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**

- **Application**
  - Event Queues, Transactions, Message Queues

- **Common Services**
  - Timers, Threads, Interrupts, Databases, File Systems, Comm. Protocols
  - CPU, Memory, Storage Devices

- **Distribution Middleware**
  - NIC, Switch, Router, Queues

- **Infrastructure Middleware**

- **Operating Systems**

- **Network**

**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

1. for i = 1 to N
2. getTime(before)
3. nanosleep(period)
4. getTime(after)
5. record(after - before)

System Conditions

- No System Load
- Competing CPU intensive non-realtime processes
- Competing I/O intensive non-realtime processes
Linux 2.4.7

- **Scheduler**
  - \(O(n)\) -- variable time and not scalable
  - 100 priorities (0 - 99)

- **Kernel Preemptibility**
  - Not preemptible
  - No protection for kernel or user priority inversion

- **Kernel routines are not schedulable**
- **Low resolution nanosleep() - clock granularity**
**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/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/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
ATL QoS Performance Evaluation

Welcome to Advanced Technology Labs!
Agent and Distributed Objects Quality of Service (QoS) Home Page

Middleware Comparator Analysis Tool

Overview.
Pick Results to Plot:
Plot | Reset | Range (< 4)
Diff 2 Curves

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:
  - task(task#, subtask#)
  - cpu(task#, subtask#) = execution demand of subtask, etc.
Holistic End-to-End Scheduling

Sources of Techniques
- Scheduling Literature
- ATL Innovations

Types of Experiments
- Scheduling Analysis Algorithms
- Discrete Event Simulations
- RT System Prototypes (OS and MW)

Bounds Differences
- Theoretical
- Overestimation
- Modeled
- Oversimplification
- Measured

Types of Results
- Comparisons
- Pitfalls
- Suggestions

Research Goals
- RT System Design Guidance
- RT System Customer Confidence

Test Cases:
- Random Workloads from J. Sun’s Thesis
- ATL-Developed Configurations
- Prototype C4ISR Problem

Research Goals

ATL Innovations

Scheduling Literature

Types of Experiments

Bounds Differences

Types of Results

Research Goals

Test Cases:

ATL Innovations

Scheduling Literature

Types of Experiments

Bounds Differences

Types of Results

Research Goals

Test Cases:
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

Application Critical
Monotonic
Prioritization:
All deadlines not met

OK (reduced margin)
Not-OK
OK (with Margin)
Uni-Processor, Rate Monotonic Scheduling

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

Rate Monotonic Prioritization:
All deadlines not met
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

Deadline Monotonic Prioritization:
All deadlines met
Multi-processor Test Case

Node 0:
- task(0,0)
- task(0,2)
- task(2,0)
- task(2,2)

Node 2:
- task(1,0)
- task(1,2)

Node 3:
- task(0,1)
- task(2,1)
- task(1,1)

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

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
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.mco.com

Currently working to overcome this case

OK
Examining Task 0 and 2 Completion Times

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