CPU and Memory Over-commitment for HPC Cloud

Anonymized

Abstract

Traditionally, High Performance Computing (HPC) workloads have been running on dedicated bare-metal clusters ranging from tens to thousands of nodes. Recently, cloud computing has attracted more and more HPC users due to its key benefits, such as ease of management, low total cost of ownership, on-demand resource provisioning, etc. A significant challenge for running HPC in the cloud, however, is to maintain high performance and quality-of-service while simultaneously supporting multiple tenants sharing the same infrastructure. In this work, we design *virtual throughput clusters*, as a novel approach to allocating cloud resources, with CPU and memory over-commitment to achieve high resource utilization. Built on top of virtualization, virtual throughput clusters guarantees security and fairness for multi-tenancy support. By benchmarking popular HPC workloads, we demonstrate that virtual throughput clusters with CPU over-commitment can achieve better system throughput than bare-metal clusters, and memory over-commitment can be practical in certain scenarios.

Keywords Virtualization, HPC, Resource Management, Over-commitment, Multi-tenancy

1 Introduction

Today's scientific discoveries and business intelligence are driven by high-fidelity, large-scale simulation and data analytics. To meet the increasing computing demands from virtually every aspect of the society, HPC is continuously evolving to solve more complex and challenging problems. On the one hand, national labs and research institutes run HPC on supercomputers for scientific breakthroughs and national security. On the other hand, enterprises and organizations deploy HPC on small to medium sized clusters to process data and extract insights. Recently, the explosively growing machine learning applications have increased the adoption as well as impact of HPC as they also exploit parallelism and hardware acceleration to speed up the processing of massive amount of data.

HPC workloads have traditionally been run only on bare-metal, unvirtualized hardware to drive maximum performance. The roadblock to virtualization was due to the concern that the extra hypervisor layer could introduce performance overhead. However, this has started to change with the introduction of increasingly sophisticated hardware support for virtualization and software optimization [15, 6]. Performance of these highly parallel HPC workloads has increased dramatically over the last decade, enabling organizations to begin to embrace the numerous benefits that a virtualization platform can offer [8]. As a result, we are witnessing a popular trend that enterprises convert their on-prem bare-metal clusters to virtualized, shared private cloud. For instance, the Johns Hopkins University Applied Physics Laboratory recently virtualized their 3728-core bare-metal cluster to share between Windows and Linux users. The reported improvement in resource utilization ranges from 9.1% to 29.2%, and simulations speed up by 4% on average [18].

At the same time, public cloud, such as Amazon AWS and Google GCP, is becoming a popular alternative for HPC practitioners. Recent studies show that the usage of public cloud has grown more than five-fold among all HPC sites worldwide, from 13% in 2011 to 74% in 2018 [7]. With virtually unlimited scalability and on-demand resource subscription, public cloud starts to host compute- and data-intensive workloads across various industry verticals. These workloads span the traditional HPC applications, like genomics and weather prediction, as well as emerging applications, like machine learning and deep learning.

There is a fruitful body of research on resource management in Cloud Computing [19, 24, 10]. Dynamic resource scheduling and load balancing are used to maximize system utilization and efficiency [2, 17]. These techniques, however, are not straightforward to apply to HPC workloads which are highly sensitive to resource change and interference. Actually, resource management has been identified as one of the open challenges for HPC cloud [16]. Currently, cloud service providers (CSPs) are often limited to statically and conservatively reserve resources based on peak resource requirements to respect service level agreements (SLAs). For example, Microsoft Azure allocates dedicated supercomputers from Cray, and Amazon AWS offers dedicated nodes for full-size VMs. This essentially offsets the elasticity and efficiency benefits of the Cloud Computing business model.

In this paper, we present *virtual throughput clusters (VTC)* as a novel approach for cloud resource allocation to efficiently and effectively support HPC workloads with multi-tenancy. Based on virtual machine (VM), VTC goes beyond traditional way of statically splitting resources among tenants and applies resource over-commitment to optimize system utilization and throughput. By giving each tenant a virtual cluster that mimics the underlying physical cluster, VTC delegates the resource management task to the hypervisor to improve flexibility as well as efficiency. When all tenants are busy consuming their cycles, VTC guarantees that each tenant is getting his/her fair share according to pre-defined SLA terms. When some tenant is not fully using the allocated resources, VTC takes advantage of the work-conserving property of the hypervisor scheduler to assign

the idle resources to other tenant(s) who can benefit from additional resources. Consequently, CSPs can ensure quality-of-service while maximizing system utilization.

The rest of the paper is organized as follows. Section II provides background and motivation. Section III introduces the design of VTC, followed by validation and empirical evaluation results in Section IV. Section V concludes this work and points out future directions.

2 Background and Motivation

2.1 Why virtualized HPC?

Although HPC workloads are most often run on bare-metal systems, this has started to change with the realization that many of the benefits that virtualization offers to enterprises can often also add value in HPC environments. The following are among those benefits:

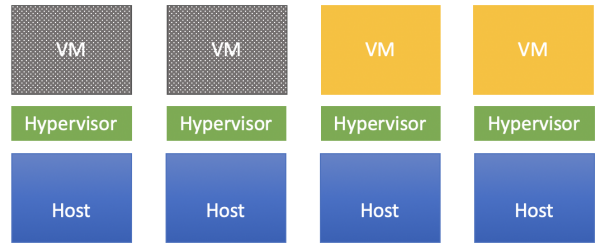
- Supports a more diverse end-user population with differing software requirements
- Provides multi-tenancy data security by isolating user workloads into separate VMs
- Provides fault isolation, root access, and other capabilities not available in traditional HPC environments
- Creates a more dynamic execution environment in which VMs and their encapsulated workloads can be live-migrated across the cluster for load balancing, for maintenance, for fault avoidance, and so on

In addition to above benefits, the performance of HPC applications on virtual platform has dramatically improved over the past years. This is especially true for HPC throughput workloads, which in contrast to MPI workloads, consist of a large number of independent tasks that can be executed in parallel. Throughput workloads represent a significant portion of HPC workloads and include life sciences, electronic design automation, image rendering, etc. Studies have demonstrated that the performance gap between virtual and bare metal for HPC throughput workloads is closing, with just 1 or 2 percentage difference [8].

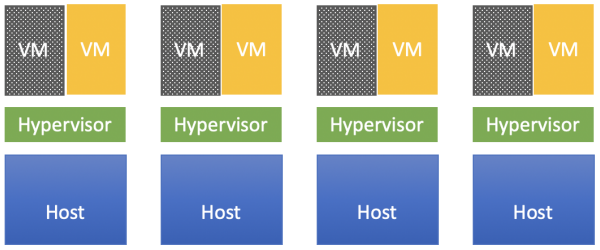
2.2 HPC cloud resource allocation

Virtualization brings new flexibility to HPC. With that flexibility, however, one must be careful to configure the cloud environment to simultaneously achieve both agility and high performance. Although it is possible to create a single virtual cluster that spans an entire data center partition with one maximally sized VM per node, this approach misses the opportunity to enable several important virtualization benefits. Among these are the ability to support per-tenant or per-project software stacks as well as security and fault separation between the tenant workloads. In the more typical case, multiple virtual clusters should be hosted simultaneously on the physical cluster for isolation between tenants.

In a situation in which a tenant is using resources intensively, it might make most sense to assign a dedicated subset



(a) Partition on non-overlapping nodes



(b) Partition on same nodes

Figure 1. Example of traditional static allocation with four hosts and two tenants. Gray VMs for one tenant and yellow VMs for the other.

of hardware to the tenant and to configure VMs on those nodes appropriately. With two such tenants, one can either place their VMs on a non-overlapping set of nodes (Figure 1a) or place them on the same nodes, being careful to size their VMs to avoid any over-commitment (Figure 1b). The major flaw with these approaches is that resources are statically partitioned and thus liable to under-utilization. Although tenants might be very busy in some time periods, they can also be less busy and even idle in other periods. In such cases, sizing VMs as described above can lead to commensurate losses in throughput because the idle resources serving one VM are not available to the other busy VM.

2.3 Resource over-commitment

The key to avoiding the above resource waste issue is resource over-commitment. In a virtualized environment, resource over-commitment means configuring VMs with more than the available physical resources. For example, on a host with 4 CPU cores and 8 GB memory, one could create and power on two VMs each with 4 virtual CPUs and 6 GB virtual memory. Resource over-commitment has been studied, but previous works did not optimize for HPC workloads and only considered CPU over-commitment [21, 20].

CPU over-commitment is accommodated by multiplexing virtual CPUs onto the physical CPU cores. In a modern hypervisor like VMware ESXi, a share-based mechanism can be enabled in the scheduler so that each VM can get a different portion of the physical CPUs based on its configured shares, even in the case of over-commitment [1]. Furthermore, a

work-conserving scheduler allows one VM to consume more than its fair CPU share if there are idle cycles from other VM(s). Memory over-commitment, on the other hand, is more challenging since multiplexing is not applicable. It is achieved by a set of memory reclamation techniques, including transparent page sharing (TPS), ballooning, compression, and hypervisor swapping [22, 5]. These techniques differ in the incurred overhead and are individually controlled by the hypervisor based on system memory state.

3 Virtual Throughput Clusters

In the previous section we explained why state-of-the-art resource allocation in cloud is sub-optimal for HPC workloads. This section will introduce our novel design that incorporates CPU and memory over-commitment for resource optimization. First, we start by introducing Virtual Throughput Clusters (VTC) with CPU over-commitment. Then, we continue to discuss the inclusion of memory over-commitment which is much more challenging. Lastly, we complete the design with dynamic VM migration, which mitigates potential performance penalties from resource over-commitment.

3.1 VTC with CPU over-commitment

With a share-based CPU scheduler in the hypervisor, the allocation of CPU resource among VMs on a shared host does not entirely rely on the number of vCPUs, but also is affected by the configurable shares. This design change offers the CSPs or system admins the freedom to change VM vCPU numbers without worrying about impacting resource allocation. VTC takes advantage of this flexibility and configures each tenant with the same amount of virtual CPUs as the physical CPUs on a node. In this way, each tenant can get access to all the CPUs inside the node, and consequently, if one tenant is not consuming the entire quota, the free CPU cycles can be used by other tenant(s). This approach can be easily scaled to a cluster by repeating the configuration on every single node. Then, each tenant gets allocated a virtual cluster that consists of one VM per node. From the end user perspective, each tenant gets exclusive access to a dedicated cluster with isolation in security, fault, and even performance. Using the previous example of sharing four nodes between two tenants, VTC is illustrated in Figure 2. In the experiment section later, we will demonstrate support of upto 4 tenants simultaneously on a single physical cluster with 4X CPU over-commitment.

3.2 Adding memory over-commitment

Similar to CPU over-commitment, memory can be configured in a way that the combined VM memory can be larger than the physical memory capacity. It is impossible, however, that all VMs actively access all configured memory beyond the physical capacity. This is mainly due to performance considerations. TPS, ballooning, and compression all have no guarantee to reclaim memory in a timely manner, and if all these techniques fail, hypervisor swapping will occur to swap out guest OS memory to disks, rendering unacceptable performance to the HPC workloads. Two questions that we address through performance study in later section are: 1) whether memory over-commitment can be practical; 2) how far can we go with active memory usage under memory over-commitment.

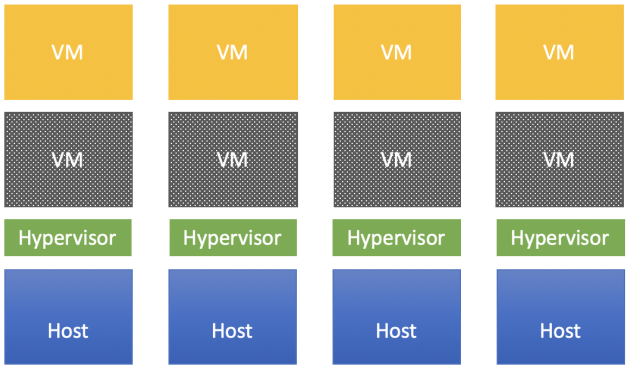


Figure 2. Illustration of VTC on four nodes for two tenants.

3.3 Dynamic VM migration

It's very restricting that one can configure more VM memory but not able to consume all of them at the same time. This constraint applies to every node in the cluster because if memory is over stressed on any node, the progress of the whole workload is impacted. Fortunately, dynamic VM migration can be applied to relax the constraint [13, 12]. It is often the case that nodes in an HPC cluster have varying memory load [11]. While some nodes have memory contentions between VMs, other nodes may have a decent amount of free memory. In such cases, VTC with memory over-commitment is much more promising because VMs can be dynamically migrated based on load changes to spread the memory load evenly across the physical cluster. Ideally, none of the nodes will experience sustained memory pressure as long as the total active memory does not exceed the physical cluster memory capacity. This essentially relaxes the previous per-node constraint to a constraint per cluster, which has much more room for memory pressure tolerance.

4 Performance Evaluation

Extensive experiments have been done to validate our design of VTC as well as evaluate its performance under various scenarios. This section covers our testbeds, experiment methodologies, performance tuning, results, and analysis.

4.1 Experiment setup

The experiments are conducted in two phases. In the first phase, only CPU is over-committed to validate the design

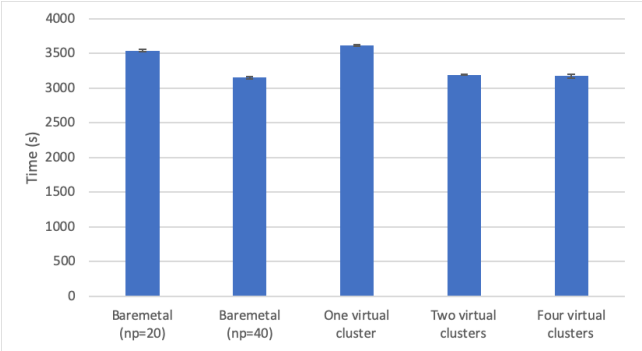


Figure 3. Comparison of wall clock execution time for a job stream between bare metal and VTC with different CPU over-commitment ratios. Results are average of three runs.

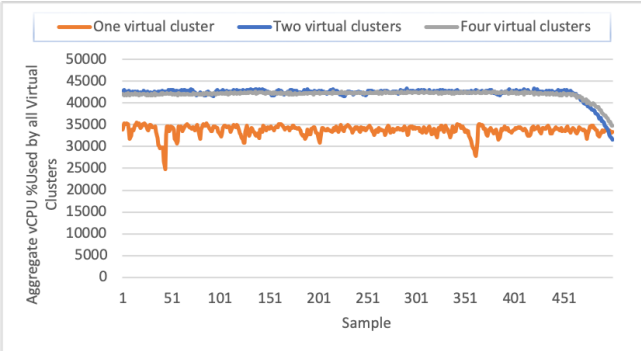


Figure 4. Aggregate CPU utilization across 16 nodes at 5-seconds intervals.

of VTC with over-commitment. In this phase, we used a cluster of 18 nodes (1 login node, 1 management node, and 16 compute nodes) to mimic a realistic HPC computing environment. Each node has dual 10-core Intel Xeon E5-2660 processor and 128 GB DDR4 memory. The hypervisor used is VMware ESXi 6.5, OS is CentOS 7.3, and resource manager is TORQUE 6.1 with the default job scheduler. We use compute-intensive benchmarks from PolyBench/C 3.1 and BioPerf [4]. In the second phase, we evaluate the full-fledged VTC design with both CPU and memory over-commitment on another testbed with huge memory capacity for memory-intensive workloads. The second cluster has 1 login, 1 management, and 8 compute nodes. Each node has dual 16-core Intel Xeon Gold 6130 processor and 768 GB memory. The hypervisor used is VMware ESXi 6.7, OS is CentOS 7.6, and resource manager is TORQUE 6.1 with the default job scheduler. We add memory-intensive benchmarks from HPCC [9].

4.2 CPU over-commitment

For performance evaluation, each node is installed with two execution environments on two separate booting disks, that is, a native OS and an ESXi hypervisor. With the hypervisor, we further define three VTC scenarios: 1) one virtual cluster; 2) two virtual clusters; 3) four virtual clusters. The latter two scenarios have 2X and 4X CPU over-commitment, respectively. When comparing performance among different execution scenarios (including bare metal cluster), we run the same job stream which consists of 3248 jobs that are randomly sampled from the PolyBench and BioPerf suites, and the wall clock execution time is shown in Figure 3.

Because each node in our testbed has 20 cores, in the first experiment we configured each TORQUE worker with 20 job slots. As reflected in Figure 3, the execution with one virtual cluster is very close to that of bare metal (first column), with only a 2.2 percent overhead. Furthermore, when multiple virtual clusters are used with CPU over-commitment, the execution time is surprisingly shorter, implying an improved throughput despite the virtualization overhead. Through careful analysis, we identified that the throughput improvement can mainly be attributed to increased CPU utilization when more jobs are concurrently scheduled to execute. This has been verified by modifying each TORQUE worker in the bare-metal environment to use 40 job slots, after which the throughput is improved to the same level as the over-committed virtual environment. This can be seen in the second column of Figure 3.

Besides execution time, another performance metric that we monitored is the total CPU utilization across the whole cluster. On the ESXi hypervisor, we used esxtop running on each compute node to sample CPU utilization of all VMs at 5-second intervals, and the results for one, two, and four virtual clusters are shown in Figure 4. With two and four virtual clusters, the decreasing trend at the end is due to job completion. Clearly, CPU utilization is consistent with the job execution times in Figure 3. For example, the lowest CPU utilization case–one virtual cluster–matches the longest job execution time. The higher CPU utilization with two and four virtual clusters is due to resource consolidation brought about by virtualization. That is, when multiple virtual clusters share a physical cluster, the CPU scheduler on each ESXi host has more jobs to schedule and is therefore able to make better scheduling decisions by taking advantage of Hyper-Threading and eliminating idle cycles. At the same time, improved utilization comes with better consistency, where the utilization with two and four virtual clusters is much smoother than in the single-cluster case.

In a production cloud environment, an important principle is fairness when multiple tenants are sharing the computing resources. We further examined the per-cluster CPU utilization in the multiple-cluster cases, and the case of four virtual clusters is shown in Figure 5. It is clear that the ESXi scheduler effectively maintains fairness so that each virtual cluster gets the same amount of CPU resources. The case of two virtual clusters is the same thus not discussed.

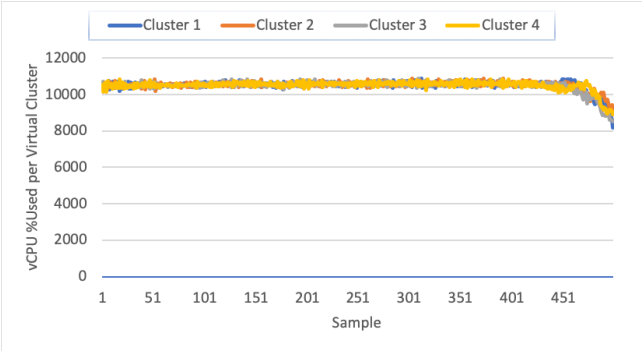


Figure 5. Plot of CPU utilization per virtual cluster at 5-seconds intervals for fairness check.

Figure 6. CPU utilization for four virtual clusters with 2:1:1:1 CPU shares. Utilization is sampled at 5-second intervals.

4.3 CPU over-commitment with shares

In addition to the option of equally sharing resources, the proportional, share-based scheduler of the ESXi hypervisor offers a very useful degree of flexibility. This subsection continues the CPU over-commitment study by configuring virtual clusters with different shares, to demonstrate the capability of creating a multi-tenant environment with quality-of-service guarantees.

With the use of CPU shares, each VM can be given a particular guaranteed share of CPU resources. When there are multiple VMs running on an ESXi host, the ESXi scheduler allocates CPU based on the ratio of shares among all the running VMs. This feature extends to a virtualized HPC cloud. Specifically, each user or group can be given an appropriate share of the physical system when their virtual cluster is allocated. In this experiment, we focus on the case of four virtual clusters and set the shares ratio among the four VMs as 2:1:1:1 on every compute node. The CPU utilization is shown in Figure 6. Clearly, the CPU utilization ratio among the virtual clusters is the same as the specified shares ratio. Another important observation is that the aggregate CPU utilization across 4 virtual clusters is the same as previous no-shares case as depicted in Figure 4.

4.4 CPU + memory over-commitment

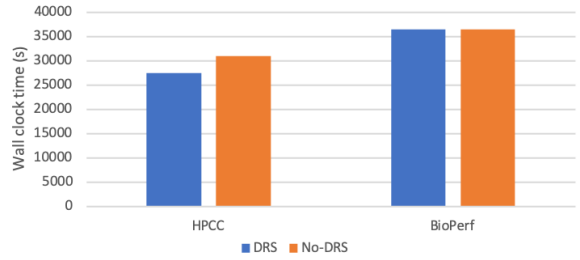
Given that memory over-commitment is more challenging, we only test two virtual clusters with 2X CPU and memory over-commitment. vSphere Dynamic Resource Scheduler (DRS) is used to dynamically control VM migration for load balancing [12]. Considering the VM migration cost, we decided to use four quarter-size VMs to replace the single full-size VM on each node for each tenant. As a result, each tenant in this experiment gets a virtual cluster of 32 VMs across the 8 node cluster.

To stress the system memory, we added benchmarks from HPCC. For all these HPCC benchmarks, a problem size of 53664 is chosen to have the memory consumption ranging from 16 GB to 22 GB per job instance. All the benchmarks are run with a single threaded process to mimic throughput workloads. To model a real HPC tenant, we specify the job arrival time to follow a Gamma distribution based on the study of several production HPC systems [14]. The resulted mean job inter-arrival time is 20.88 seconds

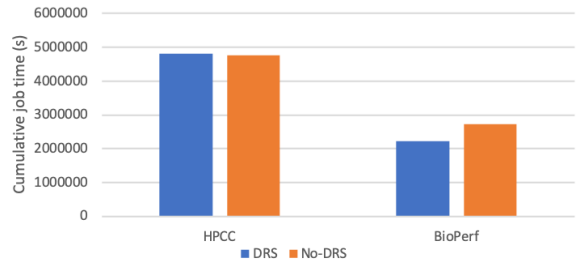
Two execution scenarios are designed to represent different workload patterns. In the first scenario, 1600 HPCC jobs (memory-intensive) are submitted to the first virtual cluster all at the beginning, while 1500 BioPerf jobs (memory-light) arrive at the second virtual cluster following the above Gamma distribution. The number of jobs are determined to let the two virtual clusters finish at roughly the same time. In the first virtual cluster, all the job slots will be consumed immediately and the jobs will maximize their utilization of the cluster's CPU and memory resources. In the second cluster, CPU and memory load will gradually build up. We make the second scenario more demanding by running two HPCC streams of 1600 jobs following the Gamma distribution. It's demanding because, even if each job only consumes 16 GB memory, 64 job instances on each node will require 64 * 16 GB = 1024 GB, which is much larger than the node capacity. It is well understood that ESXi cannot support that kind of memory over-usage. But, it is still worthwhile to explore in the realistic case where jobs randomly come and go, whether DRS can collaborate with ESXi memory reclamation techniques to accommodate this usage pattern.

In each scenario, we run w/ and w/o DRS and measure two metrics: 1) wall clock time (WCT), i.e., the elapsed time from the start of job submission to the completion of the last job; 2) cumulative job execution time (CJET), i.e., the cumulative sum of all job instances' execution time.

The results in scenario 1 are plotted in Figure 7. As the figures suggest, while VTC successfully supported 2X CPU and memory over-commitment regardless of DRS, DRS reduces HPCC's WCT by 11.1%, which is a great improvement in throughput. The reason why DRS doesn't reduce BioPerf's WCT is that BioPerf jobs strictly follow the Gamma distribution in job arrivals. But as we can see in Figure 7b, DRS reduces BioPerf's CJET by 18.4%, making room available for



(a) Wall clock time



(b) Cumulative job execution time

Figure 7. Comparison between DRS enabled and DRS disabled for VTC with 2X CPU and memory over-commitment. Workloads are from scenario 1.

more jobs. HPCC's CJET slightly increases with DRS and it indicates that DRS can cause minimal overhead to individual jobs due to telemetry sampling and VM migration. The number of VM migrations in each run is in the range of 30-40. It's interesting to notice that all migrations happened to the BioPerf VMs, which is because BioPerf VMs have smaller memory footprint and are lighter to migrate.

As mentioned above, the second scenario is much more demanding because HPCC jobs can potentially consume all the configured VM memory. It's not a surprise the cluster is not able to finish two simultaneous streams of HPCC jobs, regardless of whether DRS is used. The guest OS encountered CPU soft lockup errors when hypervisor swapping occurs and when swapping is not responsive enough. Our analysis identified HPL (one benchmark in the HPCC suite) jobs as the bottleneck. Each HPL job needs more than 3 hours to finish, and once enough HPL instances accumulate on any node, the hypervisor is not able to handle their excessive memory requirements. Therefore, we decided to remove HPL from the job stream. After this change, we see that two HPCC streams can finish when DRS is on. Without DRS, the same failure occurs. Clearly, this demonstrates the effectiveness of DRS in load balancing and mitigating memory pressure.

Though HPCC VMs are heavier to migrate, on average 67 VM migrations occurred per run, due to the extreme memory stress. Naturally, one may concern that the large number of migrations could introduce jitters to the running workloads. To quantify that, we collect the execution time

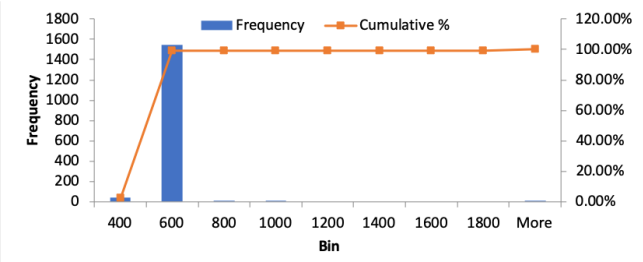


Figure 8. Histogram of individual RandomAccess job execution time from both virtual clusters.

distribution among all instances for each benchmark. It turns out that the execution time is quite consistent. For example, the histogram for RandomAccess (another benchmark in the HPCC suite) is shown in Figure 8.

5 Conclusion

With the rapid growth of AI/ML applications, HPC is increasingly changing our daily life. At the same time, in today's digital, data-driven world, cloud computing is transforming HPC by providing ubiquitous access and virtually unlimited scalability. In this paper, we present and validate an innovative design of resource management to enhance resource utilization while guaranteeing QoS for HPC cloud. This is pioneering work to apply CPU plus memory over-commitment to HPC workloads with comprehensive performance study on its feasibility and limitation. In the future, we plan to extend the scope of this work to cover more workloads, such as MPI workloads, and to build analytical models to predict the trade-off between performance and over-commitment degree based on workload characterization [3, 23].

References

[1] 2013. *The CPU Scheduler in VMware vSphere 5.1.* Technical Report. VMware.

[2] Mainak Adhikari and Tarachand Amgoth. 2018. Heuristic-based load-balancing algorithm for IaaS cloud. *Future Generation Computer Systems* 81 (2018), 156–165.

[3] Martin F Arlitt and Carey L Williamson. 1997. Internet web servers: Workload characterization and performance implications. *IEEE/ACM Transactions on networking* 5, 5 (1997), 631–645.

[4] D. A. Bader, Yue Li, Tao Li, and V. Sachdeva. 2005. BioPerf: a benchmark suite to evaluate high-performance computer architecture on bioinformatics applications. In *IEEE International. 2005 Proceedings of the IEEE Workload Characterization Symposium, 2005.* 163–173. https://doi.org/10.1109/IISWC.2005.1526013

[5] Ishan Banerjee, Fei Guo, Kiran Tati, and Rajesh Venkatasubramanian. 2013. Memory overcommitment in the ESX server. *VMware Technical Journal* 2, 1 (2013), 2–12.

[6] Edouard Bugnion, Jason Nieh, and Dan Tsafrir. 2017. Hardware and software support for virtualization. *Synthesis Lectures on Computer Architecture* 12, 1 (2017), 1–206.

[7] Steve Conway, Alex Norton, Bob Sorensen, and Earl Joseph. 2019. *Cloud Computing for HPC Comes of Age.* Technical Report. HYPERION RESEARCH.

[8] Xiaolong Cui and Josh Simons. 2018. *Virtualizing HPC Throughput Computing Environments*. Technical Report. VMware.

[9] Jack J Dongarra and Piotr Luszczek. 2004. *Introduction to the hpc-challenge benchmark suite*. Technical Report. TENNESSEE UNIV KNOXVILLE.

[10] Sukhpal Singh Gill, Inderveer Chana, Maninder Singh, and Rajkumar Buyya. 2018. CHOPPER: an intelligent QoS-aware autonomic resource management approach for cloud computing. *Cluster Computing* 21, 2 (2018), 1203–1241.

[11] Abhishek Gupta, Osman Sarood, Laxmikant V Kale, and Dejan Milojicic. 2013. Improving hpc application performance in cloud through dynamic load balancing. In *2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*. IEEE, 402–409.

[12] VMware Infrastructure. 2006. Resource management with VMware DRS. *VMware Whitepaper* 13 (2006).

[13] R. Kanniga Devi, G. Murugaboopathi, and M. Muthukannan. 2018. Load monitoring and system-traffic-aware live VM migration-based load balancing in cloud data center using graph theoretic solutions. *Cluster Computing* 21, 3 (01 Sep 2018), 1623–1638. https://doi.org/10.1007/s10586-018-2303-z

[14] Uri Lublin and Dror G Feitelson. 2003. The workload on parallel supercomputers: modeling the characteristics of rigid jobs. *J. Parallel and Distrib. Comput.* 63, 11 (2003), 1105–1122.

[15] Rajesh S Madukkarumukumana, Gilbert Neiger, and Ioannis Schoinas. 2008. Resource partitioning and direct access utilizing hardware support for virtualization. US Patent 7,467,381.

[16] Marco AS Netto, Rodrigo N Calheiros, Eduardo R Rodrigues, Renato LF Cunha, and Rajkumar Buyya. 2018. HPC cloud for scientific and business applications: Taxonomy, vision, and research challenges. *ACM Computing Surveys (CSUR)* 51, 1 (2018), 8.

[17] Reena Panwar and Bhawna Mallick. 2015. Load balancing in cloud computing using dynamic load management algorithm. In *2015 International Conference on Green Computing and Internet of Things (ICGCIoT)*. IEEE, 773–778.

[18] Josh Simons, Edmond DeMattia, and Charu Chaubal. 2017. *Virtualizing HPC and Technical Computing with VMware vSphere*. Technical Report. VMware.

[19] Sukhpal Singh and Inderveer Chana. 2016. A survey on resource scheduling in cloud computing: Issues and challenges. *Journal of grid computing* 14, 2 (2016), 217–264.

[20] Selome Kostentinos Tesfatsion, Cristian Klein, and Johan Tordsson. 2018. Virtualization Techniques Compared: Performance, Resource, and Power Usage Overheads in Clouds. In *Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering (ICPE '18)*. ACM, New York, NY, USA, 145–156. https://doi.org/10.1145/3184407.3184414

[21] Manh-Hung Tran, Thien-Binh Dang, Vi Van Vo, Duc-Tai Le, Moonseong Kim, and Hyunseung Choo. 2019. On Analyzing the Trade-Off Between Over-Commitment Ratio and Quality of Service in NFV Datacenter. In *Future Data and Security Engineering*, Tran Khanh Dang, Josef Küng, Makoto Takizawa, and Son Ha Bui (Eds.). Springer International Publishing, Cham, 323–331.

[22] Carl A. Waldspurger. 2002. Memory Resource Management in VMware ESX Server. *SIGOPS Oper. Syst. Rev.* 36, SI (Dec. 2002), 181–194. https://doi.org/10.1145/844128.844146

[23] Xin Xu, Na Zhang, Michael Cui, Michael He, and Ridhi Surana. 2019. Characterization and Prediction of Performance Interference on Mediated Passthrough GPUs for Interference-aware Scheduler. In *11th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 19)*. USENIX Association, Renton, WA. https://www.usenix.org/conference/hotcloud19/presentation/xu-xin

[24] Zhi-Hui Zhan, Xiao-Fang Liu, Yue-Jiao Gong, Jun Zhang, Henry Shu-Hung Chung, and Yun Li. 2015. Cloud computing resource scheduling and a survey of its evolutionary approaches. *ACM Computing Surveys (CSUR)* 47, 4 (2015), 63.