



Achieving Bounded End-to-End Latencies with Real-time Linux and Realtime CORBA

**Gautam H. Thaker
Patrick J. Lardieri
Donald Krecker
Keith O'Hara
Chuck Winters**

**LM Advanced Technology Labs
Camden, NJ
{gthaker,plardier}@atl.lmco.com**

Purpose and Outline



■ Purpose

- Experimental work to apply distributed scheduling theory to RT-CORBA-based DRE systems on (Real-time) Linux platforms

■ Outline

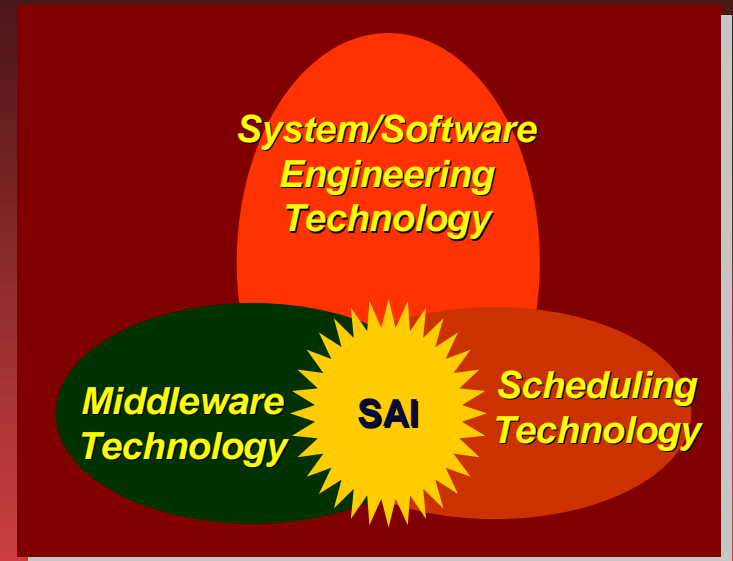
- Research Vision
- Evaluation of Linux QoS mechanism
- Evaluation of QoS mechanisms suitability for distributed realtime scheduling
 - Comparison between theory, simulation, and testbed

Research Vision

Research Challenge



- **Grand Architectural Vision** for a standards compliant COTS-based infrastructure that enables predictable systems
 - Focus on bounded time operations
 - Multi-property problem extension is more challenging



- **Critical enabling technologies**
 - Middleware (includes QoS and networks)
 - Task and network scheduling theory
 - System and software engineering methods and tools

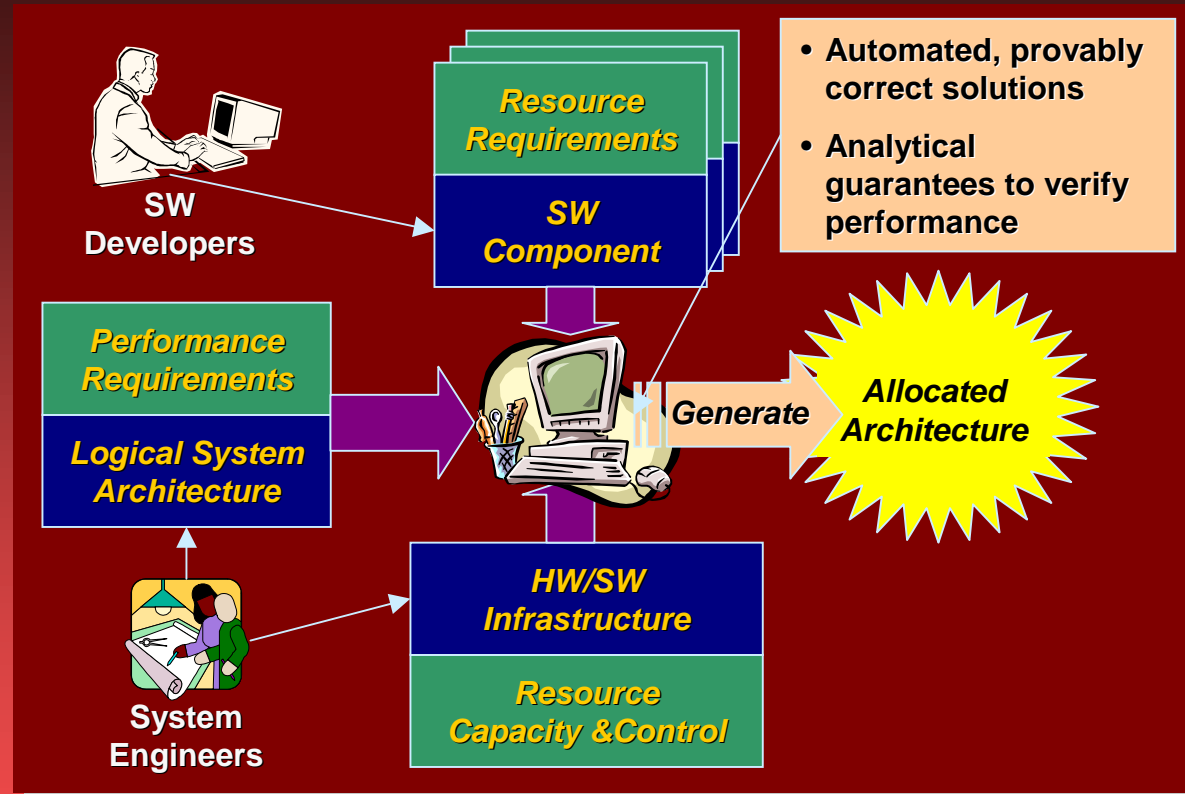
Challenge is essentially the problem of distributed resource management (specification and allocation)

ATL Research Focus



■ Architectural vision and associated research problems

- Resource-requirements language
- Performance-requirements language
- Resource-capacity language
- QoS mechanisms

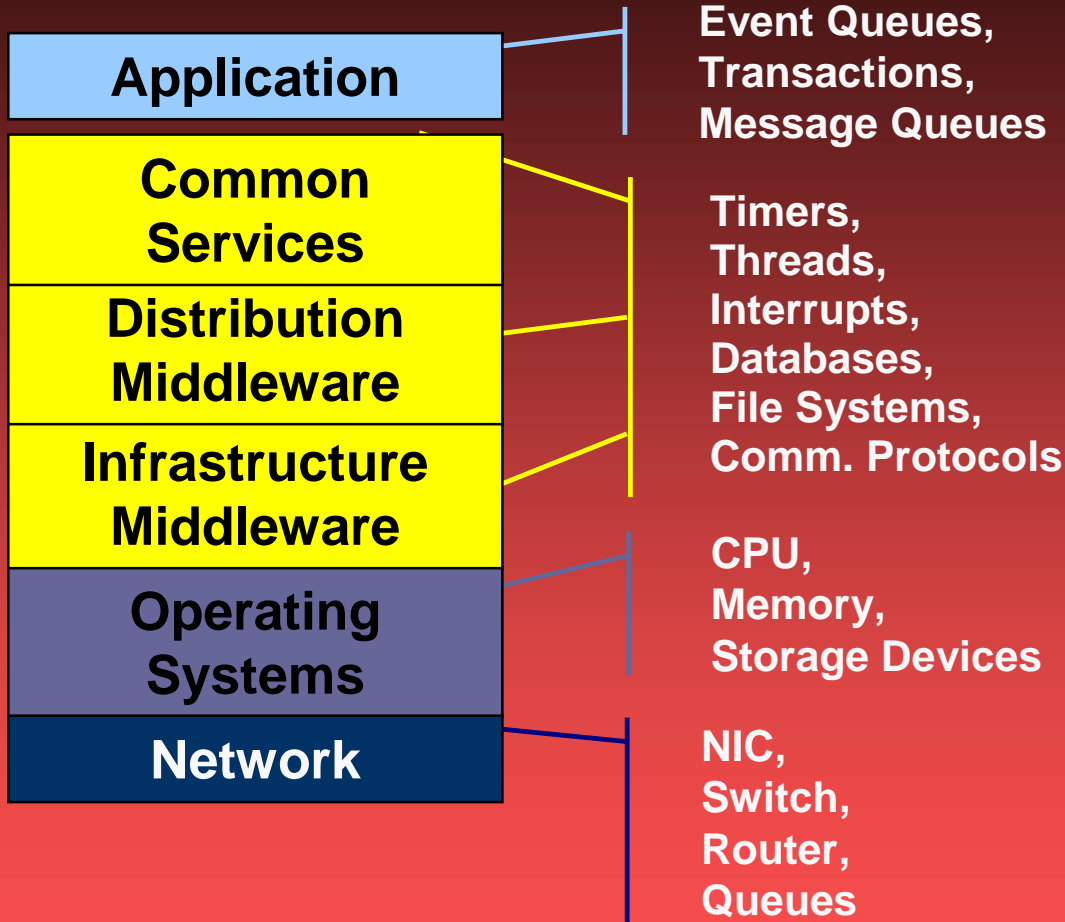


- Holistic “end-to-end system” resource management

***Supports proactive and reactive
Resource-management approaches***

Evaluation of Linux QoS Mechanism

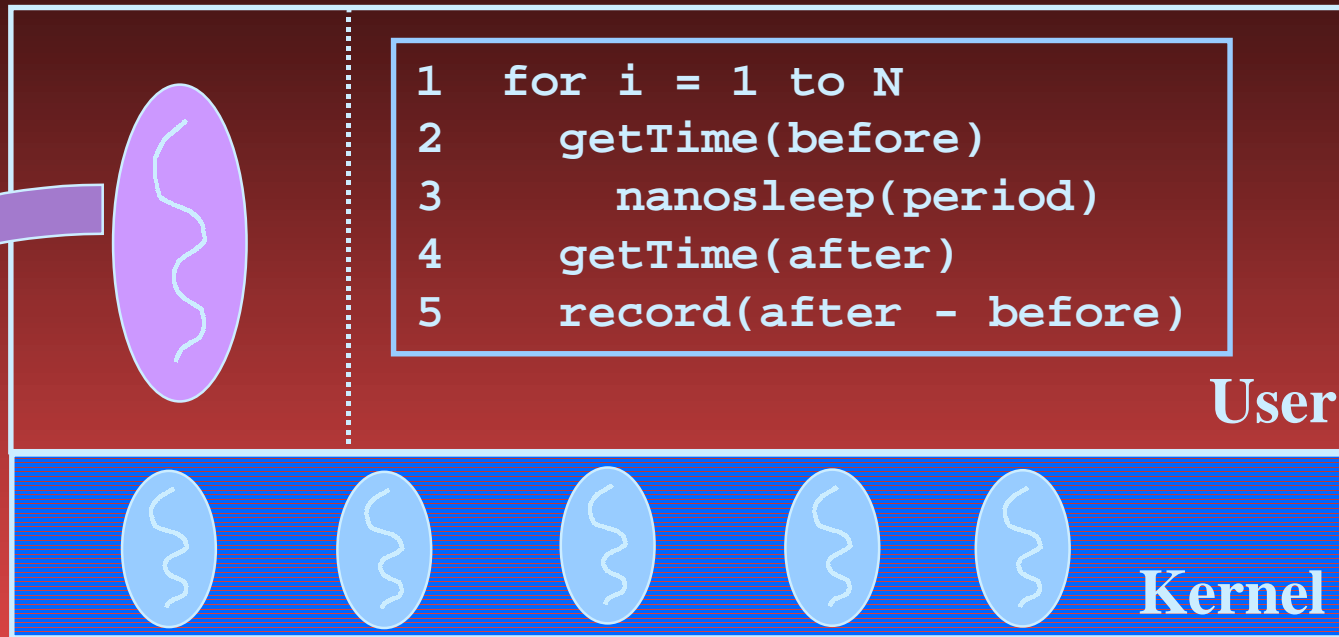
QoS Performance Evaluation



- **Experimental approach**
 - Simple test design
 - Focus on worst case behavior
 - Establish upper bounds
 - Conduct large sample tests
 - Verify against simulations
 - Collect comprehensive data
 - Share results and work with vendors/technologist to explain and improve performance

ATL has conducted over 2000 experiments and collected results on its QoS Technology website

Measuring Scheduling Jitter

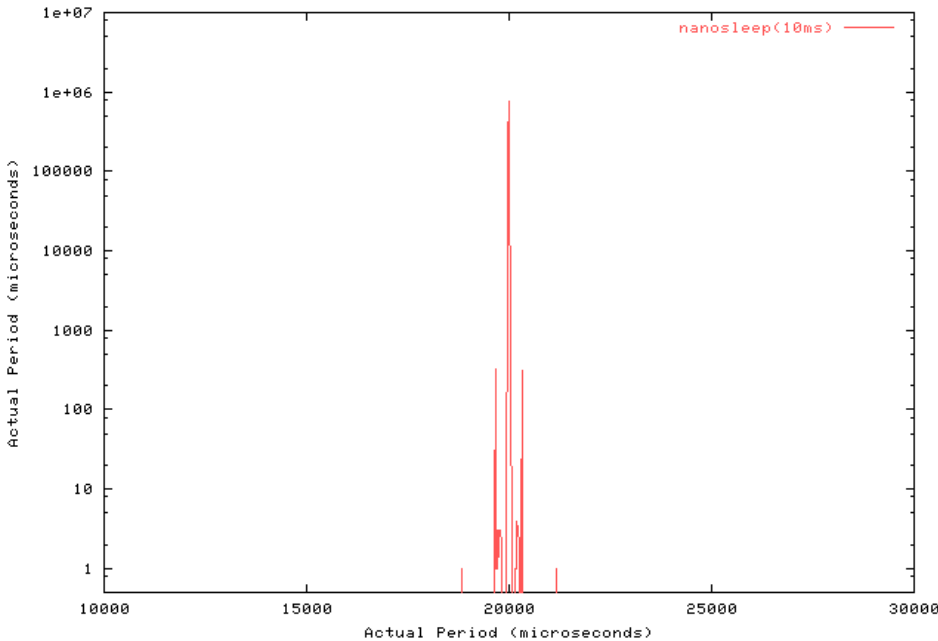


System Conditions

- No System Load
- Competing CPU intensive non-realtime processes
- Competing I/O intensive non-realtime processes

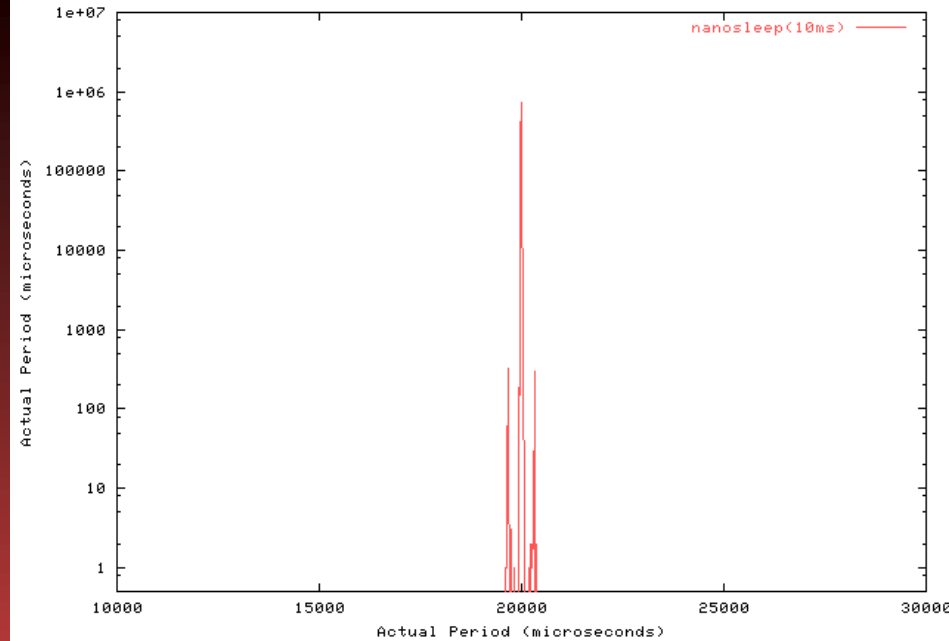
Linux 2.5.20 Periodic Test - No Load

System: Pentium II 450 MHz, 256M RAM, SCSI Wed Jun 12 16:28:20 EDT 2002 kjo
min: 18839 max: 21162 mean: 20000 var: 82.6406 n: 1000000



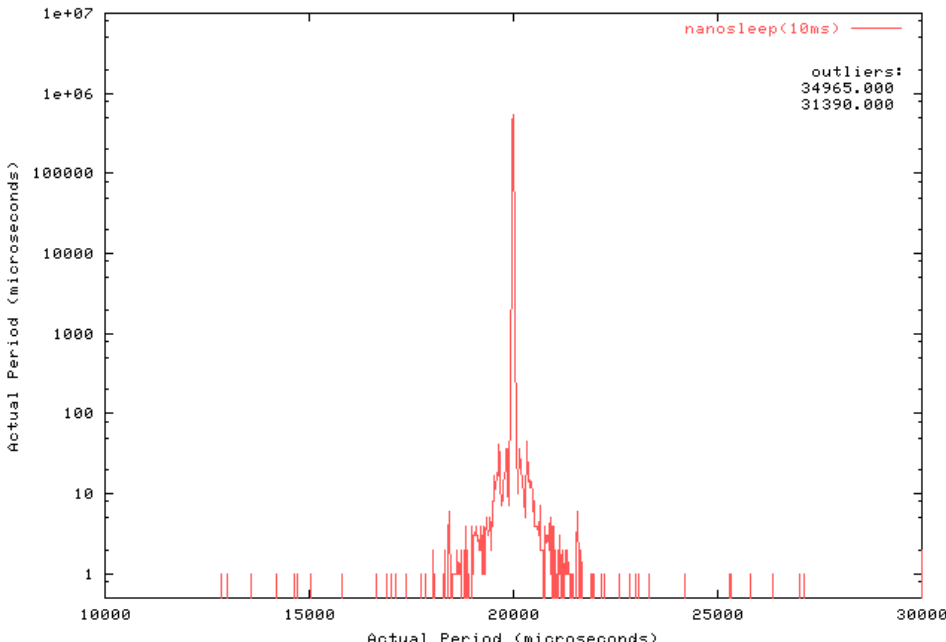
Linux 2.5.20 Periodic Test - CPU Load

System: Pentium II 450 MHz, 256M RAM, SCSI Wed Jun 12 22:01:41 EDT 2002 kjo
min: 19614 max: 20390 mean: 20000 var: 81.1103 n: 1000000



Linux 2.5.20 Periodic Test - DiskLoad

System: Pentium II 450 MHz, 256M RAM, SCSI Thu Jun 13 03:35:03 EDT 2002 kjo
min: 12859 max: 34965 mean: 20000 var: 1362.78 n: 1000000



Linux 2.5.20

Scheduler

- O(1) – constant time and scalable
- SMP processor affinity
- 100 priorities (0 - 99)

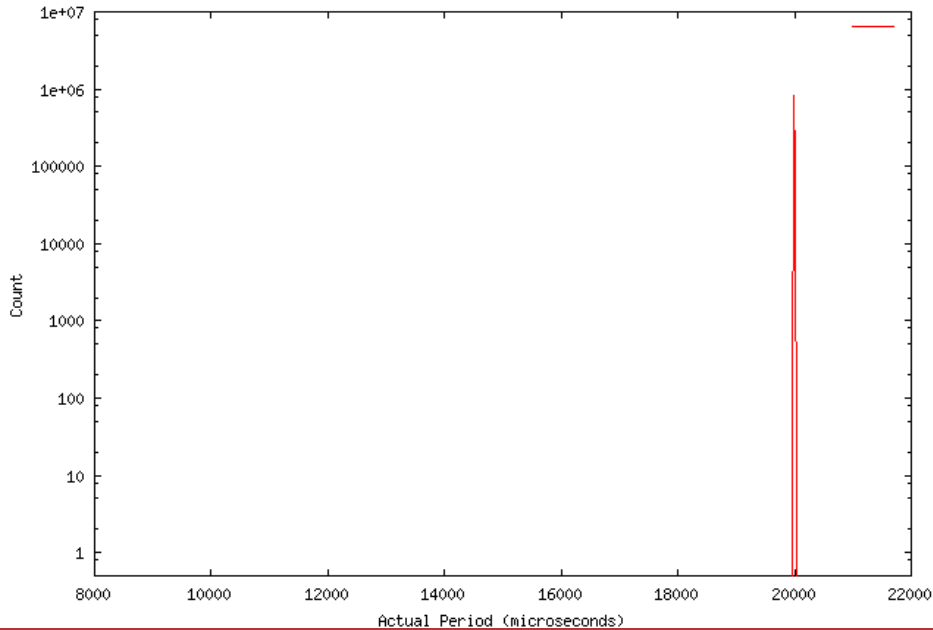
Kernel Preemptibility

- Small regions of non-preemptibility marked by spinlocks
- No protection for kernel or user-land priority inversion

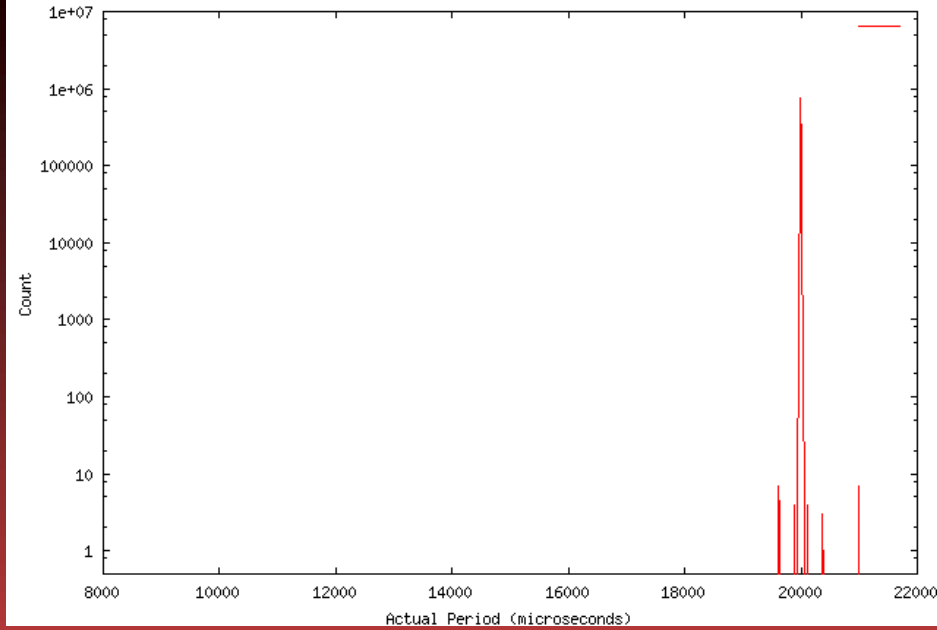
Kernel routines are not schedulable

Low nanosleep() resolution due to clock granularity

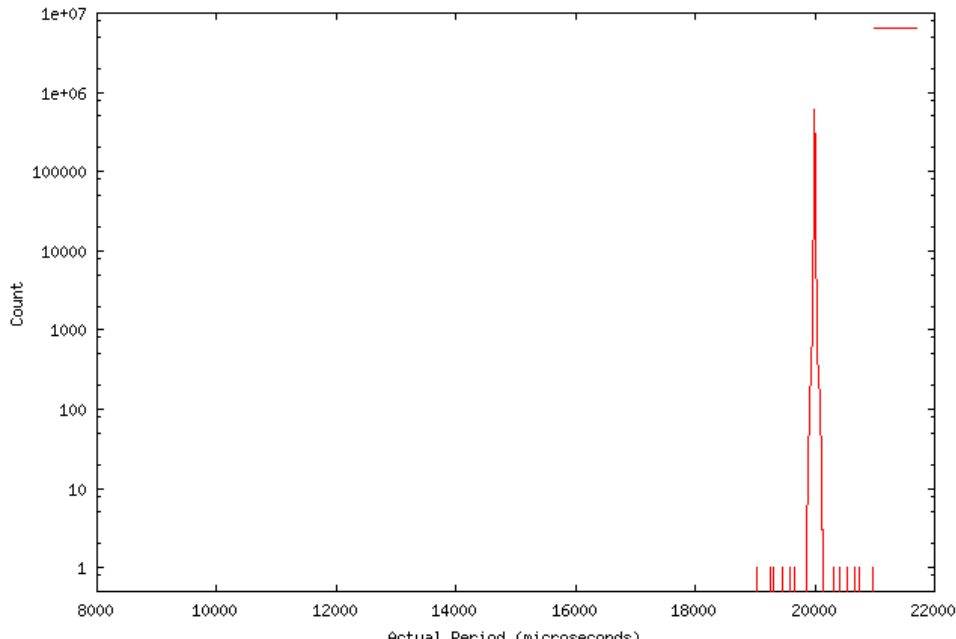
Timesys Linux v3.0 Periodic Test - No Load
System: Pentium II 450 MHz, 256M RAM, SCSI [1.7.2002.kjo]
min: 19960 max: 20045 mean: 20000 var: 20.7364 n: 999999



Timesys Linux v3.0 Periodic Test - CPU Load
System: Pentium II 450 MHz, 256M RAM, SCSI [1.7.2002.kjo]
min: 19604 max: 30396 mean: 20000.1 var: 787.823 n: 999999



Timesys Linux v3.0 Periodic Test - Disk Load
System: Pentium II 450 MHz, 256M RAM, SCSI [1.7.2002.kjo]
min: 19024 max: 20977 mean: 20000 var: 52.7655 n: 803543



Timesys Linux/GPL

Scheduler

$O(1)$ – constant-time and scalable

Variable number of priorities (4 - 32,768)

Kernel Preemptibility

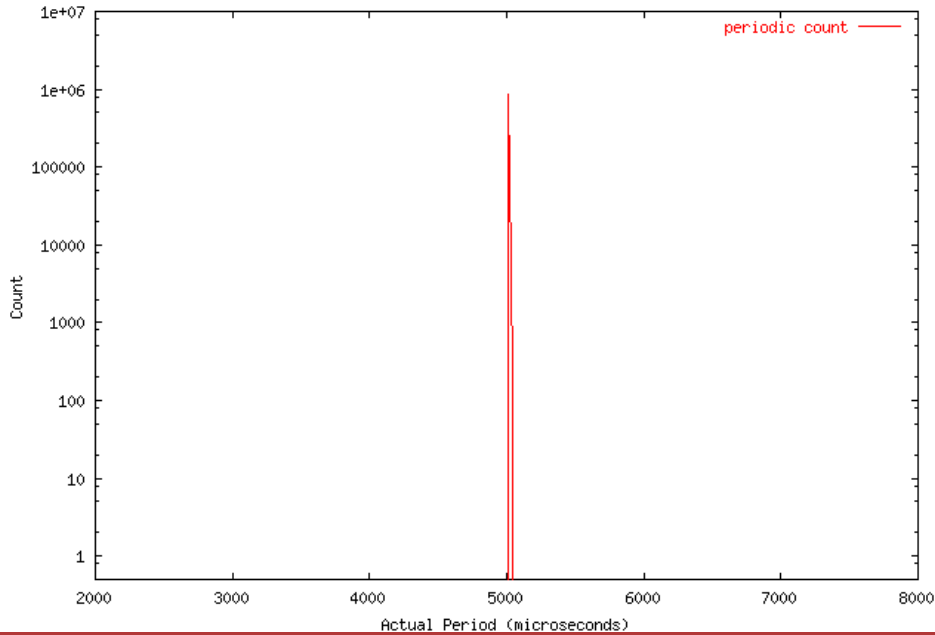
Fully preemptible using mutexes to guard critical sections

Protection against kernel-level priority inversions via priority inheritance

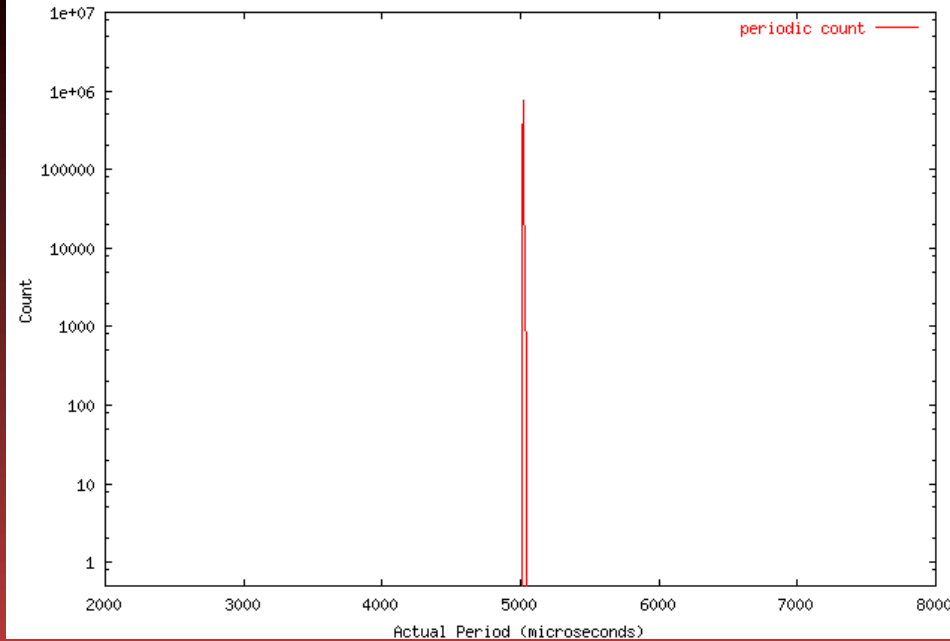
Kernel routines are schedulable

Low resolution nanosleep() due to clock granularity

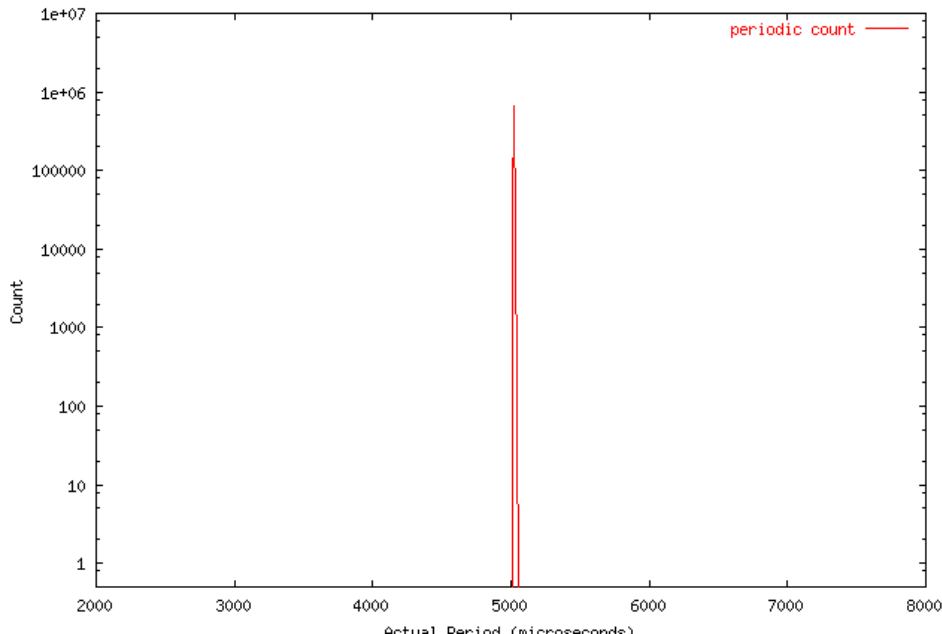
Timesys Linux/Realtime v3.0 Periodic Test - No Load
System: Pentium II 450 MHz, 256M RAM, SCSI Wed Jan 9 17:55:09 EST 2002
min: 5010 max: 5050 mean: 5016.99 var: 9.00754 n: 1000000



Timesys Linux/Realtime v3.0 Periodic Test - CPU Load
System: Pentium II 450 MHz, 256M RAM, SCSI Wed Jan 9 19:18:47 EST 2002
min: 5011 max: 5052 mean: 5018.8 var: 7.09363 n: 1000000



Timesys Linux/Realtime v3.0 Periodic Test - DiskLoad
System: Pentium II 450 MHz, 256M RAM, SCSI Wed Jan 9 20:42:28 EST 2002
min: 5010 max: 5058 mean: 5022.67 var: 14.1738 n: 1000000



Timesys Linux/Real-time

Scheduler

$O(1)$ – constant-time and scalable

Variable number of priorities (4 - 32,768)

Kernel Preemptibility

Fully preemptible using mutexes to guard critical sections

Protection against kernel and user-land priority inversions via priority inheritance

Kernel routines are schedulable

High resolution nanosleep() using ktimer kernel module

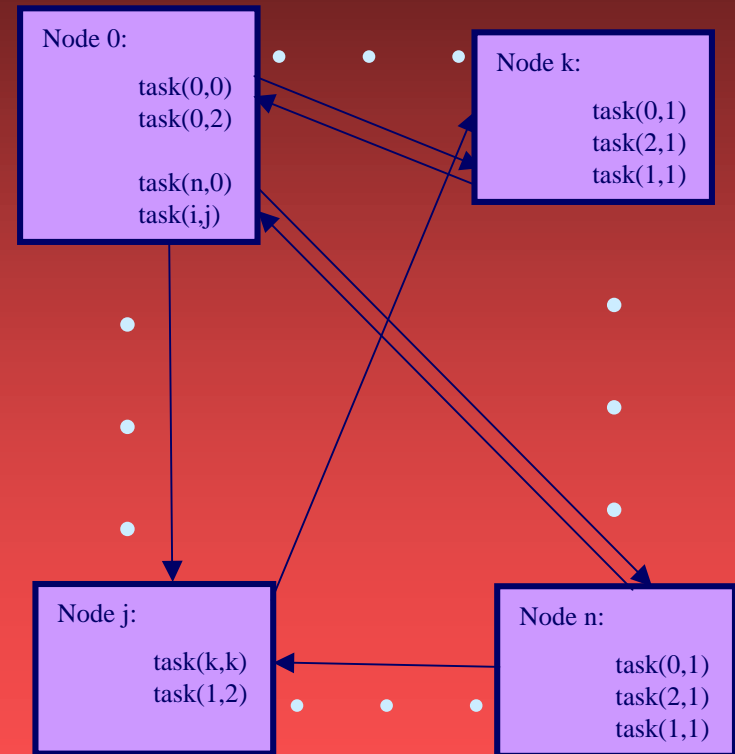


Evaluation of QoS Mechanisms Suitability for Distributed Realtime Scheduling

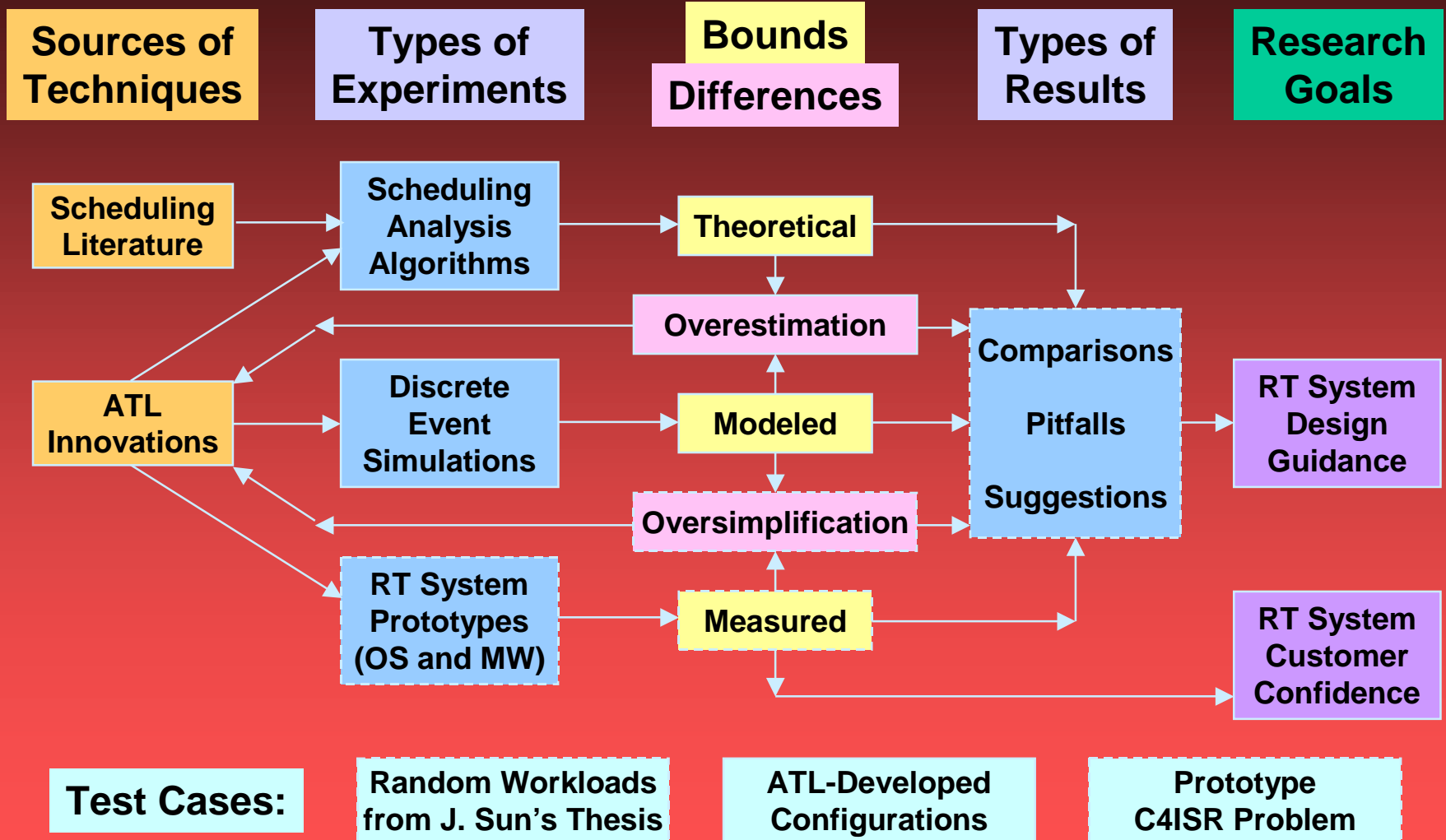
Problem Introduction



- Multiple, inter-connected nodes
- Multiple task chains
 - Each task has its own period and deadline
 - Each task consists on “n” subtasks
 - Completion of subtask(j) signals release of subtask(j+1)
- Notation:
 - $\text{task}(\text{task\#}, \text{subtask\#})$
 - $\text{cpu}(\text{task\#}, \text{subtask\#}) =$ execution demand of subtask, etc.



Holistic End-to-End Scheduling



Experimental Evaluations



■ **Uni-processor case**

- Four periodic tasks scheduled by different techniques (priority, rate monotonic, deadline monotonic)

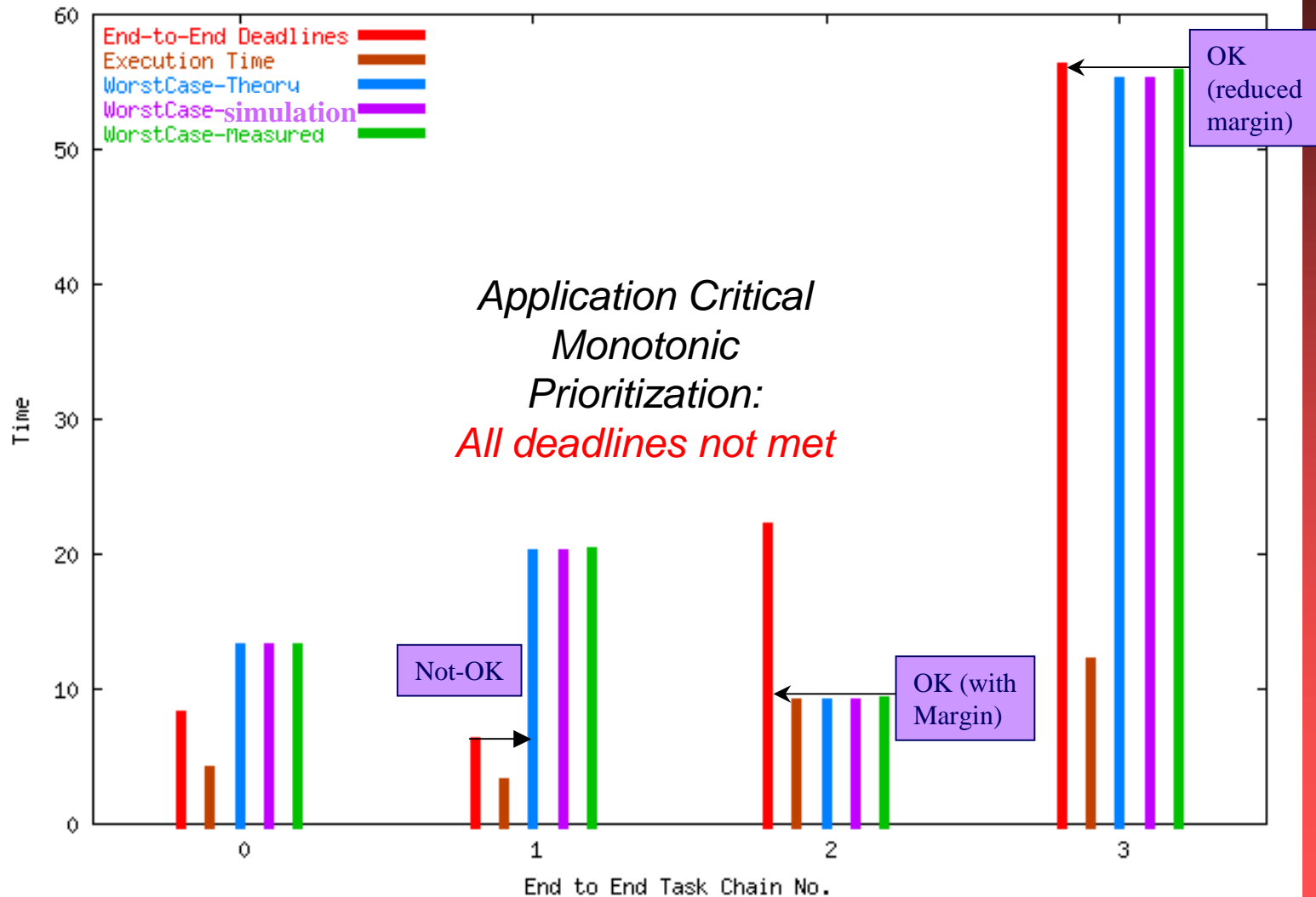
■ **Distributed (three processor) case**

- Three periodic tasks; each task is a chain consisting of three subtasks

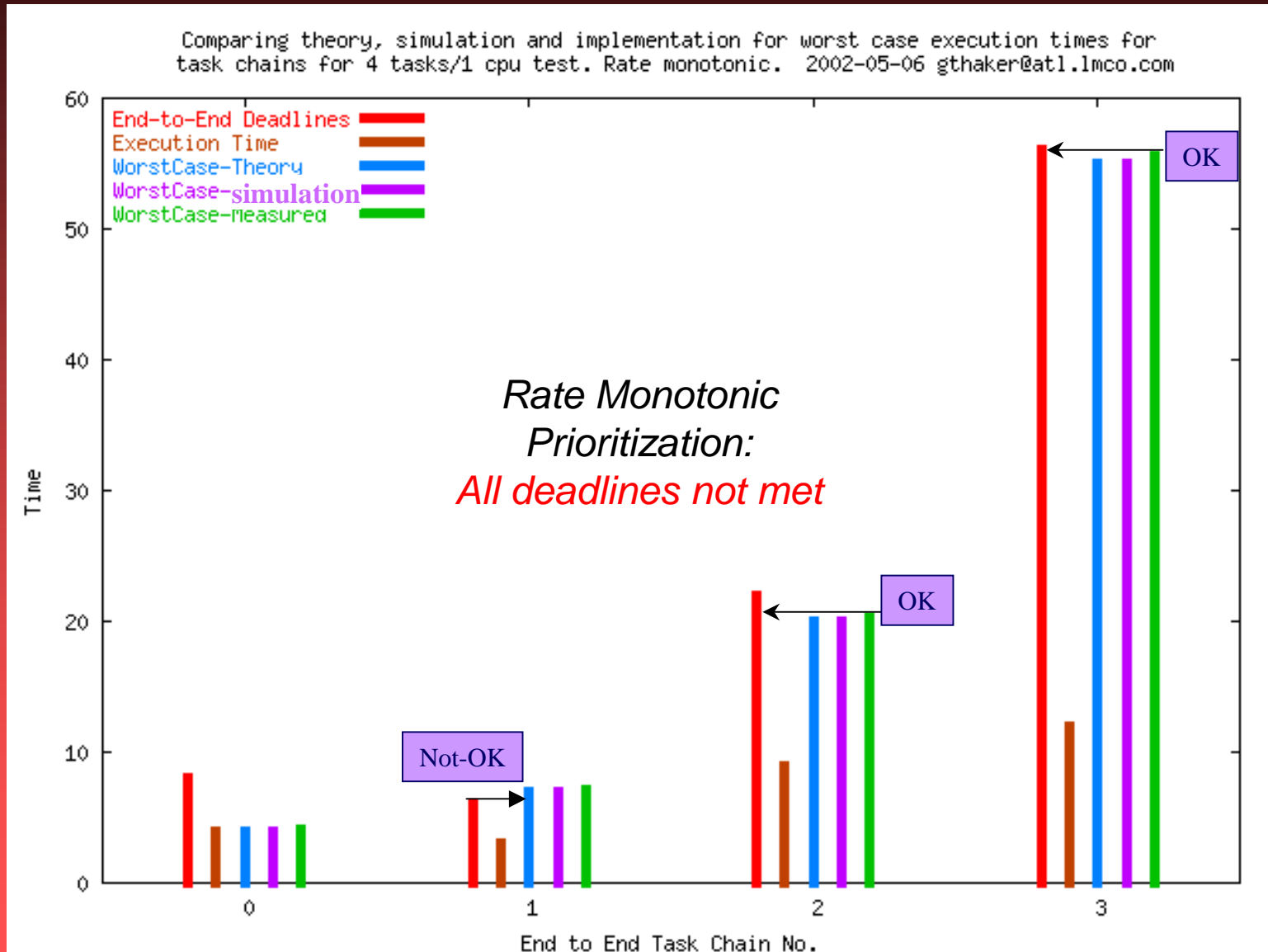
Uni-Processor, Priority Scheduling



Comparing theory, simulation and implementation for worst case execution times for task chains for 4 tasks/1 cpu test. criticality monotonic. 2002-05-06 gthaker@atl.lmco.com



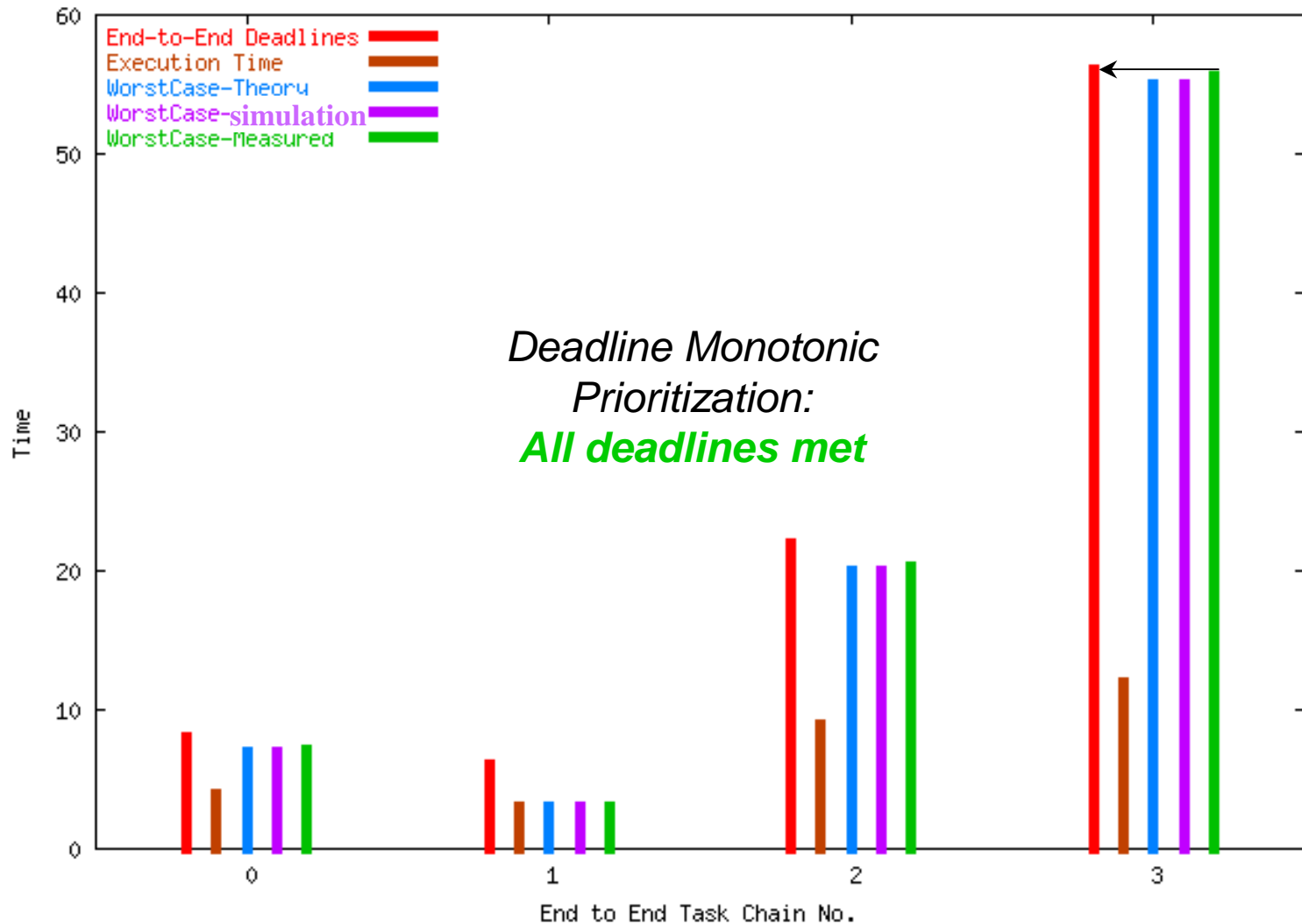
Uni-Processor, Rate Monotonic Scheduling



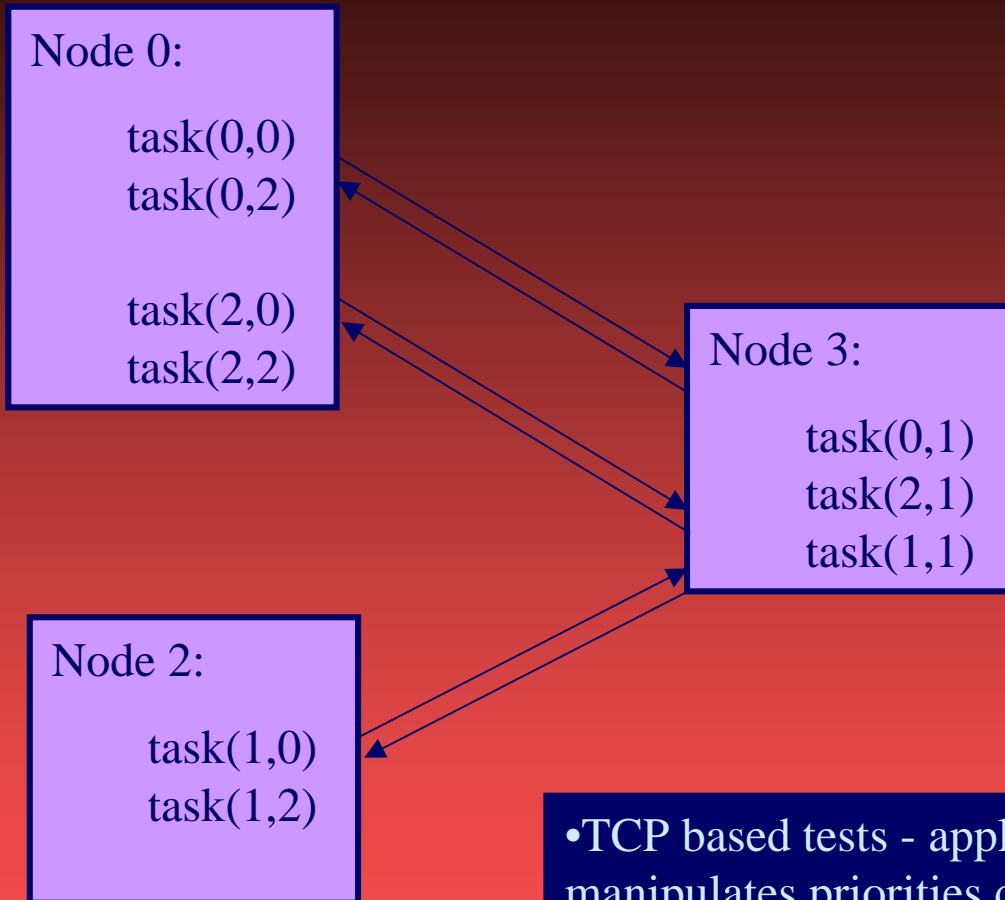
Uni-Processor, Deadline Monotonic Scheduling



Comparing theory, simulation and implementation for worst case execution times for task chains for 4 tasks/1 cpu test. Deadline monotonic. 2002-05-06 gthaker@atl.lmco.com



Multi-processor Test Case



Task 0

period = 104

deadline = 88

cpu(0,0) = 7

cpu(0,1) = 8

cpu(0,2) = 25

Task 1

period = 30

deadline = 30

cpu(1,0) = 9

cpu(1,1) = 5

cpu(1,2) = 11

Task 2

period = 54

deadline = 54

cpu(2,0) = 10

cpu(2,1) = 6

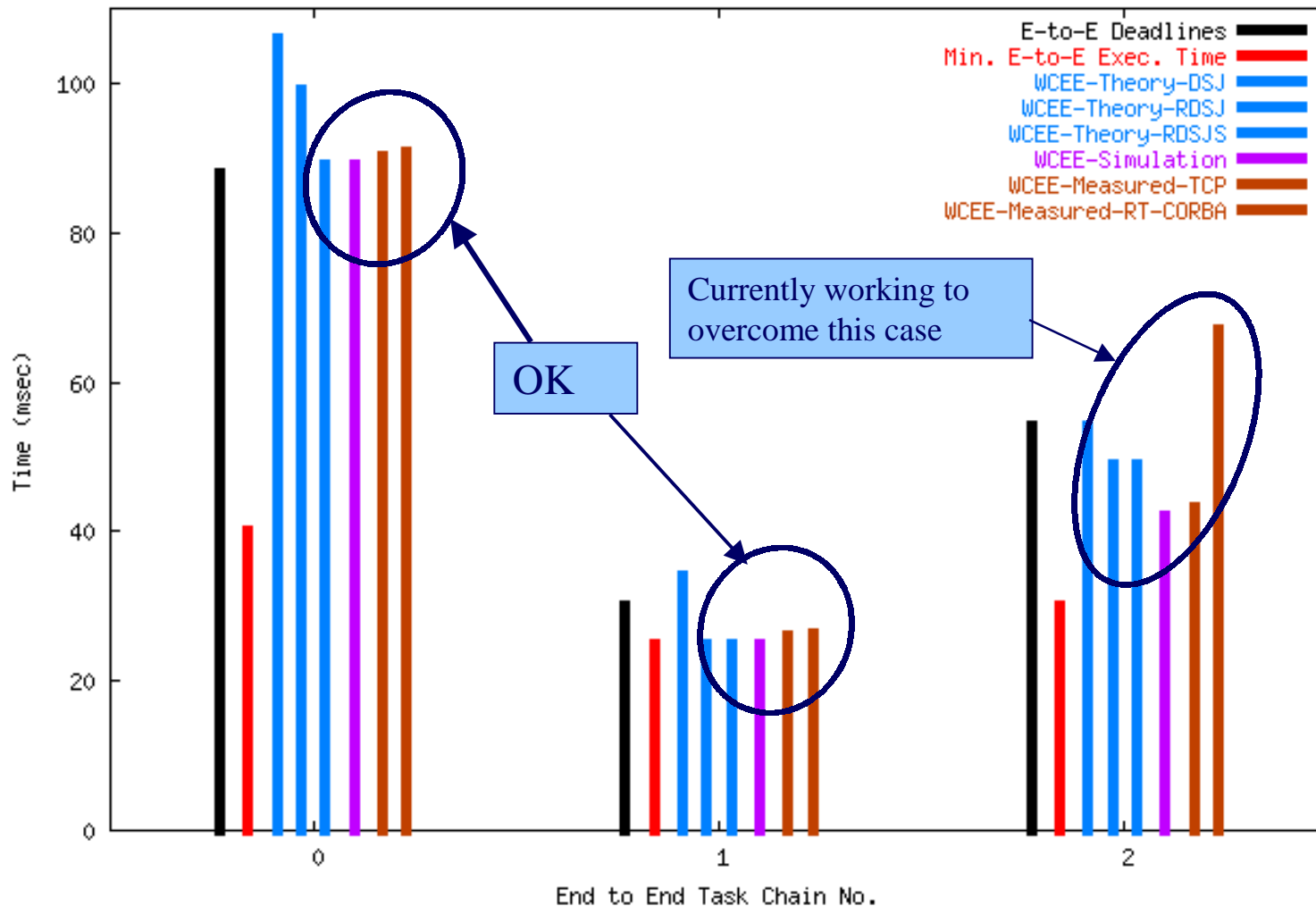
cpu(2,2) = 14

- TCP based tests - application manipulates priorities directly
- RT-CORBA based tests extend examples/RTCORBA/Activity and use thread pools, lanes and bands

Summary of Multi-Processor Results



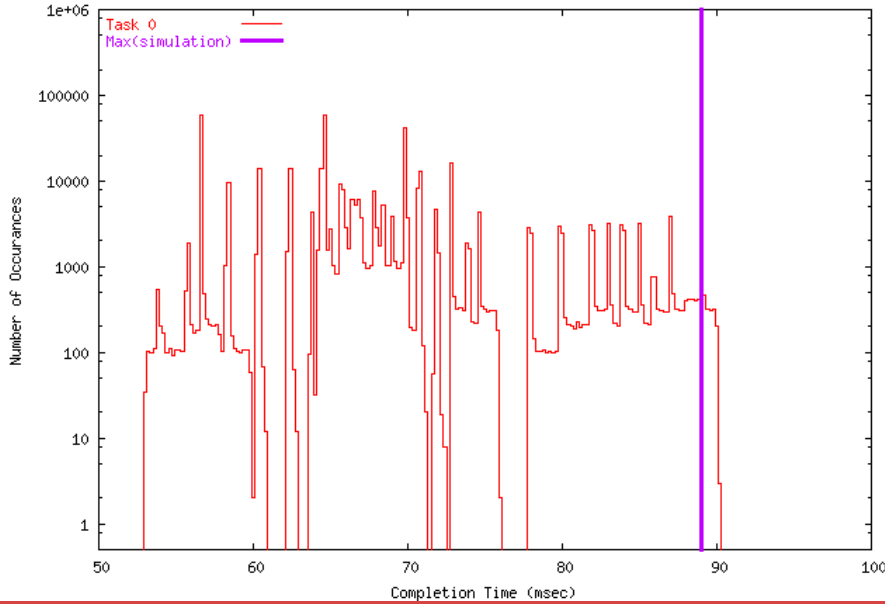
Comparing worst case end to end (WCEE) execution times for theory, simulation and implementation for 3 task chains (each w/ 3 subtasks) on 3 cpus.
Proportional Deadline monotonic. 2002-06-13 gthaker@atl.lmco.com



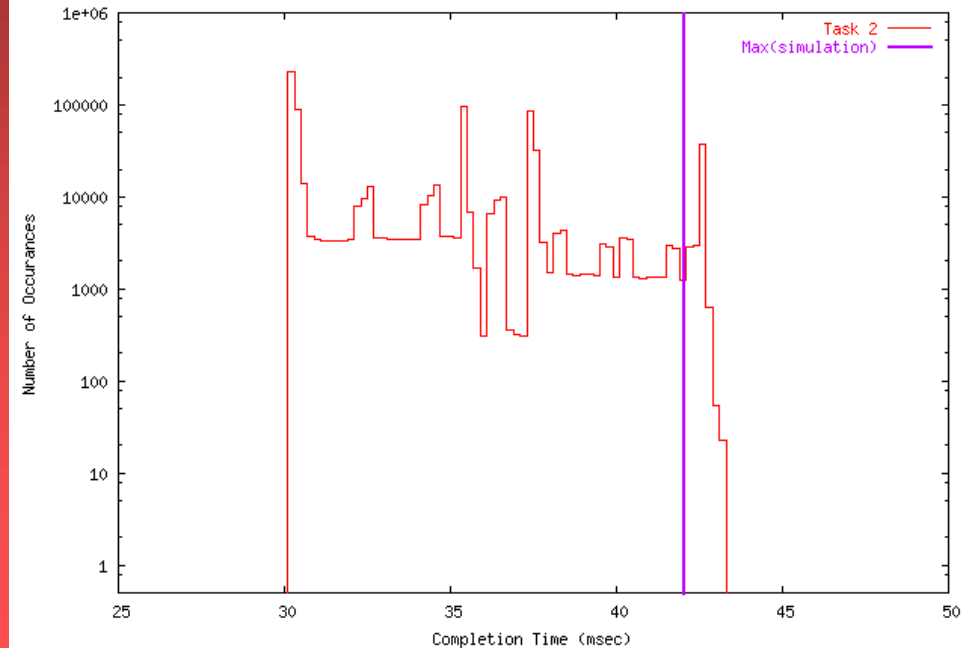
Examining Task 0 and 2 Completion Times



Tested on Timesys 2.4.7-timesys-3.1.214, P-IV, 1.6GHz, 3 nodes.
TCP based test by cwintrs. 99.6% of samples fall below the
predicted WCRT of 89 2002-06-13 gthaker@atl.lmco.com



Tested on Timesys 2.4.7-timesys-3.1.214, P-IV, 1.6GHz, 3 nodes.
TCP based test by cwintrs. 94.4% of samples fall below the
predicted WCRT of 42 2002-06-14 gthaker@atl.lmco.com



There is minimal to modest disagreement due to communication costs being ignored in simulation for the time being

Conclusions



- **Pessimism in worst case end-to-end response estimation by theory is reduced**
 - In general all pessimism can not be eliminated but still further improvements are being addressed by future work
- **Real-time Linux offers sufficient support for building small- to medium-scale (hard) real-time system**
- **RT-CORBA support for thread pools, lanes and bands leveraged with generally good success**

Future Work



- **Release Guard based system**
- **Analytical techniques**
 - seek further refinements in algorithms that yield still less pessimism
 - further explore off-line, on-line & scaling issues
- **Integration of these improved techniques with TAO real-time scheduling service**
 - **Automatic generation of all svc.conf.x files with proper parameters for lanes, bands etc.**
 - **Explore use of these techniques in CORBA Components and Model Driven Architecture**
- **Integrate testbed with our SCIOP (GIOP over SCTP) work**