Versatile Dependability

Tudor Dumitraș, Priya Narasimhan

Carnegie Mellon University
Pittsburgh, PA 15213-3890
Outline

- Motivation
- Fault-tolerance vs. real-time trade-offs
- Versatile dependability
  - Definition
  - Metrics
- Status of the implementation
  - Replication design space exploration
- Conclusion
Motivation

- There is a fundamental conflict between fault-tolerance (FT) and real-time (RT)
  - There is also a *de-facto* conflict, which must be dealt with by carefully engineering the systems

- Existing dependable systems are “heavy-weight” and costly
  - Need more flexibility

- Fault-tolerance must become *tunable*
  - So does real-time
## Real-Time Systems vs. Fault-Tolerant Systems

<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>RT-Determinism $\Rightarrow$ Bounded predictable temporal behavior</td>
<td>FT-Determinism $\Rightarrow$ Coherent state across replicas for every input</td>
</tr>
<tr>
<td>Multithreading for concurrency and efficient task scheduling</td>
<td>FT-Determinism prohibits the use of multithreading</td>
</tr>
<tr>
<td>Use of timeouts and timer-based mechanisms</td>
<td>FT-Determinism prohibits the use of local processor time</td>
</tr>
</tbody>
</table>
FT vs. RT trade-off

- The conflict between fault-tolerance and real-time must be seen as a trade-off
  - The trade-off can be tuned

- Dependable systems can be represented in a 3D space:
  - Fault-tolerance axis
  - Real-time axis
  - Resources axis

- Currently, this 3D space is sparsely populated
  - Existing systems are point solutions in this space
Versatile Dependability (1)

- Provides a set of “knobs” to tune the FT vs. RT trade-off
  - Automatic adaptation to the working environment
  - Intervention of a human operator
- Allows to predict the resource usage
- Covers a larger volume of the 3D space
Versatile Dependability (2)
Versatile dependability metrics

- **Fault-tolerance axis:**
  - Strength of fault-model
  - Group communication style
  - FT granularity
  - No. of faults tolerated
  - Frequency of failures
  - Window of vulnerability
  - Overhead of FT

- **Real-time axis:**
  - Fault detection latency
  - Replica launch latency
  - Fault-recovery latency
  - No. of missed deadlines
  - Amount of schedule slack required
  - Scheduling algorithms
Outline

■ Motivation

■ Fault-tolerance vs. real-time trade-offs

■ Versatile dependability
  ▼ Definition
  ▼ Metrics

■ Status of the implementation
  ▼ Replication design space exploration

■ Conclusion
Study of Replication Design Space

- **Replication styles**
  - Active replication
  - Passive replication
    - Cold passive
    - Warm passive
  - Semi-active replication (the leader-follower model)
  - Semi-passive replication

- **Implementation options**
  - State transmission, methods for achieving quiescence, replica consistency, etc.
  - What assumptions are necessary to avoid:
    - atomic multicasts?
    - The “no multithreading” requirement?
  - What kind of applications can run under these assumptions?
Passive & Active Replication

Assumptions
- Per client ordering (GIOP request IDs)
- No ordering across clients
- No leakage of order into the application
- Deterministic app. (for active)
- 2-tiered system

Passive (Warm / Cold)
Active

- \texttt{get\_global\_state()} / \texttt{set\_global\_state()}
  - If warm, can be done more often (with frequency \(f\))
  - If active, done only on launch of replica
  \(\rightarrow\) \(P\) \(A\) (recovery)

- Invocation from interceptor
  - Direct invocation (same process); server must be linked with the \(-rdynamic\) flag for the symbols to be accessible from the interceptor
  \(\rightarrow\) \(P\) \(A\) (recovery)

- State transmission (reliable)
  - Reliable broadcast to all passive replicas (if warm) \(\rightarrow\) SPREAD
  - Reliable storage
  \(\rightarrow\) \(P\) \(A\) (recovery)

- Quiescence
  - Achieved by matching CORBA sequence numbers of request and response (using GIOP parser) \(\rightarrow\) brute force approach
  - Achieved inside the application, using mutex (\texttt{get\_state()} and all operations that change the state need to acquire the mutex)
  \(\rightarrow\) \(A\) \(P\)

- Client request queuing
  - Only messages between checkpoints need to be queued
  - Do we need a globally unique client ID? Inside msg.? \(\rightarrow\) \(A\) \(P\)

- Primary reelection
  - Rotating coordinator
  - First available file descriptor
  \(\rightarrow\) \(P\)

- Duplicate detection & suppression
  - Done at the interceptor level; needed for temporal redundancy
  \(\rightarrow\) \(A\) \(P\)

- Fault detection
  - \texttt{close()} system call

- Finding the replicas
  - From the configuration options

- Configuration options
  - From environment variables
  \(\rightarrow\) Assume synchronized clocks (e.g. using NTP)
  \(\rightarrow\) Keep state for each client in the server (separate msg. queues for each client)
  \(\rightarrow\) Use logical clocks (can do with scalar logical clocks ?)

Is it possible to relax the “no multithreading” requirement?
Conclusion

- Versatile dependability gives knobs to tune the FT vs. RT trade-off
  - Study of fundamental relations between FT and RT
  - Implementation of a prototype

- We have started by exploring the replication design space
  - How do the implementation choices affect FT, RT, resources?

- Open questions
  - What kind of high-level knobs should we implement?
    - E.g. availability, survivability, level of replica consistency
  - What kind of applications could use versatile dependability?
    - E.g. enterprise middleware, embedded systems