Proactive, Resource-Aware, Tunable Real-time Fault-tolerant Middleware

Priya Narasimhan

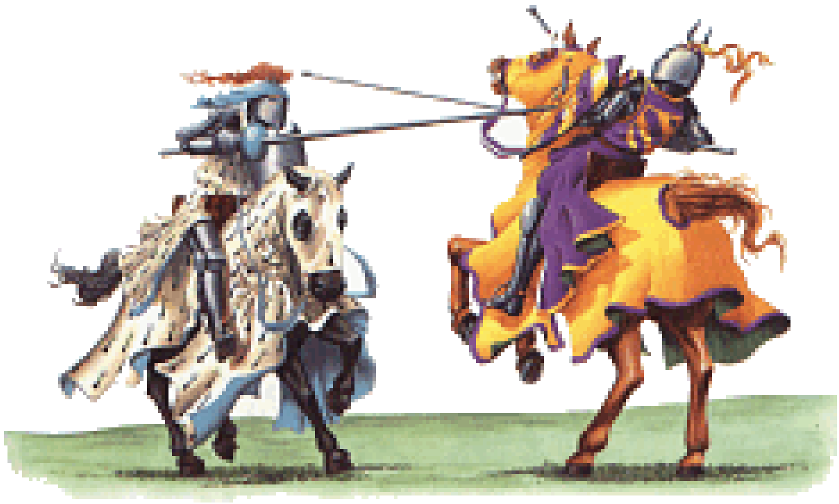
T. Dumitraş, A. Paulos, S. Pertet, C. Reverte, J. Slember, D. Srivastava

Carnegie Mellon University

Problem Description

- **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

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 ⇨ Bounded predictable temporal behavior	FT-Determinism ⇨ 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



Technology Development

- **Trade-offs between RT and FT for specific scenarios**
  - Effective ordering of operations to meet both RT and FT requirements
  - Proactive fault-tolerance to meet both RT and FT requirements
- **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
- **Additional features**
  - Tools for (re-)configuring fault-tolerance in an resource-aware manner
  - Tools for the structured injection of different kinds of faults
  - Metrics (and measurement techniques) for objectively evaluating fault-tolerance

MEAD Architectural Overview

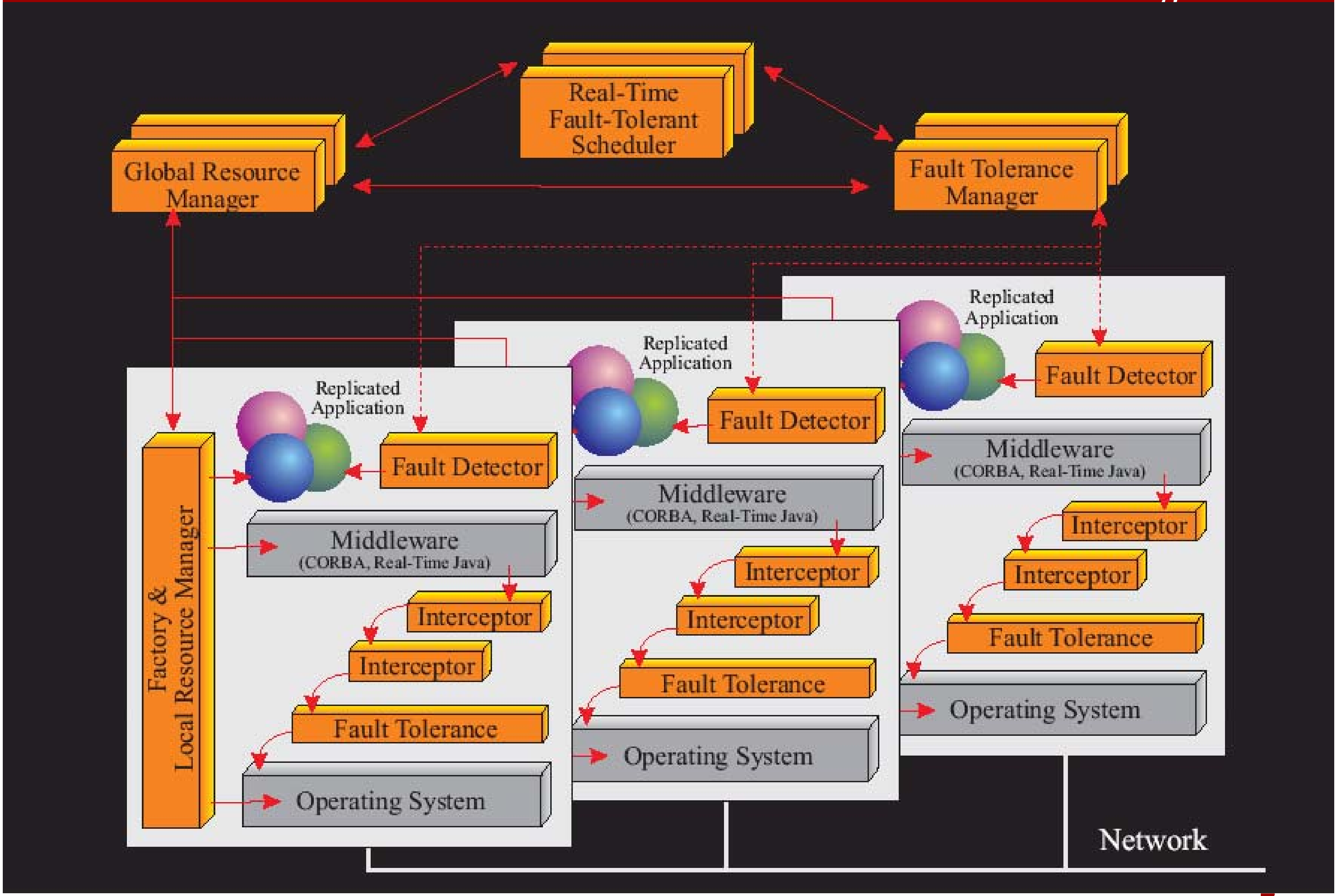
- **Use replication to protect**
  - Application objects
  - Scheduler and global resource manager
- **Special RT-FT scheduler**
  - Real-time resource-aware scheduling service
  - Fault-tolerant-aware to decide when to initiate recovery
- **Hierarchical resource management framework**
  - Local resource 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

Fault Model for MEAD

- **Crash faults**
  - ✓ Hardware and/or OS crashes in isolation
  - ✓ Process and/or Object crashes
- **Communication faults**
  - ✓ Message loss and message corruption
  - ✓ Network partitioning
- **Resource-exhaustion faults**
  - ✓ Running out of memory, CPU, bandwidth
- **Omission faults**
  - ✓ Missed deadline in a real-time system

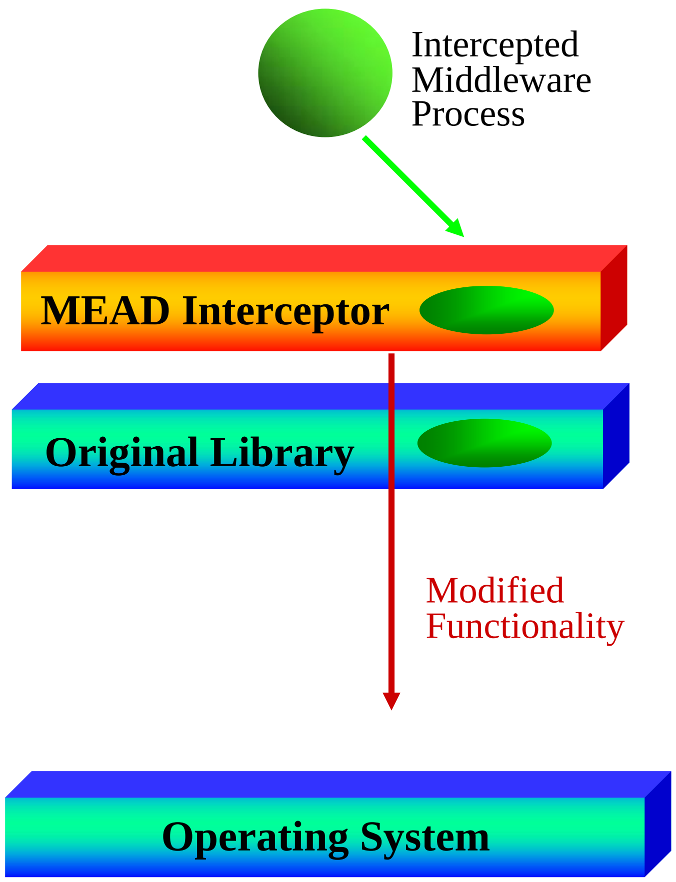
Fault Model
Kinds of faults that MEAD is designed to tolerate

- **Malicious faults (commission/Byzantine) – not in current version**
  - ✗ Processor/process/object maliciously subverted


Real-Time
Fault-Tolerant
Scheduler
Global Resource
Manager
Fault Tolerance
Manager
Replicated
Application
Fault Detector
Middleware
(CORBA, Real-Time Java)
Interceptor
Interceptor
Fault Tolerance
Operating System
Factory &
Local Resource Manager
Replicated
Application
Fault Detector
Middleware
(CORBA, Real-Time Java)
Interceptor
Interceptor
Fault Tolerance
Operating System
Replicated
Application
Fault Detector
Middleware
(CORBA, Real-Time Java)
Interceptor
Interceptor
Fault Tolerance
Operating System
Network

MEAD Interceptors for Transparency

- **Interceptor - User-level extension to add new capabilities; works with**
  - Unmodified operating systems
  - Unmodified ORBs and JVMs
  - Unmodified applications
- **MEAD employs interception to**
  - Plug in fault-tolerance at run-time
  - Profile application for patterns of communication and resource usage
  - Provide run-time support for different "aspects" of FT and RT
  - Mix-and-match "serial" and "parallel" interceptors, at run-time, to achieve specific goals

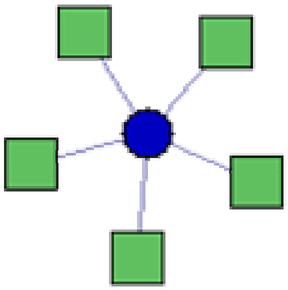


Proactive Fault-Tolerance: Overview

- **Involves predicting, with some confidence, when a failure might occur, and compensating for the failure even before it occurs**
  - For instance, if we knew that a processor had an 80% chance of failing within the next 5 minutes, we could perform process-migration
- **Our goal in MEAD is to**
  - Lower the impact faults have on real time schedules
  - Implement proactive dependability in a transparent manner
- **Proactive dependability has two aspects:**
  - Fault prediction: Reducing the unpredictable nature of faults
  - Proactive recovery: Reducing fail-over times and number of failures experienced at the application-level (primary focus in MEAD)
- **Complements, but does not replace, the classical reactive fault-tolerance schemes since we cannot predict every fault**

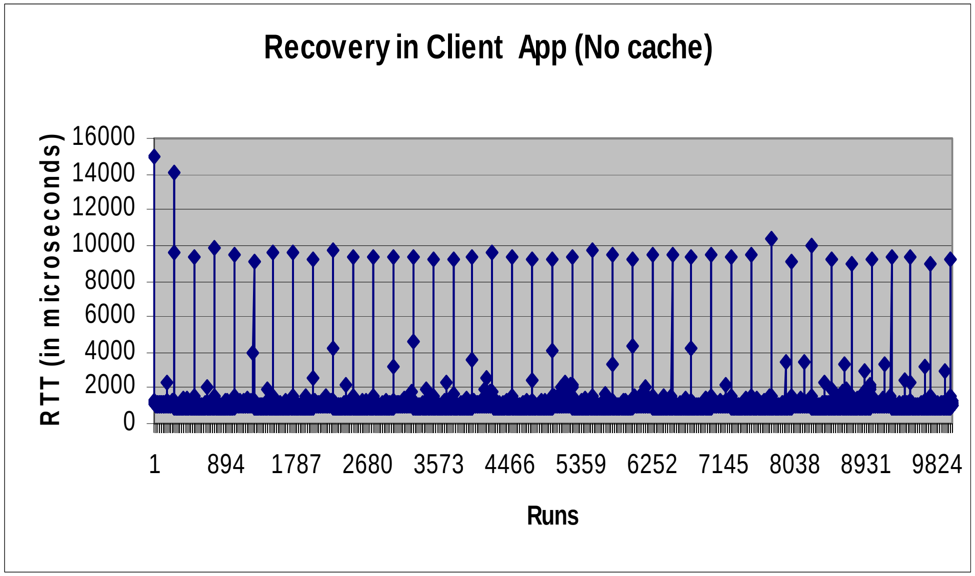
Experimental Results

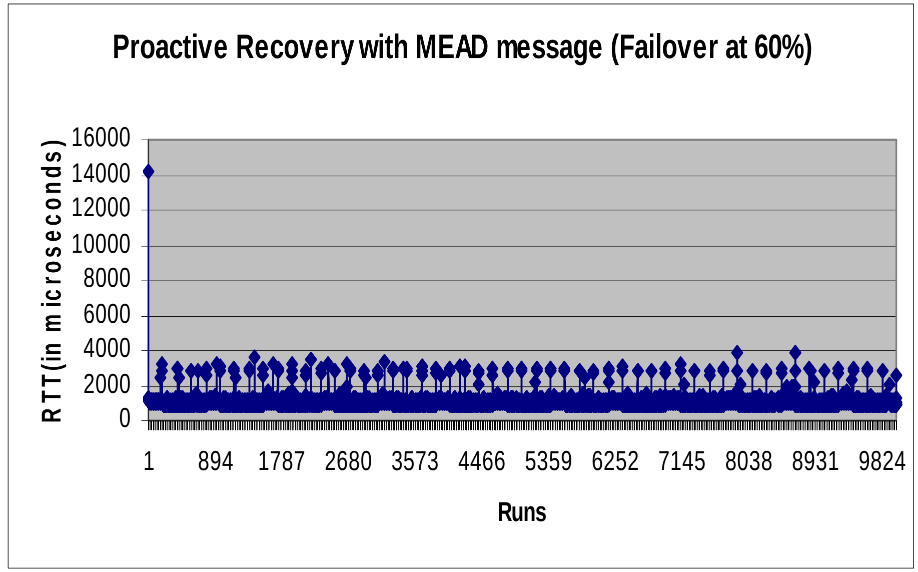
- **Setup:**
  - Run on 5 PC850 Emulab nodes (100 Mbps LAN and UAV-RHL73-STD operating system)
  - 4 server replicas,1 client
    - Warm passive replication with memory leak fault
    - Used round robin algorithm for failover



- **Recovery schemes**
  - At Client
    - Reactive recovery at client (with and without cached object references)
  - At Interceptor
    - On abrupt failure, client contacts group for next server reference
    - Proactive failover (i.e. failover when resource usage < 100%) send either:
      - GIOP LOCATION_FORWARD message
      - Custom MEAD message that causes client to reconnect to next replica transparently at client

Summarized Results



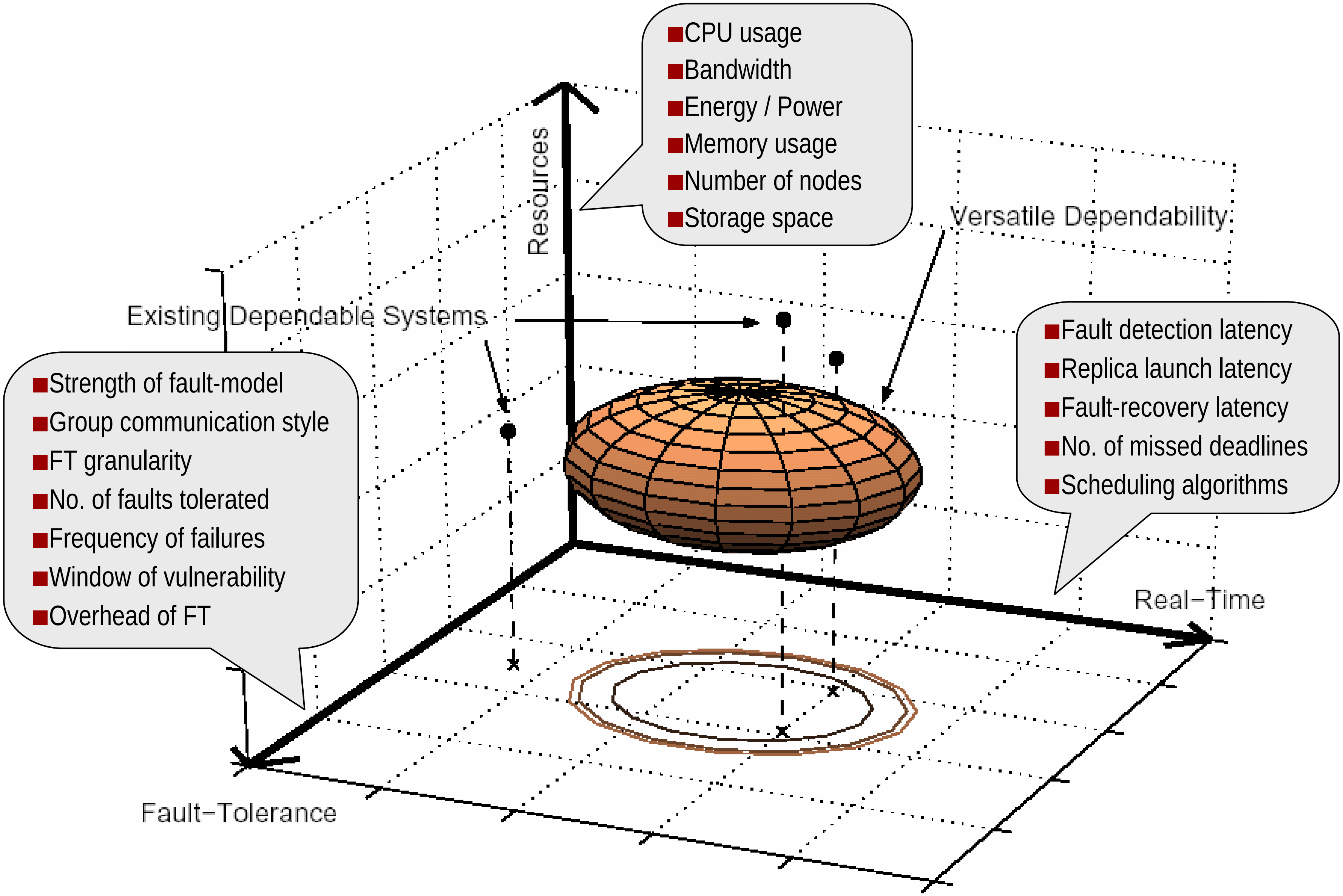


- **Overhead in fault-free runs:**
  - GIOP LOCATION_FORWARD: 61%, MEAD message Overhead: 5% ☺
- **Approximate failover times:**
  - Reactive recovery at client app (no cache) 9400 µs (COMM_FAILUREs)
  - Proactive recovery with GIOP message: 11000 µs
  - Proactive recovery with MEAD message: 3400 µs
- **Failures: None visible at the client side in proactive schemes**
  - Reactive schemes have 1:1 correspondence between client and server side

Benefits for Operational Scenarios

- **Provides a framework for proactive recovery that is transparent to the client application**
- **Proactive recovery can:**
  - Significantly reduce failover times, lowering the impact of a failure on real-time schedules
  - Reduce the number of failures experienced at the application level
  - Provide advance warning of failures to other servers “further down the line” (multi-tiered applications)
  - Request the recovery manager to launch new replicas so that a consistent number of replicas are retained in the group (useful for active replication where a certain number of servers are required to reach agreement)
- **Caveat**
  - Not applicable to every kind of fault, of course

Versatile Dependability

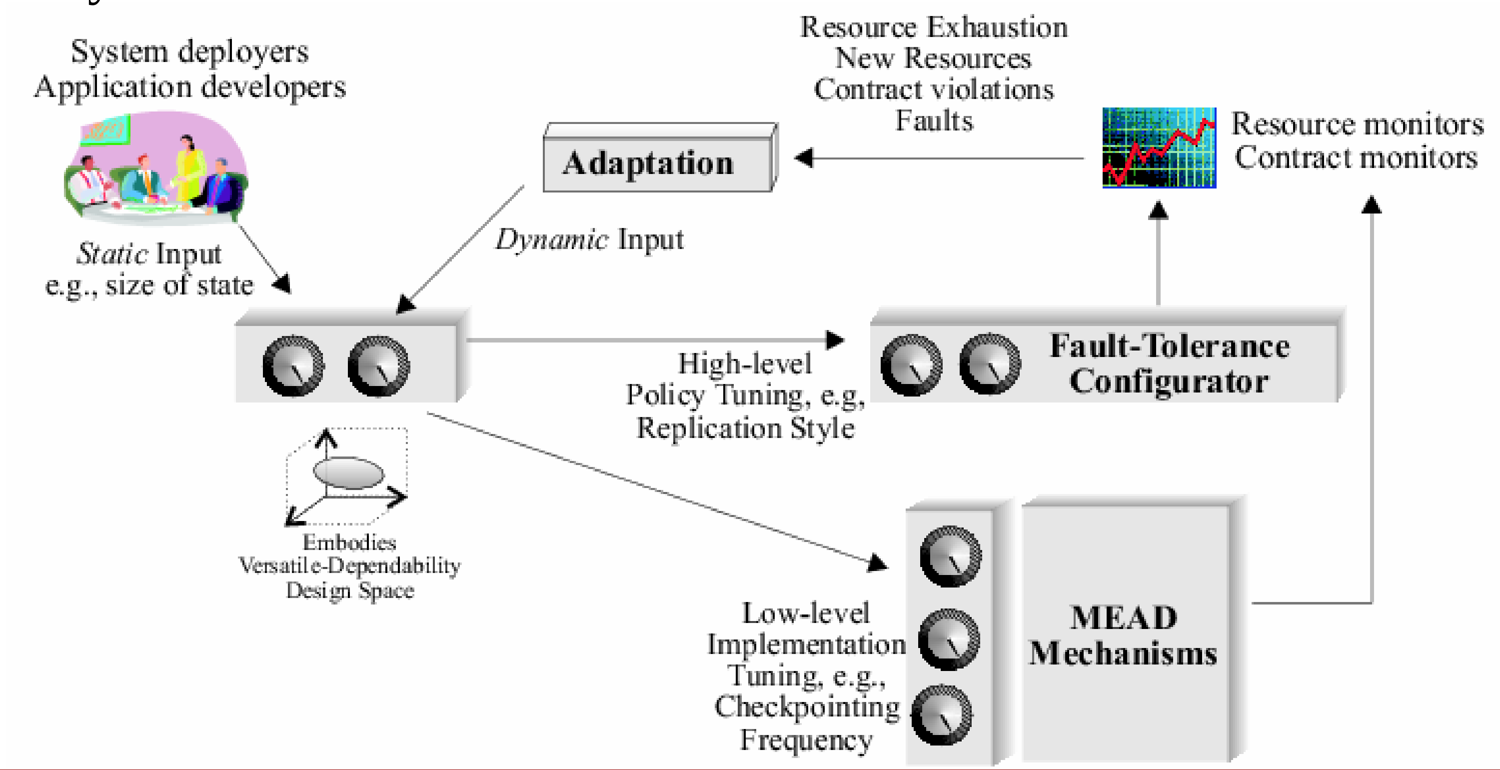


Resource-Aware RT-FT Scheduling

- **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

Versatile Dependability: System Architecture

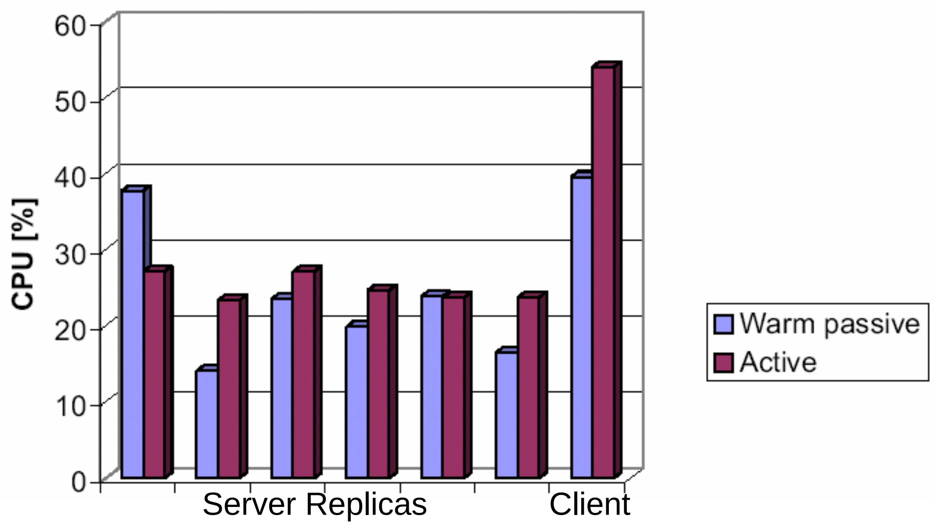
- **Design goals**
  - One infrastructure, multiple "knobs"
  - Quantifiable trade-offs: FT vs. RT. vs. resources (benchmarking)
  - Dynamic adaptation to working conditions
  - Transparency

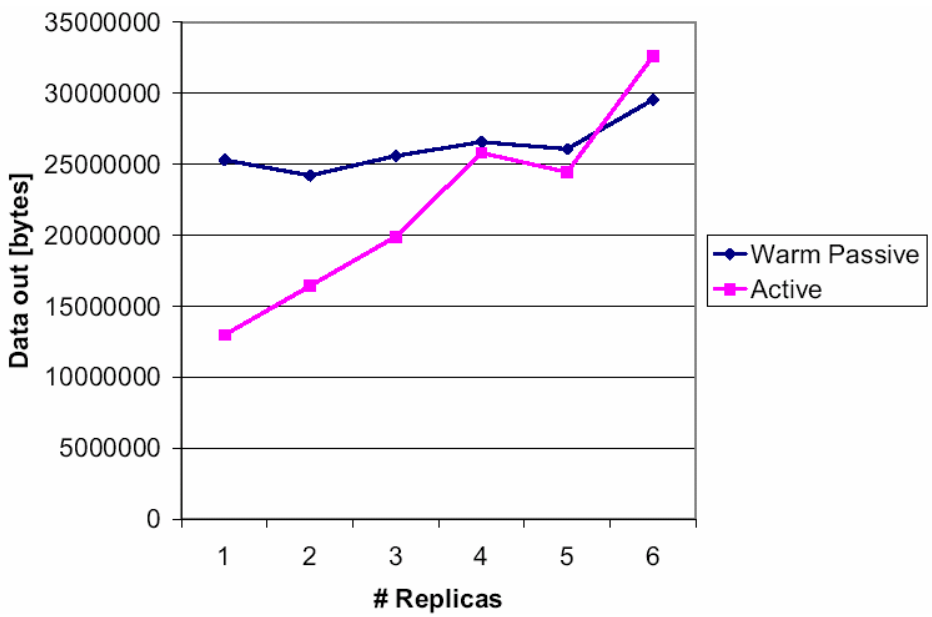


Versatile Dependability: Overheads

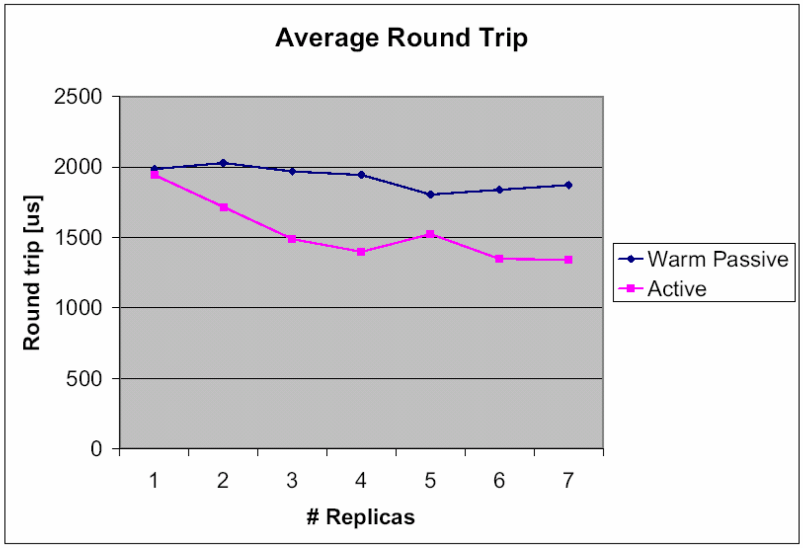
- **Fault-tolerance overhead varies**
  - Active replication has less overhead than warm passive replication
  - Comparing the application with and without replication
    - Overheads are currently in the range of 200%
    - Most of the overhead comes from using a group communication system underneath – we are looking at configuring this better
  - Not yet optimized
- **Resource usage (e.g. bandwidth) varies too**
  - Active
    - Uniformly distributed across replicas
  - Warm passive
    - Not uniform
    - Primary is worse than backups
- **Experiments on reactive recovery**

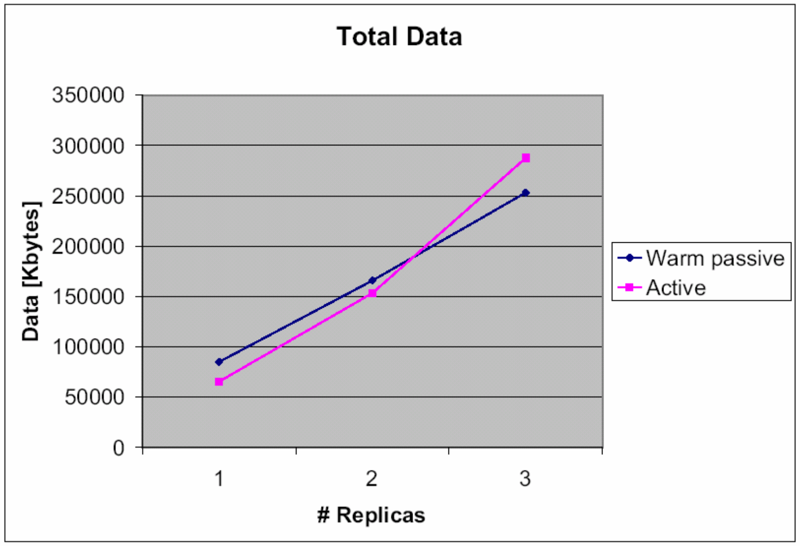
Resource Usage Experiments

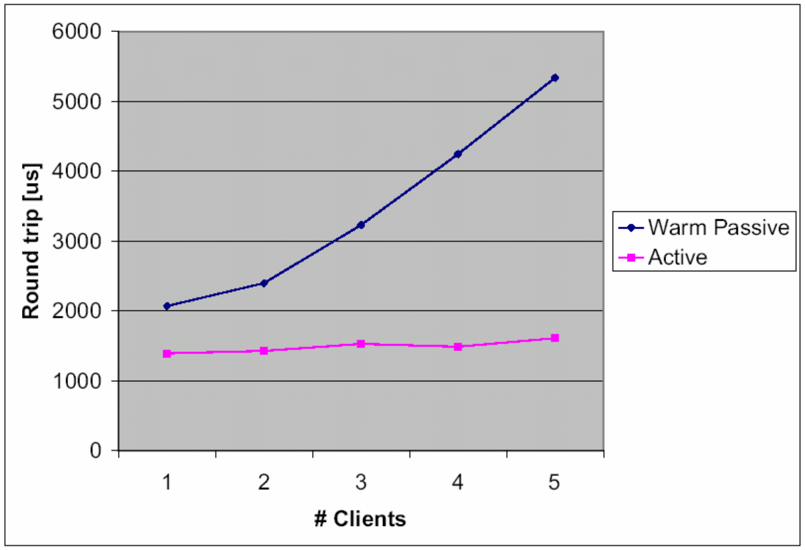


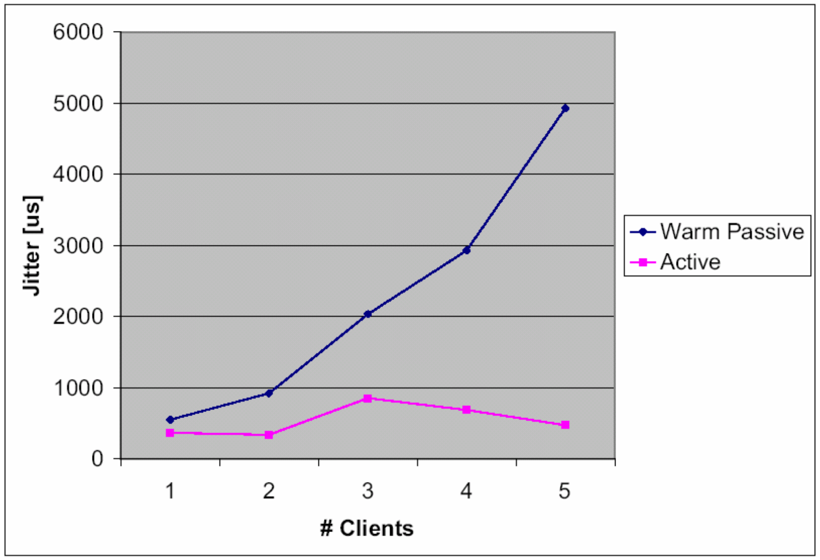


Scalability Experiments (Clients/Replicas)



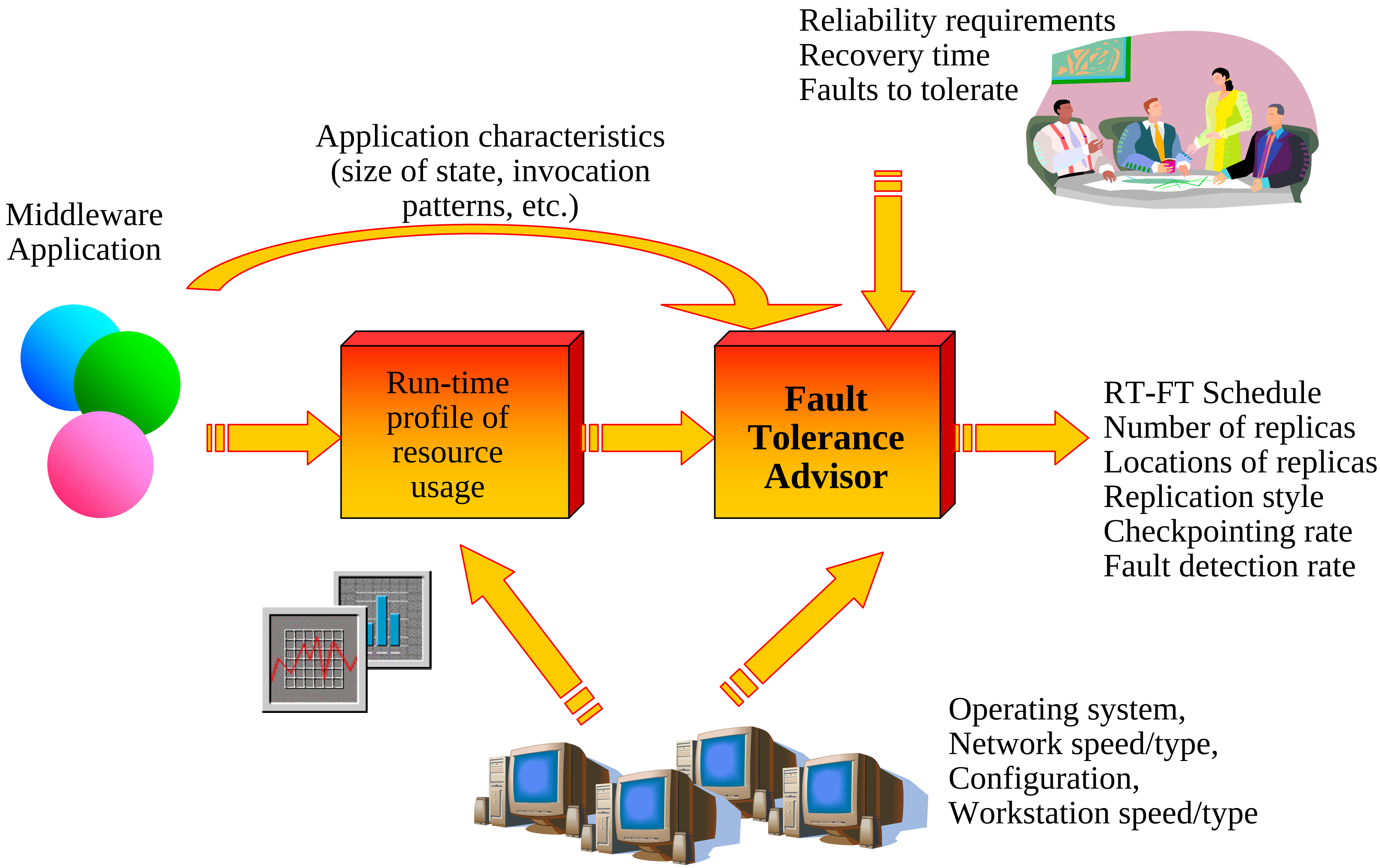






Reactive Fault-Tolerance Experiments

Fault-Tolerance Advisor



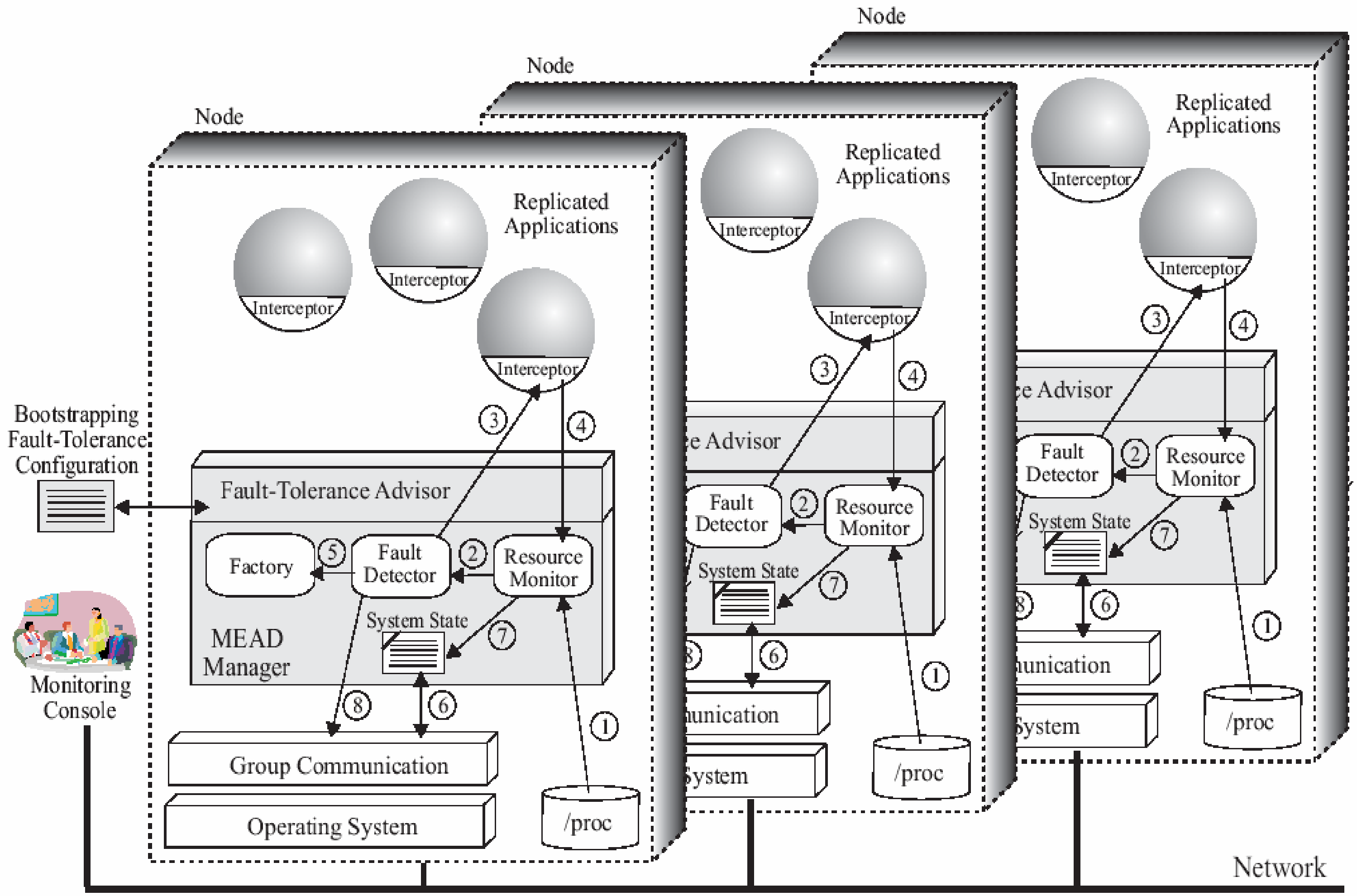
Fault-Tolerance Advisor

- **Fault-tolerance configuration of reliable applications is often ad-hoc, done by hand with little or no optimization**
- **Reliability and performance can be greatly improved by tuning the fault-tolerance parameters**
  - Replication style, checkpointing rate, fault-detection rate, number of replicas, locations of replicas
- **Static fault-tolerance configurations quickly go out of tune as systems change**
- **Dynamic re-tuning can adapt to changing system characteristics**
  - Varying fault rates, system load, resource availability, reliability requirements
- **The Fault-Tolerance Advisor gives deployment-time and run-time advice to tune fault-tolerant applications running over MEAD**

Fault-Tolerance Advisor Architecture

- **MEAD Manager components enforce FT configuration**
  - Factory
  - Fault detector
  - Resource monitor / interceptor library
  - Fault-tolerance advisor interface
- **Fault-Tolerance Advisor**
  - Dynamically tunes fault-tolerance configuration
- **Decentralized architecture**
  - No single point-of-failure
  - Managers are symmetrically replicated on all nodes
    - Synchronized view of system state via Spread group communication bus
    - Managers take local action without requiring coordination
    - Faster recovery, better scalability

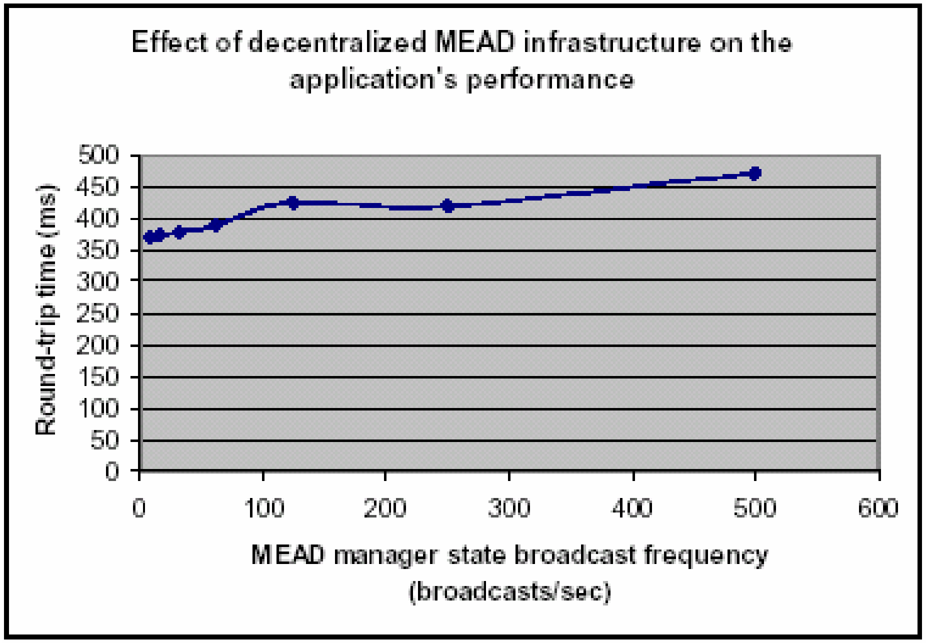
Fault-Tolerance Advisor Architecture



① Capture of CPU and memory resource usage
② Push fault-monitoring
③ Pull fault-monitoring
④ Capture of network resource usage
⑤ Reflexive process re-spawn (fault-recovery)
⑥ State synchronization of MEAD Managers
⑦ Update of local system state
⑧ Fault-notification broadcast to MEAD Managers

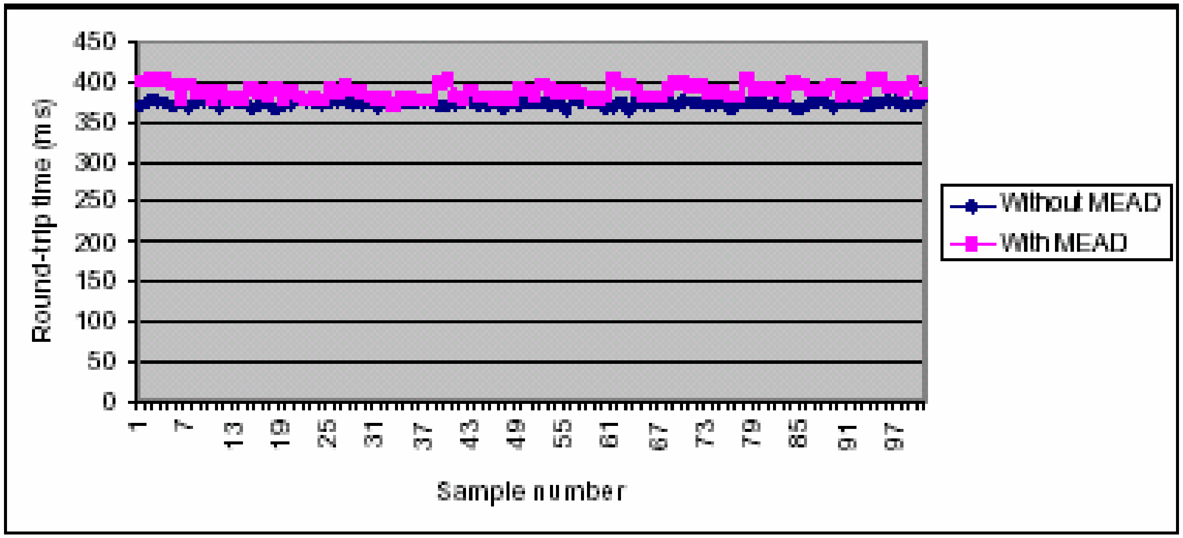
Resource Monitoring Results

- **Tested the scalability of resource monitoring software**
  - Measured round-trip time of batch image transfers
- **Observed good scaling of Advisor's monitoring overhead**
  - Linear as advising-update frequency increases
  - Linear as number of nodes increases
  - Linear as network traffic increases



Fault-Tolerance Advising Results

- **Tested the use of interceptor libraries for network monitoring**
  - Monitor library intercepted network system calls
  - Monitored incoming and outgoing network traffic for each process
  - Measured round-trip time of batch image transfers
- **Low performance impact**
  - Minimal increase of average round-trip time (~4%)
  - Minimal increase in jitter



Offline Program Analysis

- **Application may contain RT *vs.* FT conflicts**
- **Application may be non-deterministic**
  - Multi-threading
  - Direct access to I/O devices
  - Local timers
- **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
- **Offline program analyzer feeds its recovery-time estimates to the Fault-Tolerance Advisor**

Summary

- **Resolving trade-offs between real-time and fault tolerance**
  - 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 and timing faults
  - Proactive dependability framework
  - Fault-tolerance advisor to take the guesswork out of configuring reliability
- **Release of MEAD on Emulab**
  - Active and warm passive replication for CORBA
  - Uses the Spread group communication system for underlying transport
- **Ongoing work with fault-tolerant CCM with active replication**

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