τ_1 ,...,τ_n, ordered by increasing periods, and a DS with a higher priority. The worst-case condition for the periodic tasks, as derived for the RM analysis, is such that T_1 ¡ T_n ¡ $2T_1$ In the presence of a DS, however, the derivation of the worst-case is more complex and requires the analysis of three different cases, as discussed by Strosnider, Lehoczky, and Sha [3]. For the sake of clarity, here we analyze one case only, the most general, in which DS may execute three times within the period of the highest-priority periodic task. This happens when DS defers its service at the end of its period and also executes at the beginning of the next period. More information can be seen from Fig. 5..

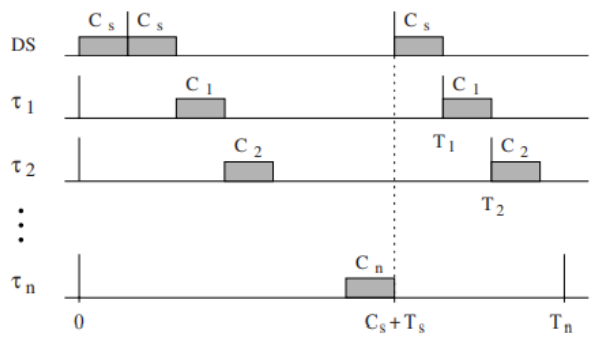

Fig. 5. Worst-case task relations for a Deferrable Server. [1].

III. Scheduling Example in RM

Initially, periodic tasks T_1 and T_2 are scheduled, according to the RM algorithm. This can be seen from Fig. 6. Then, we can add the deferrable server, scheduled according to the rate monotonic priority, but with the budget consumption and replenishment rules affecting its execution time. The DS is usually run at highest priority, but this is not strictly required.

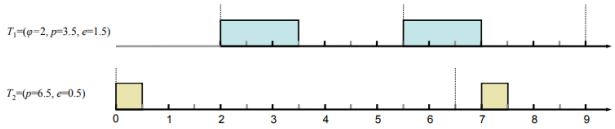

Fig. 6. An example of DS for periodic tasks included, T_1 and T_2 [4].

As seen from this example, specifically, the consumption and replenishment rules that define a DS (p_s , e_s) are as follows.

Consumption Rule: The execution budget of the server is consumed at the rate of one per unit time whenever the server executes.

Replenishment Rule: The execution budget of the server is set to es at time instants kp_k, for k = 0, 1, 2 . . . The server is not allowed to cumulate its budget from period to period. Stated in another way, any budget held by the server immediately before each replenishment time is lost.

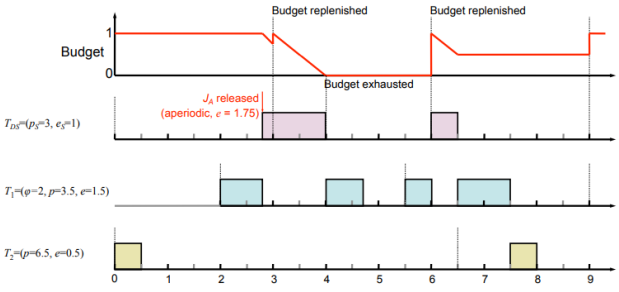

Fig. 7. An example of DS for periodic tasks included, T_1 and T_2, together with budget replenished/exhausted (consumed) [4].

In the DS, maximum utilisation test fails. Utilisation varies depending on arrival times of jobs executed by server. Therefore, it is essential to use time demand analysis based on critical instants to determine if the system can be scheduled. In finding the critical instants, we can assume a fixed-priority system T in which $D_i \leq p_i \ \forall \ i$ scheduled with a deferrable server (p_S, e_S) that has the highest priority among all tasks.

A critical instant of every periodic tasks T_i occurs at a time t_0 when all of the following are true:

- One of its jobs $J_{i,c}$ is released at t_0
- A job in every higher-priority periodic task is released at t_0
- The budget of the server is e_S at t_0, one or more aperiodic jobs are released at t_0, and they keep the server backlogged hereafter
- The next replenishment time of the server is t_0 + e

The definition of critical instant is identical to that for the periodic tasks without the deferrable server and the worst-case requirements for the server. The time-demand function is:

$$w_i(t) = e_i + \sum_{k=1}^{i-1}[\frac{t}{p_k} \cdot e_k + e_s + [\frac{t - e_s}{p_s} \cdot e_s]] = 1$$

Over here, the e_i refers to the execution time of job J_i, and the left-side fraction in summation with e_k is the execution time of higher priority jobs started during this interval, and the right-side fraction multiplied with e_s is the execution time of DS. To determine whether the task T_i can be schedule, we simply have to check whether $w_i(t) \leq t$ for some $t \leq D_i$. Also, this is a sufficient condition, not necessary. For instance, if this condition is not true, the schedule might still be correct.

In general, there is no maximum utilisation test for a fixed priority system with a DS. Except in one special case: a system of n independent, preemptable periodic tasks, whose periods satisfy p_s ¡ p_1 ¡ p_2 ¡ $\cdots$ ¡ p_n ¡ $2p_s$ and p_n ¿ $p_s + e_s$, where the relative deadlines are equal to their respective periods, can be scheduled rate monotonically with a deferrable server provided U ¡ $U_{RM/DS}(n)$ where:

$$U_{RM/DS}(n) = (n-1) \cdot [(\frac{u_s + 2}{u_s + 1})^{\frac{1}{n-1}} - 1]$$

It is easier to reason about the schedulability of a deadline-driven system with a deferrable server. The deadline of a deferrable server is its next replenishment time. A periodic task T_i in a system of N independent, preemptable, periodic tasks is schedulable with a deferrable server with period p_S, execution budget e_S and utilization u_S, according to the earliest deadline first (EDF) algorithm if:

$$\sum_{k=1}^{N} \frac{e_k}{min(D_k, p_k)} + u_s \cdot (1 + \frac{p_s - e_s}{D_i}) \leq 1$$

This must be calculated for each task in the system, since Di included. For example, tasks T_1=(3, 0.6), T_2=(5.0, 0.5), T_3=(7, 1.4) scheduled with a DS p_s=4, e_s=0.8 yield the left-hand side of inequality to be 0.913, 0.828 and 0.792 respectively; hence the three tasks are schedulable.

IV. Evaluation

Compared to a PS, it is more efficient in terms of carrying out the tasks with less time consumed. However, due to its complexity, it is not possible to prove correctness for RM. Yet, it can be effective for EDF systems, as shown in the previous example. Another limitation of DS would be that it may delay lower-priority tasks for more time than a periodic task with the same period and execution time, so it may be inefficient, as shown in Fig. 8.

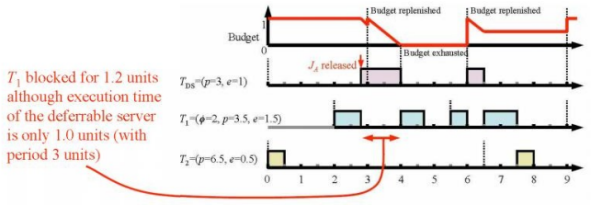

Fig. 8. A limitation of DS [5].

V. Conclusion

It is important to discuss about different kinds of servers together in comparison, such as sporadic and polling servers. This is because they have their own architectures that have advantages and limitations, and they are to be used at specific situations. DS is used at a more complex situation where EDF algorithm can be followed, and how it can be mathematically be proven with the right use of given formula.

IEEEtran

References

[1] Giorgio C. Buttazzo. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Real-Time Systems Series 24. Springer US, 3 edition, 2011.

[2] John P. Lehoczky, Lui Sha, and Jay K. Strosnider. Enhanced aperiodic responsiveness in hard real-time environments. In *Unknown Host Publication Title*, pages 261–270. IEEE, December 1987.

[3] The deferrable server algorithm for enhanced aperiodic responsiveness in hard real-time environments. *IEEE Transactions on Computers*, 44(1):73–91, January 1995.

[4] Real-time scheduling of aperiodic and sporadic tasks (1). https://csperkins.org/teaching/2012-2013/adv-os/lecture04.pdf. Accessed: 2022-05-15.

[5] Deferrable server. https://benchpartner.com/deferrable-server. Accessed: 2022-05-15.