Migration Techniques in HPC Environments

Simon Pickartz[1,⋆], Ramy Gad[2,⋆], Stefan Lankes[1], Lars Nagel[2,⋆], Tim Süß[2], André Brinkmann[2], and Stephan Krempel[3]

[1] Institute for Automation of Complex Power Systems,
E.ON Energy Research Center, RWTH Aachen University, Aachen, Germany
{spickartz,slankes}@eonerc.rwth-aachen.de
[2] Zentrum für Datenverarbeitung, Johannes Gutenberg Universität, Mainz, Germany
{gad,nagell,suesst,brinkman}@uni-mainz.de
[3] ParTec Cluster Competence Center GmbH, Munich, Germany
krempel@par-tec.com

Abstract. Process migration is an important feature in modern computing centers as it allows for a more efficient use and maintenance of hardware. Especially in virtualized infrastructures it is successfully exploited by schemes for load balancing and energy efficiency. One can divide the tools and techniques into three groups: Process-level migration, virtual machine migration, and container-based migration.

This paper presents a qualitative and quantitative investigation of the different migration types for their application in High-Performance Computing (HPC). In addition to an overhead analysis of the various migration frameworks, our performance indicators include the migration time. The overall analysis suggests that VM migration has the most advantages and can even compete performance-wise.

The results are applied in the research project FaST addressing the problem of process scheduling in exascale environments. It is assumed that a shift in hardware architectures will result in a growing gap between the performance of CPUs and that of other resources like I/O. To avoid that these resources become bottlenecks, we suggest to monitor key performance indicators and, if conducive, trigger local amendments to the schedule requiring the efficient migration of jobs so that the downtime is reduced to a minimum.

1 Introduction

The fastest computers listed in the *Top 500* are able to execute 10^{16} FLOPS. The next generation of computer clusters will move into new dimensions and be a hundred times faster. Such *exascale computers* will not have significantly more nodes, but considerably more cores per node. It is predicted that this increase of CPU performance will not be matched by other resources resulting in an imbalance between CPU performance on the one hand and I/O performance on the other hand [1].

⋆ Supported by the Federal Ministry of Education and Research (BMBF) under Grant 01IH13004B (Project FaST).

L. Lopes et al. (Eds.): Euro-Par 2014 Workshops, Part II, LNCS 8806, pp. 486–497, 2014.

The FaST[1] project develops dynamic scheduling strategies balancing the system's load such that resource bottlenecks are avoided. It is assumed that the exclusive assignment of jobs to nodes or vice versa will be inefficient, if not impossible, for computing centers of the exascale area. In fact, it will be necessary to schedule (sub-)jobs subject to their resource requirements. The approach in FaST is twofold: (1) an initial placement of the jobs provided by a global scheduler, (2) local adjustments by the migration of jobs to other nodes during the applications' runtime.

In this paper we present an investigation of migration techniques that can be part of the solution to the second problem. We discuss their qualitative and quantitative properties and determine virtualization as the solution most suitable for FaST. Generally, there are three types of migration, namely process-level, virtual machine, and container-based migration. The first is supposed to have the least overhead, as it restricts the migrating only to the process and its context. Yet, the gathering of the context can be a problem and is certainly easier when most of it is already wrapped into a VM or container. Other advantages of Virtual Machines (VMs) and containers are the support of live migration and the ability to run on basically any system, while existing tools for process-level migration do not offer live migration and usually require a homogenous cluster. Finally, the decisive factors pro virtualization are (1) that in contrast to containers a more flexible range of application is provided, e. g., guest and host do not necessarily have to use equivalent operating systems, and (2) that the experiments conducted reveal a competitive performance of virtualization including the migration itself compared to the other approaches.

The rest of the paper is structured as follows: First we explain the different types of migration in Section 2, display their pros and cons, and give a detailed survey of the related work. In Section 3 we describe the experiments and analyze their results. Section 4 concludes the paper with a summary and future work.

2 Process Migration in HPC Environments

In this section we discuss three different approaches for the realization of process migration. The first, process-level migration, achieves minimal overhead by restricting the transferred data to the process and its context. Virtual machine migration provides more flexibility and a migration framework that can be integrated more easily. Finally, container-based migration is discussed depicting a compromise between these two approaches.

2.1 Process-Level Migration

Migration on the process-level is the operation of moving a process, i. e., the execution context of a running program including registers and physical memory addresses, from one node to another. Process-level migration can be regarded as a

[1] Find a Suitable Topology for Exascale Applications (FaST) is a project funded by Germany's Federal Ministry of Education and Research (BMBF).

special kind of Checkpoint/Restart (C/R) operation where a checkpoint is copied to another node before it is restarted [2]. While C/R mechanisms are intended to recover long-running applications in case of node failures, process migration techniques may have other motivations. Besides the prevention of application interruptions due to node failures [3], they can also be used for the conductance of readjustments to the cluster's workload to improve energy efficiency or balance the load more evenly, like in FAST.

There are several C/R implementations available such as Condor's checkpoint library, the libckpt library, and Berkley Lab Checkpoint/Restart (BLCR) [4–6]. We use BLCR for the evaluation of process-level migration because the open source tool was specifically designed for HPC applications. It targets at CPU and memory intensive batch-scheduled parallel jobs and consists of two components: a kernel module performing the C/R operations inside the Linux kernel and a shared library enabling the access to user-space data [7]. This library needs to be loaded with the application to activate the support for checkpointing. Applications using sockets, block devices, or SystemV IPC mechanisms are not natively supported by BLCR. However, Sankaran et al. developed in [8] an extension to LAM/MPI with a callback interface enabling any library or application code to cooperate in the C/R procedure. This allows for closing communication channels prior to the migration and restoring them afterwards [7]. Meanwhile, the callback interface is availabe for LAM/MPI 7.x, MPICH, and Open MPI [8,9].

For the evaluation of process-level migration, we chose Open MPI 1.7 and its BLCR plug-in. Migrations are initiated by the *ompi-checkpoint* command creating a checkpoint of the running MPI job on the source nodes. After killing the job and all its processes, the checkpoint file containing their states is copied to the destination nodes, and the job is restarted by calling *ompi-restart*. The successful restoration of the job demands all libraries and files required for its execution to be present in exactly the same version on all nodes participating in the migration and prelinking of shared libraries has to be disabled. Prelinking is a feature which is used by some Linux distributions to perform a relocation of library code in advance of its execution. This technique accelerates the startup of applications by the assignment of fixed addresses to shared libraries. Furthermore, the source and destination nodes should have the same kernel version and hardware architecture. A successful migration of a process to a remote node is only possible if all resources that were allocated at the origin, i. e., the *residual dependencies*, are provided by the migration target as well [10]. With resources like communication channels, open files, or subprocesses this is not possible, as the respective file descriptors would not be valid on the target host and had to be closed in advance of the migration. This restriction could require a non-transparent migration from the application's point of view.

2.2 Virtual Machine Migration

As an alternative to process-level migration we investigate the deployment of VMs which reduce the aforementioned problem of residual dependencies [11]. Open files and virtual I/O devices do not cause any problems as the according

descriptors are still valid within the resumed VM on the target node. The only residual dependencies that remain are the Instruction Set Architecture (ISA) as well as the hardware state of the virtualized devices. Since most hardware can be virtualized efficiently, these dependencies generally do not cause any issues. If the origin and target Virtual Machine Monitor (VMM) have the same hardware configuration, the latter only needs to receive the guest memory state and the guest device model state in order to start the VM on the new host. Thus, a migration transparent to the application can be realized.

I/O Virtualization. In contrast to CPUs and memory components of VMs, the virtualization of I/O devices may result in an unacceptable performance degradation. The emulation of high-performance networks like InfiniBand with native performance is still not possible. For this reason virtualization has mostly been disregarded in the area of HPC in the last years [12]. However, progress in this field of research accompanied by new hardware technologies changed this situation [13]. Driven by industry, a shift to cloud computing approaches can be observed in the area of HPC [14].

With Intel VT-d extensions it is possible to perform a physical device passthrough to a VM while providing DMA and interrupt isolation [15]. This technology gives I/O devices direct access to the memory space of a VM. The VM, in turn, is able to control the device by accessing the according hardware registers without intervention by the host system. However, this solution suffers from scalability issues as one physical device can only be assigned to exactly one VM at a time. Hence, if a single VM was used per high-performance process, one physical Host Channel Adapter (HCA) would be required per process. Such a setup would dramatically reduce the maximal amount of processes per node within a cluster.

A solution to this issue is addressed by the Peripheral Component Interconnect Special Interest Group (PCI-SIG) with the Single Root I/O Virtualization (SR-IOV) specification. This technology enables the native sharing of I/O devices by a replication of all necessary resources for each VM [16]. For this purpose, two new PCIe function types are introduced, namely Physical Functions (PFs) and Virtual Functions (VFs). An I/O device supporting SR-IOV may be configured to appear in the PCI configuration as multiple functions including one and only one PF. This function covers all PCIe capabilities including SR-IOV. Furthermore, there may be several VFs covering the necessary capabilities for data movement. Each of these VFs may then be assigned to one VM with the mechanisms described above. Although the VMs get the impression of possessing the I/O device exclusively, they share the same physical device with nearly native performance.

Hypervisor. There is a variety of virtualization techniques and tools today including Xen and KVM [17,18]. Although the former has been the tool of choice in the open source world in the past, KVM is taking over this status more and more [14]. While Xen is a bare-metal hypervisor, KVM is integrated into Linux as kernel-module, and hence benefits from existing resources like the scheduler,

the memory management, etc. The tight integration into the upstream Linux kernel with version 2.6.20 in 2007 allows KVM to take advantage of the kernels evolution [19]. Bugfixes and improvements within the kernel code will automatically apply to KVM-based systems using the current kernel version. In contrast, Xen is still not part of the Linux kernel and patches have to be applied explicitly. These facts led to the decision to focus on KVM as basis for the virtualization approach. It was further supported by performance evaluations showing that KVM is at least as good as other hypervisors [14].

KVM is providing full virtualization on x86 hardware depending on the VT-x or AMD-V hardware extensions [20, 21]. A VM is started as an ordinary Linux process that can be scheduled by the host system. If the VM is configured with more than one virtual CPU, one thread is created for each of them so that they can be scheduled individually. Furthermore, a migration framework supporting cold as well as live migration is already provided. Hence, KVM would allow for the realization of a first prototype of the migration framework within a narrow time frame.

2.3 Container-Based Migration

Traditional virtualiziation solutions like KVM result in multiple kernel instances running on one node. A light-weight alternative is *Container-based Virtualization* (or *Operating System Virtualization*) using the host-sytem kernel for the managment of so-called virtual containers as well. This concept aims at the provision of an isolation similar to full virtualization, but promises a better utilization and less overhead. An application running within a container can use standard system calls to interact with the server system but does not have to use hypercalls, e. g., when accessing virtualized I/O devices. However, this virtualization approach comes along with a certain inflexibility. It is not possible to run different operating systems on the same hardware and a crash of the kernel would halt the complete system, since it is shared among all instances.

OpenVZ[2] and LinuX Containers (LXC)[3] are typical representatives of this virtualization technique. In contrast to LXC, OpenVZ is not part of the vanilla Linux kernel, although efforts have been made to add their container functionality to LXC. Regola and Ducom conducted an analysis of OpenVZ with respect to its application in HPC and could show that some container-based virtualization solutions offer near native CPU and I/O performance [22]. Yet, since OpenVZ comes with its own kernel, which does not support our new InfiniBand adapters from Mellanox, we did not analyze OpenVZ more deeply. It would complicate the integration of new hardware. Figure 1 vizualizes the difference between container-based and full virtualization in the case of LXC and KVM, respectively. Both examples present a setup with two VMs und two containers, respectively. It is visible that container-based virtualization provides lower overhead as only one kernel instance is required for the host and all containers.

[2] `https://openvz.org`
[3] `http://www.linuxcontainers.org`

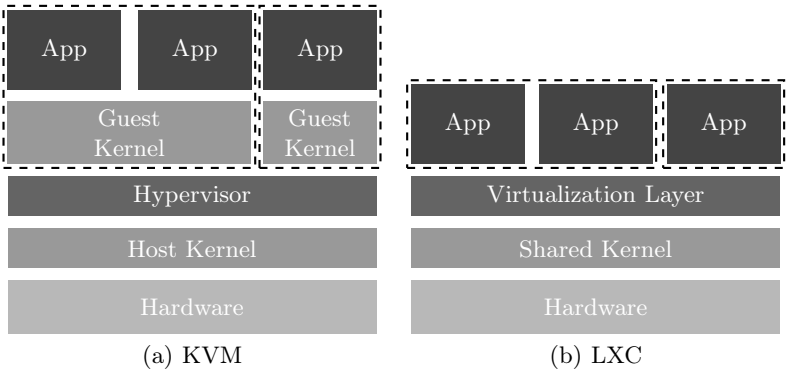

Fig. 1. Comparison of KVM- and LXC-based Virtualization

With Checkpoint/Restore In Userspace (CRIU)[4], there exists a mechanism similar to BLCR for LXC and OpenVZ. Yet, in contrast to BLCR, CRIU allows live migration[5] which can be valuable for many applications. Currently, CRIU offers no support for checkpointing of applications using file locks, block devices, or System V IPC mechanisms.

Our test system is based on Centos 6.5 constituting an extremely stable system. Yet, the LXC part of CentOS is not up to date and it is not possible to pass through general character devices from the host to the guest. Without this feature, which is supported by newer versions of LXC, it is not possible to use InfiniBand in LXC guests. The situation is similar concerning CRIU, which requires Linux kernel version 3.11 or newer. For these reasons, the quantative evaluation of LXC and CRIU was postponed.

3 Evaluation

The focus of this paper is a comprehensive evaluation of process-level and VM migration. Besides a qualitative comparison of these two approaches, a quantitative evaluation is indispensable to make an informed decision. Here, two key figures are important, namely the general overhead imposed by the respective migration technique on the application's performance and the characteristics of the migration itself in terms of the time needed to transfer a process from one node to the other.

All benchmarks were performed on an InfiniBand-based cluster comprising four NUMA nodes exposing 32 virtual cores, each on two sockets with 8 physical cores. While the hardware assembly is generally equal to all of the systems, two nodes are equipped with Intel SandyBridge CPUs (E5-2650) clocked at 2 GHz. The other two host systems are supplied with newer generation Intel IvyBridge

[4] `http://criu.org/Main_Page`
[5] `http://criu.org/Live_migration`

CPUs (E5-2650 v2) clocked at 2.6 GHz. The InfiniBand fabric is built by using Mellanox hardware. Therefore, each host system is equipped with a ConnectX-3 VPI two-port HCA implementing the PCIe 3.0 standard. The theoretical peak throughput for point-to-point connections is at 56 Gbit/s in accordance with the FDR signaling rate and the HCAs implement the SR-IOV technology which can be enabled and disabled by flashing the adapter's firmware.

To allow for a comparison of the results, we applied the same optimization techniques in the test scenarios. On the one hand, low-level benchmarks were used that came as binary with the Mellanox OFED stack in version 2.1-1.0.6. On the other hand, applications and benchmarks that are avaiable as source code were compiled with the same level of optimization.

3.1 Overhead

In order to see the impact of either migration technique on the runtime of our test application, we started with a general analysis of the overhead caused by them without actually performing a migration. In case of process-level migration with BLCR, the requirement of disabling the prelinking feature might have a negative impact on the application's performance. In contrast, the VM approach does not demand any modifications of the executed code. However, the additional software layer may introduce a certain overhead even if full virtualization is applied. The VM runs in *guest-mode* on x86 hardware with virtualization support. On the execution of a privileged instruction, i. e., an instruction that traps if the CPU is in user-mode while it does not trap in kernel-mode, the CPU switches to *host-mode* returning control back to the hypervisor. Moreover, an additional overhead might be introduced by the SR-IOV technology. Despite its realization on the hardware layer, the logic for the multiplexing of the VFs to the hardware consumes time that might result in performance penalties.

Microbenchmarks. For the investigation of the influence of the respective migration technique on the communication performance, a microbenchmark analysis was performed in terms of throughput and latency measurements. Therefore, we compared the results when using one of the two approaches with those obtained by native execution on the host systems. This was done on the MPI layer by using a self-written PingPong application and Open MPI 1.7 with BLCR support. For the measurement of the throughput and latency on the InfiniBand layer the `ib_write_bw` and `ib_write_lat` tools were used that come with the OFED stack.

The latencies on the InfiniBand layer in Table 1 reveal a slight impact of the SR-IOV technology. Although increased by roughly 27 % compared to native host execution, the communication latency between two guests equipped with passed-through VFs is only at 1.16 µs. As the pure pass-through of the InfiniBand hardware does not have any influence on the latencies, this difference must be caused by the SR-IOV technology itself. The additional software layer in terms of the Open MPI results in a further increase of the latency. With 1.48 µs the

Table 1. Latencies in µs (RTT/2)

Layer	Native	Pass-Through	SR-IOV	BLCR
InfiniBand	0.91	0.90	1.16	–
Open MPI	1.19	1.21	1.48	1.61

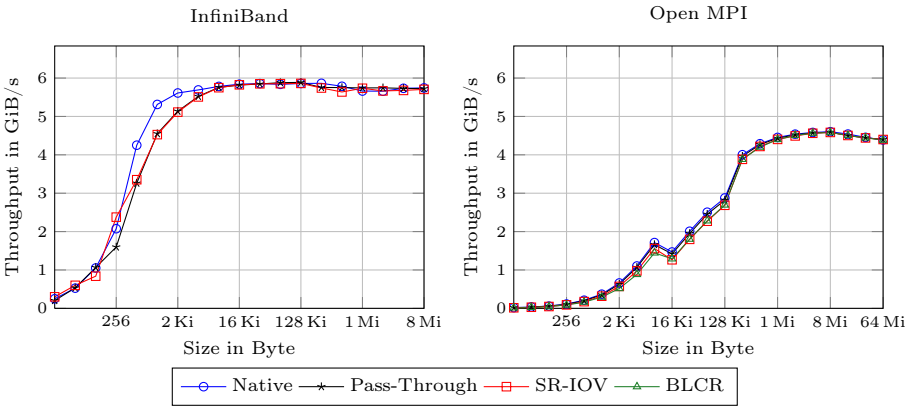

Fig. 2. Throughput Results

VM approach still performs slightly better than the native host execution while having the BLCR feature enabled.

The throughput results (see Fig. 2) show a similar trend. For small message sizes the SR-IOV technology as well as the BLCR framework have a marginal influence on the achievable performance. Larger messages instead can be transferred at nearly native performance. However, the results of the microbenchmark analysis are so close in all cases that they do not allow to make a decision for either of the frameworks.

Application Benchmarks. One reference application in the FAST project is mpiBLAST [23]. This is a parallelization of the BLAST algorithm applied in biological research that searches in a short query sequence of DNA or amino acids for similarities within a database of longer sequences.

Like in the previous section, an overhead analysis was conducted comparing the application runtime with the different frameworks. The left part of Figure 3 summarizes the results of different runs with 8 to 32 processes. A comparison between the native environment and execution with enabled BLCR yields a slight increase of the runtime in the order of 1 %. This increase is rather negligible and affiliates BLCR good performance characteristics in this application scenario. In contrast, the execution of the same scenario within a virtualized environment decelerates the runtime by 7 % to 8 %. This certainly constitutes an important performance degradation compared to BLCR and native execution, however it

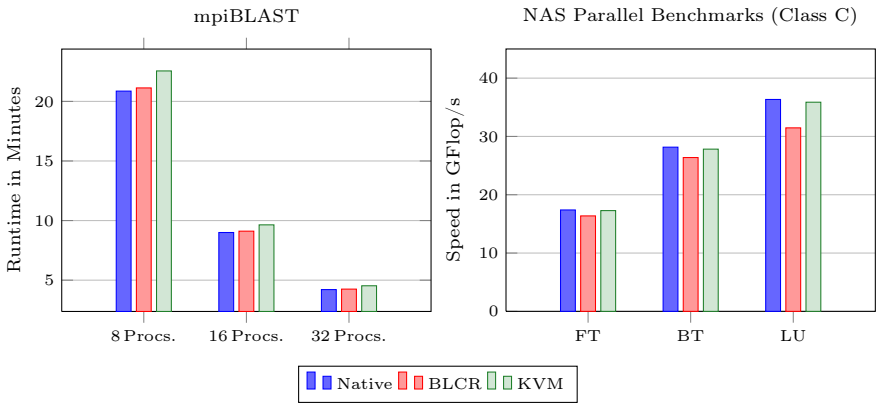

Fig. 3. Application Benchmarks

should be kept in mind that there has only been put little effort in the optimization of the KVM environment.

A second test was performed with the NAS Parallel Benchmarks (NPBs). This benchmark suite targets at the emulation of large-scale fluid dynamics applications [24]. We used the FT kernel calculating a discrete 3D Fast Fourier Transformation as well as the two pseudo applications, BT and LU, which are solvers for linear equation systems. The results were obtained by starting the benchmarks with 16 processes on two hosts, i. e., NUMA effects do not have to be considered as all processes on one host could be pinned to the same socket. Here, the results are quite different from the mpiBLAST evaluation. In fact, the virtualization layer reduces the performance by only 1 % outperforming BLCR which, in turn, generates an overhead of 6 % to 14 %. We think that this overhead is not caused by the BLCR but rather by the implementation of the callback interface inside Open MPI in order to support checkpointing with BLCR.

These results lead to the conclusion that the actual overhead generated by the particular approach is highly application dependent. Both solutions exhibit fairly good performance results compared to native execution. While BLCR shows better results for mpiBLAST, though with decreasing advance for higher process counts, the virtualization approach has the edge over process-level migration for the NPBs while offering more flexibility as discussed before.

3.2 Migration Time

Finally, an investigation of the migration time was performed. This is a key value particular important for the evaluation of the differen migration techniques. In this timeframe the nodes participating in the migration are not responsive and these phases depict an overhead that has to be compensated by sophisticated scheduling strategies to improve the overall utilization of the cluster. To perform this evaluation, the following scenario was conducted.

An mpiBLAST job with three processes on two different nodes was started, i. e., on one node a single process was launched while the other holded two processes. The single process was then migrated to a third node not yet participating in the execution. For the evaluation of the virtualization approach with KVM, two VMs were launched on the origin nodes. Instead of moving the mpiBLAST process to a remote node, the VM holding this process was then transferred to the same remote node.

We started with an evaluation of the *overall* migration time from initiating the migration command until its successful return. With the BLCR framework we were able to migrate the MPI process within 0.51 s averaged over multiple runs. In contrast, the VM migration required 2.87 s constituting an important overhead. The VMs were configured with 256 MiB of RAM. Although a lightweight CentOS 6.5 installation was used for the guest systems, the migration time might be reduced by the usage of a minimal kernel only providing the necessary environment for executing MPI jobs (e. g. a system configured with Buildroot). We could observe a dependency between the migration time and the assigned memory when migrating a KVM guest. Hence, a kernel optimized to memory utlization might improve the results presented above. However, this dependency has to be further investigated in this context. As we used the libvirt tool set to access the KVM hypervisor, time is not only consumed by the migration itself but also by preparatory tasks like establishing connections to the daemons on the respective host systems. Moreover, the process representing the VM on the source host has to be properly removed subsequent to the successful migration.

To get an impression of the *real* downtime of the VM, we wrote a socket-based PingPong application using the UDP protocol. The server was started on the VM being migrated. It listens for incoming UDP packets on a dedicated port and directly responds to the sender. The client was started on one of the cluster nodes not participating in the migration. This runs two threads, a sender and a receiver thread. The first posts with a fixed interval of 500 µs UDP packets containing a sequence number while the receiver thread constantly listens for the responses from the server running in the VM. This benchmarks allows for the determination of packet losses due to unresponsiveness of the VM during migration. The actual downtime may then be determined by multiplying the amount of packets that were not answered with the time interval the packets have been transmitted.

For the mpiBLAST scenario described above we captured a downtime of about 1.2 s reducing the previously measured advantage of process-level migration. With KVM it would even be possible to perform a *live-migration*, i. e., not stopping the VM during migration. Hu et al. could show that this technique allows for a considerable decrease of the downtime to the order of 0.2 s in the best case [25]. Furthermore, it should be noted that the current implementation of BLCR requires *all* process to be halted in advance of the migration. Within the VM approach applications may benefit from the fact that only those process have to be freezed at some point in time that are situated within the migrated VM.

4 Conclusion

In this paper two migration techniques have been examined, namely process-level migration using BLCR and virtual machine migration on top of KVM. We have conducted a qualitative and a quantitative comparison of these two techniques. In particular, we have studied the overhead on the application's performance imposed by each solution and the characteristics of the migration itself in terms of the time needed to transfer a process from one node to the other. In accordance with the presented results, we favor virtual machine migration over process-level migration for the FaST project. The overhead imposed by the virtualization layer is acceptable and offers more flexibility in terms of a greater application range.

In the near future, we will further investigate LXC as a complement to our migration framework. Although it imposes some restrictions compared to full virtualization (e. g., host and guest cannot use different kernels), it might have a better performance which may be more important than flexibility in some cases. Should it be possible to support both virtualization techniques in FaST, we will offer them and let end-users choose the one more suitable for their domain. Besides the implementation of the migration framework, future tasks in FaST include the development of an agent-based monitoring system and a scheduler on top of it, which triggers process migrations based on the resource utilization.

References

1. Dongarra, J.: Impact of Architecture and Technology for Extreme Scale on Software and Algorithm Design. Presented at the Department of Energy Workshop on Cross-cutting Technologies for Computing at the Exascale (February 2010)
2. Roman, E.: A Survey of Checkpoint/Restart Implementations. Technical report, Lawrence Berkeley National Laboratory, Tech (2002)
3. Wang, C., Mueller, F., Engelmann, C., Scott, S.: Proactive process-level live migration in hpc environments. In: International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2008, pp. 1–12 (November 2008)
4. Litzkow, M., Tannenbaum, T., Basney, J., Livny, M.: Checkpoint and Migration of UNIX Processes in the Condor Distributed Processing System. Technical Report UW-CS-TR-1346, University of Wisconsin – Madison Computer Sciences Department (April 1997)
5. Plank, J.S., Beck, M., Kingsley, G., Li, K.: Libckpt: Transparent Checkpointing under Unix (1995)
6. Duell, J.: The Design and Implementation of Berkeley Lab's Linux Checkpoint/Restart. Technical report, Lawrence Berkeley National Laboratory (2003)
7. Hargrove, P.H., Duell, J.C.: Berkeley Lab Checkpoint/Restart (BLCR) for Linux Clusters. Journal of Physics: Conference Series 46(1), 494 (2006)
8. Sankaran, S., Squyres, J.M., Barrett, B., Lumsdaine, A.: The LAM/MPI Checkpoint/Restart Framework: System-Initiated Checkpointing. In: Proceedings of LACSI Symposium, Sante Fe, pp. 479–493 (2003)
9. Hursey, J., Squyres, J.M., Lumsdaine, A.: A Checkpoint and Restart Service Specification for Open MPI. Technical Report TR635, Indiana University, Bloomington, Indiana, USA (July 2006)

10. Milojičić, D.S., Douglis, F., Paindaveine, Y., Wheeler, R., Zhou, S.: Process migration. ACM Computing Surveys (CSUR) 32(3), 241–299 (2000)
11. Clark, C., Fraser, K., Hand, S., Hansen, J.G., Jul, E., Limpach, C., Pratt, I., Warfield, A.: Live Migration of Virtual Machines. In: Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation, NSDI 2005, vol. 2 (2005)
12. Ranadive, A., Kesavan, M., Gavrilovska, A., Schwan, K.: Performance implications of virtualizing multicore cluster machines. In: Proceedings of the 2nd Workshop on System-level Virtualization for High Performance Computing, HPCVirt 2008, pp. 1–8. ACM, New York (2008)
13. Birkenheuer, G., Brinkmann, A., Kaiser, J., Keller, A., Keller, M., Kleineweber, C., Konersmann, C., Niehörster, O., Schäfer, T., Simon, J., Wilhelm, M.: Virtualized HPC: a contradiction in terms?. Softw., Pract. Exper. 42(4), 485–500 (2012)
14. Younge, A.J., Henschel, R., Brown, J.T., von Laszewski, G., Qiu, J., Fox, G.C.: Analysis of Virtualization Technologies for High Performance Computing Environments. In: Proceedings of 2011 IEEE International Conference on Cloud Computing (CLOUD), pp. 9–16. IEEE (2011)
15. Intel Virtualization Technology for Directed I/O. Technical report, Intel Corporation (2013)
16. Intel LAN Access Division: PCI-SIG SR-IOV Primer. Technical Report 2.5, Intel Corporation (January 2011)
17. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the Art of Virtualization. SIGOPS Oper. Syst. Rev. 37(5), 164–177 (2003)
18. Kivity, A., Kamay, Y., Laor, D., Lublin, U.: kvm: the Linux Virtual Machine Monitor. In: Proceedings of the Linux Symposium, vol. 1, Ottawa, Ontario, Canada, pp. 225–230 (June 2007)
19. Nussbaum, L., Anhalt, F., Mornard, O., Gelas, J.P.: Linux-based virtualization for HPC clusters. In: Proceedings of the Linux Symposium (July 2009)
20. Uhlig, R., Neiger, G., Rodgers, D., Santoni, A.L., Martins, F.C.M., Anderson, A.V., Bennett, S.M., Kagi, A., Leung, F.H., Smith, L.: Intel Virtualization Technology. Computer 38(5), 48–56 (2005)
21. Virtualization, A.: Secure Virtual Machine Architecture Reference Manual. AMD Publication (2005)
22. Regola, N., Ducom, J.C.: Recommendations for Virtualization Technologies in High Performance Computing. In: Proceedings of 2nd IEEE International Conference on Cloud Computing Technology and Science (CloudCom), pp. 409–416 (November 2010)
23. Darling, A., Carey, L., Feng, W.: The design, implementation, and evaluation of mpiBLAST. In: Proceedings of ClusterWorld (2003)
24. Bailey, D.H., Barszcz, E., Barton, J.T., Browning, D.S., Carter, R.L., Dagum, L., Fatoohi, R.A., Frederickson, P.O., Lasinski, T.A., Schreiber, R.S., Simon, H.D., Venkatakrishnan, V., Weeratunga, S.K.: The NAS Parallel Benchmarks. International Journal of High Performance Computing Applications 5(3), 63–73 (1991)
25. Hu, W., Hicks, A., Zhang, L., Dow, E.M., Soni, V., Jiang, H., Bull, R., Matthews, J.N.: A quantitative study of virtual machine live migration (August 2013)