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.

INTRODUCTION

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.

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.