Rethinking data centers: Power, packaging, and networks

"There is nothing more difficult to take in hand, more perilous to conduct, or more uncertain in its success, than to take the lead in the introduction of a new order of things" – Niccolo Machiavelli

Chuck Thacker

Microsoft Research

June, 2009

Problems

- Today, we need a new order of things.
- Problem: Today, data centers aren't designed as *systems*. We need apply *system engineering* to:
  - Packaging
  - Power distribution
  - Cooling
  - Computers themselves
  - Networking
- All of these can be improved to improve both capital and operating cost.

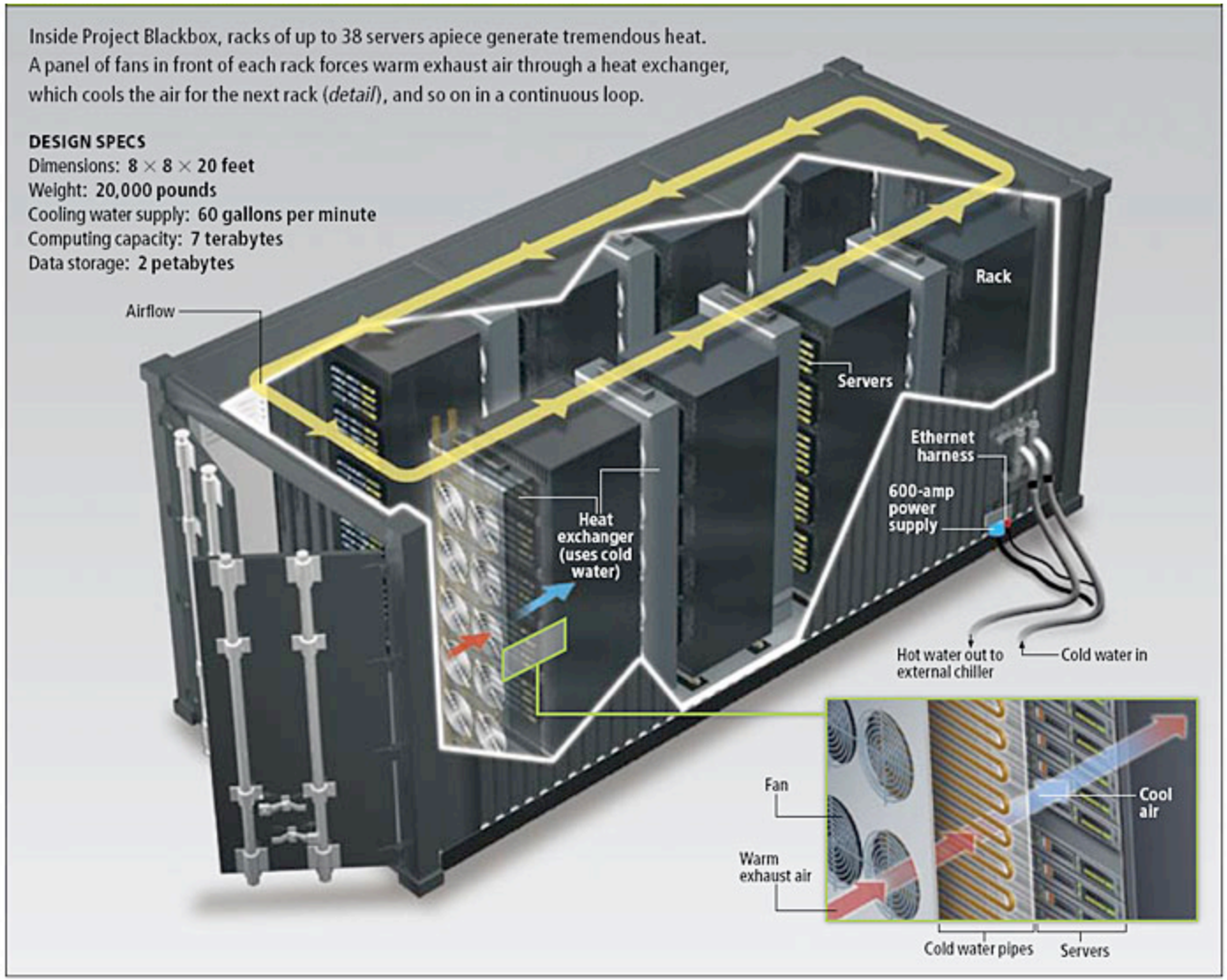
System design: Packaging

- We build large buildings, load them up with lots of expensive power and cooling, and only then start installing computers.
- Pretty much need to design for the peak load (power distribution, cooling, and networking), even though much of this is unused initially.
- We build them in remote areas.
  - Near hydro dams, but not near construction workers.
- They must be human-friendly, since we have to tinker with them a lot.

Packaging: Another way

- Use a shipping container – and build a parking lot instead of a building.
- Doesn't need to be human-friendly.
  - Might never open it.
- Assembled at one location, computers and all. A global shipping infrastructure already exists.
- Sun's version uses a 20-foot box. 40 would be better.
- Requires only networking, power, and cooled water.
- Expands as needed, in sensible increments.
- Rackable has a similar system. So does Google.

The Black Box



Container Advantages

- Side-to-side airflow is not impeded by the server case. There is no case.
  - With bottom-to-top, servers at the top are hotter.
  - With front-to-back, must provide hot and cold plenums.
- The server packaging is simplified, since they are not shipped separately. Can incorporate shock mounting at the server, not the rack level.
- Cables exit at the front, simplifying assembly and service.
- Most of this also applies to conventional data centers.

Packaging

- A 40’ container holds two rows of 16 racks.
- Each rack holds 40 “1U” servers, plus network switch. Total container: 1280 servers.
- If each server draws 200W, the rack is 8KW, the container is 256 KW.
- A 64-container data center is 16 Mw, plus cooling. Contains 82K computers.
- Each container has independent fire suppression. Reduces insurance cost.

Cooling

- Once-through air cooling is possible in some locations.
  - Unfortunately, data centers tend to be built in inhospitable places.
  - Air must be filtered.
  - Designs are not compatible with side-to-side airflow.
- Cooling towers are well understood technology.
  - And need not be used all the time.
- Once-through water cooling is attractive.
  - Pump water from a river, use it once, sell the output to farms.

System Design: Power distribution

- Need to minimize conversion steps to minimize losses.
- Power supplies aren't very efficient:
  - 12VDC -> 1VDC point-of-load regulators are ~90%.
  - AC -> 12VDC converters are now 2-stage (power factor correction, inverter). 85% efficient at full load, lower at low load. Can do better.
- AC transformers are 98% efficient. Two steps needed.
- Final efficiency, grid to chips/disks: ~80%.
- UP and backup generators aren't part of the picture until the grid fails.

Power Distribution

- Deliver 3-phase AC to the rack
  - Must balance the phases anyway
  - Lower ripple after rectification
- What voltage?
  - TBD, but probably 12-20 VAC.
  - Select to maximize overall efficiency

The Computers

- We currently use commodity servers designed by HP, Rackable, others.
- Higher quality and reliability than the average PC, but they still operate in the PC ecosystem.
  - IBM doesn't.
- Why not roll our own?

Designing our own

- Minimize SKUs
  - One for computing. Lots of CPU, Lots of memory, relatively few disks.
  - One for storage. Modest CPU, memory, lots of disks.
  - Maybe Flash memory has a home here.
- What do they look like?

More on Computers

- Use custom motherboards. We design them with ODM partners, optimizing for *our* needs, not the needs of disparate applications.
- Use commodity disks.
- All cabling exits at the front of the rack.
  - So that it's easy to extract the server from the rack.
- Redesign the power supply.
- Error correct where possible, and check what we can't correct.
  - The worst error is the undetected error.
- When a component doesn't work to its spec, fix it or select another, rather than just turning the "feature" off.
- Use lower-power processors (notebook, rather than server). Choice is dictated by workload data.

Networking

- A large part of the total cost.
- Large routers/switches are *very* expensive, and command astronomical margins.
- They are relatively unreliable – we sometimes see correlated failures in replicated units.
- Router software is large, old, and incomprehensible – frequently the cause of problems.
- Serviced by the manufacturer, and they're never on site.
- By designing our own, we save money *and* improve reliability.
- We also can get exactly what we want, rather than paying for a lot of features we don't need (data centers aren't mini-Internets).

Data center network differences

- We know (and define) the topology.
- The number of nodes is limited.
- Broadcast/multicast is *not* required.
- Security is simpler.
  - No malicious attacks, just buggy software and misconfiguration.
- We can load balance requests to minimize hot spots.
- Within the center, we don't need IP.
  - Do need it at the edge.

Data center network design goals

- Reduce the need for large switches in the core.
- Simplify the software. Push complexity to the edge of the network.
- Improve reliability
- Reduce capital and operating cost.

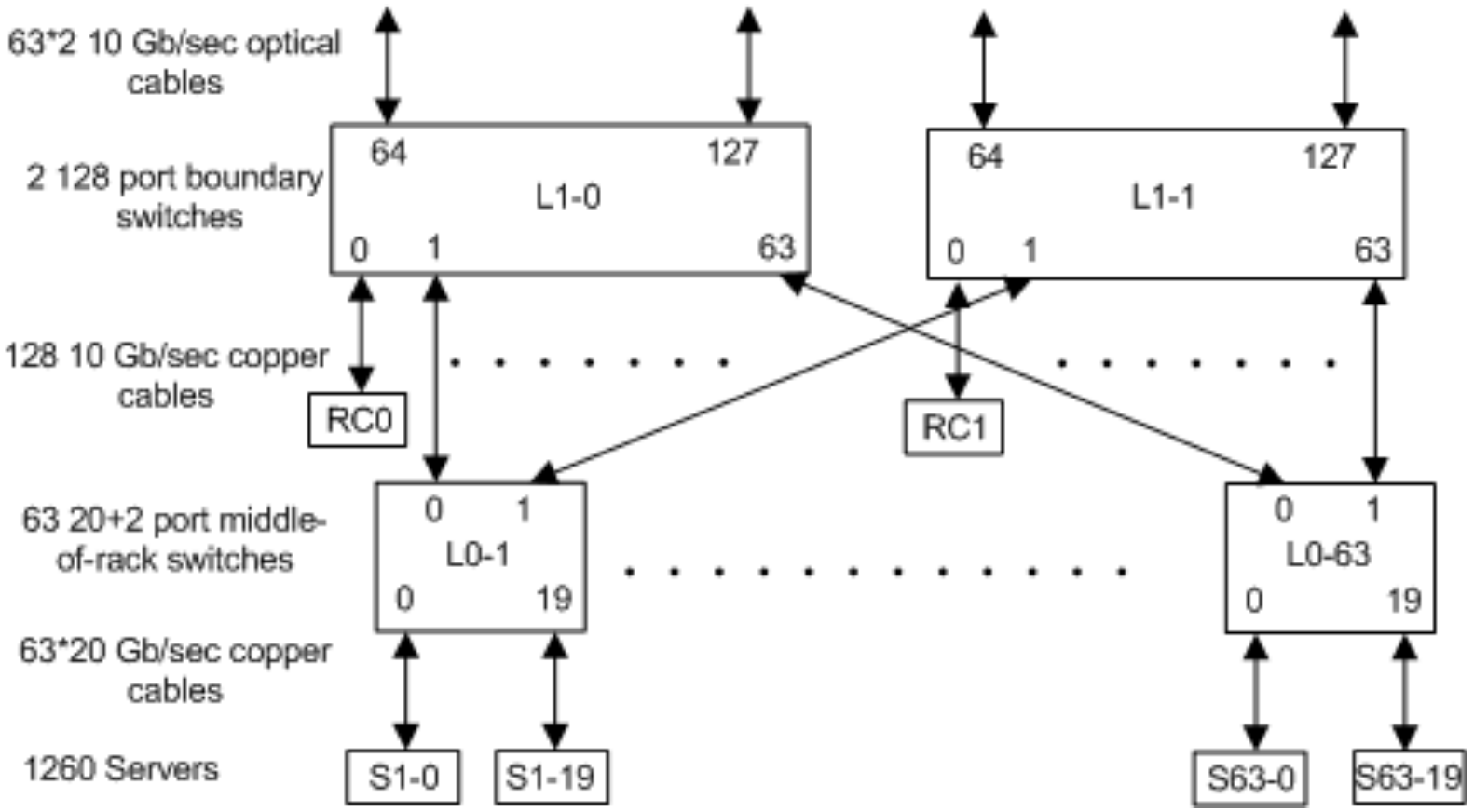
An approach to data center networks

- Use high-radix switches in the core.
- Use source routing.
  - Low hop count helps this.
- Use standard link technology, but don't need standard protocols.
  - Mix of optical and copper cables. Short runs are copper, long runs are optical.

Basic switches

- Middle-of-rack switches (2 per rack, 4096 total)
  - 20 1+1 Gb ports to servers.
  - 2 10 Gb (copper) links to container boundary switches
  - Traffic within a half-rack never leaves these switches.
- Boundary switches (2 per container, 128 total)
  - 128 10 Gb ports.
  - 64 (copper) ports to middle-of-rack switches
  - 64 (optical) links to other containers and the NOC.
- Somewhat surprisingly, both types can be implemented with Xilinx Virtex 5 FPGAs.
  - These contain 10 Gb Phys, plus buffer memories and logic.
  - MOR switch also uses Gb Ethernet Phy chips
- We can prototype both types with BEE3.
  - Real switches use less expensive FPGAs

Inside each container:



Network details

- Boundary switches operate synchronously, transferring 2 KB *frames*.
- Faster links (40 Gb/s) are driving this.
  - There simply isn't time enough to switch each packet.
- The synchronous scheduling significantly reduces the buffer memory needed at each port.
  - Only 2 frame buffers needed per port.
- Network is source routed.
- Route controller associated with each boundary switch assigns routes for *flows*, doing setup and teardown as needed.
- For traffic between containers, route controllers cooperate to define routes.

Bandwidth

- Container bandwidth: 640 Gb/sec.
- Data center bisection bandwidth: 20.5 Tb /sec.
- Uses VLB to provide more than 20 Gb /sec. between two containers.
  - In this case, 3 route controllers are involved in path setup.

Objections

- "Commodity hardware is cheaper"
  - This *is* commodity hardware. Even for one center (and we build many).
  - And in the case of the network, it's *not* cheaper. Large switches command very large margins.
- "Standards are better"
  - Yes, but only if they do what you need, at acceptable cost.
- "It requires too many different skills"
  - Not as many as you might think.
  - And we would work with engineering/manufacturing partners who would be the ultimate manufacturers. This model has worked before.
- "If this stuff is so great, why aren't others doing it"?
  - They are.

Conclusions

- By treating data centers as *systems*, and doing full-system optimization, we can achieve:
  - Lower cost, both opex and capex.
  - Higher reliability.
  - Incremental scale-out.
  - More rapid innovation as technology improves.

Questions?