Improving HPC Application Performance in Cloud through Dynamic Load Balancing

***Abhishek Gupta*, Osman Sarood, Laxmikant V. Kale**

Dejan Milojicic (HP labs)

04/15/2013

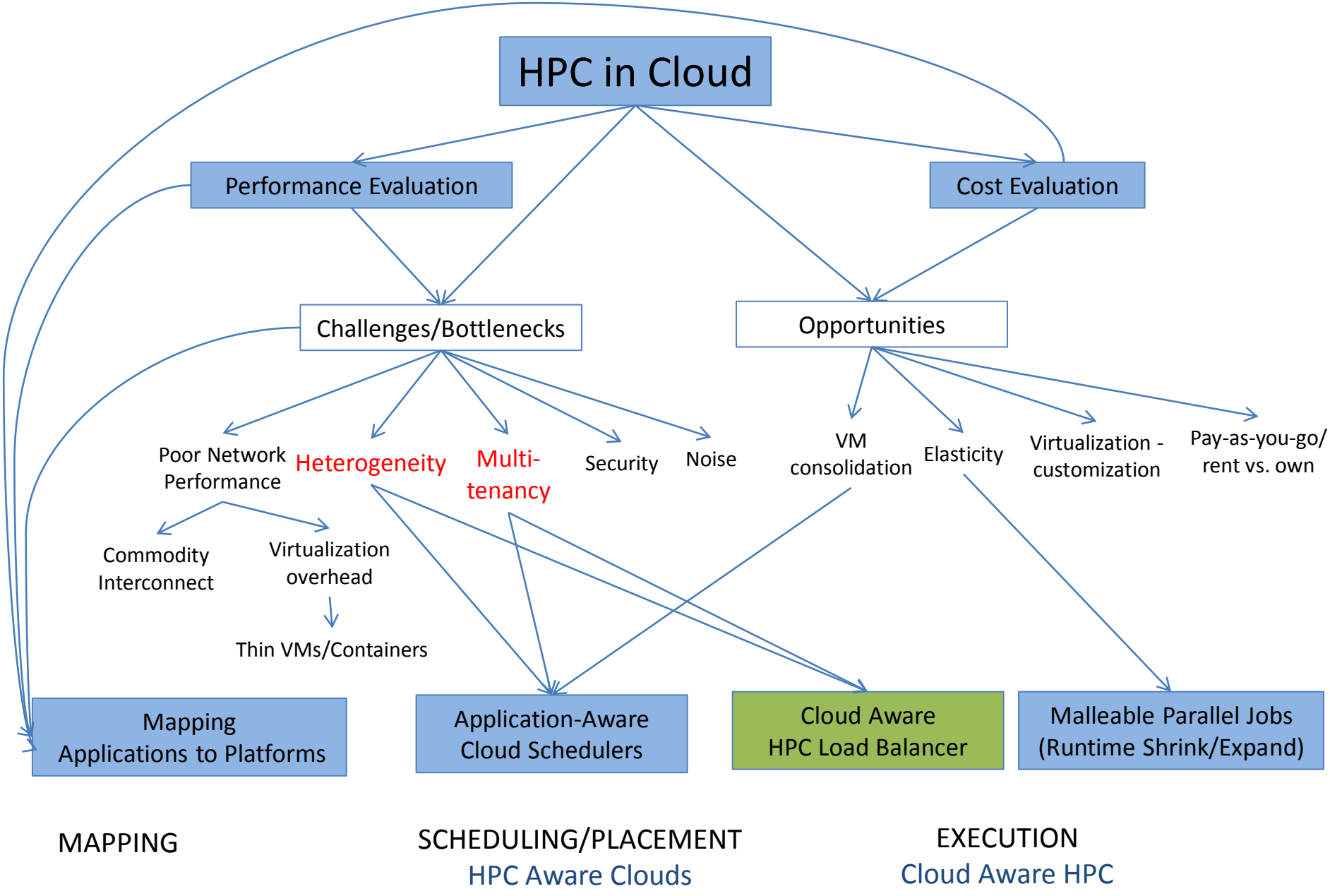
Motivation: Why Clouds for HPC ?

- Rent vs. own, *pay-as-you-go*
  - No startup/maintenance cost, cluster create time
- *Elastic* Resources
  - No risk e.g. in under-provisioning
  - Power savings, prevents underutilization
- Benefits of virtualization
  - Flexibility and Customization
  - Security and Isolation
  - Migration
  - Resource Control

- Hence, a cost-effective and timely solution
  - e.g. substitute/addition when Supercomputers are heavily loaded

Motivation: HPC-Cloud Gap

- Today's HPC not Cloud-aware, Clouds not HPC-aware!
  - Only embarrassingly parallel or small scale HPC apps run in Clouds
  - Typical Cloud interconnect, scheduler, heterogeneity, multi-tenancy largest obstacles for HPC apps


HPC in Cloud
Performance Evaluation
Cost Evaluation
Challenges/Bottlenecks
Opportunities
Poor Network Performance
Heterogeneity
Multi-tenancy
Security
Noise
VM consolidation
Elasticity
Virtualization - customization
Pay-as-you-go/ rent vs. own
Commodity Interconnect
Virtualization overhead
Thin VMs/Containers
Mapping Applications to Platforms
Application-Aware Cloud Schedulers
Cloud Aware HPC Load Balancer
Malleable Parallel Jobs (Runtime Shrink/Expand)
MAPPING
SCHEDULING/PLACEMENT
HPC Aware Clouds
EXECUTION
Cloud Aware HPC

NEED FOR LOAD BALANCER FOR HPC IN CLOUD

- *Heterogeneity and multi-tenancy intrinsic in clouds*
- Heterogeneity: Cloud economics is based on:
  - Creation of a cluster from existing pool of resources and
  - Incremental addition of new resources.
- Multi-tenancy: Cloud providers run a profitable business by improving utilization of underutilized resources
  - Cluster-level by serving large number of users,
  - Server-level by consolidating VMs of complementary nature (such as memory- and compute-intensive) on same server.
  - Hence multi-tenancy can be at resource-level (memory, CPU), node-level, rack-level, zone-level, or data center level.

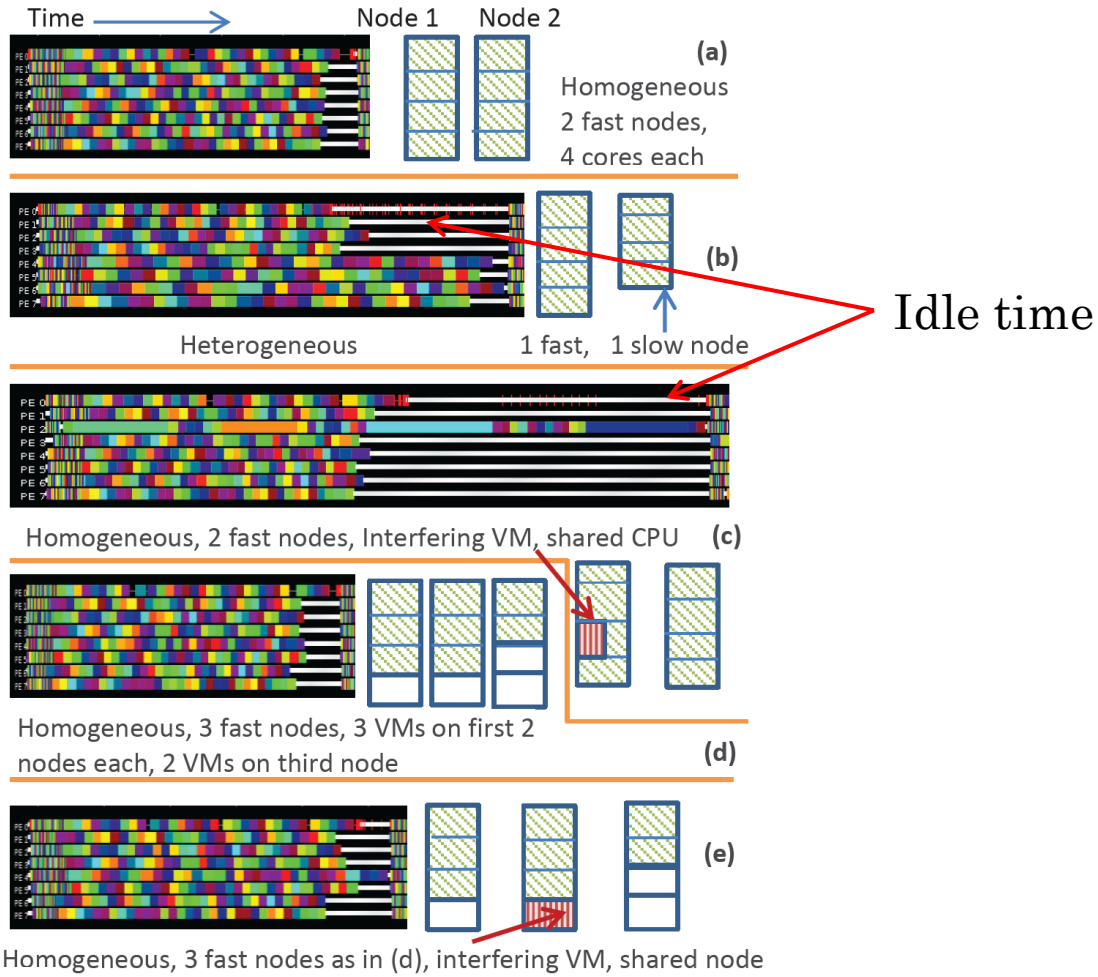
Research Goals

- Can we reduce the divide between HPC and Cloud?
  - Make Clouds HPC-aware
  - **Make HPC cloud-aware**

- Address Heterogeneity, Multi-tenancy by adaptive runtime system

- **Challenge**: Running in VMs makes it difficult to determine if (and how much of) the load imbalance is
  - Application-intrinsic or
  - Caused by extraneous factors.

NEED FOR LOAD BALANCER FOR HPC IN CLOUD



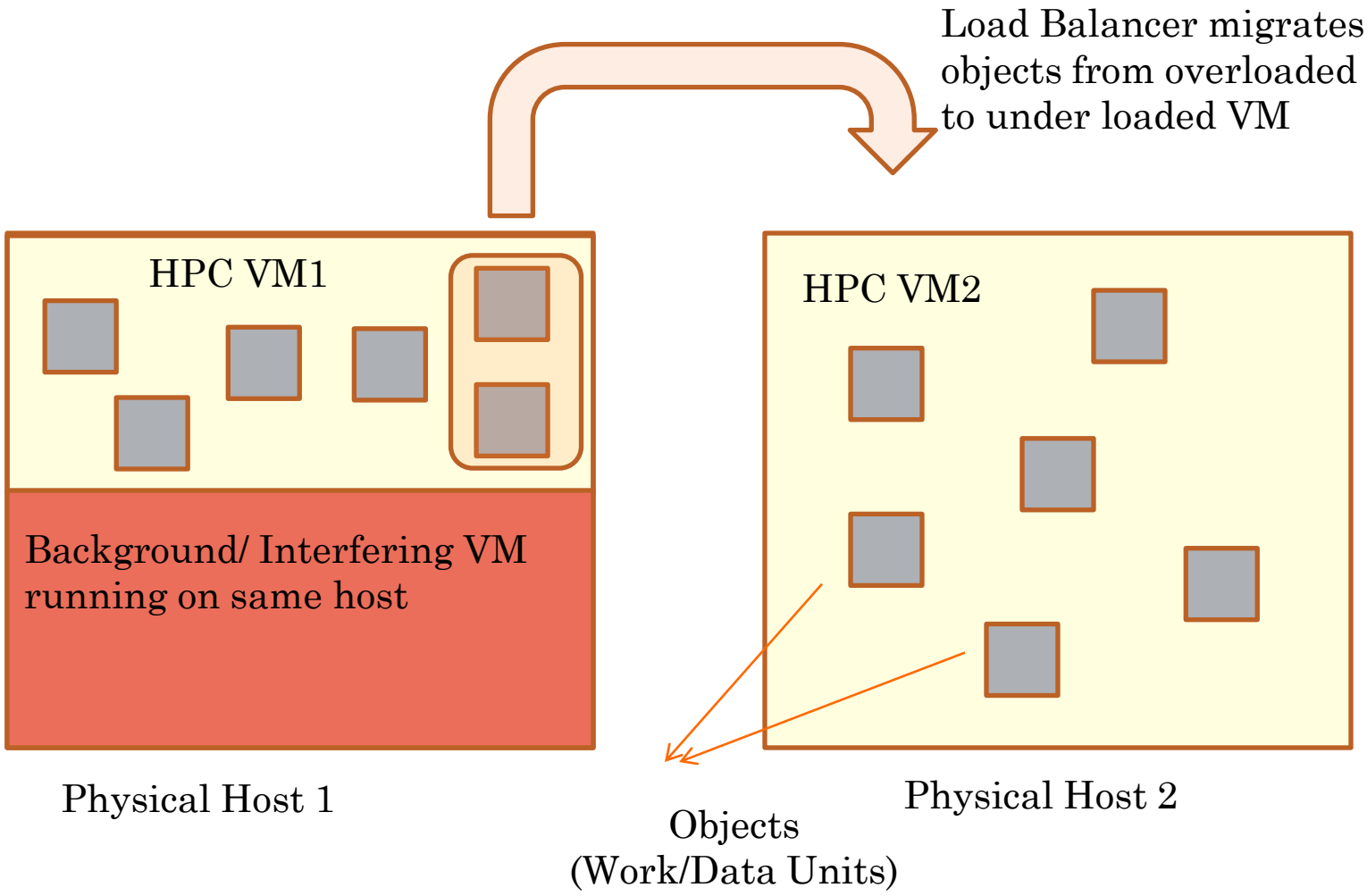
Experimental setup (on right) and timeline of 8 VMs showing one iteration of Stencil2D: white portion = idle time, colored portions = application functions.

Charm++'s and Load Balancing!

- Migratable objects
  - Mandatory for our scheme to work
  - Supports fault tolerance
- Object-based over-decomposition
  - Helpful for refinement load balancing
- Time logging for all objects
  - Central to load balancing decisions
- Supports plugin load balancer

CLOUD-AWARE LOAD BALANCER FOR HPC

- Static Heterogeneity: estimate the CPU capabilities for each VCPU, and use those estimates to drive the load balancing.
- Simple estimation strategy + periodic refinement of load distribution
- Dynamic Heterogeneity (Interfering VMs): Periodic object(task) redistribution


Load Balancer migrates objects from overloaded to under loaded VM
HPC VM1
Background/ Interfering VM running on same host
HPC VM2
Physical Host 1
Physical Host 2
Objects
(Work/Data Units)

CLOUD-AWARE LOAD BALANCER FOR HPC

- Instrumenting the time spent on each task,
- Predict future load based on the execution time of recently completed iterations.
- Impact of interference: instrument the load external to the application under consideration, referred to as the background load

Load balancing approach

To get a processor-independent measure of task loads, normalize the execution times to number of ticks

All processors should have load close to average load

$$\forall p \in P, \sum_{i=1}^{N_p} (t_i * f_{m_i^{k-1}}) + O_p * f_p - Tk_{avg} < \epsilon$$

Average load depends on task execution time and overhead

$$Tk_{avg} = \frac{\sum_{p=1}^{P}((\sum_{i=1}^{N_p} t_i + O_p) * f_p)}{P}$$

Overhead is the time processor is not executing tasks and not in idle mode.

$$O_p = T_{lb} - \sum_{i=1}^{N_p} t_i^p - t_{idle}^p$$

Charm++ load balancing database

from /proc/stat file

T_{lb}: wall clock time between two load balancing steps,
T_i: CPU time consumed by task i on VCPU p

Load balancing approach

- After each user defined time interval
  - Categorize each VCPU as overloaded/underloaded
  - Create a heap of overloaded processors (H)
  - Create a set of underloaded processors (S)
  - Until *H* is not empty:
    - Transfer tasks from most overloaded processor from the *H* to any processor from *S*
    - The largest task currently placed on donor such that it can be transferred to a core from underloaded Set without overloading it
    - Update task mappings

LOAD BALANCING APPROACH

Algorithm 1 Refinement Load Balancing for Cloud

1: On Master VCPU on each load balance step
2: **for** $p \in [1, P]$ **do**
3: **if** $isHeavy(p)$ **then**
4: $overHeap.add(p)$
5: **else if** $isLight(p)$ **then**
6: $underSet.add(p)$
7: **end if**
8: **end for**
9: createOverHeapAndUnderSet()
10: **while** $overHeap$ NOT NULL **do**
11: $donor$ = deleteMaxHeap($overHeap$)
12: $(bestTask, bestCore)$ = getBestCoreAndTask($donor, underSet$)
13: $m^k_{bestTask} = bestCore$
14: updateHeapAndSet()
15: **end while**

16: **procedure isHeavy(p)** {isLight(p) is same except that the condition at line 21 is replaced by by $Tk_{avg} - totalTicks > \epsilon$}
17: **for** $i \in [1, N_p]$
18: $totalTicks+ = t_i * f_p$
19: **end for**
20: $totalTicks+ = O_p * f_p$
21: **if** $totalTicks - Tk_{avg} > \epsilon$
22: return **true**
23: **else**
24: return **false**
25: **end if**
26: **end procedure**

Evaluation: Experimental Testbed

- OpenStack on Open Cirrus test bed at HP Labs site, 3 types of servers:
  - Intel Xeon E5450 (12M Cache, 3.00 GHz) - Fast
  - Intel Xeon X3370 (12M Cache, 3.00 GHz) - Fast
  - Intel Xeon X3210 (8M Cache, 2.13 GHz) - Slow
- KVM as hypervisor, *virtio-net* for n/w virtualization
- VMs: m1.small (1 core, 2 GB RAM, 20 GB disk)
- Connected using commodity Ethernet – 1Gbps internal to rack and 10Gbps cross-rack.
- Pin the virtual cores to physical cores using `vcpupin` command.

Benchmarks and Applications

- Stencil2D – 5-point stencil computation kernel
- Wave2D – finite differencing to calculate pressure information over a discretized 2D grid, for simulation of a wave motion.
- Mol3D – A 3-D molecular dynamics simulation application. We used the Apoa1 dataset (92K atoms).

Written in Charm++

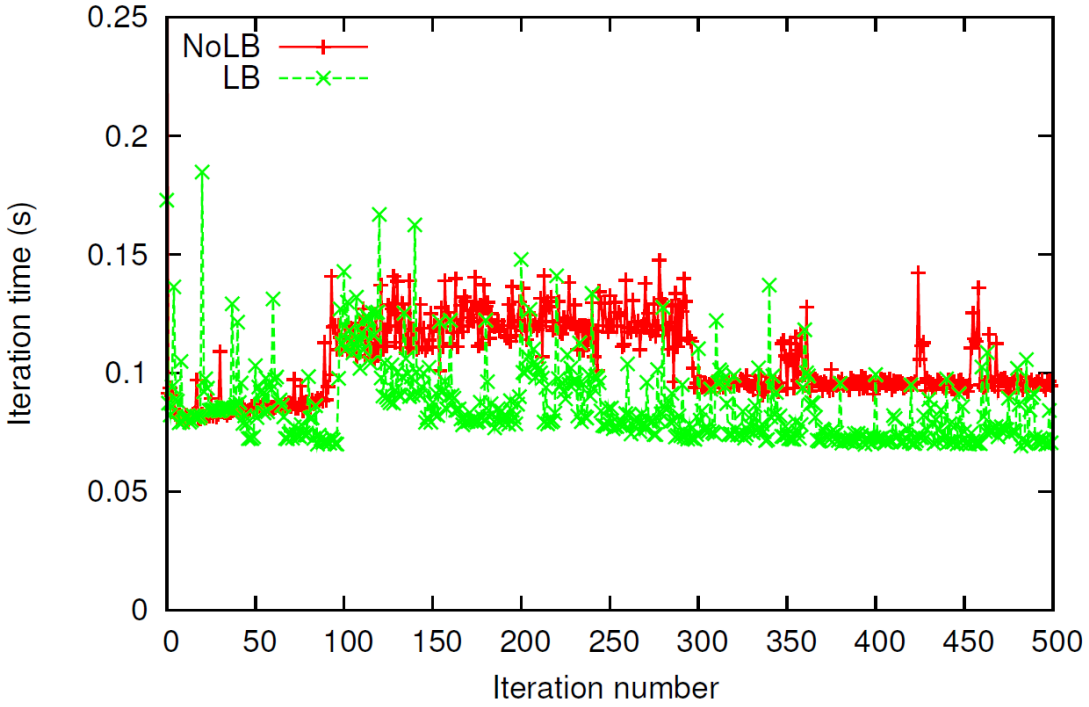
- net-linux-x86-64 machine layer
- –O3 optimization level.

- For Stencil2D, problem size 8K × 8K. For Wave2D, problem size 12K × 12K. Each object size is kept 256 × 256.

Interference:

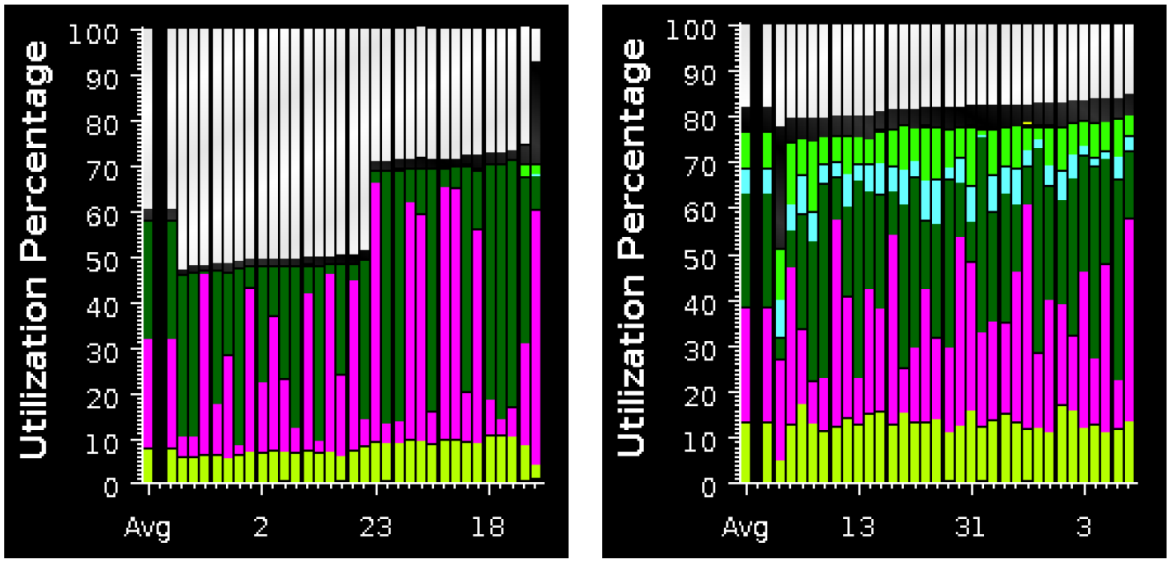
- Sequential NPB-FT (NAS Parallel Benchmark - Fourier Transform) Class A as source of interference
- Interfering VM pinned to one of the cores that the VMs of our parallel runs use

Results: Analysis using stencil3d



LB vs. NoLB: 32 VMs Stencil2D on heterogeneous hardware, interfering VM from 100^{th} to 300^{th} iteration

Results

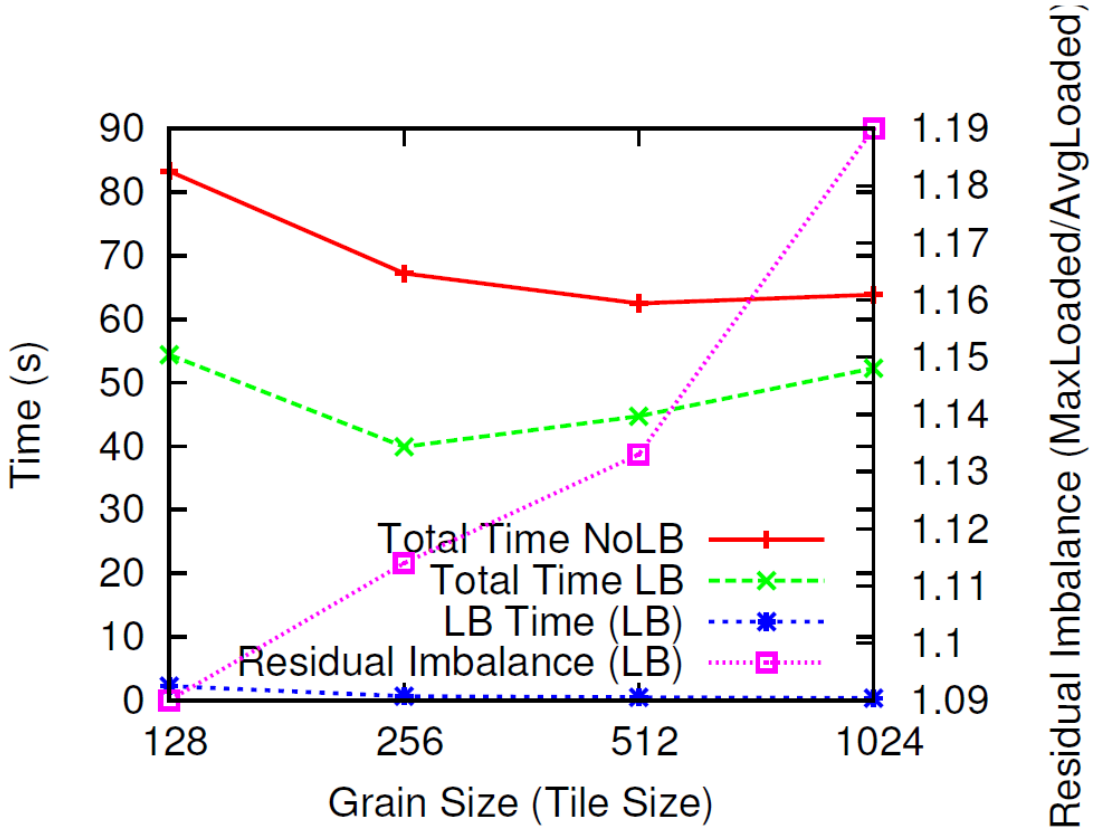


(a) NoLB (b) LB

CPU utilization of Stencil2D on 32 VMs: white = idle time, black = overhead (including background load), colored portions = application functions, x-axis = VCPU.

Results: Effect of Grain Size

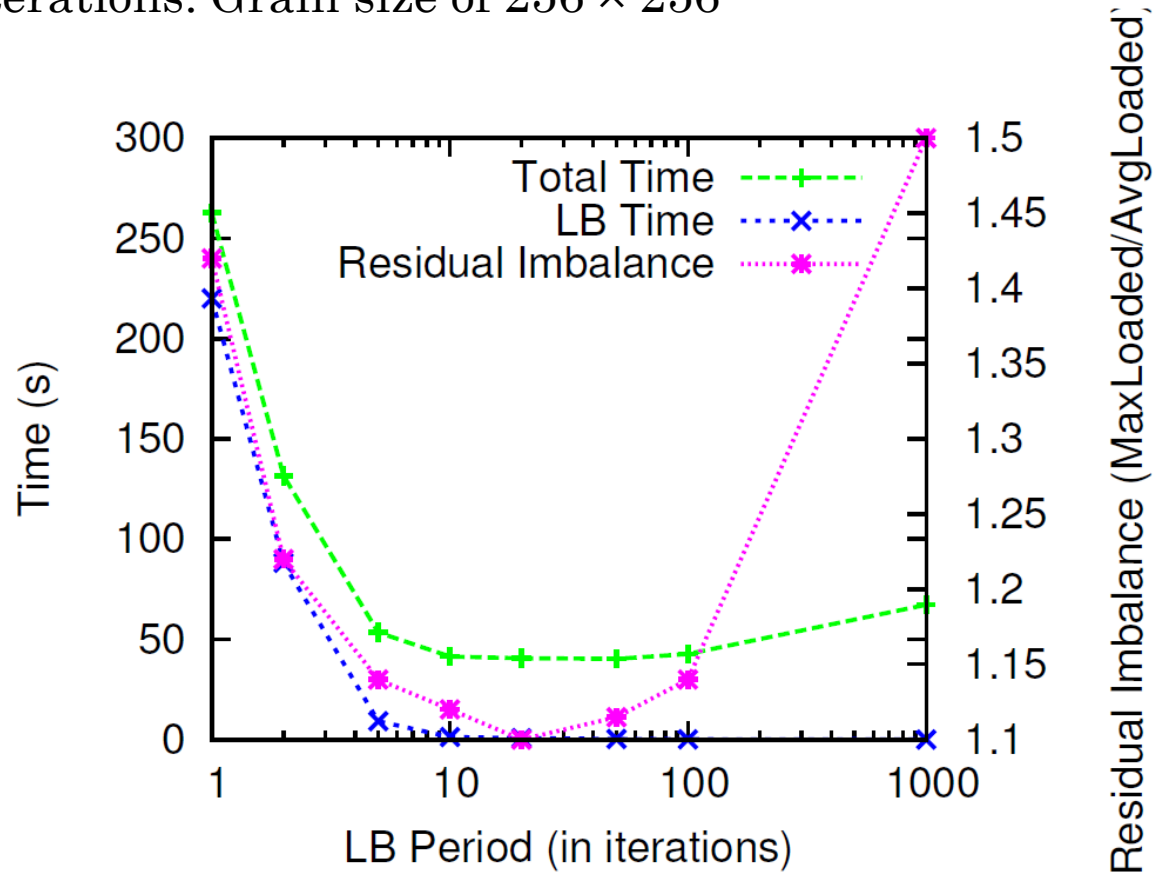
Stencil2d (8K × 8K) on 32 VMs (Fast processors, one interfering VM), 500 iterations. For LB case, load balancing every 20 steps



Lower is better

Results: Effect of LB Period

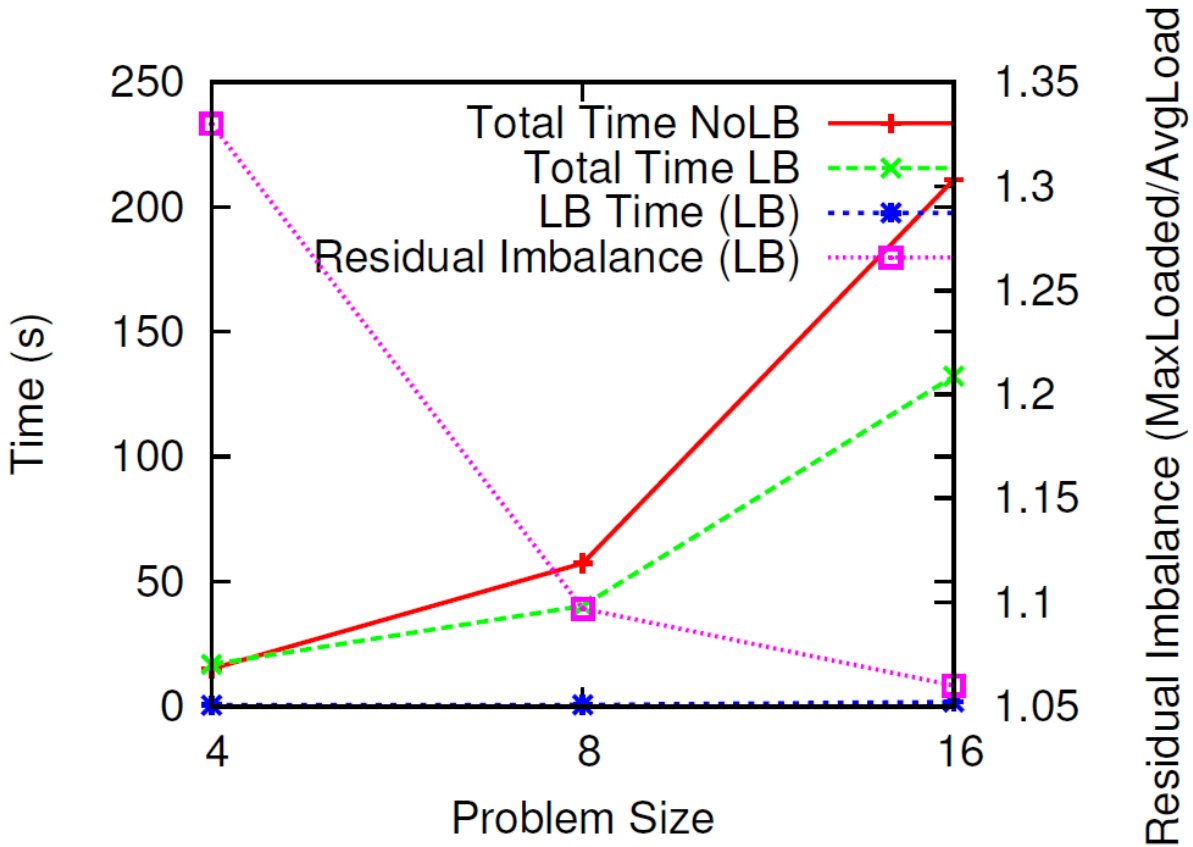
Stencil2d (8K × 8K) on 32 VMs (Fast processors, one interfering VM), 500 iterations. Grain size of 256 × 256



Lower is better

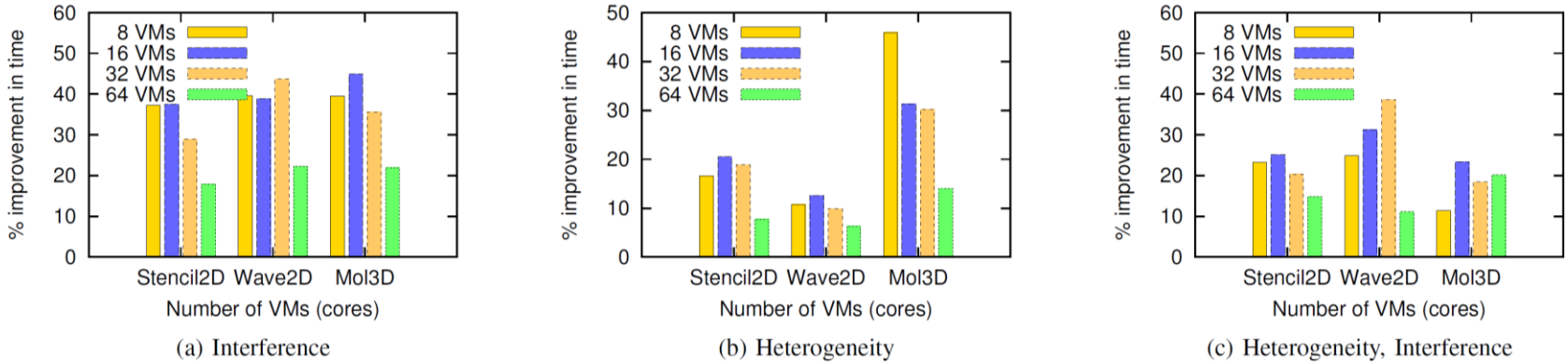
Results: Effect of Problem Size

Stencil2d on 32 VMs (Fast processors, one interfering VM), 500 iterations. Grain size of 256 × 256, load balancing every 20 steps



Lower is better

Results: Improvements by LB

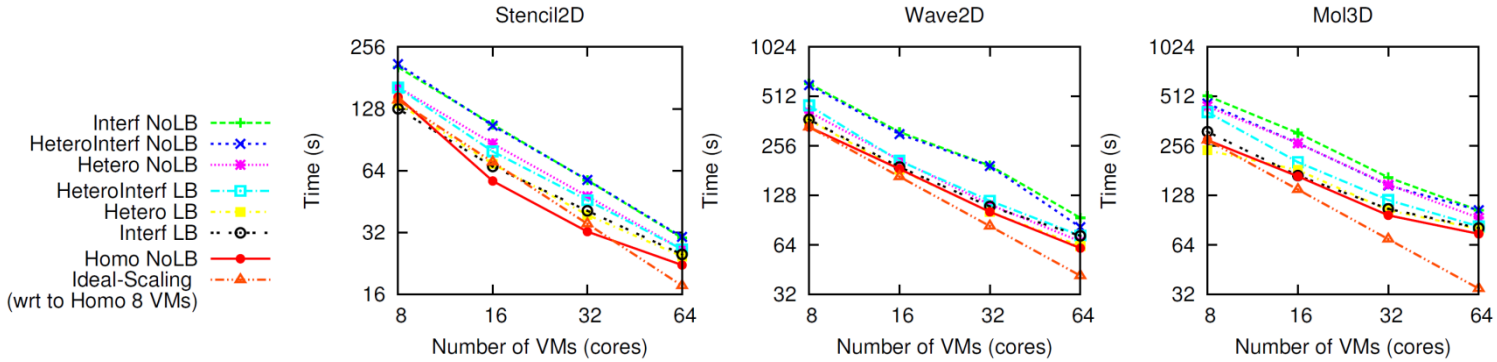


% benefits obtained by load balancing in presence of interference and/or heterogeneity for three different applications and different number of VMs, strong scaling

(a) Interference - one interfering VM, all Fast nodes,
(b) Heterogeneity – one Slow node, hence four Slow VMs, rest Fast
(c) Heterogeneity and Interference – one Slow node, hence four Slow VMs, rest Fast, one interfering VM (on a Fast core) which starts at iteration 50.

500 iterations for Stencil2D and Wave2D and 200 iterations for Mol3D, with load balancing every 20th step

Results: Parallel Scaling



Scaling curves with and without load balancing in presence of interference and/or heterogeneity for three different applications, strong scaling

Related work

- Studies on HPC in cloud
  - Walker, He et al., Ekanayake et al., DoE's Magellan project
  - Cloud can be potentially more cost-effective than supercomputers for some HPC applications
  - Challenges: insufficient network and I/O performance in cloud, resource heterogeneity, and unpredictable interference arising from other VMs.

Bridging the gap between HPC and Cloud

- Bring clouds closer to HPC
  - HPC-optimized clouds: Amazon Cluster Compute, DoE's Magellan
  - HPC-aware cloud scheduler
    - Gupta et al.: HPC Aware VM Placement in Infrastructure Clouds
    - OpenStack scheduler architecture-aware

- Bring HPC closer to clouds.
  - Fan et al. proposed topology aware deployment of scientific applications in cloud, and mapped the communication topology of an HPC application to the VM physical topology

- http://charm.cs.uiuc.edu/research/cloud

Lessons Learned

- Heterogeneity-awareness: significant performance improvement for HPC in cloud.
- Besides the static heterogeneity, multi-tenancy in cloud introduces dynamic heterogeneity, which is random and unpredictable.
  - Poor performance of tightly-coupled iterative HPC applications.
- Even without the accurate information of the nature and amount of heterogeneity (static and dynamic but hidden from user as an artifact of virtualization), the approach of periodically measuring idle time and migrating load away from time-shared VMs works well in practice.
- Tuning the parallel application for efficient execution in cloud is non-trivial.
  - Choice of load balancing period and computational granularity can have significant impact on performance
  - Optimal values depend on application characteristics, size, and scale.
  - Runtime systems which can automate the selection and dynamic adjustment of such decisions will be increasingly useful in future.

Conclusions and future work

- A load balancing technique
  - Accounts for heterogeneity
  - Handles interfering VMs in cloud
  - Uses object migration to restore load balance.
- Experimental results on actual cloud showed that we were able to reduce execution time by up to 45% compared to no load balancing

Future Work

- Extend our load balancer such that data migration is performed only if we expect gains that can offset the cost of migration.
- Evaluate our techniques on a larger scale – on an actual cloud, if available in future, or through simulated or emulated environment.
- Explore the use of VM steal cycles, where supported

Acknowledgements

- This work was supported by HP Labs' 2012 IRP award