Welcome

On faster application startup times: Cache stuffing, seek profiling and adaptive preloading

bert hubert <bert.hubert@netherlabs.nl>
Netherlabs Computer Consulting BV
PowerDNS.COM BV
http://netherlabs.nl - http://ds9a.nl - http://wiki.powerdns.com

Thanks to: Seth Arnold, Zwane Mwaikambo, Con Kolivas, Alexn, Relayfs people (IBM)

Outline of presentation

- Some theory of how disks appear to work
- Problem statement: know what to solve
- Application startup pessimization: on-demand loading
- Prior art (Andrew `KP' Morton, Linus Torvalds, Windows 95 (Intel))
- New measurements
- Solutions / Discussion

50,000 foot view of disks

- Not as simple as they appear
- Sources of latency
  - PCI/IDE
  - Head positioning
  - Rotational – waiting for data to pass under the head
  - Interrupt, copying data to userspace
- Manufacturers not being very open

Typical disk performance claims

- High-end drive: full-stroke latency of 8ms, track-to-track in 0.3ms
- Silent about rotational latency, we're ass-u-med to know.
- Calculation: Average laptop disk, 5400RPM: 0.5*60/5400 = 5.6ms
- Real life is more like 20ms (!)
- Equivalent to reading 5 megabytes contig.

Our challenge

- While `we' generally achieve month- or year-long uptimes and have staggering amounts of memory, others benefit less from the page-cache.
- **Starting an application should not wait on i/o for much longer than the amount of data it needs would've taken to read linearly**

My limited goal in all this

- Provide patch to do instrumenting
- Provide tools to interpret results
- Make pretty graphs
- Allow other people to improve Linux based on serious measurements
- Bonus: might also be useful to i/o scheduler people

Application startup

- `On-demand loading' – hip in the 80s.
- Means: mmap executable **and its libraries** into memory, and execute away
- `Missing data' will cause page faults, which will trigger actual disk reads – slick, but:
- Data access patterns determined by whims of the linker and call-graph of process!

Prior art

- Several distributions now preload binaries
- akpm has studied contents of the page cache, and attempted to restore it – to no avail
- Arjen van de Ven: readahead doesn't help
- Linus has stated that the only `right' way of doing this is to stuff the page cache from linearly read data – dangerous
- It appears Windows speculatively loads data that was touched on previous boot

What we need is DATA

- Saying which rhymes in Dutch `to measure is to know' – hence our strong scientific achievements :-)
- Anything else is mental masturbation (according to Linus)
- What you don't measure gets subverted (after a while)

Measurements

- Problem: the reads we care about are `un-straceable'
- So, we instrument the bio-layer
- Initially performed using block_dump of laptop_mode, combined with audit subsystem
- Problem: this gives blocks on devices, not file names

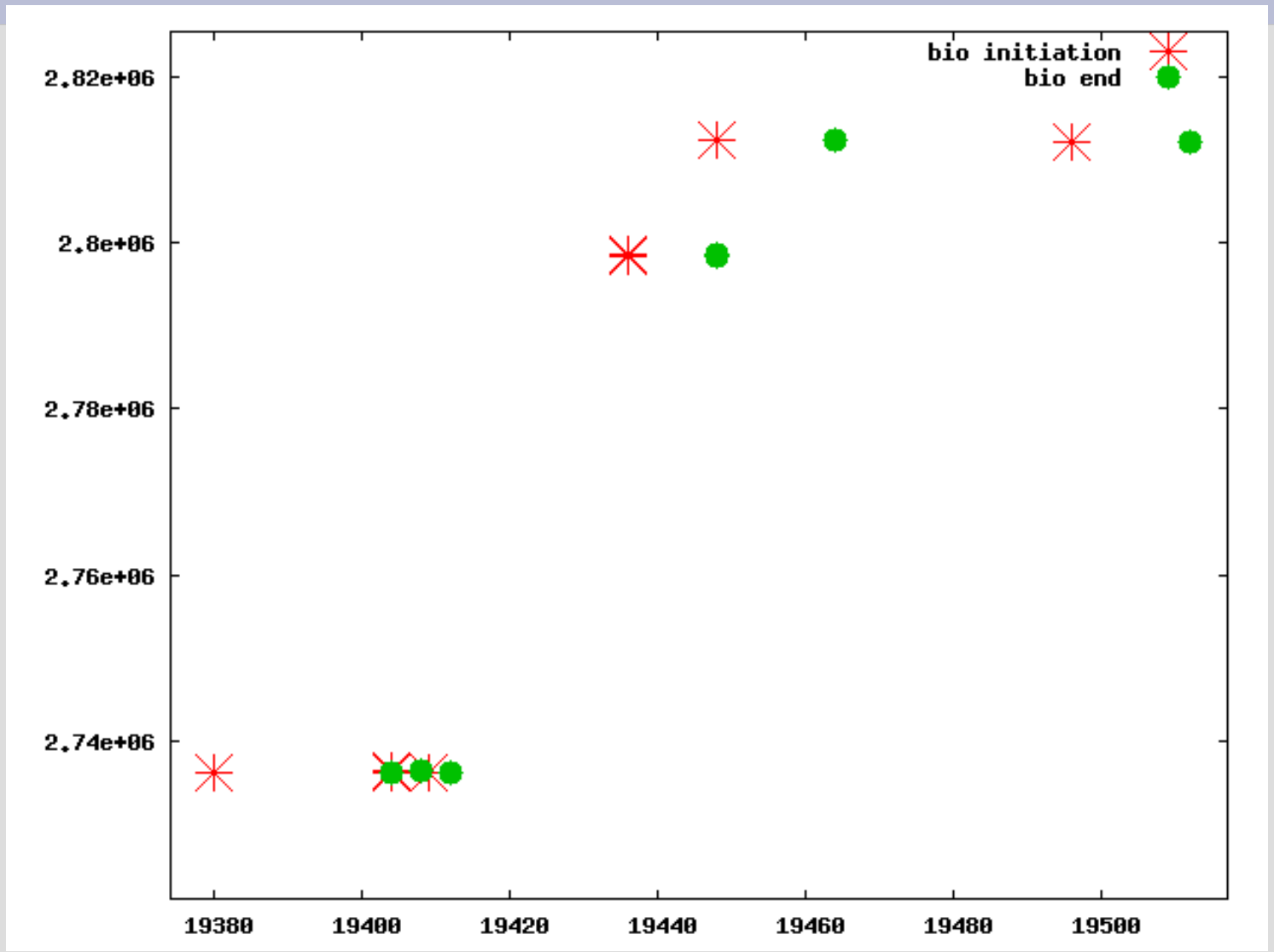
Measurements II

- Solution: instrument sys_open as well
- Use FIBMAP on all opened files to make reverse map of block->file
- To do all this in userspace, transfer data using relayfs to C++ application
- Tiny remaining problem, 'ended' bios are device-relative, they start partition-relative

Measurements III

- Validate traces (count that no bio-requests are duplicates, or end twice), confidence in data is high
- Some duplicate bios: fsck & kernel itself
- Timestamping done using jiffies + tsc, measurements with equal jiffies are shifted tsc for sub-HZ pretty graphs
- And without further ado: **GNUPLOT!**

HD cache for adjacent reads

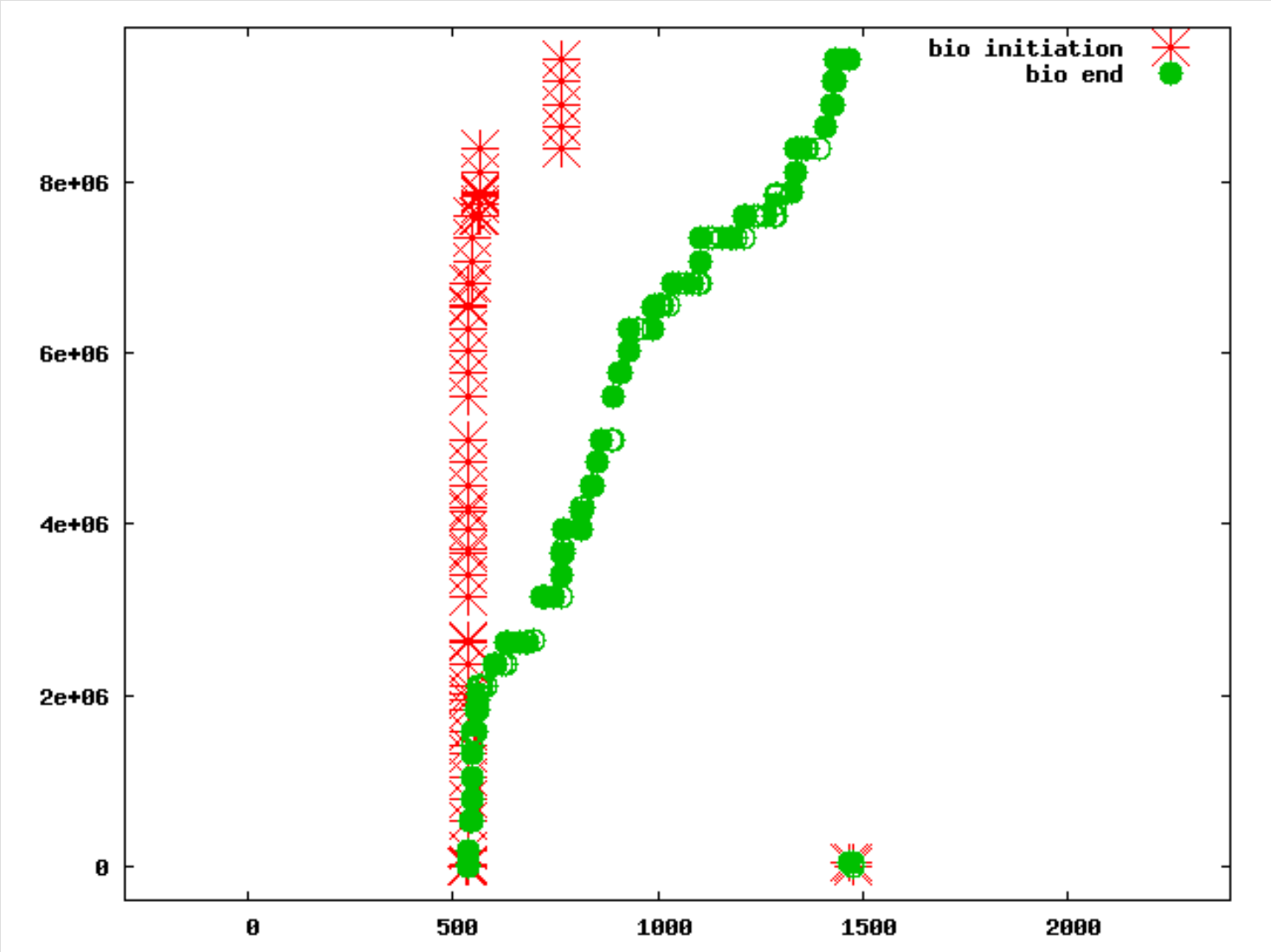


X-axis: ms
Y-axis: sector

Note the cluster of `fast bios' around 19400ms – the disk had them

Above is typical

`Storage is a lie' (Andre Hedrick)



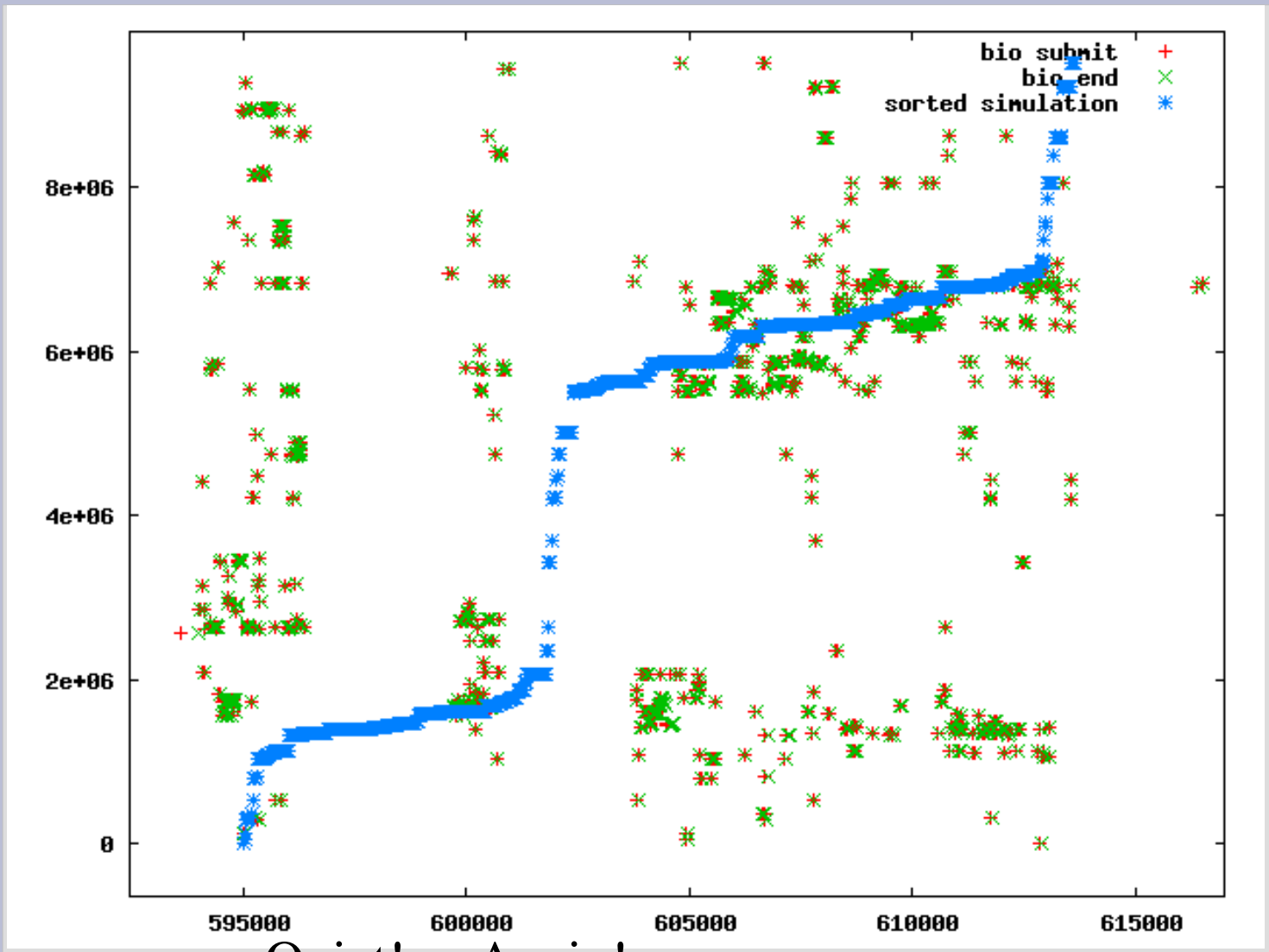
X-axis: ms
Y-axis: sectors

This depicts writes performed by the kernel itself – most likely ext3

Note how the initial writes are 'instantaneous'!

(is this bad?)

Mozilla startup + simulation



Quiet! Again!

x-axis: ms
y-axis: sectors

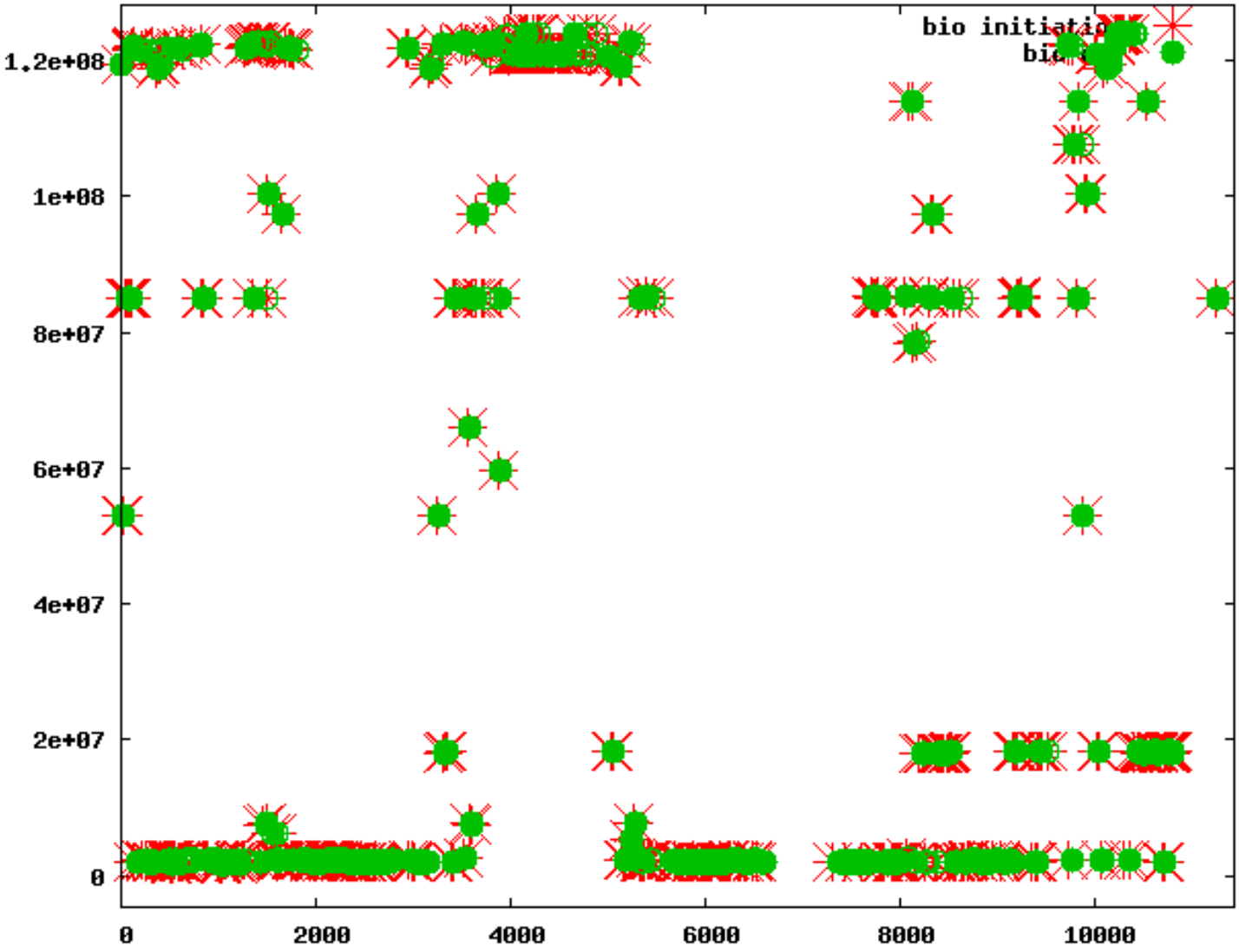
Mozilla startup on slow laptop:
20 seconds

The blue line is an artist's impression of how things could be, if requests were sorted.
Note empty areas!

More mozilla statistics

- Took 20 seconds, of which 5 were purely CPU-bound
- 942 different bios
- 19 megabytes (effective rate: 1MB/s)
- In 84 extents (defined as within 5 megabytes)
- 6 larger than 1MB, comprising 12MB
- Massive chances!

Openoffice: counter-example



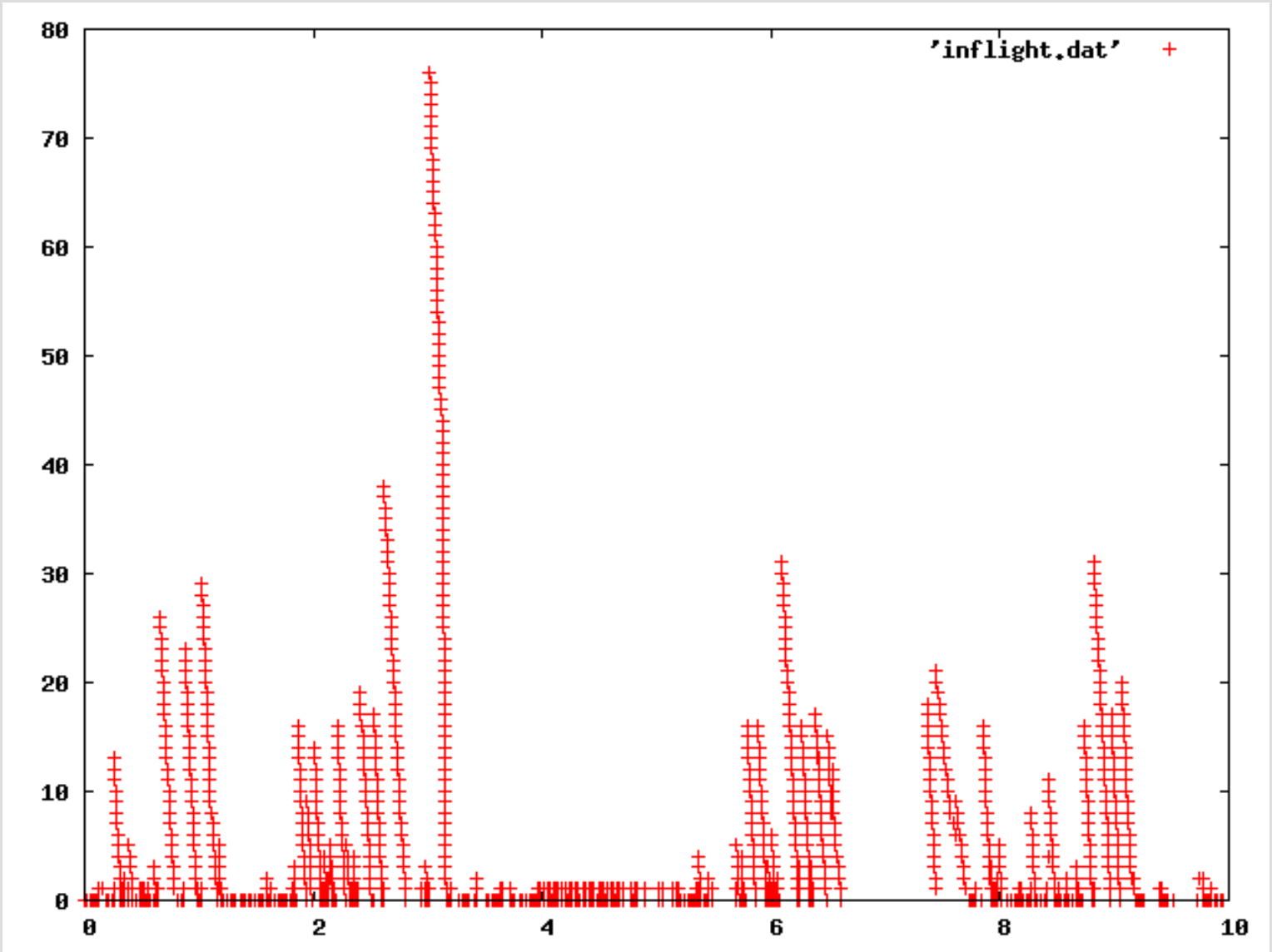
x-axis: ms
y-axis: sectors

Note high locality-of-reference

Second startup of OO is still slow.

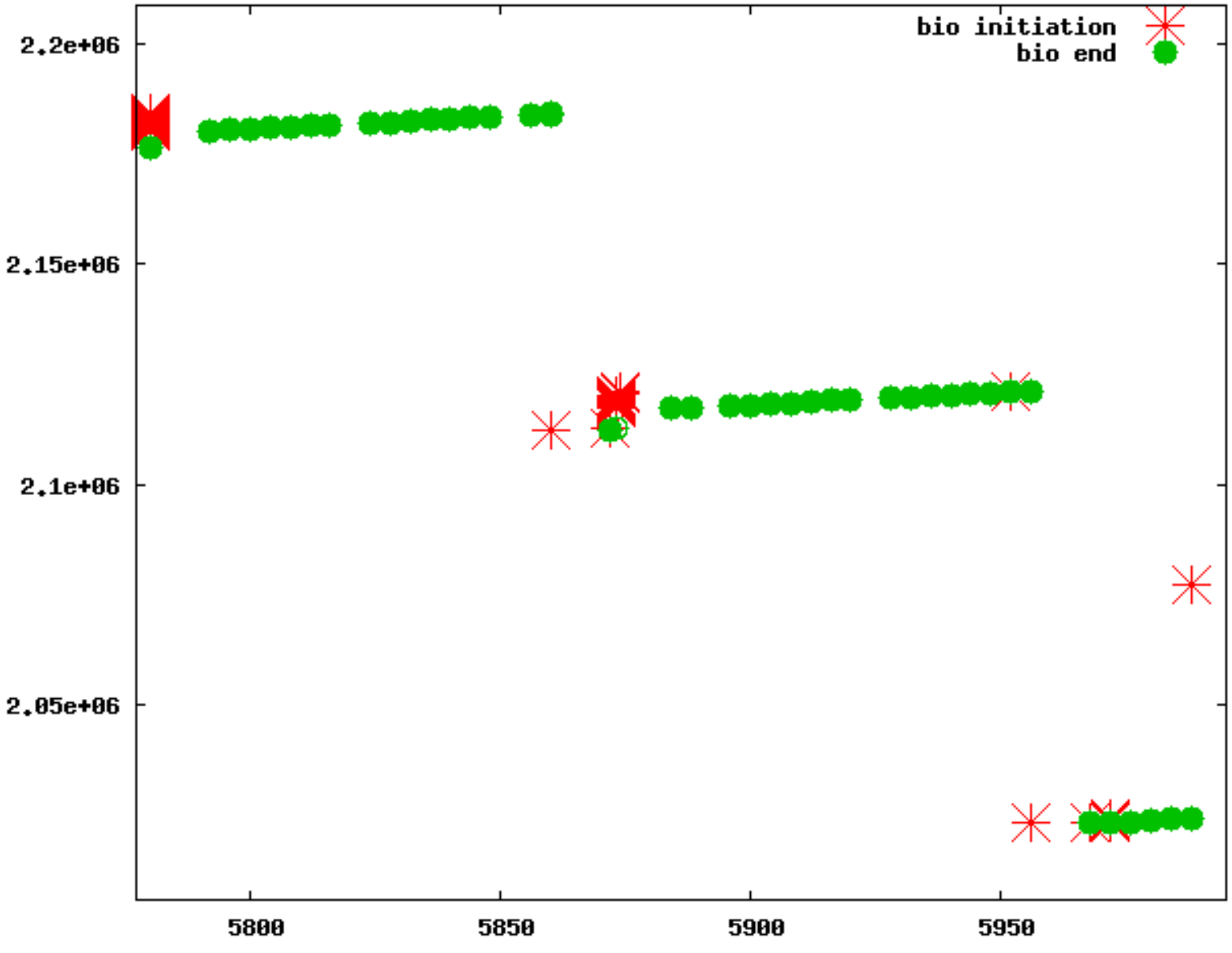
IO is only partly to blame here. However: stunning 105MB of reads!

Openoffice: requests in flight



x-axis: seconds
y-axis: number of bios in flight

Openoffice: moving backwards



x-axis: ms
y-axis: sectors
Highly zoomed, so the sectors are (somewhat) close together.

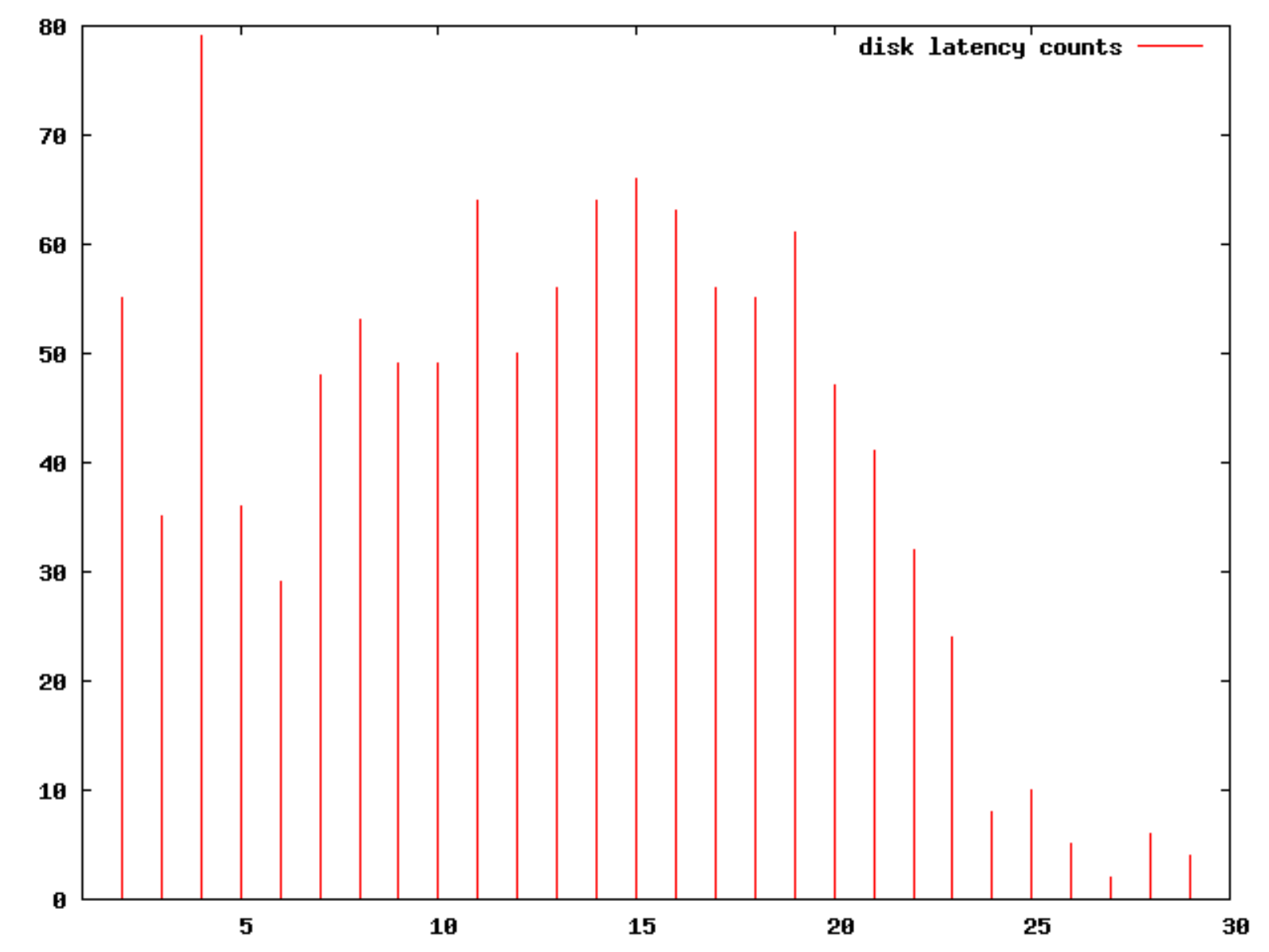
Note the backwards sense.

Note cache hits right below.

Typical bootup

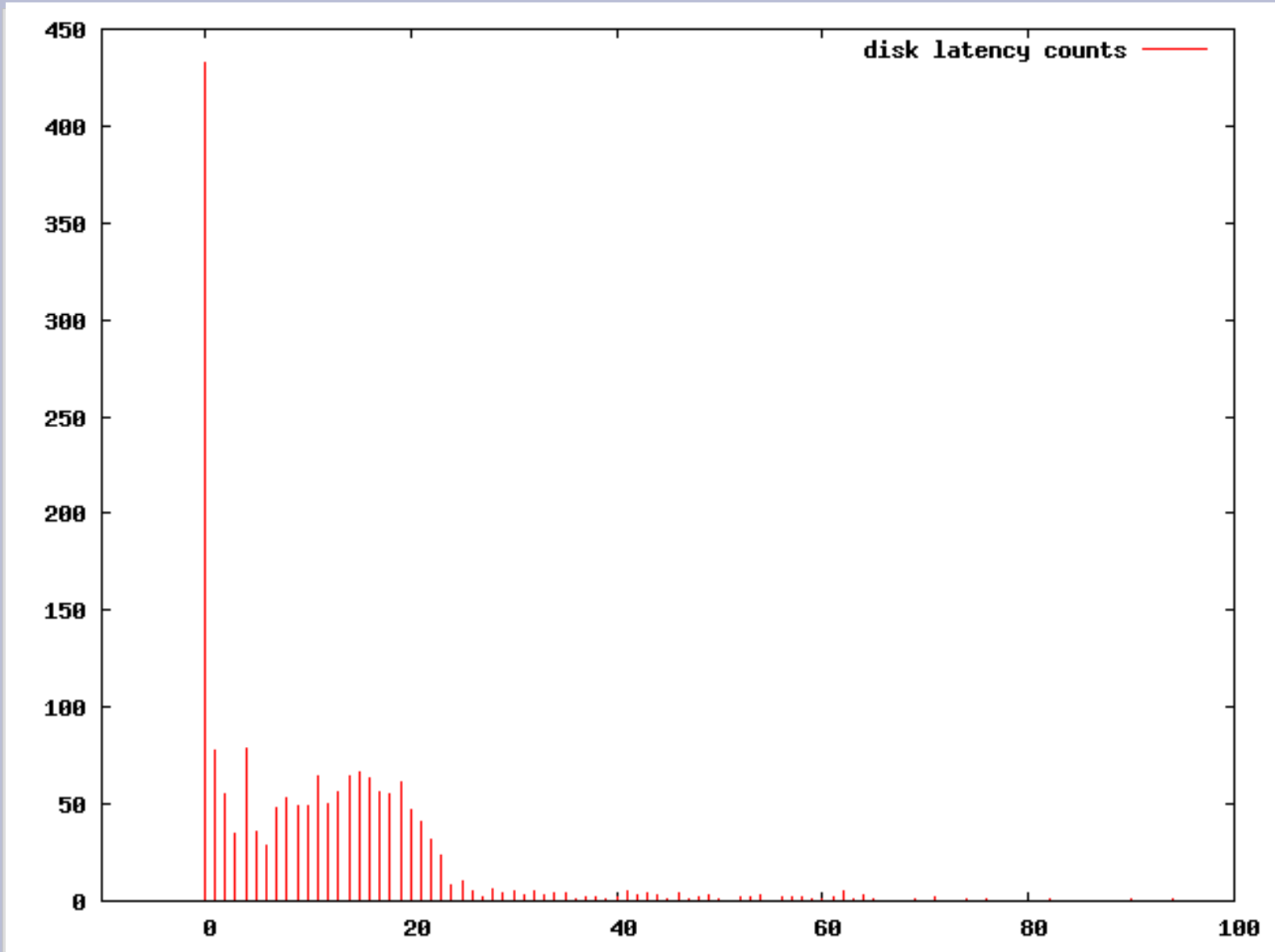
- Debian Woody, icewm desktop, startup including Mozilla: 50 megabytes, 30 excluding
- Ubuntu `Hoary', including Firefox: 150 megabytes
- Amazingly, both WRITE in excess of 10 megabytes during boot – atime?
- noatime shaves 10 seconds off boot time

Latency histogram



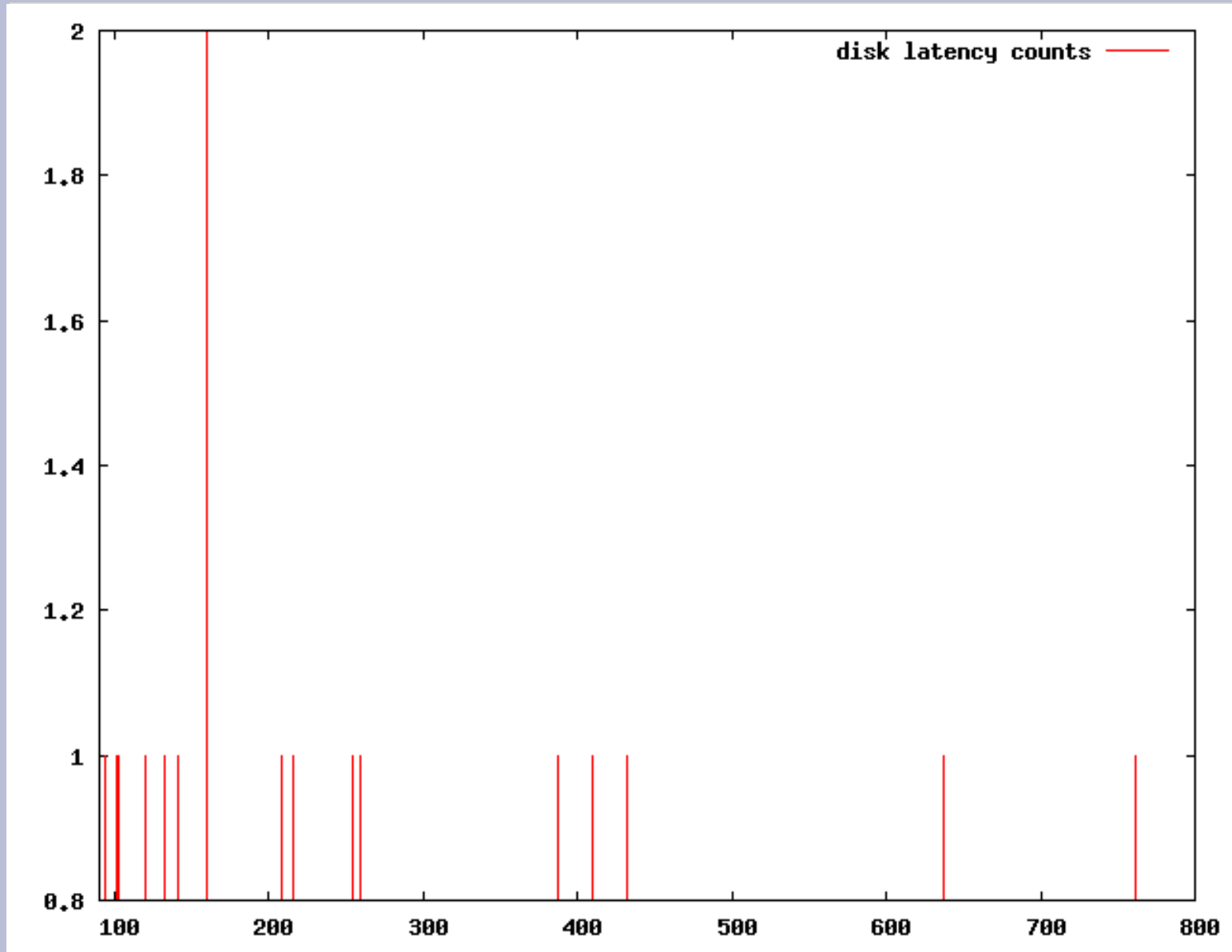
Lots of 0-ms hits elided
Pretty healthy graph

Latency histogram 2



0-ms ==
IDE disk cache
hit

Latency outliers



"Room for study"
Part of this
is disk-parking

Now what?

- Easy way (not that easy): figure out which sectors correspond to which files
- Coalesce requests based on statistics measured earlier about disk-cache behaviour
- Fire off big reads (linear: AIO only does O_DIRECT, no page cache!)
- 1) Fire up program 2) ?? .. 3)Profit!!

The bad news

- This works and generates rather impressive speedup to Firefox startup
- Bootup pretty slow though when we take priming time into account
- Turns out many bio-requests can't be traced back to files, because:
- Filesystem internals (dentries, block mappings) also cause reads

The good news!

- Several groups are working on this problem
- Given good measurements, solutions should be forthcoming
- There are some oddities that appear highly fixeable – sometimes Linux tries to read from disk **backwards**!

Some possible solutions 1

- The royal solution: stuff page cache with blocks and dentries – requires careful coordination though. Write out on shutdown.
- Unionfs a ramdisk over the / so a number of core files are in memory and read in one stretch
- Instrument exec calls and 'read-ahead' intelligently, based on bios seen
- Reorder binaries so they are read in consecutive order

Possible solutions 2

- If there is still such a thing as a buffer-cache, make submit_bio check it, and return immediately
- We can then just concentrate on touching the same sectors as we saw previously
- Does waste memory though

Toolset

- dumpstats: dumps everything
- dumpstats --bookmark: set bookmark
- dumpstats --since: dump since bookmark
- Available: RSN (end of this week)
- 40 line kernel patch + relayfs
- C++ stuff (does not burn the eyes)
- Gnuplot

Further information

- GPL tools will be available on http://ds9a.nl/diskstat/
- http://netherlabs.nl/
- bert.hubert@netherlabs.nl
- BoF Friday on Instrumenting the kernel
  - “Locating system problems with dynamic instrumentation” - Vara Prasad (IBM)
- I'll be around all week!