A QoS-aware CCM for DRE System Development

Nanbor Wang

Tech-X Corporation

5561 Arapahoe Ave., Suite A

Boulder, CO 80303

nanbor@txcorp.com

Chris Gill

Dept. of Computer Science and Engineering

Washington University

One Brookings Drive

St. Louis, MO 63130

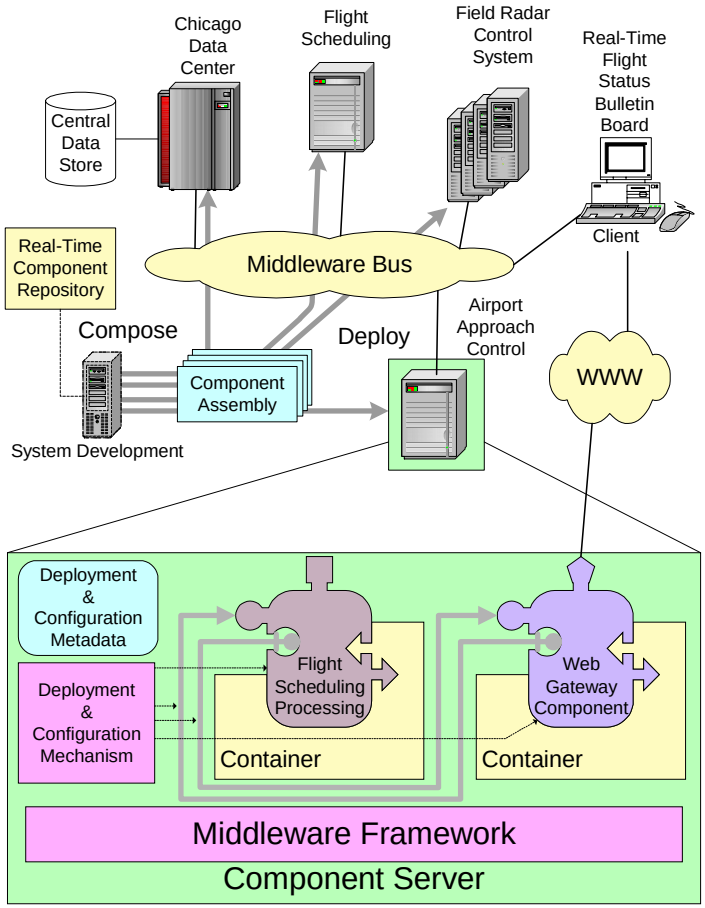
cdgill@cse.wustl.edu

OMG Real-time and Embedded Systems Workshop

Reston, VA, USA July 13, 2004

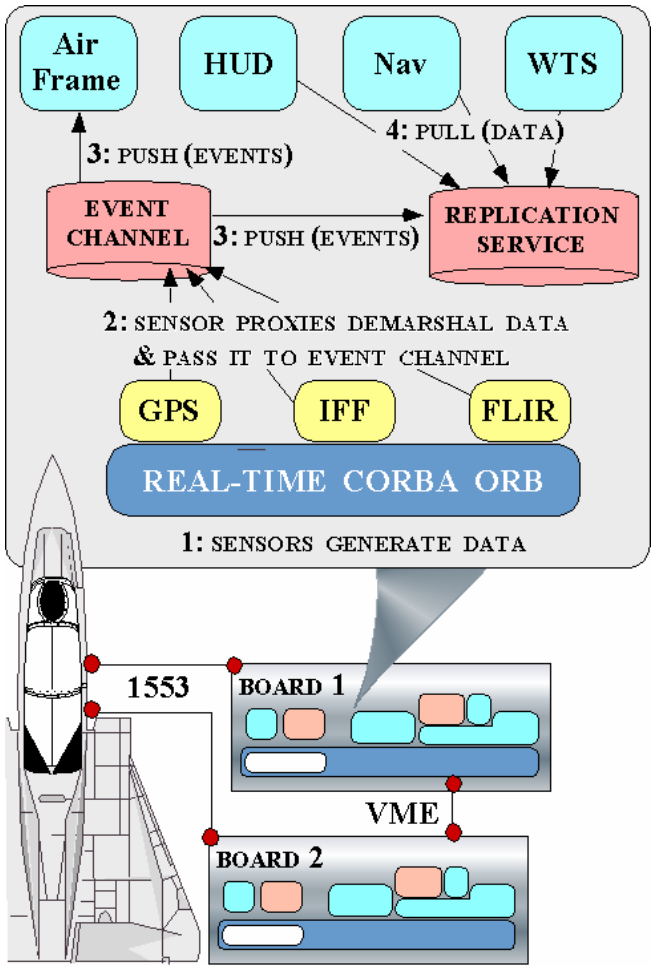
This work was supported in part by the DARPA PCES program, contracts F33615-00-C-3048 and F33615-03-C-4111

Goal: How Can We Combine RT-CORBA and CCM?



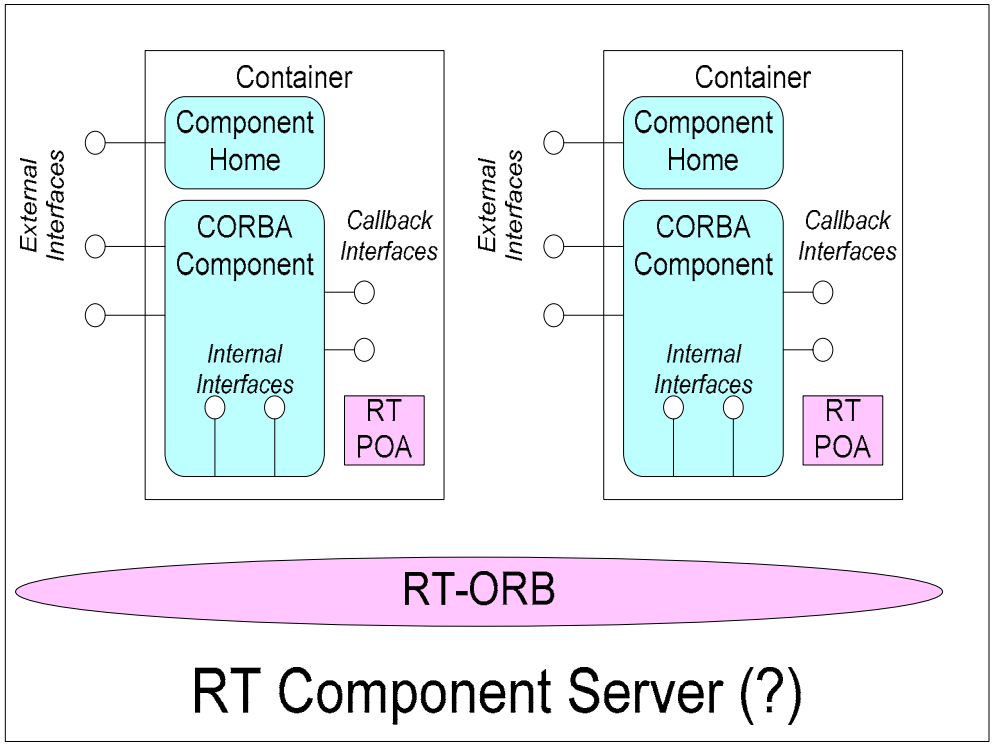
Bringing component-oriented software development paradigm into DRE application domain

Meta-programming Real-time CORBA aspects in CIAO



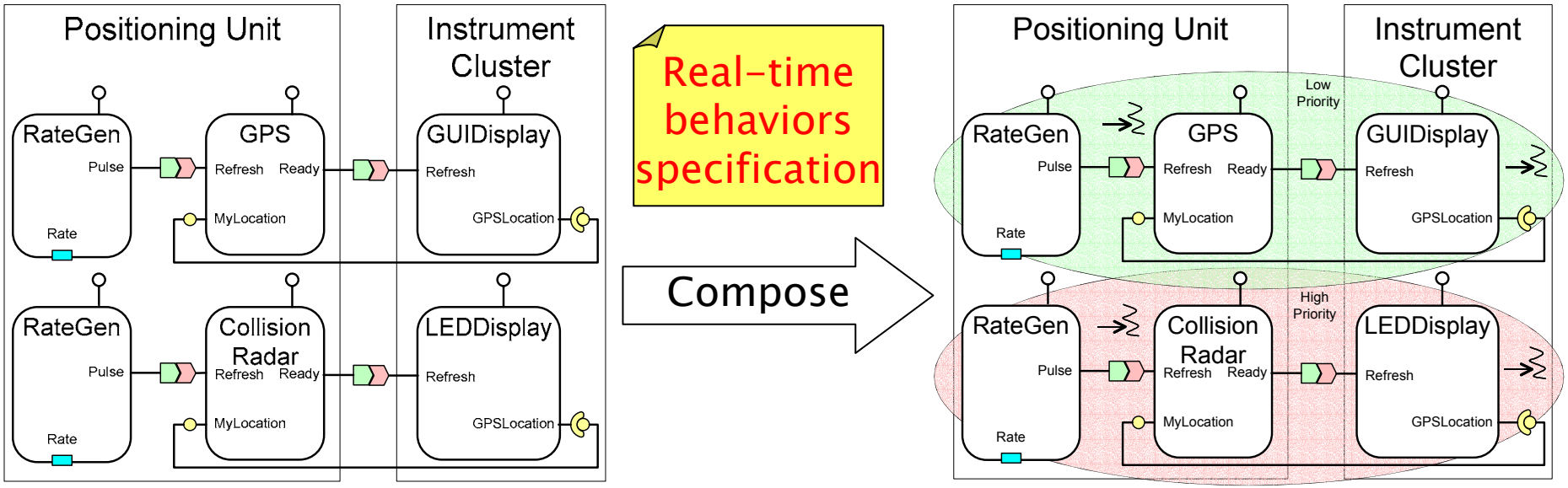
Problem: Current CCM Fails to Address DRE QoS

QoS-enabled CCM ≠
CCM + RT CORBA
Running an RT ORB beneath CCM doesn't make it a QoS-enabled CCM implementation



- Conventional CCM has few mechanisms to specify and enforce QoS policies
  - Especially for DRE systems
  - E.g., deadline, rate, priority
- Consequences:
  - Tight coupling between component implementations and QoS enforcement
  - Coupling between components that QoS aspects cross-cut
  - Difficulty in reusing existing components for different QoS contexts

Solution: Compose Real-Time Aspects in CCM



- A novel architecture to integrate component meta-programming and systemic aspects, e.g., extending CCM atop RT-CORBA features
- Makes real-time behaviors an integral part of the framework so they can be composed into DRE applications

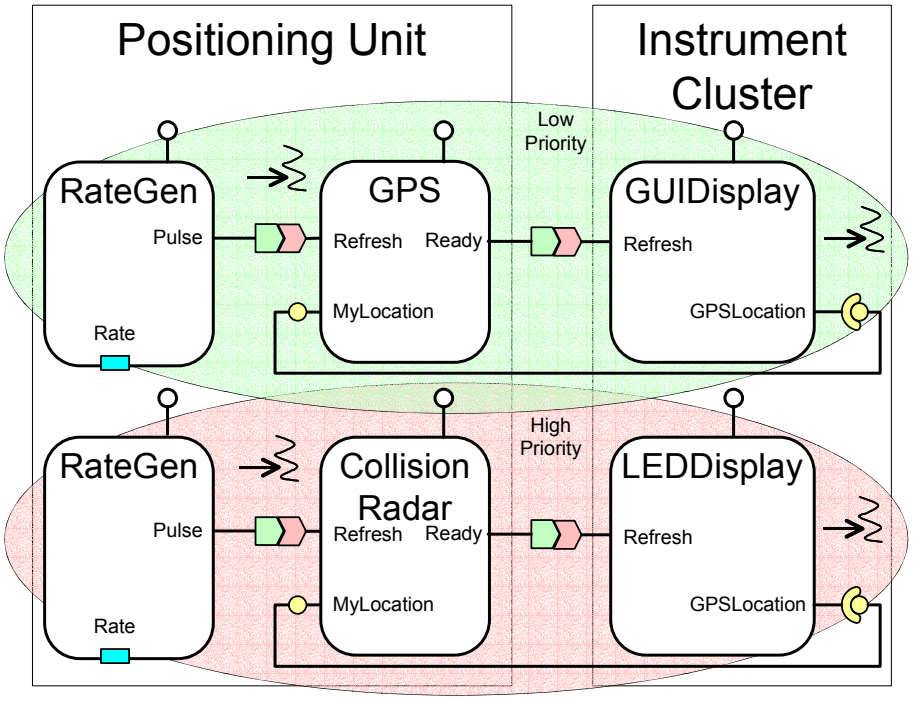
Challenge: Compose Systemic Aspects Effectively

Context:

- Need to enable consistent assembly of systemic aspects
- Need to configure QoS aspects at a variety of granularities
- Configuration information is available at different points in the application lifecycle

Problem:

- Programming QoS aspects imperatively is tedious, error prone, and insufficiently customizable

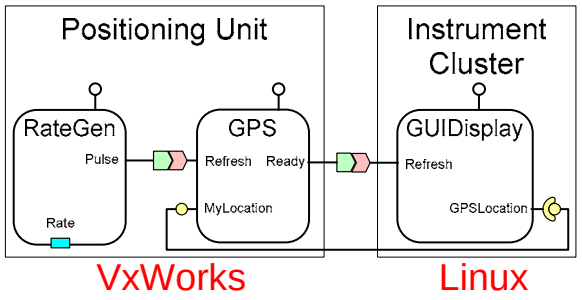


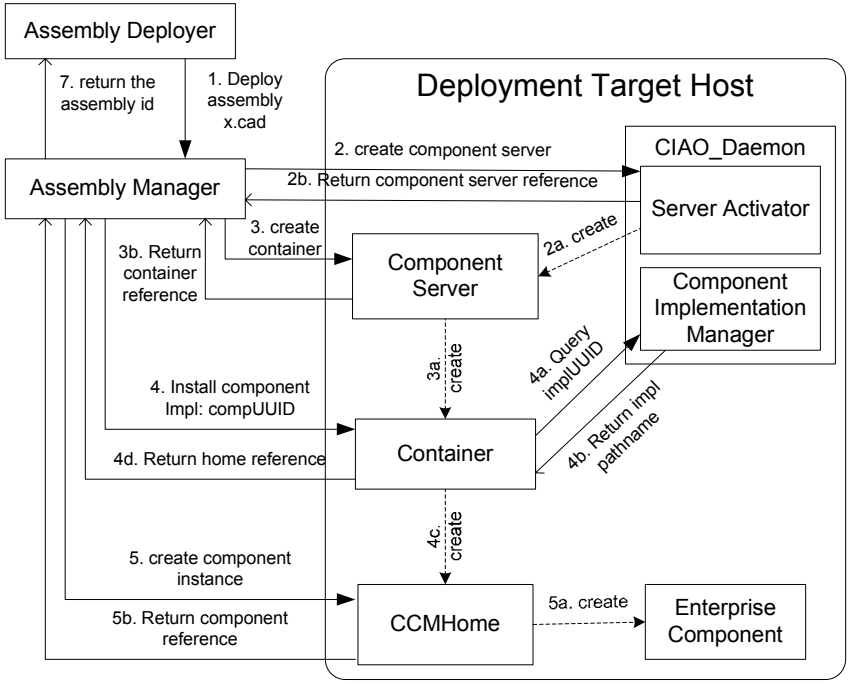
Solution: Metadata for Real-time Aspects

```
<!ELEMENT rtcad_ext
   ( rtresources?,
     rtpolicyset+)>

<!ELEMENT rtresources
   (threadpool |
    threadpoolwithlanes |
    connectionbands)* >

<!ELEMENT rtpolicyset
   (priority_model_policy,
    threadpool_policy,
    banded_connection_policy)+ >
```

- Real-time extension XML descriptors (ext. filename: .rtd)
  - Associated at assembly stage
  - Used by deployment stage
- Declaratively specify:
  - Resources for ORBs
  - Consistent groups of policies
    - for real-time behaviors
    - applied in containers
- Encapsulate all RT policies and resources in one file:
  - Resources global to a component server
  - Sets of policies an instance of component requires

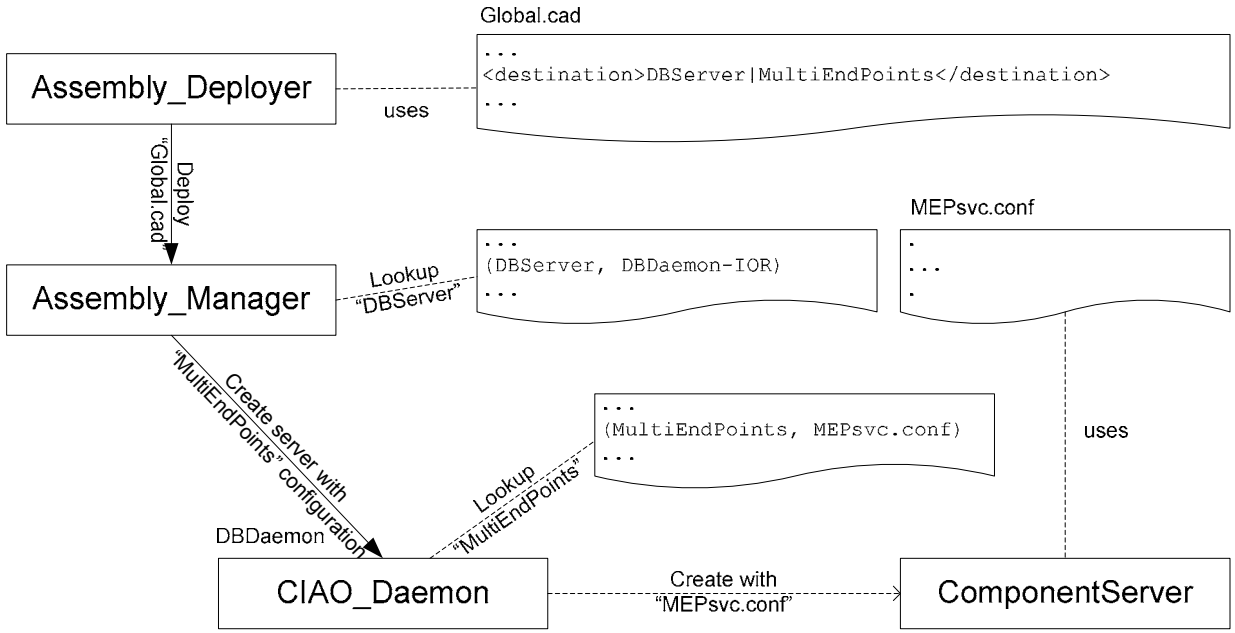
Challenge: Address Platform Diversity





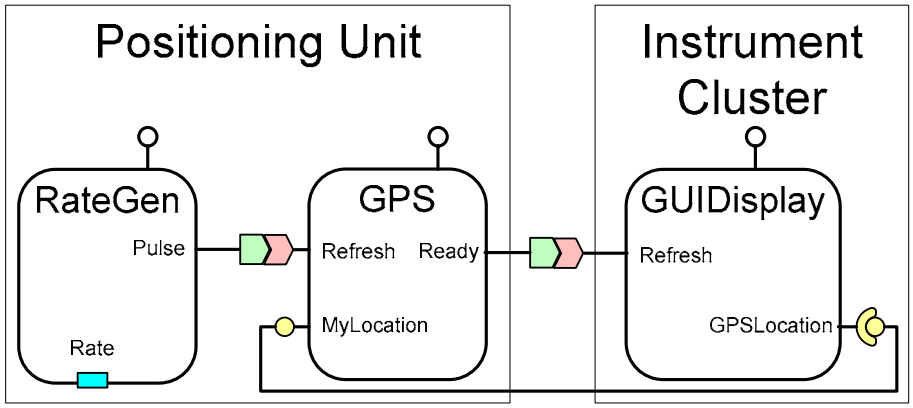
- Context:
  - CCM enables building and configuring distributed applications across network of heterogeneous platforms
  - Real-time systemic behaviors require special configurations to the component server
- Problems:
  - Different configurations are needed to support the same behaviors
  - Different mechanisms are available

Solution: Separate Logical & Physical Configurations



- Document the intentions but defer the decision of actual strategies to some later point
  - Specify only names of configurations initially
  - Select actual configuration upon deployment to a target platform
- Similar approach is used for configuration of deployment topology

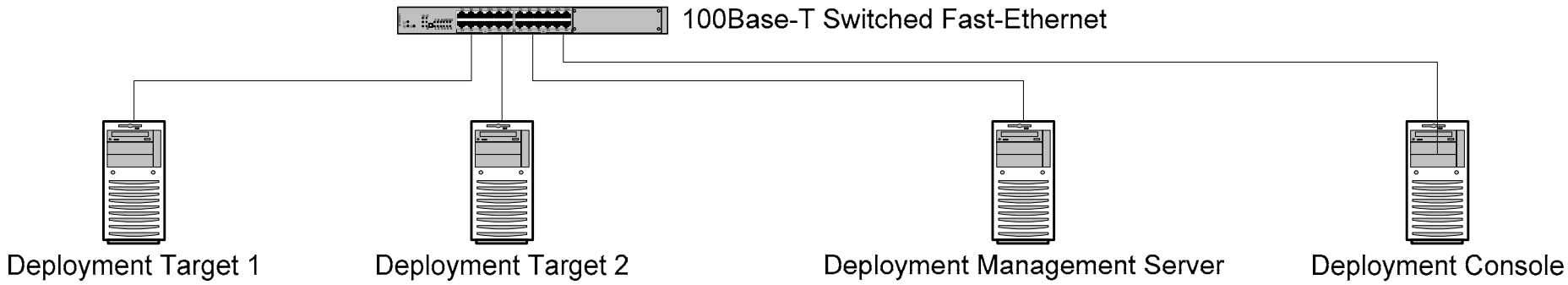
Empirically Evaluating Representative Examples



Experiments:

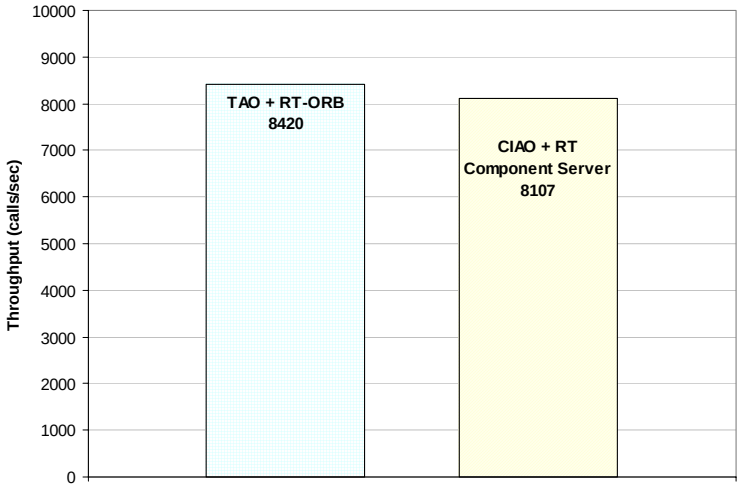
- Based on representative avionic application
- Comparison between same test program built using CIAO vs. TAO
  - Throughput
  - Latency
  - Priority enforcement
- Verify that CIAO can effect desired real-time behaviors
  - Based on canonical RT-CORBA tests for TAO
- Evaluate cost of CCM layer relative to underlying ORB middleware

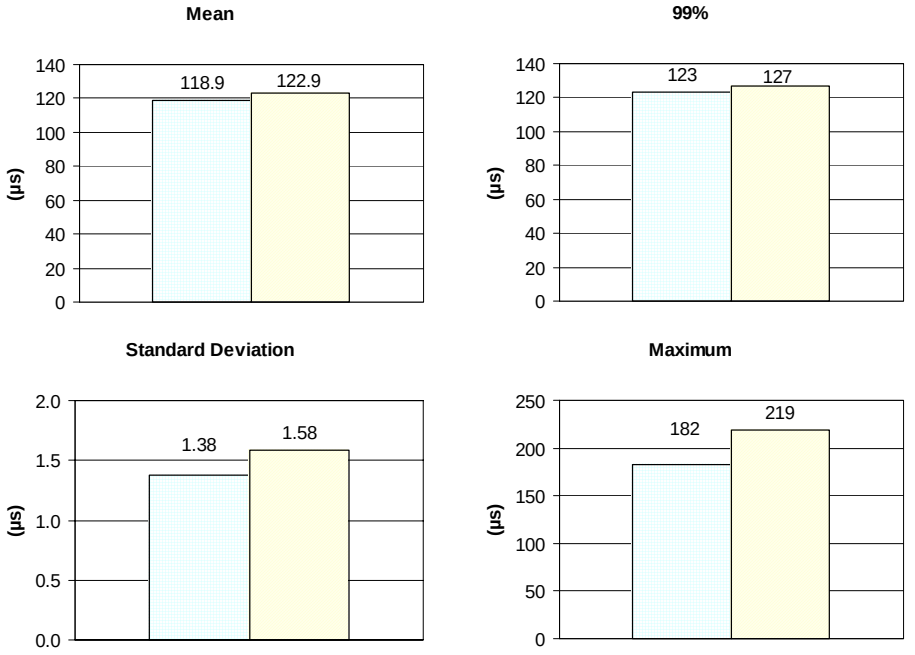
Experiment Configurations



- Linux testbed
  - Two 2.80 Ghz Pentium-4 boxes and two 2.53 Ghz Pentium-4 boxes
  - Most experiments used 2.8 Ghz-pair to execute and 2.53 Ghz-pair for deployment
  - KURT-Linux 2.4.18
  - CIAO 0.3.5 / TAO 1.3.5 / ACE 5.3.5
  - All tests compiled with highest optimization option (-O3)
  - All tests run with SCHED_FIFO scheduling policy

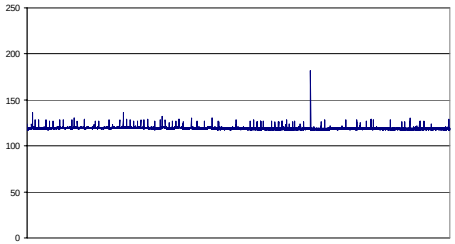
Throughput & Latency with RT Features Enabled



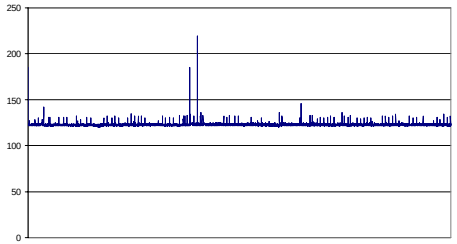


- Slightly higher throughput loss (3.7%) and corresponding latency increase (3.4%)
- Similarly CIAO doesn't affect the jitter based on std-dev, 99% case, or maximum latency

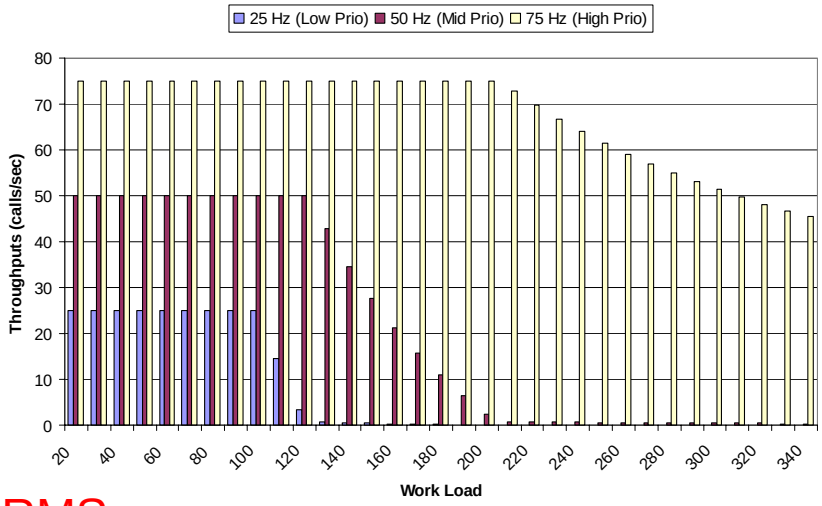
TAO



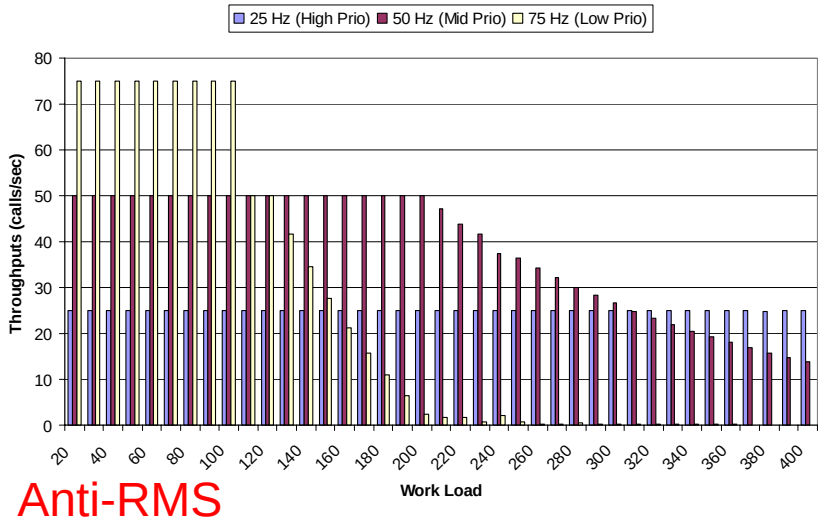
CIAO



Results and Analysis – Separate Threadpools



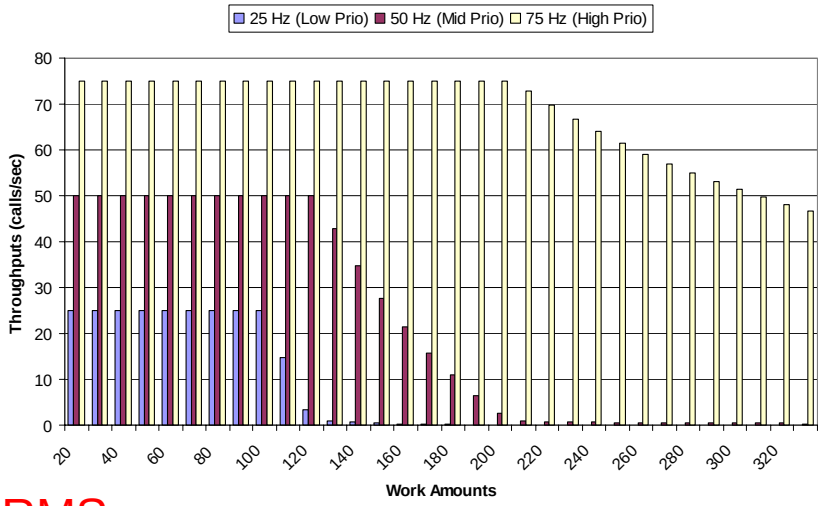
RMS



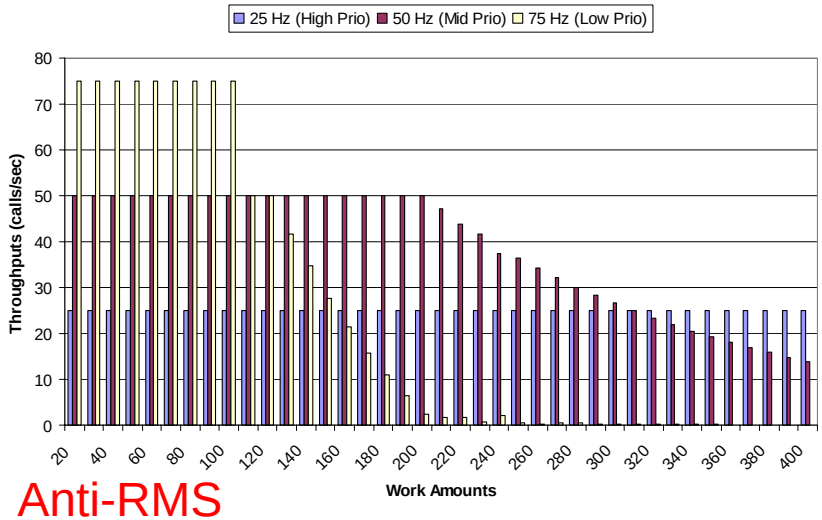
Anti-RMS

- RMS – lower rate (lower priority) controller-worker threads yield to the higher priority threads
- Anti-RMS – higher rate (lower priority) controller-worker threads yield to the higher priority threads
- The real-time systemic behaviors are effective

Results and Analysis – Shared Threadpools



RMS



Anti-RMS

- RMS – lower rate (lower priority) controller-worker threads yield to the higher priority threads
- Anti-RMS – higher rate (lower priority) controller-worker threads yield to the higher priority threads
- The real-time systemic behaviors are effective by using a different strategy

Concluding Remarks

- Combining CCM+RT aspects overcomes limitations of
  - Conventional DRE middleware
  - Conventional component middleware
- Approach provides a novel and effective meta-programming architecture for QoS aspects in DRE component middleware
  - Declarative composition of policies/mechanisms at fine granularity throughout the DRE system lifecycle
- Empirical studies show temporal cost of flexibility can be low, compared to conventional RT-CORBA middleware
- Profiling and implementation experience indicate CCM+RT aspect programming model helps DRE system development
  - Simplifies application development/assembly/deployment
  - Allows late-lifecycle performance optimizations

Future Work

- Distributed Scientific Computing Environment
  - Composing distributed scientific computation applications using Lightweight CCM
  - Composing scientific component using Common Component Architecture (CCA)
    - Incorporate CCA run-time environment in CCM component implementations
    - Incorporate legacy scientific code (FORTRAN, C, etc.)
- Extra-functional aspects, such as,
  - Parallelism
  - Streaming of data