SHRUB: Shared Reclamation of Unused Bandwidth

Scuola Superiore Sant'Anna
RETIS Lab
Technical Report

Sanjoy Baruah
University of North Carolina at Chapel Hill
baruah@cs.unc.edu

Giuseppe Lipari
Scuola Superiore Sant'Anna, Pisa
g.lipari@sssup.it

Luca Abeni
UniversitÃ di Trento
luca.abeni@dit,unitn.it

July 4, 2008

1 System model and definitions

1.1 Task Model

A real-time task τ_i is a stream of jobs $J_{i,j}$. Each job $J_{i,j}$ arrives (becomes executable) at time $r_{i,j}$, and finishes at time $f_{i,j}$ after executing for a time $c_{i,j}$. Each job $J_{i,j}$ is characterized by a deadline $d_{i,j}$ that is respected if $f_{i,j} \leq d_{i,j}$, and is missed if $f_{i,j} > d_{i,j}$.

A task is said to be periodic if $r_{i,j+1} = r_{i,j} + T_i$, where T_i is the *task period*. A task is *sporadic* if $r_{i,j+1} \geq r_{i,j} + T_i$, and in this case T_i is the *minimum interarrival time*. In both cases $d_{i,j} = r_{i,j} + D_i$, where D_i is the task's relative deadline.

For a hard real-time task even a single deadline violation is considered as a severe fault. In contrast, for a *soft* real-time task a few deadline violations are deemed acceptable provided that the anomaly is kept in check. In this case, reasonable QoS metrics can be are related to probability of missing a deadline, or to similar measurements.

1.2 Reservation model

A reservation is a *scheduling entity*. It can be attached to a task. The scheduler handles reservations in its internal queues. Whenever the scheduler selects a reservation for execution (*dispatch*), the corresponding task is executed.

A reservation S_i is identified by:

- a period P_i;
- a capacity (or *maximum budget*) Q_i, $0 < Q_i \leq P_i$;
- a weight w_i (optional);
- a priority p_i (optional).

We define the utilization (or *bandwidth*) of reservation S_i as $B_i = \frac{Q_i}{P_i}$.

There are many algorithms that implement the concept of reservation, depending on the scheduler that is used. For fixed priority schedulers, the Sporadic Server [SSL89] (that is now part of the POSIX standard) and the Deferrable Server [SSL89] are the most popular.

For the EDF schedulers, we have the Dynamic Sporadic Server [SB94], the Constant Bandwidth Server (CBS) [AB98], the GRUB server [GB00], and many others.

Reservations can also be used to build hierarchies of schedulers. In such a case, a reservation can be associated a set of tasks (or reservations) and a *local scheduler*. When the global scheler selects the reservation to execute, the local scheduler is executed which in turn selects one of the associated tasks (reservations).

In this report we concentrate on the global scheduling of reservations on a single processor system based on EDF.

2 Reservation algorithms

2.1 CBS

The Constant Bandwidth Server (CBS) [AB98] implements reservations by using the concept of *scheduling deadlines*: each task τ_i is handled by a dedicated CBS, which uses a dynamic scheduling deadline d_i^s to set its priority in the EDF scheduler. When a new job $J_{i,j}$ arrives, the reservation checks whether it can be scheduled using the last assigned deadline, otherwise the request is assigned an initial deadline equal to $r_{i,j} + P_i$. Each time the job executes for Q_i time units (i.e., its budget is depleted), its scheduling deadline is postponed by P_i. Thereby, the task is prevented from executing for more than Q_i units with the same deadline. This behaviour is achieved by using a budget q_i which is decreased while the served task executes: more formally, if a task τ_i served by a CBS (Q_i, P_i) executes for a time δ, the budget is updated as $q_i = q_i - \delta$ (this is the CBS **accounting rule**).

It can be shown (see [AB98] for a complete description of the algorithm and its properties) that if $\sum_i B_i \leq U^{lub}$ (with $U^{lub} = 1$ for EDF), then each task τ_i is reserved fraction $B_i = Q_i/T_i$ of the CPU time regardless of the behaviour of the other tasks. This property is called *temporal isolation* (meaning that the worst case schedule of a task is not affected by the temporal behaviour of the other tasks running in the system).

3 Shrub

The average-case performance of a reservation-based scheduler can be greatly improved by using a proper reclaiming policy. In this paper, we present SHRUB (SHared Reclamation of Unused Bandwidth), a new reclaiming mechanisms for reservation-based schedulers based on EDF. Unlike other reclaiming policies, SHRUB allows to precisely control the share of spare bandwidth to allocate to needing tasks by using weights.

The SHRUB algorithm is a variant of the GRUB reclaiming mechanism [GB00], which in turn uses the CBS as a basic scheduler. While a "traditional" reservation is described by a couple (Q_i, P_i), in SHRUB each reservation $RSV_i = (Q_i, P_i, w_i)$ is associated an additional *weight* w_i, which is used to distribute the reclaimed CPU time.

3.1 Basic intuition

Every reservation maintains the following internal variables to implement algorithm SHRUB:

- the remaining capacity $q_i > 0$. Notice that, due to the reclamation properties of SHRUB, at some point in time it may happen that $q_i > Q_i$.
- the reservation deadline d_i, which is used to schedule the reservations according to EDF;
- the reservation *state* (the states are the same as in GRUB).

Reminder: a reservation that has completed its jobs, and has a residual budget of q_i, becomes idle at time $d_i - \frac{q_i}{U_i}$.

Since the reservations are scheduled by EDF, to guarantee the algorithm's properties, the sum of the utilizations of all reservation must not exceed 1:

$$\sum_{i=1}^{n} U_i \leq 1.$$

We further denote by $ACT(t)$ the set of reservations that are active at time t (i.e. not idle); by $U_A(t)$ the total utilization of all active reservations at time t; $U_F(t) = 1 - U_A(t)$.

Initially, we consider that all reservations update their variables according to the CBS algorithm. Suppose a reservation S_i executes in interval $[t_1, t_2]$, with $\Delta t = t_2 - t_1$. Suppose that the amount of free bandwith in the interval is constant and equal to $U_F(t) = U_F(t_1) = 1 - U_A(t_1)$. For the sake of simplicity, in the following we drop the t parameter.

Now suppose we insert in the system a job with the following parameters:

- arrival time $a = t_1$;
- absolute deadline $d = t_2$;
- computation time $c = U_F \Delta t$.

It is easy to prove that, by inserting such a job, the system remains schedulable (it can be done by using the processor demand bound analysis). It then follows that in interval $[t_1, t_2]$ there is an extra budget equal to $c = U_F \Delta t$ that can be used by the active reservations.

In the GRUB algorithm this extra budget is entirely given to the executing reservation S_i. Therefore, in GRUB the reservation budget is decreased as follows:

$$\Delta q_i = -\Delta t + c = (1 - U_F)\Delta t = -U_A \Delta t \tag{1}$$

Notice that the budget of the executing reservation is decreased at a rate that is proportional to the current reserved bandwidth in the system. If $B_{act} < 1$, this is equivalent to temporarily increase the maximum budget of the reservation for the current period. In the limit case of a fully utilized system, $B_{act} = 1$ and the budget is decreased as in the CBS accounting rule. In the opposite limit case of only one active reservation, the budget is decreased at a rate B_i.

Here we propose to distribute the extra budget to all active reservations in proportion to their relative weigth. For simplicity, we denote by $W_A(t)$ the sum of all the weigths of the active reservations:

$$W_A(t) = \sum_{S_j \in ACT(t)} w_j \tag{2}$$

We assume that the set $ACT(t)$ does not change in $[t_1, t_2]$, so W_A is constant in the interval.

In interval $[t_1, t_2]$, each active reservation S_i will receive an extra budget equal to $U_F \Delta t \frac{w_i}{W_A}$. In addition, the budget of the executing reservation will be decreased by Δt. Therefore, all the budgets of the active reservations will be updated as follows:

$$q_i = \begin{cases} \left(-1 + U_F \frac{w_i}{W_A}\right) \Delta t & \text{if } S_i \text{ executes} \\ U_F \frac{w_i}{W_A} \Delta t & \text{if } S_i \text{ does not execute} \end{cases} \quad (3)$$

By doing the substitution $V_i = d_i - \frac{q_i}{U_i}$, it follows that $dV_i = -\frac{dq_i}{U_i}$. By rewriting the equation with derivatives, we have

$$dV_i = \begin{cases} \left(1 - U_F(t) \frac{w_i}{W_A(t)}\right) dt & \text{if } S_i \text{ executes in } t \\ -U_F(t) \frac{w_i}{W_A(t)} dt & \text{if } S_i \text{ does not execute in } t \end{cases} \quad (4)$$

Therefore, if a reservation S_i becomes active at time t_a and becomes idle at time $t_b > t_a$, for any t, $t_a \leq t \leq t_b$:

$$V_i(t) = \int_{t_a}^{t} dV_i = \int_{t_a}^{t} \sigma_i(x) - U_F(x) \frac{w_i}{W_A(x)} dx \quad (5)$$

where $\sigma_i(x) = 1$ if S_i executes in t, otherwise it is 0.

As you can see, there is one component that depends on the reservation, and one that is common to all reservations. We can rewrite it in the following way:

$$V_i(t) = \int_{t_a}^{t} \sigma_i(x) dx - w_i \int_{t_a}^{t} \frac{U_F(x)}{W_A(x)} dx \quad (6)$$

The second integral is common to all reservations, so it can be computed globally.

In the two limit cases (fully utilized system and only one reservation), rule 1 and 3 are identical. However, when there are many reservations in the system and there is some unused CPU time, SHRUB effectively distributes such time to all needing reservations in proportion to their weights. Unlike GRUB, SHRUB uses the weights to assign more spare bandwidth to reservations with higher weights.

4 Efficient implementation

It looks like the algorithms is $O(n)$, because the budgets of all active reservations must be updated at every event. Actually, this is not necessary. We introduce an additional global variable in the system:

$$V_g(t) = \int_{0}^{t} \frac{U_F(x)}{W_A(x)} dx \quad (7)$$

Now, the virtual time of every reservation can be updated as follows. Let L_i be the last instant at which the virtual time of reservation S_i has been updated. Suppose that the reservation is active but does not execute until $t > L_i$, then

$$V_i(t) = V_i(L_i) - w_i(V_g(t) - V_g(L_i)) \quad (8)$$

Suppose instead that reservation S_i starts executing in L_i and it executes until time t. Then:

$$V_i(t) = V(L_i) + t - L_i - w_i(V_g(t) - V_g(L_i)) \tag{9}$$

For every reservation, we maintain an additional helper variable $V_g(L_i)$. Then we only need to update the reservation variables at all events that are relevant to the reservation itself, and precisely:

- when the reservation becomes active $V_i = t$;
- when the reservation starts executing;
- when the reservation is suspended;
- when the reservation becomes idle.

A similar reasoning can be done by using the budgets instead of the virtual times.

By implementing the algorithm this way, at every event it is inly necessary to update a constant number of variables. Hence, the algorithm has complexity $O(1)$ per each event.

According to the complexity convention that is used in network research, the total complexity of Algorithm SHRUB is $O(n)$, because there can be up to n events (one per reservation) in any infinitesimal small interval of time.

References

[AB98] Luca Abeni and Giorgio Buttazzo. Integrating multimedia applications in hard real-time systems. In *Proceedings of the IEEE Real-Time Systems Symposium*, Madrid, Spain, December 1998.

[GB00] G.Lipari and S.K. Baruah. Greedy reclaimation of unused bandwidth in constant bandwidth servers. In *IEEE Proceedings of the 12th Euromicro Conference on Real-Time Systems*, Stokholm, Sweden, June 2000.

[SB94] M. Spuri and G. C. Buttazzo. Efficient aperiodic service under the earliest deadline scheduling. In *Proceedings of the IEEE Real-Time Systems Symposium*, December 1994.

[SSL89] B. Sprunt, L. Sha, and J. P. Lehoczky. Aperiodic scheduling for hard real-time systems. *The Journal of Real-Time Systems*, 1989.