

# Distributed Priority Inheritance for Real-Time and Embedded Systems<sup>\*</sup>

César Sánchez<sup>1</sup>, Henny B. Sipma<sup>1</sup>,  
Christopher D. Gill<sup>2</sup>, and Zohar Manna<sup>1</sup>

<sup>1</sup> Computer Science Department  
Stanford University, Stanford, CA 94305, USA  
{cesar, sipma, manna}@CS.Stanford.EDU

<sup>2</sup> Department of Computer Science and Engineering  
Washington University, St. Louis, MO 63130, USA  
cdgill@CSE.wustl.EDU

**Abstract.** We study the problem of priority inversion in distributed real-time and embedded systems and propose a solution based on a distributed version of the priority inheritance protocol (PIP). Previous approaches to priority inversions in distributed systems use variations of the priority ceiling protocol (PCP), originally designed for centralized systems as a modification of PIP that also prevents deadlock. PCP, however, requires maintaining a global view of the acquired resources, which in distributed systems leads to high communication overhead.

This paper presents a distributed PIP built on top of a deadlock avoidance schema that requires much less communication than PCP. Since the system is already deadlock free and priority inversions can be detected locally, we obtain an efficient dynamic resource allocation system that prevents deadlocks and handles priority inversions.

**Keywords:** Priority Scheduling, Deadlock Avoidance, Distributed Resource Allocation, Distributed Real-Time and Embedded Systems.

## 1 Introduction and Related Work

Modern distributed real-time and embedded systems (DRE) are built using a real-time middleware that extends conventional middleware services with real-time QoS capabilities. It is common in real-time systems to assign priorities to processes to achieve a higher confidence that the more critical tasks will be accomplished in time. A priority inversion is produced when a high priority process is blocked by a lower priority one. State-of-the-art middleware solutions, based for example on CORBA ORBs, may suffer priority inversions [24].

---

<sup>\*</sup> César Sánchez, Henny B. Sipma and Zohar Manna are supported in part by NSF grants CCR-01-21403, CCR-02-20134, CCR-02-09237, CNS-0411363, and CCF-0430102, by ARO grant DAAD19-01-1-0723, and by NAVY/ONR contract N00014-03-1-0939. Christopher D. Gill is partially supported by NSF CAREER Award CCF-0448562.

In centralized systems, priority inversions are usually handled using a protocol from the priority inheritance family [26,15]. However, priority inheritance, and synchronization in general, is significantly affected by concurrent execution. Even though some variations of these centralized protocols have been proposed for multiprocessors [15] and distributed systems [15,13], there is not yet a widely accepted general scheme. We propose here a mathematically sound method to deal with priority inversions in DRE architectures.

In this paper we consider DRE systems that consist of a set of *sites*, each of which is capable of executing a predefined set of computations or methods, connected using an asynchronous network. Processes, consisting of local and remote method calls, are created dynamically. The relevant resources are the threads (or execution contexts) to run the methods. We assume a *WaitOnConnection* [25] thread-allocation policy, that is, each method requires its own thread, including nested up-calls, and methods hold on to their thread until they finish.

Since resources are finite and we impose no restriction on the number of processes, deadlocks are reachable unless there is a mechanism in place to control those allocations. It is important to distinguish between two different kinds of deadlocks: intra-resource (caused by parallel access to a single resource) and inter-resource deadlocks (caused by interference across different allocations).

*Intra-resource deadlocks:* Absence of intra-resource deadlock is one of the requirements of a solution to mutual exclusion, together with exclusive access and, sometimes, lack of starvation. Several algorithms have been proposed for distributed mutual exclusion, which can be classified (see [28,29,17]) into:

- Permission based: A process can access a resource if there is unanimity [19] between the participants about its safety. Unanimity can be relaxed to majority quorums [8,31,12], or even majorities in coteries [7,1]. The message complexities range from  $2(N-1)$  in the original Ricart-Agrawala algorithm [19] to  $O(\log N)$  in the best case (with no failures) in the modern coterie-based approaches.
- Token-based: The system is equipped with a single token per resource, which is held by the process that accesses it. A distributed data-type is maintained to select the next recipient. For example, Suzuki-Kasami's approach [30] exhibits a complexity of  $O(N)$  messages per access, and Raymond's [16] and Naimi-Tehel's [14] approaches, use spanning trees to obtain an average case complexity of  $O(\log N)$ , still exhibiting a  $O(N)$  worst case.

However, the most efficient way (in terms of message complexity) to achieve distributed mutual exclusion is to use a centralized algorithm, like a primary site approach [2]. For every resource, a distinguished site arbitrates the accesses, reducing the problem of distributed mutual exclusion to the centralized case. Thus, allocations can be resolved with one message per request. This comes at a price of lower resiliency because, if the site managing the resource fails, the resource becomes inaccessible. However, in some cases, like DRE systems and Flexible Manufacturing Systems [6,18] resources are indeed tightly coupled to the sites where they reside, and therefore it is natural to use this site as the