Benchmarks, Round 2

Felix von Leitner
felix-linuxkongress@fefe.de

September 2006

Round 2? What happened to round 1?

- Linux-Kongress talk about scalable network programming in 2003

- Got slashdotted

- Got flamed to hell and back

- Learned some interesting new insults

- Got some nice fan mail, too

- Got cited in some academic work!

Lessons learned from round 1

- Good email : bad email = 1 : 100

- People can't read (or maybe they chose not to)

- No matter how hard you try to be fair, people will flame you for being biased

- Even if you give them the source code, nobody will run benchmarks, but hundreds of Apple and Solaris fanboys will ask you to also benchmark OS X and Solaris for them

What to benchmark

- Interested in scalable web servers

- Want to reach hardware limits, if possible

- If not, want to know what's keeping me

Two Schools of Benchmarking

- Microbenchmarks ("peak MB/sec over one TCP connection")

- "Real World" benchmarks ("how many users can this SAP handle")

- Both have some merit

Microbenchmarks

- Easy to do

- Easy to screw up

- Tells you exactly what to fix

- Easy to get (intentionally) misleading results

- Not so easy to find right thing to benchmark

- "Oops, this only contributes 0.01%"

"Real World" Benchmarks

- "How many SAP users can this hardware run?"

- Results are often meaningless

- How to reduce variance?

  - "Default SAP+Oracle setup"
    Default setup works well on Solaris, sucks on Linux?
  - "Optimize for each one"
    Oh really? Microsoft optimized for Linux?
  - "Let each vendor optimize his platform"
    The customer won't have this expertise available

What I measured last time

- Scalability of in-kernel data structures for:

  – file descriptors
  – memory management
  – processes (forking and threads)

- Many microbenchmarks

- One "macrobenchmark": HTTP request latency

What I want to benchmark this time

- File systems

- The last major component missing

- Didn't do it last time because it's **hard** to do right!

- Performance depends on hardware

- ...and on position and size of file system on disk!

- What to do about tunables?

What results everyone (on Slashdot) really wants

- "Linux is the best operating system for web servers."

- "FreeBSD is the best BSD, and in many cases gives Linux a run for its money"

- "Windows stinks, IIS cheats, and Microsoft is teh suck!1!!"

- "OS X is much better than Windows!1!!"

  OK, time for some realism here.

What questions can we actually answer?

- Who has the fastest IP stack?

- Linux, BSD or Solaris?

- ext3 or reiserfs?

- Is reiser4/ZFS really that fast?

- Soft Updates or Journaling?

- AMD64 or x86?

But first, a little story...

- I once worked with a big German auction site

- Cluster of special web servers just for static images

- $>$ 2000 actual HTTP requests per second

- Linux 2.4 with reiserfs for storage, Apache

- Used opportunity to try fnord and gatling

- Fnord appeared to work better

- In strace, gatling blocked a lot on open(2)

So I wrote some code and prepared

- Sniffer to measure latency, dump URLs

- Tool to replay list of URLs as HTTP requests

- Took 23 GB backup tarball of their images (3M files)

- Took 300 MB "update" tarball, unsorted (67k files)

- Sniffed list of 100k HTTP requests

Our benchmark is coming together

1. mkfs

2. Unpack the big tarball (via http!)

3. Unpack the second tarball on top of the first one

4. start `gatling -n`, replay 10000 HTTP requests

5. start `gatling -n -N 32`, replay 10000 HTTP requests

6. rm -rf

Our benchmark is coming together

- take wall clock timing for each step

- make sure cache is flushed between steps

- for example: reboot, or umount and mount, or dd into a large file

  Last problem: where to get the pretty pictures from?

Pretty Pictures

- I hacked a per-protocol throughput sniffer

- Use it to sniff HTTP throughput

- Plot throughput per second

Contenders

- Linux (obviously)

- FreeBSD 6.1, NetBSD 3.0, OpenBSD 3.9 (release, not current)

- Dragonfly 1.6.0 (didn't work out)

- OpenSolaris (SchilliX 0.5.2)

- Windows (Server 2003, Vista)

- OS X (didn't work out)

Hardware - Test Server

- Dell workstation

- Pentium D, dual core, 3.2 GHz

- 2 GB RAM

- WD2500JS IDE disk, 232 GB

- Broadcom BCM5751 Gigabit Ethernet

Hardware - Second Test Server

- Nforce 4 chipset

- Athlon 64 X2 4600

- 2 GB RAM

- ST3400832A hard disk, 400 GB

- Intel 82541PI Gigabit Ethernet

Hardware - Test Client

- Acer notebook

- Pentium M, 1.8 GHz

- 1 GB RAM

- Seagate ST9160821A hard disk, 160 GB

- Broadcom BCM5705 Gigabit Ethernet

Problems

- OpenBSD hung on the Dell, both AMD64 and x86, both 3.9 and 3.8 and 3.7

- OpenBSD installed fine on my Athlon 64, but the Intel Ethernet driver failed

- Dragonfly detected the Ethernet, but failed to send or receive packets

- Solaris kernel-hung in sendfile in gatling, causing an unkillable process

- OS X didn't want to boot on my non-Apple hardware

The Questions

- **Who has the fastest IP stack?**

- Linux, BSD or Solaris?

- ext3 or reiserfs?

- Is reiser4/ZFS really that fast?

- Soft Updates or Journaling?

- AMD64 or x86?

Who has the fastest IP stack?

- Method 1: download the same file 50000 times

- Method 2: do the HTTP replay benchmark on a warm cache

Method 1

- 11k file, "server.exe"

- downloaded 50k times

- 50 concurrent connections

- with HTTP keep-alive, 10 requests per connection

- measure throughput

Method 2

- replay the 10k HTTP requests on a warm cache

- working set: double digit MB, plus huge directories

- 100 concurrent connections

- with HTTP keep-alive, 10 requests per connection

- measure requests per second

Results

OS	throughput MB/sec	rps
Linux 2.6.17	84 / 84	13.5k / 21k
FreeBSD 6.1	56.6 / 60.2	6.8k / 6.7k
NetBSD 3.0	57.3 / 57.1	6.7k / 5.9k
Solaris	77.8 / 80	6k / 9k
Win2k3	46 / 69	6.7k / 7.7k
Win2k3/IIS	83	161 (!)

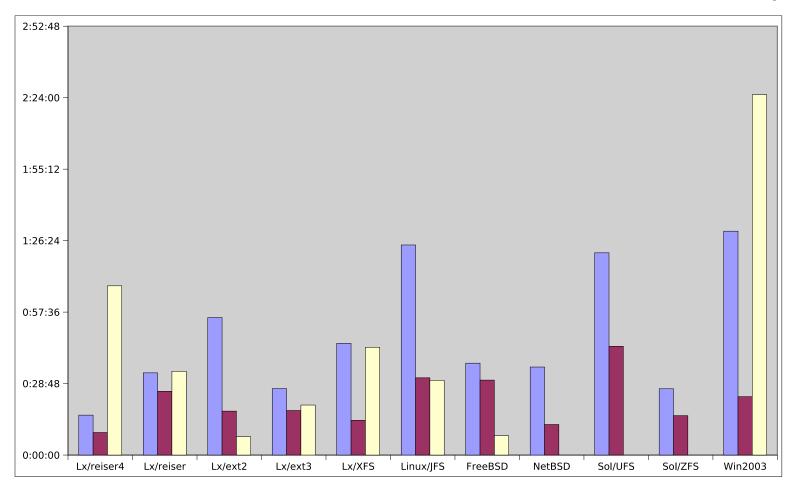First value: single threaded gatling; second value: multi-instance gatling.

Note: 84 MB/sec is the hardware limit the GigE NIC of my notebook can take. If someone wants to lend me 10 GigE hardware, please contact me.

The Questions

- Who has the fastest IP stack?

- **Linux, BSD or Solaris?**

- **ext3 or reiserfs?**

- **Is reiser4/ZFS really that fast?**

- Soft Updates or Journaling?

- AMD64 or x86?

Linux, BSD or Solaris?

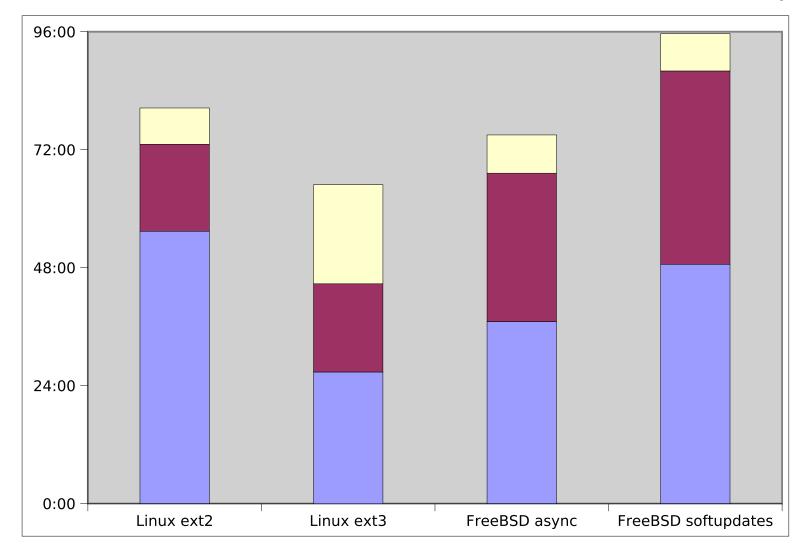| OS	tar1	tar2	http rps	rm -rf
Linux/reiser4	16:05	9:05	57 / 57	1:08:11
Linux/reiser	33:07	25:38	39 / 36	33:45
Linux/ext2	55:24	17:42	37 / 37	7:23
Linux/ext3	26:46	17:57	43 / 44	20:09
Linux/XFS	45:00	13:58	34 / 32	43:25
Linux/JFS	1:24:39	31:06	30 / 33	30:04
FreeBSD 6.1	37:01	30:12	46 / 43	7:47
NetBSD 3.0	35:28	12:19	41 / 41	n/a
Solaris/UFS	1:21:29	43:47	n/a	n/a
Solaris/ZFS	26:42	15:53	34 / 35	n/a
Win2k3	1:30:12	23:32	24 / 42	2:25:13

The Questions

- Who has the fastest IP stack?

- Linux, BSD or Solaris?

- ext3 or reiserfs?

- Is reiser4/ZFS really that fast?

- **Soft Updates or Journaling?**

- AMD64 or x86?

Soft Updates or Journaling?

OS	tar1	tar2	http rps	rm -rf
Linux/ext2	55:24	17:42	37 / 37	7:23
Linux/ext3	26:46	17:57	43 / 44	20:09
FreeBSD/async	37:01	30:12	46 / 43	7:47
FreeBSD/softupdates	48:41	39:20	46 / 43	7:35

The Questions

- Who has the fastest IP stack?

- Linux, BSD or Solaris?

- ext3 or reiserfs?

- Is reiser4/ZFS really that fast?

- Soft Updates or Journaling?

- **AMD64 or x86?**

AMD64 or x86

- 64-bit ports are just as well tuned as 32-bit ports

- Performance difference below 5%

- Probably more with more than 2 GB RAM?

Windows

- Porting gatling failed, porting fnord not fair

- Wrote web server, using AcceptEx, TransmitFile and I/O Completion Ports

- Supposedly fastest and most scalably way to do it on Windows

- On Vista, my server had 50% more throughput after I installed IIS on a different port (!?)

- Much less functionality than gatling, does less syscalls in the hot path, puts Windows at an unfair advantage

Windows

- `wget -O- | tar xzf` did not work.
  tar got an EOF, wget got a SIGPIPE.

- So I ported my tar to Windows, and added in-process gunzip (using zlib) and HTTP get

- This puts Windows at an unfair advantage

- Still Windows performed worst, in particular NTFS

- Also, IIS disables the buffer cache, but the replacement sucks

AcceptEx

- "High performance" hack for Winsock

- Can be told to return not after connect, but when data available

- And you give it the buffer, so you don't need an extra read

- What if someone connects, but does not send anything?

BTW: I used the native Win32 APIs using mingw32, not Cygwin! No emulation layer!

Lessons Learned

- Expected multi-process gatling to speed up cold cache case

- Turned out not to

- Turned out to speed up warm cache case instead!

- Only helps on Linux, Solaris and Windows, though

- Was shocked that ext3 is faster than ext2

- Also shocked that soft-updates cost 20% performance

Questions?

```
felix-linuxkongress@fefe.de
```