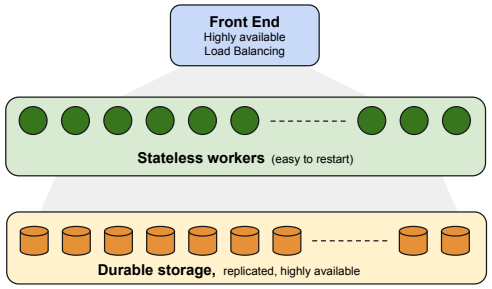
Parallelism in the Cloud

Eric Brewer
UC Berkeley & Google

HotPar Keynote, June 24, 2013

Giant-scale Services

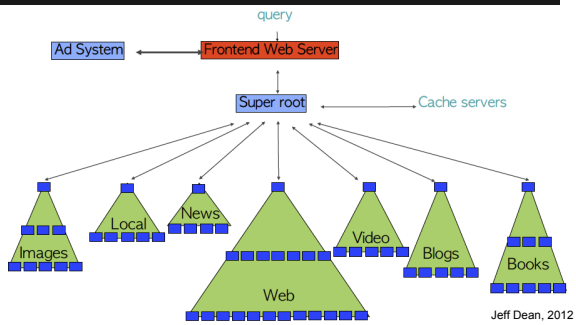


Latency matters (a lot)

Various claims:

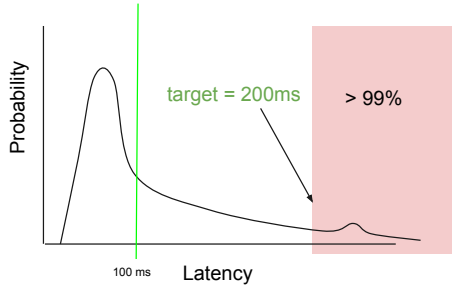
- Google: 0.5 second => -20% page views
- Amazon: extra 100ms => -1% revenue
- Aberdeen Group: extra second =>
  - -11% page views
  - -7% conversion rate
  - -16% customer satisfaction

Reduce latency via parallelism, caching



Jeff Dean, 2012

Tail Latency



Our tricks hurt tail latency

Caching
- Prediction in general

Parallelism
- Limited by slowest replies

Virtualization
- Extra scheduling, memory pressure
- Worse if actual cores < expected cores
- ***Virtualization is a lie revealed by tail latency***

Logs
- Faster writes, but occasional compactions

Parallelism & Tail Latency

	50%ile latency	95%ile latency	99%ile latency
One random leaf finishes	1ms	5ms	10ms
95% of all leaf requests finish	12ms	32ms	70ms
100% of all leaf requests finish	40ms	87ms	140ms

Dean and Barroso, CACM February 2013

Strawman

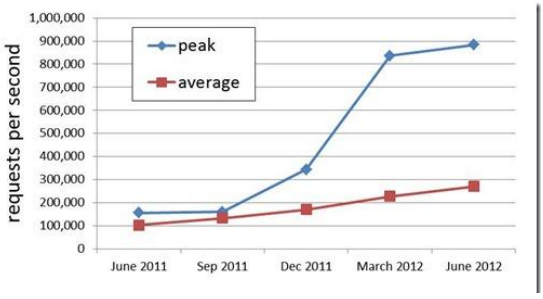
Allocate dedicated resources to live services

- No other jobs on those servers
  - No scheduling
  - Virtual machines don't hurt much
- No page faults

Also, just to be sure:

- no power management
- no background tasks
- rare upgrades or failures

Windows Azure: Peak vs. Average



Windows Azure Blog, July 18, 2012

Batch Computing

Huge cluster to handle peak loads

- But huge waste off peak...

Batch computing is "free" *as it fills in the gaps*

- Led to MapReduce, Hadoop,
  - Also led to *Big Data?*
- Enables *extensive* precomputation
  - Google maps, book scanning, web indexing, ...
- Also much easier
  - Easy to retry failures
  - Low stress

Amazon Spot Instances

"spot market" for unused servers

- price increases with demand

Price, June 2013, Linux "medium instance"

Instance	Cost per Hour	Ratio
Spot	**1.3 cents**	-
On Demand	**12.0 cents**	10x
Reserved (1 year)	**6.8 cents**	5x

Needs of the Cloud

Live service jobs:

- Minimize latency, including tail latency
- Minimize layering, virtualization
- Predictable efficient performance

Batch jobs:

- Lower priority
- Should fill in the peak/average gap
- Delays tolerated

Akaros

A new research OS made for the cloud:

- Single-node OS
  - Scheduling decisions made elsewhere!
  - No user interface, limited devices
- Mix of low latency and batch workload
- Transparent not virtual resources

Open Source: http://akaros.cs.berkeley.edu

Barret Rhoden, Kevin Klues, David Zhu

Provisioning vs. Allocation

Provisioning

- Guaranteed future access to resources
- Used for low-latency services
  - Estimated based on *peak* load
  - Allocated a subset at any time

Allocation

- Real-time resources being used (active load)
- With provisioning: uninterruptable, irrevocable
- Without provisioning: can be revoked at any time
  - Used for batch jobs
  - **Revocation time is 2-3 microseconds**

Many Cores

Moving to 100 cores per server:

- Provision them to services (space partitioning)
  - Also partition memory
  - Ideally divide bandwidth as well (not done yet)
- Three-level scheduling:
  - Cluster OS decides on provisions, batch work
  - Node OS allocates cores
  - User level: service schedules threads on cores
- Akaros view:
  - Partition the cores/memory (like exokernel)
  - *Revoke* cores to return to provisioned service
  - Minimize interference (from other jobs, interrupts, etc.)

Many-core Process (MCP)

Manages k cores as one process:

- Single address space
- User-level maps threads onto k cores
  - Knows what k is!
  - **Cores for parallelism, threads for blocking I/O**
    - Thread blocking does not lose core
    - Similar to scheduler activations
  - Notified of change in number of cores
  - (if timeslicing) cores are *gang scheduled* -- enables efficient spinlocks

MCP Implementation

Process has a vcoremap:

- Maps virtual cores (vcores) onto physical cores
  - vcores are 1:1 and pinned
  - But we can move them around as needed
  - And can revoke any core as needed
- Better predictability
  - Long quanta (competing jobs are batch only!)
  - Limited interrupts => less interference
  - Careful memory partitioning
  - No page faults for low-latency services

User-level scheduling

Services see a "dedicated" SMP

All syscalls are asynchronous:

- User-level thread blocks
- Core reused by user-level scheduler
- Syscall completions returned on event queue

Similar to many-core version of Capriccio

- User-level threads atop event-based kernel
- Context switch ~3x faster than Linux

Can also support pthreads, TBB, Go threads

Virtual Machines

View 1: "needed less over time"

- VMs reduce predictability, efficiency
- Akaros really aims for bare metal resources
  - e.g. TritonSort leveraged *knowledge of actual hardware* to set sorting records
- Containers address *some* of the VM gains
  - Bundling and isolation
  - New support for migration (CRIU)
  - ... but still close to bare metal

Virtual Machines (2)

View 2: "... but VMs still useful"

- Great for legacy code
- Server consolidation
- Untrusted code

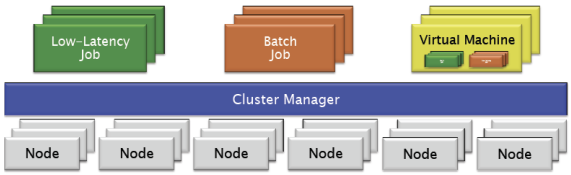
Solution: VM on top of an MCP

- MCP provides raw cores/memory
- Should make VMs more predictable
  - Stable resources, less interference
- Can run side-by-side with non-VM MCP

Mixing them altogether...

Cluster manager schedules MCPs on nodes

- Services, batch workers, VMs collocated
- All three use user-level schedulers for threads



Akaros Status

32-bit version working for C

- Can mix services and batch work well
- User-level scheduling, async syscalls work
- Network stack partially done
- Go port in progress

Still needs tons of work:

- 64-bit version in progress
- KVM-style VM solution on MCP
- Integration with cluster scheduler (Mesos?)

Summary

Cloud has different OS needs:

1. Predictable low-latency services
2. Batch work to fill in gaps left by peak allocation
3. Node OS is remotely controlled platform

Akaros is Berkeley's take on this space

- Bare metal many-core processes
  - Threads != Cores
- Spatial partitioning
- User-level scheduling (keep your cores)

Backup

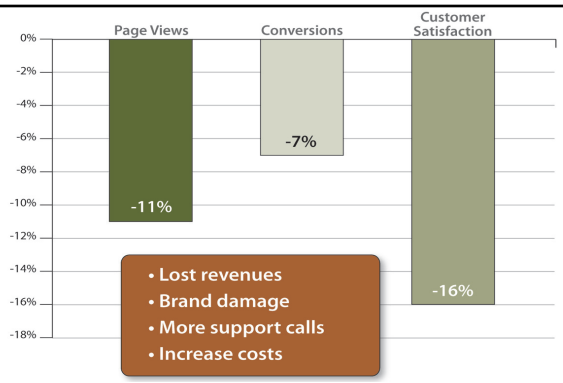
Abstract

Parallelism in the cloud comes in two basic flavors: low-latency queries and batch processing. These flavors are both fundamental but quite different in their needs. By separating the two we can build simpler systems that support both well. In particular, we cover the design of a new OS for cloud computing that brings these concepts all the way down to the metal, where the job shifts from virtualizing resources for typically local human users to more direct support for these two classes of parallelism under the remote control of a cluster-wide scheduler.

Macro Computing

When clusters are big enough:

- law of large numbers wins...
  - all bad things happen
- can't track individual servers, files, etc.
- must think probabilistically
- must think about balance of resources



Cost of an extra Second

Aberdeen Group, 2006