RED HAT :: SAN DIEGO :: 2007
SUMMIT

Tuning for financial services

Rod Nayfield

11 May 2007

How to do it?

How to do it?

Buy this exact piece of hardware

FHU-348399-MFJJ/A

Set the following parameters for /dev/null

fast = yes ; **DWIM** = on

Use Token Ring or DECNet

Use 120v power

240v is a long flight away (reverse for europe)

Somebody set up us the bomb.

RED HAT :: SAN DIEGO :: 2007 SUMMIT

Thank you for attending.

Latency

Where does it come from?

How can I reduce it?

There are many factors that impact determinism

Performance tuning is an iterative process involving comparative testing

This presentation is an overview of many simple things

you can test right now.

Problem Description

It's not *just* about going faster

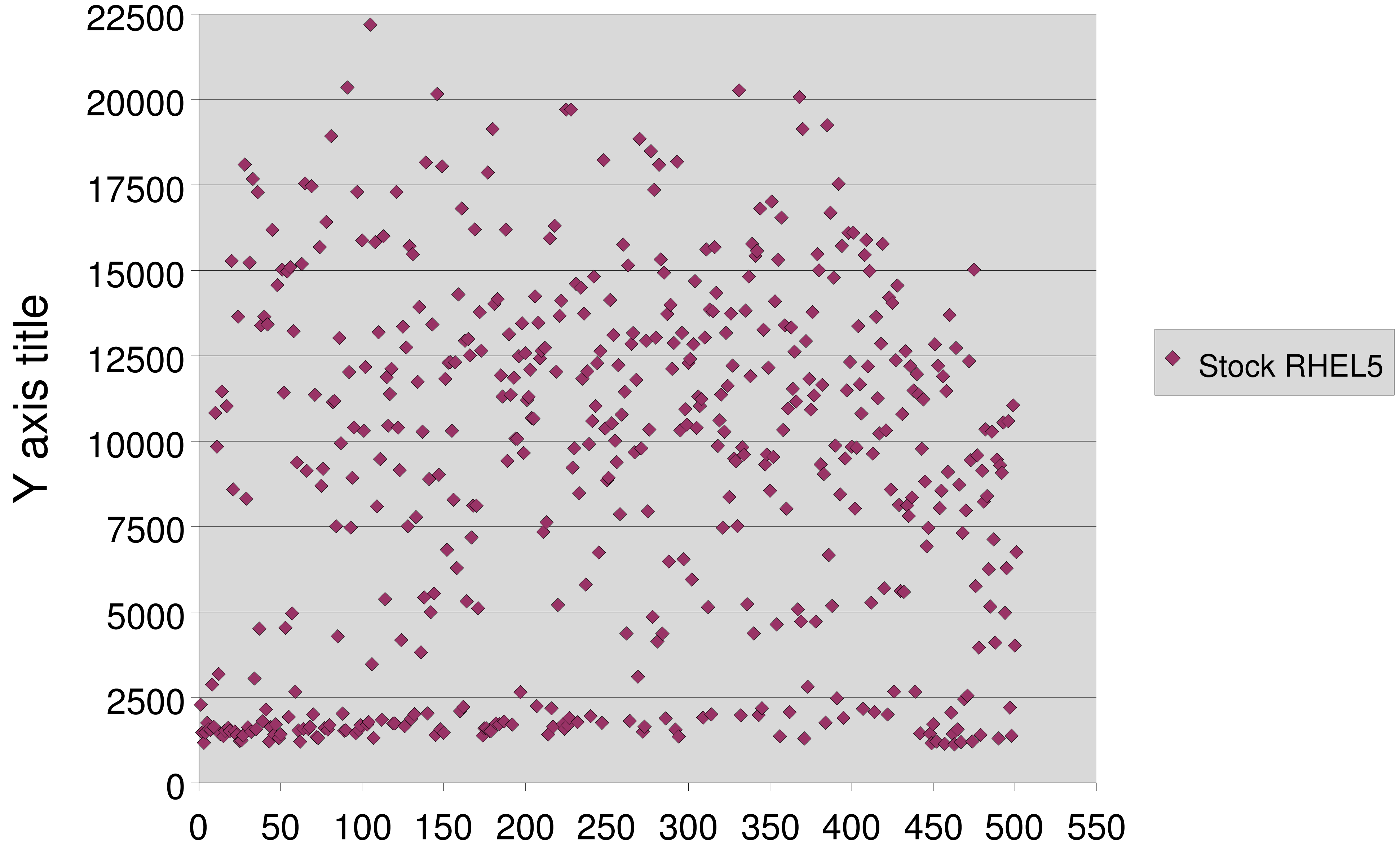
Increasing determinism of events on a small scale

- Example: 10,000 transactions:
    - 9,950 at 1ms, but 50 above 20ms not acceptable
    - 9,999 at 2ms acceptable

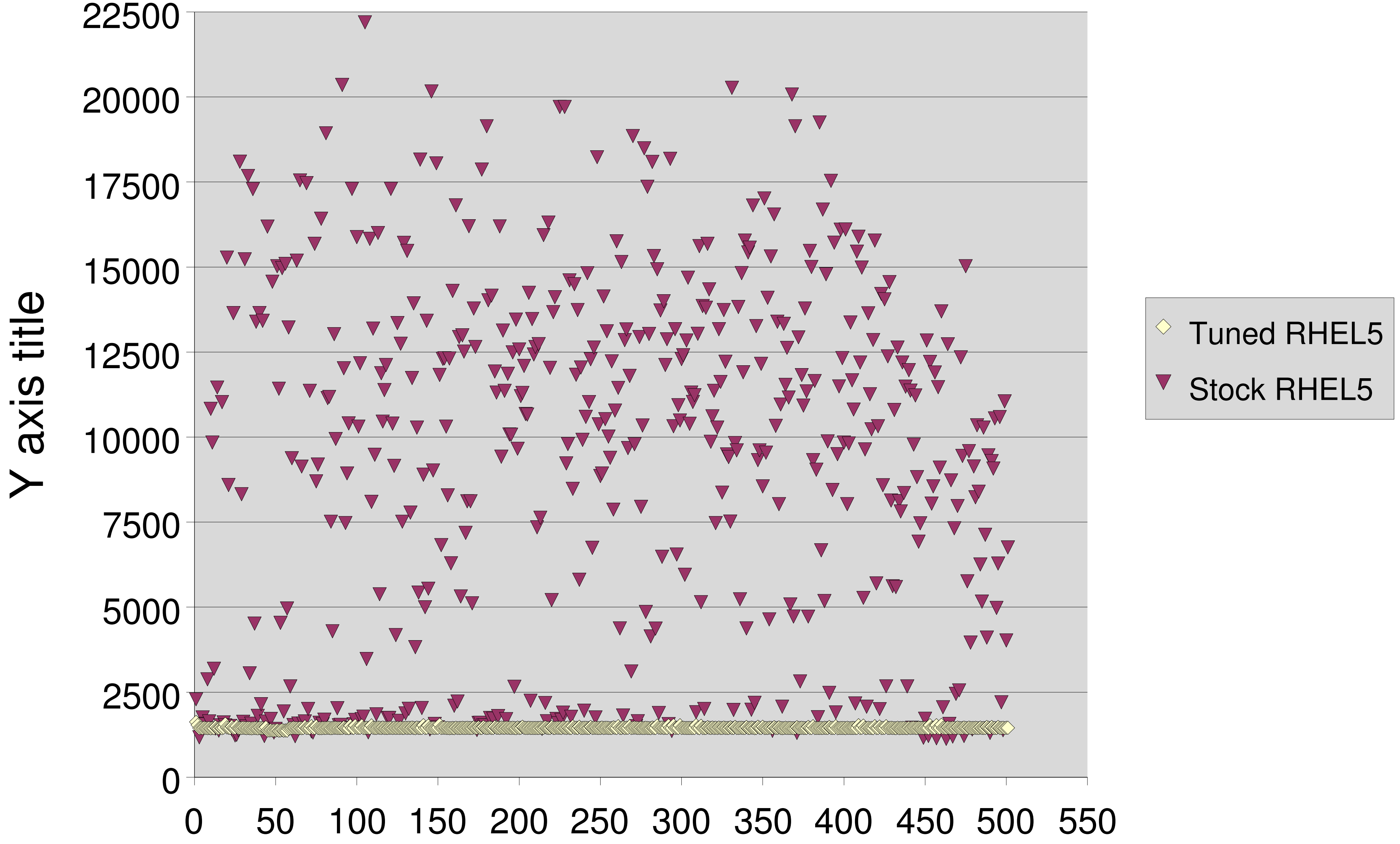
Why?

- Service Level Agreements
- Regulations
- And other big consequences

Stock vs. Tuned



Stock vs. Tuned



Write better code

Or have your vendors write better code

analysis often show design issues

- malloc()/free() over and over and over
- gettimeofday() 30x more than needed
- re-reading and re-parsing config files 58 times
- re-building caches even though source files have not changed

Problems in your app code aren't going to be solved with performance tuning

Don't run extra stuff

The oldest trick in the book

- You'd be surprised at what we've found
  - Full GNOME desktop on servers for console over IP
  - Sendmail
  - Rpc stuff, nfs, etc
  - Console mouse (gpm)
- Anacron jobs
- Kernel modules

- Also, your third party apps
  - Ex: Hardware monitoring modules can be invasive

Vm behavior

Swapping pages out to disk can introduce latency in any environment

- Size your system physical RAM for your application
- Vmstat – watch at si/so
- swappiness – try lowering
- dirty_background_ratio – try making it higher, pdflush starts later

Future: RHEL 4.6 proposed patch to prevent system from swapping when there are unmapped pagecache pages to reclaim

Filesystems

- Journal activity can introduce latency
  - Ordering changes, committing metadata, committing data
- Third-party filesystems can do the wrong thing
  - Especially if they re-implement their own untuneable bdflush

- Don't write when you don't have to (noatime, ro)
- Don't journal when you don't have to
  - A journal is designed to speed recovery from a crash
  - Is your data valuable 20 minutes after a crash?

I/O elevators

CFQ is the default elevator. Fairness is for wimps.

Deadline elevator is most deterministic.

- RHEL 4 has a boottime option
  - Change it requires reboot
  - Affects all queues on system
  - elevator=deadline

- RHEL5 allows you to change on the fly
  - And have a different elevator per queue
  - $ echo deadline > /sys/block/<*sdx*>/queue/scheduler

Storage hardware

Slow storage and aio can get you in trouble

A local sold state disk might be the ideal storage medium for some apps

If you must journal, how about external journal on solid state?

AIO is great until you fill it up and go back to sync

- /proc/sys/fs/aio-max-nr is the tunable number of IO events
- /proc/sys/fs/aio-nr shows the max IO events

Prioritize your processes

By default, the scheduler is quite fair.

Wimp!

There are several scheduling policies available, and priorities

By default, you are in SCHED_OTHER and nice(1) controls within that policy

- sched_setscheduler(2)
  - chrt(1) command
  - Example: SCHED_RR, priority 50

Timers and power management

You probably don't want your cpu to be switching speeds while handling critical workloads – turn cpuspeed off

Applications that make lots of gettimeofday() calls are sensitive to timers.

NTP really doesn't like drift (or irqbalance for that matter)

What is your precision need – nanoseconds or milliseconds?

- Pmtimer: Precise, but slow and not scalable
- Hpet: Not available on all platforms. Somewhat slow
- TSC: Fast but P-state drift issues –
    - Use RHEL4.5 or RHEL5.0
    - service cpuspeed off
    - *nohpet nopmtimer*

Process affinity

The scheduler assumes that your apps play well with others

Like putting a 'dd' on the cpu your application is currently using

It's time to play the affinitiy game

- First, clear off some processors
  - taskset existing processes to a mask
  - exodus - people.redhat.com/williams/exodus
- Then run your app on the clear CPUs
  - sched_setaffinity(2) taskset(1)
- Build it into production
  - default_affinity or isolcpus on cmdline
  - Taskset your app in its initscript

Interrupt affinity

Interrupts live on CPUs too. irqbalance(1) moves them around.

Network and storage interrupts tend to be firing all the time

- /proc/irq/*<n>*/smp_affinity contains a mask
- service irqbalance off
- --oneshot option, or balance by hand
- New irqbalance in RHEL5.1

When does the affinity game break?

Lots of application threads

Apps that “take everything you give it”

Lots of network interfaces

Small SMP systems

TCP

- TCP's job is to add latency
  - To obtain efficiency
  - Congestion control
  - Reliable delivery
- Do you really need ordered delivery?
- Do you need retransmissions? Is it worth destroying throughput for one lost packet?
  - Packet loss **can** be ok – retransmit might be worse than loss
- If you must use TCP, turn off Nagle (“the bus doesn't leave until it's full”)
  - use TCP_NODELAY on your socket

TCP – netstat -s examples

```
TcpExt:
    53 packets pruned from receive queue because of socket buffer overrun
    17953 delayed acks sent
    20 timeouts after reno fast retransmit
    23 timeouts after SACK recovery
    21 timeouts in loss state
    431 fast retransmits
    20 forward retransmits
    37 retransmits in slow start
    2504 other TCP timeouts
    1409 packets collapsed in receive queue due to low socket buffer
    269 connections reset due to unexpected data
    322 connections reset due to early user close
```

Network queues and buffers

sysctl.conf:

```
  # increase TCP max buffer size
  net.core.rmem_max = 16777216
  net.core.wmem_max = 16777216
  # min, default, and max number of bytes to use
  net.ipv4.tcp_rmem = 4096 87380 16777216
  net.ipv4.tcp_wmem = 4096 65536 16777216
  net.core.netdev_max_backlog = 2500
```

```
   ifconfig eth0 txqueuelen 1000
```

Network

- Interrupt coalescing
  - Ethtool -C
- Speed of light
  - Ip-over-wormhole still under development
- Queuing on switches
- Congestion
  - 802.3x back pressure on speed changes
    - Ethtool -A can be used for band aid
- Infiniband
  - IP over IB
  - RDMA

Go to RHEL5

Many enhancements from the realtime tree have arrived in stock RHEL5

Items 2.6.18 and prior are in RHEL5

Over the last year, many patches have moved from -rt to mainline kernel:

- BKL preemptable (2.6.8)
- Mutex patch (2.6.16)
- Semaphore-to-Mutex conversion (ongoing ~85% done)
- Hrtimers subsystem (2.6.16)
- Robust futexes (2.6.17)
- Lock validator (2.6.18)
- Priority inheritance futexes (PI-futex) (2.6.18)
- Generic IRQ layer (2.6.18)
- Core time re-write (2.6.18)

Go to RHEL5 realtime

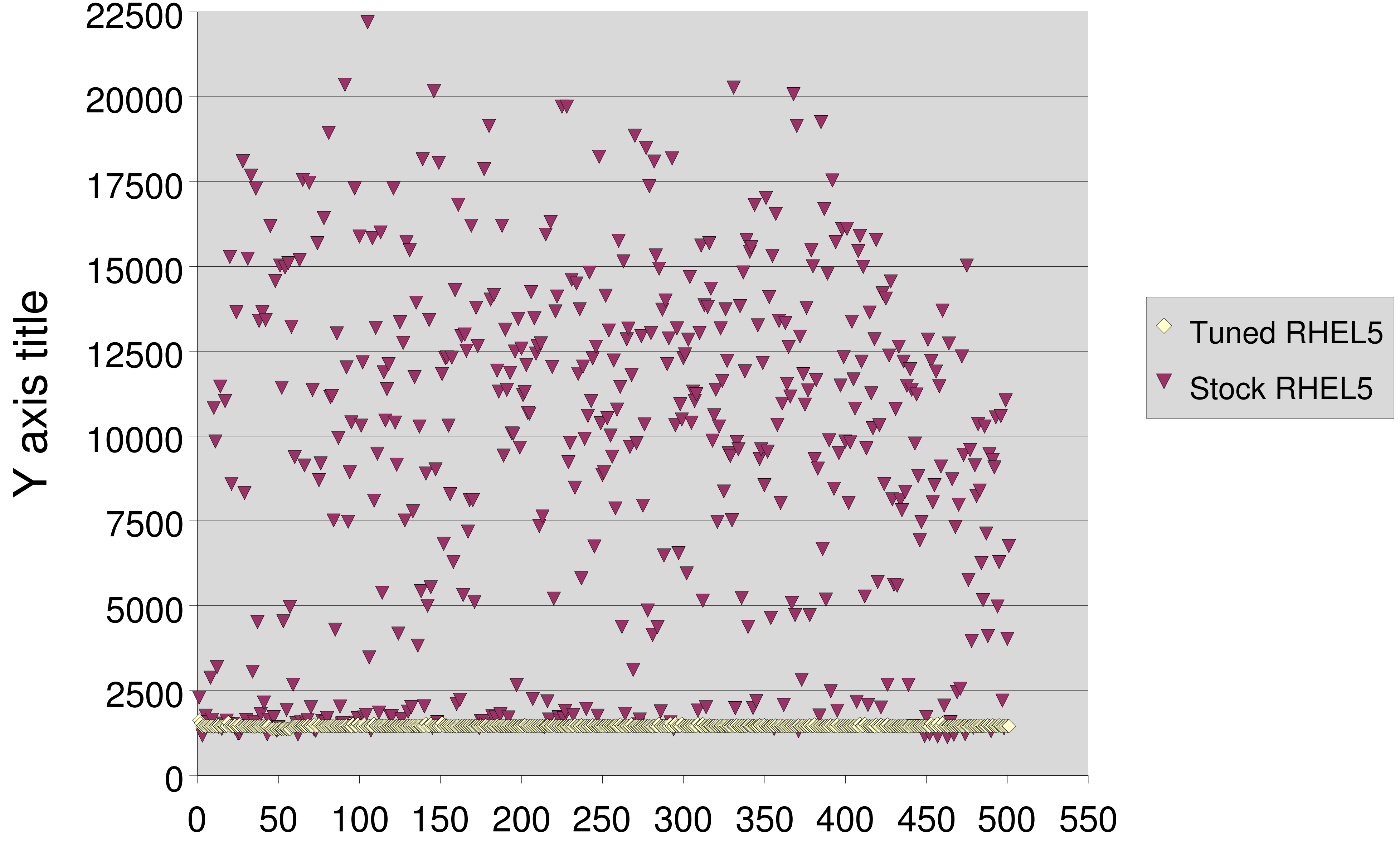
Many enhancements from the realtime tree have arrived in stock RHEL5

Items 2.6.18 and prior are in RHEL5

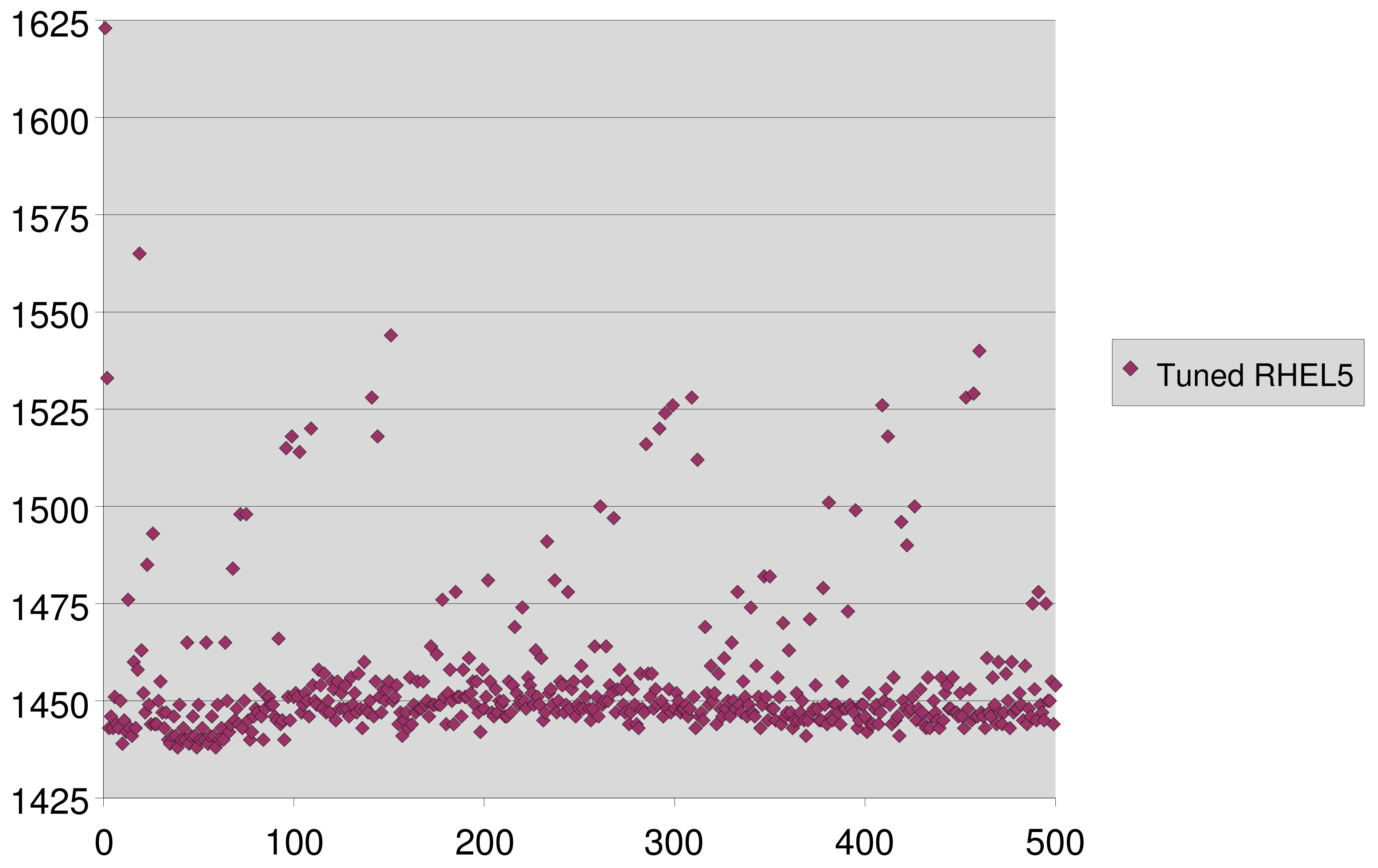
Over the last year, many patches have moved from -rt to mainline kernel:

- BKL preemptable (2.6.8)
- Mutex patch (2.6.16)
- Semaphore-to-Mutex conversion (ongoing ~85% done)
- Hrtimers subsystem (2.6.16)
- Robust futexes (2.6.17)
- Lock validator (2.6.18)
- Priority inheritance futexes (PI-futex) (2.6.18)
- Generic IRQ layer (2.6.18)
- Core time re-write (2.6.18)
- Sleepable RCU (2.6.19)
- Latency Tracer (2.6.19)
- High-res+dynticks (2.6.21)
- Conversion of spin-locks to mutex (2.6.22+)
- All Interrupt handling in threads (~2.6.22+)
- Full rt-preempt (~2.6.23+)

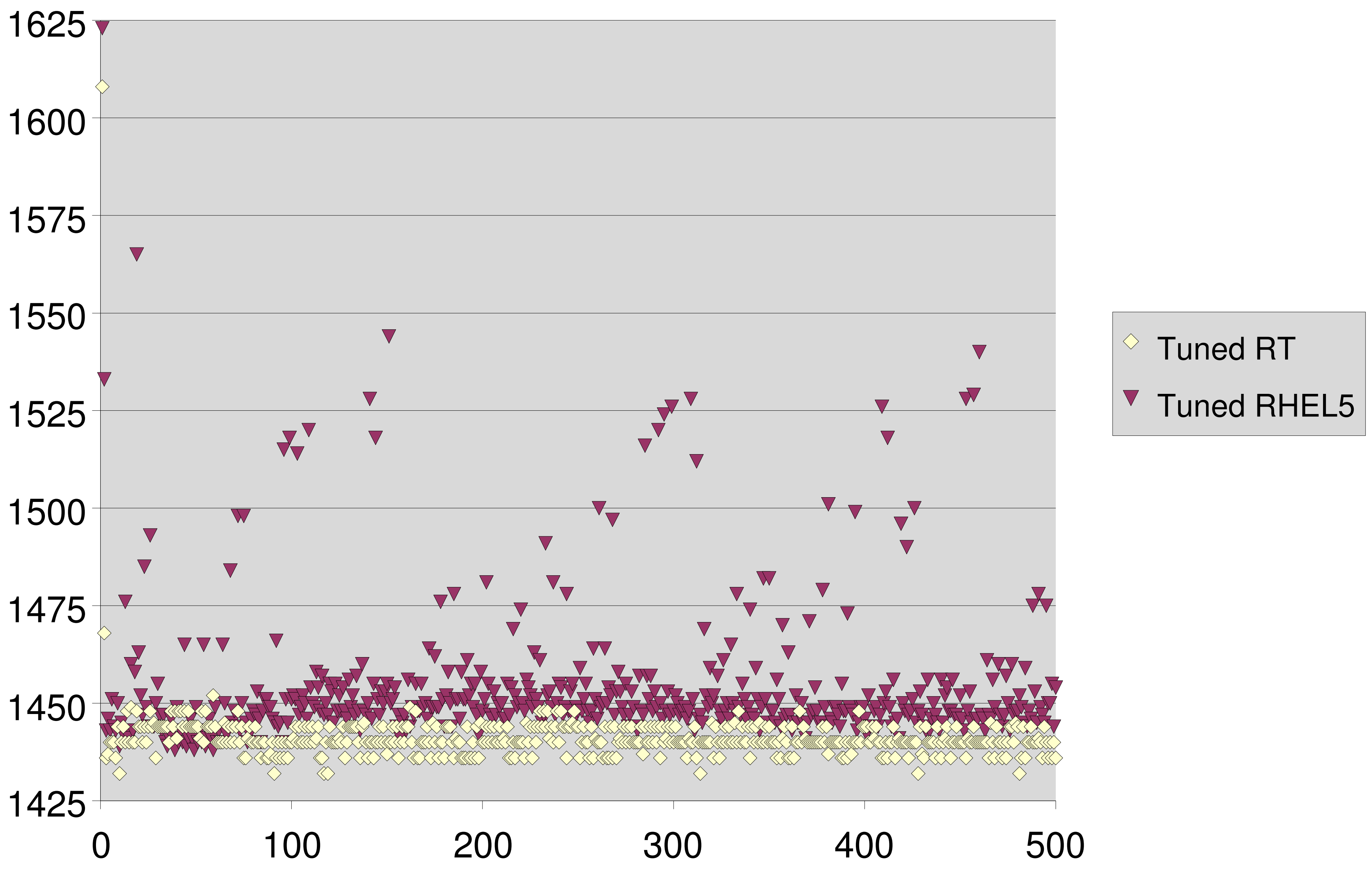
Stock vs. Tuned



Tuned vs. RT


1625
1600
1575
1550
1525
1500
1475
1450
1425
0
100
200
300
400
500
◆ Tuned RHEL5

Tuned vs. RT


1625
1600
1575
1550
1525
1500
1475
1450
1425
0
100
200
300
400
500
◇ Tuned RT
▼ Tuned RHEL5

Example : Red Hat Messaging

Red Hat's implementation of AMQP leverages some of the latest and greatest upstream enhancements (requires real time kernel)

- New event model
  - Kernel scheduled userspace
  - No pools of threads waiting around for work
- New async network i/o model
  - New API – eliminates epoll/select and sockets
  - Network IO at memcpy() speed

Learn more at AMQP sessions and in your Summit materials

Questions?

people.redhat.com/nayfield

RED HAT :: SAN DIEGO :: 2007
SUMMIT