



An Overview of Realtime Linux
Clark Williams
Senior Architect, Red Hat, Inc.

Realtime Linux

Who, What, When, Where and Why?
(and How)

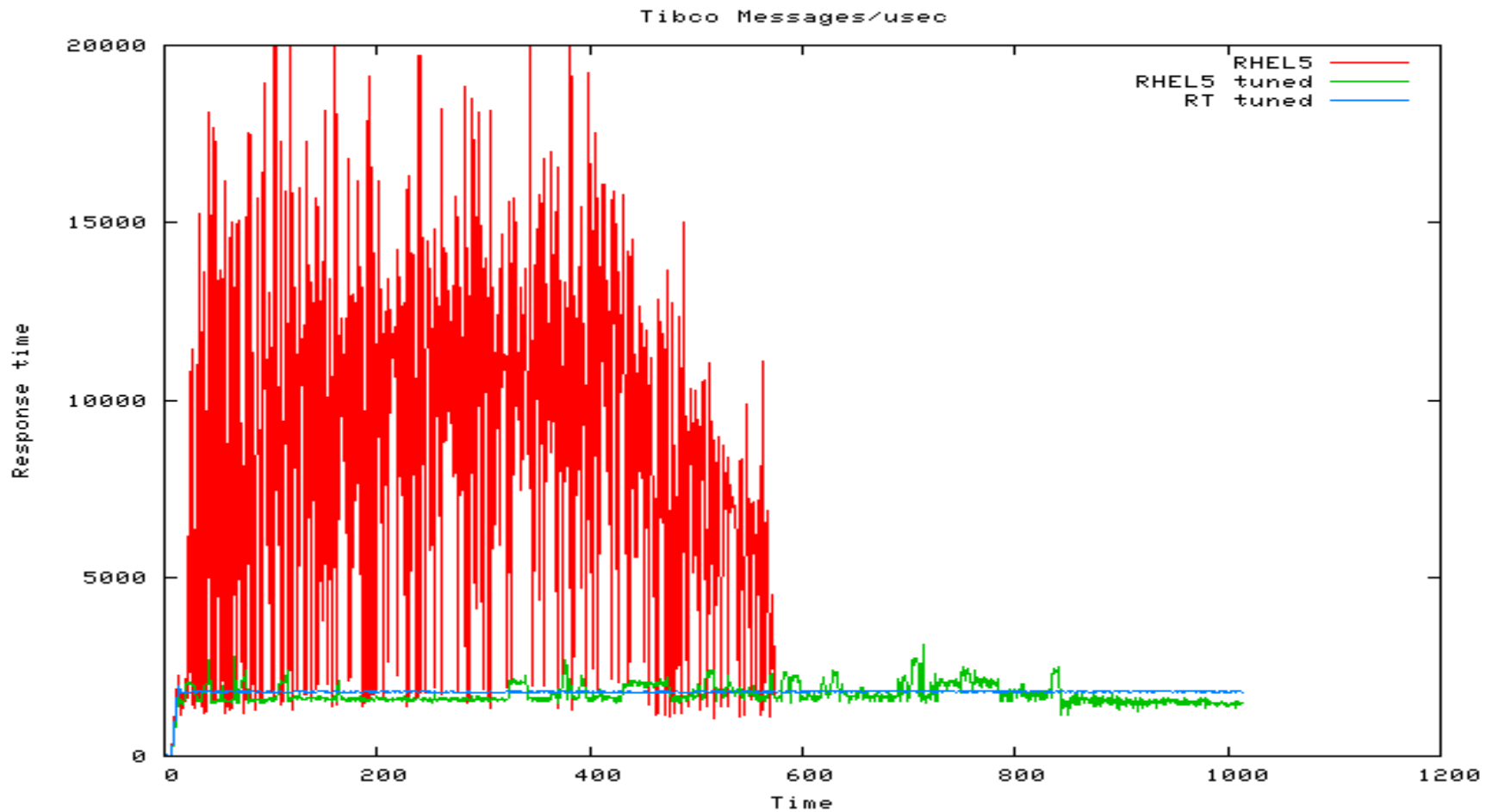
Realtime Linux

- A Modified Linux kernel that targets
 - Fast response to events
 - Consistent response times
 - Highly precise timers
- C library Interfaces to the kernel
 - POSIX threads
 - System calls for scheduler manipulations
- Measurement Tools
- Tuning Tools

Who does this stuff?

- Ingo Molnar (Red Hat)
- Thomas Gleixner (Red Hat contractor)
- Steven Rostedt (Red Hat)
- Paul McKenney (IBM)
- John Stultz (IBM)
- Gregory Haskins (Novell)
- Peter Zijlstra (Red Hat)
- many others...

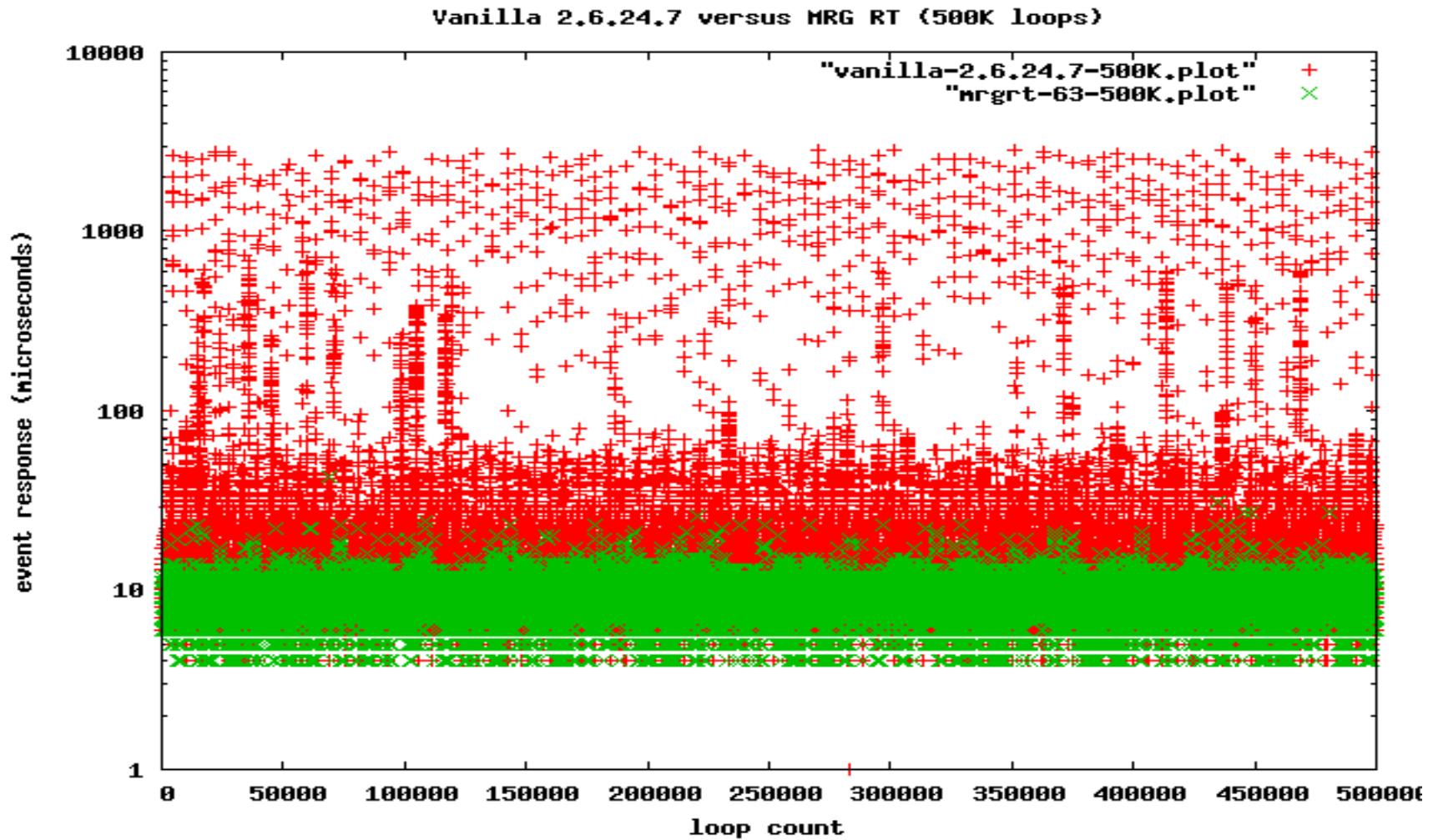
What does it do?



What does it do? (continued)

- The next slide shows data from a program named ***cyclictest***
 - *cyclictest* measures the delta from when it's *scheduled* to wake up from when it *actually does* wake up.
 - Test runs on two kernels:
 - vanilla 2.6.24.7 kernel
 - RT kernel based on the same source
 - 500,000 loops
 - ***hackbench*** load running in the background

What does it do? (continued)



What does that mean ***statistically?***

Vanilla (500K samples)

- Min: 1
- **Max: 2857**
- Mean: 11.47
- Mode: 9.00
- Median: 9.00
- **Std. Deviation: 54.94**

MRG RT (500K samples)

- Min: 4
- **Max: 43**
- Mean: 8.34
- Mode: 8.00
- Median: 8.00
- **Std. Deviation: 1.49**

What changed in the RT Kernel?

- Preemption
 - Most locks converted to `rt_mutex`
 - priority inheritance for mutexes
 - threaded interrupt handlers (both hard and soft)
 - Spinlocks can sleep
 - Interrupts not turned off for almost all operations
- high-resolution timers
- Completely Fair Scheduler (CFS) *
- Read-Copy-Update (RCU) *
- Ftrace tracing logic

* *now in upstream kernel*

What changed in the C library?

- pthread_mutex_t has kernel support for PRIO_INHERIT
 - *Priority Inheritance* is a mechanism used to avoid the deadlock condition known as *Priority Inversion*
 - The RT kernels implement priority inheritance (PI) in *futexes* (fast user-space mutexes) used by pthreads
- Fast user-space mutexes (futexes) used for pthread mutexes
- POSIX interfaces to scheduler APIs
 - sched_*
- Timer interfaces
- Note that you don't have to have an RT kernel for most of these APIs to work

What Measurement tools Are available?

- rt-tests from Thomas Gleixner
 - cyclicttest
 - signaltest
 - pi_stress
- LTP realtime tests
 - sched_latency
 - sched_football
 - pi-tests

What Tuning tools are available?

■ **tuna**

- isolate processor cores
- adjust thread priorities
- change interrupt affinity

■ **ftrace**

- rt kernel built-in mechanism to trace events

■ **oprofile**

- system-wide sampling profiler

■ **systemtap**

- custom data probes for kernel space

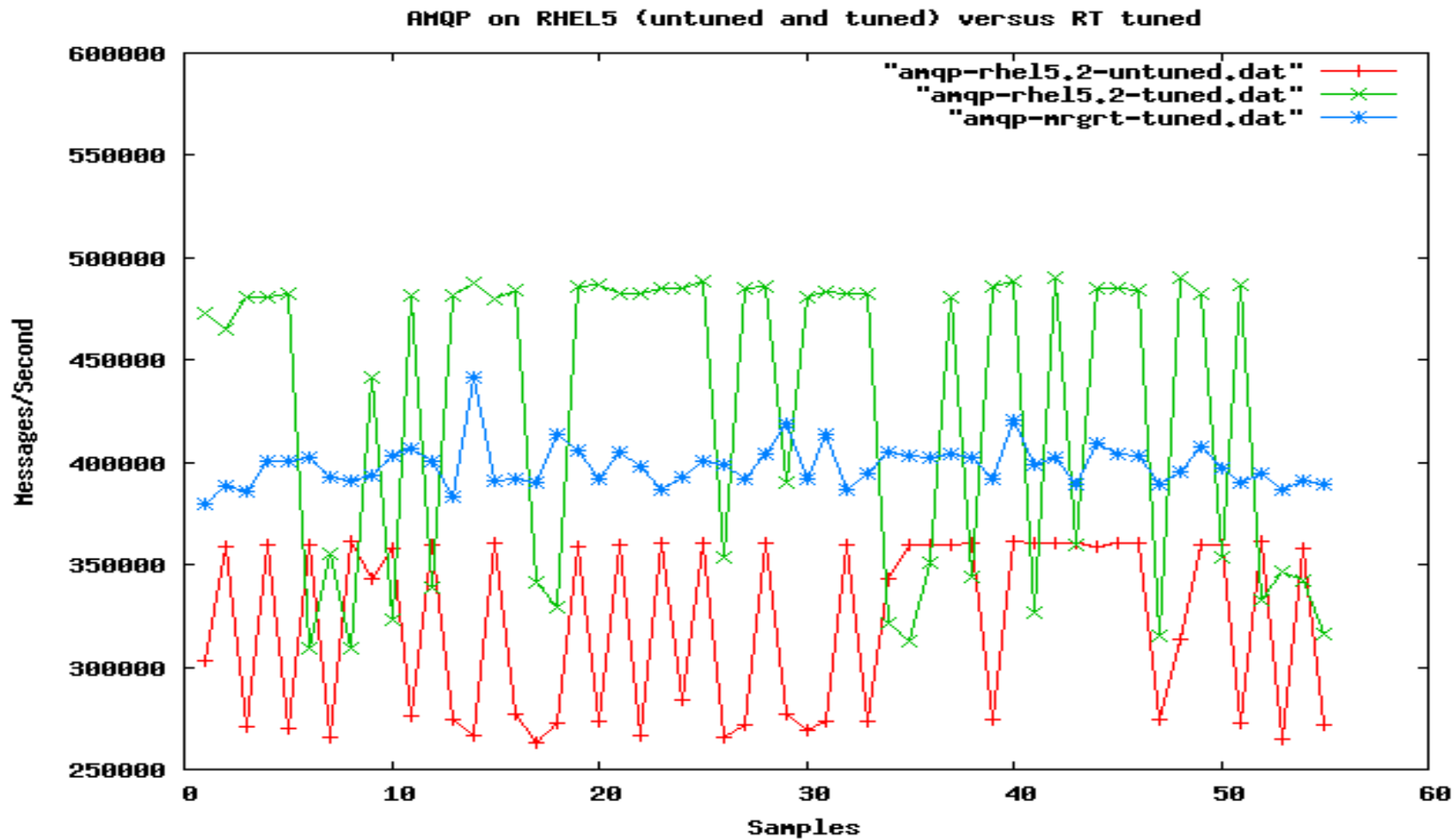
When did this RT stuff happen?

- First steps (2000-2004)
 - Ingo Molnar / Andrew Morton – low latency patch
 - Robert Love – preemption patch
- Current State (2004 - today)
 - First started on 2.6.9 kernel – Ingo Molnar's realtime patch
 - Originally called realtime-preempt patch
 - Latest is 2.6.25.4-rt4
 - HRT, RCU and CFS merged upstream
 - new latency tracer (ftrace)

When does RT help?

- When you need fast, consistent response times
- The following slide shows the difference between running the QPID messaging application on a stock Linux kernel and an RT Linux kernel
 - Y-axis is messages-per-second
 - X-axis is sample point (55 samples)

When does RT help? (continued)



When does RT help? (continued)

- In the previous slide, the tuned RHEL5.2 kernel actually processed more messages than tuned RT kernel
 - The main focus of the MRG RT kernel is determinism over raw throughput
- The tuned RT kernel was MUCH more consistent in its processing times
 - Standard Deviation of messages/sec on RT kernel is much smaller than on either of the RHEL runs
 - This consistency is the biggest strength of RT

Where is it used?

- Financial systems
 - Market data applications
 - Analysis applications
 - Message-passing platforms
- Command, Control and Communications systems
 - shipboard data systems interfaces
- Any application that cares about responding promptly to events

Why the drastic kernel changes?

- Running with interrupts off for the majority of times allows preemption (i.e. quicker reschedule to high priority threads)
- Each interrupt event is a potential reschedule point
- If you can be interrupted while holding a lock, you need to be able to come back to that context, so need to be in a thread
- Threaded interrupts allow control of interrupt processing order via thread priorities

Why should I care?

- You should care if you you need to meet deadlines in your application(s)
- Realtime Linux is an effort to make the Linux kernel *deterministically responsive* to events
 - The amount of time which elapses from event occurrence to event handling is bounded.
- Bounded by *what*?
 - dependent on the hardware you're running
 - Given the same load, a 3 GHz processor will respond to an event faster than a 700 Mhz processor
 - Better to say that we're trying to make the response time more consistent (i.e. reduce latency standard deviation)

How can I use it?

- Tune your system
 - No two applications behave the same
 - use *tuna* to tweak priorities and affinities
 - use *oprofile* to find application hotspots
 - use *ftrace* to find long latency areas
- Dedicate processors to your application threads
 - Use *tuna* or *taskset* to bind threads to specific processors and move other threads off
 - 4-way and 8-way processors getting cheaper
- Use cpu affinity field in `/proc/irqs/<n>/smp_affinity` to bind interrupts to specific processors
 - *tuna* can do this easily

Tuna Screenshot

Filter	CPU	Usage	IRQ	PID	Policy	Priority	Affinity	Events	Users
<input checked="" type="checkbox"/>	0	4	0	-1		-1	0,1	809	timer
<input checked="" type="checkbox"/>	1	1	1	391	FIFO	50	0,1	8	i8042
			3	3038	FIFO	50	0,1	1	
			4	409	FIFO	50	0,1	465	serial
			6	1751	FIFO	50	0,1	3	floppy

PID	Policy	Priority	Affinity	VolCtxtSwitch	NonVolCtxtSwitch	Command Line
1	OTHER	0	0,1	19612	1689	init [3]
2	OTHER	0	0,1	174	210	kthreadd
3	FIFO	99	0	507	0	migration/0
4	FIFO	99	0	2	0	posix_cpu_timer
5	FIFO	50	0	2	0	sirq-high/0
6	FIFO	50	0	93509037	0	sirq-timer/0
7	FIFO	50	0	5	0	sirq-net-tx/0
8	FIFO	50	0	400075	1	sirq-net-rx/0
9	FIFO	50	0	65038	0	sirq-block/0
10	FIFO	50	0	338	0	sirq-tasklet/0
11	FIFO	50	0	22278917	0	sirq-sched/0
12	FIFO	50	0	65	0	sirq-hrtimer/0
13	FIFO	50	0	2058939	0	sirq-rcu/0
14	FIFO	99	0	2	0	watchdog/0
15	OTHER	0	0	40007	235	desched/0
16	FIFO	99	1	431	0	migration/1
17	FIFO	99	1	2	0	posix_cpu_timer
18	FIFO	50	1	2	0	sirq-high/1
19	FIFO	50	1	93512225	0	sirq-timer/1
20	FIFO	50	1	3	0	sirq-net-tx/1
21	FIFO	50	1	39260	0	sirq-net-rx/1
22	FIFO	50	1	38679	1	sirq-block/1
23	FIFO	50	1	113	0	sirq-tasklet/1
24	FIFO	50	1	22511725	0	sirq-sched/1
25	FIFO	50	1	3120	0	sirq-hrtimer/1
26	FIFO	50	1	2027617	0	sirq-rcu/1
27	FIFO	99	1	2	0	watchdog/1
28	OTHER	0	1	33935	229	desched/1

How can I use it? (continued)

- use POSIX threads
 - finer grained applications mean more parallelism, so can take advantage of multiple cores
- Use POSIX threads synchronization mechanisms
 - Mutexes
 - Barriers
 - Condition variables
- Set appropriate priorities for your threads
 - Any SCHED_FIFO thread is higher priority than any SCHED_OTHER thread
 - ensure that you high priority threads don't hog the processor

How can I get it?

- Red Hat Messaging, Realtime, Grid (MRG)
 - Separate product
 - Layered on Red Hat Enterprise Linux 5.2
 - Kernel is 2.6.24.7-rt11 based
 - Supports i686 and x86_64 architectures only
 - Comes with tuna and other support packages
 - No application changes required for RHEL5 applications
 - No recompiles needed

Questions?

Clark Williams
williams@redhat.com