Final Scientific/Technical Report: MOLAR: Modular Linux and Adaptive Runtime Support for High-end Computing Operating and Runtime Systems

DOE award: DE-FG02-05ER25664
reporting period: 02/05-01/09 (Y1—Y4)

PI: Frank Mueller
Department of Computer Science
North Carolina State University

1 Executive Summary

MOLAR is a multi-institution research effort that concentrates on adaptive, reliable,and efficient operating and runtime system solutions for ultra-scale high-end scientific computing on the next generation of supercomputers. This research addresses the challenges outlined by the FAST-OS - forum to address scalable technology for runtime and operating systems — and HECRTF — high-end computing revitalization task force — activities by providing a modular Linux and adaptable runtime support for high-end computing operating and runtime systems.

The MOLAR research has the following goals to address these issues.

- Create a modular and configurable Linux system that allows customized changes based on the requirements of the applications, runtime systems, and cluster management software.
- Build runtime systems that leverage the OS modularity and configurability to improve efficiency, reliability, scalability, ease-of-use, and provide support to legacy and promising programming models.
- Advance computer reliability, availability and serviceability (RAS) management systems to work cooperatively with the OS/R to identify and preemptively resolve system issues.
- Explore the use of advanced monitoring and adaptation to improve application performance and predictability of system interruptions.

The overall goal of the research conducted at NCSU is to develop scalable algorithms for high-availability without single points of failure and without single points of control.

2 Accomplishments

Throughout the project, NCSU has developed a scalable membership protocol for a group communication systems without single points of failure and without single points of control. The membership protocol combines scalability with low stabilization overheads. The algorithm is the key component of a group communication framework utilizing a fully decentralized protocol that maintains group memberships in presence of faults. The group communication component enhances the RAS capabilities developed within the overall project. Our group membership service and integrates with the services for distributed control and data replication in the mid-tier of the high-availability framework for active/active head node redundancy. It is also designed to support fault tolerance for compute nodes in MPI runtime systems, as detailed in the following contributions:

1. NCSU developed runtime system mechanisms to support scalable group communication with fluctuating number of nodes, reuse of network connections, transparent coordinated checkpoint scheduling and a BLCR enhancement for job pause.
2. NCSU further developed a process-level live migration mechanism as a proactive FT to complement reactive one. Through health monitoring, a subset of node failures can be anticipated when one's health deteriorates. A novel process-level live migration mechanism supports continued execution of applications during much of processes migration. This scheme is integrated into an MPI execution environment to transparently sustain health-inflicted node failures, which eradicates the need to restart and requeue MPI jobs.
3. NCSU is further developing an incremental checkpointing technique for MPI tasks complementary to full checkpoints to capture only data changed since the last checkpoint. This significantly reduces the

size of checkpoint files and the overhead of checkpoint operations while only moderately increasing restart costs relative to a restore from a single, full checkpoint.

The objectives listed in the executive summary have been met as follows:

- Modularity and Configurability: We have devised a set of methods to provide fault tolerance in HEC systems that can be deployed individually or combined in a complementary manner.
- Efficiency: The complementary approach of fault-tolerant mechanisms results in significantly lower overhead of the RAS software layer.
- Cooperation: The RAS layer and the OS/runtime layers cooperate in providing fault tolerance. Our contributions range from Linux kernel enhancements (dirty pit support at page level within the memory management layer), Linux kernel module enhancements (for BLCR to support advanced, lower overhead checkpointing) and the runtime layer (via LAM/MPI enhancements in both the LAM daemon and the MPI runtime layer, which are applicable to other MPI systems as well).
- Monitoring and Adaptation: Proactive fault tolerance provided through process-level live migration provides adaptive capabilities of fault tolerance in the presence of immanent failures detected by health monitoring.

Benefits of this project extend particularly to high-end computing systems, such as Jaguar at ORNL's National Center for Computational Sciences, that increasingly have to counter component failures on a regular basis. Our techniques provide a low overhead means to shield the effects of component failures from applications and users. Activities past the funding period include technology transfer to incorporate our BLCR enhancements into LBNL's BLCR code base (in cooperation with LBNL), transfer of enhancements from LAM/MPI into OpenMPI and further development of advanced RAS techniques **(all subject to obtaining sufficient funds beyond the MOLAR project)**.

The overall aim is to fully integrate our techniques into community open-source codes such that future software deployment can directly benefit from our results.

3 Research and Educational Activities

Partnership

The project is conducted in conjunction with Oak Ridge National Laboratory, the Ohio State University and Louisiana State University. Frequent meetings enforce the collaborative efforts of this project and their overall integrity.

3.1 Research Activities

Activities are reported in several parts:

1. We provide an overview of the general research objectives.
2. We briefly summarize results from past reporting periods.
3. We report new results obtained during the final period.

General Research Objectives

In this project, we investigate a scalable approach to reconfigure the communication infrastructure after node failures within the runtime system of the communication layer. Building on the experience of group communication frameworks, we propose a decentralized (peer-to-peer) protocol that maintains group membership in the presence of faults.

While existing approaches provide either scalability or small reconfiguration overhead, our protocol combines these features. Instead of seconds for reconfiguration, our protocol shows overhead in the order of micro-seconds over TCP on FastEther and over MPI on Myrinet/GM. Our protocol can be configured to match the network topology to increase communication throughput.

We utilize radix trees to implicitly encode routing information into node IDs and additionally represent the tree structure as an array (dynamically resized upon node joins/failures) to provide access to the data structure of individual nodes in constant time.

We also verify our experimental results against a performance model to assess the scalability of the approach. Experimental results further indicate that configuration choices of the protocol for asynchronous communication may depend on parameters, such as latency and gap.

Our membership service combines the best of both worlds, the scalability of group membership and the performance of existing fault-tolerant mechanisms within high-performance runtime systems. We are currently assessing the protocol's suitability for deployment with the services for distributed control and data replication in the mid-tier of the high-availability framework for active/active head node redundancy.

Our approach is also designed to support fault tolerance for compute nodes in MPI runtime systems in the future, e.g., within the MPI Component Architecture (MCA) of OpenMPI, specifically as an add-on to the Point-to-point Management Layer (PML).

Overall, our approach is general and can be applied for any group membership service or in other frameworks that require scalable group communication, such as efficient multicast services.

Results from Past Reporting Periods

In year 1, research activities have produced 1 paper. Key findings included:

1. We have designed and implemented a group communication protocol that provides the basis for key services in high-availability of clusters.
2. The approach is fully decentralized to avoid bottlenecks.
3. The protocol design embeds a radix tree representation that provides scalability and efficiency. Single-node information can be accessed in constant time. The routing information is natively encoded into node IDs of the tree representation. It also provides the means to quickly reconfigure routing paths upon node failures.
4. The protocol can sustain single and multi-node failures. It will rebuild internal communication structures in an efficient manner.
5. The protocol is configurable to reflect switch topology.
6. The protocol scales for large number of nodes.
7. The stabilization time for reconfiguration has been assessed both qualitatively and quantitatively.

8. Experiments show that the qualitative model matches the quantitative measurements obtained for TCP and GM/Myrinet.
9. The overall stabilization time is in the order of microseconds. This is considerably lower than an past work on group communication (generally in the order of seconds).

In Y2, research activities have produced 1 paper submission and 1 M.S. thesis. The findings include:

1. We have developed a transparent mechanism for job pause within LAM/MPI+BLCR that allows live nodes to remain active and roll back to the last checkpoint while failed nodes are dynamically replaced by spares before resuming from the last checkpoint.
2. The mechanism complements LAM/MPI with the scalable group communication framework based on our work from past reporting periods.
3. We designed a fault detector based on a single timeout mechanism with the group communication framework.
4. The mechanism supplements LAM/MPI with a novel, decentralized, scalable scheduler that transparently controls periodic checkpointing and triggers process migration and job pause upon node failures.
5. The overhead of the scheduler is only in the order of hundreds of microseconds.
6. The mechanism avoids requeuing overhead upon node failures.
7. BLCR is supplemented with a cr_pause mechanism to reuse the existing process resource, and LAM/MPI is enhanced to reuse the network connections between live nodes upon the faults. Both decrease the overhead of job pause.
8. We provide a special process migration approach, which enables seamless continuation of execution across node failures and is suitable for proactive fault tolerance with diskless migration, to dynamically replace the failed nodes with spares.
9. Experiments show that the overhead for job pause is comparable to that of a complete job restart, albeit at full transparency and automation with the additional benefit of reusing existing resources and continuing to run within the scheduled job.
10. The scheme offers additional potential for savings through incremental checkpointing and proactive diskless live migration. We are currently working on investigating these topics.

In Y3, research activities have produced 2 papers. The findings include:

1. We have developed a process level live migration mechanism to complement reactive with proactive FT. The mechanism allows continued execution of an application during much of processes migration.
2. This scheme is integrated into an MPI execution environment, LAM/MPI + BLCR, to transparently sustain health-inflicted node failures, which also eradicates the need to restart and requeue MPI jobs.
3. By exploiting health monitoring capabilities, a subset of node failures can be anticipated due to deteriorating health of a node.

4. The mechanism supplements LAM/MPI with a novel, decentralized, scalable scheduler that transparently coordinates the live migration at the process level based on the information from the health monitor.

5. LAM/MPI is enhanced to create the connections between the migrated MPI processes and the live processes on the operational nodes.

6. The mechanism incrementally transfers the state of the process through network connections to avoid the overhead of storage I/O for reading/writing checkpoint files. This also avoids roll-backs to prior checkpoints.

7. BLCR is supplemented with a series of utilities (cr_precopy_source, cr_precopy_dest, cr_stop, cr_start, and cr_suspend_continue etc.) to support incremental live migration and coordinate the synchronization among the migrated processes and the live processes of the MPI application on the operational nodes.

8. We developed a feature at the kernel level to check the dirty status of the process memory pages transparently for the incremental migration without incurring much of any overhead.

9. We were deploying experiments with NPB benchmarks on an x86_64 cluster and planned to compare the result with that of OS-level virtualization live migration solution.

10. This proactive FT scheme has the potential to prolong the meantime- to-failure, reactive schemes can lower their checkpoint frequency in response, which implies that proactive FT can lower the cost of reactive FT.

New Results Obtained during the Final Period

In the final year, research activities have produced 1 paper. The following novel key findings can be reported:

1. For the process-level live migration mechanism we developed in Y3, we deployed and finished experiments with NPB benchmarks on an x86_64 cluster. Experiments indicate that 1-6.5 seconds of prior warning are required to successfully trigger live process migration while similar operating system virtualization mechanisms require 13-24 seconds. This self-healing approach complements reactive FT by nearly cutting the number of checkpoints in half when 70% of the faults are handled proactively.

2. We have developed an incremental checkpointing technique for MPI tasks complementary to full checkpoints to capture only data changed since the last checkpoint. This significantly reduces the size of checkpoint files and the overhead of checkpoint operations while only moderately increasing restart costs relative to a restore from a single, full checkpoint.

3. This hybrid full/incremental checkpoint/restart scheme is integrated into an MPI execution environment, LAM/MPI + BLCR, to provide a fault-tolerant MPI runtime system.

4. The mechanism supplements LAM/MPI with a decentralized, scalable scheduler that coordinates the full or incremental checkpoint commands based on user-configured intervals or the system environment, such as the execution time of the MPI job, storage constraints for checkpoint files and the overhead of preceding checkpoints.

5. LAM/MPI is enhanced to drain the in-flight messages among the MPI tasks before the incremental checkpoint, and restore them after the checkpoint.

6. The mechanism incrementally saves the state of the process to the storage.

7. BLCR is supplemented with a series of utilities (cr_full_checkpoint, cr_incr_checkpoint and cr_fullplusincr_restart etc.) to support hybrid full/incremental checkpoint/restart.

8. A set of three files serve as storage abstraction for a checkpoint snapshot: *Checkpoint file a* contains the memory *page content*, *i.e.*, the data of only those memory pages modified since the last checkpoint; *Checkpoint file b* stores memory *page addresses*, *i.e.*, address and offset of the saved memory pages for each entry in *file a*; and *Checkpoint file c* covers other *meta information*, *e.g.*, linkage of threads, register snapshots, and signal information pertinent to each thread within a checkpointed process / MPI task.

9. We used the same feature as we used for live migration at the kernel level to check the dirty status of the process memory pages transparently for the incremental memory pages without incurring much of any overhead.

10. We are deploying experiments with NPB benchmarks and mpiBLAST on an x86_64 cluster and plan to compare the result with that of single full checkpoint solution.

3.2 Educational Activities

Educational activities include coverage of high-end computing, compilation and optimization for parallelism and performance analysis/tuning in two graduate classes, namely Parallel Systems and Code Optimization for Scalar and Parallel Programs. The classes place strong emphasis on recent research in high-performance computing. Students receive critical skills for careers in these areas. Industry contacts confirm that these skills are highly valued.

4 Training

In the first year, 1 M.S. and 1 Ph.D. student were trained in conjunction with this project. In the second year, 1 Ph.D. student was trained, and 1 M.S. thesis was produced. In the third and fourth year, 1 Ph.D. student was trained, and this Ph.D. student is scheduled to graduate within 6 months of the end of the project period (Summer 2009). The student is interviewing with ORNL opening the door to continued technology transfer of our technology into the DOE realm and HEC systems in general.

Skills acquired through the project are in the area of team work, communication and writing, the latter through contributing to paper publications. This includes presentation skills (invited talks, poster presentations at a premier conference, and thesis defenses). Furthermore, a number of independent studies were supervised by the PIs in related areas emphasizing problem solving, design and implementation as well as writing skills.

5 Internet

The following web site disseminates publications, abstracts and talks for this project.

http://moss.csc.ncsu.edu/∼mueller/molar.html

6 Contributions

This work pools a community of collaborators from labs, universities, and industry, to investigate adaptive, reliable, and efficient solutions to the problems surrounding Operating and Runtime Systems (OS/R) for Extreme Scale Scientific Computation. Building on the current open-source operating system, Linux, target High-End Computing (HEC) applications for the next generation of supercomputers. These HEC OS/Rs must scale to the levels predicted by hardware architects for both shared memory and distributed memory platforms. Furthermore, applications must operate efficiently and reliably on any of these architectures as transparently as possible. As described in recent reports by HECRTF and the DoE Scales workshop, system software is a key challenge in exploiting the promise of extreme-scale scientific computing. The MOLAR project contributes to the research by:

- Creating a modular and configurable Linux system that allows customized changes based on the requirements of the applications, runtime systems, and cluster management software;
- Building runtime systems that leverage the OS modularity and configurability to improve efficiency, reliability, scalability, ease-of-use, and provide support to legacy and promising programming models;
- Advancing computer RAS management systems to work cooperatively with the OS/R to identify and preemptively resolve system issues; and
- Exploring the use of advanced monitoring and adaptation to improve application performance and predictability of system interruptions.

6.1 Contributions To Human Resources

This project contributes to human resource development by educating students through research, providing new educational materials in the classroom, and giving students the communication and writing skills needed to advance science and engineering for future generations. Hands-on computer systems and new educational materials effectively integrate research and teaching.

6.2 Contributions to Resources for Research and Education

Hardware and software has been installed to facilitate high-throughput simulation and is managed by the PIs and students. The new infrastructure benefits the PIs' departments and colleges beyond the scope of the project.

7 Publications

”MOLAR: adaptive runtime support for high-end computing operating and runtime systems” by Christian Engelmann, Stephen L. Scott, David E. Bernholdt, Narasimha R. Gottumukkala, Chokchai Leangsuksun, Jyothish Varma, Chao Wang, Frank Mueller, Aniruddha G. Shet, P. Sadayappan in ACM SIGOPS Operating Systems Review, Vol. 40, No. 2, April 2006, pages 63-72.

”Scalable, Fault-Tolerant Membership for MPI Tasks on HPC Systems” by Jyothish Varma, Chao Wang, Frank Mueller, Christian Engelmann, Stephen L. Scott in ICS'06.

”A Job Pause Service under LAM/MPI+BLCR for Transparent Fault Tolerance” by Chao Wang, Frank Mueller, Christian Engelmann, Stephen L. Scott in IPDPS'07.

”Proactive Fault Tolerance for HPC with Xen Virtualization” by A. Nagarajan and F. Mueller and C. Engelmann and S. Scott in ICS'07.

”Proactive Process-Level Live Migration in HPC Environments” by Chao Wang, Frank Mueller, Christian Engelmann, Stephen L. Scott in SC'08

8 Additional Documents

A copy of the publication(s) is submitted with the report.