Middleware for Embedded Adaptive Dependability (MEAD)
Real-Time Fault-Tolerant Middleware Support

Priya Narasimhan
Assistant Professor of ECE and CS
Carnegie Mellon University
Pittsburgh, PA 15213-3890

Thomas D. Bracewell
Senior Principal Engineer
Raytheon
Sudbury, MA 01176
Motivation for MEAD

- CORBA is increasingly used for applications, where dependability and quality of service are important
  - The Real-Time CORBA (RT-CORBA) standard
  - The Fault-Tolerant CORBA (FT-CORBA) standard

- But ……
  - Neither of the two standards addresses its interaction with the other
  - Either real-time support or fault-tolerant support, but not both
  - Applications that need both RT and FT are left out in the cold

- Focus of MEAD
  - Why real-time and fault tolerance do not make a good “marriage”
  - Overcoming these issues to build support for CORBA applications that require both real-time and fault tolerance
Existing Technologies

- **The Real-time CORBA (RT-CORBA) standard**
  - Scheduling of entities (threads)
  - Assignment of priorities of tasks
  - Management of process, storage and communication resources
  - End-to-end predictability

- **The Fault tolerant CORBA (FT-CORBA) standard**
  - Replication of entities (CORBA objects or processes)
  - Management and distribution of replicas
  - Logging of messages, checkpointing and recovery
  - Strong replica consistency
### Real-Time Systems | Fault-Tolerant Systems
---|---
Requires *a priori* knowledge of events | No advance knowledge of when faults might occur
Operations ordered to meet task deadlines | Operations ordered to preserve data consistency (across replicas)
RT-Determinism $\Rightarrow$ Bounded predictable temporal behavior | FT-Determinism $\Rightarrow$ Coherent state across replicas for every input
Multithreading for concurrency and efficient task scheduling | FT-Determinism prohibits the use of multithreading
Use of timeouts and timer-based mechanisms | FT-Determinism prohibits the use of local processor time
Combining Real-Time and Fault-Tolerance

- Trade-offs between RT and FT for specific scenarios
  - Effective ordering of operations to meet both RT and FT requirements
  - Resolution of non-deterministic conflicts (e.g., timers, multithreading)

- Impact of fault-tolerance and real-time on each other
  - Impact of faults/restarts on real-time behavior
  - Replication of scheduling/resource management components
  - Scheduling (and bounding) recovery to avoid missing deadlines

- For large-scale systems
  - Scalable fault detection and recovery
  - Considering nested (multi-tiered) middleware applications
  - Tolerance to partitioning faults
Architectural Overview

- Use replication to protect
  - Application objects
  - Scheduler and global resource manager

- Special RT-FT scheduler
  - Real-time resource-aware scheduling service
  - Fault-tolerant-aware to decide when to initiate recovery

- Hierarchical resource management framework
  - Local resource managers feed into a replicated global resource manager
  - Global resource manager coordinates with RT-FT scheduler

- Ordering of operations
  - Keeps replicas consistent in state despite faults, missed deadlines, recovery and non-determinism in the system
So, What Do We Want To Tolerate?

- **Crash faults**
  - ✓ Hardware and/or OS crashes in isolation
  - ✓ Process and/or Object crashes

- **Communication faults**
  - ✓ Message loss and message corruption
  - ✓ Network partitioning

- **Malicious faults (commission/Byzantine)**
  - ✗ Processor/process/object maliciously subverted

- **Omission faults**
  - ✓ Missed deadline in a real-time system

- **Design faults**
  - ✗ Correlated software/programming/design errors

---

© 2003 Priya Narasimhan

MEAD: Real-Time Fault-Tolerant Middleware
MEAD (Middleware for Embedded Adaptive Dependability)

- Our RT-FT Architecture
- Why MEAD?
- Legendary ambrosia of the Vikings
- Believed to endow its imbibers with
  - Immortality (dependability)
  - Reproductive capabilities (replication)
  - Wisdom for weaving poetry (cross-cutting aspects of real-time and fault tolerance)
  - Happy and long married life (partition-tolerance)
MEAD: Real-Time Fault-Tolerant Middleware
Resource-Aware RT-FT Scheduling

- Requires ability to predict and to control resource usage
  - Example: Virtual memory is too unpredictable/unstable for real-time usage
  - RT-FT applications that use virtual memory need better support

- Needs input from the local and global resource managers
  - Resources of interest: load, memory, network bandwidth
  - Parameters: resource limits, current resource usage, usage history profile

- Uses resource usage input for
  - Proactive action
    - Predict and perform new resource allocations
    - Migrate resource-hogging objects to idle machines before they start executing
  - Reactive action
    - Respond to overload conditions and transients
    - Migrate replicas of offending objects to idle machines even as they are executing invocations
Proactive Dependability

What if we knew, with some confidence, when a fault was to occur?

Needs input from a fault-predictor (error-log analysis)
- To determine when, and what kinds of, faults can occur
- To schedule fault detection time based on prediction

Needs input from a recovery-predictor
- Offline predictor: Source code analysis for worst-case recovery time
  - Look at each object’s data structures
  - Looks at the object’s containing process and ORB interactions
  - Not comprehensive: unable to predict dynamic memory allocations
- Runtime predictor: Object execution and memory allocation profile
  - Intercepts and observes runtime memory allocations (e.g., object instantiation, library loading), connection establishment, etc.
  - Prepares for the worst-case replica recovery time
Offline Program Analysis

- Application may contain RT vs. FT conflicts
- Application may be non-deterministic

Program analyzer sifts interactively through application code
  - To pinpoint sources of conflict between real-time and fault-tolerance
  - To determine size of state, and to estimate recovery time
  - To determine the appropriate points in the application for the incremental checkpointing of the application
  - To highlight, and to compensate for, sources of non-determinism
    - Multi-threading
    - Direct access to I/O devices
    - Local timers

- Offline program analyzer feeds its recovery-time estimates to the Fault-Tolerance Advisor
Fault-Tolerance Advisor

- Configuring fault tolerance today is mostly ad-hoc

- To eliminate the guesswork, we deployment/run-time advice on
  - Number of replicas
  - Checkpointing frequency
  - Fault-detection frequency, etc.

- Input to the Fault-Tolerance Advisor
  - Application characteristics (using output from Ask Andy)
  - System reliability characteristics
  - System’s and application’s resource usage

- Fault-Tolerance Advisor works with other MEAD components to
  - Enforce the reliability advice
  - Sustain the reliability of the system, in the presence of faults
Fault-Tolerance Advisor

- Middleware
- Application
- Offline program analyzer
- Run-time profile of resource usage
- Fault Tolerance Advisor
- Reliability requirements
- Recovery time
- Faults to tolerate
- RT-FT Schedule
  - Number of replicas
  - Replication style
  - Checkpointing rate
  - Fault detection rate
  - Locations of replicas
- Operating system,
  - Network speed/type,
  - Configuration,
  - Workstation speed/type

© 2003 Priya Narasimhan
Mode-Driven Fault-Tolerance

Most applications have multiple modes of operation
- Example: an unmanned aerial vehicle (UAV) might be modal
  - Surveillance mode
  - Target recognition mode
  - Tracking mode
  - Feedback/Control mode

Each mode might require different fault-tolerance mechanisms
- The critical elements in the path might differ
- The resource usage might differ, e.g., more bandwidth used in some modes
- The notion of distributed system “state” might be different

MEAD aims to provide the “right mode-specific fault-tolerance”
- Based on the Fault-Tolerance Advisor’s inputs
- In response to (omens heralding) mode changes
Looking Ahead ……

■ OMG RT-SIG in the process of drafting an RFP for RT-FT CORBA

■ Consider (and seek means to reconcile) the fundamental conflicts/tensions between real-time and fault-tolerance

  ▼ To avoid point solutions that might work well, but only for well-understood applications, and only under certain constraints

  ▼ To allow for systems that are subject to dynamic conditions, e.g., changing constraints, new environments, overloads, faults, ……

■ Expose interfaces that support the

  ▼ Capture of the application’s fault-tolerance and real-time needs

  ▼ Tuning of the application’s fault-tolerance and real-time configurations

  ▼ Query of the provided “level” of fault-tolerance and real-time

  ▼ Scheduling of both real-time and fault-tolerance (fault-detection, fault-recovery and fault-forecasting) activities
Summary

- Resolving trade-offs between real-time and fault tolerance
  - Ordering of tasks to meet replica consistency and task deadlines
  - Bounding fault detection and recovery times in asynchronous environment
  - Estimating worst-case performance in fault-free, faulty and recovery cases

- MEAD’s RT-FT middleware support
  - Tolerance to crash, communication, timing and partitioning faults
  - Resource-aware RT-FT scheduler to schedule recovery actions
  - Proactive dependability framework
  - Fault-tolerance advisor to take the guesswork out of configuring reliability
  - Offline program analysis to detect, and to compensate for, RT-FT conflicts

- Ongoing research and development with RT-CORBA and RT-Java

- Intention to participate in the standardization efforts of the OMG
For More Information on MEAD

http://www.ece.cmu.edu/~mead

Priya Narasimhan
Assistant Professor of ECE and CS
Carnegie Mellon University
Pittsburgh, PA 15213-3890
Tel: +1-412-268-8801
priya@cs.cmu.edu