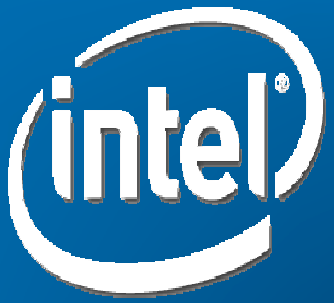


Optimizing a Cloud with SLAs and QoS

April 5, 2012

Naresh K. Sehgal, Ph.D.

Software Architecture Manager

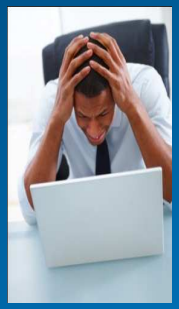
Data Center Solutions Group, Intel Corp.

Contributors: Mrittika Ganguli, Alok Prakash, Jaideep Moses, John Leung, Doug Mason

Pain Points in IaaS Cloud

Cloud User

Requests services from cloud account



Load balancer

L M S

Virtual Infrastructure Services template

Cloud Infrastructure & Operations

Cloud Portal

Cloud OS

Cloud Ops

Instantiate VM

Fetch Image

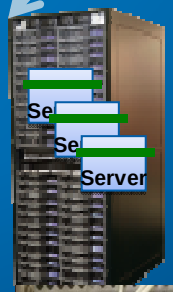
Connect image to VM

Server Server Server

Scheduling DB

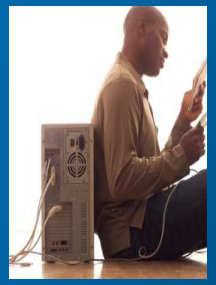
Logical Resource Pool Created from physical infrastructure

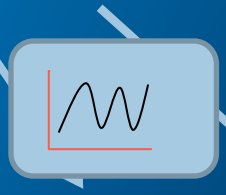
Machine Images



Service Provider

Monitor & manage cloud QoS





Monitor Infrastructure performance

Plan infrastructure, capacity, capability, consumption, compliance

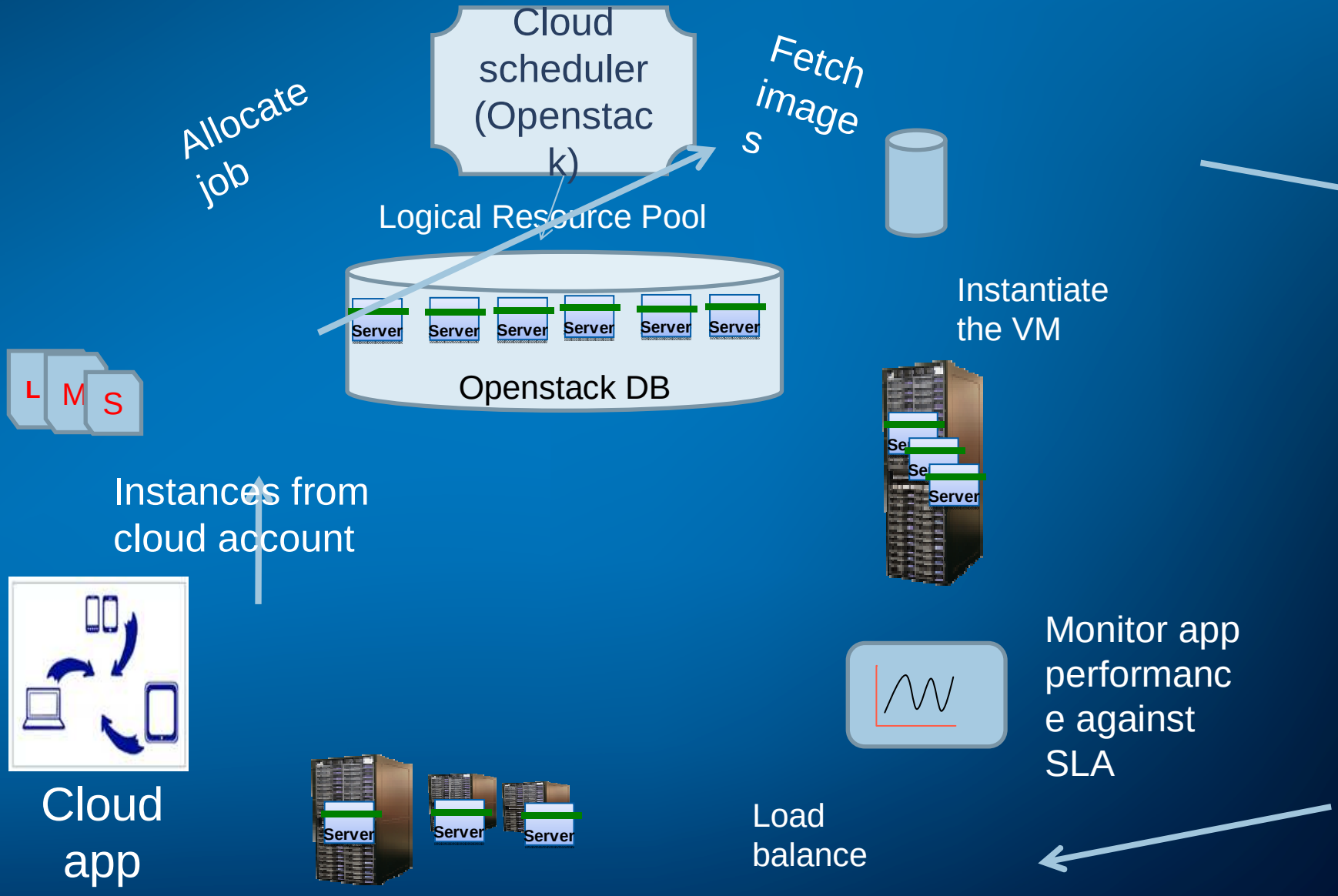
"Match workload to platform c

"Automated, secure, efficient infrastructure & Operations"

"Noisy neighbor VMs"
"New Services"
"New Service Levels"

Cloud OS can benefit from fine-grained platform resource monitoring and controls to assure *predictable performance, efficient and secure operations*

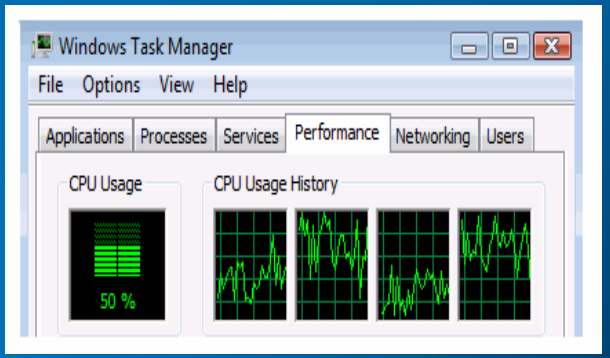
Cloud scheduling environment



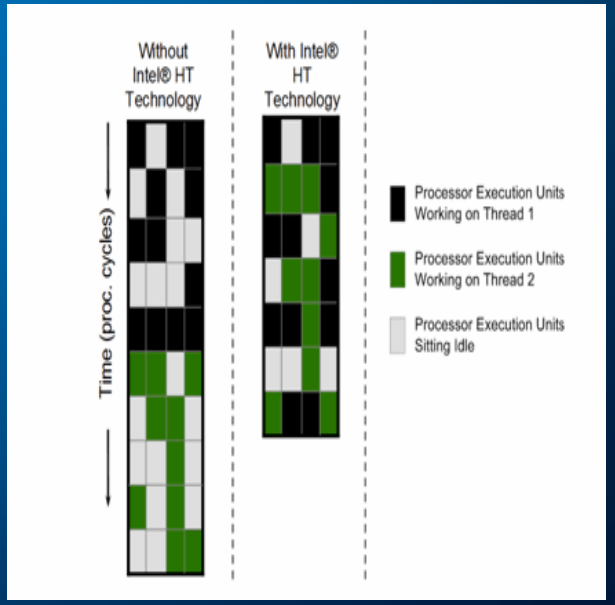
Scheduling not efficient as it is not based on hardware capability and workload characteristics.

Problem Statement

- Need to ensure a consistent User experience
  - By not over-saturating a platform's resources
- "CPU Utilization" measures usage during the time a thread is scheduled on a core:
  - Good enough for compute bound workloads on a core
  - But unreliable[1] for a multi-core system with multi-level caches, non-uniform memory, Simultaneous Multithreading (SMT), pipelining, Out-of-order execution etc., muddies thread mapping to cores:
    - In fig 1, 2nd and 4th HW threads are utilized the most, but are these sibling threads or separate cores?



Four threads with two cores and Intel HT Technology enabled



With 4-wide execution units on a core, by giving access to two threads in the same time slice, Intel® HT Technology reduces idle hardware resources, increases efficiency and throughput.

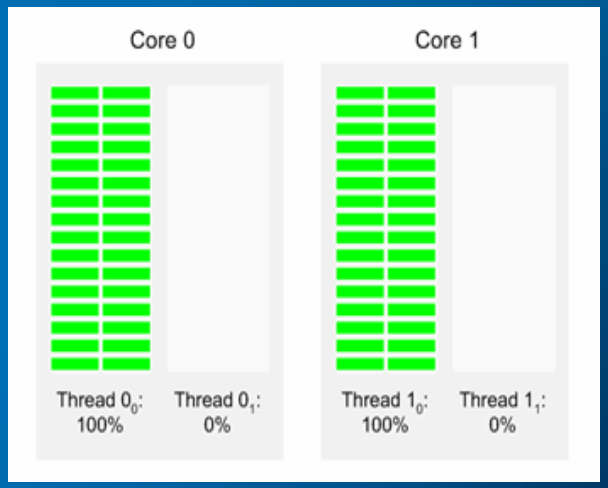
[1] http://software.intel.com/en-us/articles/performance-insights-to-intel-hyper-threading-technology/

CPU Utilization doesn't give the whole picture

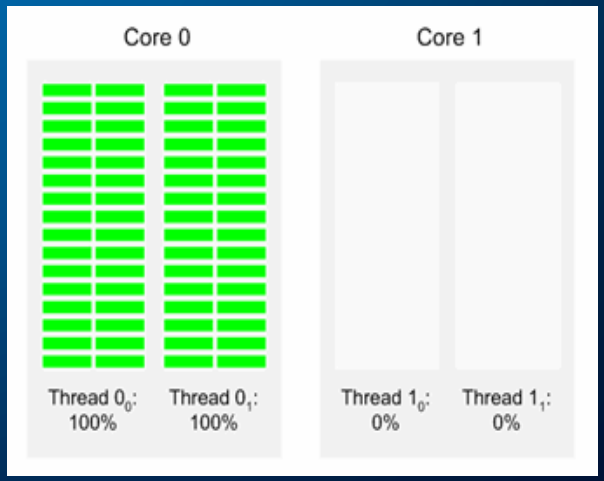
- Assume that 1 thread on a core does 1 unit of work/sec, and Intel® HT Technology gives 1.25x performance gain with 2 threads on a core

- Total CPU Utilization calculated by perfmon and SAR is 50% in both cases = (100+0+100+0)/4
  - But Different amounts of work gets done
  - 2X in the first case, vs. 1.25X in the second

- CPU utilization alone is not a good measure of work done or headroom to place new tasks



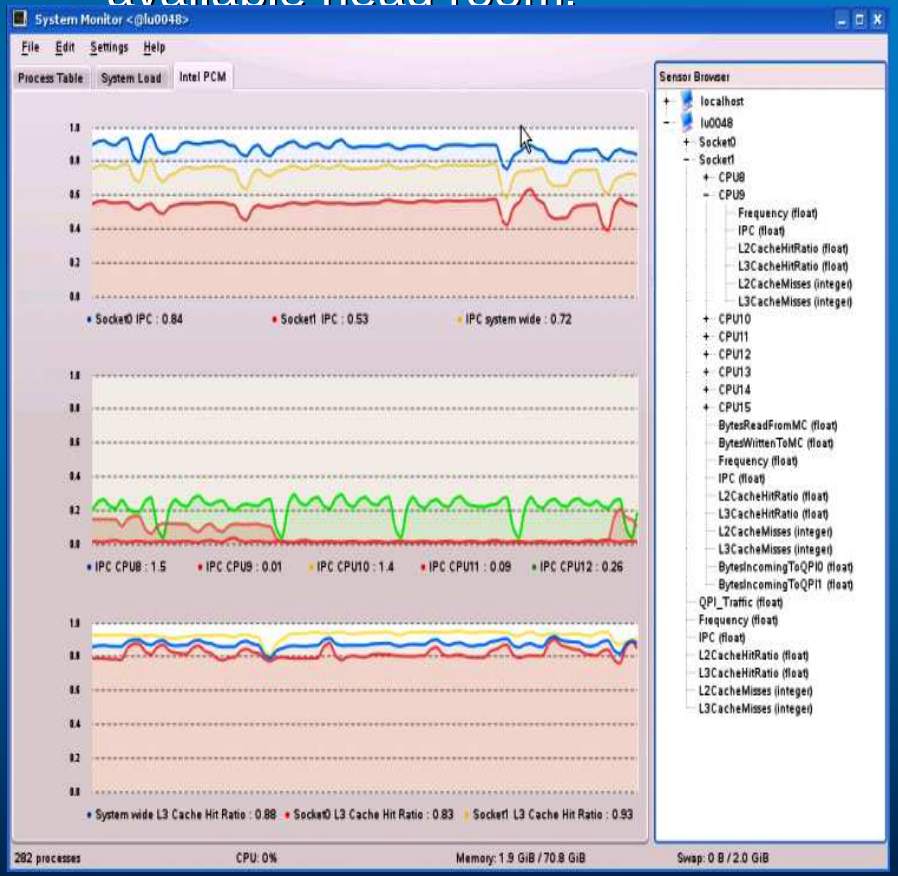
System does 2 units of work/sec

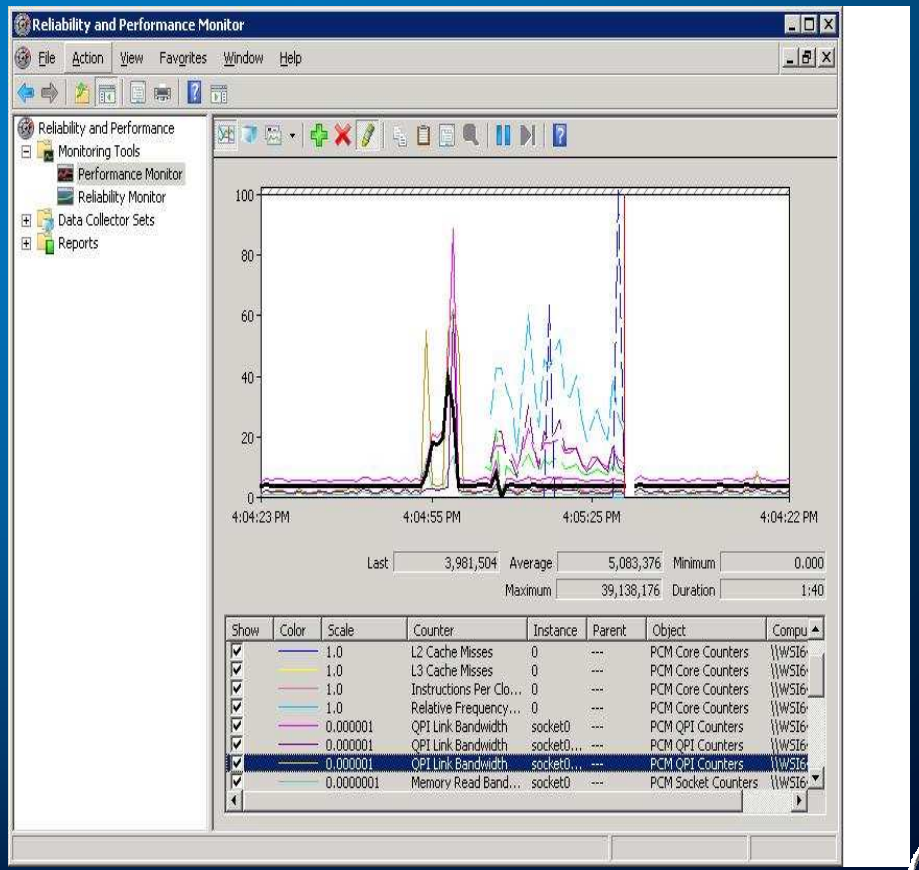


System does 1.25 units of work/sec

A better way to measure System Performance

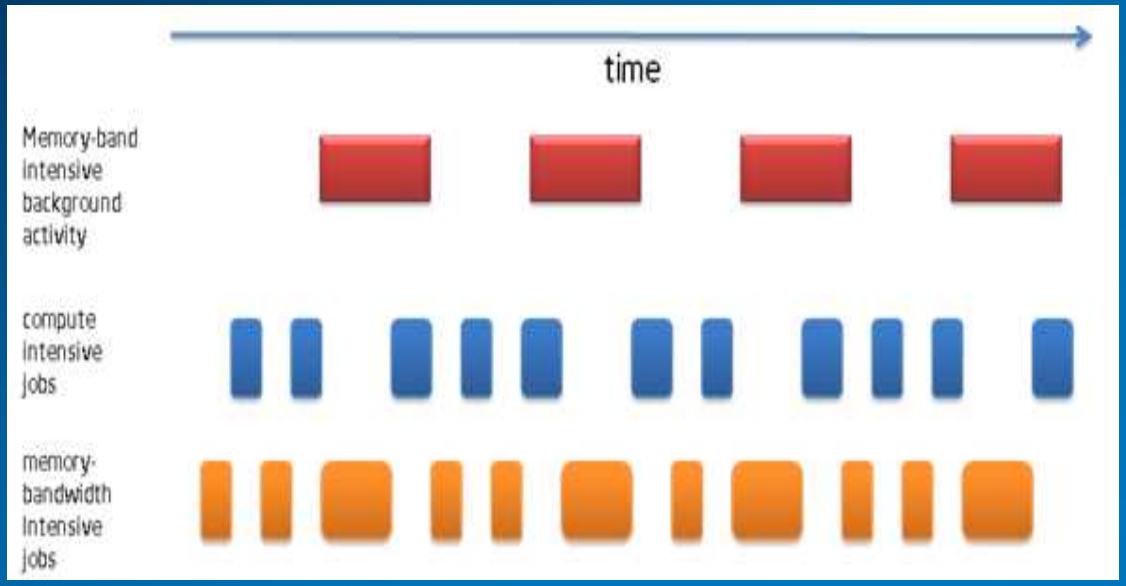
- IA provides capability to monitor performance events inside a processor
  - Using Performance Monitoring Units (PMU)
    - Core: instructions retired, elapsed core clock ticks, core freq, L2 and L3 cache hits and misses, including or excluding snoops
    - Uncore: bytes read from memory controllers, written to memory controllers, data traffic transferred by QPI etc.
- Memory BW is measured [2] and tracked as a measure of system loading and available head room.





[2] http://software.intel.com/en-us/articles/intel-performance-counter-monitor/

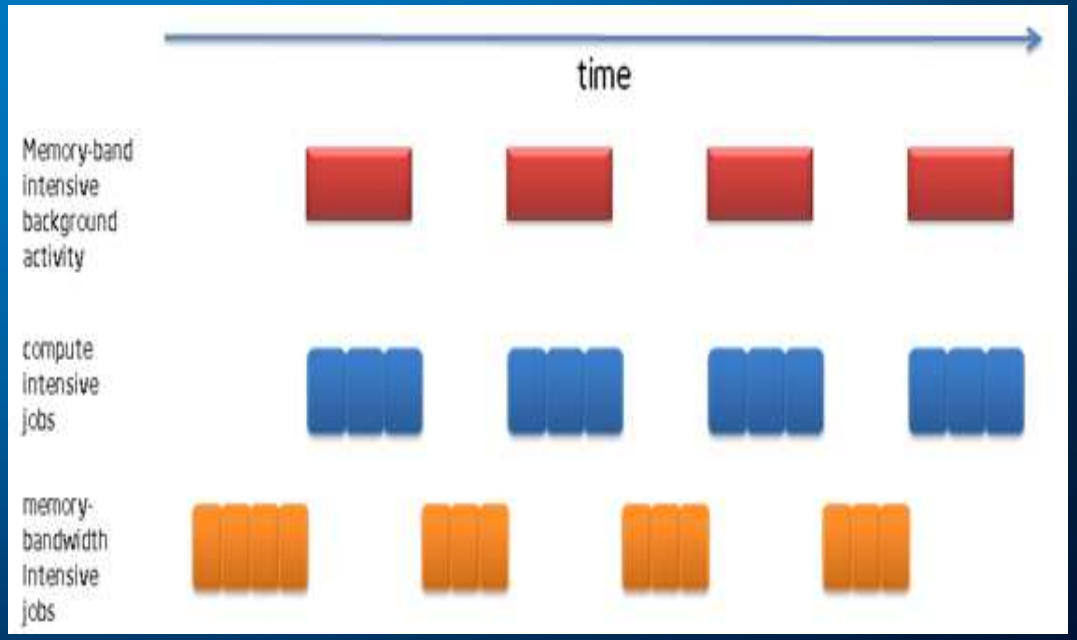
A better Task Scheduler



Scheduler without Intel® Performance Counter Monitor

A simple scheduler that executed 1000 compute intensive and 1000 memory-bandwidth intensive jobs in a single thread.

If the scheduler can detect that server memory bandwidth is used by a different process, it can adjust the future placement. A simulation shows that 2000 jobs can execute16% faster than a generic unaware scheduler.



Scheduler using Intel® Performance Counter Monitor

Why isn't workload scheduling efficient?

OpenStack Database View

Name	CPU (Cores)		CPU Type	Memory (MB)		Memory Bandwidth (%)		Storage (GB)		Network (Nics)		Power (Watts)		Running VMs
	Max	Free		Max	Free	Max	Used	Max	Free	Max	Free	Max	Free	
10.223.84.51	8	4	Nehalem	8192	4096	80	80	200	100	4	2	700	300	VM1,VM2,VM3,VM4
10.223.84.47	4	2	Nehalem	4096	2048	80	80	100	50	4	2	700	300	VM5,VM6
10.223.84.32	4	2	Nehalem	4096	2048	80	40	100	50	4	2	700	300	VM7

2 cores free?

Mem B/W available

Scheduler chooses the first server with 2 cores free to place VM8

- Ran a new workload VM8 on which was already a memory hog
- Workload on other VMs (VM5, VM6) became slow

Set of Workloads used for PoCs

1. Mcf
   - Vehicle scheduling. Uses a network simplex algorithm (which is also used in commercial products) to schedule public transport.
2. Milc
   - A gauge field generating program for lattice gauge theory programs with dynamical quarks.
3. Libqunatum
   - Simulates a quantum computer, running Shor's polynomial-time factorization algorithm.
4. H264
   - A reference implementation of H.264/AVC, encodes a videostream using 2 parameter sets. The H.264/AVC standard is expected to replace MPEG2
5. Lbm
   - Implements the "Lattice-Boltzmann Method" to simulate incompressible fluids in 3D.
6. Soplex
   - Solves a linear program using a simplex algorithm and sparse linear algebra. Test cases include railroad planning and military airlift models.
7. Omnetpp
   - Uses the OMNet++ discrete event simulator to model a large Ethernet campus network.
8. Povray
   - Image rendering. The testcase is a 1280x1024 anti-aliased image of a landscape with some abstract objects with textures using a Perlin noise function.
9. Astar
   - Pathfinding library for 2D maps, including the well known A* algorithm

Before and After WL Optimization

Node	Existing Workload	Newly scheduled workload	Average completion time of the new workload
Master	Povray (2)	Omnetpp(5)	496.8086
Slave1	Bzip2 (3)	Gcc (6)	28.49254
Slave2	Omnetpp(1)	Omnetpp(4)	451.9113

Openstack Simple Round Robin scheduler

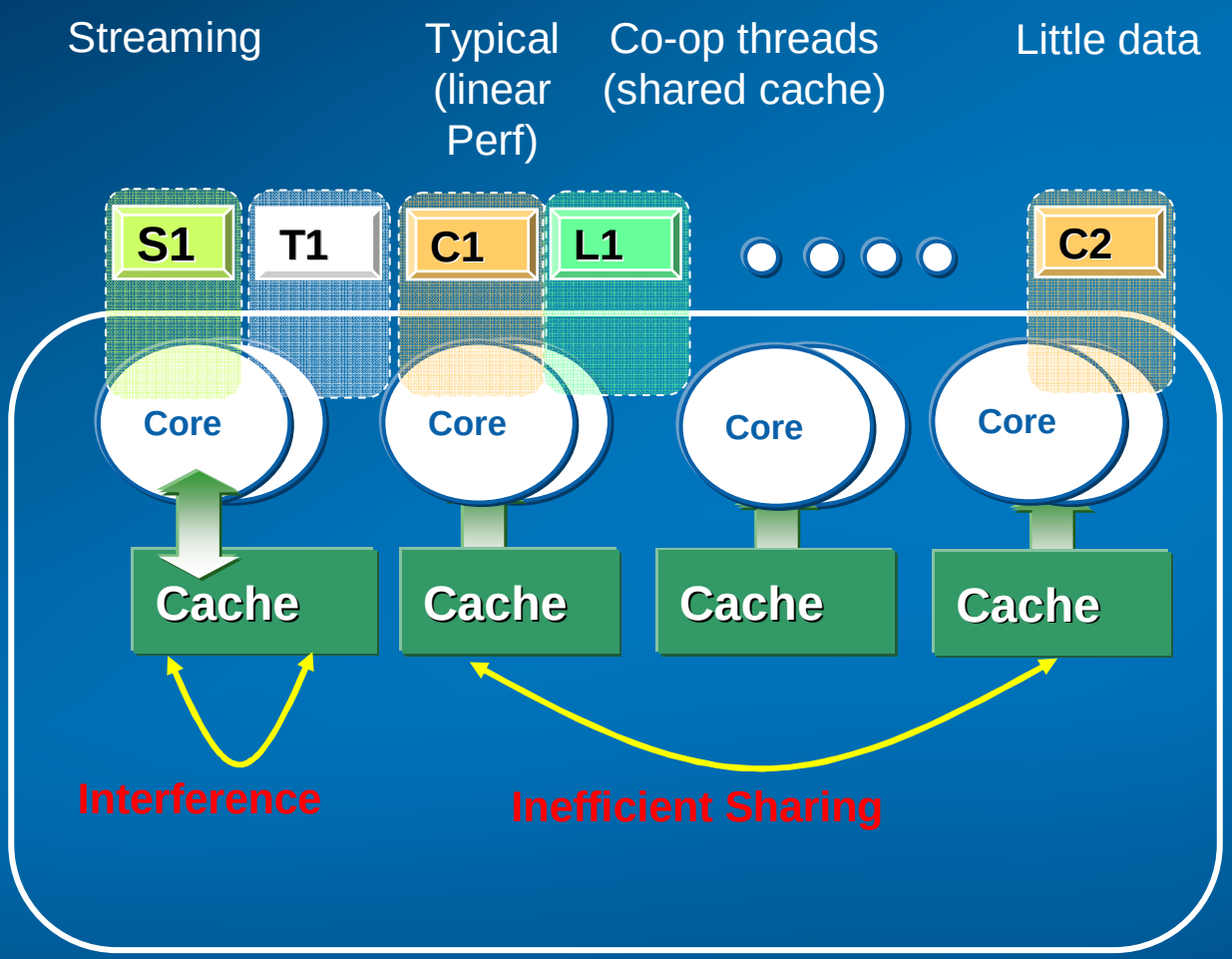
Total time1 = 977 seconds

Node	Existing Workload	Newly scheduled workload	Average completion time of the new workload
Master	Omnetpp (1)	Gcc (6)	30.3033
Slave1	Povray (2)	Omnetpp (5)	396.8547
Slave2	Bzip2 (3)	Omnetpp (4)	418.2613

Modified Openstack Policy scheduler using heuristics

Total time2 = 845.4 seconds
Run-time Improvement = 13.5%

Application Behavior & Overall Performance



Potential to improve overall performance

Monitor resource usage and group/partition accordingly

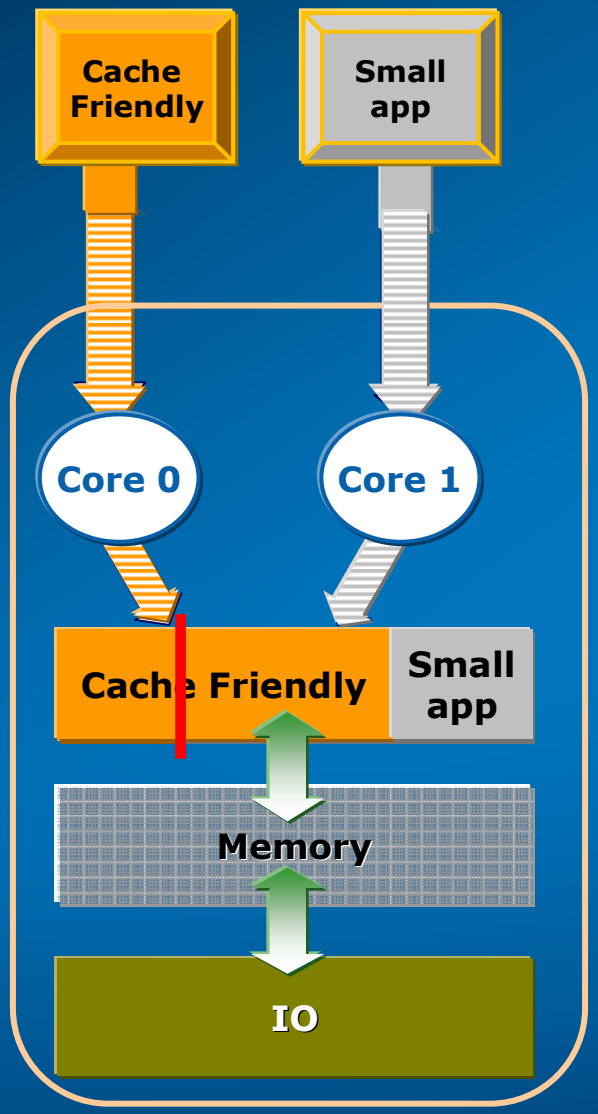
Performance Loss

No understanding of resource usage behavior

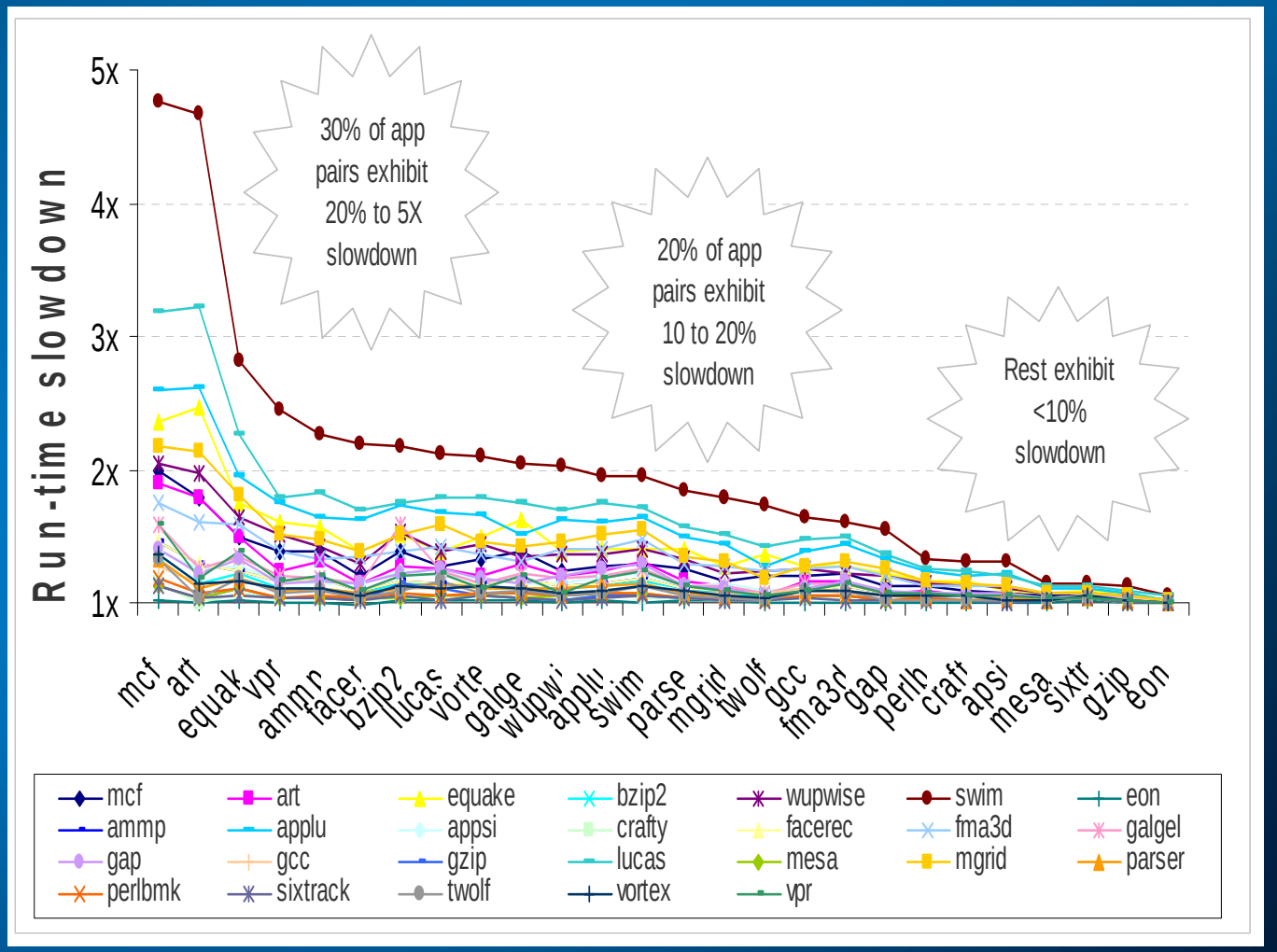
What else is needed on the Network, Storage, Manageability and Security side to achieve efficient sharing?

Managing resource contention can improve overall throughput

An example: Shared Cache Resource Contention



Minimize contention

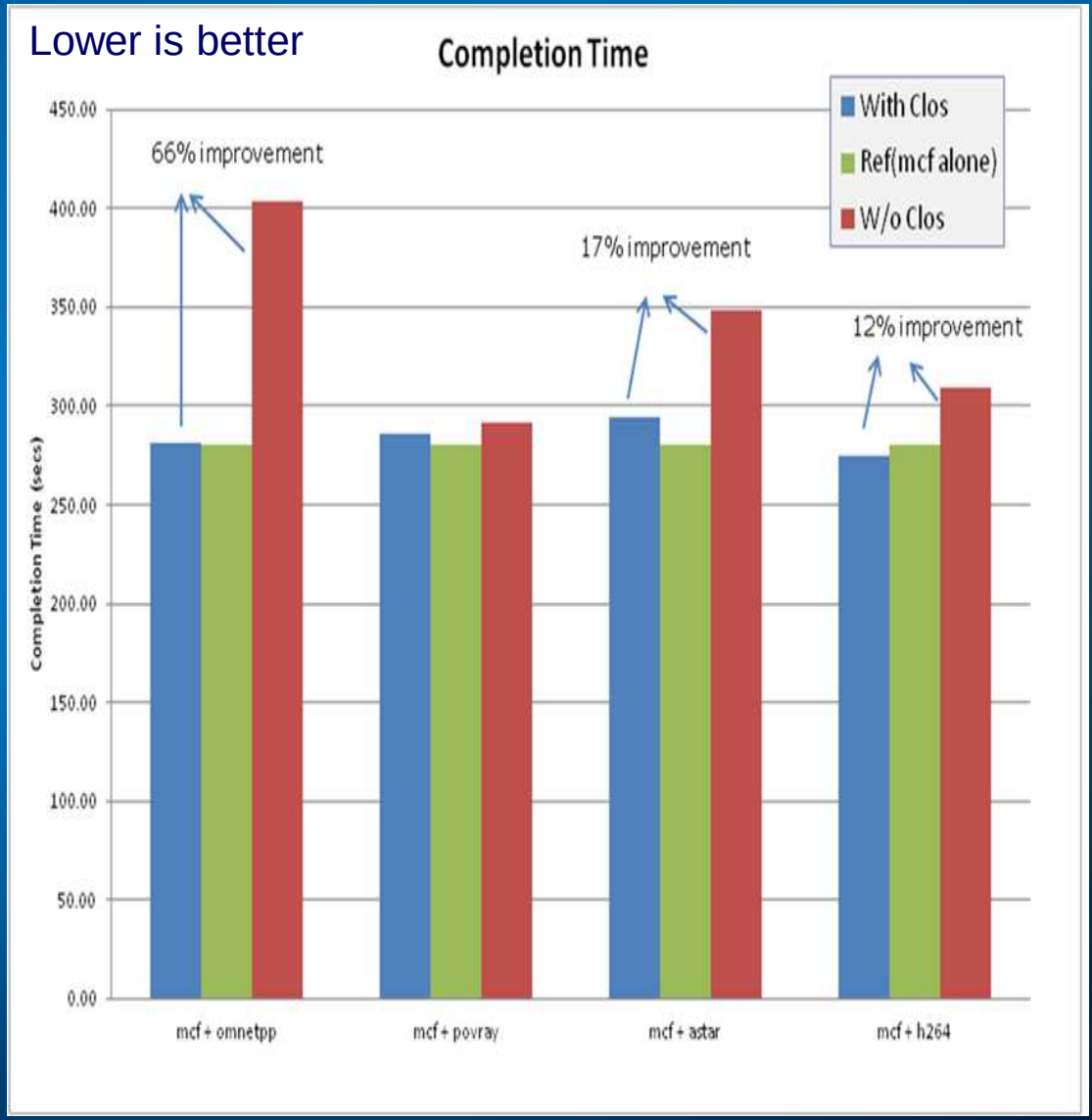


Example: Performance degradation due to Cache contention

Need a contention aware scheduling mechanism

Noisy Neighbor Detection & Prevention for Platform Cache QoS

- Establish technical feasibility
- Determine platform touch points & ecosystem enabling needs
- Demonstrate valuable usage models in IaaS



Static Partitioning shows about 35% improvement for mcf

Next Steps

- Run real or representative WLs in a Cloudy environment
- Quantify any improvements using lower level metrics
- Broaden the learnings to get better EDA efficiency and results