FLARe: a Fault-tolerant Lightweight Adaptive Real-time Middleware for Distributed Real-time and Embedded Systems

Jaiganesh Balasubramanian
Department of Electrical Engineering and Computer Science
Vanderbilt University, Nashville, TN 37203
jai@dre.vanderbilt.edu

ABSTRACT

An important class of distributed real-time and embedded (DRE) applications consists of periodic soft real-time tasks. Timeliness and availability are essential requirements for the correct operation of these applications. Conventional solutions to these challenges tend to use non-adaptive and load-agnostic fault tolerance solutions within a real-time system, which often end up making ad hoc fault tolerance (e.g., failover targets) decisions that can further overload already strained resources. Potential adverse consequences of these ad hoc actions include excessive delays for real-time tasks and cascades of resource failures.

This paper presents FLARe, which is a middleware that provides adaptive fault tolerance for DRE systems. FLARe's resource management infrastructure monitors various system metrics, including CPU utilization, and makes informed, load-aware, and adaptive decisions about the application's fault tolerance configurations (e.g., failover targets, physical placement of replicas). FLARe also employs decision making algorithms to adapt these decisions at runtime as faults occur and provides trade-offs between timeliness, availability, and performance as resources get overloaded, removed, or added.

1. INTRODUCTION

Many distributed real-time and embedded (DRE) systems, such as shipboard computing systems [26], consist predominantly of soft real-time tasks that must continue to provide real-time quality of service (QoS) even when hardware and software faults occur. For example, the behavior of the object tracking subsystem of a shipboard computing environment is influenced by external sensor readings. The object tracking system should be available and responsive even if processes or processors fail. Likewise, it should continue to provide timely response even when system workload varies significantly at runtime, *e.g.*, due to faults, dynamic task arrival, or intrusion detection.

MDS 2007, November 26-30, 2007, Newport Beach, CA, Canada

Conventional fault tolerance solutions replicate servers that may fail independently, giving clients a robust service that appears as though it was provided by a single server. Two common approaches for maintaining replicas are ACTIVE and PASSIVE replication. In ACTIVE replication, a collection of servers maintains identical state, and all client requests are executed atomically in all of the replicas using a group communication service. If a server fails other servers continue to execute the protocol, so clients are not affected by the failures of individual replicas.

Although ACTIVE replication has been used for some hard real-time systems [17] it is expensive for systems that do not require strong state consistency or hard real-time guarantees due to its high resource usage [31]. For such systems, PASSIVE replication may be preferred, where one replica—called the primary—handles all client requests, and backup replicas receive state updates from the primary. If the primary dies, a backup is elected to become the new primary.

To decouple application logic from dealing with the complexities of the different fault tolerance mechanisms, middleware such as CORBA, and J2EE, alleviate many inherent and accidental complexities in DRE application integration with QoS provisioning mechanisms, and have become essential to the development and integration of DRE systems. Specialized middleware specifications (for e.g., Fault-Tolerant CORBA (FT-CORBA) [21], and Real-Time CORBA (RT-CORBA) [22]) have been adopted to provide real-time and fault tolerance capabilities to DRE applications using the PASSIVE replication scheme.

These middleware specifications provide applications with a transparent way to configure the fault tolerance (e.g., replication degree, checkpointing interval) and fault recovery (e.g., address of next primary) mechanisms to meet an application's fault tolerance requirements. For example, middleware keeps a list of replica references, and automatically redirects clients to one of the replicas (e.g., chosen in a roundrobin fashion) in case of the primary failure. However, it is likely that those fault tolerance and recovery configurations are not valid at runtime because of resource failures. This causes delays in recovering from failures which is not acceptable for real-time tasks to maintain their deadlines. Similarly, as system resources fail at runtime, it may not be possible to operate all the replicas at the deployment-time fault tolerance configuration, and trade-offs need to be made between availability of resources and strength of fault tolerance.

Current solutions lack runtime decision making algorithms

working in conjunction with fault-tolerant middleware to make fault tolerance and recovery configuration decisions in a resource-aware manner. Specifically, existing solutions suffer from the following drawbacks:

Lack of support for timely and appropriate client failover. Current middleware support [23, 25, 1, 24] configures PASSIVE replication recovery strategies in a static fashion, which allows timely client redirection. However, after failover, client-perceived response times depend on the load on the processor hosting the new primary. Since the failover targets are chosen statically, and without the knowledge of the current system resource availability, client failovers could cause system pollution, where different processor failures caused all the clients to failover to the same processor. This could lead to cascading resource failures seriously affecting the real-time and fault tolerance capabilities of the system. DRE applications need support from middleware for timely and appropriate failover.

Lack of support for reconfiguration as part of the recovery process. PASSIVE replication scheme relies on appropriate physical placement of replicas to handle failures, and recovers by redirecting clients to one of the deployed replicas. So, even an optimal initial distribution might not suffice as tasks might be added to the system at runtime causing (1) increase in resource utilization in certain processors, and (2) load imbalance amongst the processors in the system. As failures followed by client failovers happen, this load imbalance increases causing some of the tasks to miss real-time deadlines. DRE applications need middleware and algorithmic support for fast reconfigurations at runtime to handle load imbalances and overloads caused by resource failures and subsequent client failovers. Existing middleware support for adaptive fault tolerance [16, 15, 12] do not solve such issues.

Lack of support for making trade-offs between real-time performance and availability requirements. When both real-time and fault tolerance must be satisfied within the same system, it is rather likely that trade-offs [19] are made during the composition. For example, in conditions where overloads cannot be controlled by migration, performance needs to be compromised by operating tasks with implementations which consume less resources but give a performance lower than the possible capacity. A better solution could be to stop certain implementations, and run the remaining implementations in the same performance. A run-time decision making unit with support from utility optimization algorithms could make the best possible decision to be implemented by the middleware. However, current fault tolerance solutions [20, 10, 1] do not provide such algorithmic support to work in conjunction with middleware-based solutions to make trade-offs between performance and fault tolerance capabilities.

The goal of our research is to devise novel techniques and middleware platform services that provide real-time and fault tolerance capabilities to soft real-time DRE applications while using PASSIVE replication schemes. In particular, effective PASSIVE replication schemes for DRE systems require innovations in (1) timely failover, (2) adapting fault tolerance configurations (e.g., to determine failover targets) in response to changing resource availabilities, (3) adaptive reconfiguration of the system to recover from processor failures and overloads caused by subsequent client failovers, and (4) selecting and applying trade-offs amongst fault tolerance, real-time, and performance capabilities of the application while recovering from resource failures, as described below.

What DRE systems need, therefore, are a combination of middleware and algorithmic capabilities that integrate real-time and fault tolerance by design, and are adaptive and load-aware so that these solutions can maintain soft real-time performance in the face of failures. This paper describes FLARe, which is a middleware that provides an adaptive, fault tolerance solution for Real-time CORBA [22] applications. Specifically, FLARe uses the following novel techniques to provision real-time and fault tolerance capabilities to DRE applications:

- *Adaptive, load-aware, and timely client failover techniques*, where decision-making algorithms determine failover targets based on up-to-date utilization estimates, and these targets are proactively updated with the middleware managing client redirection, so that recovery can be fast, transparent, appropriate, and help maintain timely response to client's requests after a failover.
- *Dynamic reconfiguration, and load shedding techniques*, where adaptive reconfiguration and overload management algorithms (1) work with middleware redirection mechanisms to proactively (before the occurrence of a failure) redirect clients to appropriate targets to handle overloads caused by client failovers, and (2) assist in reassigning tasks from failed processors to fault-free processors to maintain replication degree requirements of the applications.
- *Utility optimization techniques*, where fault tolerance configurations (e.g., checkpointing, replication degree, application implementations) are changed dynamically in response to changing resource availabilities to trade-off performance and availability requirements of applications.

2. DESIGN OF FLARE

This section describes the design of the FLARe middleware. We describe its replication style, real-time system model, and fault model, as well as its software design.

2.1 FLARe's Real-time System Model

FLARe supports DRE systems consisting predominantly of soft periodic real-time tasks, such as those found in shipboard computing and intelligence, surveillance, and reconnaissance systems.[1] We assume the soft periodic tasks are deployed on a RT-CORBA [22] infrastructure, such as that provided by the TAO middleware (`www.dre.vanderbilt.edu`). These tasks are scheduled on different nodes of the DRE system, with tasks on each node scheduled using Rate Monotonic Scheduling (RMS) [18].

The client-side request rate defines the priority at which the task will execute at the server. We therefore use RT-CORBA's CLIENT_PROPAGATED priority model, which allows the clients to dictate the priority at which the servers have to service their requests. Since a single server process may handle multiple clients with different priorities, we also use the RT-CORBA *thread pool with lanes* feature,

[1] In such systems we assume that each node has a single processor and hard real-time tasks are provisioned separately, with dedicated hardware and software, and as such are outside the scope of this discussion.

which partitions the available number of threads across different priorities, so that the server can simultaneously serve multiple client requests with multiple priorities. While our current implementation employs RMS, our middleware architecture can easily incorporate other scheduling policies, such as Maximum Urgency First [27] (MUF).

2.2 Fault Model

Our research focuses on a fail-stop model of failures, where processes or processors can fail and the remainder of the system can continue executing. We assume that processor faults are *hard* faults, *i.e.*, when a processor has a fault it stops permanently. These types of faults may occur due to aging or acute damage, though in domains like shipboard computing acute damage is the main concern since hardware components are periodically replaced through routine maintenance. Considering unpredictable behavior of processes or processors is beyond the scope of our research.

We assume that networks provide bounded communication latencies and do not fail. This assumption is reasonable for certain DRE systems, such as shipboard computing, where nodes are connected by highly redundant high-speed networks. Relaxing this assumption through integration of our middleware with network level fault tolerance techniques is an area of future work.

2.3 FLARe's Middleware Architecture

We now describe how FLARe is designed to provide the key capabilities as described in Section 1. Figure 1 depicts the architecture of the FLARe. We first describe the responsibilities of the major components of FLARe. Then we describe how the major middleware components work together with different runtime algorithms to provide the capabilities described in Section 1.

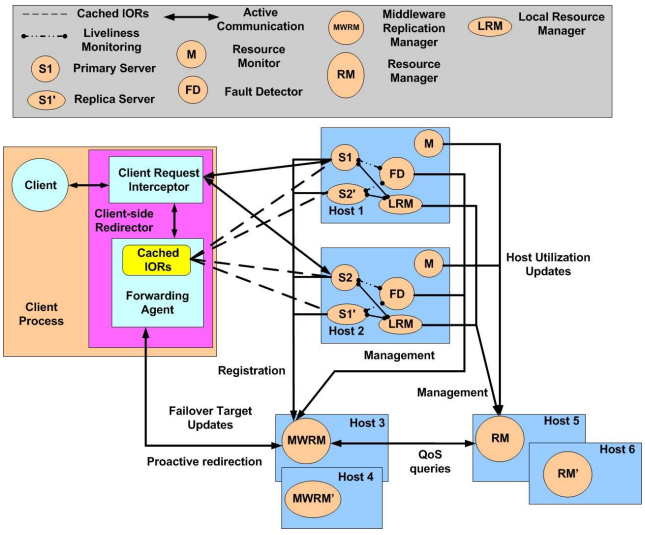

Figure 1: FLARe Middleware Architecture

Middleware replication manager. FLARe's middleware replication manager provides interfaces for registering and managing information about the server objects and their backup replicas. The purpose of the middleware replication manager is to manage the fault tolerance and recovery requirements of the applications, and use them as a blueprint to adapt configurations at runtime in response to changing resource availabilities.

Resource Manager. The resource manager acts as a bridge between the decision making algorithms and the middleware implementing the decisions of the decision making algorithms. The function of the resource manager is to use adaptive resource management algorithms to make runtime, resource-aware, QoS-aware decisions about the configuration of the fault tolerance and real-time service of the system. The resource manager is designed to be extensible to allow plugging in many different algorithms which are used for different purposes in the context of FLARe. For example, in certain scenarios, after a client failover, the processor may not get overloaded, in which case the resource manager does not apply the dynamic reconfiguration algorithms.

Local resource manager. Local resource manager resides in each processor and works in conjunction with the global resource manager to implement the processor-specific trade-off decisions between performance and availability of applications.

Client interceptors and forwarding agent. Client interceptors and the forwarding agent help shield the applications from managing the responsibilities associated with client redirection.

Resource monitors and fault detectors. Resource monitors monitor different resources (e.g., CPU, memory, network bandwidth) in the system, and periodically updates the resource manager. Fault detectors monitor the health of processes and processors and periodically updates the middleware replication manager.

2.4 Functionality of FLARe

We now describe how the major middleware components work together with different runtime algorithms to provide the capabilities described in Section 1 and also provide solutions to the drawbacks listed in Section 1.

Adaptive, load-aware, and timely client failover techniques. Middleware replication manager manages information such as server's multiple implementations, their resource utilization requirements, and their utility values, and decides on an appropriate implementation to operate in resource-constrained scenarios. Middleware replication manager also keeps track of information about the server replica and their backup placements, replication degree, and consistency requirements. Such information can also be fed to the middleware replication manager using a modeling tool like MDDPro [28]. At runtime, middleware replication manager adapts such deployment-time fault tolerance (e.g., backup location) and performance configuration information by working in conjunction with the resource manager.

Resource manager collects metrics like CPU utilization from the processors hosting the replicas, and utilizes replica selection algorithms to make resource-aware fault tolerance configuration decisions like determination of failover targets for each server replica. The middleware replication manager periodically makes QoS queries on the resource manager to determine the failover target, and redirects the information to the client-side middleware so that client failover can be timely and appropriate.

Forwarding agent resides in the client middleware process and periodically queries the middleware replication manager about the appropriate failover targets to redirect clients. When the client interceptors catch exceptions created by processor or process failures, they make a request on the for-

warding agent to know the backup location address. Since the request from the interceptor to the forwarding agent traverses the same in-process address space, no network latency is involved, and the client redirection is fast, and appropriate.

To detect the failure of a process quickly, each application process on a processor opens up a passive POSIX local socket (also known as a UNIX domain socket), and registers the port number with the **fault detector**. The fault detector connects to and performs a blocking read on the socket. If an application process crashes, the socket and the opened port will be invalidated. The fault detector then receives an invalid read error on the socket, which indicates the failure of the process.

The effectiveness of the adaptive, timely, and load-aware client failover technique depends on the replica selection algorithm used by the resource manager to periodically update the forwarding agents. We have implemented a least-loaded replica selection algorithm described in Algorithm 1.

Algorithm 1 Determine per-object failover targets

```
1: N = number of processors
2: for i = 1 to N do
3:    reset expected utilization of all the processors to the
      current utilization
4:    P = number of processes in this processor i
5:    for j = 1 to P do
6:       O = number of objects running in this process j
7:       for k = 1 to O do
8:          find all the processors of the object k's replicas
9:          find the processor MIN with the minimum ex-
            pected utilization
10:         failover target for object k is the object running
            in MIN
11:         expected utilization of processor MIN += object
            k's load
12:      end for
13:   end for
14: end for
```

FLARe's replica selection algorithm chooses the processor with the lowest utilization from among all processors hosting an object's replicas as its failover target. The expected utilization variable is used to account for the failover decision of other objects located on the same processor. By selecting the processor with the lowest expected utilization, our replication selection algorithm distributes the failover targets of objects on a single processor to multiple processors. A detailed evaluation of FLARe's adaptive, load-aware and timely failover techniques can be obtained from [4].

Dynamic reconfiguration, and load shedding techniques. The goal of FLARE's replica selection algorithm is to allow clients failover to replicas hosted in processor, whose load does not increase beyond a threshold after the failover. However, in certain scenarios, after a client failover, the load of a processor might increase beyond the utilization bound within which RMS can guarantee scheduling of real-time tasks in a processor. Rather than delaying the failover, and affecting the real-time schedules of tasks, FLARe proceeds with the failover. However, if the overloads are not controlled appropriately, some of the tasks in the processor will miss their deadlines.

We are developing and implementing overload management algorithms in the context of FLARe's resource manager. The primary goal of the overload management algorithms is to provide a fast reconfiguration with minimum disturbance to the clients, whose availability requirements might get affected during the reconfiguration. The idea is to take advantage of the replicas hosted in processors which are lightly loaded, but whose primary replica is hosted in the processor that is overloaded. This is a case of forced or proactive redirection of clients from one replica to another even though the failure has not occurred. Once forced redirection targets are identified, the middleware replication manager notifies the forwarding agents. The forwarding agents proceed with the redirection at a moment that causes the least disturbance for the clients (e.g., after the end of a request and before the start of the next period).

Utility optimization techniques. In scenarios, where dynamic reconfiguration and load shedding algorithms cannot control the overloads in processors, resource manager makes use of utility optimization techniques [30] to get the resource consumption in the processors down to an acceptable value (e.g, RMS utilization bound). The resource manager looks at the utility values of all the objects operating in the processor, and tries all possible combinations which can bring the utilization down to the target threshold. It then picks the combination, that gives the maximum utility value. The utility values could be assigned based on (1) the implementation the object operates, and (2) whether the object operates or not.

Local resource manager works in conjunction with the resource manager to implement the trade-off decisions between performance and availability of applications. For example, in certain resource-constrained scenarios, the resource manager might decide to operate a less resource consuming implementation of a server hosted in a processor. The recovery manager conveys this information to the local resource manager, which uses the techniques described in [3] to swap server implementations. We are in the process of developing sophisticated utility optimization algorithms and strategies that can work in conjunction with middleware mechanisms to provide degraded QoS to applications.

3. RELATED WORK

Real-time fault tolerant scheduling. Fundamental ideas and challenges in combining real-time and fault tolerance are described in [29], where the notion of imprecise computations have been used to provide degraded QoS to applications operating in the presence of failures. [10] proposes adaptive fault tolerance mechanisms to choose a suitable redundancy strategy for dynamically arriving aperiodic tasks based on system resource availability. [8] proposes a feasibility test to determine if a given task set is schedulable for fault-tolerant purposes using earliest deadline first (EDF) scheduling. [12] proposes a fixed priority-driven preemptive scheduling scheme to preallocate time intervals to both the primary and backup replicas of a task, and adaptively executes either the primary or a backup depending on failures and available time. [15] generates a FT schedule for tasks with precedence constraints and plans for sufficient slack time to handle recovery actions in case of failures. FLARe differs from these approaches in providing fault tolerance capabilities to soft real-time applications. Rather than ensuring hard deadlines are met in the presence of failures, therefore, FLARe focuses on minimizing the impact

of failure recovery on client response times and system resource utilization, and also provides timely client failover to appropriate failover targets.

Allocation of resources for fault tolerance. Other research has focused on deployment-time allocation of resources to tasks operating in a multi-processor environment while considering fault tolerance. [9] focuses on choosing appropriate task implementations and degrees of replication for fault tolerance depending on system resource availability. [11] proposes a fully polynomial-time approximation algorithm to map tasks and their replicas to heterogeneous multiprocessors. [2] proposes a bi-criteria heuristic for scheduling operations in heterogeneous architectures while minimizing schedule length and maximizing reliability. [6] proposes a polynomial-time approximation scheme for replication of periodic hard real-time tasks in identical multiprocessor environments while minimizing system utilization. The FLARe middleware can be extended readily to support deployment-time allocation planning using such algorithms. Furthermore, as failures occur and tasks arrive dynamically at run-time, FLARe can also adapt by changing failover targets on the fly so that client response times are not overly affected by failures.

Real-time fault-tolerant middleware. Delta-4/XPA [24] was an early effort to provide real-time fault-tolerant solutions to distributed systems by using the semi-active replication model, where all the replicas are active, but only one replica sends output responses. ARMADA [1] defines a set of communication and middleware services that support fault tolerance and end-to-end guarantees for real-time distributed applications. MEAD [23] and its proactive recovery strategy for distributed CORBA applications can minimize the recovery time for DRE systems. The Time-triggered Message-triggered Objects (TMO) project [16] considers replication schemes such as the primary-shadow TMO replication (PSTR) scheme, for which recovery time bounds can be quantitatively established, and real-time fault tolerance guarantees can be provided to applications. FLARe's research contributions are similar to these projects in providing modular middleware services to add fault tolerance capabilities to object-based systems. FLARe also enhances traditional fault tolerance techniques with utilization monitoring techniques, however, so as to minimize the effect of recovery on client response times, and to manage system resources efficiently.

Dynamic migration and reconfiguration. Deplanche et all [7] have studied task migration in the context of reconfiguration in fault-tolerant distributed systems. Hou et all [13] have used the minimum laxity first served algorithm to do task migration to meet real-time deadlines. Bettati et all [5] provide timing guarantees to clients by dynamically changing resource allocations and migrating resources from one node to another. Kalogeraki et all [14] propose two algorithms to gracefully migrate objects from the processors when processor overloads and high task latencies are detected. FLARe's dynamic reconfiguration goals are not to migrate objects from one node to another. But FLARe wants to make use of the available replicas, and redirect clients from one replica to another, thereby exploiting lightweight migration with minimal disturbance to clients. Moreover, FLARe applies dynamic reconfiguration in the context of PASSIVE replication scheme as opposed to ACTIVE replication schemes.

4. CONCLUDING REMARKS AND ONGOING WORK

This paper describes the design of FLARe, which is a lightweight middleware that enhances RT-CORBA to provide adaptive and load-aware fault tolerance solutions for DRE systems. We have implemented a prototype of FLARe with a least loaded replica selection algorithm as described in [4]. Our initial evaluations have shown that FLARe's proactive load-aware failover strategy can support transparent and timely failure handling for DRE applications by selecting failover targets on processors with the least load, thereby minimizing the impact of failures, such as unpredictable system utilization and increased client-perceived end-to-end response times.

Currently, our work focuses on the following research issues:

- We are developing dynamic reconfiguration and load shedding algorithms in the context of the resource manager and hope to have a prototype available by November 2006. We also plan to evaluate the performance of the algorithms in a representative DRE system case study deployed in a Linux test bed, and plan to write a technical paper by December 2007.
- We are also in the process of developing utility optimization techniques to manage the trade-offs between performance and fault tolerance, and provide solutions that work in conjunction with the middleware mechanisms to manage the real-time and fault tolerance capabilities of the system. We plan to have a prototype implementation with few domain-specific utility optimization strategies (e.g., changing implementations for image processing controllers) under given constraints in a practical deployment scenario. We plan to evaluate the performance of the algorithms in a representative DRE system case study and write a technical paper by March 2008.
- Supporting stateful applications in DRE systems not only requires timely failover, but client consistency requirements, such as weak or strong consistency models. FLARe is currently designed for stateless applications, so our future work will enhance the replica selection algorithm to consider consistency levels of the replicas while choosing failover targets. We are also enhancing FLARe to support replication requirements for different consistency levels. We plan to approach the research as an aperiodic scheduling problem with application requiring support from the fault tolerant middleware to schedule its state synchronization activities. We plan to have a prototype implementation ready by May 2008, and plan to write a technical paper at that time.

5. REFERENCES

[1] T. F. Abdelzaher, S. Dawson, W. chang Feng, F. Jahanian, S. Johnson, A. Mehra, T. Mitton, A. Shaikh, K. G. Shin, Z. Wang, H. Zou, M. Bjorkland, and P. Marron. ARMADA middleware and communication services. *Real-Time Systems*, 16(2-3):127–153, 1999.

[2] I. Assayad, A. Girault, and H. Kalla. A bi-criteria scheduling heuristic for distributed embedded systems under reliability and real-time constraints. In *DSN '04*.

[3] J. Balasubramanian, B. Natarajan, D. C. Schmidt, A. Gokhale, G. Deng, and J. Parsons. Middleware Support for Dynamic Component Updating. In *International Symposium on Distributed Objects and Applications (DOA 2005)*, Agia Napa, Cyprus, Oct. 2005.

[4] J. Balasubramanian, S. Tambe, A. Gokhale, C. Lu, C. Gill, and D. C. Schmidt. FLARe: a Fault-tolerant Lightweight Adaptive Real-time Middleware for Distributed Real-time and Embedded Systems. Technical Report ISIS-07-812, Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN, May 2007.

[5] R. Bettati and A. Gupta. Dynamic Resource Migration for Multiparty Real-Time Communication. *Proceedings of the IEEE 16th International Conference on Distributed Computing Systems*, pages 646–655.

[6] J.-J. Chen, C.-Y. Yang, T.-W. Kuo, and S.-Y. Tseng. Real-time task replication for fault tolerance in identical multiprocessor systems. *IEEE RTAS*, 0:249–258, 2007.

[7] A. Deplanche and J. Elloy. Task Redistribution with Allocation Constraints in a Fault-Tolerant Real-Time Multiprocessor System. *Distributed Processing–Proceedings of the IFIP WW6*, 10(3):133–150.

[8] S. Ghosh, R. Melhem, and D. Mosse. Enhancing real-time schedules to tolerate transient faults. In *RTSS '95*.

[9] S. Ghosh, R. Rajkumar, J. Hansen, and J. Lehoczky. Scalable resource allocation for multi-processor qos optimization. In *ICDCS '03*.

[10] O. Gonzalez, H. Shrikumar, J. A. Stankovic, and K. Ramamritham. Adaptive fault tolerance and graceful degradation under dynamic hard real-time scheduling. In *RTSS '97*.

[11] S. Gopalakrishnan and M. Caccamo. Task Partitioning with Replication upon Heterogeneous Multiprocessor Systems. In *RTAS 2006*.

[12] C.-C. Han, K. G. Shin, and J. Wu. A fault-tolerant scheduling algorithm for real-time periodic tasks with possible software faults. *IEEE Transactions on Computers*, 52(3):362–372, 2003.

[13] C. Hou and K. Shin. Load sharing with consideration of future task arrivals inheterogeneous distributed real-time systems. *Computers, IEEE Transactions on*, 43(9):1076–1090, 1994.

[14] V. Kalogeraki, P. Melliar-Smith, and L. Moser. Dynamic migration algorithms for distributed object systems. *icdcs*, 00:0119, 2001.

[15] N. Kandasamy, J. P. Hayes, and B. T. Murray. Transparent recovery from intermittent faults in time-triggered distributed systems. *IEEE Transactions on Computers*, 52(2):113–125, 2003.

[16] K. H. K. Kim and C. Subbaraman. The pstr/sns scheme for real-time fault tolerance via active object replication and network surveillance. *IEEE Transactions on Knowledge and Data Engineering*, 12(2):145–159, 2000.

[17] H. Kopetz, A. Damm, C. Koza, M. Mulazzani, W. Schwabl, C. Senft, and R. Zainlinger. Distributed fault-tolerant real-time systems: The mars approach. *IEEE Micro*, 09(1):25–40, 1989.

[18] J. Lehoczky, L. Sha, and Y. Ding. The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior. In *Proceedings of the 10th IEEE Real-time Systems Symposium (RTSS 1989)*, pages 166–171. IEEE Computer Society Press, 1989.

[19] P. Narasimhan. Trade-Offs Between Real-Time and Fault Tolerance for Middleware Applications. Workshop on Foundations of Middleware Technologies, Nov. 2002.

[20] P. Narasimhan, T. Dumitras, A. M. Paulos, S. M. Pertet, C. F. Reverte, J. G. Slember, and D. Srivastava. MEAD: support for Real-Time Fault-Tolerant CORBA. *Concurrency - Practice and Experience*, 17(12):1527–1545, 2005.

[21] Object Management Group. *Fault Tolerant CORBA, Chapter 23, CORBA v3.0.3*, OMG Document formal/04-03-10 edition, Mar. 2004.

[22] Object Management Group. *Real-time CORBA Specification v1.2 (static)*, OMG Document formal/05-01-04 edition, Nov. 2005.

[23] S. Pertet and P. Narasimhan. Proactive Recovery in Distributed CORBA Applications. In *DSN 2004*.

[24] D. Powell. Distributed fault tolerance: Lessons from delta-4. *IEEE Micro*, 14(1):36–47, 1994.

[25] Y. Ren, D. Bakken, T. Courtney, M. Cukier, D. Karr, P. Rubel, C. Sabnis, W. Sanders, R. Schantz, and M. Seri. AQuA: an adaptive architecture that provides dependable distributed objects. *Computers, IEEE Transactions on*, 52(1):31–50, 2003.

[26] D. C. Schmidt, R. Schantz, M. Masters, J. Cross, D. Sharp, and L. DiPalma. Towards Adaptive and Reflective Middleware for Network-Centric Combat Systems. *CrossTalk - The Journal of Defense Software Engineering*, Nov. 2001.

[27] D. B. Stewart and P. K. Khosla. Real-time Scheduling of Sensor-Based Control Systems. In W. Halang and K. Ramamritham, editors, *Real-time Programming*. Pergamon Press, Tarrytown, NY, 1992.

[28] S. Tambe, J. Balasubramanian, A. Gokhale, and T. Damiano. MDDPro: Model-Driven Dependability Provisioning in Enterprise Distributed Real-Time and Embedded Systems. In *Proceedings of the International Service Availability Symposium (ISAS)*, Durham, New Hampshire, USA, 2007.

[29] F. Wang, K. Ramamritham, and J. A. Stankovic. Determining redundancy levels for fault tolerant real-time systems. *IEEE Transactions on Computers*, 44(2):292–301, 1995.

[30] Zhongtang Cai and Vibhore Kumar and Brian F. Cooper and Greg Eisenhauer and Karsten Schwan and Robert E. Strom. Utility-Driven Proactive Management of Availability in Enterprise-Scale Information Flows. In *Proceedings of ACM/Usenix/IFIP Middleware*, pages 382–403, 2006.

[31] H. Zou and F. Jahanian. A real-time primary-backup replication service. *Parallel and Distributed Systems, IEEE Transactions on*, 10(6):533–548, 1999.