Predictable Execution: Operating Systems Issues

Michael B. Jones — Microsoft Research

Joint work with

John Regehr — University of Virginia

Goal: Coexisting Independent Real-Time Applications

- **Independently developed**
- **Predictable concurrent execution of**
  - **real-time *and* non-real-time apps**
- **Meeting all apps' timing needs**
  - **Informing apps when not possible**

Overview

- Developing soft real-time scheduler for Windows NT
  - Described in second half of talk
- Predictability issues
  - How large are observed worst-case thread scheduling latencies?
  - What causes them?
  - What has been done about them?
  - Described in first half of talk

Part I:

The Problems You’re Having May Not Be the Problems You Think You’re Having:

Results from a Latency Study of Windows NT

Research Context

- Developing soft real-time scheduler for Windows NT
- Predictability issues:
  - How large are observed worst-case thread scheduling latencies?
  - Can they be improved?
- Measured actual latencies

NT Latency Results

- Typically can schedule tasks every small number of milliseconds
- But ill-behaved drivers, hardware can take many milliseconds
  - Software delays of up to 16ms observed
  - Hardware delays of up to 30ms observed

Results from NT 5, Pentium II-333

Deferred Procedure Calls

- Analogous to Unix bottom halves
- Are preempted by interrupts
- Preempt normal threads
- May not block
- Are run in FIFO order
- Typical Uses:
  - I/O Completion Processing
  - Background Driver Housekeeping

Non-Problems

- Interrupts
  - Interrupt handlers needing substantial work queue DPCs
  - Never observed interrupt handler taking substantial fraction of ms
- Ethernet Packet Processing
  - With back-to-back 100Mbit incoming packets of UDP or TCP data:
    - Longest observed DPC 600µs
    - Longest delay of user code ~2ms
- Tested four common Ethernet cards

Problem: "Unimportant" Background Work

- DEC dc21x4 PCI Fast 10/100 Ethernet
- 6ms periodic DPC every 5s
  - Autosense processing
- Most of 6ms in five 0.88ms calls to routine that reads device register that:
  - Writes a HW register – 1.5µs
  - Stalls for 5µs
  - Writes HW register again – 1.5µs
  - Stalls for 5µs
  - Reads a HW register – 1.5µs
  - Stalls for 5µs
- And does this 16 times! (once per bit)

Another Long DPC: Intel EE 16

- Intel EtherExpress 16 ISA Ethernet
- 17ms DPC every 10s
- Card reset for no received packets

Amusing Observation

- Unplugging Ethernet makes latency worse!
  - Despite conventional wisdom to the contrary

Even Worse: Video Cards

- Video cards and drivers conspire to hog the PCI bus
- Dragging large window locks out interrupts for up to 30ms
- Obliterates sound I/O, for instance
- Can set registry key to ask drivers to behave, but not default
  - No problem when set correctly
- Manufacturers' motivation: WinBench ~ 5% improvement

Video Card Misbehavior Details

- Don’t check if card FIFO full before write
  - Eliminate a PCI read
- Stalls PCI bus if full to prevent overflow
  - Even with AGP, big blits are slow
- Problem observed on:
  - AccelStar II AGP
  - Matrox Millenium II
- Several other major cards also do this

Example Bug: Multimedia Timers

- MP HAL uses 976μs interrupt period
- Multimedia timers compute absolute time for next wakeup in whole ms
  - Converted to relative wakeup time and passed to kernel
- Interrupt occurs just before wakeup
  - Timer doesn't fire
  - Next time, fires twice to catch up
- Fix: compute wakeup in 100ns units

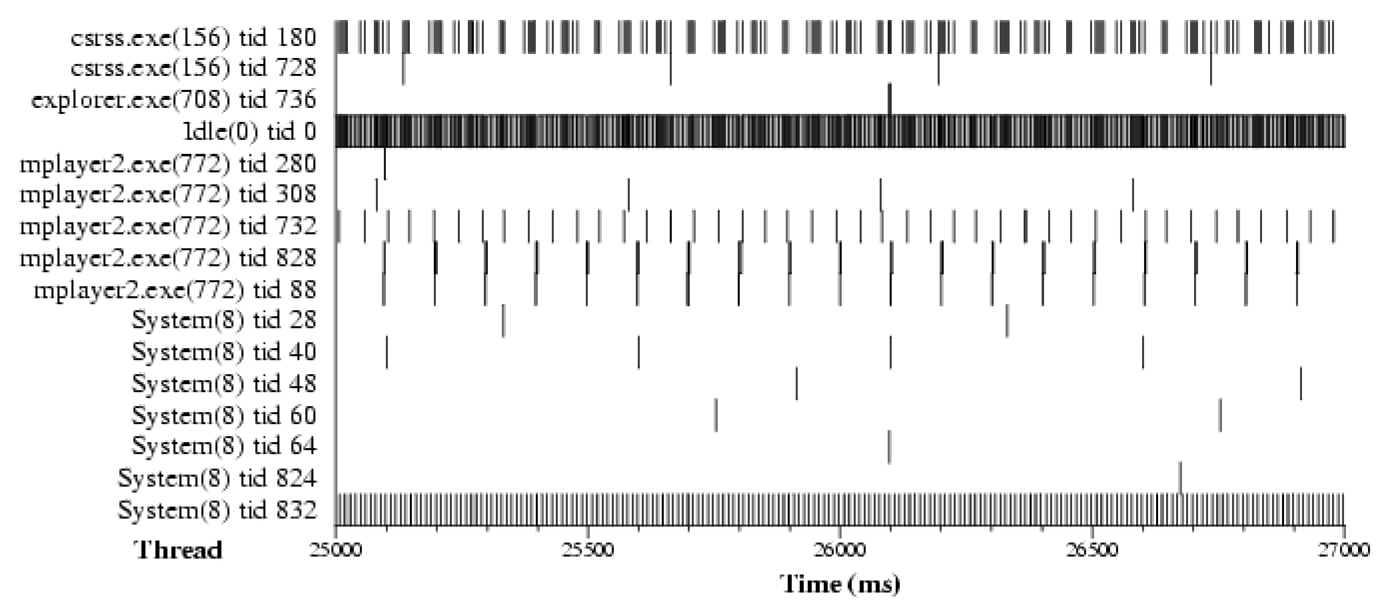
Windows Media Player Audio Dropouts

- Playback glitches when in contention with other apps
- Concerted effort to find, fix causes before Windows 2000 ship

Media Player Thread Structure (Simplified)

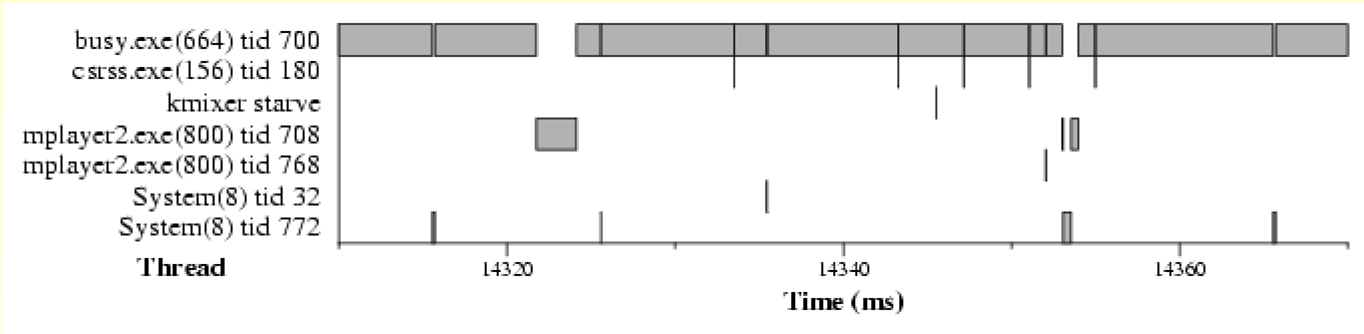
Thread	Period (ms)	Priority
Kernel Mixer	10	24
MP3 Decoder	100	9
Disk Reader	2000	8
User Interface	45	8

MP3 Playback w/o Contention



- **Working fine**
- **Kmixer thread *(bottom)* runs every 10ms**
- **MP3 decoder *(middle)* runs every 100ms**

Priority Inversion Caused By Competing Thread



- **Priority inversion at time 14324ms**
- **Busy thread *(top)* preempts decoder thread while holding kmixer buffer lock**
- **Kmixer *(bottom)* starves causing audio dropout**

- **Fix: Raise priority in decoder to that of kmixer before acquiring lock**
  - **Manual application of Priority Ceiling Protocol**

Lesson

Your Intuition About Performance is Wrong

Only Measurement Reveals the Truth!

Bottom Line

- **Yes, NT can do RT scheduling**
- **Have done a prototype**
  - **But will be of limited value if unscheduled activities continue taking tens of milliseconds**
- **NT developed, tested for *throughput***
  - **Not small numbers of ms of latency**
  - **Improvement will require**
    - **Systematic latency testing**
    - **Latency requirements specifications**

Progress Since Initial Work

- WHQL tests for video drivers
  - Verify PCI timing with hardware
- Many timing bugs found, fixed
  - E.g., media player priority inversion

- Attempting to institute systematic latency specifications and testing
  - Interrupt hold times & counts
  - DPC hold times & counts
  - Standards for use of priority values

Part II:

CPU Reservations and Time Constraints: Implementation Experience on Windows NT

Part II Outline

Introduction

Rialto Background

Windows NT Implementation

Performance and Traces

Related Work and Conclusions

What We Did

- **Created Rialto/NT**
  - **Based on Windows 2000**
  - **Added CPU Reservations & Time Constraints**
    - **Abstractions originally developed within *Rialto* real-time system at Microsoft Research**
- **What's new**
  - **Coexistence with Windows NT scheduler**
  - **Multiprocessor capability**
  - **Periodic clock**
    - **As opposed to fine-grained individually scheduled interrupts**

Real-Time

- Real-time computations have deadlines
- Examples
  - Fly-by-wire aircraft:
    - Missed deadline may endanger the aircraft
  - Soft modem:
    - Missed deadline may reset the connection
  - Video conferencing:
    - Missed deadline degrades audio or video quality

Why not use Windows NT as is?

- **Real-time using priorities requires global coordination**
  - **Windows is an open system**
    - **Priority inflation**
  - **No path for timing information**
- **There are scheduling algorithms that *do not* require global coordination**
  - **CPU Reservations and Time Constraints**
  - **Apps state timing requirements directly**
  - **Independently developed apps can run concurrently**

Teaser Capability

- **Apps can ask scheduler:**
  - **"Can I do 5ms of work between now+30ms and now+40ms?"**
- **Scheduler answers either:**
  - **"I guarantee it" or**
  - **"You probably can't"**
- ***Guaranteeing this 5ms work in future 10ms interval does not require reserving 50% of CPU for next 40ms***

How did we do it?

- Explicitly represent future time
- Map app declarations of timing needs into grants of future time

---

Enables:

- Advance guarantees to applications, or
- Denial of requests up front

Introduction

Rialto Background

Windows NT Implementation

Performance and Traces

Related Work and Conclusions

Abstraction: CPU Reservation

- **Guaranteed execution rate and granularity for a thread**
  - ***X* units of time out of *every Y* units, e.g.**
    - 1ms every 5ms
    - 7.5ms every 33.3ms
    - one second every minute

Abstraction: Time Constraint

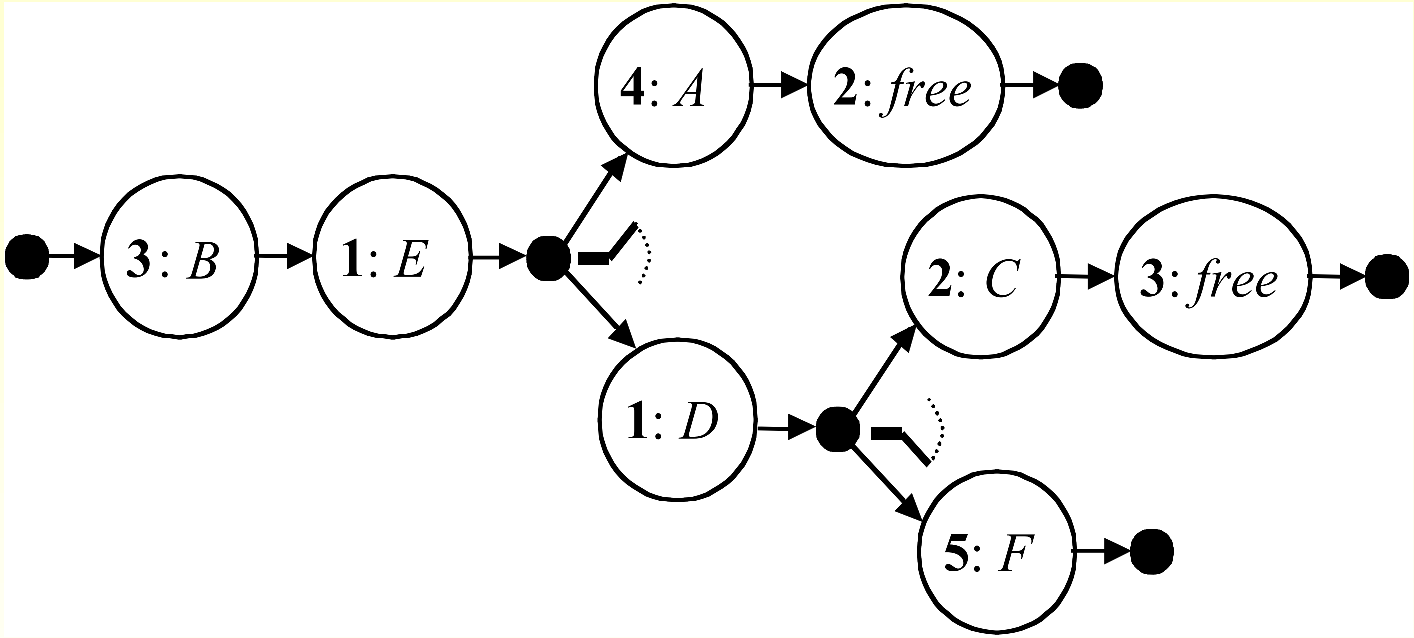
- **Deadline-based thread execution**
  - **Guarantees execution within interval, or**
  - **Proactively denies constraint request**

```
schedulable = BeginConstraint (time_interval, estimate);
if (schedulable) then
    Do normal work under constraint
else
    Transient overload -- shed load if possible
time_taken = EndConstraint ();
```

Implementation: Precomputed Scheduling Plan

- Tree-based periodic map of time
  - Supports widely varying periods
- Allocation of future time intervals
  - Ongoing for CPU Reservations
  - One-shot for Time Constraints
- Enables efficient:
  - Scheduling decisions
  - Feasibility analysis for constraints
  - Guarantees for reservations, constraints

Scheduling Plan Example



Thread	A	B	C	D	E	F
Amount	4ms	3ms	2ms	1ms	1ms	5ms
Period	20ms	10ms	40ms	20ms	10ms	40ms

Introduction

Rialto Background

Windows NT Implementation

Performance and Traces

Related Work and Conclusions

Using the Windows NT Scheduler

- **Rialto/NT uses existing priority scheduler to schedule its threads**
  - **Rialto/NT elevates thread priorities to cause dispatching**
- **Existing apps, abstractions work as before**
- **Windows NT scheduler also can schedule Rialto/NT threads**

Multiprocessor Issues

- One scheduling plan per processor
  - Tree walking happens on all plans
  - Heuristic: add new reservation to plans in increasing order of processor utilization
- Plans *not* pinned to particular CPUs
  - Allow NT scheduler to choose CPU
  - Rely on schedule properties, affinity to run threads mostly on same CPU
  - Permits opportunistic scheduling on other processors by existing scheduler

Affinity vs. Priority

- **Rialto/NT counts on priority elevation to cause thread dispatching**
  - **No contention because at most one elevated (Rialto/NT) thread per CPU**
- **On MP highest priority threads not always scheduled**
  - **Heuristics sometimes elevate thread affinity over thread priority**
- ***Changed scheduler to immediately dispatch Rialto/NT elevated-priority threads when ready***

Discrete Time

- **Windows NT clock interrupts on periodic basis**
  - **Typically 10-15ms, HAL-dependent**
  - **Can usually be set to 1ms period**
- **Discrete interrupts limit enforceable scheduling granularity**
- **So, Rialto/NT:**
  - **Initializes interrupt period to 1ms**
  - **Aligns rescheduling with clock interrupts**

Implementation Details

- Reschedule runs in DPC context
  - Use NT kernel timers to schedule DPCs
- Rialto/NT threads run at priority 30
  - Second highest Windows NT priority
  - Other values could be chosen
- New plans for reservations computed in requesting thread context
  - Optimistic concurrency control to avoid perturbing existing schedule

Non-invasive Implementation

- Easier to argue correctness
- Modified only two kernel routines
  - Changed behavior of one
- Added to the kernel:
  - 6000 lines of C
  - 4 system calls

Complication: Unpredictable Dispatch Latency

- **When latency occurs we:**
  - **Penalize the running thread**
  - **Keep the schedule on time**
- **Causes of scheduling latency:**
  - **Interrupt handlers**
  - **Kernel code running at high IRQL**
  - **Long DPCs**
- **Latencies controllable through concerted latency testing discipline**

Better Living Through Simulation

- **Rialto/NT runs in simulator in addition to kernel**
  - **Exactly the same sources**
- **Makes some debugging *much* easier**
  - **Reproducible runs**
  - **Better tools**
  - **No race conditions**
  - **Reboot time not in critical path**

Introduction

Rialto Background

Windows NT Implementation

Performance and Traces

Related Work and Conclusions

Test Platform

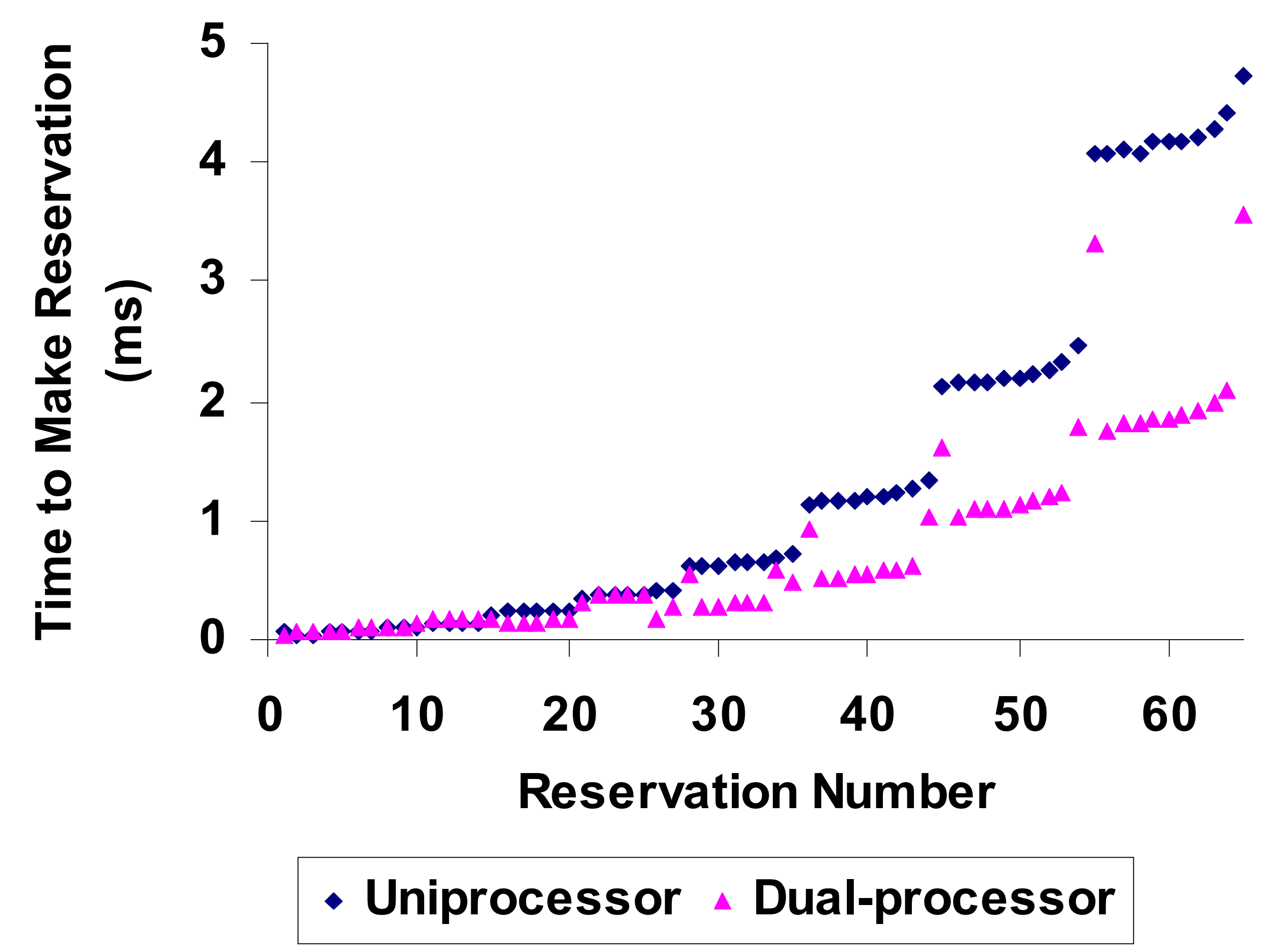
- 333 MHz Pentium II PCs
  - 128MB RAM
  - Intel EtherExpress Pro
  - Adaptec SCSI
- Single- and dual-processor tests

Context Switch Time

- **Tested:**
  - **10 threads on released Windows 2000 beta 3**
  - **10 Rialto/NT threads with CPU Reservations**
- **Rialto/NT adds 8μs to median context switch time**
  - **0.8% overhead at 1ms scheduling granularity**

Time to Acquire Reservations

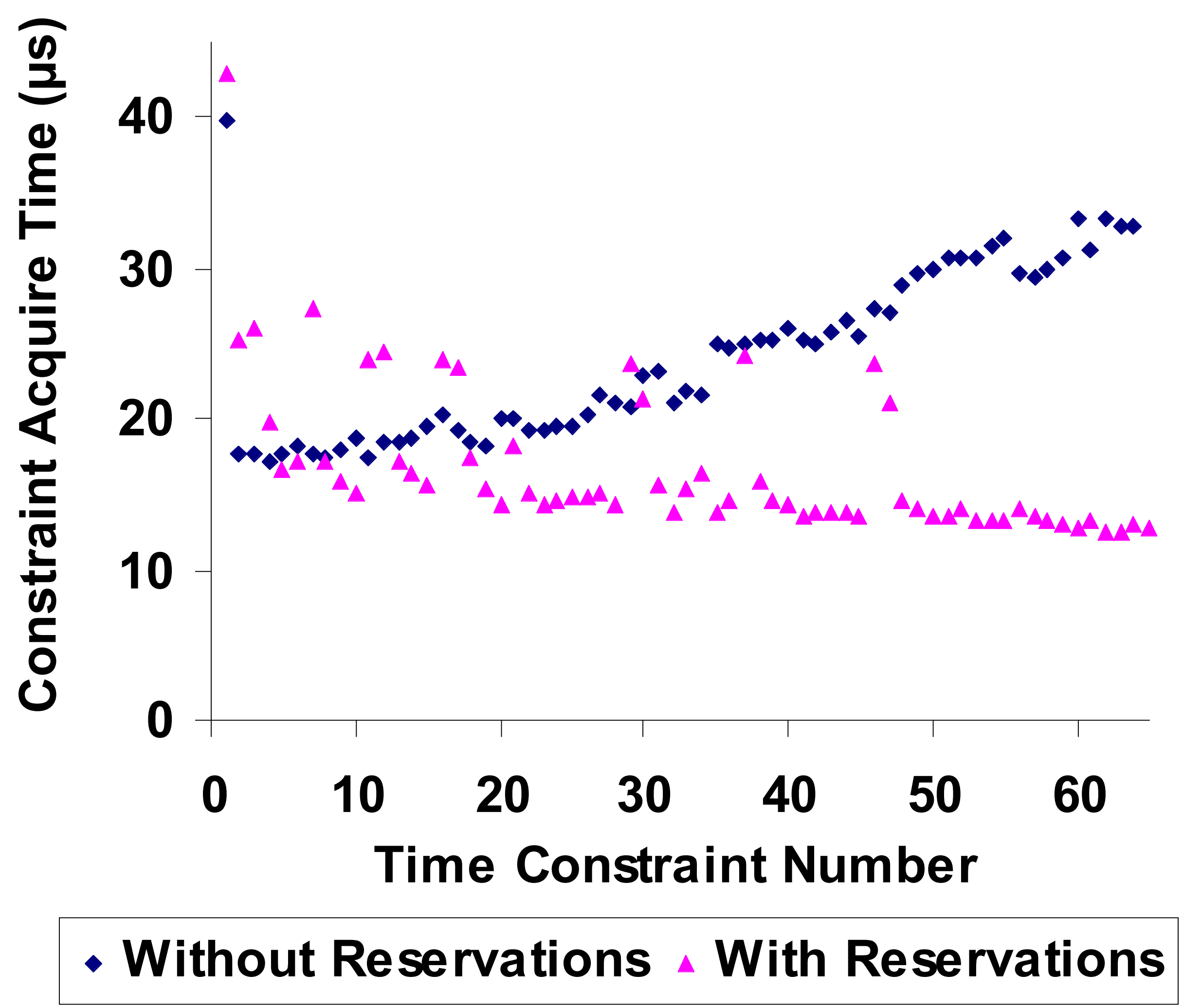
Reasonable even in pathological cases



Times to begin simultaneous reservations

Time to Acquire Constraints

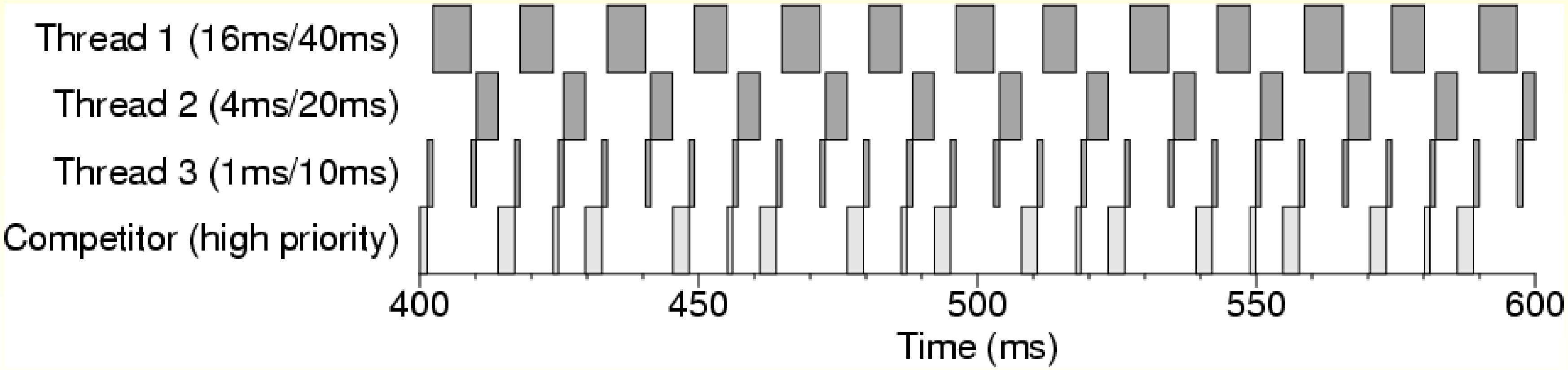
Reasonable even in pathological cases



Times to begin simultaneous constraints

Reservations with a Background Thread

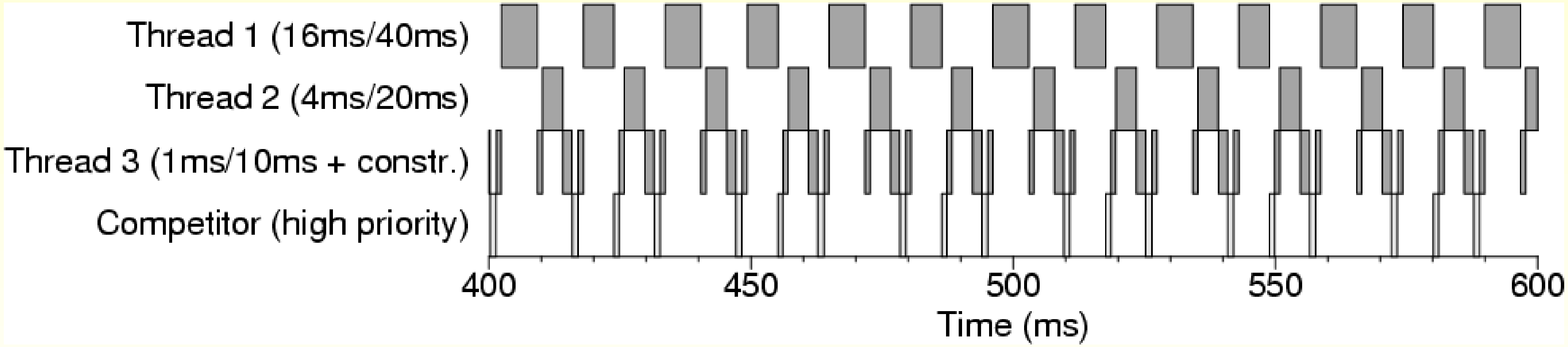
Threads run only during time assigned to their reservations



1 processor, 3 threads with reservations, 1 high-priority competitor thread

Reservations and Constraints

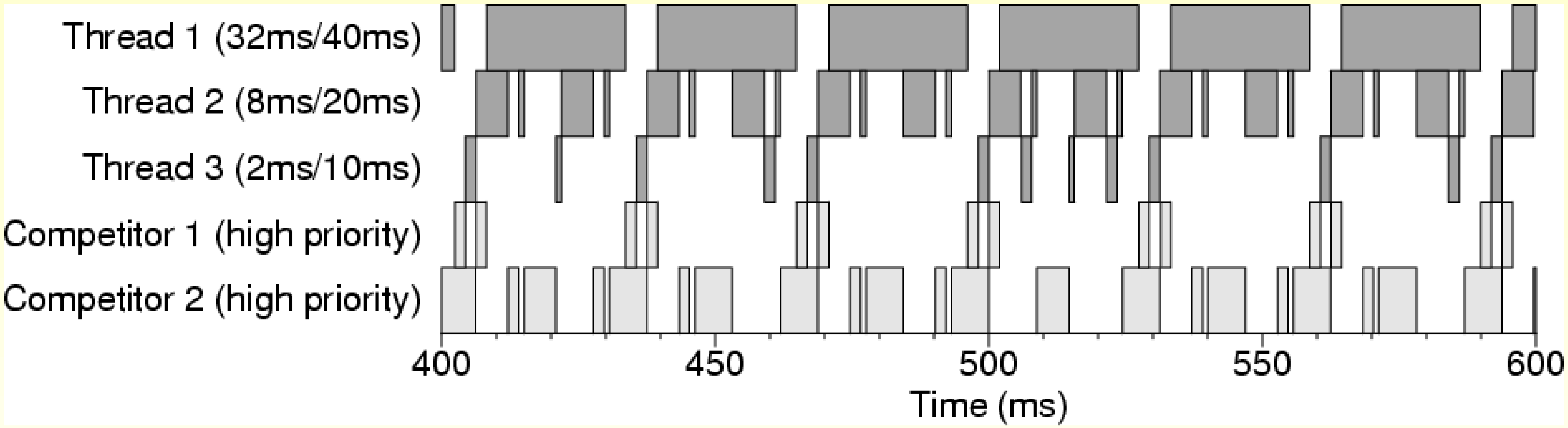
Thread 3 gains additional time with constraints



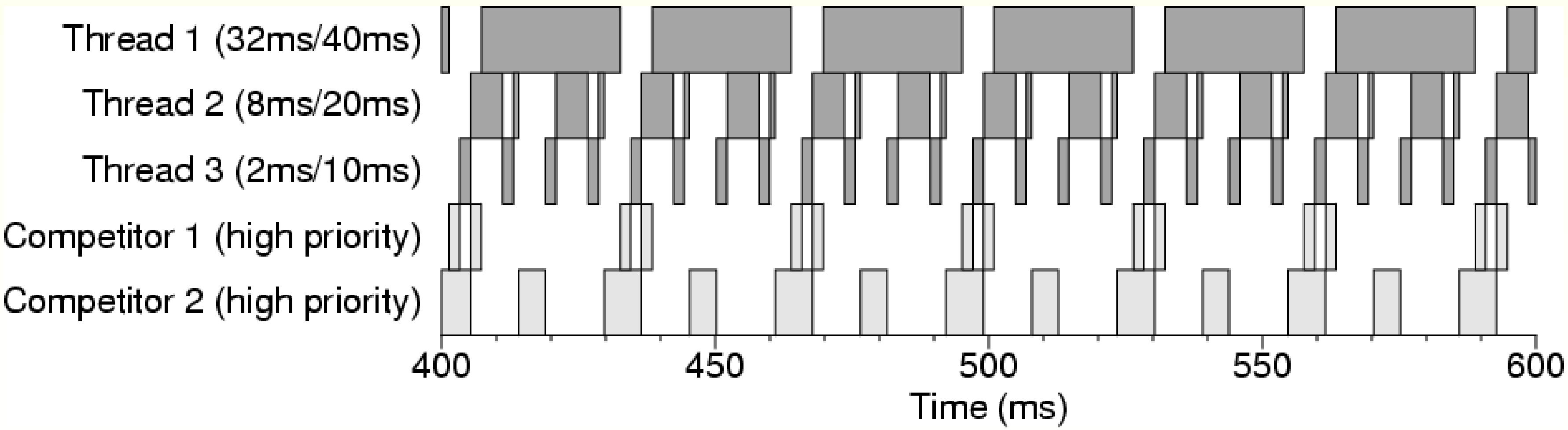
1 processor, 3 threads with reservations (one also using constraints), 1 high-priority competitor thread

Dual Processor Traces

Without affinity change: thread 3 not always scheduled



With affinity change: all threads properly scheduled



Introduction

Rialto Background

Windows NT Implementation

Performance and Traces

Related Work and Conclusions

Related Work

- Real-time add-ins for Windows NT
  - RTX from VenturCom, INtime from RadiSys, Hyperkernel from Imagination Systems
- Lin et. al '98
  - Windows NT soft real-time scheduling server
- Candea & Jones '98
  - Vassal loadable scheduler framework
- Lots of reservation- and deadline-based scheduling work

Further Research

- Evaluate when applied to real apps
  - Some work submitted to RTAS 2000
- Lots of policy issues
  - Resource management
  - Placement of reservations among CPUs

Conclusions

- Scheduling plan effective on MPs
- Plan adapted to use periodic clock
- New scheduler cooperatively coexists with, uses Windows NT scheduler
- Rialto/NT brings CPU Reservations and Time Constraints to Windows NT

Thanks for Your Invitation!

- **References:**
  - **Latency study published in 1999 RTAS**
  - **Rialto/NT published in 1999 USENIX Windows NT Symposium**

- **For more on this research see http://research.microsoft.com/~mbj/**
- **For a *great* priority inversion story be sure to follow the What really happened on Mars? link**