Evaluating and Improving the Performance and Scheduling of HPC Applications in Cloud

Abhishek Gupta, Paolo Faraboschi, *Fellow, IEEE*, Filippo Gioachin, Laxmikant V. Kale, *Fellow, IEEE*, Richard Kaufmann, Bu-Sung Lee, Verdi March, Dejan Milojicic, *Fellow, IEEE*, and Chun Hui Suen

Abstract—Cloud computing is emerging as a promising alternative to supercomputers for some high-performance computing (HPC) applications. With cloud as an additional deployment option, HPC users and providers are faced with the challenges of dealing with highly heterogeneous resources, where the variability spans across a wide range of processor configurations, interconnects, virtualization environments, and pricing models. In this paper, we take a holistic viewpoint to answer the question—*why* and *who* should choose cloud for HPC, for *what* applications, and *how* should cloud be used for HPC? To this end, we perform comprehensive performance and cost evaluation and analysis of running a set of HPC applications on a range of platforms, varying from supercomputers to clouds. Further, we improve performance of HPC applications in cloud by optimizing HPC applications' characteristics for cloud and cloud virtualization mechanisms for HPC. Finally, we present novel heuristics for online application-aware job scheduling in multi-platform environments. Experimental results and simulations using CloudSim show that current clouds cannot *substitute* supercomputers but can effectively *complement* them. Significant improvement in average turnaround time (up to 2X) and throughput (up to 6X) can be attained using our intelligent application-aware dynamic scheduling heuristics compared to single-platform or application-agnostic scheduling.

Index Terms—HPC, cloud, performance analysis, economics, job scheduling, application-awareness, characterization

1 Introduction

Increasingly, some academic and commercial HPC users are looking at clouds as a cost effective alternative to dedicated HPC clusters [1], [2], [3], [4]. *Renting* rather than owning a cluster avoids the up-front and operating expenses associated with a dedicated infrastructure. Clouds offer additional advantages of a) *elasticity*—on-demand provisioning, and b) virtualization-enabled flexibility, customization, and resource control.

Despite these advantages, it still remains unclear whether, and when, clouds can become a feasible substitute or complement to supercomputers. HPC is performance-oriented, whereas clouds are cost and resource-utilization oriented. Furthermore, clouds have traditionally been designed to run business and web applications. Previous studies have shown that commodity interconnects and the overhead of virtualization on network and storage performance are major performance barriers to the adoption of cloud for HPC [1], [2], [3], [4], [5], [6]. While the outcome of these studies paints a rather pessimistic view of HPC clouds, recent efforts towards HPC-optimized clouds, such as Magellan [1] and Amazon's EC2 Cluster Compute [7], point to a promising direction to overcome some of the fundamental inhibitors.

- *A. Gupta and L.V. Kale are with the Department Computer Science, University of Illinois at Urbana Champaign, Urbana, IL 61801. Email: {charm, kale}@illinois.edu.*
- *P. Faraboschi, F. Gioachin, R. Kaufmann, B.-S. Lee, V. March, D. Milojicic, and C.H. Suen are with the Hewlett Packard Labs, Palo Alto, CA94304.*

Manuscript received 3 Feb. 2014; revised 1 July 2014; accepted 2 July 2014. Date of publication 17 July 2014; date of current version 7 Sept. 2016. Recommended for acceptance by I. Bojanova, R.C.H. Hua, O. Rana, and M. Parashar. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below. Digital Object Identifier no. 10.1109/TCC.2014.2339858

HPC clouds rapidly expand the application user base and the available platform choices to run HPC workloads: from in-house dedicated supercomputers, to commodity clusters with and without HPC-optimized interconnects and operating systems, to resources with different degrees of virtualization (full, CPU-only, none), to hybrid configurations that offload part of the work to the cloud. HPC users and cloud providers are faced with the challenge of choosing the optimal platform based upon a limited knowledge of application characteristics, platform capabilities, and the target metrics such as cost.

This trend results in a potential mismatch between the required and selected resources for HPC application. One possible undesirable scenario can result in part of the infrastructure being overloaded, and another being idle, which in turn yields large wait times and reduced overall throughput. Existing HPC scheduling systems are not designed to deal with these issues. Hence, novel scheduling algorithm and heuristics need to be explored to perform well in such scenarios.

Unlike previous works [1], [2], [3], [4], [5], [8], [9] on benchmarking clouds for science, we take a more holistic and practical viewpoint. Rather than limiting ourselves to the problem—*what* is the performance achieved on cloud versus supercomputer, we address the bigger and more important question—*why* and *who* should choose (or not choose) cloud for HPC, for *what* applications, and *how* should cloud be used for HPC? While addressing this research problem, we make the following contributions.

- We evaluate the performance of HPC applications on a range of platforms varying from supercomputer to cloud. Also, we analyze bottlenecks and the correlation between application characteristics

TABLE 1
Testbed

Resource	Platform					
	Ranger	Taub	Open Cirrus	Private Cloud	Public Cloud	EC2-CC Cloud
Processors in a Node	16×AMD Opteron QC @2.3 GHz	12×Intel Xeon X5650 @2.67 GHz	4×Intel Xeon E5450 @3.00 GHz	2×QEMU Virtual CPU @2.67 GHz	4×QEMU Virtual CPU @2.67 GHz	16×Xen HVM VCPU @2.6 GHz
Memory	32 GB	48 GB	48 GB	6 GB	16 GB	60 GB
Network	Infiniband (1 GB/s)	QDR Infiniband	10GigE internal, 1GigE x-rack	Emulated 1GigE	Emulated 1GigE	Emulated 10GigE
OS	Linux	Sci. Linux	Ubuntu 10.04	Ubuntu 10.04	Ubuntu 10.10	Ubuntu 12.04

and observed performance, identifying *what* applications are suitable for cloud. (Section 3, Section 4)

- We also evaluate the performance when running the same benchmarks on exactly same hardware, without and with different virtualization technologies, thus providing a detailed analysis of the isolated impact of virtualization on HPC applications (Section 6).
- To bridge the divide between HPC and clouds, we present the complementary approach of (1) making HPC applications cloud-aware by optimizing an application's computational granularity and problem size for cloud and (2) making clouds HPC-aware using thin hypervisors, OS-level containers, and hypervisor- and application-level CPU affinity, addressing – *how* to use cloud for HPC. (Section 5, Section 6)
- We investigate the economic aspects of running in cloud and discuss *why* it is challenging or rewarding for cloud providers to operate business for HPC compared to traditional cloud applications. We also show that small/medium-scale users are the likely candidates *who* can benefit from an HPC-cloud. (Section 7)
- Instead of considering cloud as a substitute of supercomputer, we investigate the co-existence of multiple platforms—supercomputer, cluster, and cloud. We research novel heuristics for application-aware scheduling of jobs in this multi-platform scenario significantly improving average job turnaround time (up to $2X$) and job throughput (up to $6X$), compared to running all jobs on supercomputer. (Section 8)

The insights from performance evaluation, characterization, and multi-platform scheduling are useful for both—HPC users and cloud providers. Users can better quantify the benefits of moving to a cloud. Cloud providers can optimize the allocation of applications to their infrastructure to maximize utilization and turnaround times.

2 Evaluation Methodology

In this section, we describe the platforms which we compared and the applications suite used in this study.

2.1 Experimental Testbed

We selected platforms with different interconnects, operating systems, and virtualization support to cover the dominant classes of infrastructures available today to an HPC user. Table 1 shows the details of each platform. In case of cloud a *node* refers to a virtual machine and a *core* refers to a virtual core. For example, "2 × QEMU Virtual CPU @2.67 GHz" means each VM has two virtual cores. Ranger [10] at TACC was a supercomputer (decommissioned in Feb. 2013), and Taub at UIUC is an HPC-optimized cluster. Both use Infiniband as interconnect. Moreover, Taub uses scientific Linux as OS and has QDR Infiniband with bandwidth of 40 Gbps. We used physical nodes with commodity interconnect at Open Cirrus testbed at HP Labs site [11]. The next two platforms are clouds—a private cloud setup using Eucalyptus [12], and a public cloud. We use KVM [13] for virtualization since it has been shown to be a good candidate for HPC virtualization [14]. Finally, we also used an HPC-optimized cloud—Amazon EC2 Cluster Compute Cloud [7] of US West (Oregon) zone, *cc2.8xlarge* instances with Xen HVM virtualization launched in same *placement group* for best networking performance [7].

Another dedicated physical cluster at HP Labs Singapore (HPLS) is used for controlled tests of the effects of virtualization (see Table 2). This cluster is connected with a Gigabit Ethernet network on a single switch. Every server has two CPU sockets, each populated with a six-core CPU, resulting in 12 physical cores per node. The experiment on the HPLS cluster involved benchmarking on four configuration: physical machines (bare), LXC containers [15], VMs configured with the default emulated network (plain VM), and VMs with pass-through networking by enabling input/output memory management unit (IOMMU) on the Linux hosts to allow VMs to directly access the Ethernet hardware (thin VM) [16]. Both the plain VM and thin VM run atop KVM. This testbed is designed to test the isolated impact of virtualization, impossible to execute on public clouds, due to the lack of direct access to public cloud's hardware.

TABLE 2
Virtualization Testbed

Resource	Virtualization		
	Phy., Container	Thin VM	Plain VM
Processors in a Node/VM	12×Intel Xeon X5650 @2.67 GHz	12×QEMU Virtual CPU @2.67 GHz	12×QEMU Virtual CPU @2.67 GHz
Memory	120 GB	100 GB	100 GB
Network	1GigE	1GigE	Emulated 1GigE
OS	Ubuntu 11.04	Ubuntu 11.04	Ubuntu 11.04

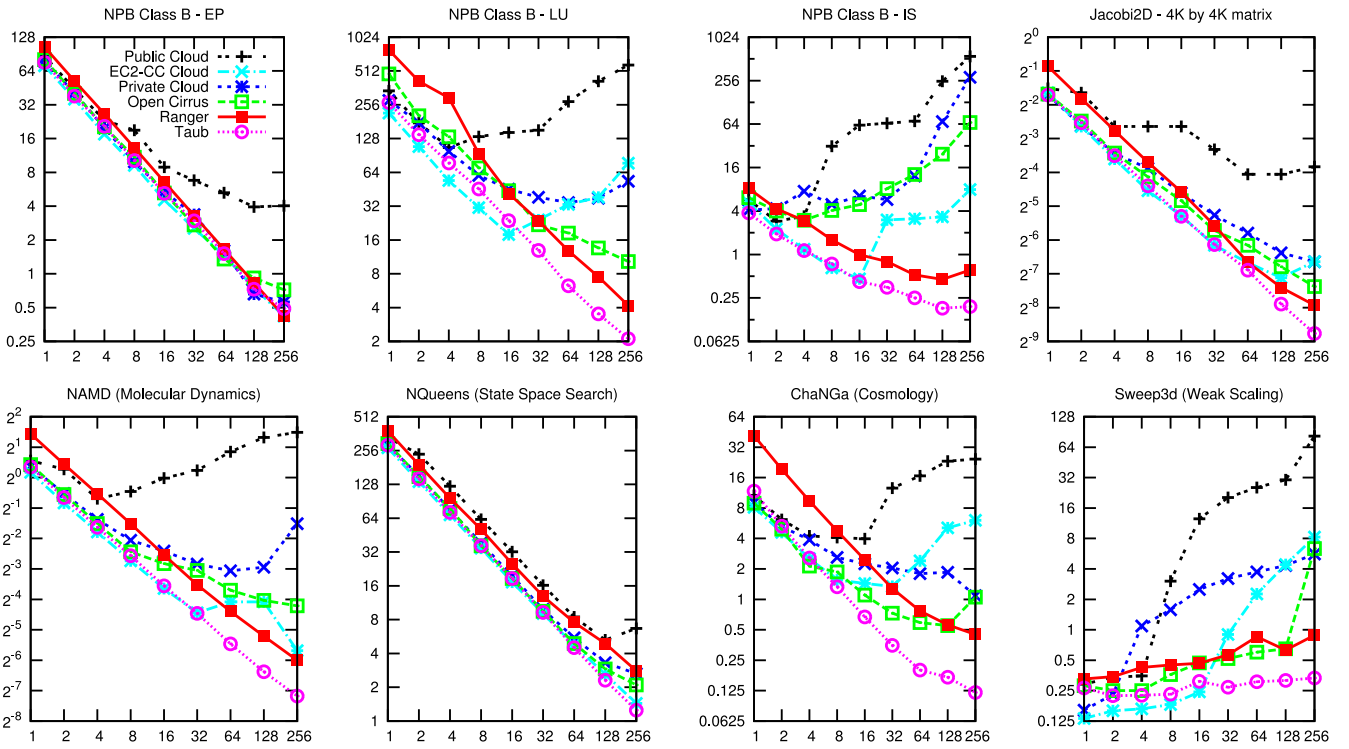


Fig. 1. Time in seconds (*y*-axis) versus core count (*x*-axis) for different applications (strong scaling except Sweep3D). All applications scale well on supercomputers and most scale moderately well on Open Cirrus. On clouds, some applications scale well (e.g., EP), some scale till a point (e.g., ChaNGa) whereas some do not scale (e.g., IS).

2.2 Benchmarks and Applications

To gain thorough insights into the performance of selected platform, we chose benchmarks and applications from different scientific domains and those which differ in the nature, amount, and pattern of inter-processor communication. Similarly to previous work [2], [3], [4], [5], we used NAS parallel benchmarks (NPB) class B [17] (the MPI version, NPB3.3-MPI), which exhibit a good variety of computation and communication requirements. Moreover, we chose additional benchmarks and real world applications, written in two different parallel programming environments – MPI [18] and CHARM++ [19]:

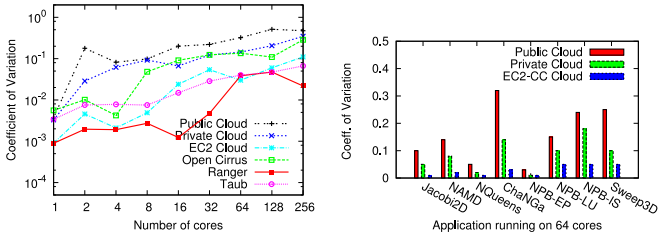
- Jacobi2D—A five-point stencil kernel, which averages values in a 2D grid, and is common in scientific simulations, numerical linear algebra, solutions of partial differential equations, and image processing.
- NAMD [20]—A highly scalable molecular dynamics application representative of a complex real world application used ubiquitously on supercomputers. We used the ApoA1 input (92k atoms).
- ChaNGa [21]—A cosmological simulation application which performs collisionless N-body interactions using Barnes-Hut tree for calculating forces. We used a 300,000 particle system.
- Sweep3D [22]—A particle transport code widely used for evaluating HPC architectures. Sweep3D exploits parallelism via a wavefront process. We ran the MPI-Fortran77 code in weak scaling mode maintaining $5 \times 5 \times 400$ cells per processor.
- NQueens—A backtracking state space search implemented as tree structured computation. The goal is to place N queens on an $N \times N$ chessboard ($N = 18$ in our runs) so that no two queens attack each other. Communication happens only for load balancing.

On Ranger and Taub, we used MVAPICH2 for MPI and CHARM++ ibverbs layer. On remaining platforms we installed Open MPI and net layer of CHARM++.

3 Benchmarking HPC Performance

Fig. 1 shows the scaling behavior of our testbeds for the selected applications. These results are averaged across multiple runs (five executions) performed at different times. We show strong scaling results for all applications except Sweep3D, where we chose to perform weak scaling runs. For NPB, we present results for only embarrassingly parallel (EP), LU solver (LU), and integer sort (IS) benchmarks due to space constraints. The first observation is the difference in sequential performance: ranger takes almost twice as long as the other platforms, primarily because of the older and slower processors. The slope of the curve shows how the applications scale on different platforms. Despite the poor sequential speed, Ranger's performance crosses Open Cirrus, private cloud and public cloud for some applications at around 32 cores, yielding a much more linearly scalable parallel performance. We investigated the reasons for better scalability of these applications on Ranger using application profiling, performance tools, and microbenchmarking and found that network performance is a dominant factor (see Section 4).

We observed three different patterns for applications on these platforms. First, some applications such as EP, Jacobi2D, and NQueens scale well on all the platforms up to 128–256 cores. The second pattern is that some applications such as LU, NAMD, and ChaNGa scale on private cloud till



(a) ChaNGa: varying core counts (b) Different Applications

Fig. 2. Performance variation.

32 cores and stop scaling afterwards. These do well on other platforms including Open Cirrus. The likely reason for this trend is the impact of virtualization on network performance (which we confirm below). On public cloud, we used VM instances with four virtual cores, hence inter-VM communication starts after four cores, resulting in sudden performance penalty above four cores. Similar performance dip can be observed for EC2-CC cloud at 16 cores where each VM had 16 cores. However, in contrast to private and public cloud, EC2-CC cloud provides good scalability to NAMD. Finally, some applications, especially the IS benchmark, perform very poorly on the clouds and Open Cirrus. Sweep3D also exhibits poor weak scaling after four-eight cores on cloud.

In case of cloud, we observed variability in the execution time across runs, which we quantified by calculating the coefficient of variation (standard deviation/mean) for runtime across five executions. Fig. 2a shows that there is significant performance variability on cloud compared to supercomputer and that the variability increases as we scale up, partially due to decrease in computational granularity. At 256 cores on public cloud, standard deviation is equal to half the mean, resulting in low run to run predictability. In contrast, EC2-CC cloud shows less variability. Also, performance variability is different for different applications (See Fig. 2b). Co-relating Figs. 1 and 2b, we can observe that the applications which scale poorly, e.g. ChaNGa and LU, are the ones which exhibit more performance variability.

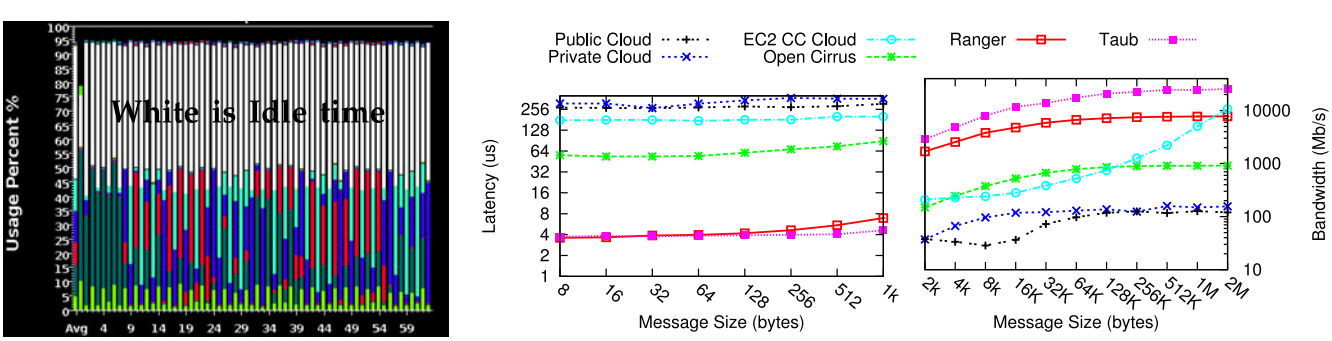
4 Performance Bottlenecks in Cloud

To investigate the reasons for the different performance trends for different applications, we obtained the applications' communication characteristics. We used tracing and visualization – Projections tool for Charm++ applications and MPE and Jumpshot [23] for MPI applications. Table 3 shows the results obtained by running on 64 cores of Taub. These numbers are cumulative across all processes. For MPI applications, we have listed the data for point-to-point and collective operations (such as `MPI_Barrier` and `MPI_AlltoAll`) separately. The numbers in parentheses correspond to collectives. It is clear from Table 3 that Jacobi2D, NQueens, and EP perform relatively small amount of communication. Moreover, we can categorize applications as latency-sensitive, i.e. large message counts with relatively small message volume, e.g. ChaNGa, or bandwidth-sensitive, i.e. large message volume with relatively small message count, e.g. NAMD, or both, e.g. Sweep3D. The point-to-point communication in IS is negligible. However, it is the only application in the set which performs heavy communication using collectives. This was validated by using Jumpshot visualization tool for MPE logs [23]. Fig. 4 shows the timeline of execution of IS during this benchmarking, with red (dark) color representing `MPI_AlltoAllv` collective communication with contributions of 2 MB by each of the 64 processors. It is evident that this operation is the dominant component of execution time for this benchmark.

TABLE 3
Application Communication Characteristics: MPI Collectives in Parentheses, Bottlenecks in Bold

Application	Message Count (K/s)	Message Volume (MB/s)
Jacobi2D	220.9	309.62
NAMD	293.3	**1705.63**
NQueens	37.4	3.99
ChaNGa	**2075.8**	308.53
NPB-EP	0 (0.2)	0 (0.01)
NPB-LU	**907.3**(0.2)	**415.31** (0.01)
NPB-IS	0.3 (9.0)	0.00 **(5632)**
Sweep3D	**2312.2** (7.2)	**2646.17** (0.02)

Juxtaposing Table 3 and Fig. 1, we can observe the correlation between the applications' communication characteristics and the performance attained, especially on cloud. To validate that communication performance is the primary bottleneck in cloud, we used Projections performance analysis tool. Fig. 3a shows the CPU utilization for a 64-core Jacobi2D experiment on private cloud, *x*-axis being the (virtual)



(a) CPU% for Jacobi2D (b) Latency and Bandwidth

Fig. 3. (a) CPU utilization for Jacobi2D on 32 two-core VMs of private cloud. White portion: idle time, colored portions: application functions. (b) Network performance on private and public clouds is off by almost two orders of magnitude compared to supercomputers. EC2-CC provides high bandwidth but poor latency.

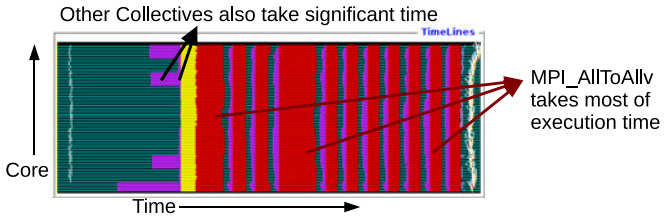


Fig. 4. Timeline of execution of IS on 64 cores on Taub. Red/dark (MPI_AlltoAllv) is the dominating factor.

core number. It is clear that CPU is under-utilized for almost half the time, as shown by the idle time (white portion) in the figure. A detailed timeline view revealed that this time was spent waiting to receive data from other processes. Similarly, for other applications which performed poorly on cloud, communication time was a considerable portion of the parallel execution time in cloud.

Since many HPC applications are highly sensitive to communication, we focused on network performance. Fig. 3b shows the results of a simple ping-pong benchmark written in Converse, the underlying substrate of CHARM++. Unsurprisingly, we found that the latencies and bandwidth on private and public clouds are a couple of orders of magnitude worse compared to Ranger and Taub, making it challenging for communication-intensive applications, such as IS, LU, and NAMD, to scale. EC2-CC cloud provides high bandwidth, enabling bandwidth-sensitive applications, such as NAMD, to scale. However, large latency results in poor performance of latency-sensitive applications (e.g. ChaNGa).

While the inferior network performance explains the large idle time in Fig. 3a, the surprising observation is the notable difference in idle time for alternating cores (0 and 1) of each VM. We traced this effect to network virtualization. The light (green) colored portion at the very bottom in Fig. 3a represents the application function which initiates inter-processor communication through socket operations, and interacts with the virtual network. The application process on core 0 of the VM shares the CPU with the network emulator. This interference increases as the application communicates more data. Hence, virtualized network degrades HPC performance in multiple ways: increases network latency, reduces bandwidth, and interferes with application process.

We also observed that even when we used only core 0 of each VM, for iterative applications containing a barrier after each iteration, there was significant idle time on some processes at random times. Communication time could not explain such random idle times. These random idle times can be attributed to the interference (*noise* or *jitter*) by other system or application processes on the same node. Quantification of this noise using a micro-benchmark showed that a fixed tiny sequential work on commodity server can have up to 100 percent variation in runtime across multiple runs (details in Section 6.2). Noise can severely degrade performance, especially for bulk-synchronous HPC applications since the slowest thread dictates the speed. Unlike supercomputers, where operating system (OS) is specifically tuned to minimize noise, e.g., Scientific Linux on Taub, clouds typically use non-tuned OS. In addition, clouds have a further intrinsic disadvantage due to the presence of the hypervisor.

Causes of performance variability: As seen in Fig. 2, there is also significant run-to-run performance variability in clouds. This can be attributed to two features intrinsic in clouds—*hardware heterogeneity* and *multi-tenancy*, that is multiple users sharing the cloud, which cause variability in the following ways. (1) Heterogeneity in physical hardware coupled with hardware-agnostic VM placement results in non-uniformity across different allocations (same total number of VMs placed on different nodes). (2) Multi-tenancy at cluster-level results in shared cluster network, which may result in network contention and *dynamic communication heterogeneity* [1]. (3) Multi-tenancy at the node-level (sometimes even core-level) results in *dynamic compute heterogeneity* and also degrades performance due to sharing of resources (such as cache, memory, disk and network bandwidth, CPU) in a multi-core node with other users' VM placed on the same node. Like most cloud environments, in our private and public clouds, physical nodes (not cores) were shared by VMs of external users, hence providing a multi-tenant environment.

5 Optimizing HPC for Cloud

In Section 4, we found that the poor cloud network performance is a major bottleneck for HPC. Hence, to achieve good performance in cloud, it is imperative to either adapt HPC runtime and applications to slow cloud networks (cloud-aware HPC), or improve networking performance in cloud (HPC-aware clouds). Next, we explore the former approach, that is making HPC cloud-aware. The latter approach is discussed in Section 6.

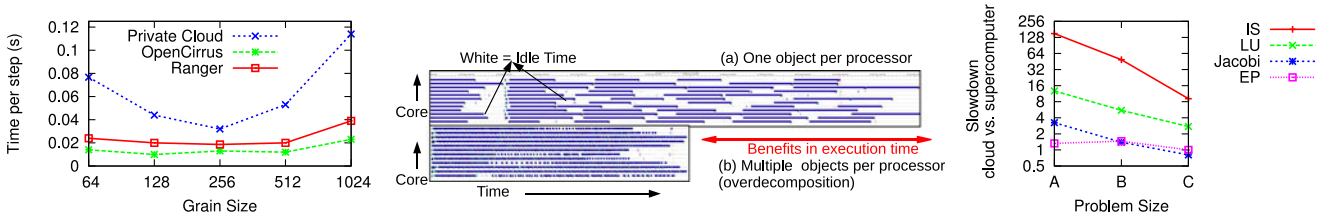
5.1 Computational Granularity/Grain Size

One way to minimize the sensitivity to network performance is to hide network latencies by overlapping computation and communication. A promising direction is asynchronous object/thread-centric execution rather than MPI-style processor-centric approach.

When there are multiple medium-grained work/data units (objects/tasks) per processors (referred to as *over-decomposition*), and an object needs to wait for a message, control can be asynchronously transferred to another object which has a message to process. Such scheduling keeps the processor utilized and results in automatic overlap between computation and communication. Our hypothesis is that overdecomposition and proper grainsize control can be crucial in clouds with slow networks.

To validate our hypothesis, we analyze the effect of the CHARM++ object grain size (or decomposition block size) on execution time of Jacobi2D on 32 cores of different platforms (Fig. 5a). Fig. 5a shows that the variation in execution time with grain size is significantly more for private cloud as compared to other platforms. As we decrease the grain size, hence increasing number of objects per processor, execution time decreases due to increased overlap of communication and computation. However, after a threshold execution time increases. This trend results from the tradeoff between the speedup due to the overlap and the slowdown due to parallel runtime's overhead of managing large number of objects.

We used Projections tool to visualize the achieved benefit. Fig. 5b shows the timelines of 12 processes of Jacobi2D execution with and without over-decomposition. Blue represent application functions whereas white represents idle time. There is lot less idling in Fig. 5b resulting in reduced overall execution time.



(a) Jacobi2d ($4k \times 4k$ grid): Grain size tuning improves performance in cloud.

(b) Timelines: Benefits of over-decomposition

(c) Slowdown on cloud vs. supercomputer reduces with increasing problem size.

Fig. 5. Optimizing HPC for cloud: Effect of grain size and problem size.

5.2 Problem Sizes and Runtime Retuning

Fig. 6 shows the effect of problem size on performance (speedup) of different applications on private cloud and supercomputer (Taub). With increasing problem sizes (A→B→C), applications scale better, and the gap between cloud and supercomputer reduces. Fig. 5c reaffirms the positive impact of problem size. For Jacobi, we denote class A as 1k × 1k, class B as 4k × 4k, and class C as 16k × 16k grid size. As problem size increases (say by a factor of X) with *fixed* number of processors, for most scalable HPC applications, the increase in communication (e.g. $\theta(X)$ for Jacobi2D) is less than the increase in computation ($\theta(X^2)$ for Jacobi2D). Hence, the communication to computation ratio decreases with increase in problem size, which results in reduced performance penalty of execution on a platform with poor interconnect. Thus, adequately large problem sizes such that the communication to computation ratio is adequately small can be run more effectively in cloud. Furthermore, applying our cost analysis methodology (Section 7), Fig. 5c can be used to estimate the cross-over points of the problem size where it would be cost-effective to run on supercomputer versus cloud.

While performing experiments, we learned that parallel runtime systems have been tuned to exploit fast HPC networks. For best performance on cloud, some of the network parameters need to be re-tuned for commodity cloud network. E.g., in case of CHARM++, increasing the maximum datagram size from 1,400 to 9,000, reducing the windows size from 32 to 8, and increasing the acknowledgement delay from 5 to 18 ms resulted in 10-50 percent performance improvements for our applications.

6 Optimizing Cloud for HPC

Cloud-aware HPC execution reduces the penalty caused by the underlying slow physical network in clouds, but it does not address the overhead of network virtualization. Next, we explore optimizations to mitigate the virtualization overhead, hence making clouds HPC-aware.

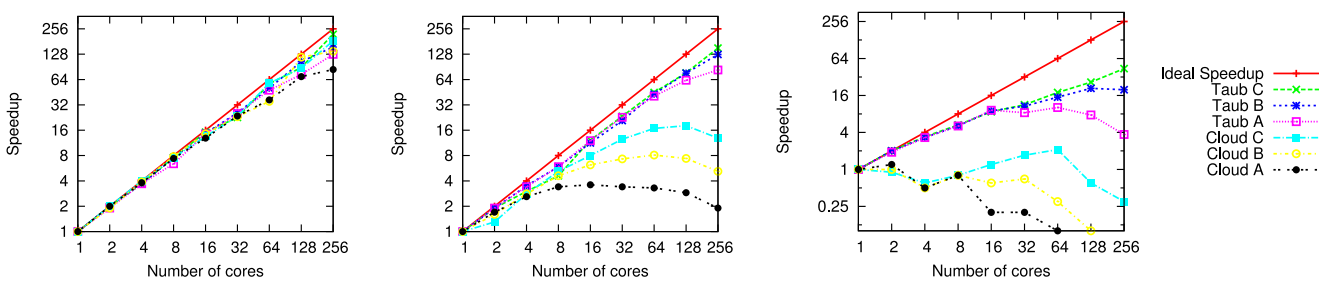
6.1 Lightweight Virtualization

We consider two lightweight virtualization techniques, *thin VMs* configured with PCI pass-through for I/O, and *containers*, that is OS-level virtualization. Lightweight virtualization reduces the overhead of network virtualization by granting VMs native accesses to physical network interfaces. Using thin VM with IOMMU, a physical network interface is allocated exclusively to a VM, preventing the interface to be shared by the sibling VMs and the hypervisor. This may lead to under-utilization when the thin VM generates insufficient network load. Containers such as LXC [15] share the physical network interface with its sibling containers and its host. However, containers must run the same OS as their underlying host. Thus, there is a trade-off between resource multiplexing and flexibility offered by VM.

Table 4 first five columns, validate that network virtualization is the primary bottleneck of cloud. These experiments were conducted on the virtualization testbed described earlier (Table 2). Plain VM attains poor scaling, but on thin VM, NAMD execution times closely track those on the physical machine even as multiple nodes are used (i.e., 16 cores onwards). The performance trend of containers also resembles that of physical machine. This demonstrates that thin VM and containers significantly lower the communication overhead. This low overhead was further validated by the ping-pong test.

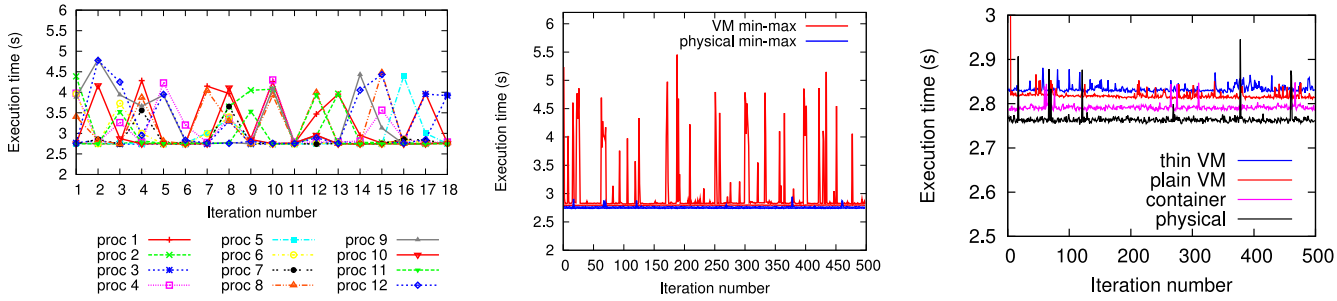
6.2 Impact of CPU Affinity

CPU affinity instructs the OS to bind a process (or thread) to a specific CPU core. This prevents the OS from inadvertently migrating a process. If all important processes have non-overlapping affinity, it practically prevents multiple processes from sharing a core. In addition, cache locality can be improved by processes remaining on the same core



(a) NPB EP (b) NPB LU (c) NPB IS

Fig. 6. Effect of problem size class on attained speedup on supercomputer (Taub) versus private cloud.



(a) Physical Machine (No CPU Affinity)

(b) Thin VM (Application-level Affinity only) and Physical Machine (CPU Affinity)

(c) All Configurations (CPU Affinity)

Fig. 7. Impact of CPU affinity on CPU performance.

throughout their execution. In the cloud, CPU affinity can be enforced at the *application level*, which refers to binding processes to the virtual CPUs of a VM, and at the *hypervisor level*, which refers to binding virtual CPUs to physical CPUs.

Fig. 7 presents the results of our micro-benchmarks with various CPU affinity settings on different types of virtual environments. In this experiment, we executed 12 processes on a single 12-core virtual or physical machine. Each process runs 500 iterations, where each iteration executes 200 millions of $y = y + rand()/c$ operations, with c being a constant. Without CPU affinity (Fig. 7a), we observe wide fluctuation on the process execution times, up to twice the minimum execution time (i.e., 2.7 s). This clearly demonstrates that frequently two or more of our benchmark processes are scheduled to the same core. The impact of CPU affinity is even more profound on virtual machines. Fig. 7b shows the minimum and maximum execution times of the 12 processes with CPU affinity enabled on the physical machine, while only application-level affinity (appAFF) is enabled on the thin VM. We observe that the gap between minimum and maximum execution times is narrowed in this case. However, on the thin VM, we still notice the frequent spikes, which is attributed to the absence of hypervisor-level affinity (HyperAFF). Even though each process is pinned to a specific virtual core, multiple virtual cores may still be mapped to the same physical core. With hypervisor-level affinity enabled, execution times across virtual cores stabilize close to those of the physical machine (Fig. 7c).

In conducting these experiments, we also learned some lessons. First, virtualization introduces a small amount of computation overhead – execution times on containers, thin VM, and plain VM are higher by 1–5 percent (Fig. 7c). Second, for best performance, it is crucial to minimize I/O operations unrelated to applications. Even on the physical machine, execution time increased by 3–5 percent due to disk I/O generated by the launcher shell script and its `stdout/stderr` redirection. The spikes on the physical machine in Fig. 7c are caused by short `ssh` sessions which simulate the scenarios where users log in to check the job progress. Thus, minimizing unrelated I/O is an important issue for HPC cloud providers.

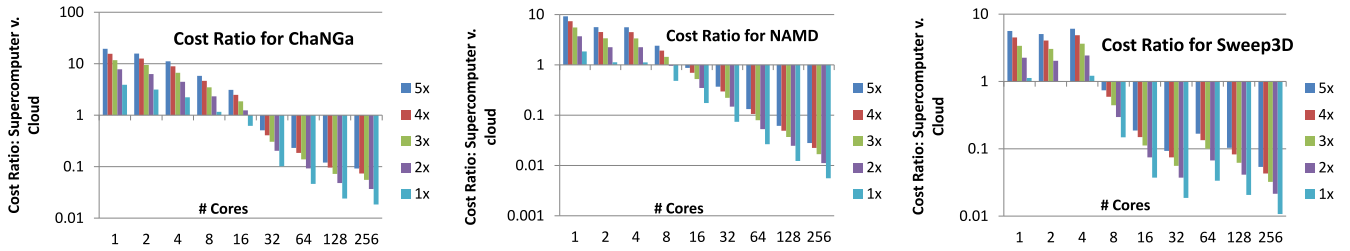
Table 4 also shows the positive impact of enabling hypervisor-level affinity , application-level affinity, or both (dualAFF). Significant benefits are obtained for thin-VM dualAFF case compared to the case with no affinity.

6.3 Network Link Aggregation

Even though network virtualization cannot improve network performance, an approach to reduce network latency using commodity Ethernet hardware is to implement link aggregation and a better network topology. Experiments from [24] show that using four-six aggregated Ethernet links in a torus topology can provide up to 650 percent improvement in overall HPC performance. This would allow cloud infrastructure using commodity hardware to improve raw network performance. Software defined networking (SDN) based on open standards such as Openflow, or similar concepts embedded in the cloud software stack, can be used to orchestrate the link aggregation and Vlan isolation necessary to achieve such complex network topologies on an on-demand basis. The use of SDN for controlling link aggregation is applicable to both bare-metal and virtualized compute instances. However, in a virtualized environment, SDN can be integrated into network virtualization to provide link aggregation to VM transparently.

TABLE 4
Impact of Virtualization and CPU Affinity Settings on NAMD's Performance

Cores	Execution Timeper step (s) of NAMD for specific virtualization and affinity setting											
	bare	container	plain-VM	thin-VM	bare-appAFF	container-appAFF	plainVM-hyperAFF	plainVM-dualAFF	plainVM-appAFF	thinVM-hyperAFF	thinVM-dualAFF	thinVM-appAFF
1	1.479	1.473	1.590	1.586	1.460	1.477	1.584	1.486	1.500	1.630	1.490	1.586
2	0.744	0.756	0.823	0.823	0.755	0.752	0.823	0.785	0.789	0.859	0.854	0.823
4	0.385	0.388	0.428	0.469	0.388	0.385	0.469	0.422	0.429	0.450	0.449	0.469
8	0.230	0.208	0.231	0.355	0.202	0.203	0.355	0.226	0.228	0.354	0.244	0.355
16	0.259	0.267	0.206	0.160	0.168	0.197	0.227	0.166	0.189	0.186	0.122	0.160
32	0.115	0.140	0.174	0.108	0.079	0.082	0.164	0.141	0.154	0.106	0.079	0.108
64	0.088	0.116	0.166	0.090	0.079	0.071	0.150	0.184	0.195	0.089	0.066	0.090
128	0.067	0.088	0.145	0.077	0.062	0.056	0.128	0.154	0.166	0.074	0.051	0.077



(a) ChaNGa (b) NAMD (c) Sweep3D

Fig. 8. Cost ratio of running in cloud and a dedicated supercomputer for different scale (cores) and cost ratios (1x–5x). Ratio>1 imply savings of running in the cloud, <1 favor supercomputer execution.

7 HPC Economics in the Cloud

There are several reasons why many commercial and web applications are migrating to public clouds from fully owned resources or private clouds: variable usage in time resulting in lower utilization, trading capital expenditure (CAPEX) for operating expenditure (OPEX), and the shift towards a delivery model of Software as a Service. These arguments apply both to cloud providers and cloud users. Cloud users benefit from running in the cloud when their applications fit the profile we described e.g., variable utilization. Cloud providers benefit if the aggregated resource utilization of all their tenants can sustain a profitable pricing model when compared to the substantial upfront investments required to offer computing and storage resources through a cloud interface.

Why not cloud for HPC: HPC is however quite different from the typical web and service-based applications. (1) Utilization of the computing resources is typically quite high on HPC systems. This conflicts with the desirable property of low average utilization that makes the cloud business model viable. (2) Clouds achieve improved utilization through consolidation enabled by virtualization—a foundational technology for the cloud. However, as evident from our analysis, the overhead and noise caused by virtualization and multi-tenancy can significantly affect HPC applications' performance and scalability. For a cloud provider that means that the multi-tenancy opportunities are limited and the pricing has to be increased to be able to profitably rent a dedicated computing resource to a single tenant. (3) Many HPC applications rely on optimized interconnect hardware to attain best performance, as shown by our experimental evaluation. This is in contrast with the commodity Ethernet network (1Gbps today moving to 10 Gbps) typically deployed in most cloud infrastructures to keep costs small. When networking performance is important, we quickly reach diminishing returns of scaling-out a cloud deployment to meet a certain performance target. If too many VMs are required to meet performance, the cloud deployment quickly becomes uneconomical. (4) The CAPEX/OPEX argument is less clear for HPC users. Publicly funded supercomputing centers typically have CAPEX in the form of grants, and OPEX budgets may actually be tighter and almost fully consumed by the support and administration of the supercomputer with little headroom for cloud bursting. (5) Software-as-a-service offering are also rare in HPC to date.

Why cloud for HPC: So, what are the conditions that can make HPC in the cloud a viable business model for both, HPC users and cloud providers? Unlike large supercomputing centers, HPC users in small-medium enterprises are much more sensitive to the CAPEX/OPEX argument. These include startups with nascent HPC requirements (e.g., simulation or modeling) and small-medium enterprises with growing business and an existing HPC infrastructure. Both of them may prefer the pay-as-you-go approach in clouds versus establishing/growing on-premise resources in volatile markets. Moreover, the ability to leverage a large variety of heterogeneous architectures in clouds can result in better utilization at global scale, compared to the limited choices available in any individual organization. Running applications on the most economical architecture while meeting the performance needs can result in savings for consumers.

7.1 Quantifiable Analysis

To illustrate a few possible HPC-in-the-cloud scenarios, we collected and compared cost and price data of supercomputer installations and typical cloud offerings. Based on our survey of cloud prices, known financial situations of cloud operators, published supercomputing costs, and a variety of internal and external data sources [25], we estimate that a cost ratio between 2x and 3x is a reasonable approximate range capturing the differences between a cloud deployment and on-premise supercomputing resources today. In our terminology, 2x indicates the case where one supercomputer core-hour is twice as expensive as one cloud core-hour. Since these values can fluctuate, we expand the range to [1x–5x] to capture different future, possibly unforeseen scenarios.

Using the performance evaluations for different applications (Fig. 1), we calculated the cost differences of running the application in the public cloud versus running it in a dedicated supercomputer (Ranger), assuming different per core-hour cost ratios from 1x to 5x. Fig. 8 shows the cost differences for three applications, where values>1 indicate savings of running in the cloud and values<1 an advantage of running it on a dedicated supercomputer. We can see that for each application there is a scale in terms of the number of cores up to which it is more cost-effective to execute in the cloud versus on a supercomputer. For example, for Sweep3D, NAMD, and ChaNGa, this scale is higher than 4, 8, and 16 cores respectively. This break-even point is a function of the application scalability and the cost ratio. However our observation is that there is little sensitivity to the cost ratio and it is relatively

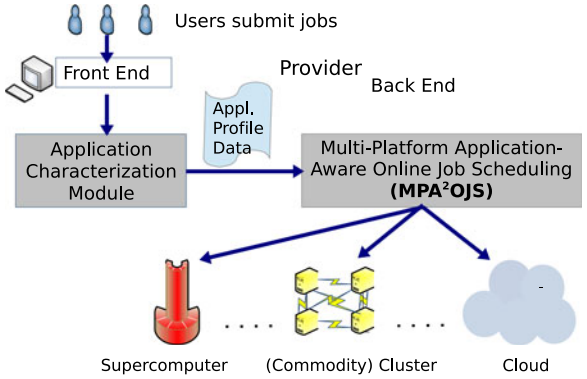


Fig. 9. Application-aware scheduling on platforms with different processors/VMs and interconnections.

straightforward to determine the break-even point. This is true even for the cost ratio of 1. This might be the artifact of slower processors for the Ranger versus newer and faster processors in the cloud.

7.2 Qualitative Discussion

Next, we address few economic topics on HPC in cloud.

We have primarily compared supercomputers versus HPC in clouds. However, there are other alternatives that we discuss in Section 8, such as bursting out to cloud. Yet another alternative is outsourcing supercomputer. In many ways, we consider the latter similar to HPC in the cloud, with the exception of the way of use. Supercomputers are batch oriented while clouds offer dedicated use, at least at the virtualization level. There is also no reason why someone would not put a whole supercomputer behind cloud interfaces, and make it available on demand. Hence, these are variations of the key cases discussed in the paper. At the same time, current supercomputers are almost fully utilized, so there is little incentive to benefit from on-demand use of supercomputers, as compared to departmental level of servers, e.g. in computer aided design (CAD) or computer aided engineering (CAE) which can largely benefit from improved utilization.

In addition, cloud providers can offer the most recent equipment. Because they will share it among many customers they can amortize the high cost more easily than any single customer. This equipment can be used for exploration or in a production manner for early adopters. Movie rendering is a classic case of a cloud HPC (compute-intensive) application. Most recent case is post-production of the film "Life of Pi". Movie companies can always use the most recent equipment in the cloud and eventually acquire those that benefits them.

In this paper, we have not discussed accelerators, such as GPUs, which are becoming important for the HPC and compute-intensive applications. Because we have not conducted any experiments with GPUs, we cannot elaborate with any substance on the implications of the use of GPUs in the cloud. However, there is no reason not to treat them the same way as the regular compute instances. For example, Amazon prices them in the similar dedicated instances class with the price of $0.715 for GPU (g2.2xlarge) instance versus similarly sized (c3.2xlarge) compute instance for $0.462 per hour. The price difference is attributed to the hardware cost, the number of instances offered (many more compute than GPUs), and the power consumed.

One additional complication arising from the use of accelerators is that they do not virtualize well. While there is ongoing work making good progress in that direction, like NVidia GRID [26], it is still a young area with several unresolved issues. For example, the current sharing model of virtual GPUs is appropriate for concurrent execution of multiple jobs in a dedicated supercomputer, but does not provide the encapsulation, protection, and security support that would make it appropriate in the cloud. Any resource that does not virtualize at fine granularity poses a serious challenge to the cloud adoption model because it forces the cloud provider to adopt a very rigid pricing scheme if the resource cannot be sliced for multiple concurrent users. We believe this is an interesting area for future research. Finally, we would like to conclude that it is was non-trivial to do a fair comparison of HPC in the cloud and supercomputers. For clouds, the prices are well documented and they are public information, however the costs are undocumented and they are proprietary and really a differentiator for each cloud provider. On contrary, the costs for supercomputers are well documented by owners while the prices are typically hidden and not publicized due to subsidies and price reduction. That was one of the primary reasons why we used range of cost ratios in Section 7.3 (Fig. 8). Hence, the economic comparison needs to be taken conservatively.

8 Scheduling HPC Jobs in the Cloud

In the previous sections, we provided empirical evidence that applications behave quite differently on different platforms. This observation opens up several opportunities to optimize the mapping and scheduling of HPC jobs to platforms. In our terminology, *mapping* refers to selecting a platform for a job, and *scheduling* includes mapping and deciding when to execute the job on the chosen platform. Here, we research techniques to perform intelligent scheduling and evaluate the benefits.

8.1 Problem Definition

Consider the case when the dedicated HPC infrastructure cannot meet peak demands and the provider is considering offloading jobs to additional available cluster or cloud. The problem can be defined as follows. Given a set of owned platforms with resources having different processor configurations, interconnection networks, and degrees of virtualization, how can we effectively schedule an incoming stream of HPC jobs to these platforms based on intrinsic application characteristics, job requirements, platform capabilities, and dynamic demand and load fluctuations to achieve the goals of improved job completion time, makespan, and hence throughput.

8.2 Methodology

To address this problem, we adopt a two-step methodology, shown in Fig. 9: 1) perform a one-time offline benchmarking or analytical modeling of applications-to-platforms performance sensitivity, and 2) use heuristics to schedule the application stream to available platforms based on the output of step 1 and current resource availabilities. In this

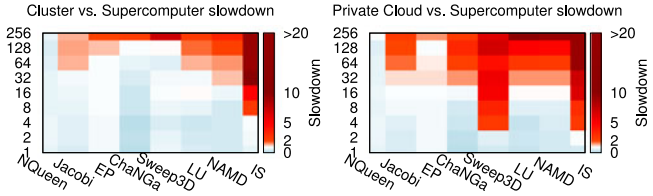


Fig. 10. Effect of application and scale on slowdown. Light: no slowdown, dark (red): more slowdown.

paper, our focus is on step 2, which translates to an online job scheduling problem with the additional complexity of having to decide *which platform* a job should be run on besides the decision regarding *which job* to execute next. The problem is even more challenging since different application react differently to different platforms. We will refer to this as *Multi-Platform Application-Aware Online Job Scheduling (MPA^2OJS)*.

For step 1, we rely on one-time benchmarking of application performance on different platforms to drive the scheduling decisions. In our earlier work, we have shown that in the absence of the benchmark data, it is possible to perform application characterization followed by relative performance prediction when considering multiple platforms [27]. Also, other known techniques for performance prediction can be used. These include analytical modeling, simulations, application profiling through sample execution (e.g. the first few iterations) on actual platform, and interpolation. It is not our intention in this paper to research accurate techniques for parallel performance prediction of complex applications. Our goal is to quantify the benefits of MPA^2OJS to develop an understanding and foundation for HPC in cloud, which can promote further research towards additional characterization and scheduling techniques.

Traditional HPC job scheduling algorithms do not consider the presence of multiple platforms. Hence, they are agnostic of the application to platform performance sensitivity. In MPA^2OJS, the mapping decision could be *static* or *dynamic*. Static decisions are independent of current platform load and made a-priori to job scheduling, whereas dynamic decisions are aware of the current resource availability and load and they are made when job is scheduled. With dynamic mapping, the same job can be scheduled to run on different platforms across its multiple executions depending upon the state of the system when it was scheduled.

Hence, MPA^2OJS algorithms can be classified as static versus dynamic, or job-characteristics aware versus unaware. Next, we present some heuristics for MPA^2OJS.

8.2.1 Static Mapping Heuristics

The analysis in Section 4 showed that the slowdown in cloud versus supercomputer depends on the application under consideration. Also, for the same application, the sensitivity to a platform varies with scale, i.e., with core counts. To visualize the behavior of HPC jobs along these two dimensions, i.e., application type and scale, we used our performance data to generate the map of a job's slowdown when running on the commodity cluster, i.e. Open Cirrus (Fig. 10a) and private cloud (Fig. 10b) with respect to its execution on supercomputer (Ranger).

In Fig. 10, each grid cell represents the execution of a particular application (x-axis) at a particular scale (number of cores on y-axis). Here, light colors (blue and white) represent that job suffered no slowdown. In some cases, job attained speedup due to worse sequential performance on Ranger compared to the other platforms. Dark (reddish) shades represent slowdown. Based on Fig. 10, two possible heuristics for static mapping are:

- *ScalePartition*: Assign large scale jobs (say 64-256 cores) to supercomputer, medium scale (16–32 cores) to cluster, and small scale (1-8 cores) to cloud by partitioning the slowdown map along y-dimension.
- *ApplicationPartition*: Assign specific applications to specific platforms by partitioning the map along x-dimension. E.g. IS, NAMD, and LU to supercomputer, ChaNGa and Sweep3D to cluster, and EP, Jacobi, and NQueens to cloud. A variation of *ApplicationPartition* can be to use finer application characteristics such as message count and volume for partitioning (Table 3, Section 4).

Other examples of static policies include scheduling all jobs to a supercomputer *(SCOnly)*, to a cluster *(ClusterOnly)*, or to a cloud *(CloudOnly)*.

8.2.2 Dynamic Mapping Heuristics

The motivation for dynamic selection of a platform for a job is to perform resource availability driven scheduling. Some such heuristics that we explored are:

- *MostFreeFirst*: Assign job to least loaded platform.
- *RoundRobin*: Assign jobs to platforms round-robin.
- *BestFirst*: Assign the current job to platform with best available resources. E.g. in the order supercomputer, cluster, and cloud.
- *Adaptive*: Assign the job to the platform with largest *Effective Job-specific Supply (EJS)*.

EJS is defined to capture both, current resource availability and a job's suitability to a particular platform. We define a platform's *EJS* for a particular job as the product of free cores on that platform and the job's normalized performance obtained on that platform. The intuition is that the core-hours taken for a job to complete on a platform are directly proportional to the slowdown it suffers on that platform compared to the supercomputer. Hence, the Adaptive heuristic optimizes along two dimensions: it balances load across multiple platforms and it matches application characteristics to platforms. In contrast, the first three dynamic heuristics are application-agnostic.

Table 5 classifies our heuristics into static versus dynamic, and application-aware versus application-agnostic.

8.3 Implementation and Evaluation Using CloudSim

We implemented the MPA^2OJS heuristics in CloudSim [28], which is a widely used tool for simulation of scheduling algorithms in a data center or a cloud. We modified CloudSim to enable simulation of HPC job scheduling across multiple platforms.

In CloudSim, a fixed number of VMs are created at the start of simulation, and jobs (cloudlets in CloudSim terminology) can be submitted to these VMs. For our simulation

TABLE 5
Classification of Heuristics for MPA^2OJS

Heuristics	Dynamic	App-aware
SCOnly, ClusterOnly, CloudOnly		
ScalePartition, ApplicationPartition		Yes
RoundRobin, BestFirst, MostFreeFirst	Yes	
Adaptive	Yes	Yes

purpose, a one-to-one mapping of cloudlets to VMs is sufficient but we needed to provide dynamic VM creation and termination. Also, we extended existing VM allocation policy in CloudSim to enable first come first serve (FCFS) scheduling of HPC jobs to resources.

The scheduling of cloudlets is performed by the datacenter broker. Hence, we created a new datacenter broker to perform MPA^2OJS. We introduced a periodic event in CloudSim, which checks for new job arrivals. The scheduler is triggered when a new job arrives or when a running job completes. Based on the current state of available datacenters and the scheduling heuristic, new jobs are assigned to a specific datacenter queue. Internally within a datacenter, FCFS policy is honored.

8.3.1 Simulation Approach

For our simulation, we created three datacenters—supercomputer, cluster, and cloud (256 cores each). These correspond to Ranger, Open Cirrus (typical commodity cluster), and private cloud (typical hypervisor-based resources) respectively. We simulated the execution of first 1000 jobs corresponding to the METACENTRUM-02.swf job logs of parallel workload archive [29]. Each job record contains the job's arrival time, requested number of cores (P), and its runtime. However, our goal is to simulate multiple platforms, where the runtime will vary from one platform to the other. Hence, we used a uniform distribution random number generator to map each job to one of the applications from the set evaluated in this paper. We modified the job records to contain the application name (AppName) and the normalized performance for various platforms corresponding to (AppName,P). We used the same seed for random number generator while comparing different heuristics.

Furthermore, to evaluate how our heuristics perform under varying system load, we modified the runtimes of jobs in the log file. *Medium* load represents the original runtimes, *low* load represent runtimes scaled down by 2X, and *high* load represent runtimes scaled up by 2X.

8.3.2 Results: Makespan and Throughput

Using simulation we found that most application-agnostic strategies of Table 5, specifically ClusterOnly, CloudOnly, RoundRobin, and MostFreeFirst, performed very poorly compared to other heuristics. This is attributed to the tremendous slowdown that some application suffer when running on cloud versus their execution on supercomputer (e.g. up to 400X for IS). Hence, in this paper, we present and analyze results of the remaining five heuristics, which yield more reasonable solutions.

Fig. 11 compares the makespan (total completion time) and throughput for different heuristics under varying system load. It is evident from Fig. 11 that Adaptive heuristic outperforms the rest when the system is reasonably loaded (medium and high load). Moreover, the benefits increase as the system load increases. Adaptive heuristic results in around 1.05X, 1.6X, and 1.8X improvement in makespan at low, medium, and high load respectively compared to SCOnly, that is running all applications on supercomputer. Similarly, there is significant improvement in throughput (number of completed jobs per second). For instance, after 1 hour of execution (3,600 s), Adaptive strategy attains 1.25X, 4X, and 6X better throughput compared to SCOnly under different system loads. The benefits are even higher compared to other application-agnostic strategies, such as RoundRobin, as mentioned earlier. AppPartition performs well under low and medium load but yields poor results under high load.

For better understanding of the reasons for the benefits and sensitivity to load, we measured various other metrics. A potential cause of the benefits is the improvement in average response time (job's start time—arrival time). Adaptive, ScalePartition, and AppPartition achieved the most benefits in terms of response time. In some cases, AppPartition (at low and medium load) or ScalePartition (at low and high load) achieve even better response time compared to Adaptive. However, from Fig. 11, we saw that overall Adaptive performed significantly better at medium and high load. This is because Adaptive performs the best in terms of average runtime in all three cases (loads) since ScalePartition and AppPartition are static mapping schemes.

Static heuristics cannot dynamically change the mapping of a job even if a better platform is available. Hence, high-end resources may be left unused waiting for a matching application to arrive. Also, on further investigation, we learned that 1D characterization may not be sufficient since that can still result in some suboptimal mappings, e.g. ScalePartition

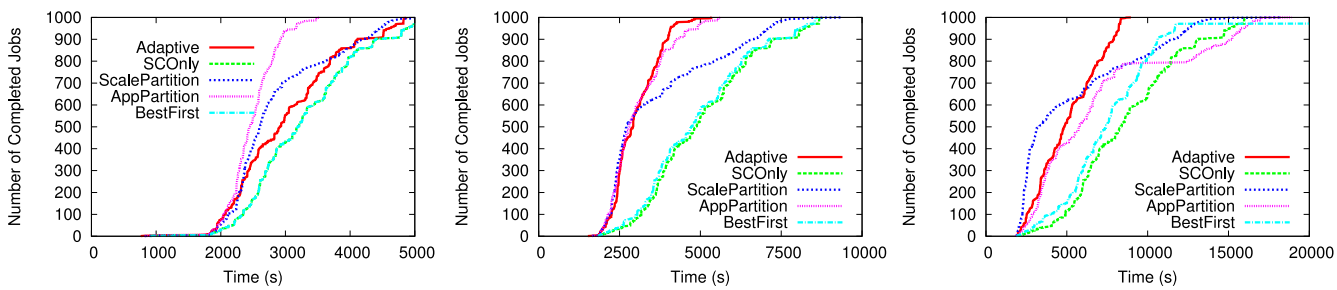


Fig. 11. Adaptive heuristic significantly improves makespan and throughput when system is reasonably loaded.

TABLE 6
Findings and Our Approach to Address the Research Questions on HPC in Cloud

Question	Answers
Who	(1) Small and medium scale organizations, startups or growing businesses, which can benefits from pay-as-you-go model. (2) Users with applications which result in best performance/cost ratio in cloud versus other platforms.
What	(1) Applications with less-intensive communication patterns and less sensitivity to interference. (2) Applications with performance needs that can be met at small to medium scale execution (in terms of number of cores).
Why	(1) Small-medium enterprises benefit from pay-as-you-go model since they are highly sensitive to CAPEX/OPEX argument. (2) Clouds enables multiple organizations to access a large variety of shared architectures, leading to improved utilization.
How	(1) Technical approaches: (a) *Making HPC cloud-aware* e.g. tuning computational granularity and problem sizes, and (b) *making clouds HPC-aware* e.g. providing lightweight virtualization and enabling CPU affinity. (2) Business models: Hybrid supercomputer–cloud approach with application-aware scheduling and cloud bursting.

maps IS at 32 cores to cluster even if supercomputer is free. For benefitting from multiple platforms, we need to a) consider both, the application characteristics and the scale at which it will be run, and b) dynamically adapt to the platform loads. Adaptive heuristic meets these two goals.

9 Related Work

In this section, we summarize the related research on HPC in cloud, including performance evaluation studies.

9.1 Performance and Cost Studies of HPC on Cloud

Walker [5], followed by several others [1], [2], [3], [4], [6], [8], [9], [27], [30], conducted the study on HPC in cloud using benchmarks such as NPB and real applications. Their conclusions can be summarized as:

- Primary challenges for HPC in cloud are insufficient network and I/O performance in cloud, resource heterogeneity, and unpredictable interference arising from other VMs [1], [2], [3], [4].
- Considering cost into the equation results in interesting trade-offs; execution on clouds may be more economical for some HPC applications, compared to supercomputers [4], [27], [31], [32].
- For large-scale HPC or for centers with large user base, cloud cannot compete with supercomputers based on the metric $/GFLOPS [1], [9].

In this paper, we explored some of the similar questions from the perspective of smaller scale HPC users, such as small companies and research groups who have limited access to supercomputer resources and varying demand over time. We also considered the perspective of cloud providers who want to expand their offerings to cover the aggregate of these smaller scale HPC users.

Furthermore, our work explored additional dimensions: (1) With a holistic viewpoint, we considered all the different aspects of running in cloud-performance, cost, and business models, and (2) we explored techniques for bridging the gap between HPC and clouds. We improved HPC performance in cloud by (a) improving execution time of HPC in cloud and (b) by improving the turnaround time with intelligent scheduling in cloud.

9.2 Bridging the Gap between HPC and Cloud

The approaches towards reducing the gap between traditional cloud offerings and HPC demands can be classified into two categories—(1) those which make clouds HPC-aware, and (2) those which makes HPC clouds-aware. In this paper, we presented techniques for both. For (1), We explored techniques in low-overhead virtualization, and quantified how close we can get to physical machine's performance for HPC workloads. There are other recent efforts on HPC-optimized hypervisors [33], [34]. Other examples of (1) include HPC-optimized clouds such as Amazon Cluster Compute [7] and DoE's Magellan [1] and hardware- and HPC-aware cloud schedulers (VM placement algorithms) [35], [36].

The latter approach (2) has been relatively less explored, but has shown tremendous promise. Cloud-aware load balancers for HPC applications [37] and topology aware deployment of scientific applications in cloud [38] have shown encouraging results. In this paper, we demonstrated how we can tune the HPC runtime and applications to clouds to achieve improved performance.

9.3 HPC Characterization, Mapping, and Scheduling

There are several tools for scheduling HPC jobs on clusters, such as ALPS, OpenPBS, SLURM, TORQUE, and Condor. They are all job schedulers or resource management systems which aim to utilize system resources in an efficient manner. They differ from our work on scheduling since we perform application-aware scheduling and provide solution for multi-platform case. GrADS [39] project addressed the problem of scheduling, monitoring, and adapting applications to heterogeneous and dynamic grid environment. Our focus is on clouds and hence we address additional challenges such as virtualization, cost and pricing models for HPC in cloud.

Kim et al. [40] presented three usage models for hybrid HPC grid and cloud computing: acceleration, conservation, and resilience. However, they use cloud for sequential tasks and do not consider execution of parallel applications. Inspired by that work, we evaluate models for HPC-clouds: substitute, complement, and burst.

10 Conclusions, Lessons, Future Work

Through a performance, economic, and scheduling analysis of HPC applications on a range of platforms, we have

shown that different applications exhibit different characteristics that determine their suitability towards a cloud environment. Table 6 presents our conclusions. Next, we summarize the lessons learned from this research and the emerging future research directions.

Clouds can successfully complement supercomputers, but using clouds to substitute supercomputers is infeasible. Bursting to cloud is also promising. We have shown that by performing multi-platform dynamic application-aware scheduling, a hybrid cloud-supercomputer platform environment can actually outperform its individual constituents. By using an underutilized resource which is "good enough" to get the job done sooner, it is possible to get better turnaround time for job (user perspective) and improved throughput (provider perspective). Another potential model for HPC in cloud is to use cloud only when there is high demand (cloud *burst*). Our evaluation showed that application-agnostic cloud bursting (e.g. BestFirst heuristic) is unrewarding, but application-aware bursting is a promising research direction. More work is needed to consider other factors in multi-platform scheduling: job quality of service (QoS) contracts, deadlines, priorities, and security. Also, future research is required in cloud pricing in multi-platform environments. Market mechanisms and equilibrium factors in game theory can help automate such decisions.

For efficient HPC in cloud, HPC need to be cloud-aware and clouds needs to be HPC-aware. HPC applications and runtimes must adapt to minimize the impact of slow network, heterogeneity, and multi-tenancy in clouds. Simultaneously, clouds should minimize overheads for HPC using techniques such as lightweight virtualization and link aggregation with HPC-optimized network topologies. With low-overhead virtualization, web-oriented cloud infrastructure can be reused for HPC. We envisage hybrid clouds that support both HPC and commercial workloads through tuning or VM re-provisioning.

Application characterization for analysis of the performance-cost tradeoffs for complex HPC applications is a non-trivial task, but the economic benefits are substantial. More research is necessary to quickly identify important traits for complex applications with dynamic and irregular communication patterns. A future direction is to evaluate and characterize applications with irregular parallelism [41] and dynamic datasets. For example, challenging data sets arise from 4D CT imaging, 3D moving meshes, and computational fluid dynamics (CFD). The dynamic and irregular nature of such applications makes their characterization even more challenging compared to the regular iterative scientific applications considered in this paper. However, their asynchronous nature, i.e. lack of fine-grained barrier synchronizations, makes them promising candidates for heterogeneous and multi-tenant clouds.

REFERENCES

[1] K. Yelick, S. Coghlan, B. Draney, R. S. Canon, L. Ramakrishnan, A. Scovel, I. Sakrejda, A. Liu, S. Campbell, P. T. Zbiegiel, T. Declerck, P. Rich, "The magellan report on cloud computing for science", U. S. Department of Energy Office of Science, Office of Advanced Scientific Computing Research (ASCR), Dec. 2011.

[2] P. Mehrotra, J. Djomehri, S. Heistand, R. Hood, H. Jin, A. Lazanoff, S. Saini, and R. Biswas, "Performance evaluation of Amazon EC2 for NASA HPC applications," in *Proc. 3rd Workshop Scientific Cloud Comput.* 2012, pp. 41–50.

[3] C. Evangelinos and C. N. Hill, "Cloud computing for parallel scientific HPC applications: Feasibility of running coupled atmosphere-ocean climate models on Amazon's EC2," in *Proc. IEEE Cloud Comput. Appl.*, Oct. 2008, pp. 2–34.

[4] A. Gupta and D. Milojicic, "Evaluation of HPC Applications on Cloud," in *Proc. Open Cirrus Summit (Best Student Paper)*, Atlanta, GA, USA, Oct. 2011, pp. 22–26.

[5] E. Walker, "Benchmarking Amazon EC2 for high-performance scientific computing," *LOGIN*, vol. 33, pp. 18–23, 2008.

[6] A. Gupta, L. V. Kalé, D. S. Milojicic, P. Faraboschi, R. Kaufmann, V. March, F. Gioachin, C. H. Suen, and B.-S. Lee, "The who, what, why, and how of HPC applications in the cloud," in *Proc. 5th IEEE Intl. Conf. Cloud Comp. Techno. and Sc. Best Paper*, 2013, pp. 306–314.

[7] High Performance Computing (HPC) on AWS [Online]. Available: http://aws.amazon.com/hpc-applications

[8] A. Iosup, S. Ostermann, M. N. Yigitbasi, R. Prodan, T. Fahringer, and D. H. J. Epema, "Performance Analysis of Cloud Computing Services for Many-Tasks Scientific Computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 6, pp. 931–945, Jun. 2011.

[9] J. Napper and P. Bientinesi, "Can Cloud Computing reach the Top500?' in *Proceedings of the Combined Workshops on Unconventional High Performance Computing Workshop Plus Memory Access Workshop*. New York, NY, USA: ACM, 2009.

[10] Ranger User Guide. [Online]. Available: http://services.tacc.utexas.edu/index.php/ranger-user-guide

[11] A. I. Avetisyan, R. Campbell, I. Gupta, M. T. Heath, S. Y. Ko, G. R. Ganger, M. A. Kozuch, D. O'Hallaron, M. Kunze, T. T. Kwan, and others, "Open Cirrus: A Global Cloud Computing Testbed," *Computer*, vol. 43, no. 4, pp. 35–43, Apr. 2010.

[12] D. Nurmi R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "Cluster Computing and the Grid, 2009. CCGRID'09. 9th IEEE/ACM International Symposium on," in *Proc. Cloud Comput. Appl.*, Oct. 2009, pp. 124–131.

[13] Kivity, A. Y. Kamay, D. Laor, U. Lublin, and A. Liguori, "kvm: the Linux virtual machine monitor," in *Proc. of the Linux Symposium*, vol. 1, pp. 225–230, 2007.

[14] A. J. Younge, R. Henschel, J. T. Brown, G. Von Laszewski, J. Qiu, and G. C. Fox, "Analysis of virtualization technologies for high performance computing environments," in *Proc. IEEE Int. Conf. Cloud Comput.*, 2011, pp. 9–16.

[15] D. Schauer et al, (June). Linux containers version 0.7.0. (2010) [Online]. Available: http://lxc.sourceforge.net/

[16] Intel Corporation. (Feb.) Intel(r) Virtualization Technology for Directed I/O. Tech. Rep. D51397-006, (2011) [Online]. Available: http://download.intel.com/technology/computing/vptech/Intel(r)_VT_for_D irect_IO.pdf

[17] NPB [Online]. Available: http://nas.nasa.gov/publications/npb.html

[18] "MPI: A Message Passing Interface Standard," in *MPI Forum*, 1994.

[19] L. Kalé and S. Krishnan, "CHARM++: A Portable Concurrent Object Oriented System Based on C++," in *Proc. 8th Annu. Conf. Object-Oriented Program. Syst., Languages, Appl.*, 1993, pp. 91–108.

[20] A. Bhatele, S. Kumar, C. Mei, J. C. Phillips, G. Zheng, and L. V. Kale, "Overcoming scaling challenges in biomolecular simulations across multiple platforms," in *Proc. IEEE Int. Symp. Parallel Distrib. Process.*, 2008, pp. 1–12.

[21] P. Jetley, F. Gioachin, C. Mendes, L. V. Kale, and T. R. Quinn, "Massively Parallel Cosmological Simulations with ChaNGa," in *Proc. IEEE Int. Symp. Parallel Distrib. Processing*, 2008, pp. 1–12.

[22] The ASCII Sweep3D code [Online]. Available: http://wwwc3.lanl.gov/pal/software/sweep3d

[23] O. Zaki, E. Lusk, W. Gropp, and D. Swider, "Toward scalable performance visualization with Jumpshot," *Int. J. High Perform. Comput. Appl.*, vol. 13, no. 3, pp. 277–288, Fall 1999.

[24] T. Watanabe, M. Nakao, T. Hiroyasu, T. Otsuka, and M. Koibuchi, "Impact of topology and link aggregation on a PC cluster with ethernet," in *Proc. IEEE CLUSTER*, Sep./Oct. 2008, pp. 280–285.

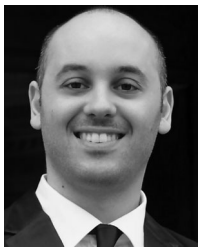
[25] C. Bischof, D. anMey, and C. Iwainsky. (2011). Brainware for Green HPC. *CS - Research and Development*, pp. 1–7 [Online]. Available: http://dx.doi.org/10.1007/s00450-011-0198-5

[26] NVIDIA GRID [Online]. Available: http://nvidia.com/object/virtual-gpus.html

[27] A. Gupta, Abhishek, Kalé, V. Laxmikant, D. S. Milojicic, P. Faraboschi, R. Kaufmann, V. March, F. Gioachin, C. H. Suen, and B. -S. Lee, "Exploring the Performance and Mapping of HPC Applications to Platforms in the cloud," in *Proc. 21st Int. Symp. High-Perform. Parallel Distrib. Comput.*, 2012, pp. 121–122.
[28] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Softw. Pract. Exper.*, vol. 41, no. 1, pp. 23–50, Jan. 2011.
[29] Parallel Workloads Archive. [Online]. Available: http://www.cs.huji.ac.il/labs/parallel/workload/
[30] J. Ekanayake, X. Qiu, T. Gunarathne, S. Beason, and G. C. Fox, "High performance parallel computing with clouds and cloud technologies," in *Cloud Computing*. New York, NY, USA: Springer, 2010.
[31] E. Roloff, M. Diener, A. Carissimi, and P. Navaux, "High performance computing in the cloud: Deployment, performance and cost efficiency," in *Proc. CloudCom*, 2012, pp. 371–378.
[32] A. Marathe, R. Harris, D. K. Lowenthal, B. R. de Supinski, B. Rountree, M. Schulz, and X. Yuan, "a comparative study of high-performance computing on the cloud," in *Proc. 21st Int. Symp. High-Performance Parallel Distrib. Comput.*, 2013, pp. 239–250.
[33] J. Lange, K. Pedretti, T. Hudson, P. Dinda, Z. Cui, L. Xia, P. Bridges, A. Gocke, S. Jaconette, M. Levenhagen, and R. Brightwell, "Palacios and Kitten: New high performance operating systems for scalable virtualized and native supercomputing," in *Proc. IEEE Int. Symp. Parallel Distrib. Process.*, 2010, pp. 1–12.
[34] B. Kocoloski, J. Ouyang, and J. Lange, "A case for dual stack virtualization: consolidating HPC and commodity applications in the cloud," in *Proc. 3rd ACM Symp. Cloud Comput.*, New York, NY, USA, 2012, pp. 23:1–23:7.
[35] HeterogeneousArchitectureScheduler. [Online]. Available: http://wiki.openstack.org/HeterogeneousArchitectureScheduler
[36] A. Gupta, L. Kale, D. Milojicic, P. Faraboschi, and S. Balle, "HPC-Aware VM Placement in Infrastructure Clouds," in *Proc. IEEE Int. Conf. Cloud Eng.*, Mar. 2013, pp. 11–20.
[37] A. Gupta, O. Sarood, L. Kale, and D. Milojicic, "Improving HPC application performance in cloud through dynamic load balancing," in *Proc. 13th IEEE/ACM Int. Symp. Cluster, Cloud, Grid Comput.*, 2013, pp. 402–409.
[38] P. Fan, Z. Chen, J. Wang, Z. Zheng, and M. R. Lyu, "Topology-aware deployment of scientific applications in cloud computing," in *Proc. IEEE 5th Int. Conf. Cloud Comput.*, 2012, pp. 319–326.
[39] F. Berman, A. Chien, K. Cooper, J. Dongarra, I. Foster, D. Gannon, L. Johnsson, K. Kennedy, C. Kesselman, J. Mellor-Crumme, and others, "The grads project: Software support for high-level grid application development," *Int. J. High Perform. Comput. Appl.*, vol. 15, pp. 327–344, 2001.
[40] H. Kim, Y. el Khamra, I. Rodero, S. Jha, and M. Parashar, "Autonomic management of application workflows on hybrid computing infrastructure," *Sci. Program.*, vol. 19, no. 2, pp. 75–89, Jan. 2011.
[41] M. Kulkarni, M. Burtscher, R. Inkulu, K. Pingali, and C. Casçaval, "How much parallelism is there in irregular applications?" *SIGPLAN Not.*, vol. 44, no. 4, pp. 3–14, Feb. 2009.

Abhishek Gupta received the BTech degree in computer science and engineering from Indian Institute of Technology (IIT), Roorkee, India, in 2008, and the MS and PhD degrees in computer science from the University of Illinois at Urbana-Champaign (UIUC) in 2011 and 2014, respectively. He is currently a cloud security architect at Intel Corp. His current research interests include parallel programming, HPC, scheduling, and cloud computing.

Paolo Faraboschi received the PhD degree in electrical engineering and computer science from the University of Genoa, Italy. He is currently a distinguished technologist at HP Labs. His current research interests include intersection of architecture and software. From 2004 to 2009, he led the HPL research activity on system-level simulation. From 1995 to 2003, he was the principal architect of the Lx/ST200 family of VLIW cores. He is a fellow of the IEEE and an active member of the computer architecture community.



Filippo Gioachin received the Laurea degree in computer science and engineering from the University of Padova, and the PhD degree in computer science from the University of Illinois at Urbana-Champaign. He is currently a research manager and senior researcher at HP Labs Singapore, where he is contributing to the innovation in cloud computing. His main focus is on integrated online software development.

Laxmikant V. Kale received the BTech degree in electronics engineering from Benares Hindu University, India, in 1977, an the ME degree in computer science from Indian Institute of Science in Bangalore, India, in 1979, and the PhD degree in computer science from State University of New York, Stony Brook, in 1985. He is currently a full professor at the University of Illinois at Urbana-Champaign. His current research interest includes parallel computing. He is a fellow of the IEEE.

Richard Kaufmann received the BA degree from the University of California at San Diego, Sin 1978, where he was a member of the UCSD Pascal Project. He is currently the VP of the Cloud Lab at Samsung Data Systems. Previously, he was chief technologist of HP's Cloud Services Group and HP's cloud and high-performance computing server groups.

Bu Sung Lee received the BSc (Hons.) and PhD degrees from the Department of Electrical and Electronics, Loughborough University of Technology, United Kingdom, in 1982 and 1987, respectively. He is currently an associate professor with the School of Computer Engineering, Nanyang Technological University. He held a joint position as the director (Research) HP Labs Singapore from 2010 to 2012. His current research interests include mobile and pervasive networks, distributed systems, and cloud computing.

Verdi March received the BSc degree from Faculty of Computer Science, University of Indonesia in2000, and the PhD degree from the Department of Computer Science, National University of Singapore in 2007. He is currently a lead research scientist with Visa Labs. Prior to working with Visa Labs, he held various research or engineering positions with HP Labs, Sun Microsystems Inc., and the National University of Singapore.

Dejan Milojicic received the BSc and MSc degrees from Belgrade University, Belgrade, Serbia, in 1983 and 1986, respectively. He received the PhD degree from the University of Kaiserslautern, Kaiserslautern, Germany, in 1993. He is currently a senior researcher at HP Labs, Palo Alto, CA. He is the IEEE Computer Society 2014 president and the founding editor-in-chief of the IEEE Computing Now. He was in OSF Research Institute, Cambridge, MA, from 1994 to 1998, and Institute Mihajlo Pupin, Belgrade, Serbia, from 1983 to 1991. He is a fellow of the IEEE.

ChunHui Suen received the BEng (First Class Hons.) from National University of Singapore, Singapore, and the MSc and PhD degrees from Technische Universitaet Munchen, Munich, Germany. He is currently an engineer and researcher with keen interest in the field of IT security, virtualization and cloud, with research experience in TPM related technologies, various hypervisors (xen, kvm, vmware), Linux kernel development, and various cloud stacks (openstack, AWS).