IPP HURRAY!

www.hurray.isep.ipp.pt

Technical Report

Handling Shared Resources and Precedence Constraints in Open Systems

Luís Nogueira

Luís Miguel Pinho

TR-070704

Version: 1.0

Date: July 2007

Handling Shared Resources and Precedence Constraints in Open Systems

Luís Nogueira, Luís Miguel Pinho

IPP-HURRAY!

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8340509

E-mail: {luis, lpinho}i@dei.sep.ipp.pt

http://www.hurray.isep.ipp.pt

Abstract

There is an increasing demand for highly dynamic realtime systems where several independently developed applications with different timing requirements can coexist. This paper proposes a protocol to integrate shared resources and precedence constraints among tasks in such systems assuming no precise information on critical sections and computation times is available. The concept of bandwidth inheritance is combined with a capacity sharing and stealing mechanism to efficiently exchange bandwidth among needed tasks, minimising the cost of blocking.

Handling Shared Resources and Precedence Constraints in Open Systems

Luís Nogueira, Luís Miguel Pinho
IPP Hurray Research Group
Polytechnic Institute of Porto, Portugal
{luis,lpinho}@dei.isep.ipp.pt

Abstract

There is an increasing demand for highly dynamic real-time systems where several independently developed applications with different timing requirements can coexist. This paper proposes a protocol to integrate shared resources and precedence constraints among tasks in such systems assuming no precise information on critical sections and computation times is available. The concept of bandwidth inheritance is combined with a capacity sharing and stealing mechanism to efficiently exchange bandwidth among needed tasks, minimising the cost of blocking.

1 Introduction

A common approach to limit the effects of overruns is based on a resource reservation approach, assigning a fraction of the available resources and handling tasks through dedicated servers, preventing the served tasks from demanding more than the reserved amount [1, 13, 4, 15, 12].

However, a more flexible overload control in highly dynamic open real-time systems, where is not possible to have a precise knowledge of how many services will need to be concurrently executed neither which resources will be accessed and by how long they will be held, can be achieved with the combination of guaranteed and best-effort servers and reducing isolation in a controlled fashion in order to donate reserved, but still unused, capacities to currently overloaded servers. The Capacity Sharing and Stealing (CSS) algorithm [16] was proposed to efficiently handle soft-tasks' overloads by making additional capacity available from two sources: (i) reclaiming unused allocated capacity when jobs complete in less than their budgeted execution time; and (ii) stealing allocated capacities to non-isolated servers used to schedule best-effort jobs.

This paper relaxes the common assumption that all tasks are independent and handles access to shared resources and precedence constraints among services' tasks, enhancing CSS with the ability to work in more general real world scenarios. We propose a simple protocol that integrates the concept of bandwidth inheritance [11] with the efficient capacity sharing and stealing mechanism of CSS to mitigate the cost of blocking. Preliminary results suggest that without introducing any significant overhead in the original CSS scheduler, the protocol outperforms currently available solutions to handle shared resources in open systems and introduces a new method to handle precedence constraints without an offline knowledge of tasks' execution times.

2 System model

Each task generates a virtually infinite sequence of jobs. The j^{th} job of task τ_i arrives at time $a_{i,j}$, is released to the ready queue at time $r_{i,j}$, and starts to be executed at time $s_{i,j}$ with deadline $d_{i,j} = a_{i,j} + T_i$, where T_i is the period of τ_i. The arrival time of a particular job is only revealed during execution, and the exact execution requirements $e_{i,j}$ and which resources will be accessed and by how long will be held can only be determined by actually executing the job to completion until time $f_{i,j}$. These times are characterised by the relations $a_{i,j} \leq r_{i,j} \leq s_{i,j} \leq f_{i,j}$.

Jobs may simultaneously need exclusive access to one or more of the system's resources, during part or all of their executions. If task τ_i is using resource R_i, it locks that resource. Since no other task can access R_i until it is released by τ_i, if τ_j tries to access R_i it will be blocked by τ_i. Blocking can also be indirect (or transitive) if although two tasks do not share any resource, one of them may still be indirectly blocked by the other through a third task.

A task τ_i is said to precede another task τ_k if τ_k cannot start until τ_i is finished. Such a precedence relation is formalised as $\tau_i \prec \tau_k$ and is guaranteed if $f_{i,j} \leq s_{k,j}$. Precedence constraints are defined in the service's description at admission time by a direct graph G, where each node represents a task and each directed arc represents a precedence constraint $\tau_i \prec \tau_k$ between two tasks τ_i and τ_k. Given a partial order $\prec$ on the tasks, the release times and the deadlines are said to be consistent with the partial order if $\tau_i \prec \tau_k \Rightarrow r_{i,j} \leq r_{k,j}$ and $d_{i,j} < d_{k,j}$.

Each task τ_i is assigned to a server S_i that is characterised by a pair (Q_i, T_i), where Q_i is the maximum reserved capacity and T_i is the server period. Each server S_i records its current available capacity c_i, its deadline d_i and a specific recharging time r_i, implementing a hard reservation [17]. As such, capacity replenishment explicitly occurs only at time r_i and not on every capacity exhaustion.

CSS allows to serve tasks with different isolation guarantees. *Isolated* servers have a guaranteed execution capacity every period that is independent of the behaviour of other servers. On the other hand, inactive *non-isolated* servers can have some or all of its reserved capacity stolen by active overloaded servers. Non-isolated servers were proposed to support the increasing use of of imprecise computation models, anytime algorithms, and best-effort services in dynamic open real-time systems.

A server S_i is active if (i) its served task is ready to execute; (ii) it is executing; (iii) or it is supplying its residual capacity to other servers until its deadline. S_i is inactive when the (i) it has no pending jobs and no available capacity. Note that on an early completion of its current job, a server remains active supplying its residual capacity until its deadline, contributing to the global system's activity.

CSS's dynamic budget accounting mechanism considers residual capacities, generated by early completions, as well as inactive non-isolated capacities, as a common resource that can be shared by active servers. The server to which the budget accounting is going to be performed is dynamically determined at the time instant when a capacity is needed. A server starts to greedily reclaim residual capacities with an earlier deadline than the one assigned to its current job, then consumes its own reserved capacity, and finally steals inactive non-isolated capacities while executing job. The used capacity is correctly decremented from the reserved capacity of the pointed server.

3 Sharing resources in open systems

A great amount of work has been addressed to minimise the adverse effects of blocking when considering shared resources among tasks. Resource sharing protocols such as the Priority Ceiling Protocol [19], Dynamic Priority Ceiling [7], and Stack Resource Policy [2] have been proposed to provide guarantees to hard real-time tasks accessing mutually exclusive resources. Solutions based on these protocols were already proposed [10, 6, 5, 3] but require a prior knowledge of the maximum resource usage for each task and cannot be directly applied to dynamic open systems.

The Bandwidth Inheritance (BWI) [11] protocol allows resource sharing between tasks without requiring any prior knowledge about the tasks' structure and temporal behaviour by extending CBS [1] to work in the presence of shared resources, adopting the Priority Inheritance Protocol (PIP) [19] to handle task blocking. The main drawback of BWI is allow a blocking task to consume other servers' capacity without any later compensation. As a consequence, blocking tasks can consume more than their allocated capacities, while blocked tasks only get the remaining capacities of their servers (if any) not used to execute blocking tasks. This violation of the original capacity distribution can have a huge negative impact in the overall system's performance.

Improvements to BWI were already proposed in BWE [22] and CFA [18] to compensate the extra work on blocked servers. Both algorithms try to fairly compensate blocked servers in exactly the same amount of capacity that was consumed by a blocking task while executing in a blocked server. To achieve this, BWI maintains a global $n*n$ matrix (n is the number of servers in the system) in order to record the amount of budget that should be exchanged between servers, a budget list at each server to keep track of available budgets, and dynamically manages resource groups at each blocking and releasing of a shared resource. With CFA, each server manages to task lists with different priorities and a counter that keeps track of the amount of borrowed capacity from a higher priority server. Contracted debts are payed by blocking servers, until the blocked servers' counter is successively decremented to zero

This paper follows a different approach and proposes to exchange capacities between all the system's servers by merging the benefits of a smart greedy capacity reclaiming with the concept of bandwidth inheritance. The increased computational complexity of trying to fairly assign consumed capacities to the servers a task blocked during execution and the fact that CSS tends to fairly distribute residual capacities in the long run, lead us to propose to statistically achieve a fair bandwidth compensation through an efficient greedy capacity reclaiming policy. Adding to the lower complexity of our approach, preliminary results demonstrate that taking advantage of all available residual capacity instead of only exchanging budgets within the same resource group leads to a better system's performance.

In the proposed protocol, every server maintains a list of served tasks. Initially, each server S_i has only its dedicated task T_i and, as long as no task is blocked, servers behave as in the original CSS scheduler. Whenever a task finishes, the remaining capacity of its dedicated server is released as residual capacity and it is immediately available to other servers. The next running server greedily consumes available residual capacities with earlier deadlines than the one assigned to its current job before consuming its own reserved capacity. With blocking, the following rules are introduced.

1. **Rule A:** When a high priority task τ_i is blocked by a lower priority task τ_j when accessing a resource R, τ_j inherits server S_i. The execution time of τ_j is now

accounted to the currently pointed server by S_i. If task τ_j has not yet released the shared resource R when S_i exhausts all the capacity it can use, τ_j continues to be executed by the earliest deadline server with available capacity that needs to access R, until τ_j releases R.

2. **Rule B:** If at time t, no active server with pending jobs can continue to execute by reclaiming residual capacities or consuming its own reserved capacity, and there is at least one active server S_r with residual capacity greater than zero, available residual capacities with deadlines greater than the one assigned to the current job $j_{p,k}$ of the earliest deadline server S_p with pending work can be used to execute $j_{p,k}$ through bandwidth inheritance.

Rule A describes the integration of bandwidth inheritance in the dynamic budget accounting of CSS. Rule B maximises the amount of capacity that can be exchanged between hard reservation servers. Bandwidth inheritance can be used by any active server to execute unfinished tasks, including those from servers that do not directly or indirectly share any resource with the selected server, if at a particular time no active server in the system is able to reclaim new residual capacities or steal inactive non-isolated capacities to continue executing its pending work after a capacity exhaustion. Rule B is easily incorporated in CSS with a minimum overhead. Since the queue of active servers is ordered by deadlines, when CSS is traversing it to select the next running server, it keeps track of the earliest deadline server with pending work and no capacity left S_p as well as the earliest deadline server with available residual capacity S_r. If the end of the queue of active servers is reached without finding a server with pending work and available capacity, S_r is selected as the running server. S_r then inherits the first task of S_p' list and executes it consuming its own residual capacity. Since in CSS a server always starts to consume the earliest residual capacity available, no modification to the budget accounting mechanism is needed to correctly account for the consumed capacity.

Note that CSS ensures that residual capacities originated by earlier completions can be reclaimed by any active eligible server. Blocked servers can then take advantage of any residual capacity, even if it is released by a server that does not share any resource with the reclaiming server. Although a greedy reclamation of residual capacities is used, excess capacity always tends to be exchanged in a fair manner among needed servers across the time line [13, 16].

4 Precedence constraints in open systems

It is well known that precedence constraints can be guaranteed in real-time scheduling by priority assignment. With dynamic scheduling, any task will always precede any other task with a later deadline. This suggests that precedence constraints that are consistent with the tasks' deadlines do not affect the schedulability of the task set. In fact, the idea behind the consistency with the partial order is to enforce a precedence constraint by using an earlier deadline.

Formal work exists, showing how to modify deadlines in a consistent manner so that EDF can be used without violating the precedence constraints. Garey et al. [9] show that the consistency of release times and deadlines can be used to integrate precedence constraints in the task model. Spuri and Stankovic [21] introduce the concept of quasi-normality to give more freedom to the scheduler so that it can also obey shared resource constraints, and provide sufficient conditions for schedules to obey a given precedence graph. The authors prove that with deadline modification and some type of inheritance it is possible to integrate precedence constraints and shared resources. Mangeruca et al. [14] consider situations where the precedence constraints are not all consistent with the tasks' deadlines and show how schedulability can be recovered by considering a constrained scheduling problem based on a more general class of precedence constraint.

However, all these works are based on a previous knowledge of tasks' execution times. To make use of those previous results in open systems, we have to enforce the consistency of release times and deadlines with the partial order at admission time using a technique similar to those which have already appeared in several works [9, 20, 14, 8], but considering average execution times and handle overloads of the precedent tasks at runtime.

Considering average execution times implies to handle situations where tasks require more than the declared capacity and their current servers cannot use any available capacity to complete their works, disabling their successors to start. The capacity exchange mechanism proposed in the previous section can be used to handle blocking due to precedence violations in the same way as for a critical section blocking, minimising the impact of misbehaved tasks on the overall system's performance. We base our approach on the idea that if task $\tau_j \prec \tau_i$ has not yet finished at time $s_{i,k}$, when the k^{th} instance of τ_i is selected to execute, it is blocking its successor.

The problem of knowing if the set of precedent tasks has already finished is easily solved with CSS. A server that has completed its job only remains in the Active state until its deadline if it is supplying residual capacity to the other servers. As such, if the dedicated server of a task $\tau_j \prec \tau_i$ is still active, S_i must only check the current value of S_j's residual capacity. If its equal to zero, then τ_j has not yet been completed at time s_i. Otherwise, the job has already been completed and the server is supplying residual capacity. Since precedence constraints are imposed by earlier deadlines, the dynamic budget accounting of CSS ensures that no additional verifications are needed than the ones already being performed.

5 Conclusions and ongoing work

This paper combines bandwidth inheritance with a greedy capacity sharing and stealing mechanism to efficiently handle shared resources and precedence constraints among tasks in dynamic open real-time systems. The approach allows a server to reclaim residual capacities allocated but unused when jobs complete in less than their budgeted execution time and to steal capacity from inactive non-isolated servers used to schedule best-effort jobs to mitigate the costs of blocking.

The protocol is currently being validated through extensive simulations in highly dynamic scenarios. Preliminary results clearly justify the use of a capacity exchange mechanism that reclaims as much residual capacity as possible and does not restrict itself to exchange capacities only within a resource sharing group.

A theoretical validation of the protocol is currently under development, detailing the conditions under which it is possible to guarantee hard real-time tasks.

Acknowledgements

This work was partly supported by FCT, through the CISTER Research Unit (FCT UI 608) and the Reflect project (POSC/EIA/60797/2004), and the European Comission through the ARTIST2 NoE (IST-2001-34820).

References

[1] L. Abeni and G. Buttazzo. Integrating multimedia applications in hard real-time systems. In *Proceedings of the 19th IEEE RTSS*, page 4, Madrid, Spain, December 1998.

[2] T. P. Baker. A stack-based resource allocation policy for realtime processes. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 191–200, Lake Buena Vista, Florida, USA, December 1990.

[3] S. K. Baruah. Resource sharing in edf-scheduled systems: A closer look. In *Proceedings of the 27th IEEE Real-Time Systems Symposium*, pages 379–387, Rio de Janeiro,Brazil, December 2006.

[4] M. Caccamo, G. Buttazzo, and L. Sha. Capacity sharing for overrun control. In *Proceedings of 21th IEEE RTSS*, pages 295–304, Orlando, Florida, 2000.

[5] M. Caccamo, G. C. Buttazzo, and D. C. Thomas. Efficient reclaiming in reservation-based real-time systems with variable execution times. *IEEE Transactions on Computers*, 54(2):198–213, February 2005.

[6] M. Caccamo and L. Sha. Aperiodic servers with resource constraints. In *Proceedings of the 22nd IEEE Real-Time Systems Symposium*, pages 161–170, London, UK, December 2001.

[7] M.-I. Chen and K.-J. Lin. Dynamic priority ceilings: a concurrency control protocol for real-time systems. *Real-Time Systems*, 2(4):325–346, 1990.

[8] H. Chetto, M. Silly, and T. Bouchentouf. Dynamic scheduling of real-time tasks under precedence constraints. *Real-Time Systems*, 2(3):181–194, 1990.

[9] M. R. Garey, D. S. Johnson, B. B. Simons, and R. E. Tarjan. Scheduling unit-time tasks with arbitrary release times and deadlines. *SIAM Journal on Computing*, 10(2):256–269, May 1981.

[10] K. Jeffay. Scheduling sporadic tasks with shared resources in hard-real-time systems. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 89–99, Phoenix, Arizona, USA, December 1992.

[11] G. Lamastra, G. Lipari, and L. Abeni. A bandwidth inheritance algorithm for real-time task synchronization in open systems. In *Proceedings of the 22nd IEEE Real-Time Systems Symposium*, pages 151–160, London, UK, December 2001.

[12] C. Lin and S. A. Brandt. Improving soft real-time performance through better slack reclaiming. In *Proceedings of the 26th IEEE RTSS*, pages 410–421, 2005.

[13] G. Lipari and S. Baruah. Greedy reclamation of unused bandwidth in constant-bandwidth servers. In *Proceedings of the 12th ECRTS*, pages 193–200, Stockholm, Sweden, 2000.

[14] L. Mangeruca, A. Ferrari, and A. L. Sangiovanni-Vincentelli. Uniprocessor scheduling under precedence constraints. In *Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 157–166, San Jose, CA, USA, April 2006.

[15] L. Marzario, G. Lipari, P. Balbastre, and A. Crespo. Iris: A new reclaiming algorithm for server-based real-time systems. In *Proceedings of the 10th IEEE RTAS*, page 211, Toronto, Canada, 2004.

[16] L. Nogueira and L. M. Pinho. Capacity sharing and stealing in server-based real-time systems. In *Proceedings of the 21th IEEE International Parallel and Distributed Processing Symposium (to appear)*, Long Beach,CA,USA, March 2007.

[17] R. Rajkumar, K. Juvva, A. Molano, , and S. Oikawa. Resource kernels: A resource-centric approach to real-time and multimedia systems. In *Proceedings of the SPIE/ACM Conference on Multimedia Computing and Networking*, 1998.

[18] R. Santos, G. Lipari, and J. Santos. Scheduling open dynamic systems: The clearing fund algorithm. In *Proceedings of the 10th International Conference on Real-Time and Embedded Computing Systems and Applications*, pages 114–129, Gothenburg, Sweden, August 2004.

[19] L. Sha, R. Rajkumar, and J. P. Lehoczky. Priority inheritance protocols: an approach to real-time synchronisation. *IEEE Transaction on Computers*, 39(9):1175–1185, 1990.

[20] M. Spuri and G. Buttazzo. Efficient aperiodic service under earliest deadline scheduling. In *Proceedings of the 15th IEEE RTSS*, pages 2–11, San Juan, Puerto Rico, 1994.

[21] M. Spuri and J. A. Stankovic. How to integrate precedence constraints and shared resources in real-time scheduling. *IEEE Transactions on Computers*, 43(12):1407–1412, 1994.

[22] S. Wang, K.-J. Lin, and S. Peng. Bwe: A resource sharing protocol for multimedia systems with bandwidth reservation. In *Proceedings of the 4th IEEE International Symposium on Multimedia Software Engineering*, pages 158–165, New-port Beach,CA,USA, December 2002.