PAPER Special Section on Parallel and Distributed Computing and Networking

Cooperative VM Migration: A Symbiotic Virtualization Mechanism by Leveraging the Guest OS Knowledge

Ryousei TAKANO[†a)], **Hidemoto NAKADA**[†], **Takahiro HIROFUCHI**[†], **Yoshio TANAKA**[†], ***Nonmembers***, ***and*** **Tomohiro KUDOH**[†], ***Member***

SUMMARY A virtual machine (VM) migration is useful for improving flexibility and maintainability in cloud computing environments. However, VM monitor (VMM)-bypass I/O technologies, including PCI passthrough and SR-IOV, in which the overhead of I/O virtualization can be significantly reduced, make VM migration impossible. 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 user-level dynamic device configuration 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, Ethernet, 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.

key words: *virtualization, VM migration, HPC cloud, fault tolerance*

1. 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 CycleCloud [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]–[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, and 3) easy deployment.

To meet the above requirements, we adopt a gray-box approach [8], [9], which is a cross-layer technique to improve the performance and the functionality on a virtualized environment by leveraging the knowledge of a guest operating system (OS). In this paper, we propose a *Symbiotic Virtualization (SymVirt)* mechanism, for enabling VM-level migration and checkpoint/restart on a virtualized cluster with VMM-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 user-level dynamic device configuration and global coordination of distributed VMMs. Second, to meet requirement 3), the implementation supports all PCI passthrough enabled devices without implementing specific para-virtualized drivers; it also requires no modifications to applications and an MPI runtime inside a guest

Manuscript received December 26, 2012.
Manuscript revised April 20, 2013.

[†]The authors are with Information Technology Research Institute, National Institute of Advanced Industrial Science and Technology (AIST), Tsukuba-shi, 305–8568 Japan.
a) E-mail: takano-ryousei@aist.go.jp

DOI: 10.1587/transinf.E96.D.2675

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.

The rest of the paper is organized as follows. Section 2 describes the background of HPC clouds, especially focused on fault tolerance. The design and implementation of the proposed mechanism is presented in Sect. 3. Section 4 shows the experimental results, and we discuss further optimization techniques in Sect. 5. In Sect. 6, we mention related work. Finally, Sect. 7 summarizes the paper.

2. 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 [10]. For instance, 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.

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 on top of them. The user submits an application job on the virtualized cluster, and application processes are invoked on distributed VMs. 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.

There are two FT approaches: reactive FT and proactive FT. Checkpoint/restart is a popular reactive FT approach. 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 the network and storage 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 by several researchers. 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 [11]. 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, as mentioned in Sect. 1. From the view point of FT, we address this issue.

3. SymVirt: Symbiotic Virtualization

3.1 Approach

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. VMM-bypass I/O devices can be detached and re-attached using PCI device hotplugging. For instance, by detaching all VMM-bypass I/O devices currently attached, it is possible to migrate a VM. After migration, the devices are re-attached to the VM.

However, an existing VM migration has following two problems. First, VMM does not know the time when VMM-bypass I/O devices are detached safely. To perform such migration without losing in-flight data, packet transmission to/from the VM should be stopped prior to detaching. In the worst case, applications could be aborted by such communication errors. With a VMM, it is hard to know the communication status of an application inside a guest OS, especially if VMM-bypass I/O devices are used. We also must guarantee the ability to create a globally consistent snapshot of the entire virtualized cluster. Second, VMM cannot migrate the state of VMM-bypass I/O devices from the source to the destination. For instance, with Infiniband, VMMs have to save and restore states of location dependent resources, including Local IDs and Queue Pair Numbers before and after migration.

We tackle above problems by cooperating with a VMM and an application inside the guest OS. We call this approach *a cooperative VM migration*. An existing VM migration is a black-box approach. The portability is good, but the performance overhead is high. In contrast, cooperative VM migration is a gray-box approach [8], [9], which is a cross-layer technique to improve the performance and the functionality on a virtualized environment by leveraging the knowledge of a guest OS. Concretely, we transfer two features, i.e., VMM-bypass I/O device management and global coordination of a parallel application from a VMM to inside the guest OS, as shown in Fig. 1. The former provides a thin user-level abstraction communication layer of VMM-bypass I/O devices, and it enables us to dynamically configure devices in cooperation with PCI device hotplugging. 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.

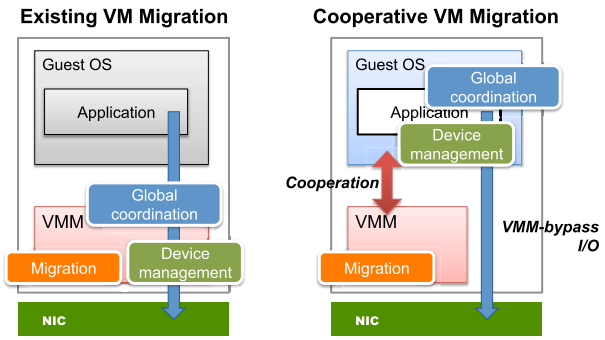

Fig. 1 A cooperative VM migration.

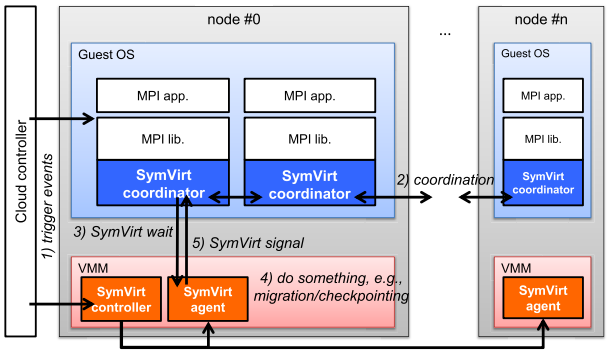

Fig. 2 An overview of the SymVirt mechanism.

3.2 Design

To realize a cooperative VM migration, we propose a *Symbiotic Virtualization (SymVirt)* mechanism that enables a VMM to cooperate with a message passing layer on the guest OS. Our target application is an MPI program. 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, and it provides global coordination and VMM-bypass I/O device management. SymVirt controller is a master program on the VMM side. SymVirt controller and SymVirt agents work together to control distributed VMMs.

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-del, and migration, can be issued.

The work-flow of SymVirt is summarized as follows:

1. A cloud controller requests both an MPI runtime and the SymVirt controller to checkpoint/restart or migrate a VM. The MPI runtime invokes SymVirt coordinators at each MPI process.

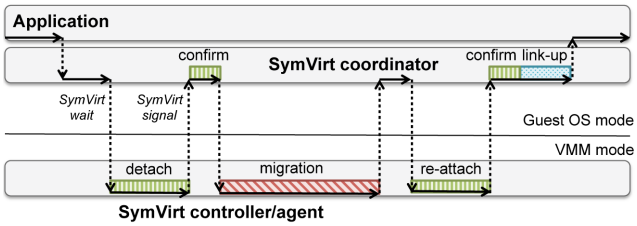

Fig. 3 The control flow of the SymPFT.

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 Sect. 2. 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 re-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), SymVirt coordinator needs to recognize the addition and removal of a device to migrate a VM safely, in cooperation with a PCI device hotplug mechanism on the guest OS. The period of time is denoted as "confirm" in Fig. 3.

3.3 Implementation

We implemented the proposed mechanism on top of the QEMU/KVM [12] and Open MPI [13]. We can use all PCI devices, including Infiniband, Ethernet, and Myrinet, if the device supports PCI passthrough. 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 [14] 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.

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. A SymVirt signal is implemented by a QEMU/KVM monitor command. The SymVirt signal command resumes the VM.

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

Open MPI provides a modular checkpoint/restart framework [15], 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 user-level checkpoint feature (SELF) and BLCR [10] checkpoint/restart systems.

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; SymVirt does not use a restart callback. SELF handler routines for SymVirt are implemented as a shared library. Using the `LD_PRELOAD` environment variable, the library is loaded into an MPI process at runtime.

OMPI CRS releases all resources allocated on Infiniband devices in the pre-checkpoint phase. OMPI Byte Transfer Layer (BTL) provides an interconnect agnostic abstraction, used for MPI point-to-point messages on several types of networks. BTL modules are reconstructed and connections are re-established in the continue and restart phases. Therefore, there are no problems even if Local IDs (port addresses) or Queue Pair Numbers are changed after a migration. This design can be considered because BLCR does not support mechanisms to save and restore network connections, i.e., sockets.

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.

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. 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.

```
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 = {'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. 4 SymPFT script.

Figure 4 shows a SymPFT script. This script consists of 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 SymCR-checkpoint and SymCR-restart scripts. 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. Experiment

4.1 Experimental Setting

We used a 16 node-cluster, which is a part of the AIST Green Cloud cluster. The cluster consists of Dell PowerEdge M610 blade servers, and is comprised of 2 quad-core Intel Xeon E5540/2.53GHz CPUs, 48 GB of memory, a 300 GB SAS disk, 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 on top of the physical cluster. A single VM, which had 8 CPU cores and 20 GB of memory, was run on a physical machine. 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 version 3.

The proposed mechanism was implemented based on the Linux kernel version 3.2.18 and QEMU/KVM version 1.1-rc3. 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 [13] as an MPI implementation, and the option was set to "–mca mpi_leave_pinned 0 -am ft-enable-cr".

4.2 The Overhead of SymVirt

To demonstrate the proposed mechanism, we evaluate the overhead of SymVirt using two benchmark programs: memtest and NAS Parallel Benchmarks. We used 8 VMs, and an MPI process ran on each VM. The overhead of SymVirt is divided into three parts: hotplug, link-up, and checkpointing/migration, as shown in Fig. 3. The hotplug time is the sum of the detach, re-attach, and confirm times. The link-up time is a wait time until a link is active on a guest OS. Theoretically, both the hotplug and link-up times are constant; checkpointing/migration time is dependent on the memory footprint.

4.3 Memtest Micro Benchmark

Memtest is a simple memory intensive micro benchmark, which sequentially wrote data to a memory array that ranged from 2 GB to 16 GB. Figure 5 shows the total execution overhead of SymPFT. We have confirmed that the variation of the overhead is within 2 seconds in other experiments. Most of the variation is caused by a migration part. Analyzing the breakdown of the overhead, the migration time is dependent on the memory footprint; both hotplug and link-up times are approximately constant. The migration time is not exactly proportional to the memory footprint. This is because QEMU/KVM traverses the whole of the guest OS's memory during a migration. The migration mechanism compresses pages that contain uniform data, e.g., "zero pages," to reduce the amount of transferred memory pages. The migration time can be also affected by network condition. Network contention can increase the migration time. In this experiment, however, the effect is negligible because there is no network contention.

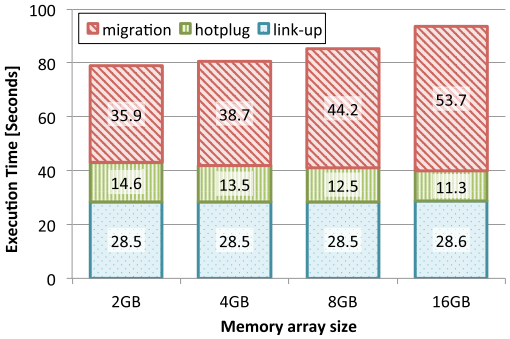

Fig. 5 The overhead of SymPFT on a memtest benchmark [seconds].

Table 1 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	53.7	11.3	28.6	93.6

Table 2 The overhead of SymPFT on a 1.6 GB-memory access memtest benchmark over Open-MX [seconds].

	migration	hotplug	link-up	total
SymPFT (GbE)	17.6	5.1	3.7e-05	20.4
SymPFT (10GbE)	6.9	5.5	4.9e-05	9.5

Table 1 shows the overhead of both SymCR and SymPFT. The throughput of SymCR and SymPFT are 61.8 MB/s and 298.0 MB/s, respectively. Note that the throughput is given by dividing the transferred data size, i.e., 16 GB, by the checkpointng or migration time. 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.

The link-up time costs about 30 seconds. This is not a negligible overhead. We have confirmed how long another device takes the link-up time. Table 2 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 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. A single VM, which has 6 CPU cores and 2 GB of memory, was run on a physical machine. 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 Sect. 5.

Table 3 Execution time of NPB 3.3 class D [seconds (difference from the baseline in seconds)].

	BT	CG	FT	LU
Baseline	1075.5 (-)	634.1 (-)	451.2 (-)	882.2 (-)
SymCR	1172.4 (96.9)	761.2 (127.1)	1040.1 (588.9)	989.7 (107.5)
SymPFT	1166.6 (91.1)	722.4 (88.3)	615.5 (164.3)	977.3 (95.1)
Bare metal	915.7 (−159.8)	616.6 (−17.5)	403.5 (−47.7)	885.2 (3.0)

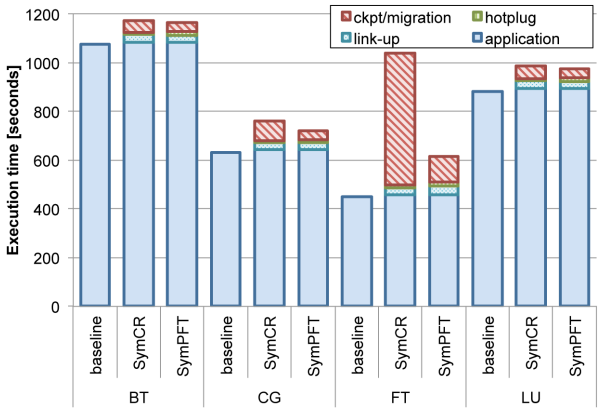

Fig. 6 The overhead of SymCR and SymPFT on NPB 3.3 class D.

4.4 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 four benchmark programs from NPB: BT (Block Tridiagonal), CG (Conjugate Gradient), 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 or SymCR script is launched once at three minutes after each benchmark start time.

Table 3 shows the results of the execution time, in order of increasing overhead. Figure 6 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. First, both SymCR and SymPFT have no performance overhead during normal operations. In Fig. 6, the application portions increase by about 1% compared with baseline. To be exact, this performance degradation should be included the overhead of SymCR or SymPFT instead of that of normal operations. Note that in both cases of SymCR and SymPFT, the execution paths are exactly same as baseline during normal operations. There can be some reason that have a negative impact on the application performance, including cache pollution after restarting and the overhead of checkpoint helper threads of Open MPI. Second, the checkpoint and migration times depend on the memory footprint, where the memory footprints of BT, CG, FT, and LU are 4.4 GB, 3.4 GB, 16 GB, and 2.3 GB, respectively; both hotplug and link-up times are constant.

For the FT benchmark, SymCR suffers from large overhead compared with SymPFT. The write I/O throughput with a VM image, in which QEMU/KVM accesses via NFS, might obviously decrease due to some reasons. We need to investigate this behavior in detail. The performance analysis of VM-level checkpointing is an open issue.

5. Discussion

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

During checkpointing and migration using SymVirt, an application is completely frozen. Therefore, the execution time of SymPFT is exactly equal to the service down time. Although the impact depends on applications and the frequency of migrations, reducing overhead costs is another important open issue.

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. We do not consider it is a fundamental flaw of the design, and we believe the overhead can be reduced significantly. In contrast, Open-MX, which relies on Ethernet, does not have the same problem, as shown in Table 2. 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 2.5 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, 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.

SymPFT relies on VM migration technologies. There are two types of VM live migration, precopy and postcopy. In our experiment, we used an existing precopy live migration in QEMU/KVM. We have developed a postcopy live migration for QEMU/KVM, called *Yabusame* [18]. Postcopy live migration can be effective to reduce the down time of SymPFT. Processes of SymPFT include *a*) hot-del, *b*) migration, *c*) hot-add, and *d*) waiting for link-up of the device. In precopy migration, the down time of an application is given by $a + b + c + d$. 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 enable us

to execute phases *c* and *d* in the background of migration, that is to say, we can overlap phase *b* with phases *c* and *d*. Therefore, the down time can be reduced. We have preliminary evaluated an integration of postcopy migration into SymPFT [19].

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-del, 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.

6. Related Work

High availability systems, Remus [20] and Kemari [21], 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 [22], [23]. VNsnap [22] 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 [24]. This system supports only para-virtualized Ethernet drivers.

Some para-virtualized Infiniband drivers for Xen and VMWare ESXi have been proposed [25]–[27]. In contrast to these studies, the proposed mechanism relies on VMM-bypass I/O technologies and hotplugging 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. Nomad, in particular, supports migration of virtual machines with an Infiniband device [26]. Nomad virtualizes location dependent resources, including Local IDs (port addresses), Queue Pair Numbers, and memory keys for RDMA operations. The proposed system does not need such virtualization because it relies on Open MPI's checkpoint/restart framework to re-establish all connections after a migration. This contributes the simplicity and easy deployment of our implementation. Another advantage of SymVirt is a great potential for extensions by leveraging cooperation between a VMM and a communication middleware inside the guest OS. For instance, our ongoing work achieves VM migration between an Infiniband cluster and an Ethernet cluster, as shown in paper [28]. To the best of our knowledge, it is the first successful attempt of an interconnect transparent migration. However, the disadvantage of SymPFT is large overhead of migration. SymPFT currently suffers from the large service down time at a migration, as mentioned in Sect. 5.

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) [29] 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, e.g., APICv, will significantly reduce the number of VM Exits at a virtual interrupt injection. As another approach to achieve the combination of performance and dependability, H.B. Chen, et al., have proposed a self-virtualization technique [30], 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 [9] provides an upcall mechanism from a VMM to a guest OS, using a nested VM Exit call. In contrast, Socket outsourcing [31] 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.

7. 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. To be able to disconnect the devices only when such functions are needed, SymVirt provides a mechanism offering a combination of a user-level dynamic device configuration 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. For easy deployment, it supports all PCI devices without implementing any para-virtualized drivers; it also requires no modification to applications and an MPI runtime inside a guest OS. 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. As a result, we have confirmed that 1) the proposed mechanism has no performance overhead during normal operations, 2) the overhead depends on both the memory footprint and the media access speed, and

3) the link-up time after re-attaching an Infiniband device is not negligible. SymPFT is three times faster than SymCR. This is caused by a difference of write access throughput between the local disk and the 10 GbE network.

We plan to design and implement a generic communication layer supporting the SymVirt mechanism, which does not rely on an MPI system. Other future tasks include analyzing the performance impact of SymVirt on real application, 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.

Acknowledgements

This work was partly supported by JSPS KAKENHI Grant Number 24700040 and ARGO GRAPHICS, Inc.

References

[1] Amazon EC2, http://aws.amazon.com/ec2/
[2] Google Compute Engine, https://developers.google.com/compute/
[3] CycleCloud, http://cyclecomputing.com/cyclecloud/overview
[4] L. Nussbaum, F. Anhalt, O. Mornard, and J.P. Gelas, "Linux-based virtualization for HPC clusters," Proc. Ottawa Linux Symposium, pp.221–233, July 2009.
[5] N. Regola and J.C. Ducom, "Recommendations for virtualization technologies in high performance computing," Proc. Second IEEE International Conference on Cloud Computing Technology and Science, pp.409–416, Dec. 2010.
[6] P. Luszczek, E. Meek, S. Moore, D. Terpstra, V.M. Weaver, and J. Dongarra, "Evaluation of the HPC Challenge Benchmarks in Virtualized Environments," Proc. 6th Workshop on Virtualization in High-Performance Cloud Computing, Aug. 2011.
[7] R. Takano, T. Ikegami, T. Hirofuchi, and Y. Tanaka, "Toward a practical "HPC Cloud": Performance tuning of a virtualized InfiniBand cluster," Proc. 6th International Conference on Ubiquitous Information Technologies and Applications (CUTE 2011), Dec. 2011.
[8] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif, "Black-box and Gray-box strategies for virtual machine migration," Proc. 4th USENIX Symposium on Networked Systems Design and Implementation, 2007.
[9] J. Lange and P. Dinda, "SymCall: Symbiotic virtualization through VMM-to-guest upcalls," Proc. 2011 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE 2011), pp.193–204, March 2011.
[10] P.H. Hargrove and J.C. Duell, "Berkeley lab checkpoint/restart (BLCR) for linux clusters," Proc. SciDAC 2006, 2006.
[11] R.K. Sahoo, A.J. Oliner, I. Rish, M. Gupta, J. Moreira, and S. Ma, "Critical event prediction for proactive management in large-scale computer clusters," Proc. 9th ACM international conference on knowledge discovery and data mining (SIGKDD), pp.426–435, 2003.
[12] A. Kivity, "KVM: The Linux virtual machine monitor," Proc. Ottawa Linux Symposium, pp.225–230, July 2007.
[13] Open MPI, http://www.open-mpi.org/
[14] B. Goglin, "High-performance message passing over generic ethernet hardware with open-MX," J. Parallel Computing, vol.37, pp.85–100, 2011.
[15] J. Hursey, J.M. Squyres, T.I. Mattox, and A. Lumsdaine, "The design and implementation of checkpoint/restart process fault tolerance for open MPI," Proc. 12th IEEE Workshop on Dependable Parallel, Distributed and Network-Centric Systems (DPDNS), March 2007.
[16] E. Zhai, G.D. Cummings, and Y. Dong, "Live migration with pass-through device for Linux VM," Proc. Ottawa Linux Symposium, pp.261–267, July 2008.
[17] W. Huang, Q. Gao, J. Liu, and D.K. Panda, "High performance virtual machine migration with RDMA over modern interconnects," Proc. IEEE International Conference on Cluster Computing, 2007.
[18] T. Hirofuchi, H. Nakada, S. Itoh, and S. Sekiguchi, "Reactive consolidation of virtual machines enabled by postcopy live migration," Proc. 5th international workshop on virtualization technologies in distributed computing (VTDC2011), pp.11–18, June 2011.
[19] R. Takano, H. Nakada, T. Hirofuchi, Y. Tanaka, and T. Kudoh, "Cooperative VM migration for a virtualized HPC cluster with VMM-bypass I/O devices," Proc. IEEE 8th International Conference on eScience, Oct. 2012.
[20] B. Cully, G. Lefebvre, D. Meyer, M. Feeley, N. Hutchinson, and A. Warfield, "Remus: High availability via asynchronous virtual machine replication," Proc. 5th USENIX Symposium on Networked Systems Design and Implementation, pp.161–174, 2008.
[21] Y. Tamura, "Kemari: Virtual machine synchronization for fault tolerance using DomT," Xen Summit 2008, 2008.
[22] A. Kangarlou, P. Eugster, and D. Xu, "VNsnap: Taking snapshots of virtual networked environments with minimal downtime," Proc. IEEE/IFIP Dependable Systems and Networks (DSN 2009), pp.524–533, June 2009.
[23] K. Chanchio, C. Leangsuksun, H. Ong, V. Ratanasamoot, and A. Shafi, "An efficient virtual machine checkpointing mechanism for hypervisor-based HPC systems," Proc. High Availability and Performance Computing Workshop, 2008.
[24] A.B. Nagarajan and F. Mueller, "Proactive fault tolerance for HPC with xen virtualization," Proc. 22nd International Supercomputing Conference (ISC 2007), 2007.
[25] J. Liu, W. Huang, B. Abali, and D.K. Panda, "High performance VMM-bypass I/O in virtual machines," Proc. USENIX Annual Technical Conference, pp.29–42, June 2006.
[26] W. Huang, J. Liu, M. Koop, B. Abali, and D.K. Panda, "Nomad: Migrating OS-bypass networks in virtual machines," Proc. 2007 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE), 2007.
[27] B. Davda and J. Simons, "RDMA on vSphere: Update and future directions." Open Fabrics Workshop, March 2012.
[28] R. Takano, H. Nakada, T. Hirofuchi, Y. Tanaka, and T. Kudoh, "Ninja migration: An interconnect-transparent migration for heterogeneous data centers," Proc. High-Performance Grid and Cloud Computing Workshop (HPGC 2013), May 2013.
[29] A. Gordon, N. Amit, N. Har'El, M. Ben-Yehuda, A. Landau, A. Schuster, and D. Tsafrir, "ELI: Bare-metal performance for I/O virtualization," Proc. 17th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2012), March 2012.
[30] H.B. Chen, R. Chen, F. Zhang, B. Zang, and P.C. Yew, "Mercury: Combining performance with dependability using self-virtualization," J. Computer Science and Technology, vol.27, no.1, pp.92–104, 2012.
[31] H. Eiraku, Y. Shinjo, C. Pu, Y. Koh, and K. Kato, "Fast networking with socket-outsourcing in hosted virtual machine environments," Proc. 24th ACM Symposium on Applied Computing, pp.310–317, 2009.

Ryousei Takano received his Ph.D. degree from Tokyo University of Agriculture and Technology in 2008. He joined AXE, Inc. in 2003. He joined Institute of Advanced Industrial Science and Technology (AIST) in 2008. His research interests include operating systems, virtualization, high performance networking, and distributed parallel computing. He is a member of ACM, IEEE, and IPSJ.

Hidemoto Nakada received his Ph.D. degree from the University of Tokyo in 1995. He joined the Electrotechnical Laboratory in 1995, which was merged into the National Institute of Advanced Industrial Science and Technology (AIST) in 2001. He also served as a visiting associate professor at the Tokyo Institute of Technology from 2001 to 2005. His interests lie in the area of parallel / distributed computing, including Grid and Cloud technologies. He is a member of ACM and IPSJ.

Takahiro Hirofuchi is a researcher of National Institute of Advanced Industrial Science and Technology (AIST) in Japan. He is working on virtualization technologies for advanced cloud computing and Green IT. He obtained his Ph.D. of engineering in March 2007 at the Graduate School of Information Science of Nara Institute of Science and Technology (NAIST). He obtained his BS of Geophysics at Faculty of Science in Kyoto University in March 2002. He is an expert of operating system, virtual machine, and network technologies.

Yoshio Tanaka received his Dr. Eng. degree in 1995 from Keio University. In 2000, he joined the Electrotechnical Laboratory which was re-organized as the National Institute of Advanced Industrial Science and Technology (AIST) in 2001. He is currently a principal researcher of Information Technology Research Institute of AIST. His research interests include middleware and security for distributed environments including Grid and Cloud. He is a member of ACM, IEEE CS, and IPSJ.

Tomohiro Kudoh received his Ph.D. degree from Keio University in Japan in 1992. He joined the National Institute of Advanced Industrial Science and Technology (AIST) in 2002. He currently serves the deputy director of the Information Technology Research Institute, AIST. In the past few years his research has focused on the network as a Grid infrastructure. His recent work also includes the G-lambda project, which is attenmpting to define an interface to manage the network as a Grid resource.