

# RapidSched: Static Scheduling And Analysis For Real-Time CORBA

Victor Fay Wolfe,	Russell Johnston	Peter Kortmann and Ben Watson	Steven Wohlever
Lisa C. DiPippo Rama Bethmagalkar, and Gregory Cooper	SPAWAR Systems Center San Diego, CA USA	Tri-Pacific Software Alameda, CA USA	MITRE Corporation Bedford, MA
The University of Rhode Island Kingston, RI USA 02881 lastname@cs.uri.edu	russ@spawar.navy.mil	peter@tripac.com	wohlever@mitre.org

## Abstract

This paper presents a real-time CORBA Scheduling Service called *RapidSched*. *RapidSched* uses a global, distributed deadline monotonic priority assignment, enforcement of priorities by commercial real-time operating systems, and distributed priority ceiling resource management. *RapidSched* is integrated with an enhanced version of the PERTS real-time analysis tool.

## 1 Introduction

We have developed *RapidSched*, an implementation the proposed Real-time CORBS Scheduling Service. *RapidSched* uses a global deadline monotonic priority assignment technique along with a distributed priority ceiling resource access protocol for fixed priority static distributed systems. *RapidSched* is integrated with the PERTS scheduling tool from Tri-Pacific Software [1] that was originally developed at the University of Illinois [2]. In this integrated system, we augmented PERTS to produce a schedulability analysis (using rate-monotonic analysis techniques), an assignment of unique priorities across the distributed system, and the priorities that should be used on the local operating systems. The PERTS schedulability analysis accounts for the additional potential blocking introduced by *priority mapping*, the mapping of global system-wide priorities to the priorities on the local real-time operating systems. We have implemented the Real-Time CORBA Scheduling Service to automatically retrieve the optimal priorities specified by PERTS and use them to assign the priorities for client and server threads in the distributed system.

## 2 Real-Time CORBA Related Work

Many distributed real-time applications, such as command and control, military combat systems, and automated factory control could benefit from the services provided by CORBA middleware software that facilitates client/server communication in a time-cognizant manner. A detailed review of real-time middleware can be found in [11]. Below we describe a few particular implementations and designs.

A Special Interest Group (SIG) has been formed within the Object Management Group with the goal of extending the CORBA standard with support for real-time applications. The real-time SIG recently put out a request for proposals seeking static real-time scheduling in a real-time CORBA framework [3]. Among other things, the RFP asked for end-to-end predictability of client requests, ordered execution of tasks, and real-time control of resource allocation.

There have been several real-time CORBA projects initiated recently. MITRE has done work [4,5] to identify requirements for the use of real-time CORBA in command and control systems. They have prototyped the approach by porting the ILU ORB from Xerox to the Lynx real-time operating system. This system provides a static distributed scheduling service supporting rate-monotonic and deadline-monotonic techniques.

The ACE ORB (TAO)[6], developed at The University of Washington in St. Louis, is a high-performance endsystem architecture for real-time CORBA. It provides support for specification and enforcement of quality of service (QoS), as well as a real-time scheduling service. TAO's scheduling service relies on an off-line, rate-monotonic scheduler to guarantee that deadlines of real-time tasks are met.

Current work at the University of Illinois Urbana-Champaign is extending the TAO system to allow for on-line schedulability testing. Along with the statically guaranteed real-time tasks, the new system will perform admissions tests on dynamic tasks to ensure scheduling feasibility [7].

Previous work at the University of Rhode Island and the SPAWAR Systems Center has developed a dynamic real-time CORBA system [8] that provides expression and best-effort end-to-end enforcement of soft real-time client method requests. Clients create Timed Distributed Method Invocations (TDMI) that include timing information such as deadline, priority and quality of service. The dynamic real-time CORBA system provides end-to-end enforcement of the specified timing constraints through extensions to CORBA's object services. A Global Priority Service provides a uniform global priority for each client request based on the constraints specified in the TDMI. It then translates the global priority to a priority that the server's local operating system can handle. The current system uses a variation of Earliest Deadline First (EDF) scheduling, but can easily be changed to support other priority assignment schemes.

The dynamic real-time CORBA system also has a real-time Event Service for prioritizing delivery of events, and a real-time Concurrency Control Service that provides priority inheritance for requests that are queued on a server. This work in dynamic RT CORBA has laid the groundwork for the work that we present in this paper on static scheduling and priority mapping in a real-time CORBA environment.

### **3 RapidSched**

RapidSched is an implementation of the Real-Time CORBA Scheduling Service, as specified in the current Real-Time CORBA draft specification [14]. It uses a scheduling algorithm based on deadline monotonic priority assignment, distributed priority ceiling resource management, and an optimal priority mapping algorithm to map distributed global priorities to local real-time operating system priorities. The implementation uses the PERTS tool to generate the scheduling parameters and automatically set them in Real-Time CORBA application code. This Section describes the Scheduling Service interface in the draft Real-time CORBA specification. It then describes our scheduling algorithm and RapidSched's implementation.

#### ***3.1 Real-Time CORBA Scheduling Service Interface***

The proposed OMG Real-Time CORBA standard [14] uses the notion of a global, uniform priority assignment to threads of clients and servants in the CORBA system. Global priority is a total ordering of those threads in the system that are assigned a priority by the application programmer. Note that some threads may not be explicitly assigned a priority in application code (e.g. servant threads that inherit the priority of the client), but that eventually every thread will be assigned a global priority. The programmer assigns each client thread one or more fixed, unique global priorities from 1 to N, with 1 being lowest priority and N being highest priority. A client may have more than one priority due to parts of its execution that have tighter timing constraints or higher importance

Fixed priority scheduling entails, whenever possible, resolving scheduling conflicts by allowing the highest global priority thread to use a resource on which the conflict occurs. When, for some reason such as consistency of a shared resource, the RT CORBA system does not resolve conflicts in priority order and causes a higher priority thread to wait for a lower priority thread, "priority inversion" is said to occur. Analyzable real-time systems require that priority inversion be bounded. Our implementation's use of the DPCP resource management protocol supports the bounding of priority inversion by bounding it resulting from queuing of clients waiting to access server methods.

RT CORBA also specifies a *Scheduling Service* that uses the RT CORBA primitives to facilitate enforcing various fixed-priority real-time scheduling policies across the RT CORBA system. The Scheduling Service

abstracts away from the application some of the complication of using low-level RT CORBA constructs, such as the POA policies. For applications to ensure that their execution is scheduled according to a uniform policy, such as global Rate Monotonic Scheduling, RT ORB primitives must be used properly and their parameters must be set properly in all parts of the RT CORBA system. A Scheduling Service implementation will choose CORBA Priorities, POA policies, and priority mappings in such a way as to realize a uniform real-time scheduling policy. Different implementations of the Scheduling Service can provide different real-time scheduling policies.

The Scheduling Service uses “names” (strings) to provide abstraction of scheduling parameters (such as CORBA Priorities). The application code uses these names to specify CORBA Activities and CORBA objects. The Scheduling Service internally associates these names with actual scheduling parameters and policies. This abstraction improves portability with regard to real-time features, eases use of the real-time features, and reduces the chance for errors.

The Scheduling Service provides a `schedule_activity` method that accepts a name and then internally looks up a pre-configured CORBA priority for that name. The Scheduling Service also provides a `create_POA` method to create a CORBA POA, an object in the CORBA server that controls access to the CORBA objects. The `create_POA` call sets the POA’s RT CORBA policies to support the uniform scheduling policy that the Scheduling Service is enforcing. For instance, if the Scheduling Service were enforcing a scheduling policy with priority ceiling semantics, it might create thread pools with priority lanes at the priority ceiling of the objects it manages to ensure that threads start at a high enough priority before dispatch. The Scheduling Service provides a third method,

```
0      install_priority_mapping(. ..);

Client
C1     sched = create scheduling service object;
C2     obj = bind to server object
C3     sched->schedule_activity ("activity1");
C4     obj->method1( params );    // invoke the object
C5     sched->schedule_activity ("activity2");
C6     obj->method2(params );

Server Main
S1     sched = create scheduling service object;
S3     poa1 = sched->create_POA(. . .);
S4     obj = poa1->creat_object ( params );    // create object
S5     sched->schedule_object(obj, "Object1" );
      ...
```

**Figure 1: Example of RT CORBA Static Scheduling Service**

called `schedule_object`, that accepts a name for the object and internally looks up scheduling parameters for that object. For instance, it could set its priority ceiling so that it can do a priority ceiling check at dispatch time.

The example in Figure 1, from the RT CORBA draft standard [14] illustrates how the Scheduling Service could be used and also illuminates some of the issues in creating RT CORBA clients and servers. Assume that a CORBA object has two methods: `method1` and `method2`. A client wishes to call `method1` under one deadline and `method2` under a different deadline.

In Step 0, the Scheduling Service installs a priority mapping that is consistent with the policy enforced by the Scheduling Service implementation. For instance, a priority mapping for an analyzable Deadline Monotonic policy might be different than the priority mapping for an analyzable Rate Monotonic policy.

The `schedule_activity` calls on lines C3 and C5 specify names for CORBA Activities. The Scheduling Service internally associates these names with their respective CORBA priorities. These priorities are specified when the Scheduling Service is instantiated at system startup. For instance, our RapidSched specifies deadline monotonic priorities through a configuration file.

The server in the example has two Scheduling Service calls. The call to `create_POA` allows the application programmer to set the non-real-time policies, and internally sets the real-time policies to enforce the scheduling algorithm of the Scheduling Service. The resulting POA is used in line S4 to create the object. The second Scheduling Service call in the server is the `schedule_object` call in line S5. This call allows the Scheduling Service to associate a name with the object. Any RT scheduling parameters for this object, such as the priority ceiling, are assumed to be internally associated with the object's name by the Scheduling Service implementation.

### ***3.2 RapidSched Algorithm***

RapidSched uses a global Deadline Monotonic priority assignment, Distributed Priority Ceiling resource management, and an optimal priority mapping algorithm. We now described each of these aspects of its scheduling algorithm.

**Deadline Monotonic Scheduling.** The deadline monotonic (DM) priority assignment scheme assumes periodic tasks and statically assigns highest priority to tasks with the shortest deadline [3]. This technique works well with Real-Time CORBA for several reasons. First, DM is a fixed priority assignment scheme, which is required by the current Real-Time CORBA draft. Second, the periodic tasks can have deadlines that are possibly different from their periods, and so DM is a better choice than rate monotonic (RM),

which only takes period into account. Third, the schedulability analysis of DM is well-known [3], although not optimal in a distributed system [9]. In fact, it has been shown that the problem of scheduling any non-trivial system of tasks requiring ordered execution on more than two processors is NP-hard [10].

**Distributed Priority Ceiling.** In our scheduling approach, we use the *distributed priority ceiling protocol* (DPCP) for resource access, such as the access of servers by clients. In a single node system, schedulability of hard real-time tasks that require resources can be computed using well-known analyses [11,12,15] that take into account the timing and resource requirements of all tasks in the system. In a distributed system, this analysis is complicated by the fact that tasks may require resources that reside on other nodes than their own.

The Distributed Priority Ceiling Protocol (DPCP) [15] extends the priority ceiling protocol (PCP) [15] by taking into account accesses to remote resources. In the DPCP, a resource that is accessed by tasks allocated to different processors than its own is called a *global resource*. All other resources (those only accessed by local tasks) are *local resources*. A critical section on a global resource is referred to as a *global critical section* (GCS). A *local critical section* (LCS) refers to a critical section on a local resource. The base priority (BP) of a system of tasks is a fixed priority, strictly higher than the priority of the highest priority task in the system. We assume that higher numbers correspond to higher priorities. As in the single-node PCP, the priority ceiling of a local resource is the priority of the highest priority task that will ever access it. The priority ceiling of a global resource is the sum of the BP and the priority of the highest priority task that will ever access it. When a task executes a GCS, the task suspends itself on its local processor, and the GCS executes at a priority equal to the sum of the BP and the priority of the calling task on the host processor of the resource. Each processor in the system runs the PCP given the priorities and priority ceilings as described above.

The schedulability analysis of the DPCP is an extension of the schedulability analysis of the PCP. The only difference is that there are more forms of blocking due to access of remote resources. For instance, the DPCP analysis must take into account blocking that occurs when a task requests a global resource on another node, but must wait for a lower priority task that currently holds the resource.

**Priority Mapping.** The theory behind the analysis of DM+DPCP assumes unique priorities assigned to tasks and GCS's. However, consider an example with 100 clients on a node, each with 2 intermediate deadlines, which map to 300 tasks, all invoking methods (GCSs) on other nodes. If the node was running VXWorks as its local real-time operating system, there would be only 256 local priorities with which to execute the 300 tasks. This is an instance of the *priority mapping* problem. We have developed an optimal algorithm to perform priority mapping for fixed priority Real-Time CORBA systems. The algorithm essentially traverses all tasks and GCSs in global priority order attempting to "squeeze" global priorities

into local priorities. After each attempted squeeze, a schedulability analysis is done to determine if the system remains schedulable. This process continues until each node has fit all global priorities assigned to it. Details of this algorithm are presented in [13].

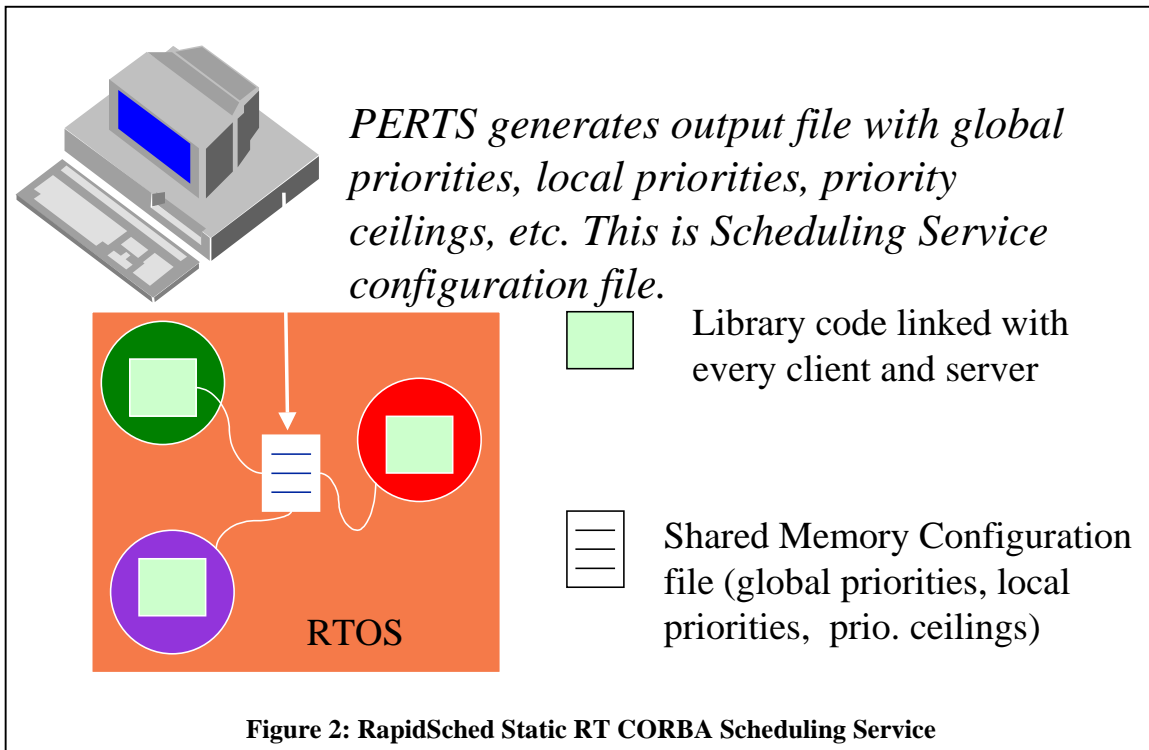
### **3.3 RapidSched Implementation**

RapidSched consists of a PERTS front-end to generate scheduling parameters and a set of Real-Time CORBA libraries to enforce the semantics of the Scheduling Service.

**PERTS Front-End.** We have developed an extended version of the PERTS [1] real-time analysis tool to determine the schedulability of a RT CORBA system [13]. PERTS provides a graphical interface to allow users to enter real-time task information, such as deadline, execution time, resource requirements. PERTS then computes a schedulability analysis on the given system using well-known techniques, such as rate-monotonic analysis [11,12,15]. PERTS was originally developed at the University of Illinois, Urbana-Champaign, and commercialized by Tri-Pacific Software/. We have developed a mapping from RT CORBA clients and servers to PERTS primitives – tasks and resources. A periodic client with  $m$  intermediate deadlines is mapped to  $m$  dependent tasks, each with the same period, and with deadlines corresponding to the intermediate deadlines of the client. Each server in the RT CORBA system is mapped to a PERTS resource. This allows users to enter RT CORBA constructs, and have PERTS automatically translate them into primitives that it can analyze. The extended PERTS analyzes the RT CORBA system using deadline monotonic scheduling and distributed priority ceiling protocol for concurrency control. Given the real-time requirements of each client and server in the system, if the system is found to be schedulable, the extended PERTS system produces priorities for each client task, and priority ceilings for each server resource in the system. If the system is found to be non-schedulable, PERTS produces graphs and other information for each client task to indicate what caused the system to be non-schedulable.

**Scheduling Service Libraries.** The Real-Time CORBA Scheduling Service interface described in Section 3.1 requires CORBA priorities and a priority mapping function. The implementation of DPCP requires knowing the priority ceiling for CORBA servers. RapidSched takes these parameters automatically from the output of PERTS.

Recall that the extended PERTS produces a mapping of global priorities to local system priorities. PERTS also produces a second mapping of unique task names to global priorities and a third mapping of priority ceilings associated with unique names for each server in the system. These mappings are generated by PERTS as a set of configuration files that are read in by RapidSched when it is instantiated at system startup.



RapidSched currently implements deadline monotonic scheduling with DPCP for control of shared resources. All priorities and priority ceilings are computed *a priori* through PERTS, as described above. RapidSched uses ORB *interceptors* to implement the PCP on each node. An interceptor is an ORB feature that provides an interface to allow application code to be executed in the internals of the ORB. RapidSched installs an interceptor that catches all calls to the object’s methods. Before the method is executed and a result is passed back to the calling client, the interceptor executes the priority ceiling check; i.e. the priority of the client task is strictly higher than the highest priority ceiling of servers on the node that are locked by other tasks.

The objects of RapidSched are implemented as shared library code and are co-located with their respective clients and servers. Thus, there is no network delay for scheduling service calls, and inter-process communication on the same node is minimized. The scheduling objects communicate via shared memory (see Figure 6), mutexes, and condition variables to implement the concurrency control mechanism. Information about priority mapping is also stored in shared memory for fast run-time access.

## Conclusion

This paper has described RapidSched’s technique for real-time fixed priority scheduling in middleware for static applications. It assumes the existence of preemptive priority-based scheduling in the real-time

operating systems on the nodes in the system. In our technique, client threads have their priorities set using deadline monotonic assignment of global priorities across the distributed system. Server threads have their priority. RapidSched adheres to the current proposed standard Real-Time CORBA Scheduling Service interface. RapidSched is integrated with an enhanced version of the commercial PERTS real-time analysis tool which provides schedulability analysis and the optimal global and local priority settings. These settings are automatically used by RapidSched to relieve the application programmer from determining and entering them by hand. We have prototypes of RapidSched for the ORBExpress Real-Time ORB from OIS Inc on VXWorks RTOS from WindRiver Systems, and for ChorusORB on ClassiX RTOS from Sun, and for Orbix ORB from Iona on Solaris from Sun.

## References

- [1] TriPacific Software. at [www.tripac.com](http://www.tripac.com).
- [2] Jane W. S. Liu, et. al. PERTS: A Prototyping Environment for Real-Time Systems. *Technical Report UIUCDCS-R-93-1802*, The University of Illinois, Urbana, May 1993. Commercial version information available at [www.tripac.com](http://www.tripac.com).
- [3] OMG. Real-Time Special Interest Group's Request For Proposals. Electronic document at <http://www.omg.org/docs/realtime/97-05-03.txt>.
- [4] P. Krupp, A. Schafer, B. Thuraisingham, and V.F. Wolfe. On Real-Time Extensions to the Common Object Request Broker Architecture. In *Proceedings of the Object Oriented Programming, Systems, Languages, and Applications (OOPSLA) '94 Workshop on Experiences with CORBA*, Sept. 1994.
- [5] E. Bensley, et. al. Object-Oriented Approach for Designing Evolvable Real-Time Command and Control Systems. In *WORDS '96*, February, 1996.
- [6] D. Schmidt, R. Bector, D. Levine, S. Mungee, G. Parulkar. TAO: A Middleware Framework for Real-Time ORB Endsystems. In *Proceedings of the 1997 IEEE Workshop on Middleware for Distributed Real-Time Systems and Services*, San Francisco, CA, December 1997.
- [7] W. Feng, U. Syyid and J. W.-S. Liu. Providing for an Open, Real-Time CORBA. In *Proceedings of the 1997 IEEE Workshop on Middleware for Distributed Real-Time Systems and Services*, San Francisco, CA, December 1997.
- [8] L. DiPippo, V.F. Wolfe, R. Johnston, R. Ginis, M. Squadrito, S. Wohlever, I. Zyxh. Expressing and Enforcing Timing Constraints in a Dynamic Real-Time CORBA System. *Real-Time Systems*. to be published.
- [9] Jun Sun. *Fixed-Priority End-to-End Scheduling in Distributed Real-Time Systems*. PhD Thesis. University of Illinois, Urbana-Champaign, 1997.
- [10] J. K. Lenstra, A. H. G. Rinnooy Kan, and P. Brucker. Complexity of Machine Scheduling Problem. *Annals of Discrete Mathematics*, 1:343-362, 1977.
- [11] C. Liu and J. Layland. Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment. *Journal of the ACM*, vol. 30, pp. 46-61, January 1973.
- [12] J. Lehoczky, L. Sha, and Y. Ding. The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior. In *Proceedings fo the IEEE Real Time Systems Symposium*, 1989.

---

[13] L. DiPippo, V.F. Wolfe, L. Esibov, G. Cooper, R. Johnston, B. Thuraisingham, J. Mauer. Scheduling and Priority Mapping for Static Real-Time Middleware. University of Rhode Island Technical Report , November 1998. Submitted to *Real-Time Systems* special issue on real-time middleware.

[14] OMG. *Realtime CORBA*. Electronic document at <http://www.omg.org/docs/orbos/98-10-05.pdf>.

[15] Ragnathan Rajkumar. *Synchronization in Real-Time Systems: A Priority Inheritance Approach*. Kluwer Academic Publishers, Boston, MA. 1991.