PROCEEDINGS OF SCIENCE

Live process migration for load balancing and/or fault tolerance

A. Reber* **and P. Väterlein**
Hochschule Esslingen - University of Applied Sciences
E-mail: adrian.reber@hs-esslingen.de,
peter.vaeterlein@hs-esslingen.de

With the mean time between failure in the range of hours for large jobs one of today's problems in high performance computing is frequent job abortion and data loss. Writing checkpoints helps to restart the job after a failure, but a lot of computing time is wasted waiting for checkpoints to be written. To minimize the lost computing time and to offer a way to move running jobs from one node to another for load balancing we suggest live process migration.

The International Symposium on Grids and Clouds (ISGC) 2012,
February 26 - March 2, 2012
Academia Sinica, Taipei, Taiwan

*Speaker.

© Copyright owned by the author(s) under the terms of the Creative Commons Attribution-NonCommercial-ShareAlike Licence.

PoS(ISGC 2012)031

1. Introduction

A HPC compute job on a compute cluster usually consists of a number of individual processes on different compute nodes, which communicate by sending messages over some sort of network interconnect. To achieve this, many applications use the MPI standard [1], usually each process will remain on the node it has been created on, until it is terminated.

However, there are situations, where it might be desirable to move one or more processes from one node of the cluster to another. One reason might be an improved load balancing, or the optimization of the job with respect to the network topology. On a large cluster, usually a large number of jobs are executed at the same time. The optimal distribution of the processes on the cluster might well change during the cluster's lifetime.

Another reason to move a process on another node is an imminent hardware failure, which often are signaled by a rise in CPU temperature, or other sensor data. On Petaflop systems the frequency of hardware failure increases, due to an increased number of components[2]. The mean time between failure (MTBF) of jobs running on 30000 or even 50000 cores is in the range of hours[3]. To make efficient use of these large clusters, jobs running on such systems need a mechanism to continue to run, even if one of the components (compute node, interconnect, cooling, ...) fails.

The classic method to achieve this goal, is to regularly store snapshots of the application on disk (checkpointing), and to restart the application later using this snapshot to minimize the loss of data and/or compute time. Usually, this checkpoint/restart mechanism has to be implemented by the application. The checkpoint/restart mechanisms must be re-implemented over and over again. The ability to do checkpointing on the operating system (OS) level is rare and typically depends on certain properties of the applications. A well known example is the Parallel Environment Runtime Edition of IBM's proprietary AIX operating system [4]. An example of OS level checkpointing for Linux is Berkeley Labs C/R (BLCR)[5]). Unfortunately BLCR is not part of the official kernel tree and has therefore never reached a broad adoption outside of high performance computing.

To be successful, an OS-level checkpoint/restart solution has to be as transparent as possible [6]. Special hardware requirements and/or the need to recompile applications are serious barriers for its adoption.

There are potential drawbacks when using checkpointing/restarting to avoid losses due to failing hardware components. The bandwidth of the storage subsystems has not kept pace with the size of the used and available memory in existing HPC systems[7].

If thousands of nodes start to write a checkpoint snapshot, a very high load is placed on the storage backends. This can lead to situations where up to 50% of the computing time is wasted[8] waiting for the checkpoint I/O operations to finish.

In addition, the interval at which checkpoints are written has to be selected carefully. Otherwise it might lead to much longer waiting periods for all users[9].

An alternative to the checkpoint/restart mechanism is to move the process directly from a source node to a target node without storing the memory image on disk. This live migration, which has been suggested before [10], requires that the source node must remain available for the duration of the migration. If the process is migrated for load balancing reasons, live migration should be far superior to checkpoint/restart. It also might serve as a tool for proactive fault-tolerance, even

PoS(ISGC 2012)031

though the failure of a component can be difficult to predict. Therefore, it is not easy to determine the perfect moment for the migration[11].

If, however, a node fails without warning, then checkpoint/restart is still the only way to minimize losses of data and/or CPU time. As a consequence, live migration will never completely replace checkpoint/restart, but could instead be an additional instrument to improve the efficiency and reliability of HPC compute clusters.

2. Process migration

In this paper, we suggest live process migration as a partial alternative to checkpoint/restart, in order to help to avoid the above mentioned drawbacks of the latter. In addition, live process migration should not only decrease unnecessary pressure from the storage backend. Furthermore, the downtime of the process during migration should also be minimized.

The migration of a process might be initiated by the cluster load balancer, which should be aware of the state of all nodes in the cluster, or some other monitoring tool. First, the process to be migrated, shall be suspended by the OS kernel. Afterwards, the process table entry and the uspace shall be copied to the target node. This small amount of data is sufficient to restart the process on the target node. The subsequent page faults initiate the transfer of the required memory pages on demand. In parallel, the transfer of all other memory pages shall be initiated to avoid memory fragmentation between nodes.

3. Implementation

Due to the conceptual similarity of checkpoint/restart and live process migration, we started to evaluate previously published approaches to the former. In particular, we studied two projects, which are aimed to OS-level checkpoint/restart for Linux.

The first approach by Ladaan and Hallyn [12] includes changes to numerous subsystems of the Linux kernel (*kernel based*). The second project by Emelyanov and Gorcunov [13] is using well-known Linux kernel interfaces, which are accessible from the user space, as far as possible (*userspace based*).

The intention of the authors of [12] was to provide a checkpoint/restart implementation on OS-level for Linux, which should become part of the upstream Linux kernel tree. However, the development of this project ended with the kernel version 2.6.37 of Linux.

We took the existing code and ported it to the Linux kernel version 3.2.0. Based on these changes we implemented live process migration, which is working now for simple processes. But even though the development has been done in continuous consultation with the Linux community, the chances of the kernel based checkpoint/restart to be accepted for inclusion into the mainstream Linux kernel sources, are very small. This is due to the large number of kernel subsystems, which are affected by the over 100 patches, necessary to implement checkpoint/restart. Without the inclusion into the standard kernel, however, a wide acceptance of kernel based checkpoint/restart and live process migration is not to be expected.

As the *kernel based* approach has little chance of becoming the adopted standard for checkpoint/restart, recently an alternative project has been published [13], which aims to minimize the

PoS(ISGC 2012)031

changes required to the Linux kernel by using existing interfaces as much as possible. This approach has been met with much greater acceptance and some parts have already been accepted and will be part of the upcoming Linux kernel release 3.3.

Using both approaches for checkpoint/restart, we were able to implement live process migration and move processes from one system to another. At present, the versions of the kernel and the shared libraries in use have to be the same on the source and target systems. Using statically linked binaries reduces the requirements on the systems involved. On the other hand, it increases the size of the binary and therefore the time required for the migration.

Additionally the instruction set architecture of the source and destination system has to be same. Otherwise, the hardware may be different and it does not matter if it is real or virtualized.

4. Outlook

Even though we were able to demonstrate the feasibility of live process migration based on the mentioned approaches to checkpoint/restart, this is only the beginning.

A large number of issues has to be addressed, e.g. the behavior of file handles, network sockets, child processes, to name just a few. Some environment variables have to be modified upon migration (e.g. the hostname), others have to be kept the same.

To be useful in the HPC field, it is not sufficient to simply enable live process migration in the operating system, as it also has to be supported by MPI. However, an approach has been published which enables pause/resume in connection with the migration of a process, which is part of a MPI job[14, 15].

An alternative to the migration of single processes or process trees might be the migration of complete virtual machines [16]. However, the latter requires the transfer of much more data and hence will usually use much more network bandwidth. Migration of a single process is also much more flexible, since other processes on the source system will not be affected.

5. Conclusion

In this paper, we presented the first steps to implement live process migration within compute clusters running Linux. This will be a valuable complement to the well-known checkpoint/restart mechanism to improve the efficiency and reliability of HPC clusters.

The live migration of single processes is also in many cases preferable to the migration of complete virtual machines, due to its higher performance and larger flexibility.

We would like to acknowledge the use of the computing resources provided by bwGRiD[17], member of the German D-Grid initiative, funded by the Ministry for Education and Research (Bundesministerium für Bildung und Forschung) and the Ministry for Science, Research and Arts Baden-Wuerttemberg (Ministerium für Wissenschaft, Forschung und Kunst Baden-Württemberg).

The authors would like to thank M. Resch and U. Küster (High Performance Computing Center Stuttgart) for the useful discussions and their continuing support.

PoS(ISGC 2012)031

References

[1] Message Passing Interface Forum, *MPI: A Message-Passing Interface Standard, Version 2.2*, http://www.mpi-forum.org/docs/mpi-2.2/mpi22-report.pdf, 2009.

[2] B. Schroeder and G. A. Gibson, *Understanding failures in petascale computers*, *Journal of Physics: Conference Series* **78**, http://stacks.iop.org/1742-6596/78/i=1/a=012022, 2007.

[3] B. Schroeder and G. A. Gibson, *A large-scale study of failures in high-performance computing systems*, in *Proceedings of the International Conference on Dependable Systems and Networks*, http://portal.acm.org/citation.cfm?id=1135532.1135705, IEEE Computer Society, Washington, DC, 2006.

[4] IBM Corporation, *Parallel Environment Runtime Edition for AIX: Operations and Use*, http://publibfp.dhe.ibm.com/epubs/pdf/c2367811.pdf, 2012.

[5] J. Duell, *The design and implementation of Berkeley Lab's linux Checkpoint/Restart*, 2003.

[6] O. Laadan and D. Phung and J. Nieh, *Transparent Checkpoint-Restart of Distributed Applications on Commodity Clusters*, 2005.

[7] N. DeBardeleben and J. Laros and J. T. Daly and S. L. Scott and C. Engelmann and B. Harrod, *High-End Computing Resilience: Analysis of Issues Facing the HEC Community and Path-Forward for Research and Development*, http://www.csm.ornl.gov/~engelman/publications/debardeleben09high-end.pdf, 2009.

[8] K. Ferreira and R. Riesen and R. Oldfield and J. Stearley and J. Laros and K. Pedretti and R. Brightwell and T. Kordenbrock, *Increasing Fault Resiliency in a Message-Passing Environment*, 2009.

[9] W. M. Jones and J. T. Daly and N. DeBardeleben, *Impact of sub-optimal checkpoint intervals on application efficiency in computational clusters*, in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, http://doi.acm.org/10.1145/1851476.1851509 ACM, Ney York, 2010.

[10] F. Cappello and H. Casanova and R. Yves, *Checkpointing vs. Migration for Post-Petascale Supercomputers*, in *Proceedings of the 2010 39th International Conference on Parallel Processing*, http://dx.doi.org/10.1109/ICPP.2010.26, IEEE Computer Society, Washington, DC, 2010.

[11] H. Jin and Y. Chen and H. Zhu and X. Sun, *Optimizing HPC Fault-Tolerant Environment: An Analytical Approach*, in *39th International Conference on Parallel Processing (ICPP'10)*, http://dx.doi.org/10.1109/ICPP.2010.80, 2010.

[12] O. Ladaan and S. E. Hallyn, *Linux-CR: Transparent Application Checkpoint-Restart in Linux*, in *The Linux Symposium 2010*, http://systems.cs.columbia.edu/files/wpid-ols2010-linuxcr.pdf, Ottawa, 2010.

[13] http://criu.org/, 2012

[14] J. Hursey and J. M. Squyres and T .I. Mattox and A. Lumsdaine, *The Design and Implementation of Checkpoint/Restart Process Fault Tolerance for Open MPI*, in *Proceedings of the 21st IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, http://doi.acm.org/10.1145/1851476.1851509, IEEE Computer Society, Long Beach, 2007.

[15] C. Wang and F. Mueller and C. Engelmann and S. L. Scott, *Hybrid Checkpointing for MPI Jobs in HPC Environments*, in *ICPADS*, http://dx.doi.org/10.1109/ICPADS.2010.48, 2010.

PoS(ISGC 2012)031

[16] VMware, *vMotion Architecture, Performance, and Best Practices in VMware vSphere 5*, http://www.vmware.com/resources/techresources/10202, 2011.

[17] http://www.bw-grid.de/, 2012.

PoS(ISGC 2012)031