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 order to present these results, we need to analyse and elaborate some basic concepts that will be used throughout this paper.

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 real-time computing systems running multiple concurrent tasks, a fundamental property that has to 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 [1] represents the most powerful scheduling mechanism specifically conceived to achieve such a property. 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, audio/video streaming, etc. 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.

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 [2], [3]. 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 [4], [5], 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. Resource Reservation is typically implemented by assigning to each application a dedicated real-time server, called 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 [HCBS]

INTRODUCTION

Many complex control applications include tasks which have to be completed within strict time constraints, called 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, tasks that require regular activations are called periodic, whereas tasks which have irregular arrival times are called aperiodic.

The basic idea behind the CBS mechanism can be explained by using this analogy: 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 (i.e., 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 [1].

The Constant Bandwidth Server (CBS) is an algorithm for providing temporal protection and real-time guarantees to real-time sporadic tasks. Recently, an implementation of this algorithm called SCHED\_DEADLINE has been included in the Linux kernel. Therefore, the CBS algorithm is now used to serve more generic tasks than do not obey to the classical sporadic task model. One important type of tasks which was not considered by the original CBS algorithm is the so called "self-suspending task model", where a task instance can suspend itself waiting for an external event. Even if the original algorithm is adapted so that the temporal protection property continues to hold, it is difficult for developers to provide guarantees and to select the most appropriate server parameters for such tasks. This paper investigates the problem of using the CBS algorithm for serving self-suspending tasks, by analysing it from a theoretical point of view and showing how to select the server parameters (budget and periods) for self-suspending tasks. Finally, the effectiveness of these proposals is shown through both simulations and real experiments on Linux / SCHED\_DEADLINE.

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.

Reference [Book]

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 (i.e., 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 (deadlinebased) 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 .

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.

DEFINITION OF CBS

The CBS can be defined as follows:

-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 (e.g., 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.

SCHEDULING EXAMPLE

Chart

Description automatically generated

Figure 6.14 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).

FORMAL DEFINITION

In order to provide a formal definition of the CBS, let a k and dk be the release time and the deadline of the kth chunk generated by the server, and let c and n be the actual server budget and the number of pending requests in the server queue (including the request currently being served). These variables are initialized as follows:

d0 = 0, c = 0, n = 0, k = 0

Using this notation, the server behavior can be described by the algorithm shown below:

Text

Description automatically generated

CBS PROPERTIES

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. See the original work by Abeni and Buttazzo [AB98] for the proof.

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).

ADVANTAGES OF CBS

**Modifications for improvement from TBS**

One major problem of the TBS and TB\* algorithms is that they do not use a server budget for controlling aperiodic execution, but rely on the knowledge of the worstcase computation time specified by each job at its arrival. When such a knowledge is not available, not reliable, or too pessimistic (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.