Slurm-V: Extending Slurm for Building Efficient HPC Cloud with SR-IOV and IVShmem

Jie Zhang(✉), Xiaoyi Lu, Sourav Chakraborty,
and Dhabaleswar K. (DK) Panda

Department of Computer Science and Engineering,
The Ohio State University, Columbus, USA
{zhanjie,luxi,chakrabs,panda}@cse.ohio-state.edu

Abstract. To alleviate the cost burden, efficiently sharing HPC cluster resources to end users through virtualization is becoming more and more attractive. In this context, some critical HPC resources among Virtual Machines, such as Single Root I/O Virtualization (SR-IOV) enabled Virtual Functions (VFs) and Inter-VM Shared memory (IVShmem) devices, need to be enabled and isolated to support efficiently running multiple concurrent MPI jobs on HPC clouds. However, original Slurm is not able to supervise VMs and associated critical resources, such as VFs and IVShmem. This paper proposes a novel framework, **Slurm-V**, which extends Slurm with virtualization-oriented capabilities such as job submission to dynamically created VMs with isolated SR-IOV and IVShmem resources. We propose several alternative designs for Slurm-V: Task-based design, SPANK plugin-based design, and SPANK plugin over OpenStack-based design, to manage and isolate IVShmem and SR-IOV resources for running MPI jobs. We evaluate these designs from aspects of startup performance, scalability, and application performance in different scenarios. The evaluation results show that VM startup time can be reduced by up to 2.64X through snapshot scheme in Slurm SPANK plugin. Our proposed Slurm-V framework shows good scalability and the ability of efficiently running concurrent MPI jobs on SR-IOV enabled InfiniBand clusters. To the best of our knowledge, Slurm-V is the first attempt to extend Slurm for the support of running concurrent MPI jobs with isolated SR-IOV and IVShmem resources. The capabilities of Slurm-V can be used to build efficient HPC clouds.

1 Introduction

To meet the increasing demand for computational power, HPC clusters have grown tremendously in size and complexity. Efficient sharing of such HPC resources is becoming more important to achieve faster turnaround time and lower the cost per user. Furthermore, a large number of users experience large variability in workloads depending on business needs, which makes predicting

This research is supported in part by National Science Foundation grants #CNS-1419123, #IIS-1447804, #ACI-1450440, and #CNS-1513120.

P.-F. Dutot and D. Trystram (Eds.): Euro-Par 2016, LNCS 9833, pp. 349–362, 2016.
DOI: 10.1007/978-3-319-43659-3_26

the required resources for future workloads a difficult task. Therefore, virtualized HPC clusters can be an attractive solution that can offer on-demand resource acquisition, high configurability while delivering near bare-metal performance at a low cost.

While virtualization technology has come a long way since its inception, achieving near-native performance for latency-critical HPC application remains a challenge to this date. A significant bottleneck exists in the virtualized I/O subsystem, which is one of the biggest hindrances to large scale adoption of virtualization in the HPC community. The recently introduced Single Root I/O Virtualization (SR-IOV) [3] technology for InfiniBand and High Speed Ethernet is quickly changing the landscape by providing native I/O virtualization capabilities [12]. Through SR-IOV, a single physical device, or a Physical Function (PF), can be presented as multiple virtual devices, or Virtual Functions (VFs). However, our previous studies [10] have shown that SR-IOV lacks support for locality-aware communication, which leads to performance overheads for inter-VM communication within the same physical node. In this context, Inter-VM Shared Memory (IVShmem) [15] has been proposed and can be hot-plugged to a VM as a virtualized PCI device to support shared memory backed intra-node-inter-VM communication. The performance improvements enabled by SR-IOV and IVShmem have contributed to their adoption by the HPC community. For example, the MVAPICH2 MPI library is able to take advantage of SR-IOV and IVShmem to deliver near-native performance for MPI applications [9,10,18].

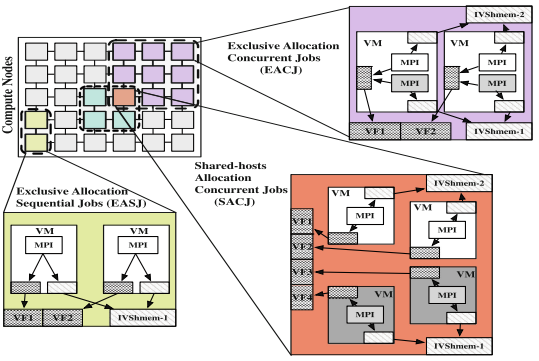


Fig. 1. Different scenarios of running MPI jobs over VMs on HPC cloud

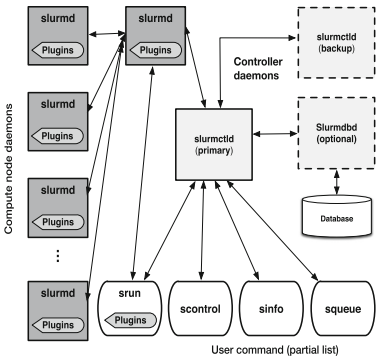


Fig. 2. Slurm architecure

1.1 Motivation

For improved flexibility and resource utilization, it is important to manage and isolate virtualized resources of SR-IOV and IVShmem to support running multiple concurrent MPI jobs. As this requires knowledge of and some level of control over the underlying physical hosts, it is difficult to achieve this with the MPI

library alone, which is only aware of the virtual nodes and resources inside. Thus, extracting the best performance from virtualized clusters require support from other middleware like job launchers and resource managers, which have a global view of the VMs and the underlying physical hosts. Figure 1 illustrates three possible scenarios of running MPI jobs over VMs in shared HPC clusters. **Exclusive Allocation for Sequential Jobs (EASJ):** Users exclusively allocate the physical nodes and add dedicated SR-IOV and IVShmem devices for each VM to sequentially run MPI jobs. This scenario requires co-resident VMs select different Virtual Functions, like VF1 and VF2, and add virtualized PCI devices mapping to the same IVShmem region, like IVShmem-1 as shown in Fig. 1. **Exclusive Allocation for Concurrent Jobs (EACJ):** Users get exclusive allocations, but multiple IVShmem devices, like IVShmem-1 and IVShmem-2 in Fig. 1 need to be added to each VM for multiple MPI jobs running concurrently. Because each MPI job at least needs one IVShmem device on one host to support Inter-VM shared memory based communications. **Shared-hosts Allocation for Concurrent Jobs (SACJ):** In shared HPC clusters, different users might allocate VMs on the same physical node. Each VM needs to have a dedicated SR-IOV virtual function, like VF1 to VF4. And IVShmem devices in different users' VMs need to point to different shared memory regions on the physical node, like IVShmem-1 and IVShmem-2 in Fig. 1.

Unfortunately, to the best of our knowledge, none of the currently available studies on resource managers such as Slurm [6,11,16] are SR-IOV and IVShmem aware. Therefore, they are not able to handle the above three scenarios of running MPI jobs. Moreover, one of the major contributors to the increasing popularity of virtual cluster computing is OpenStack [2]. It provides scalable and efficient mechanisms for creation, deployment, and reclamation of VMs on a large number of physical nodes. This offers us with further optimization opportunities - by integrating OpenStack with Slurm, which might be possible to drastically reduce the required interaction and turnaround time for a user attempting to utilize a virtualized cluster. To achieve the above goals, the following challenges need to be addressed:

- Can Slurm be extended to manage and isolate SR-IOV and IVShmem resources for running concurrent MPI jobs efficiently?
- What kind of design alternatives be proposed to achieve better deployment/job launching times as well as application performance?
- Can Slurm and OpenStack be combined to provide a scalable solution for building efficient HPC clouds?
- Can MPI library running on the extended Slurm with SR-IOV and IVShmem support provides bare-metal performance for end HPC applications on different scenarios?

1.2 Contributions

To address the above challenges, this paper proposes a framework, called **Slurm-V**, which extends Slurm to manage and isolate SR-IOV and IVShmem

resources for running MPI applications concurrently on virtual machines. In the proposed Slurm-V, three new components are introduced: VM Configuration Reader, VM Launcher and VM Reclaimer. To support these components, we propose three alternative designs: Task-based design, SPANK plugin-based design, and SPANK plugin over OpenStack-based design. We evaluate these designs from various aspects such as startup time, scalability, and application performance. Our evaluations show that our proposed Slurm-V framework has good deployment performance and scalability. With the proposed designs, VM startup time can be reduced by up to 2.64X through snapshot scheme in Slurm SPANK plugin. The sequential and concurrent MPI jobs can be efficiently executed on shared HPC clusters while maintaining minor overhead.

To the best of our knowledge, our proposed Slurm-V is the first attempt to extend Slurm for the support of running concurrent and sequential MPI jobs with isolated SR-IOV and IVShmem resources. The capabilities of Slurm-V can be used to build efficient HPC Clouds with SR-IOV and IVShmem.

2 Background

2.1 Slurm and SPANK

Simple Linux Utility for Resource Management (Slurm) [17] is an open-source resource manager for large scale Linux based clusters. Slurm can provide users with exclusive and/or shared access to cluster resources. As shown in Fig. 2, Slurm provides a framework including controller daemons (`slurmctld`), database daemon (`slurmdbd`), compute node daemons (`slurmd`), and a set of user commands (e.g. `srun`, `scontrol`, `squeue`) to start, execute and monitor jobs on a set of allocated nodes and manage a queue of pending jobs. Slurm Plug-in Architecture for Node and job (K)control (SPANK) [4] provides a generic interface to be used for dynamically modifying the job launch code. SPANK plugins have the ability to add user options when using `srun`. It may be built without accessing Slurm source code and will be automatically loaded at the next job launch. Thus, SPANK provides a low-cost and low-effort mechanism to change runtime behavior of Slurm.

2.2 SR-IOV and IVShmem

Single Root I/O Virtualization (SR-IOV) [3] is a new PCI Express technology, which specifies the native I/O virtualization capabilities in PCIe adapters. A single Physical Function (PF) can present itself as multiple Virtual Functions (VFs) through SR-IOV. Each VF can be passthroughed to a single VM. However, an efficient management mechanism is required to detect and select an exclusive VF for each VM. Inter-VM Shared Memory (IVShmem) (e.g. Nahanni) [15] provides zero-copy access to data residing on VM shared memory on the KVM platform. The host shared memory region is exposed to VM by serving as a virtualized PCI device in VM. Thus, shared memory based communication can be executed

between processes in co-resident VMs. However, the difference from host shared memory is that IVShmem device does not support hierarchical file structure. To support multiple concurrent MPI jobs, multiple IVShmem devices need to be provided accordingly. Therefore, managing and isolating IVShmem devices among different concurrent MPI jobs is critical.

2.3 OpenStack

OpenStack [2] is an open-source middleware for cloud computing that controls large pools of computing, storage, and networking resources. It provides several components, such as Nova, Neutron, Glance, etc. to efficiently manage and quickly deploy cluster resources. OpenStack can work with many available virtualization technologies. It has been widely deployed in many private and public cloud environments.

3 Proposed Design

3.1 Architecture Overview of Slurm-V

This section presents an overview of Slurm-V framework. As we can see in Fig. 3, it is based on the original architecture of Slurm. It has a centralized manager, Slurmctld, to monitor work and resources. Each compute node has a Slurm daemon, which waits for the task, executes that task, returns status, and waits for more tasks [17]. Users can put their physical resource requests and computation tasks in a batch file, submit it by `sbatch` to the Slurm control daemon, Slurmctld. Slurmctld will respond with the requested physical resources according to its scheduling mechanism. Subsequently, the specified MPI jobs are executed on those physical resources.

In our framework Slurm-V, three new components are integrated into the current architecture. The first component is VM Configuration Reader, which extracts the related parameters for VM configuration. Each time when users request physical resources, they can specify the detailed VM configuration information, such as vcpu-per-vm, memory-per-vm, disk-size, vm-per-node, etc. In order to support high performance MPI communication, the user can also specify SR-IOV devices on those allocated nodes, and the number of IVShmem devices which is the number of concurrent MPI jobs they want to run inside VMs. The VM Configuration Reader will parse this information, and set them in the current Slurm job control environment. In this way, the tasks executed on those physical nodes are able to extract information from job control environment and take proper actions accordingly. The second component is the VM Launcher, which is mainly responsible for launching required VMs on each allocated physical node based on user-specified VM configuration. The zoom-in box in Fig. 3 lists the main functionalities of this component. If the user specifies the SR-IOV enabled device, this component detects those occupied VFs and selects a free one for each VM. It also loads user-specified VM image from the publicly accessible storage system, such as NFS or Lustre, to the local node. Then it generates

XML file and invokes libvirtd or OpenStack infrastructure to launch VM. During VM boot, the selected VF will be passthroughed to VM. If the user enables the IVShmem option, this component assigns a unique ID for each IVShmem device, and sequentially hotplugs them to VM. In this way, IVShmem devices can be isolated with each other, such that each concurrent MPI job will use a dedicated one for inter-VM shared memory based communication. On the aspect of network setting, each VM will be dynamically assigned an IP address from an outside DHCP server. Another important functionality is that the VM Launcher records and propagates the mapping records between local VM and its assigned IP address to all other VMs. Other functionalities include mounting global storage systems, etc. Once the MPI job reaches completion, the VM Reclaimer is executed. Its responsibilities include reclaiming VMs and the critical resources, such as unlocking the passthroughed VFs, returning them to VF pool, detaching IVShmem devices and reclaiming corresponding host shared memory regions.

If OpenStack infrastructure is deployed on the underlying layer, VM Launcher invokes OpenStack controller to accomplish VM configuration, launch and destruction.

3.2 Alternative Designs

We propose three alternative designs to effectively support the three components.

Task-based Design: The three new components are treated as three tasks/steps in a Slurm job. Therefore, the end-user needs to implement corresponding scripts and explicitly insert them in the job batch file. After the job being submitted, `srun` will execute these three tasks on allocated nodes.

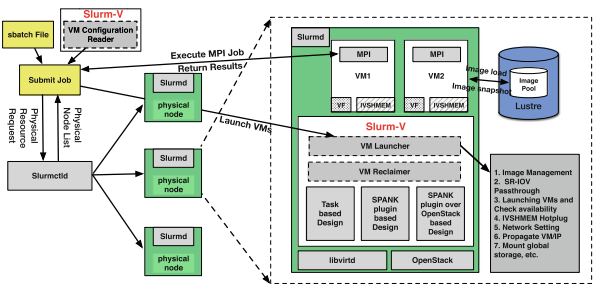


Fig. 3. Architecture overview of Slurm-V

Listing 1.1. SPANK Plugin-based Script

```
1  #!/bin/bash
2  #SBATCH -J Slurm-V
3  #SBATCH -N 2
4  #SBATCH -p All
5  #SBATCH --vm-per-node=2
6  #SBATCH --vcpu-per-vm=2
8  #SBATCH --disk-size=10G
10 #SBATCH --sriov-ib=1
11 #SBATCH --ivshmem=1
12 #SBATCH --num-ivshmem=1
13 #SBATCH --ivshm-sz=128M
14
15 Slurm-V-run -np 8 a.out
```

The Task-based design is portable and easy to integrate with existing HPC environments without any change to Slurm architecture. However, it is not transparent to end users as they need to explicitly insert the three extra tasks in their jobs. More importantly, it may incur some permission and security issues. VF passthrough requires that VM Launcher connects to the libvirtd instance running

with the privileged system account 'root', which in turn exposes security threats to the host system. In addition, the scripts implementation may be varied for different users. This will impact the deployment and application performance. To address these issues, we propose SPANK plugin-based design as discussed below.

SPANK Plugin-based Design: As introduced in Sect. 2.1, the SPANK plugin architecture allows a developer to dynamically extend functions during a Slurm job execution. Listing 1.1 presents an example of a SPANK plugin-based batch job in the Slurm-V framework. As we can see from line5-line13, the user can specify all VM configuration options as inherent ones preceded with #SBATCH. The `Slurm-V-run` on line15 is a launcher wrapper of `srun` for launching MPI jobs on VMs. Also, there is no need to insert extra tasks in this job script. Thus, it is more transparent to the end user compared to the Task-based design. Once the user submits the job using `sbatch` command, the SPANK plugin is loaded and the three components are invoked in different contexts.

Figure 4(a) illustrates the workflow of the SPANK plugin-based design in detail under the Slurm-V framework. Once the user submits the batch job request, SPANK plugin is loaded, and `spank_init` will first register all VM configuration options specified by the user and do a sanity checking for them locally before sending to the remote side. Then, `spank_init_post_opt` will set these options in the current job control environment so that they are visible to all Slurmd daemons on allocated nodes later. Slurmctld identifies requested resources, environment and queues the request in its priority-ordered queue. Once the resources are available, Slurmctld allocates resources to the job and contacts the first node in the allocation for starting user's job. The Slurmd on that node responds to the request, establishes the new environment, and initiates the user task specified by `srun` command in the launcher wrapper. `srun` connects to Slurmctld to request a job step and then passes the job step credential to Slurmds running on allocated nodes.

After exchanging the job step credential, SPANK plugin is loaded on each node. During this process, `spank_task_init_privileged` is invoked to execute VM Launcher component in order to setup VM for the following MPI job. `spank_task_exit` is responsible for executing VM Reclaimer component to tear down VMs and reclaim resources. In this design, we utilize the file-based lock mechanism to detect occupied VFs and exclusively allocate VFs from available VF pool. With this design, each IVShmem device will be assigned a unique ID and dynamically attached to VM. In this way, IVShmem devices can be efficiently isolated to support running multiple concurrent MPI jobs.

In this design, we utilize snapshot and the multi-threading mechanism to speed up the image transfer and VM launching, respectively. This will further reduce VM deployment time.

SPANK Plugin over OpenStack-based Design: This section discusses the design that combines SPANK plugin and OpenStack infrastructure. In this design, the VM Launcher and VM Reclaimer components will accomplish their functionalities by offloading the tasks to OpenStack infrastructure.

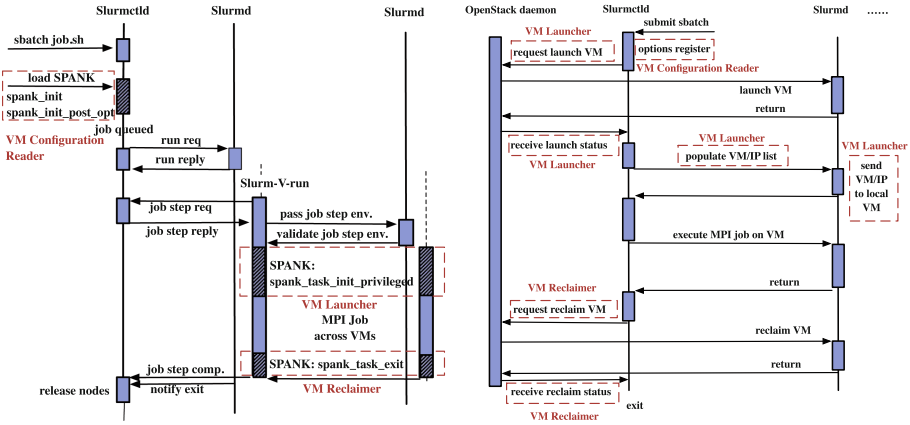


Fig. 4. SPANK Plugin-based and SPANK Plugin over OpenStack-based Design

Figure 4(b) presents the workflow of SPANK plugin over OpenStack. When the user submits a Slurm job, SPANK plugin is loaded first. VM configuration options are registered and parsed. The difference is that, on local context, VM Launcher will send a VM launch request to OpenStack daemon on its controller node. The core component of OpenStack, Nova, is responsible for launching VMs on all allocated compute nodes. Upon the launch completes, it returns a mapping list between all VM instance names and their IP addresses to VM Launcher. VM Launcher propagates this VM/IP list to all VMs. The MPI job will be executed after this. Once the result of MPI job is returned, VM Reclaimer in local context sends a VM destruction request to OpenStack daemon. Subsequently, VMs are torn down and associated resources are reclaimed in the way that OpenStack defines. In addition, our earlier work [18] describes in details about VF allocation/release and enabling IVShmem devices for VM under OpenStack framework. In this design, except VM Configuration Reader, the other two components work by sending requests to OpenStack controller and receiving its returning results. There are dedicated services in OpenStack infrastructure to manage and optimize different aspects of VM management, such as identification, image, networking. Therefore, the SPANK plugin over OpenStack-based design is more flexible and reliable.

4 Performance Evaluation

4.1 Experiment Setup

Cluster-A: This cluster has four physical nodes. Each node has dual 8-core 2.6 GHz Intel Xeon E5-2670 (Sandy Bridge) processors with 32 GB RAM and equipped with Mellanox ConnectX-3 FDR (56 Gbps) HCAs. **Chameleon:** [1]

It has eight physical nodes, each with 24 cores delivered in dual socket Intel Xeon E5-2670 v3 (Haswell) processors, 128 GB RAM and equipped with Mellanox ConnectX-3 FDR (56 Gbps) HCAs as well.

CentOS Linux 7 (Core) 3.10.0-229.el7.x86_64 is used as both host and guest OS. In addition, we use KVM as the Virtual Machine Monitor (VMM), and Mellanox OpenFabrics MLNX_OFED_LINUX-3.0-1.0.1 to provide the InfiniBand interface with SR-IOV support. Our Slurm-V framework is based on Slurm-14.11.8. MVAPICH2-Virt library is used to conduct application experiments.

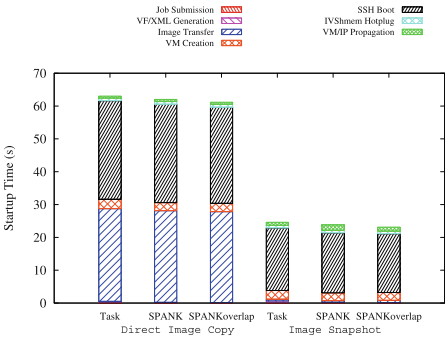


Fig. 5. VM launch breakdown on Cluster-A

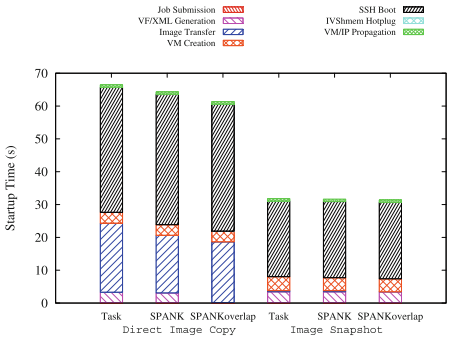


Fig. 6. VM launch breakdown on Chameleon

4.2 Startup Performance

To analyze and optimize the startup performance of the Slurm-V framework, we break down the whole VM startup process into several parts. Table 1 describes the time period of each part.

Overlapping: We found that image transfer is independent of VF/XML generation, so they can start simultaneously after submitting the job. As shown in Figs. 5 and 6, the time spent on direct image copy (2.2 GB) is larger than the time spending on VF selection and XML generation. So it can be completely overlapped. The overlapping effect can be clearly observed between SPANK and SPANKoverlap under direct image copy scheme on Chameleon.

Snapshot: We also observe that direct image copy takes a large proportion of the whole VM startup time for any startup methods on both Cluster-A and Chameleon. In order to shorten the time of image transfer, the external snapshot mechanism is applied. The original image file that user specified will be in a read-only saved state. The new file created using external snapshot will be the delta for the changes and take the original image as its backup file. All the changes from here onwards will be written to this delta file. Instead of transferring a large-size image file, we only create a small-size snapshot file for each VM, which clearly reduces the image transfer time. In addition, the backup file can be read in

Table 1. VM startup breakdown

Part	Time period description
Job submission	From submitting sbatch job to starting VM configuration
VF/XML generation	Reading VM configurations, selecting available VF to generate XML
Image transfer	Transferring VM image from public location to store location of each VM
VM creation	Time between invoking libvirt API to create VM and its return
SSH boot	Booting VM, getting available IP address until starting SSH service
IVShmem hotplug	Time of completing IVShmem hotplug operation
VM/IP propagation	Propagating VMs' hostname/IP records to all VMs

parallel by running VMs. Therefore, the snapshot mechanism enhances the VM startup performance significantly. The evaluation result shows that the whole VM startup time is shortened by up to 2.64X and 2.09X on Cluster-A and Chameleon, respectively.

Total VM Launch Time: We discussed the SPANK plugin over OpenStack-based design in Sect. 3.2. As VM Launcher offloads its task to OpenStack infrastructure as a whole task, we do not breakdown timings within the OpenStack operations. The evaluation results show that the total VM launch times are 24.6 s, 23.8 s, and 20.2 s for SPANK plugin-based design, SPANK plugin-based design with overlap and SPANK plugin over OpenStack-based design, respectively. Compared to other designs, SPANK plugin over OpenStack has better total VM launch time, which is around 20 s. This is because OpenStack, as a well-developed and relatively mature framework, has integrated optimizations on different steps of VM launch.

4.3 Scalability

In this section, we evaluate the scalability of proposed Slurm-V framework using single-threading (ST) and multi-threading (MT) schemes. In the evaluation, snapshot with overlapping is used for both schemes. In MT case, each thread is responsible for launching one VM. From Figs. 7 and 8, it can be observed that MT scheme significantly improves the VM startup performance, compared to ST scheme on both Cluster-A and Chameleon. For instance, to launch 32 VMs across 4 nodes on Chameleon, ST scheme takes 260.11 s, while MT only spends 34.88 s. Compared with ST scheme, MT scheme reduces the VM startup time by up to 86 % and 87 % on Cluster-A and Chameleon, respectively. As the number of physical nodes increases, we do not see the clear increase for startup time of MT scheme. These results indicate that our proposed Slurm-V framework scales well.

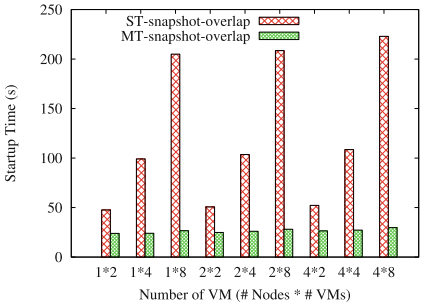


Fig. 7. Scalability study on Cluster-A

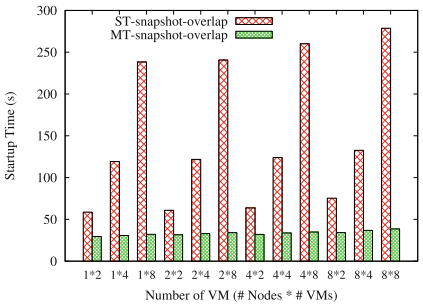
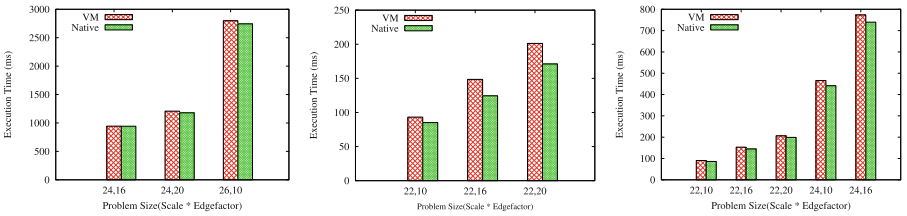


Fig. 8. Scalability study on Chameleon

4.4 Application Performance

The Slurm-V framework extends Slurm to manage and isolate virtualized resources of SR-IOV and IVShmem to support running multiple concurrent MPI jobs under different scenarios. In this section, we evaluate the Graph500 performance under three scenarios (EASJ, EACJ, and SACJ) as indicated in Sect. 1.1 with 64 processes across 8 nodes on Chameleon. Each VM is configured with 6 cores and 10 GB RAM.

For **EASJ**, two VMs are launched on each node. Figure 9(a) shows the Graph500 performance with 64 processes on 16 VMs in this scenario. The evaluation results indicate that the VM launched by Slurm-V with SR-IOV and IVShmem support can deliver near-native performance, with less than 4 % overhead. This is because the Slurm-V framework is able to efficiently isolate SR-IOV VFs and enable IVShmem device across co-resident VMs. Co-resident VMs can execute shared memory based communication through IVShmem device. On the other hand, each VM with the dedicated VF can achieve near-native inter-node communication performance. For **SACJ**, four VMs VM(0–3) are launched on each node. Graph500 is executed across all VM(0–1), while the second MPI job is executed across all VM(2–3) simultaneously. We run NAS as the second MPI jobs. For the native case, we use 8 cores corresponding to VM(0–1) to run Graph500, while another 8 cores corresponding to VM(2–3) to run the second job. As shown in Fig. 9(b), the execution time of Graph500 on VM is similar with the native case with around 6 % overhead. This indicates that the Slurm-V framework is able to efficiently manage and isolate the virtual resource of SR-IOV and IVShmem on both VM and user level, although in the shared allocation. One dedicated VF is passthroughed to each VM and one unique IVShmem device is attached to all co-resident VMs of each user. For **EACJ**, similarly, our Slurm-V framework can also deliver the near-native performance, with around 8 % overhead, as shown in Fig. 9(c). The Slurm-V framework supports the management and isolation of IVShmem on MPI job level, so each MPI job can have a unique IVShmem device to execute shared memory backend communication across the co-resident VMs.



(a) EASJ (b) SACJ (c) EACJ

Fig. 9. Graph500 performance with 64 processes on different scenarios

From these application studies, we see that VMs deployed by Slurm-V with appropriately managed and isolated SR-IOV and IVShmem resources are able to deliver high performance for concurrent MPI jobs, which can be seen as promising results for running applications on shared HPC clouds.

5 Related Work

For building cloud computing environments with Slurm, Jacobsen et al. [11] present 'shifter' tightly integrated into Slurm for managing Docker and other user-defined images. Ismael [6] uses VM for dynamic fractional resource management and load balancing in a batch cluster environment. Markwardt et al. [16] propose a solution to run VMs in a Slurm-based batch system. They use a VM scheduler to keep track of the status of Slurm queue on the VMs. For building HPC cloud environments, studies [7,8,13] with Xen demonstrate the ability to achieve near-native performance in VM-based environment for HPC applications. Ruivo et al. [5] explore the potential use of SR-IOV on InfiniBand in an Open Nebula cloud towards the efficient support of MPI-based workloads. Our previous evaluation [10] has revealed that IVShmem can significantly improve intra-node inter-VM communication on SR-IOV enabled InfiniBand clusters. Further, we redesigned MVAPICH2 library [9] to take advantage of this feature and proposed an efficient approach [18] to build HPC clouds by extending OpenStack with redesigned MVAPICH2 library. However, none of these has discussed how to effectively manage and isolate IVShmem and SR-IOV resources in shared HPC cluster under Slurm framework in order to support running MPI jobs in different scenarios as indicated in Sect. 1.1. The initial idea of this work had been presented in Slurm Forum [14], and we further complete the whole Slurm-V design, implementation, and evaluation in this paper.

6 Conclusion and Future Work

In this paper, we proposed a novel Slurm-V framework to efficient support running multiple concurrent MPI jobs with SR-IOV and IVShmem in shared HPC clusters. The proposed framework extends Slurm architecture and introduces three new

components: VM Configuration Reader, VM Launcher, and VM Reclaimer. We present three alternative designs to support these components, which are: Task-based design, SPANK plugin-based design and SPANK plugin over OpenStack-based design. We evaluate our Slurm-V framework from different aspects including startup performance, scalability and application performance under different scenarios. The evaluation results indicate that the VM startup time can be reduced by up to 2.64X by using snapshot scheme. Compared with the single-threading scheme, multi-threading scheme reduces the VM startup time by up to 87 %. In addition, Slurm-V framework shows good scalability and is able to support running multiple MPI jobs under different scenarios on HPC clouds. In the future, we plan to explore other alternative SPANK-based designs to further extend Slurm framework to have more virtualization support.

References

1. Chameleon. http://chameleoncloud.org/
2. OpenStack. http://openstack.org/
3. PCI-SIG Single-Root I/O Virtualization Specification. http://www.pcisig.com/specifications/iov/
4. SPANK - Slurm Plug-in Architecture for Node and job (K)control. http://slurm.schedmd.com/spank.html
5. De Lacerda Ruivo, T., Altayo, G., Garzoglio, G., Timm, S., Kim, H.W., Noh, S.Y., Raicu, I.: Exploring infiniband hardware virtualization in OpenNebula towards efficient high-performance computing. In: 2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), pp. 943–948 (2014)
6. Estrada, I.F.: Overview of a Virtual Cluster using OpenNebula and SLURM
7. Huang, W., Koop, M.J., Gao, Q., Panda, D.K.: Virtual machine aware communication libraries for high performance computing. In: Proceedings of the 2007 ACM/IEEE Conference on Supercomputing, SC 2007, pp. 9: 1–9: 12. ACM, New York (2007)
8. Huang, W., Liu, J., Abali, B., Panda, D.K.: A case for high performance computing with virtual machines. In: Proceedings of the 20th Annual International Conference on Supercomputing, ICS 2006, New York, NY, USA (2006)
9. Zhang, J., Lu, X., Jose, J., Li, M., Shi, R., Panda, D.K.: High performance MPI library over SR-IOV enabled InfiniBand clusters. In: Proceedings of International Conference on High Performance Computing (HiPC), Goa, India (2014)
10. Zhang, J., Lu, X., Jose, J., Shi, R., Panda, D.K.: Can Inter-VM shmem benefit MPI applications on SR-IOV based virtualized InfiniBand clusters? In: Silva, F., Dutra, I., Santos Costa, V. (eds.) Euro-Par 2014. LNCS, vol. 8632, pp. 342–353. Springer, Heidelberg (2014)
11. Jacobsen, D., Botts, J., Canon, S.: Never Port Your Code Again Docker functionality with Shifter using SLURM. http://slurm.schedmd.com/SLUG15/shifter.pdf
12. Jose, J., Li, M., Lu, X., Kandalla, K., Arnold, M., Panda, D.K.: SR-IOV support for virtualization on infiniband clusters: early experience. In: On 13th IEEE/ACM International Symposium Cluster, Cloud and Grid Computing (CCGrid), pp. 385–392 (2013)

13. Lu, X., Lin, J., Zha, L., Xu, Z.: Vega LingCloud: a resource single leasing point system to support heterogeneous application modes on shared infrastructure. In: Proceedings of the 2011 IEEE Ninth International Symposium on Parallel and Distributed Processing with Applications, ISPA 2011, pp. 99–106. IEEE Computer Society, Washington, DC (2011)
14. Lu, X., Zhang, J., Chakraborty, S., Subramoni, H., Arnold, M., Perkins, J., Panda, D.K.: Supporting SR-IOV and IVSHMEM in MVAPICH2 on Slurm: Challenges and Benefits. http://slurm.schedmd.com/SLUG15/mv2_virt_slug_luxi_osu.pdf
15. Macdonell, A.C.: Shared-Memory Optimizations for Virtual Machines. Ph.D. Thesis. University of Alberta, Edmonton, Alberta, Fall 2011
16. Markwardt, U., Jurenz, M., Rotscher, D., Muller-Pfefferkorn, R., Jakel, R., Wesarg, B.: Running Virtual Machines in a Slurm Batch System. http://slurm.schedmd.com/SLUG15/SlurmVM.pdf
17. Yoo, A.B., Jette, M.A., Grondona, M.: SLURM: simple linux utility for resource management. In: Feitelson, D.G., Rudolph, L., Schwiegelshohn, U. (eds.) JSSPP 2003. LNCS, vol. 2862, pp. 44–60. Springer, Heidelberg (2003)
18. Zhang, J., Lu, X., Arnold, M., Panda, D.K.: MVAPICH2 over OpenStack with SR-IOV: an efficient approach to build HPC clouds. In: 2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), pp. 71–80 (2015)