Constant Bandwith Server

Bin Abdul Malik, Muhammad Amjad Student of Electronic Engineering Hochschule Hamm-LippstadtLippstadt, Germanymuhammad-amjad-bin.abdul-malik@stud.hshl.de

*Abstract*— Over the last few years, several algorithms and methodologies have been proposed to improve the predictability of real-time systems. This paper discusses about one of the scheduling systems which is Constant Bandwidth Server also known as CBS. This scheduling technique is frequently used to handle overruns and implement resource reservation in real-time systems where tasks have variable execution requirements. In particular, different service methods are introduced to reduce the average response time of aperiodic requests without compromising the scheduling sequence of hard periodic tasks. In order to present these results, we need to analyse and elaborate some basic concepts that will be used throughout this paper.

# Introduction

In real-time computing systems running multiple concurrent tasks, a fundamental property that must be ensured to support a component-based development is temporal protection, which prevents unexpected overruns occurring in a task from affecting the execution of other tasks. Resource Reservation represents the most powerful scheduling mechanism specifically conceived to achieve such a property [1]. The idea behind the notion of Resource Reservation is that each task (or set of tasks) is assigned a fraction of the CPU and is scheduled in such a way that it will never demand more than its reserved bandwidth. With this abstraction, processor capacity is viewed as a quantifiable resource that can be reserved, like physical memory or disk blocks.

The need for temporal isolation arises in many contexts. In the real-time community, its primary motivation was to integrate hard, soft, and non-real-time tasks. Indeed, many real-time systems are not characterized by hard timing constraints, as is the case of multimedia applications such as audio or video streaming. For these applications, missing a deadline has no catastrophic consequences, but it only leads to performance degradation. When dealing with hybrid task-sets, composed of hard and soft tasks, temporal isolation allows protecting hard tasks from overruns generated by soft tasks. More in general, achieving temporal isolation is necessary whenever a timely service must be ensured in a system with heterogenous timing requirements and potential overload conditions. In case of dynamic or unpredictable computational workload, the system must be able to reconfigure or adapt itself, without affecting other functionalities. In such circumstances, each application can be protected from the timing interferences of other components by using a proper enforcement mechanism that preserves the temporal isolation [4].

The resource reservation framework is also effectively employed for hierarchical systems composed of a set of modular components, each handling its own application, where a different scheduling algorithm may be used within each component. Component-based design is increasingly used as a de facto approach to design complex embedded systems. It gives the possibility to handle the growing complexity of current industrial software systems and to support the design of open environments, where independently developed real time applications need to be validated and executed in isolation. Resource reservation can be efficiently used in such situations, by allocating different applications on different virtual processors, so that each application can execute in isolation, without being affected by the behaviour of the other components [2]. But before that, it is good to have a grasp on the idea behind the type of tasks for a better understanding on this topic.

Many complex control applications include tasks which must be completed within strict time constraints and this is what we called as deadlines. If meeting a given deadline is critical for the system operation, and may cause catastrophic consequences, that deadline is considered to be hard. If meeting time constraints is desirable but missing a deadline does not cause any serious damage, then that deadline is considered to be soft. In addition to their criticalness, the tasks are divided into three which are periodic, aperiodic and sporadic. Basically, tasks that require regular activations are called periodic. In other words, periodic tasks execute their invocations within regular time intervals. Meanwhile, tasks which have irregular arrival times are called aperiodic and aperiodic tasks with hard deadlines are called sporadic tasks. Aperiodic tasks are invoked only once. Their arrival times are unknown at design time. A firm aperiodic task has the following a certain set of parameters which are, the arrival time, worst case execution time and relative deadline while soft aperiodic tasks have no deadline constraints. On the other hand, sporadic tasks can arrive at the system at arbitrary points in time, but with defined minimum inter-arrival times between two consecutive invocations. A sporadic task is characterized by its relative deadline, minimum inter-arrival time and worst case execution time. These attributes are known before the run-time of the system.

The aim of this research studies is to learn more efficient algorithms for the joint scheduling of random soft aperiodic requests and hard periodic tasks under the EDF policy. This research studies includes three algorithms having different implementation overheads and different performances. A completely new “bandwidth preserving algorithm”, called Total Bandwidth Server, is also introduced. The algorithm significantly enhances the performance of the previous servers and can be easily implemented with very little overhead, thus showing the best performance and cost ratio. We will also be discussing an optimal algorithm, the EDL Server and Constant Bandwidth Server. The proposed algorithms provide a useful framework to assist an HRT system designer in selecting the most appropriate method for his or her needs, by balancing efficiency with implementation overhead[8].

# Scheduling

## Scheduling Problems

We consider a real-time system as consisting of a fixed set of periodic tasks and dynamically occurring sporadic tasks. Each periodic task periodically generates service requests. The static timing attributes of every periodic task Ti are its worst execution time q, its period Pi and its critical delay Ri assumed to be less than or equal to the period. Sporadic tasks occur and require to be nm m the node at unpredictable times. Every sporadic task is ready to be processed upon acceptance on the node. Its static attributes are its worst-case execution time and its deadline. Both periodic and sporadic tasks have dynamic attributes which reflect their current evolution i.e the remaining execution time and the current deadline [5].

A hard real-time system must execute a set of concurrent real-time tasks in such a way that all time-critical tasks meet their specified deadlines. Every task needs computational, data, and other resources for example, input/output devices to proceed. The scheduling problem is concerned with the allocation of these resources to satisfy all timing requirements [3].

The major problem faced in the analysis of sporadic task systems concerns the unpredictability of the task requests. For periodic task systems, the set of task requests to be encountered is known a priori. However, there is no way of knowing which set of task requests would be encountered in irregular task systems. The key scheduling issue thus is on-line feasibility testing, which can be phrased as follows: The system must decide if a new sporadic task can be accepted when it arrives at the node. Acceptance signifies that the job can be completed in the time between its arrival and its deadline without jeopardizing the deadlines of periodic tasks or previously accepted sporadic activities. The accepted task is then queued until it is selected to be processed by the local scheduler. Otherwise, another module of the scheduler in responsibility of dispatching this task to another computer in the distributed system can take refusal into account [5].

## Static Versus Dynamic Scheduling

A scheduler is called static (or pre-run-time) if it makes its scheduling decisions at compile time. It generates a dispatching table for the run-time dispatcher off-line. For this purpose it needs complete prior knowledge about the task-set characteristics, for example, maximum execution times, precedence constraints, mutual exclusion constraints, and deadlines. On the other hand, a scheduler is called dynamic (or on-line) if it makes its scheduling decisions at run time, selecting one out of the current set of ready tasks. Dynamic schedulers are flexible and adapt to an evolving task scenario. They consider only the current task requests. The run-time effort involved in finding a schedule can be substantial. Figure below is provided to have a clearer view between the distinctions stated.

Diagram

Description automatically generated

1. Fractions of real-time scheduling with comparison between dynamic and static scheduling.

In general, the system behavior is non-deterministic. Non-preemptive and Preemptive Scheduling. In non-preemptive scheduling, the currently executing task will not be interrupted until it decides on its own to release the allocated resources. Non-preemptive scheduling is reasonable in a task scenario where many short tasks (compared to the time it takes for a context switch) must be executed. In preemptive scheduling, the currently executing task may be preempted or interrupted, if a more urgent task requests service [3].

## Centralized Versus Distributed Scheduling

In a dynamic distributed real-time system, it is possible to make all scheduling decisions at one central site or to develop cooperative distributed algorithms for the solution of the scheduling problem. The central scheduler in a distributed system is a critical point of failure. Because it requires up-to-date information on the load situations of all nodes, it can also contribute to a communication bottleneck.

Many industrial applications with real-time demands are composed of tasks of various types and constraints. Arrival patterns and importance, for example, determine whether tasks are periodic, aperiodic, sporadic, and soft, firm, or hard. The controlling real-time system must provide for a combined set of such task types. The same holds for the various constraints of tasks [9].

Nowadays, most real-time control applications require both types of processes (periodic and aperiodic), which may also differ for their criticality. Typically, periodic tasks are time-driven and execute critical control activities with hard timing constraints aimed at guaranteeing regular activation rates. Aperiodic tasks are usually event-driven and may have hard, soft, or non-real-time requirements depending on the specific application. When dealing with hybrid task sets, the main objective of the kernel is to guarantee the schedulability of all critical tasks in worst-case conditions and provide good average response times for soft and non-real-time activities.

Off-line guarantee of event-driven aperiodic tasks with critical timing constraints can be done only by making proper assumptions on the environment, that is, by assuming a maximum arrival rate for each critical event. This implies that aperiodic tasks associated with critical events are characterized by a minimum interarrival time between consecutive instances, which bounds the aperiodic load. Aperiodic tasks characterized by a minimum interarrival time are called sporadic. They are guaranteed under peak-load situations by assuming their maximum arrival rate. If the maximum arrival rate of some event cannot be bounded a priori, the associated aperiodic task cannot be guaranteed off-line, although an online guarantee of individual aperiodic requests can still be done. Aperiodic tasks requiring online guarantee on individual instances are called firm. Whenever a firm aperiodic request enters the system, an acceptance test can be executed by the kernel to verify whether the request

Next, we will discuss the problem of scheduling soft aperiodic tasks and hard periodic tasks under dynamic priority assignments. In particular, different service methods are introduced, the objective of which is to reduce the average response time of aperiodic requests without compromising the schedulability of hard periodic tasks. Periodic tasks are scheduled by the Earliest Deadline First (EDF) algorithm. With respect to fixed-priority assignments, dynamic scheduling algorithms are characterized by higher schedulability bounds, which allow the processor to be better utilized, increase the size of aperiodic servers, and enhance aperiodic responsiveness [6].

# Server

As mentioned earlier in this paper, Resource Reservation is typically implemented by assigning to each application a dedicated real-time server. This is called the reservation server. Each server is characterized by a budget Q and period P, so that it provides to the corresponding application Q units of service every P time-units. The ratio α = Q/P is called server bandwidth. If an application A is assigned a reservation bandwidth α, it behaves as it were executing on a dedicated slower processor, with speed α times the original speed. However, the reserved budget may be granted with some delay with respect to a dedicated virtual processor, depending on the particular implementation of the server [4].

In the next part of the paper, we will mainly discuss one of the most popular dynamic servers which is the Constant Bandwidth Server (CBS) alongside with the other two algorithms which are Total Bandwidth Server (TBS) and Earliest Deadline Late Server (EDL).

## TBS

An approach that we can follow to improve the aperiodic response times is to assign a possible short deadline to each aperiodic request. The assignment must be done in such a way that the overall processor utilization of the aperiodic load never exceeds a specified maximum value Us. This approach is the main idea behind the Total Bandwidth Server (TBS), which we define in the following section. The name of the server comes from the fact that, each time an aperiodic request enters the system, the total bandwidth of the server, whenever possible, is immediately assigned to it.

The implementation of the TB server is the simplest among those seen so far. In order to correctly assign the deadline to the new issued request, we only need to keep track of the deadline assigned to the last aperiodic request (dk-1). Then, the request can be queued into the ready queue and treated by EDF as any other periodic instance. Hence, the overhead is only due to the increased length of the ready queue if several aperiodic requests are pending at the same time. However, this problem can be solved by managing a separate FIFO queue for the aperiodic requests, and inserting only the first one into the ready queue. In this way the overall overhead is practically negligible.

## EDL

The The Total Bandwidth algorithm is able to achieve good aperiodic response times with extreme simplicity. Still we could desire a better performance if we agree to pay something more. For example, looking at the schedule in Figure 2, we could argue that the second and the third aperiodic requests may be served as soon as they arrive, without compromising the schedulability of the system. The reason for this is that, when the requests arrive, the active periodic instances have enough effective laxity (i.e., the interval between the completion time and the deadline) to be safely preempted. The main idea of the EDL algorithm is to take advantage of these laxities.

The EDL server mechanism is based on the following idea: the idle times of an EDL scheduler are used to schedule aperiodic requests as soon as possible, postponing the execution of periodic activities, similarly to the effect of the “Slack Stealer” of [8]. The optimality stated in Theorem 3 will give us the optimality of the server built with this idea. In particular, when there are no aperiodic activities in the system, the periodic tasks are scheduled according to the EDF algorithm. The analysis of the EDL server schedulability is quite straightforward. In fact, the server allocates to the aperiodic activities only the idle times of a particular EDF schedule, without compromising the timeliness of the periodic tasks.

Although optimal, the algorithm described in the previous section has too much overhead to be considered practical. However, its main idea can be usefully adopted to develop an implementable algorithm, still maintaining a nearly optimal behaviour, as shown later in the discussion of the simulations. What makes the EDL server not practical is the complexity of computing the idle times at each new aperiodic arrival. This computation must be done each time in order to take into account the periodic instances partially executed or already completed at the time of arrival. We can avoid the heavy idle time computation using the mechanism of priority exchanges. With this mechanism, in fact, the system can easily keep track of the time advanced to periodic tasks and possibly reclaim it at the right priority level. The idle times of the EDL algorithm can be precomputed off-line. The server can use them to schedule aperiodic requests, when there are any, or to advance the execution of periodic tasks [8].

## Constant Bandwith Server Insights

One major problem of the TBS algorithm is that it does not use a server budget for controlling aperiodic execution, but rely on the knowledge of the worst case computation time specified by each job at its arrival. When such a knowledge is not available, not reliable, or too pessimistic most probably due to highly variable execution times, then hard tasks are not protected from transient overruns occurring in the soft tasks and could miss their deadlines. The CBS algorithm can be efficiently used in these situations, since it has a performance comparable to the one of the TBS and also provides temporal isolation, by limiting the bandwidth requirements of the served tasks to the value Us specified at design time. Therefore, in this section we will start discussing on the Constant Bandwith Server and how it works.

In this section we present a novel service mechanism, called the Constant Bandwidth Server (CBS), which efficiently implements a bandwidth reservation strategy. As the Dynamic Sporadic Server, the Constant Bandwidth Server guarantees that, if Us is the fraction of processor time assigned to a server (i.e., its bandwidth), its contribution to the total utilization factor is no greater than Us, even in the presence of overloads. Note that this property is not valid for a TBS, whose actual contribution is limited to Us only under the assumption that all the served jobs execute no more than the declared WCET. With respect to the DSS, however, the CBS shows a much better performance, comparable with the one achievable by a TBS.

The basic idea behind the CBS mechanism can be explained as follows: when a new job enters the system, it is assigned a suitable scheduling deadline (to keep its demand within the reserved bandwidth) and it is inserted in the EDF ready queue. If the job tries to execute more than expected, its deadline is postponed (its priority is decreased) to reduce the interference on the other tasks. Note that by postponing the deadline, the task remains eligible for execution. In this way, the CBS behaves as a work conserving algorithm, exploiting the available slack in an efficient (deadline based) way, thus providing better responsiveness with respect to non-work conserving algorithms and to other reservation approaches that schedule the extra portions of jobs in background, as proposed by Mercer, Savage, and Tokuda [1].

If a subset of tasks is handled by a single server, all the tasks in that subset will share the same bandwidth, so there is no isolation among them. Nevertheless, all the other tasks in the system are protected against overruns occurring in the subset. In order not to miss any hard deadline, the deadline assignment rules adopted by the server must be carefully designed. The next section precisely defines the CBS algorithm, and formally proves its correctness for any (known or unknown) execution request and arrival pattern.

A CBS is characterized by a budget cs and by an ordered pair (Qs, Ts), where Qs is the maximum budget and Ts is the period of the server. The ratio Us = Qs/Ts is denoted as the server bandwidth. At each instant, a fixed deadline d s,k is associated with the server. At the beginning ds,0 = 0. Each served job Ji,j is assigned a dynamic deadline di,j equal to the current server deadline ds,k. Whenever a served job executes, the budget c s is decreased by the same amount. When cs = 0, the server budget is recharged at the maximum value Qs and a new server deadline is generated as ds,k+1 = ds,k + Ts. Note that there are no finite intervals of time in which the budget is equal to zero.

A CBS is said to be active at time t if there are pending jobs (remember the budget cs is always greater than 0); that is, if there exists a served job Ji,j such that ri,j ≤ t. A CBS is said to be idle at time t if it is not active. When a job Ji,j arrives and the server is active the request is enqueued in a queue of pending jobs according to a given arbitrary discipline like First In First Out (FIFO). When a job Ji,j arrives and the server is idle, if cs ≥ (ds,k − ri,j )Us the server generates a new deadline ds,k+1 = ri,j + Ts and cs is recharged at the maximum value Qs, otherwise the job is served with the last server deadline ds,k using the current budget. When a job finishes, the next pending job, if any, is served using the current budget and deadline. If there are no pending jobs, the server becomes idle. At any instant, a job is assigned the last deadline generated by the server.

Chart

Description automatically generated

1. CBS implementation.

Figure above illustrates an example in which a hard periodic task, τ 1, with computation time C1 = 4 and period T1 = 7, is scheduled together with a soft task, τ2, served by a CBS having a budget Qs = 3 and a period Ts = 8. The first job of τ2 (J2,1), requiring 4 units of execution time, arrives at time r 1 = 3, when the server is idle. Being cs ≥ (d0 − r1)Us, the job is assigned a deadline d1 = r1 + Ts = 11 and cs is recharged at Qs = 3. At time t = 7, the budget is exhausted, so a new deadline d2 = d1+Ts = 19 is generated and cs is replenished. Since the server deadline is postponed, τ1 becomes the task with the earliest deadline and executes until completion. Then, τ2 resumes and job J2,1 (having deadline d2 = 19) is finished at time t = 12, leaving a budget cs = 2. The second job of task τ2 arrives at time r2 = 13 and requires 3 units of time. Since cs < (d2 − r2)Us, the last server deadline d2 can be used to serve job J2,2. At time t = 15, the server budget is exhausted, so a new server deadline d3 = d2 + Ts = 27 is generated and cs is replenished at Qs. For this reason, τ1 becomes the highest priority task and executes until time t = 19, when job J 1,3 finishes and τ2 can execute, finishing job J2,2 at time t = 20 leaving a budget cs = 2.

It is worth noting that under a CBS a job Jj is assigned an absolute time-varying deadline dj that can be postponed if the task requires more than the reserved bandwidth. Thus, each job Jj can be thought as consisting of a number of chunks Hj,k, each characterized by a release time aj,k and a fixed deadline dj,k. An example of chunks produced by a CBS is shown in Figure 6.14. To simplify the notation, we will indicate all the chunks generated by the server with an increasing index k (in the example of Figure 6.14, H1,1 = H1, H1,2 = H2, H2,1 = H3, and so on).

## Constant Bandwith Server Properties

After The proposed CBS service mechanism presents some interesting properties that make it suitable for supporting applications with highly variable computation times (e.g., continuous media applications). The most important one, the isolation property, is formally expressed by the following theorem and lemma.

Theorem:

The CPU utilization of a CBS S with parameters (Qs, Ts) is Us = Qs Ts , independently from the computation times and the arrival pattern of the served jobs.

The following lemma provides a simple guarantee test for verifying the feasibility of a task set consisting of hard and soft tasks.

Given a set of n periodic hard tasks with processor utilization Up and a set of m CBSs with processor utilization, the whole set is schedulable by EDF if and only if Up + Us ≤ 1.

The isolation property allows us to use a bandwidth reservation strategy to allocate a fraction of the CPU time to soft tasks whose computation time cannot be easily bounded. The most important consequence of this result is that soft tasks can be scheduled together with hard tasks without affecting the a priori guarantee, even in the case in which the execution times of the soft tasks are not known or the soft requests exceed the expected load. In addition to the isolation property, the CBS has the following characteristics.

The CBS behaves as a plain EDF algorithm if the served task τi has parameters (Ci, Ti) such that Ci ≤ Qs and Ti = Ts. This is formally stated by the lemma which stated, a hard task τi with parameters (Ci, Ti) is schedulable by a CBS with parameters Qs ≥ Ci and Ts = Ti if and only if τi is schedulable with EDF. It is proven by for any job of a hard task we have that ri,j+1 − ri,j ≥ Ti and ci,j ≤ Qs. Hence, by definition of the CBS, each hard job is assigned a deadline di,j = ri,j+ Ti and it is scheduled with a budget Qs ≥ Ci. Moreover, since ci,j ≤ Qs, each job finishes no later than the budget is exhausted; hence the deadline assigned to a job is never postponed and is exactly the same as the one used by EDF.

The CBS automatically reclaims any spare time caused by early completions. This is due to the fact that whenever the budget is exhausted, it is always immediately replenished at its full value and the server deadline is postponed. In this way, the server remains eligible, and the budget can be exploited by the pending requests with the current deadline. This is the main difference with respect to the processor capacity reserves proposed by Mercer et al. [MST93, MST94a]. Knowing the statistical distribution of the computation time of a task served by a CBS, it is possible to perform a QoS guarantee based on probabilistic deadlines (expressed in terms of probability for each served job to meet a deadline).

## Simulation Results Comparing TBS and CBS

This section shows how the CBS can be efficiently used as a service mechanism for improving responsiveness of soft aperiodic requests. Its performance has been tested against that of TBS and DSS, by measuring the mean tardiness experienced by soft task:

Ei,j = max{0, fi,j − di,j},

where fi,j is the finishing time of job Ji,j .

Such a metric was selected because in many soft real-time applications for example multimedia, meeting all soft deadlines is either impossible or very inefficient; hence, the system should be designed to guarantee all the hard tasks and minimize the mean time that soft tasks execute after their deadlines. All the simulations presented in this section have been conducted on a hybrid task set consisting of 5 periodic hard tasks with fixed parameters and 5 soft tasks with variable execution times and interarrival times.

A picture containing graphical user interface

Description automatically generated

1. Result on comparing TBS with CBS.

This experiment compares the mean tardiness experienced by soft tasks when they are served by a CBS and TBS. In this test, the utilization factor of periodic hard tasks was Uhard = 0.7. As we can see, TBS slightly outperforms CBS, but does not protect hard tasks from transient overruns that may occur in the soft activities. Note that since the CBS automatically reclaims any available idle time coming from early completions. The advantage of the CBS over the TBS can be appreciated when WCETi >> ci,j . In this case, in fact, the TBS can cause an underutilization of the processor, due to its worst-case assumptions.

## Implementation In UPPAAL

##### Conclusion

Short to say, I have learned a lot through this research study that has been conducted. Not only I got the chance to learn on the topic that is assigned to me, but I also earned some knowledges on a few other topics, for example, EDL and also TBS which I could say the precedent algorithm in a timeline or a step of improvement before coming to CBS existence. It is regarded as an example or guide to be considered in this topic for a clearer understanding. Apart from that, this research study gives me an opportunity for me to strengthen my knowledge on Real Time System which I believe one of the most critical parts and important to comprehend in pursuing my course of study.

##### References

1. C. W. Mercer, S. Savage, and H. Tokuda, “Processor capacity reserves for multimedia operating systems,” in Proceedings of IEEE international conference on Multimedia Computing and System, May 1994.
2. Abeni, L. and Buttazzo, G., 2004. Resource Reservation in Dynamic Real-Time Systems. *Real-Time Systems*, 27(2), pp.123-167.
3. Kopetz, H., 2004. *Real-time systems*. Boston, Mass.: Kluwer Acad.
4. Biondi, A., Melani, A. and Bertogna, M., 2022. *Hard Constant Bandwidth Server: Comprehensive formulation and critical scenarios*. [online] Ieeexplore.ieee.org. Available at: <https://ieeexplore.ieee.org/document/6871182/> [Accessed 10 May 2022].
5. Silly, M., 1995. A Dynamic Scheduling Algorithm for Sporadic Tasks under Resource and Timing Constraints. *IFAC Proceedings Volumes*, 28(5), pp.247-252.
6. Buttazzo, G., 2011. *Hard real-time computing systems*. New York: Springer.
7. Abeni, L., Lipari, G. and Lelli, J., 2015. Constant bandwidth server revisited. *ACM SIGBED Review*, 11(4), pp.19-24.
8. Spuri, M. and Buttazzo, G., 1994. *Efficient aperiodic service under earliest deadline scheduling*. [online] Ieeexplore.ieee.org. Available at: <https://ieeexplore.ieee.org/document/342735/> [Accessed 7 May 2022].
9. Isovic, D. and Fohler, G., n.d. *Efficient scheduling of sporadic, aperiodic, and periodic tasks with complex constraints*. [online] Ieeexplore.ieee.org. Available at: <https://ieeexplore.ieee.org/document/896010/> [Accessed 5 May 2022].