Cooperative VM Migration for a Virtualized HPC Cluster with VMM-Bypass I/O devices

Ryousei Takano, Hidemoto Nakada, Takahiro Hirofuchi, Yoshio Tanaka, and Tomohiro Kudoh
National Institute of Advanced Industrial Science and Technology (AIST)
1-1-1 Umezono, Tsukuba, Ibaraki 305-8568 Japan
Email: {takano-ryousei, hide-nakada, t.hirofuchi, yoshio.tanaka, t.kudoh}@aist.go.jp

***Abstract*—An HPC cloud, a flexible and robust cloud computing service specially dedicated to high performance computing, is a promising future e-Science platform. In cloud computing, virtualization is widely used to achieve flexibility and security. Virtualization makes migration or checkpoint/restart of computing elements (virtual machines) easy, and such features are useful for realizing fault tolerance and server consolidations. However, in widely used virtualization schemes, I/O devices are also virtualized, and thus I/O performance is severely degraded. To cope with this problem, VMM-bypass I/O technologies, including PCI passthrough and SR-IOV, in which the I/O overhead can be significantly reduced, have been introduced. However, such VMM-bypass I/O technologies make it impossible to migrate or checkpoint/restart virtual machines, since virtual machines are directly attached to hardware devices.**

This paper proposes a novel and practical mechanism, called Symbiotic Virtualization (SymVirt), for enabling migration and checkpoint/restart on a virtualized cluster with VMM-bypass I/O devices, without the virtualization overhead during normal operations. SymVirt allows a VMM to cooperate with a message passing layer on the guest OS, then it realizes VM-level migration and checkpoint/restart by using a combination of a PCI hotplug and coordination of distributed VMMs. We have implemented the proposed mechanism on top of QEMU/KVM and the Open MPI system. All PCI devices, including Infiniband and Myrinet, are supported without implementing specific para-virtualized drivers; and it is not necessary to modify either of the MPI runtime and applications. Using the proposed mechanism, we demonstrate reactive and proactive FT mechanisms on a virtualized Infiniband cluster. We have confirmed the effectiveness using both a memory intensive micro benchmark and the NAS parallel benchmark. Moreover, we also show that postcopy live migration enables us to reduce the down time of an application as the memory footprint increases.

I. Introduction

Cloud computing is the delivery of computing as a service rather than a product. Recently, cloud computing is getting increased attention from the High Performance Computing (HPC) community. To meet the demand, several systems, e.g., the Amazon EC2 Cluster Compute Instances [1], Google Compute Engine [2], and Open Cirrus's HPC on-demand [3] have been proposed. Here, we call such an "Infrastructure as a Service (IaaS)" model of cloud computing, which provides users with virtualized HPC clusters on demand, *an HPC Cloud*. By introducing cloud computing to high performance computing, all the benefits of cloud computing, such as reduced ownership cost, higher flexibility, and higher availability can be enjoyed by the users. Computer virtualization is commonly used in cloud computing infrastructure. By virtualization, each virtual machine (VM) is isolated from others, and higher security can be achieved. In addition, migration or checkpoint/restart of VMs becomes easy by virtualization. Those features are useful for realizing fault tolerance (FT), load balancing, and server consolidations, since VMs can be moved between physical computing nodes, and also intermediate state can be easily snapshotted.

Virtualization is not quite suitable for data intensive scientific computing, which requires high performance I/O for each computing element. Data intensive computing is a emerging technology especially in the fields of high energy physics, astronomy, bioinformatics, and geo science. Many researchers have reported performance evaluations of HPC applications on virtualized clusters [4], [5], [6]. These studies show virtualized clusters suffer from performance degradation due to the high overhead of virtualization, especially for I/O devices.

To cope with this problem, Virtual Machine Monitor (VMM)-bypass I/O technologies have been introduced. Nathan, et al., reported the effects of PCI passthrough [5]. Our recent work has demonstrated the I/O performance of a virtualized Infiniband cluster is comparable to that of a bare metal cluster, and shows the positive conclusion that HPC clouds are feasible [7]. However, VM migration and checkpoint/restart mechanisms cannot be used when VMM-bypass I/O technologies, including PCI passthrough and SR-IOV, are used. This is because the VMM-bypass I/O devices are directly assigned to a VM through the VMM, so the VMM cannot save and load the state of the I/O devices.

The requirements for achieving practical HPC clouds are summarized as follows: 1) bare metal comparable I/O performance during normal operations, 2) migration capability for VMs with VMM-bypass I/O devices, 3) short down time during migration, and 4) easy deployment.

In order to achieve the above requirements, a new mechanism to control VMM-bypass I/O devices on distributed VMMs, that can cooperate with resource management in a data center, is required. Therefore, we propose a *Symbiotic Virtualization (SymVirt)* mechanism, for enabling VM-level migration and checkpoint/restart on a virtualized cluster with VMM-

This work was partly supported by MEXT KAKENHI 24700040 and ARGO GRAPHICS, Inc.

978-1-4673-4466-1/12/$31.00 ©2012 IEEE

bypass I/O devices. It targets Message Passing Interface (MPI) applications. First, requirements 1) and 2) are satisfied at the same time, by using a technique based on a combination of a PCI hotplug and coordination of distributed VMMs. Second, we use postcopy live migration to address requirement 3). Third, to meet requirement 4), the implementation requires no modifications to applications and an MPI runtime inside a guest OS. We have confirmed the proposed mechanism meets these requirements on a virtualized Infiniband cluster. In addition, using the proposed mechanism, we have implemented FT mechanisms and evaluated its performance.

In this paper, 1) we show the SymVirt mechanism enables VM-level migration and checkpoint/restart on a virtualized cluster with VMM-bypass I/O devices; 2) we demonstrate a FT system based on it; and 3) we show postcopy live migration helps to reduce the down time in the proactive FT system as the memory footprint increases.

The rest of the paper is organized as follows. Section II describes the background of HPC clouds, especially focused on fault tolerance. The design and implementation of the proposed mechanism is presented in Section III. Section IV shows the experimental results, and we discuss further optimization techniques in Section V. In Section VI, we briefly mention related work. Finally, Section VII summarizes the paper.

II. Fault tolerance on HPC clouds

In HPC systems, FT is important since a large number of resources are used for a long period of time, and the possibility of encountering failures during a job is high. Some VM-level FT mechanisms and optimization techniques that rely on VM-level snapshot and migration techniques have been proposed. Traditional process-level checkpoint/restart and migration have some strict limitations [9]. For example, when restarting a process on a different node, we must ensure that the OS on all nodes supplies the exact same libraries. In contrast, VM-level FT mechanisms do not have the above limitations, and they can also be implemented independent of the underlying execution environment.

Figure 1 shows the schematic design of VM-level FT mechanisms on an HPC cloud. HPC clouds provide IaaS-type services. An HPC cloud provider manages the resources, including compute nodes, network, and storage, by using a cloud management software stack, namely, a cloud controller, e.g., CloudStack, OpenStack, or OpenNebula. A cloud controller allocates physical resources to a user, and builds a virtualized cluster top on them. The user submits an application job on the virtualized cluster. In response to FT events, e.g., hardware failures, a cloud controller re-allocates nodes in cooperation with VM-level FT mechanisms, to ensure that the application can survive the failures. Here, we assume the cloud controller schedules resources not per CPU core, but per node.

There are two FT approaches: reactive FT and proactive FT. Checkpoint/restart is a popular reactive FT approach. In Figure 1a, a set of VM-level checkpoints are taken periodically. The checkpoint images are stored in the global storage. In response to FT events at any nodes, all VMs are shutdown, and restored from the latest checkpoint images. At the time, all nodes running VMs are re-allocated. A reactive FT approach involves a high-cost overhead for users as the scale of a system increases. Since both the I/O throughput and the storage space are limited, frequent checkpointing can result in longer execution times. To extend checkpoint intervals, a proactive FT has been proposed. In Figure 1b, it predicts failures, and automatically migrates VMs from an "unhealthy" node to a "healthy" node. At the time, only the two nodes involved are re-allocated, and VMs on the other nodes keep running after the failure prediction. The assumption that failures can be predicted with 100 % accuracy is not realistic. Therefore, an approach of combining reactive and proactive FT is practical. In this paper, a fault detection and prediction mechanisms are out of scope. However, R. K. Sahoo, et al., reported that failures can be predicted with up to 70 % accuracy [8]. This means the interval between checkpoints can be extended up to 3.3 times.

VM migration and checkpoint/restart mechanisms cannot coexist with VMM-bypass I/O technologies. From the view point of FT, we address this issue.

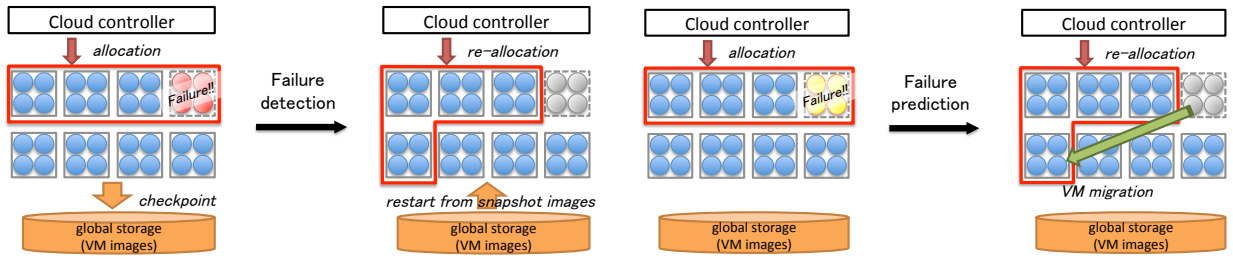
III. SymVirt: symbiotic virtualization mechanism

A. Design

We propose a *Symbiotic Virtualization (SymVirt)* mechanism that enables a VMM to cooperate with a message passing layer on the guest OS, where our target is an MPI application. The key to implementing migration and checkpoint/restart for VMs with VMM-bypass I/O devices is to unplug the devices only when such VM-level functions are required. To unplug devices safely while a parallel application is running on distributed VMMs, we must guarantee the ability to create a globally consistent snapshot of the entire virtualized cluster. Therefore, we employ a technique of combining a PCI hotplug and the coordination of a parallel application. The former enables us to add and remove devices while the OS is running. The latter is required to preserve the VM execution and communication states when the snapshots are restored in the future. Moreover, this approach enables us to avoid virtualization overhead during normal operations.

SymVirt provides a simple and intra-node communication mechanism between a VMM and the guest OS. That is, a pair of a VMM from/to the guest OS mode switch calls, *SymVirt wait* and *SymVirt signal*. From the view point of a guest OS, a SymVirt wait call is considered as a synchronous call. The execution of the VM is blocked until a SymVirt signal call is issued on the VMM. During the time between SymVirt wait and signal calls, VMM monitor commands, e.g., hot-add, hot-remove, and migration, can be issued.

Figure 2 shows an overview of the proposed mechanism, which consists of *SymVirt coordinator*, *SymVirt controller*, and *SymVirt agent*. SymVirt coordinator runs inside an application process. SymVirt controller is a master program on the VMM side. SymVirt controller and SymVirt agents work together to control distributed VMMs. This mechanism works in cooperation with a cloud controller, as shown in Section II.



a) Reactive fault tolerance

b) Proactive fault tolerance

Fig. 1. VM-level fault tolerance models on HPC clouds.

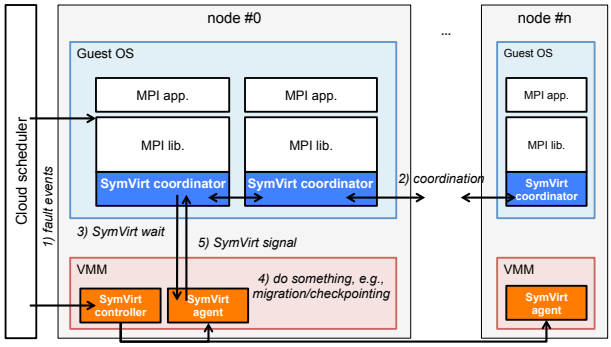


Fig. 2. An overview of the SymVirt mechanism. 1) A cloud controller delivers an FT event to both SymVirt coordinators and the SymVirt controller. 2) SymVirt coordinators create a consistent state for all VMs. 3) A SymVirt wait is issued. 4) The SymVirt controller invokes SymVirt agents and executes a procedure corresponding to the event. 5) A SymVirt signal is issued, and then the application is resumed.

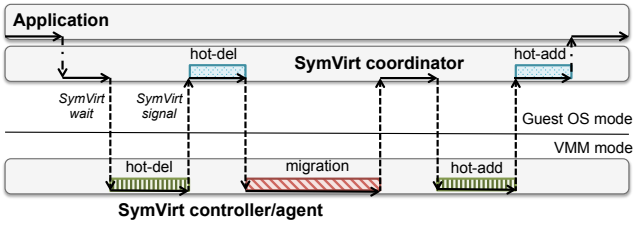


Fig. 3. The control flow of the SymPFT.

The work-flow of SymVirt is summarized as follows:

1) A cloud controller delivers an FT event to both an MPI runtime and the SymVirt controller. The MPI runtime invokes SymVirt coordinators at each MPI process.
2) SymVirt coordinators synchronize all processes and create a consistent state for the entire application by using a coordination protocol.
3) Each SymVirt coordinator issues a SymVirt wait call. The VM is paused until a SymVirt signal call is received.
4) The SymVirt controller spawns SymVirt agent threads. Each agent connects with the VMM monitor interface, and executes a procedure corresponding to the event.
5) SymVirt agents issue a SymVirt signal call, and the VMs are resumed.

We have considered two FT models on HPC clouds, as described in Section II. The first one is *SymVirt Checkpoint/Restart (SymCR)*, which enables us to checkpoint/restart VMs. The second one is *SymVirt Proactive Fault Tolerance (SymPFT)*, which enables us to migrate a VM from an "unhealthy" node before the node crashes. Both SymCR and SymPFT consist of the following three phases: 1) *hot-del*: a SymVirt agent removes a VMM-bypass I/O device from the VM, 2) *checkpointing* or *migration*, and 3) *hot-add*: a SymVirt agent attaches a VMM-bypass I/O device to the VM.

Figure 3 shows the control flow of SymPFT. Each phase involves a VMM from/to the guest OS mode transition. A SymVirt wait call is issued by a SymVirt coordinator. SymVirt agents do something to control a VM, followed by a SymVirt signal to wake up the VM. During phases 1) and 3), the guest OS needs to recognize the addition and removal of a device to migrate a VM safely. Therefore, a period of time that a PCI hotplug mechanism, i.e., the ACPI hotplug PCI controller driver `acpiphp`, can work on a guest OS is required.

B. Implementation

We implemented the proposed mechanism on top of the QEMU/KVM [10] and Open MPI [11], and we have confirmed it works on a virtualized cluster with VMM-bypass I/O devices including Infiniband, Open-MX (Myrinet over Ethernet), and 10 Gigabit Ethernet. Open-MX [12] is a software implementation of the Myrinet Express (MX) protocol that allows MPI processes to communicate with MX over Ethernet. The details of the implementation are described below.

1) SymVirt signal and wait: A SymVirt wait is implemented by using a *VMCALL* Intel VT-x instruction. VMCALL allows a guest OS to call the VMM for services. The execution of VMCALL causes a transition from a guest OS mode to a VMM mode, called a VM Exit. We extended QEMU/KVM slightly for handling SymVirt calls. QEMU/KVM handles VMCALL exceptions, and suspends the VM issued a SymVirt wait. On the other hand, a SymVirt signal is implemented by a QEMU/KVM monitor command. The SymVirt signal command resumes the VM.

2) SymVirt coordinator: A SymVirt coordinator is implemented by using the checkpoint/restart support of an MPI runtime so as to reuse an available and reliable implementation. Most of MPI implementations provide process-level

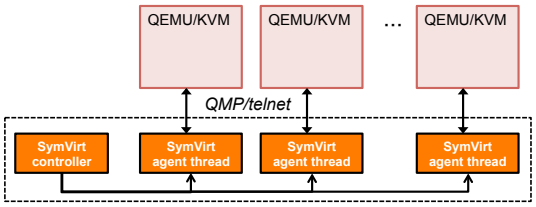


Fig. 4. A SymVirt controller and SymVirt agents.

checkpoint/restart. SymVirt coordinator exploits this for VM-level checkpoint/restart and migration.

SymVirt coordinator is implemented on top of the user-level checkpoint feature (SELF) of Open MPI. Open MPI provides a modular checkpoint/restart framework [13], which consists of a checkpoint/restart coordination protocol framework called the OMPI CRCP (Checkpoint/Restart Coordination Protocol), and a single process checkpoint/restart service framework called the OPAL CRS (Checkpoint/Restart Service). The OPAL CRS supports the SELF and BLCR [9] checkpoint/restart systems.

In order to ensure easy deployment, a SymVirt coordinator is required to work without modification of either an MPI library or applications. OMPI CRCP can be used without modification. Instead of implementing a new OPAL CRS component for SymVirt, we used a SELF component. A SELF component supports application level checkpointing by providing the application callbacks upon checkpoint, restart and continue operations. A SymVirt coordinator uses checkpoint and continue callbacks to issue SymVirt wait calls. On the other hand, SymVirt does not use a restart callback. SELF handler routines for SymVirt are implemented as a shared library, `libsymvirt.so`. Using the `LD_PRELOAD` environment variable, the library is loaded into an MPI process at runtime.

The `ompi-checkpoint` command is launched, and a checkpoint message is delivered to all MPI processes via the checkpoint callback function. At the time, a SymVirt coordinator is invoked. Note that the invocation of a SymVirt coordinator is delayed until an application executes any MPI communication functions, e.g., `MPI_Send` and `MPI_Recv`, because the OMPI CRCP starts only when the application is running inside an MPI function. The restart function is realized by loading a VM image from the checkpoint images, instead of launching the `ompi-restart` command.

3) SymVirt controller and agent: A SymVirt agent controls virtual machines by using QEMU monitor commands, including `savevm`, `migrate`, `device_add`, and `device_del`. The SymVirt controller and the SymVirt agent are implemented in Python. Figure 4 shows an overview of a SymVirt controller and SymVirt agents. The SymVirt controller invokes SymVirt agent threads for each QEMU. Each agent communicates with a QEMU process via the QEMU Monitor Protocol (QMP) or a telnet connection.

Figure 5 shows a SymPFT script. This script consists of

```
1  import symvirt
2  agent_list = [migrate_from]
3  ctl = symvirt.Controller(agent_list)
4
5  # device detach
6  ctl.wait_all()
7  kwargs = {'tag':'vf0'}
8  ctl.device_detach(**kwargs)
9  ctl.signal()
10
11 # vm migration
12 ctl.wait_all()
13 kwargs = {'postcopy':True, 'uri':'tcp:%s:%d'
14           % (migrate_to[0], migrate_port)}
15 ctl.migrate(**kwargs)
16 ctl.remove_agent(migrate_from)
17
18 # device attach
19 ctl.append_agent(migrate_to)
20 ctl.wait_all()
21 kwargs = {'pci_id':'04:00.0', 'tag':'vf0'}
22 ctl.device_attach(**kwargs)
23 ctl.signal()
24
25 ctl.close()
```

Fig. 5. SymPFT script.

three phases: hot-del (lines 5–9), migration (lines 11–16), and hot-add (lines 18–23). The `wait_all` waits until all given VMs issue the SymVirt wait call. In the case of SymPFT, only the source VM is suspended. The `signal` resumes all VMs. The other methods, including `device_detach`, `migrate`, `device_attach`, correspond to QEMU monitor commands. We assume that the cloud controller provides information, including the source and destination nodes of migration, the PCI ID of a VMM-bypass I/O device, and the names of snapshots. This is a reasonable assumption on HPC cloud environments.

SymCR consists of two scripts: SymCR-checkpoint and SymCR-restart. A SymCR-checkpoint script has the same structure as SymPFT, and an difference is only in the second phase. It executes a `savevm` command instead of a `migrate` command. Upon restarting a VM, the QEMU command is launched with the `-loadvm` option, and a SymCR-restart script executes only `device_add`.

4) Integration with postcopy migration: SymPFT relies on VM migration technologies. There are two types of VM live migration, precopy and postcopy. In precopy live migration, all states of a VM are completely copied to a destination node before the execution node is switched to the destination. Updated memory pages during memory copy are iteratively copied to the destination. It takes a long time to switch the execution node of an actively-running VM, and it is hard to estimate when migration will be completed. On the other hand, postcopy live migration executes memory page copies after the execution node is switched; it is possible to change the execution node in several hundred milliseconds, and the whole live migration process is completed in a determinable period.

We have developed a postcopy live migration for QEMU/KVM, called *Yabusame* [14]. A *umem* character device driver in the node Linux kernel works in cooperation with

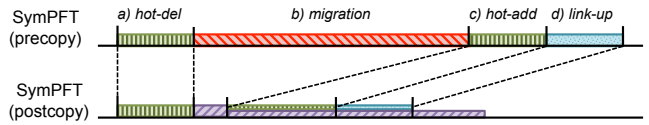


Fig. 6. The time line of SymPFT. Processes of SymPFT involving devices include a) hot-removeing a device, b) migration, c) hot-adding the device, and d) waiting for link-up of the device. In precopy migration, the downtime of an application is given by $a+b+c+d$. In postcopy migration, an application is resumed without waiting for the completion of migration.

Yabusame to hook guest OS's page faults and transfer memory pages from the source to the destination. The memory transfer mechanism supports on-demand and background modes.

Minimizing the down time helps to improve the total performance. Postcopy migration is effective for the purpose. Figure 6 shows the time line of SymPFT, contrasting precopy migration and postcopy migration. Here we separate "link-up" from "hot-add." The "link-up" indicates to wait until a link is active on a guest OS. In some cases, it consumes non-negligible time.

A precopy migration requires running every phase sequentially. This means that a precopy live migration is essentially the same as a stop and go, i.e., cold migration, because the application is resumed after phase d is finished. In contrast, a postcopy migration can overlap with phases c and d. Therefore, the down time can be reduced.

IV. Experiment

A. Experimental setting

We used a 16 node-cluster, which is a part of the AIST Green Cloud (AGC) cluster. The cluster consists of Dell PowerEdge M610 blade servers, and is comprised of 2 quad-core Intel Xeon 5540/2.53GHz CPUs, 48 GB of memory, a 300 GB SAS disk array, and a Mellanox ConnectX QDR Infiniband HCA. The Dell M1000e blade enclosure holds 16 blade servers and a 16 port Infiniband QDR blade switch. This cluster also has a 10 Gigabit Ethernet (GbE) network for using the network file system and for remote access with SSH. The MTU size is set to 9000 bytes. Hyper Threading was disabled.

We set up a virtualized cluster top on the AGC. A single virtual machine, which had 8 CPU cores and 20 GB of memory, was run on a physical machine. Although the proposed mechanism supports multiple virtual machines on a physical machine, this configuration is the key to achieving high performance on a VM environment. The host OS and the guest OS are the Debian GNU/Linux 7.0 (testing) and the Scientific Linux 6.2, respectively. The VM image is created using the qcow2 format which enables us to make snapshots internally. Live migration is required for the shared storage among the source and destination nodes. In this experiment, we used NFS.

We use KVM to provide the underlying hypervisor for our HPC cloud. The proposed mechanism is implemented based on the Linux kernel version 3.2.18 and QEMU/KVM version 1.1-rc3, integrated with Yabusame version June-04-2012 [15].

TABLE I
The overhead of SymCR and SymPFT on a 16 GB-memory access memtest benchmark over Infiniband [seconds].

	chckpointing /migration	hotplug	link-up	total
SymCR	258.7	4.2	31.5	294.4
SymPFT (precopy)	104.8	10.8	27.5	140.8
SymPFT (postcopy)	-	-	-	105.8

The virtual CPU model was set to `host` to allow the guest OS to use all available host processor features, including the SSE instruction set. To configure NUMA affinity, `-smp` and `-numa` options were also set at boot time.

On the VM environment, the OpenFabrics Enterprise Distribution (OFED) version 1.5.4.1 was used. The benchmark applications were compiled with gcc/gfortran version 4.4.6, and the optimization option was set to `-O2`. We used Open MPI version 1.6.0 [11] as an MPI implementation, and the option was set to `--mca mpi_leave_pinned 0 -am ft-enable-cr`.

B. Memtest Micro Benchmark

In order to demonstrate the proposed mechanism, we evaluate a simple memory intensive micro benchmark, called *memtest*. We compared the overhead of SymCR and SymPFT. With SymPFT, in addition, we examined the effect of postcopy migration. This benchmark sequentially writes to a 16 GB-memory area, where each iteration spends about 3 seconds. We used 8 VMs, and an MPI process ran on each VM. After 5 iterations, SymCR-checkpoint or SymPFT script was launched.

Table I shows the execution overhead with both SymCR and SymPFT. Each value is measured three times and the best is taken. Compared with the overhead of SymCR, that of SymPFT with precopy is reduced to one half; that of SymPFT with postcopy is reduced to one third. With SymPFT, postcopy has significant advantages over precopy, because it enables us to hide the overhead of hotplug and link-up by overlapping them with migration. Analyzing the breakdown of the overhead, the migration phase of SymPFT with precopy spends approximately the same amount of time as the total time of SymPFT with postcopy.

The throughput of SymCR and SymPFT are 61.8 MB/s and 152.7 MB/s, respectively. This is caused by a difference of write access throughput between the local disk and the 10 GbE network. Therefore, the larger the memory footprint, the larger the gap in performance. Note that both precopy and postcopy migration can not fully utilize the 10 GbE bandwidth due to the limitations of the QEMU implementation. We will discuss the reason in Section V. With SymPFT, the effectiveness relative to SymCR increases, as the migration throughput improves.

Table II shows the results of the memtest benchmark on the Open-MX network. Unfortunately, at the moment, Broadcom NetXtreme II does not support PCI passthrough, so we have equipped the other two machines with Intel

TABLE II
THE OVERHEAD OF SYMPFT ON A 1.6 GB-MEMORY ACCESS MEMTEST BENCHMARK [SECONDS] OVER OPEN-MX.

	migration	hotplug	link-up	total
Open-MX over GbE				
SymPFT (precopy)	17.6	5.1	3.7e-05	20.4
SymPFT (postcopy)	-	-	-	18.8
Open-MX over 10 GbE				
SymPFT (precopy)	6.9	5.5	4.9e-05	9.5
SymPFT (postcopy)	-	-	-	9.2

Xeon X5650/2.66GHz, 6 GB memory, an Intel X520 SR-IOV supported 10 GbE NIC, and a Broadcom NetXtreme II on-board GbE NIC. We measured on two configurations: the on-board GbE NIC used only for migration and the 10 GbE shared for both migration and MPI communication. In the latter, we used the SR-IOV feature. The physical function is used for migration; the virtual function is assigned to a VM. The amount of memory access is set to 1.6 GB. The result shows the link-up time is negligible for Ethernet as contrasted with Infiniband. This issue will be discussed in Section V.

C. NAS Parallel Benchmarks

We evaluate the proposed mechanism with more a practical application benchmark, the NAS Parallel Benchmarks (NPB) version 3.3.1. The problem size is class D. We used the following five benchmark programs from NPB: BT (Block Tridiagonal), CG (Conjugate Gradient), EP (Embarrassingly Parallel), FT (Fast Fourier Transform), and LU (LU Simulated CFD Application). Eight processes were executed on every node. We used eight nodes, and the total number of processes was 64. A SymPFT script is launched once at three minutes after each benchmark start time. With EP, it is launched at one minute after the benchmark start time, because the execution time is too short to do otherwise.

Table III shows the results of the execution time, in order of increasing overhead. Figure 7 breaks down the overhead caused by SymPFT. The "baseline" indicates the execution without SymVirt. For information, we also measured the performance of "bare metal," i.e., a physical cluster. The overhead of SymCR ranged from 96.9 to 588.9 seconds; that of SymPFT with precopy ranged from 44.4 to 299.1 seconds; that of SymPFT with postcopy ranged from 26.8 to 149.4 seconds.

The overhead is proportional to the memory footprint of a program. Comparing this with a memtest micro benchmark, the effect of postcopy migration is relatively limited because these memory footprints, except for FT, are small (from 723 MB to 4.4 GB) as shown in Table IV. With FT, the effect of postcopy is obvious. The memory footprint is large and the transferred memory size is 16 GB. Postcopy migration enables us to hide the overhead of hotplug and link-up by overlapping them and migration.

The results of EP are not intuitive. EP is a embarrassingly parallel program in that no communication is required, and

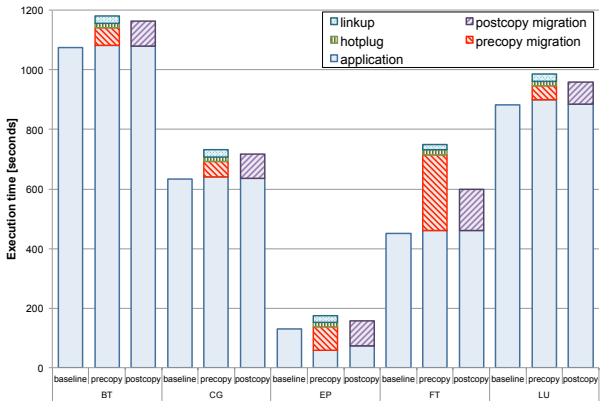


Fig. 7. The overhead of SymPFT on NPB 3.3 class D.

`MPI_Allreduce` is called at the last moment. The memory footprint is smaller than that of the others, however, the overheads of both SymCR and SymPFT are relatively large. The cause can be considered to be as follows. A SymVirt coordinator starts to execute when an MPI function is called. At that time, the OPAL CRS invokes two helper threads per process. These threads can degrade the application performance, especially for compute intensive programs. And also, the baseline is worse than that of the application phase of SymPFT. In the case where the `mpirun` command is executed with the FT option, a few processes take about 40 seconds for `MPI_Allreduce`. This problem is reproducible. We need to investigate what is happening.

V. DISCUSSION

This section disscusses the experimental results and open issues. We also present some ideas for optimization to improve the efficiency.

The pair of SymVirt wait and signal calls can be considered as a synchronous call. This approach has some benefits, including ease of implementation and a simple API. However, the drawback is that the entire VM is suspended and resumed during a call. Introducing an asynchronous call can help to reduce the down time because a PCI hotplug and other programs work in the background.

The link-up time of Infiniband devices costs about 30 seconds. It is not a negligible overhead. During that time, the hardware state keeps "polling," which indicates the port is not physically connected. We need to investigate what is happening. In contrast, Open-MX, which relies on Ethernet, does not have the same problem, as shown in Table II. In the case of Ethernet, moreover, the down time can be reduced by redirecting accesses from VMM-bypass devices to a virtual network with a bonding driver, as mentioned in [16].

In our experiment, the network throughput of migration is less than 1.3 Gbps, and it can not fully utilize a 10 GbE network. This is because of CPU bottleneck at the source node. During the migration, the utilization of one CPU core is saturated at 100 %. The current QEMU migration implementation,

TABLE III
EXECUTION TIME OF NPB 3.3 CLASS D [SECONDS (DIFFERENCE FROM THE BASELINE IN SECONDS)].

	BT	CG	EP	FT	LU
Baseline	1075.5 (-)	634.1 (-)	131.2 (-)	451.2 (-)	882.2 (-)
SymCR	1172.4 (96.9)	761.2 (127.1)	252.1 (120.8)	1040.1 (588.9)	989.7 (107.5)
SymPFT (precopy)	1180.9 (105.4)	731.8 (97.3)	175.7 (44.4)	750.3 (299.1)	986.1 (103.9)
SymPFT (postcopy)	1167.8 (92.3)	717.6 (82.5)	158.0 (26.8)	600.7 (149.4)	962.3 (80.1)
Bare metal	915.7 (-159.8)	616.6 (-17.5)	113.2 (-18.0)	403.5 (-47.7)	885.2 (3.0)

TABLE IV
TRANSFERRED MEMORY SIZE DURING VM MIGRATION [MB].

BT	CG	EP	FT	LU
4417	3394	723	15978	2348

based on TCP/IP, has a high processing overhead. RDMA-based migration [17] can reduce CPU utilization and improve the throughput, compared with TCP/IP-based migration.

Postcopy live migration is effective to reduce the down time of SymPFT. However, it may slow down the performance of a target VM for a certain time after a migration due to remote memory page transfer. In the experiment, however, such a performance degradation was not observed. In the case of memtest, the iteration just after resuming from SymCR costs about 8 seconds. There is no obvious difference between precopy and postcopy migration. This can be thought of as follows. Most memory pages have already been transferred by the time migration is finished. Therefore, there is not so much overlap between the application phase and the migration phase. In the case of NPB FT, on the other hand, there is an overlap period between them. However, same performance degradation was not observed. We need to investigate the side effects of postcopy migration using more varied applications.

The strategy of memory page transfer leaves much room for improvement. A hybrid migration, which is an approach combining with precopy and postcopy migration, can be promising to address the issue. Before a hot-remove, most memory pages are transferred to the destination in advance in the background of an executing application. Updated/dirty pages are transferred just before the hot-add. This results in reducing the period that VMM-bypass I/O devices are disconnected. There are other drawbacks in the current postcopy implementation. For example, only one half of the physical memory can be assigned to a guest OS. In future implementations, this will be fixed.

VI. RELATED WORK

High availability systems, Remus [18] and Kemari [19], have modified the VM live migration mechanism to enable highly frequent synchronization between primary and backup nodes. These systems focus on indivisual VMs whereas SymVirt focuses on a virtualized cluster. The large overhead of synchronization at high frequency rate is unacceptable for HPC applications. Some VM-level reactive and proactive FT systems have been proposed for HPC systems. VM-level proactive FT, i.e., checkpoint/restart, exploits a mature VM migration mechanism [20], [21]. VNsnap [20] focuses on distributed snapshots of a virtualized cluster, like the proposed mechanism. Unlike the proposed system, the coordination is executed by a software switch outside the VMs, and the mechanism assumes that VMs communicate via virtualized Ethernet. Therefore, it cannot coexist with VMM-bypass I/O devices. A. B. Nagarajan and F. Mueller have proposed a proactive FT system based on Xen [22]. This system supports only para-virtualized Ethernet drivers. Some para-virtualized Infiniband drivers for Xen and VMWare ESXi have been proposed [23], [24], [25]. Nomad, in particular, supports migration of virtual machines with an Infiniband device [24]. In contrast to these studies, the proposed mechanism relies on VMM-bypass I/O technologies and hotplug mechanisms instead of implementing a para-virtualized driver for a specific VMM. Therefore, there is no performance overhead and no limitation in supported devices, e.g., Myrinet and other devices.

There are many studies on the performance of VM migration under Web and enterprise workloads, however, few studies focus on HPC workloads. K. Z. Ibrahim, et al., reports on optimized precopy live migration for memory intensive HPC applications [26]. We evaluated our FT system, which supports both precopy and postcopy live migration, under HPC workloads.

Although VMM-bypass I/O technologies are effective in improving the I/O performance of a guest OS, it is still unable to achieve the levels of bare metal due to the overhead of VM Exits, which increases the communication latency. This is because a guest OS cannot selectively intercept physical interrupts. Exit-less interrupt (ELI) [27] addresses this issue. It is a software-only approach for handling interrupts within guest VMs directly and securely. We expect that next-generation hardware virtualization will support a selective interrupt for a VM Exits feature. As another approach to achieve the combination of performance and dependability, H. B. Chen, et al., have proposed a self-virtualization technique [28], which provides an OS with the capability to turn virtualization on and off on demand. It enables migration and checkpoint/restart to avoid virtualization overhead during normal operations. However, it lacks a coordination mechanism among distributed VMMs.

Para-virtualization is an optimization technique using an

implicit commnication between a guest OS and the VMM via a traditional OS interfaces such as a device driver interface. Some explicit communication mechanisms have been proposed. SymCall [29] provides an upcall mechanism from a VMM to a guest OS, using a nested VM Exit call. In contrast, Socket outsourcing [30] and SymVirt provide a simple hypercall mechanism from a guest OS to the VMM. SymVirt does not require such a complicated upcall mechanism, assuming it works in cooperation with a cloud controller. Socket outsourcing offloads a guest OS's functionality, like TCP/IP communication, to the VMM.

VII. Conclusion

We have proposed a symbiotic virtualization mechanism, called SymVirt, for enabling VM migration and checkpoint/restart on a virtualized cluster with VMM-bypass I/O devices. In order to be able to disconnect the devices only when such functions are needed, SymVirt provides a mechanism offering a combination of a PCI hotplug and coordination of a parallel application on distributed VMMs. The proposed mechanism is implemented on top of both the QEMU/KVM and the Open MPI system. Using the proposed mechanism, we have demonstrated the feasibility of a proactive FT system. We have confirmed the effectiveness using both a memory intensive micro benchmark and the NAS parallel benchmark, on a virtualized Infiniband cluster. The experimental results show that SymPFT with precopy is two times faster than SymCR; moreover SymPFT with postcopy is three times faster. With SymPFT, postcopy migration has significant advantages over precopy migration, because it enables us to hide the overhead of hotplug and link-up by overlapping migration.

We plan to design and implement a generic communication layer supporting the SymVirt mechanism, which is not dependent on an MPI system. Other future tasks include integrating the SymVirt mechanism and failure detection/prediction mechanisms to a cloud controller, and a demonstration of the effectiveness on a realistic failure condition.

In this paper, although we focused on fault tolerance, the proposed mechanism can also be applied to server consolidations and load balancing on a virtualized cluster. In those cases, we need to develop optimization techniques, e.g., an asynchronous SymVirt call without the need for pausing the entire VM, and a more efficient memory page transfer scheme for postcopy migration using the memory access characteristics of the applications themselves.

References

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

[2] Google Compute Engine. https://developers.google.com/compute/.

[3] R. Campbell, et al. Open Cirrus Cloud Computing Testbed: Federated Data Centers for Open Source Systems and Services Research. In *Proc. of the 1st USENIX Workshop on Hot Topics in Cloud Computing*, 2009.

[4] L. Nussbaum, et al. Linux-based virtualization for HPC clusters. In *Proc. of the Ottawa Linux Symposium*, pages 221–233, July 2009.

[5] N. Regola and J. C. Ducom. Recommendations for Virtualization Technologies in High Performance Computing. In *Proc. of the Second IEEE International Conference on Cloud Computing Technology and Science*, pages 409–416, December 2010.

[6] P. Luszczek, et al. Evaluation of the HPC Challenge Benchmarks in Virtualized Environments. In *Proc. of the 6th Workshop on Virtualization in High-Performance Cloud Computing*, Auguest 2011.

[7] R. Takano, et al. Toward a practical "HPC Cloud": Performance tuning of a virtualized InfiniBand cluster. In *Proc. of the 6th International Conference on Ubiquitous Information Technologies and Applications*, December 2011.

[8] R. K. Sahoo, et al. Critical Event Prediction for Proactive Management in Large-scale Computer Clusters. In *Proc. of the 9th ACM international conference on knowledge discovery and data mining*, pages 426–435, 2003.

[9] P. H. Hargrove and J. C. Duell. Berkeley Lab Checkpoint/Restart (BLCR) for Linux Clusters. In *Proc. of SciDAC 2006*, 2006.

[10] A. Kivity. KVM: the Linux Virtual Machine Monitor. In *Proc. of the Ottawa Linux Symposium*, pages 225–230, July 2007.

[11] Open MPI. http://www.open-mpi.org/.

[12] B. Goglin. High-Performance Message Passing over Generic Ethernet Hardware with Open-MX. *Journal of Parallel Computing*, 37:85–100, 2011.

[13] J. Hursey, et al. The design and implementation of checkpoint/restart process fault tolerance for Open MPI. In *Proc. of the 12th IEEE Workshop on Dependable Parallel, Distributed and Network-Centric Systems*, March 2007.

[14] T. Hirofuchi, et al. Reactive consolidation of virtual machines enabled by postcopy live migration. In *Proc. of the 5th international workshop on virtualization technologies in distributed computing*, pages 11–18, June 2011.

[15] Postcopy Live Migration for Qemu/KVM (Yabusame). http://grivon.apgrid.org/quick-kvm-migration.

[16] E. Zhai, et al. Live Migration with Pass-through Device for Linux VM. In *Proc. of the Ottawa Linux Symposium*, pages 261–267, July 2008.

[17] W. Huang, et al. High Performance Virtual Machine Migration with RDMA over Modern Interconnects. In *Proc. of the IEEE International Conference on Cluster Computing*, 2007.

[18] B. Cully, et al. Remus: High Availability via Asynchronous Virtual Machine Replication. In *Proc. of the 5th USENIX Symposium on Networked Systems Design and Implementation*, pages 161–174, 2008.

[19] Y. Tamura. Kemari: Virtual Machine Synchronization for Fault Tolerance using DomT. In *Xen Summit 2008*, 2008.

[20] A. Kangarlou, et al. VNsnap: Taking Snapshots of Virtual Networked Environments with Minimal Downtime. In *Proc. of the IEEE/IFIP Dependable Systems and Networks*, pages 524 – 533, June 2009.

[21] K. Chanchio, et al. An Efficient Virtual Machine Checkpointing Mechanism for Hypervisor-based HPC Systems. In *Proc. of the High Availability and Performance Computing Workshop*, 2008.

[22] A. B. Nagarajan and F. Mueller. Proactive Fault Tolerance for HPC with Xen Virtualization. In *Proc. of the 22nd International Supercomputing Conference*, 2007.

[23] J. Liu, et al. High Performance VMM-Bypass I/O in Virtual Machines. In *Proc. of the 2006 USENIX Annual Conference*, 2006.

[24] W. Huang, et al. Nomad: Migrating OS-bypass Networks in Virtual Machines. In *Proc. of the 2007 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, 2007.

[25] B. Davda and J. Simons. RDMA on vSphere: Update and Future Directions. In *Open Fabrics Workshop*, March 2012.

[26] K. Z. Ibrahim, et al. Optimized Pre-Copy Live Migration for Memory Intensive Applications. In *Proc. of the 24th International Conference for High Performance Computing, Networking, Storage and Analysis*, 2011.

[27] A. Gordon, et al. Schuster, and D. Tsafrir. ELI: Bare-metal Performance for I/O Virtualization. In *Proc. of the 17th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, March 2012.

[28] H. B. Chen, et al. Mercury: Combining Performance with Dependability Using Self-Virtualization. *Journal of Computer Science and Technology*, 27(1):92–104, 2012.

[29] J. Lange and P. Dinda. SymCall: Symbiotic Virtualization Through VMM-to-Guest Upcalls. In *Proc. of the 2011 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, pages 193–204, March 2011.

[30] H. Eiraku, et al. Fast Networking with Socket-Outsourcing in Hosted Virtual Machine Environments. In *Proc. of the 24th ACM Symposium on Applied Computing*, pages 310–317, 2009.