Engineered Systems with Linux

Wim Coekaerts
Oracle

Agenda

- System / Datacenter trends
- What are engineered systems?
- What did we learn using Linux on E.S.?

Mission

- Help/participate in making Linux the best kernel
  - You’re never really done
  - Better than it already is
  - Not just development, also testing/QA
- Help customers run their business efficiently
  - Understand the trends
  - Be able to remain current (and relevant)
    - New hardware support, larger systems

Trends and changes in datacenter deployments

- Virtualization changes OS deployments
  - Change from creating an OS VM to Virtual Appliances
  - Pre-packaged application VMs
  - Hosted applications
  - **OS hidden** (invisible not irrelevant)
  - Kernel important, OS distribution less important
  - Relatively small – medium sized VMs for now
  - Be a good guest kernel, be a good host kernel/ hypervisor

Trends and changes in datacenter deployments

- Large physical server deployments
  - Big boxes are back
  - More RAM, more cores, more everything
  - x86 is not just your 1 or 2 socket boxes any more
    - even 1 socket systems are more powerful
  - OS on larger systems needs to handle much more complex scenarios, workloads. Different algorithms.
  - Kernel is ‘in’ again. Scheduler, resource management, IO layer, network layer, NUMA support, irq balancing etc

Trends and changes in datacenter deployments

- Use your building blocks
  - Companies with engineers and admins wanting full control and literally build out everything themselves
  - OK if you have the knowledge and resources to do so
  - Decisions, decisions, decisions
  - Build your own plane
    - Which OS version
    - Which storage arrays
    - Which servers
    - Do they work well together for my workload?
    - Who tested this exact configuration?

Trends and changes in datacenter deployments

- Just buy the whole thing
  - Physical appliance or engineered system
  - Buy the plane
  - Hardware and software together
    - Tested together
    - Storage, network, OS, drivers, applications
    - Both small and large systems
    - *still* built using the same building blocks (doesn’t imply lock-in)

Engineered Systems

- We were looking at building the biggest, meanest, fastest database server on x86 out there
- Using Linux
  - Figure out how to get the absolute best performance
  - Distribution part is simple
    - We need like 150 or so RPMs
  - How to get the kernel to scale
    - This was a 1.5 year effort

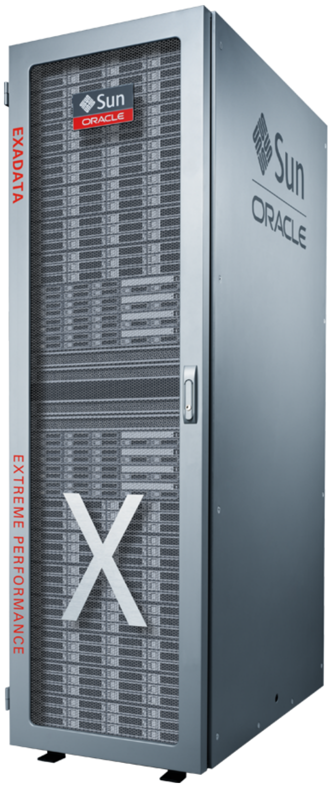
Example Systems

System Config 1

- 9 racks of hardware
- 2376 cores
- 1512 disks (2.3 PB raw storage)
- 48TB of Flash/SSD
- 6.9TB RAM

System Config 2

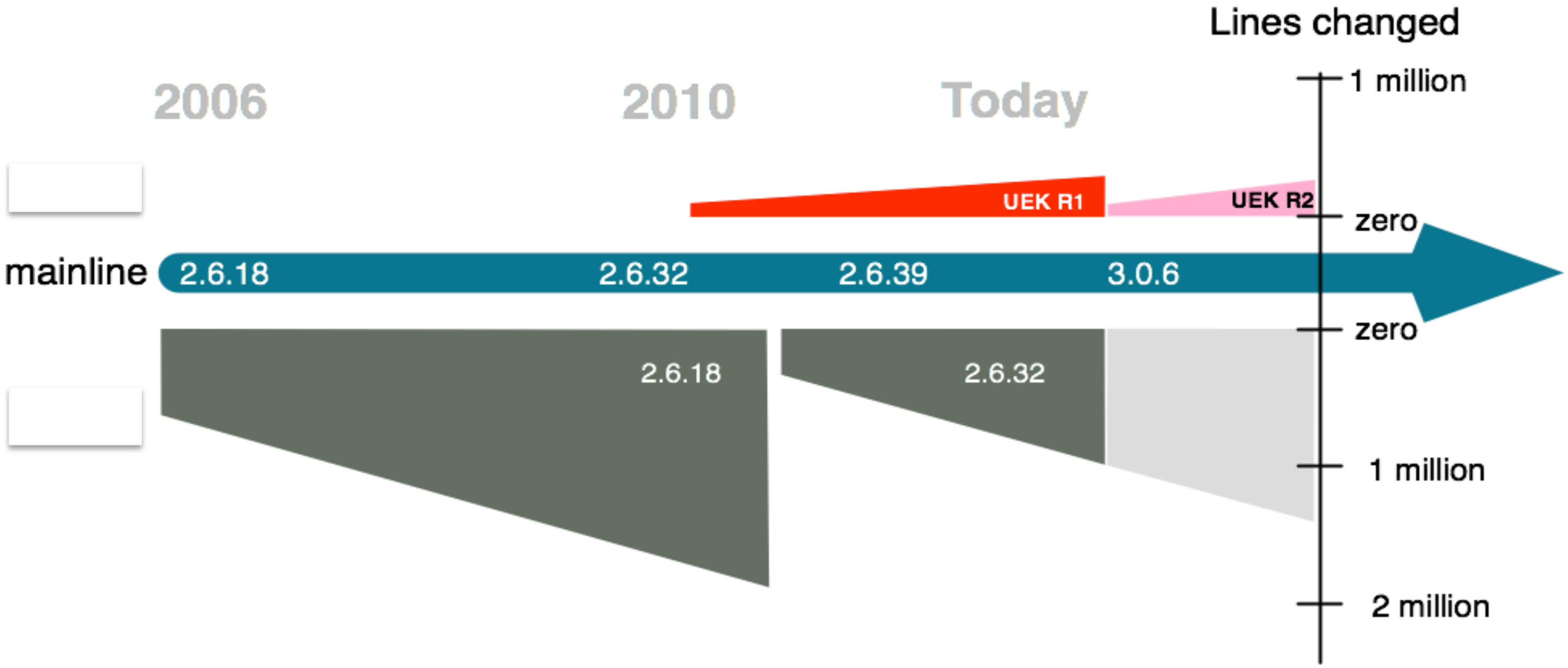
- 8 socket (160 threads)
- 4 TB ram
- 16000 luns



Lessons learned

- Effort started with a 2.6.18 based tree but…
- Current hardware really needs a current kernel
- Backporting patches has a ton of drawbacks
  - Re-test code as if it was new
  - Some one already wrote it, and you have to rewrite it yet again
  - You’re moving things written against, sometimes significantly different code
  - Can’t backport big architecture changes
- So we moved to using a stable mainline kernel 2.6.32 first, now we’ve moved on to, really, 3.0.4

Lessons learned



Lessons learned

- Power management support
- CPU features (GB hugepages, hardware CRC. Etc)
- Interrupt infrastructure
- MCElog, hardware memory poisoning
- Network + IO subsystems have gotten a lot better
- Multi-queue IO support
- Perf code helped a lot to tune
- Using GIT and mainline trees makes it so much easier to hunt down bugs, fix regressions and keep a solid history.

Lessons learned

- We were able to do months of testing with full system configurations under 100% load
- 80,000+ hours of QA/day
- Find regressions and bugs, fix them, submit them
- Features to help scaling like lockless wake-up path, IPC semaphore changes, IO affinity latency changes

Lessons learned

- 5 years ago gregkh said “distributions should be using mainline stable trees”… he was right ☺
- Using a stable mainline tree and update more regularly instead of backporting helps everyone
    - Helps us be current
    - Reduces somewhat wasted backporting efforts
    - Any bugs we find and fix are relevant to everyone
    - Helps us find mainline regressions now, not 3 years from now
- Just publish a public git repo with our kernel source and anyone can dig into it and figure out what changed, why, and immediately pull in from mainline to move forward