Optimizing VM Placement for HPC in the Cloud

Abhishek Gupta*
Dejan Milojicic
HP Labs Palo Alto, CA, USA
(abhishek.gupta2, dejan.milojicic)@hp.com

Laxmikant V. Kalé
University of Illinois at Urbana-Champaign
Urbana, IL 61801, USA
kale@illinois.edu

ABSTRACT

"Computing as a service" model in cloud has encouraged High Performance Computing to reach out to wider scientific and industrial community. Many small and medium scale HPC users are exploring Infrastructure cloud as a possible platform to run their applications. However, there are gaps between the characteristic traits of an HPC application and existing cloud scheduling algorithms. In this paper, we propose an HPC-aware scheduler and implement it atop Open Stack scheduler. In particular, we introduce topology awareness and consideration for homogeneity while allocating VMs. We demonstrate the benefits of these techniques by evaluating them on a cloud setup on Open Cirrus testbed.

Categories and Subject Descriptors

D.1.3 [**Concurrent Programming**]: Parallel Programming; K.6.4 [**System Management**]: Centralization/decentralization

Keywords

High Performance Computing, Clouds, Resource Scheduling

1. INTRODUCTION

Cloud computing is increasingly being explored as a cost effective alternative and addition to supercomputers for some HPC applications [8, 13, 17, 24]. Cloud provides the benefits of economy of scale, elasticity and virtualization to HPC community and is attracting many users which cannot afford to establish their own dedicated cluster due to up-front investment, sporadic demands or both.

However, presence of commodity interconnect, performance overhead introduced by virtualization and performance variability are some factors which imply that cloud can be suitable for some HPC applications but not all [8, 15]. Past research [15,16,18,24] on HPC in cloud has primarily focused on evaluation of scientific parallel applications (such as those written in MPI [6]) and have been mostly pessimistic. To the best of our knowledge, there have been few efforts on VM scheduling algorithms which take into account the nature of HPC application - tightly coupled processes which perform frequent inter-process communication and synchronizations. VM to physical machine placement can have a significant impact on performance. With this as motivation, the primary question that we address through this research is the following: Can we improve HPC application performance in Cloud through intelligent VM placement strategies tailored to HPC application characteristics?

Current cloud management systems such as Open Stack [3] and Eucalyptus [23] lack an intelligent scheduler for HPC applications. In our terminology, *scheduling or placement* refers to selection of physical servers for provisioning virtual machines. In this paper, we explore the challenges and alternatives for scheduling HPC applications on cloud. We implement HPC-aware VM placement strategies - specifically topology awareness and hardware awareness in Open Stack scheduler and evaluate their effectiveness using performance measurement on a cloud, which we setup on Open Cirrus [9] platform.

The benefit of our approach is that HPC-aware scheduling strategies can result in significant benefits for both HPC users and cloud providers. Using these strategies, cloud providers can utilize infrastructure more and offer improved performance to cloud users. This can allow cloud providers to obtain higher profits for their resources. They can also pass some benefits to cloud users to attract more customers.

The key contribution of this work is a novel scheduler for HPC application in cloud for Open Stack through topology and hardware awareness.We address the initial VM placement problem in this paper.

The remainder of this paper is organized as follows: Section 2 provides background on Open Stack. Section 3 discusses our algorithms followed by implementation. Next, we describe our evaluation methodology in Section 4. We present performance results in Section 5. Related work is discussed in section 6. We give concluding remarks along with an outline of future work in Section 7.

2. OPEN STACK SCHEDULER

Open Stack [3] is an open source cloud management system which allows easy control of large pools of infrastructure resources (compute, storage and networking) through-

*Abhishek, an intern at HP labs, is also a Ph.D. student at University of Illinois at Urbana-Champaign.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Workshop on Cloud Services, Federation, and the 8th Open Cirrus Summit, September 21, 2012, San Jose, CA, USA.

Copyright 2012 ACM 978-1-4503-1267-7 ...$10.00.

out a datacenter. Open Stack has multiple projects, each with a different focus, examples are compute (Nova), storage (Swift), Image delivery and registration (Glance), Identity (Keystone), Dashboard (Horizon) and Network Connectivity (Quantum).

Our focus in this work is on the compute component of Open Stack, known as Nova. A core component of a cloud setup using Open Stack is the Open Stack scheduler, which selects the physical nodes where a VM will be provisioned. We implemented our scheduling techniques on top of existing Open Stack scheduler and hence first we summarize the existing scheduler. In this work, we used the Diablo (2011.3) version of Open Stack.

Open Stack scheduler receives a VM provisioning request (`request_spec`) as part of RPC message. `request_spec` specifies the number of instances (VMs), instance type which maps to resource requirements (number of virtual cores, amount of memory, amount of disk space) for each instance and some other user specified options that can be passes to the scheduler at run time. Host capability data is another important input to the scheduler which contains the list of physical servers with their current capabilities (free CPUs, free memory etc.).

Using `request_spec` and capabilities data, the scheduling algorithm consists of two steps:

1. Filtering - excludes hosts which are incapable of fulfilling the request based on certain criteria (e.g free cores < requested virtual cores).
2. Weighing - computes the relative fitness of filtered list of hosts to fulfill the request using cost functions. Multiple cost functions can be used, each host is first scored by running each cost function and then weighted scores are calculated for each host by multiplying score and weight of each cost function. An example of cost function is free memory in a host.

Next, the list of hosts is sorted by the weighted score and VMs are provisioned on hosts using this sorted list.

There are various filtering and weighing strategies currently available in Open Stack. However, one key disadvantage of the current Open Stack scheduler is that scheduling policies do not consider application type and priorities, which could allow more intelligent decision making. Further, scheduling policies ignore processor heterogeneity and network topology while selecting hosts for VMs. Existing scheduling policies consider the k VMs requested as part of a user request as k separate VM placement problems. Hence, it runs the core of scheduling algorithm k times to find placement of k VMs constituting a single request, thereby avoiding any co-relation between the placement of VMs which comprise a single request. Some example of currently available schedulers are *Chance* scheduler (chooses host randomly across availability zones), *Availability zone* scheduler (similar to chance, but chooses host randomly from within a specified availability zone) and *Simple* scheduler (chooses least loaded host e.g. host with least number of running cores).

3. AN HPC-AWARE SCHEDULER

In this paper, we address the initial VM placement problem (Figure 1). The problem can be formulated as - Map: k VMs $(v_1, v_2, .., v_k)$ each with same, fixed resource requirements (decided by instance type: CPU, memory, disk etc) to N physical servers $P_1, P_2, ..., P_n$, which are unoccupied or partially occupied, while satisfying resource requirements. Moreover, our focus is on providing the user a VM placement optimized for HPC.

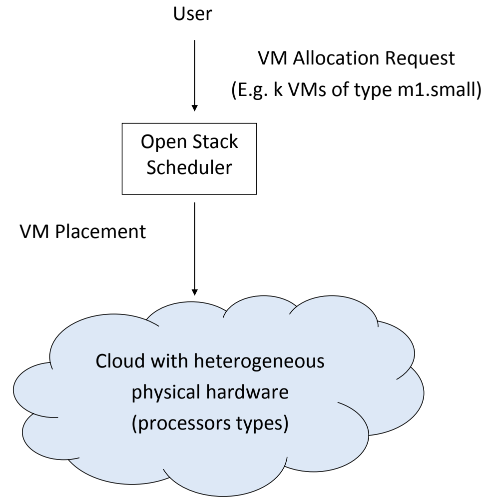


Figure 1: VM Placement

Next, we discuss the design and implementation of the proposed techniques atop existing Open Stack scheduling framework.

3.1 Techniques

In this section, we describe two techniques for optimizing the placement of VMs for an HPC-optimized allocation.

3.1.1 Topology Awareness

An HPC application consists of n parallel processes, which typically perform inter-process communication for overall progress. The effect of cluster topology on application performance has been widely explored by HPC community. In the context of cloud, the cluster topology is unknown to the user. The goal is to place the VMs on those physical machines which are as close to each other as possible with respect to the cluster topology. Let us consider a practical example - the cluster topology of Open Cirrus HP Labs site - a simple topology, each server is a 4-core node and there are 32 nodes in a rack, all nodes in a rack are connected by a 1Gbps link to a switch. All racks are connected using a 10Gbps link to a top-level switch. In this case, the 10Gbps link is shared by 32 nodes, effectively providing a bandwidth of 10Gbps/32 = 0.312 Gbps between two nodes in different rack when all nodes are communicating. However, the point-to-point bandwidth between two nodes in the same rack is 1 Gbps. Hence, it would be better to pack VMs to nodes in the same rack compared to a random placement policy, which can potentially distribute them all over the cluster.

3.1.2 Hardware Awareness/Homogeneity

Another characteristics of HPC applications is that they are generally iterative and bulk synchronous, which means that in each iteration, there are two phases - computation followed by communication/synchronization which also acts as a barrier. The next iteration can start only when all processes have finished previous iteration. Hence, a single slow process can degrade the performance of whole application. This also means that faster processors waste lot of time wait-

ing for slower processors to reach the synchronization point.

In case of cloud, the user is unaware of the underlying hardware on which his VMs are placed. Since clouds evolve over time, they consist of heterogeneous physical servers. Existing VM placement strategies ignore the heterogeneity of the underlying hardware. This can result in some VMs running on faster processors, while some running on slower processors. Some cloud providers, such as Amazon EC2 [1], address the problem of heterogeneity by creating a new compute unit and allocating based on that. Using a new compute unit enables them to utilize the remaining capacity and allocating it to a separate VM using shares (e.g 80-20 CPU share). However, for HPC applications, this can actually make the performance worse, since all the processes comprising an application can quickly become out of sync due to the effect of other VM on same node. To make sure the k VMs are at sync, all k VMS need to be scheduled together if they are running on heterogeneous platform and sharing CPU with other VMs (some form of gang scheduling). In addition, the interference arising from other VMs can have a significant performance impact on HPC application. To avoid such interference, Amazon EC2 uses a dedicated cluster for HPC [2]. However, the disadvantage of this is lower utilization which results in higher price.

To address the needs of homogeneous hardware for HPC VMs, we take an alternative approach. We make the VM placement hardware aware and ensure that all k VMs of a user request are allocated same type of processors.

3.2 Implementation

We implemented the techniques discussed in section 3.1 on top of Open Stack scheduler. The first modification to enable HPC-aware scheduling is to switch to the use of group scheduling which allows the scheduling algorithm to consider placement of k VMs as a single scheduling problem rather than k separate scheduling problems. Our topology-aware algorithm (pseudo-code shown in Algorithm 1) proceeds by considering the list of physical servers (*hosts* in Open Stack scheduler terminology). Next, the filtering phase of the scheduler removes the hosts which cannot meet the requested demands of a single VM. We also calculate the maximum number of VMs each host can fit (based on the number of virtual cores and amount of memory requested by each VM). We call it `hostCapacity`. Next, for each rack, we calculate the number of VMs the rack can fit (`rackCapacity`) by summing the `hostCapacity` for all hosts in a rack. Using this information, scheduler creates a build plan, which is an ordered list of hosts such that if $i < j$, i^{th} host belongs to a rack with higher capacity compared to j^{th} host or both host belong to same rack and `hostcapacity` of ith host is greater or equal to that of jth host. Hence, the scheduler places VMs in a rack with largest `rackCapacity` and the host in that rack with largest `hostCapacity`.

One potential disadvantage of our current policy of selecting the rack with maximum available capacity is unnecessary system fragmentation. To overcome this problem, we plan to explore more intelligent heuristics such as selecting the rack (or a combination of racks) with the minimum excess capacity over the VM set allocation.

To ensure homogeneity, the scheduler first groups the hosts into different lists based on their processor type and then applies the scheduling algorithm described above to these groups, with preference to the best configuration first. Currently, we use CPU frequency as the distinction criteria between different processor types. For more accurate distinction, we plan on incorporating additional factors such as cache sizes and MIPS.

Algorithm 1 Pseudo code for Topology aware scheduler

1: $capability$ = list of capabilities of unique hosts
2: $request_spec$ = request specification
3: $numHosts = capability.length()$
4: $filteredHostList = new\ vector < int >$
5: $rackList = new\ set < int >$
6: $hostCapacity = new\ int[numHosts]$
7: **for** $i = 1$ to $i < numHosts$ **do**
8: $hostCapacity[i] = max(capability[i].freeCores/request_spec.instanceVcpus, capability[i].freeMemory/request_spec.instanceMemory)$
9: **if** $hostCapacity[i] > 0$ **then**
10: $filteredHostList.push(i)$
11: **end if**
12: $rackList.add(capability[i].rackid)$
13: **end for**
14: $numRacks = rackList.length()$
15: $rackCapacity = new\ int[numRacks]$
16: **for** $j = 1$ to $j < numRacks$ **do**
17: $rackCapacity[j] = \sum_i hostCapacity[i]$ $\forall i$ such that $capability[i].rackid = j$
18: **end for**
19: Sort $filteredHostList$ by decreasing order of $hostCapacity[j]$ where $j \in filteredHostList$. Call this $sortedHostList$
20: Stable Sort $sortedHostList$ by decreasing order of $rackCapacity[capability[j].rackid]$ where $j \in filteredHostList$. Call this $PrelimBuildPlan$.
21: $buildPlan = new\ vector[int]$
22: **for** $i = 1$ to $i <= numFilteredHosts$ **do**
23: **for** $j = 1$ to $j <= hostCapacity[PrelimBuildPlan[i]]$ **do**
24: $buildPlan.push(PrelimBuildPlan[i])$
25: **end for**
26: **end for**
27: return $buildPlan$

4. EVALUATION METHODOLOGY

In this section, we describe the platform which we setup and the applications which we chose for this study.

4.1 Experimental Testbed

We setup a cloud using Open Stack on Open Cirrus testbed at HP Labs site [9]. Open Cirrus is a cluster established for system level research. This testbed has 3 types of servers:

- Intel Xeon E5450 (12M Cache, 3.00 GHz)
- Intel Xeon X3370 (12M Cache, 3.00 GHz)
- Intel Xeon X3210 (8M Cache, 2.13 GHz)

The topology is as described in section 3.1.1.

We used KVM [7] as hypervisor since past research has indicated that KVM is a good choice for virtualization for HPC clouds [25]. For network virtualization, we initially used the default network driver (*rtl8139*) but subsequently switched to the *virtio-net* driver on observing improved network performance (details in section 5). The VMs used for the experiments performed in this paper were of the type m1.small (1 core, 2 GB memory, 20 GB disk).

4.2 Benchmarks and Applications

For this study, we chose certain benchmarks and real world applications as representatives of HPC applications.

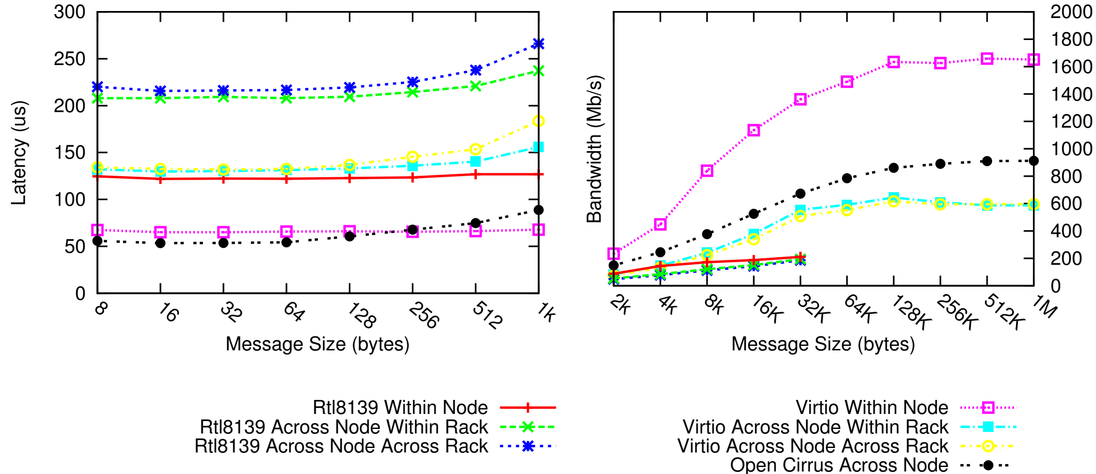


Figure 2: Latency and Bandwidth vs. Message Size for different VM placement.

- Jacobi2D - A kernel which performs 5-point stencil computation to average values in a 2-D grid. Such stencil computation is very commonly used in scientific simulations, numerical algebra, and image processing.
- NAMD [11] - A highly scalable molecular dynamics application and representative of a complex real world application used ubiquitously on supercomputers. We used the ApoA1 input (92k atoms) for our experiments.
- ChaNGa [19] (Charm N-body GrAvity solver) - A scalable application used to perform collisionless N-body simulation. It can be used to perform cosmological simulation and also includes hydrodynamics. It uses Barnes-Hut tree [10] to calculate forces between particles. We used a 300, 000 particle system for our runs.

All these applications are written in Charm++ [20], which is a C++ based object-oriented parallel programming language. We used the `net-linux-x86-64 udp` machine layer of Charm++ and used -O3 optimization level.

5. RESULTS

We first evaluate the effect of topology-aware scheduling. Figure 2 shows the results of a ping-pong benchmark. We used a Converse [12] (underlying substrate of Charm++) ping-pong benchmark to compare latencies and bandwidth for various VM placement configurations. Figure 2 presents several insights. First, we see that *virtio* outperforms *rtl8139* network driver both for intra-node and inter-node VM communication, making it a natural choice for remainder of the experiments. Second, there is significant virtualization overhead. Even for communication between VMs on same node, there is a 64 usec latency using virtio. Similarly, for inter-node communication, VM latencies are around twice compared to communication between physical nodes in Open Cirrus and there is also substantial reduction in achieved bandwidth, although the degradation in bandwidth (33% reduction) is less compared to the degradation in latencies (100% increase). Third, there is very little difference for latencies and bandwidth when comparing communication between VMs on different nodes but same rack and between VMs on different nodes on different racks. This can be attributed to the use of wormhole routing in modern network

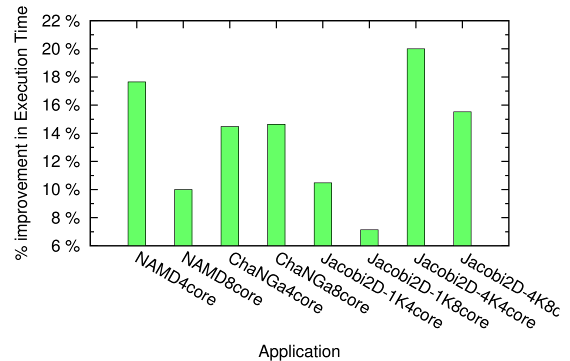


Figure 3: Percentage improvement achieved using hardware aware Scheduling compared to the case where 2 VMs were on slower processors and rest on faster processors

which means that the extra hops cause very little performance overhead. As we discussed in section 3.1.1, effects of intra-rack and cross-rack communication become more prominent as we scale up or the application performs significant collective communication such as all-to-all data movement.

We compared our topology aware scheduler with random scheduling. To perform a fair comparison, we explicitly controlled the scheduling such that the default (random) case corresponds to a mapping where the VMs are distributed to two racks, we keep the number of VM on each host as two in both cases. For the topology aware case, the scheduler placed all VMs to the hosts in same rack. For these experiments, we were able to gain up to 5% in performance compared to random scheduling. We expect the benefits of topology aware scheduling to increase as we run on higher core counts in cloud.

Figure 3 shows the effect of hardware aware VM placement on the performance of some applications for different number of VMs. We compare two cases - when all VMs are mapped to same type of processors (Homo) and when two VMs are mapped to a slower processors, rest to the faster processor (Hetero). To perform a fair comparison, we keep the number of VMs on each host as two in both cases. % im-

provement is calculated as $(T_{Hetero} - T_{Homo})/T_{Hetero}$. From Figure 3, we can observe that the improvement achieved depends on the nature of application and the scale at which it is run. The improvement is not equal to the ratio of sequential execution time on slower processor to that on faster processor since parallel execution time also includes the communication time and parallel overhead, which is not necessarily dependent on the processor speeds. We achieved up to 20% improvement in parallel execution time - which means we were able to save *20% of time * N CPU-hours*, where N is the number of processors used.

We used the Projections [21] tool to analyze the performance bottlenecks in these two cases. Figure 4 shows the CPU (VM) timelines for an 8-core Jacobi2D experiment, x-axis being time and y-axis being the (virtual) core number. White portion shows idle time while colored portions represent application functions. For figure 4a, the first 2 VMs were mapped to slower processors and are busy for all the time. It is clear that there is lot more idle time on VMs 3-7 compared to first 2 VMs since they have to wait for VMs 0-1 to reach the synchronization point after finishing the computation. The first two VMs are bottleneck in this case and result in execution time being 20% more compared to the homogeneous case. In Figure 4b all 8 VMs are on same type of processors, here the idle time is due to the communication time.

Figure 5 shows the overall performance that we achieve using these techniques on our test-bed. We compare the performance to that achieved without virtualization on the same testbed. It is clear that using our techniques, even communication intensive applications such as NAMD and ChaNGa scale quite well, compared to their scalability on the physical platform. However, effect of virtualization on network performance can quickly become a scalability bottleneck (as suggested by Figures 2 and 4).

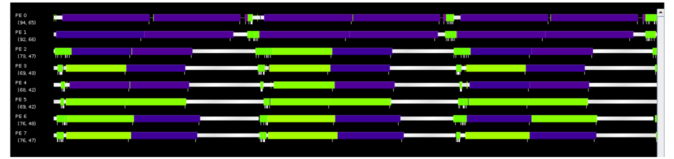
6. RELATED WORK

There have been several studies on HPC in cloud using benchmarks such as NPB and real applications [8, 13, 15–17, 22, 24]. The conclusions of these studies have been the following:

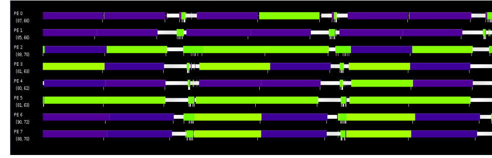
- Interconnect and I/O performance on commercial cloud severely limit performance and cause significant performance variability.
- Cloud cannot compete with supercomputers based on the metric $/GFLOPS for large scale HPC applications.
- It can be cost-effective to run some applications on cloud compared to supercomputer, specifically those with less communication and at low scale.

In our earlier work [15], we studied the performance-cost tradeoffs of running different applications on supercomputer vs. cloud. We demonstrated that the suitability of a platform for an HPC application depends upon application characteristics, performance requirements and user preferences. In another work, we explored techniques to improve HPC application performance in Cloud through an optimized parallel run-time system. We used a cloud-friendly load balancing system to reduce the performance degradation suffered by parallel application due to effect of virtualization in cloud.

In this paper, we take one step further and research VM placement strategies which can result in improved applica-



(a) Heterogeneous: First two VMs on slower processors



(b) Homogeneous: All 8 VMs on same type of processors

Figure 4: Timeline of 8 VMs running Jacobi2D (2K by 2K): white portion shows idle time while colored portions represent application functions.

tion performance. Our focus is HPC applications - which consist of k parallel instances typically requiring synchronization through inter process communication.

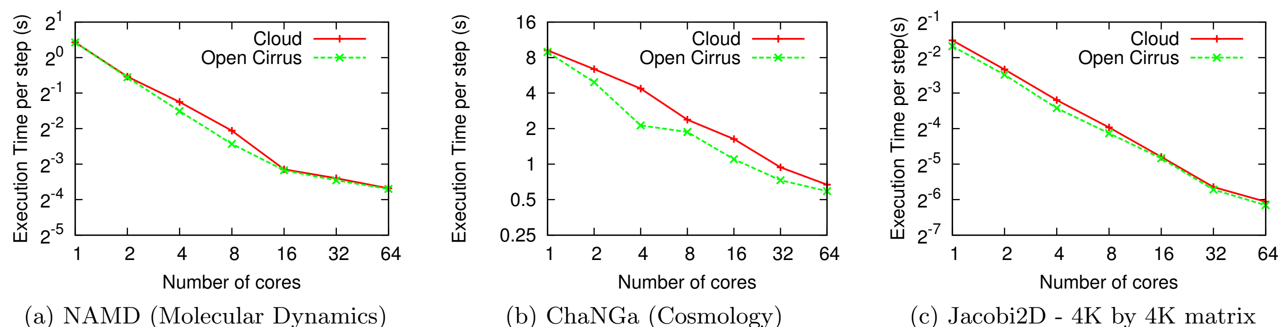
Existing scheduler do not address this problem. Cloud management systems such as Open Stack [3], Eucalyptus [23] and Open Nebula [5] lack strategies which consider such tightly coupled nature of VMs comprising a single user request, while making scheduling decisions. Fan et al. discuss topology aware deployment for scientific applications in cloud and map the communication topology of a parallel application to the VM physical topology [14]. However, we focus on allocating VMs in a topology aware manner to provide a good set of VMs to an HPC application user. Moreover, we address the heterogeneity of hardware. Amazon EC2's Cluster Compute instance introduces Placement Groups such that all instances launched within a Placement Group are expected to have low latency and full bisection 10 Gbps bandwidth between instances [2]. It is unknown and undisclosed how strict those guarantees are and what techniques are used to provide them.

There have been several efforts on job scheduling for HPC applications, such as LSF (Load Sharing Facility) [4] - a commercial job scheduler which allows load sharing using distribution of jobs to available CPUs in heterogeneous network. SLURM, ALPS, MOAB, Torque, Open PBS, PBS Pro, SGE, Condor are other examples. However, they perform scheduling at the granularity of physical machines. In Cloud, virtualization enables consolidation and sharing of nodes between different types of VMs which can enable improved server utilization.

7. CONCLUSIONS AND FUTURE WORK

In this paper, we learned that utilizing the knowledge of target application for a VM can lead to more intelligent VM placement decisions. We made a case for HPC-aware VM placement techniques and demonstrated the benefits of using HPC-aware VM placement techniques for efficient execution of HPC applications in cloud. In particular, we implemented topology and hardware awareness in Open Stack scheduler and evaluated them on a cloud setup on Open Cirrus testbed.

In future, we plan to research how to schedule a mix of HPC and non-HPC applications in an intelligent fashion to increase resource utilization. E.g. co-locating VMs



(a) NAMD (Molecular Dynamics) (b) ChaNGa (Cosmology) (c) Jacobi2D - 4K by 4K matrix

Figure 5: Execution Time vs. Number of cores/VMs for different applications.

which are network bandwidth intensive and VMs which are compute intensive to increase resource utilization. Comparison with other network virtualization techniques such as vhost_net and tcp protocol is another direction of future research.

8. REFERENCES

[1] Amazon Elastic Compute Cloud (Amazon EC2). `http://aws.amazon.com/ec2`.

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

[3] Open Stack Open Source Cloud Computing Software. `http://openstack.org`.

[4] Platform Computing. `http://www.platform.com/workload-management/high-performance-computing`.

[5] The Cloud Data Center Management Solution . `http://opennebula.org`.

[6] Mpi: A message passing interface standard. In *M. P. I. Forum*, 1994.

[7] KVM – Kernel-based Virtual Machine. Technical report, Redhat, Inc., 2009.

[8] Magellan Final Report. Technical report, U.S. Department of Energy (DOE), 2011. `http://science.energy.gov/~/media/ascr/pdf/program-documents/docs/Magellan_Final_Report.pdf`.

[9] A. Avetisyan et al. Open Cirrus: A Global Cloud Computing Testbed. *IEEE Computer*, 43:35–43, April 2010.

[10] J. Barnes and P. Hut. A hierarchical $O(N \log N)$ force-calculation algorithm. *Nature*, 324:446–449, December 1986.

[11] 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 *IPDPS 2008*, pages 1–12, April 2008.

[12] Department of Computer Science,University of Illinois at Urbana-Champaign, Urbana, IL. *The CONVERSE programming language manual*, 2006.

[13] 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. Cloud Computing and Its Applications, Oct. 2008.

[14] P. Fan, Z. Chen, J. Wang, Z. Zheng, and M. R. Lyu. Topology-aware deployment of scientific applications in cloud computing. *Cloud Computing, IEEE International Conference on*, 0, 2012.

[15] A. Gupta and D. Milojicic. Evaluation of HPC Applications on Cloud. In *Open Cirrus Summit (Best Student Paper)*, pages 22 –26, Atlanta, GA, Oct. 2011.

[16] A. Gupta et al. Exploring the performance and mapping of hpc applications to platforms in the cloud. In *Proceedings of the 21st international symposium on High-Performance Parallel and Distributed Computing*, HPDC '12, pages 121–122, New York, NY, USA, 2012. ACM.

[17] A. Iosup, S. Ostermann, N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema. Performance Analysis of Cloud Computing Services for Many-Tasks Scientific Computing. *IEEE Trans. Parallel Distrib. Syst.*, 22:931–945, June 2011.

[18] K. Jackson et al. Performance Analysis of High Performance Computing Applications on the Amazon Web Services Cloud. In *CloudCom'10*, 2010.

[19] P. Jetley, F. Gioachin, C. Mendes, L. V. Kale, and T. R. Quinn. Massively parallel cosmological simulations with ChaNGa. In *IPDPS 2008*, pages 1–12, 2008.

[20] L. Kalé. The Chare Kernel parallel programming language and system. In *Proceedings of the International Conference on Parallel Processing*, volume II, pages 17–25, Aug. 1990.

[21] L. Kalé and A. Sinha. Projections : A scalable performance tool. In *Parallel Systems Fair, International Parallel Processing Sympos ium*, pages 108–114, Apr. 1993.

[22] 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*, UCHPC-MAW '09, pages 17–20, New York, NY, USA, 2009. ACM.

[23] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov. The Eucalyptus Open-source Cloud-computing System. In *Proceedings of Cloud Computing and Its Applications*, Oct. 2008.

[24] E. Walker. Benchmarking Amazon EC2 for High-Performance Scientific Computing. *LOGIN*, pages 18–23, 2008.

[25] 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. *Cloud Computing, IEEE International Conference on*, 0:9–16, 2011.