OMG Real-Time and Distributed Object Computing Workshop, July 2002, Arlington, VA

Providing Real-Time and Fault Tolerance for CORBA Applications

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

Sponsored in part by the CMU-NASA High Dependability Computing Program (HDCP)
Outline

- Motivation
- Two standards for quality of service in CORBA
  - Real-Time CORBA and Fault-Tolerant CORBA
- Conflicts between real-time and fault tolerance
- Trade-offs between real-time and fault tolerance
- Resolving the trade-offs
  - Architecture
  - Mechanisms
- Conclusion
Motivation

- 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 talk
  - 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
Quality of Service for CORBA Applications

- **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
The RT-CORBA Standard

CORBA Client
  RT-Current
  RT-ORB
  Priority Mapping

ORB

Scheduling Service

CORBA Server
  Threadpool
  POA
  RT-POA
  RT-ORB
  Priority Mapping
End-to-End Predictability

- The most important property of an RT-CORBA system
- Priorities attached to threads (execution entities) and invocations
  - Maps to native priorities on the operating system
- Bounds on temporal properties of application
  - Bounded message transmission latency across network
  - Bounded message processing time within ORB and application
- Schedule of various tasks computed ahead of time (offline)
  - Schedule respects task priorities and task deadlines
  - Fixed-priority scheduling
- Priority banding
  - Multiple client-to-server connections, each at a different priority
  - Client-dictated or server-dictated priority
The FT-CORBA Standard

Fault Notifier

Fault Detector

Replication Manager

Client

Server

Factory

CORBA ORB

Logging Mechanism

Recovery Mechanism

Fault Detector

Recovery Mechanism

Logging Mechanism

create_object()

set_properties()

create_object()

notifications

fault reports

is_alive()
Strong Replica Consistency

- The most important property of an FT-CORBA system
- Requires deterministic behavior of application objects
- Guarantees on message transmission and delivery
  - Same sequence of messages in the same order
  - No loss of messages over the communication medium
  - No delivery of duplicate invocations or responses
- State transfer to new and recovering replicas
- Essential for both active and passive replication
  - Debunks the myth that passive replication can cure non-determinism
# Real-Time vs. Fault-Tolerance

<table>
<thead>
<tr>
<th>Real-Time Systems</th>
<th>Fault-Tolerant Systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requires <em>a priori</em> knowledge of events</td>
<td>No advance knowledge of when faults might occur</td>
</tr>
<tr>
<td>Operations ordered to meet task deadlines</td>
<td>Operations ordered to preserve data consistency (across replicas)</td>
</tr>
<tr>
<td>Synchronous</td>
<td>Not necessarily synchronous</td>
</tr>
<tr>
<td>Multithreading for concurrency and efficient task scheduling</td>
<td>Determinism prohibits the use of multithreading</td>
</tr>
<tr>
<td>Use of timeouts and timer-based mechanisms</td>
<td>Determinism prohibits the use of local processor time</td>
</tr>
</tbody>
</table>
Real-Time vs. Fault-Tolerance - 1

- RT and FT communities disagree even on basic terminology

- Determinism in the real-time sense
  - Equivalent to predictability
  - Real-time invocation is deterministic if its execution and processing times are bounded and predictable ahead of time
  - Lack of RT determinism can result in missed deadlines

- Determinism in the fault tolerance sense
  - Equivalent to reproducibility
  - Fault-tolerant invocation is deterministic if its execution, by different replicas starting from the same initial state, on different processors, produce the same state changes and the same responses
  - Lack of FT determinism can result in replica inconsistency
Real-Time vs. Fault-Tolerance - 2

- Real-time systems use multi-threading
  - To allow concurrent tasks to execute simultaneously

- Multi-threading is problematic for a fault-tolerant system
  - Unrestricted multi-threading can lead to non-determinism
  - Server with two replicas S1 and S2 on two different processors
  - S1 and S2 might run two tasks on two different concurrent threads
  - Threads modifying shared state within the server can lead to inconsistency
  - Yes, shared state exists, inside the ORB (if not in the application)!
  - Need special scheduler to enforce single-threading for determinism

- Task management
  - Multithreading for task scheduling vs. single-threading for determinism
Real-Time vs. Fault-Tolerance - 3

- **Real-time systems use the notion of wall-clock time**
  - Timeouts and timers used to finesse real-time consensus issues
  - Clients can run a timeout if server doesn’t respond in time

- **Wall-clock time is problematic in a fault-tolerant system**
  - Use of timeouts and timers can lead to non-determinism & inconsistency
  - Replicated (middle-tier) client with two replicas C1 and C2
  - C1’s and C2’s timeouts might expire at different times
  - C1 might think operation missed its deadline; C2 might think otherwise
  - Fault-tolerant systems use clock synchronization & global time service

- **Time management**
  - Maintaining determinism without making global time service a bottleneck
Real-Time vs. Fault-Tolerance - 4

- **Ordering in the real-time sense**
  - Tasks and invocations ordered to meet application deadlines

- **Ordering in the fault tolerance sense**
  - Tasks and invocations ordered to meet replica consistency

- **What if the two orders conflict?**
  - Processor P1 hosts replicas of objects A, B and C
  - Processor P2 hosts replicas of objects A and D
  - Schedules on the two processors might vary with current resources
  - P1’s replica of A and P2’s replica of A might see different orders

- **What if different machines need different task mixtures?**
  - Some tasks ordered *a la* real-time; others ordered *a la* fault tolerance
Real-Time vs. Fault-Tolerance - 5

- **Real-time assumes mostly synchronous operation**
  - Events, tasks, operations known ahead of time
  - Bounded latencies, bounded response time

- **Fault tolerance considers asynchronous environment**
  - Distributed asynchronous system
  - Unbounded latency, unbounded response time, unreliable fault detection

- **Fault tolerance assumes inherent unpredictability**
  - Faults cannot be predicted ahead of time; they are asynchronous events
  - What if faults “upset” the pre-computed real-time schedule?

- **Can we get synchronous operation in an asynchronous setting?**
  - Especially in the presence of transient faults
Real-Time vs. Fault-Tolerance - 6

- **Real-time requires bounded operation time**
  - What about operations such as fault detection and recovery?

- **Time-consuming fault detection**
  - What of common-mode (correlated) faults?
    - Crash of processor hosting 100 objects can lead to 100 fault reports

- **Time-consuming recovery**
  - Recovery must account for ORB, application and infrastructure state
  - Recovery of trivial objects is straightforward (state=simple data structure)
  - What if recovery involves object instantiation?
    - Recovery of a process that requires 100 objects to be instantiated
  - FT-CORBA talks about object-centric recovery; shared state requires process-centric recovery
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 a fault on real-time behavior
  - Impact of recovery (reboot) on real-time behavior
  - Replication of scheduling/resource management components
  - Scheduling (and bounding) recovery to avoid missing deadlines
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

- **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
RT-FT Scheduler

- Requires ability to predict and to control resource usage
- 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
RT-FT Scheduler

- Requires prediction of faults and of recovery

- Needs input from a fault predictor
  - 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
Conclusion

- **Real-time and fault tolerance don’t always make a good “marriage”**
  - Use of time and multithreading (non-determinism)
  - Ordering of tasks to meet replica consistency and task deadlines
  - Bounding fault detection and recovery times in asynchronous environment

- **RT-FT CORBA architecture requires**
  - Online fault profiler and predictor
  - Online and offline recovery predictor
  - FT-aware real-time scheduler that schedules recovery actions
  - New mechanisms to sanitize non-determinism

- **Ongoing research work with RT-CORBA implementations (TAO and Orbacus) and RTSJ reference implementation (Timesys)**
Thank You!

http://www.cs.cmu.edu/~priya

priya@cs.cmu.edu

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