High Availability through Distributed Control*

C. Engelmann, S. L. Scott, G. A. Geist
Computer Science and Mathematics Division
Oak Ridge National Laboratory, Oak Ridge, TN 37831, USA
{engelmannc,scottsl,gst}@ornl.gov
http://www.csm.ornl.gov

Abstract

Cost-effective, flexible and efficient scientific simulations in cutting-edge research areas utilize huge high-end computing resources with thousands of processors. In the next five to ten years the number of processors in such computer systems will rise to tens of thousands, while scientific application running times are expected to increase further beyond the Mean-Time-To-Interrupt (MTTI) of hardware and system software components. This paper describes the ongoing research in heterogeneous adaptable reconfigurable networked systems (Harness) and its recent achievements in the area of high availability distributed virtual machine environments for parallel and distributed scientific computing. It shows how a distributed control algorithm is able to steer a distributed virtual machine process in virtual synchrony while maintaining consistent replication for high availability. It briefly illustrates ongoing work in heterogeneous reconfigurable communication frameworks and security mechanisms. The paper continues with a short overview of similar research in reliable group communication frameworks, fault-tolerant process groups and highly available distributed virtual processes. It closes with a brief discussion of possible future research directions.

1 Introduction

Cost-effective, flexible and efficient scientific simulations in cutting-edge research areas, such as fusion energy, nano-technology and protein folding, utilize huge high-end computing resources with thousands of processors. In the next five to ten years the number of processors in such computer systems will rise to tens of thousands in order to keep up with performance demands. Furthermore, scientific application running times are expected to increase further despite or even because of the availability of such enormous computing power. For example, a real-time earthquake forecast just for the U.S. state of California using thousands of seismic detectors would need not only a gigantic amount of sustained computational performance, but would also need 24x7 availability in order to save lives and protect property by broadcasting warning messages and preemptively cutting off gas and fuel pipelines of affected areas. Another example deals with long (for six months) running climate simulations. Such data intensive applications perform regular backups (checkpoints) of huge amounts of data that is only ever read to restore processes during a recovery from expected or unexpected system interrupts (write-once/read-never backups in a failure-free case). These data backups significantly reduce efficiency.

The Mean-Time-To-Interrupt (MTTI) of hardware and system software components is still the dominant factor of scientific application running times today. Unexpected failures and scheduled maintenance are causing complete or partial system outages, which seriously affects availability and efficiency of scientific applications. High availability software environments for parallel and distributed scientific computing will try to counter such system interrupts by allowing scientific applications to dynamically adapt to changing system configurations caused by single or multiple node (processor) outages. A more resilient middleware layer provides high availability for parallel computing services, such as message passing, event notification and resource management. It automatically detects and recovers from partial system outages using dynamic reconfiguration without the need to reboot all nodes, thus improving availability and increasing efficiency of scientific applications.

This paper first describes the ongoing research in heterogeneous adaptable reconfigurable networked systems (Harness) and then documents its recent achievements in the area of high availability distributed virtual machine environments for parallel and distributed scientific computing. It

*This research is sponsored by the Mathematical, Information, and Computational Sciences Division; Office of Advanced Scientific Computing Research; U.S. Department of Energy. The work was performed at the Oak Ridge National Laboratory, which is managed by UT-Battelle, LLC under Contract No. De-AC05-00OR22725.

continues with a short overview of similar past and ongoing research projects in reliable group communication frameworks, fault-tolerant process group protocols and highly available distributed virtual processes. It closes with a brief discussion of possible future research directions.

2 Harness

The research in Harness is an ongoing collaborative effort between the Oak Ridge National Laboratory, the University of Tennessee and the Emory University. It primarily focuses on providing a pluggable light-weight heterogeneous distributed virtual machine (or DVM) environment, where clusters of PCs, workstations, and "big iron" computers can all be used together as one giant, high-performance computer (in the spirit of its widely-used predecessor, "Parallel Virtual Machine" - PVM [17]).

A variety of experiments and system prototypes are involved to explore lightweight pluggable frameworks, adaptive reconfigurable runtime environments, assembly of scientific applications from plug-ins, parallel plug-in paradigms, highly available DVMs, fault-tolerant message passing, fine-grain security mechanisms and dynamic heterogeneous reconfigurable communication frameworks. Currently, there are three system prototypes, each concentrating on different research issues. The teams at the Oak Ridge National Laboratory [11, 13, 22] and at the University of Tennessee [14, 15, 23] provide different C variants, while the team at Emory University [21, 24, 25, 27] maintains a Java-based alternative.

In general, the Harness software (figure 1) consists of two major parts: a runtime environment (kernel) and a set of plug-ins. A multi-threaded user-space kernel manages a set of non-, single and/or multi-threaded dynamically loadable plug-ins (i.e. native shared libraries or Java classes). While the kernel provides only basic functions, such as plug-in management and child process control, plug-ins may provide a wide variety of services needed in fault-tolerant parallel and distributed scientific computing, such as messaging, scientific algorithms and resource management. Multiple kernels can be aggregated to form a DVM that acts as one distributed multi-threaded kernel.

3 Distributed Control in Harness

Scientific simulations usually run much longer than the expected MTTI of hardware and system software components. Various checkpoint/restart schemes can provide certain levels of fault-tolerance that may involve losing recent computation in case of a failure and high backup communication costs. This is essentially caused by the inability of the middleware layer (e.g. message passing, resource management, etc.) to adapt to unexpected failures, to continue to

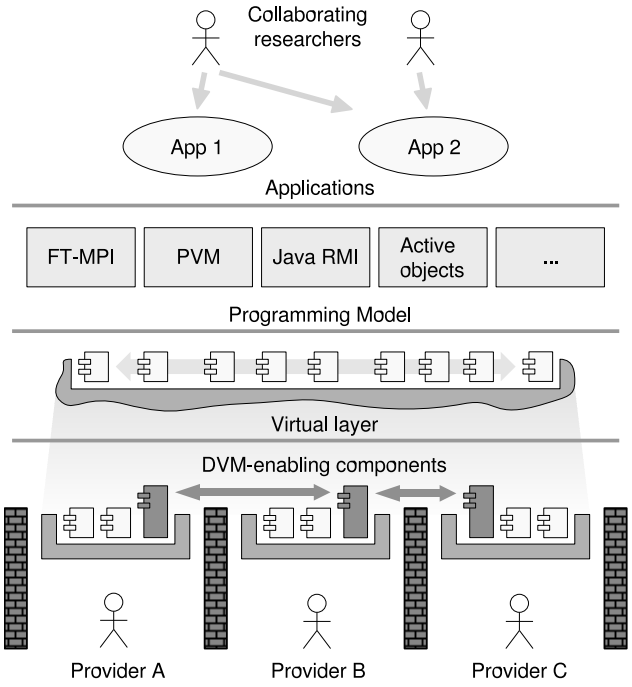

Figure 1. Harness Architecture

run in a degraded mode and to inform the application of its configuration changes caused by failures. For example, any failure usually requires a complete restart of all computational processes of a job. In contrast, fault-tolerant message passing, like in FT-MPI [15, 16], has the capability to reconfigure and to continue despite failures, while the application just needs to restore the faulty process.

The Harness research at the Oak Ridge National Laboratory mainly focuses on a highly available, adaptive and reconfigurable DVM environment capable of detecting and surviving failures in a dynamic and consistent fashion without the need to restart and rollback to a backup state. A highly available DVM service provides an encapsulation of a few hundred to a few thousand multi-threaded Harness kernel processes in one distributed heterogeneous virtual Harness kernel process. High availability is achieved by replication of the DVM service state on multiple server processes. If one or more server processes fails, the surviving ones continue to provide the DVM service because they know the state. The Harness DVM will continue to exist while at least one server process is still alive.

Since every server process of a DVM is itself part of the global DVM state and is able to change it, distributed control [13] is needed to consistently manage state changes, state replication, failure detection and recovery.

Distributed control is the ability to steer a distributed virtual process, that is composed of multiple distributed real processes connected by some way of communication

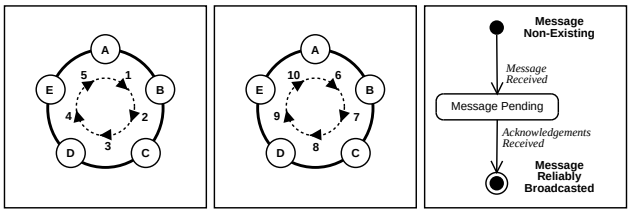

Figure 2. Reliable Broadcast in a Ring

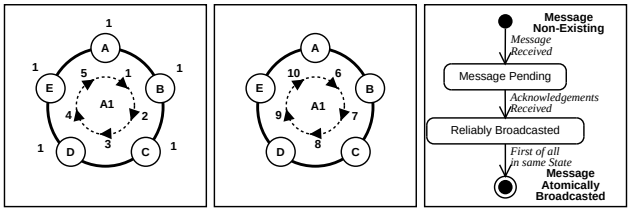

Figure 3. Atomic Broadcast in a Ring

(TCP/IP, shared memory, etc), in virtual synchrony, like one real process. While each real process has its own local process state, all local process states together form the global state of the distributed virtual process.

The global state is actively replicated to all real processes to ensure high availability of the distributed virtual process. All real processes receive global state change requests in the same order. They change their local state and their local copy of the global state as directed by the state change request. Global state change results may also need to be replicated if those real processes that do not have to change their local states cannot predict the outcome of a local state change of other real processes. In this case, all real processes receive global state change results in the same order they received global state change requests.

In contrast to distributed control, distributed locking [9] uses a database abstraction model to perform global state changes. A global state database is replicated to all real processes on change and a global lock needs to be acquired by a real process to change the global state. The locking mechanism may be a token that is passed around to all real processes or just to those that require the lock.

Both models use group communication [7, 8, 26], such as Reliable Broadcast and Atomic Broadcast, for fault-tolerant message delivery and for global message ordering. A Reliable Broadcast ensures message delivery using acknowledgment schemes with timeouts or via more sophisticated failure detectors. An Atomic Broadcast additionally ensures global message order using lockstep token passing mechanisms or more refined message numbering algorithms. Both, Reliable and Atomic Broadcast, are based on message broadcast primitives that can range from TCP/IP multicast to ring or tree based peer-to-peer systems.

The distributed control in Harness [13] relies on a ring

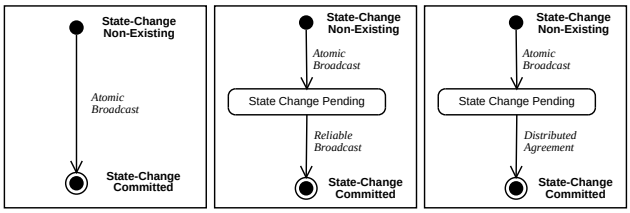

Figure 4. State Change Transaction Types

based TCP/IP peer-to-peer network (figures 2 and 3). Sending a message around the ring twice performs a Reliable Broadcast. A message is considered unstable, when it is received the first time, and stable when it is seen a second time. The algorithm terminates when any member receives it a third time. Detecting link failures and reconnecting the ring guarantees fault-tolerance, while the algorithm does not rely on the originating node to perform message state transitions. The Atomic Broadcast additionally involves a message numbering scheme that uses the deterministic message order on the ring and node priorities to order simultaneous messages. A Distributed Agreement algorithm uses collective communication to determine the final result in case of a global state change that involves local state change results of more than one Harness kernel (figure 4).

Several system prototypes of the distributed control for Harness exist to demonstrate its behavior and to allow experiments with the algorithms. For example, one prototype is able to communicate in both directions in the ring trading off bandwidth to reduce the latency of sending a message around the ring. We experienced inconsistencies while using the state change conditions from the unidirectional ring. This was partially due to the anticipated fact that state changes of messages occur always on two nodes in the ring. After further examination, it was determined that the state change conditions do not consider those cases in which messages in one direction of the ring may advance significantly faster than in the other direction. More extensive research is needed here to develop state transitions that cover all possible message phase combinations.

One of the current research efforts targets the incorporation of recent advances in heterogeneous multi-protocol remote method invocation (RMI). RMIX [25] is a dynamic heterogeneous reconfigurable communication framework, initially developed by the Harness team at Emory University, that allows Java and C applications to communicate via TCP/IP using various RMI/RPC protocols, like Sun RPC, Java RMI and SOAP. In addition to standard synchronous RMI/RPC mechanisms, RMIX also allows support for asynchronous and one-way invocations, which suits the messaging needs of the peer-to-peer distributed control in Harness. The RMIX framework itself uses Harness plug-in loading technology and consists of a base library and

a set of runtime protocol provider plug-ins. The provider plug-ins are responsible for the protocol as well as for the TCP/IP communication. In the future, provider plug-ins may support different inter-process communication methods, e.g. pipes and shared memory, or even more alternative ways of communication, such as e-mail.

Ongoing work also includes security mechanisms that go beyond simple autorization and authentication by providing an adjustable control of security levels ranging from basic to fine grain. This work mostly focuses on the Java solution for Harness (H2O [27]). It is also based on a one-to-one relationship model, not supporting more advanced group security mechanisms, such as group keys [2].

4 Related Work

There are several past and ongoing research projects [1, 3, 4, 5, 6, 10, 28] in the area of group communication, fault-tolerant process groups and highly available distributed virtual processes. Their application background usually involves two different networked computing areas: distributed computing, such as high availability middleware frameworks for scientific parallel computing, and solutions for closely coupled systems, such as highly available air traffic control and stock market trading.

Group communication systems in distributed computing rely on Internet standards (TCP, UDP) and try to reduce synchronous communication to counter latencies in wide area networks. The grade of consistency may vary from strong, with ultimate synchrony, to weak, accepting inconsistencies in case of failures or even during normal operation, since recovering from infrequent inconsistencies may be more efficient than always maintaining strong consistency. Another model embraces the distributed agreement paradigm, where all members of a group finally agree on a consistent state (equilibrium) after a period of uncertainty.

In contrast, solutions for highly available closely coupled systems can take advantage of predictable low latencies. Proprietary communication standards and global clock synchronization can improve the overall system performance and robustness up to the point of providing real-time capabilities despite failures and recoveries. Distributed agreement mechanisms are here only used for mission critical applications to counter incorrect information from false sensor readings or errant computation.

Some group communication solutions provide configurable frameworks, such as Coyote [4] and Ensemble [5], where different protocol layers/modules can be stacked or just simply plugged together. In this case, the group communication layer can be configured to fit application requirements and hardware features, which is similar to the Harness plug-in concept. However, the combination of protocols can be very complex and a deep protocol stack may result in a significant performance impact. Furthermore, such solutions tend to support only TCP and UDP as communication methods. Memory communication (distributed shared, global addressable) and other networking technology that is not IP-based and/or has additional features for performance tuning are not supported.

Security mechanisms range from non-existent for completely enclosed systems to common collaborative techniques, such as secure socket layer (SSL) and public key infrastructure (PKI), for frameworks that communicate over wide-area networks. Most solutions are peer-to-peer based, i.e. rely on multiple one-to-one connections. Recent research in a more advanced secure group layer (SGL) [2] bundles a reliable group communication system, a group access control mechanism and a group Diffie-Hellman protocol to provide a comprehensive and practical secure group communication platform. A group key agreement protocol ensures secure communication.

5 What Next

In our experience, research in group communication, fault-tolerant process groups and highly available distributed virtual processes has mostly been focused on specific applications and/or communication technologies. Implementations that are directly wired into applications are making it really hard to reproduce successes and require application programmers to understand the details of the communication algorithms and protocols. Modular, configurable frameworks provide more flexible support with different protocols, but current solutions are limited to very specific communication technologies, such as TCP/IP. Furthermore, each framework is based on its own abstraction model and uses different core technologies, such as Harness plug-in loading, with proprietary APIs.

For example, todays high-end computing systems are usually composed out of a single head node and hundreds or thousands of compute nodes. The head node provides parallel computing system services, such as a job scheduler, while the compute nodes run scientific applications. More advanced platforms, like Cray Red Storm or IBM BlueGene/L, additionally move local system services (file system I/O, etc.) from compute nodes to separate service nodes. This significantly reduces the amount of ”OS noise”, i.e. system interrupts. However, head nodes and service nodes are single points of failure and control. Failures or maintenance of these nodes can render a partition or even the complete system unaccessible and unmanageable with otherwise healthy compute nodes.

Multiple head and service nodes can solve this problem by providing some kind of high availability. Traditional active/hot-standby variants, such as HA-OSCAR [18, 19, 20], double up head and service nodes using shared

reliable disk storage and IP cloning or a redundant network. This leaves standby nodes idle and the fail-over/fail-back procedures are not without cost. In contrast, the active/active model of highly available distributed virtual processes, like the Harness DVM, utilizes all head/service nodes and provides more efficient high availability.

Service applications, like the job scheduler, have to be modified in order to implement active/hot-standby or active/active high availability. Each service might end up with a different model, algorithm and communication mechanism if directly implemented. So, which group communication framework, heterogeneous computing platform or middleware layer should be used to accomplish head/service node high availability? Is this high availability framework available on all common HPC platforms? Does the API reflect only one particular high availability model?

A high availability software framework is needed, that is extremely light-weight, modular and portable with a common API that supports different high availability models, algorithms and communication mechanisms. The main objective should be to enable existing proprietary solutions with different communication models and different distributed locking and control models to be moved out of their middleware layer into this framework as pluggable modules. Module implementations with various data replication and distributed control strategies using the common API would then allow adaptation to system properties and high availability needs. While applications can take advantage of data replication and distributed control algorithms without the need to implement them directly, the modular architecture also enables other researchers to easily contribute their solutions based on the common API. This will also support further research in efficient ultra-scale compute node replication strategies [12] for systems with hundreds of thousands of processors, where application checkpoint/restart to central storage is not feasible anymore.

References

[1] D. Agarwal. Totem: A reliable ordered delivery protocol for interconnected local-area networks. *PhD Thesis, University of California, Santa Barbara*, 1994.

[2] D. A. Agarwal, O. Chevassut, M. Thompson, and G. Tsudik. An integrated solution for secure group communication in wide-area networks. *Proceedings of ISCC 2001*, pages 22–28, 2001.

[3] K. Berket, D. A. Agarwal, P. M. Melliar-Smith, and L. E. Moser. Overview of the InterGroup protocols. *Lecture Notes in Computer Science: Proceedings of ICCS 2001*, 2073:316–325, 2001.

[4] N. T. Bhatti, M. A. Hiltunen, R. D. Schlichting, and W. Chiu. Coyote: a system for constructing fine-grain configurable communication services. *ACM Transactions on Computer Systems*, 16(4):321–366, 1998.

[5] K. Birman, B. Constable, M. Hayden, J. Hickey, C. Kreitz, R. van Renesse, O. Rodeh, and W. Vogels. The Horus and Ensemble projects: accomplishments and limitations. *Proceedings of DISCEX 2000*, 1:149–161, 2000.

[6] K. P. Birman and R. van Renesse. *Reliable Distributed Computing with the ISIS Toolkit*. IEEE Computer Society Press, 1993.

[7] G. V. Chockler, I. Keidar, and R. Vitenberg. Group communication specifications: A comprehensive study. *ACM Computing Surveys*, 33(4):1–43, 2001.

[8] F. Cristian. Synchronous and asynchronous group communication. *Communications of the ACM*, 39(4):88–97, 1996.

[9] N. Desai and F. Mueller. Scalable hierarchical locking for distributed systems. *Parallel and Distributed Computing*, 64(6):708–724, 2004.

[10] D. Dolev and D. Malki. The Transis approach to high availability cluster communication. *Communications of the ACM*, 39(4):64–70, 1996.

[11] W. R. Elwasif, D. E. Bernholdt, J. A. Kohl, and G. A. Geist. An architecture for a multi-threaded Harness kernel. *Lecture Notes in Computer Science: Proceedings of PVM/MPI User's Group Meeting 2001*, 2131:126–134, 2001.

[12] C. Engelmann and A. Geist. A diskless checkpointing algorithm for super-scale architectures applied to the fast fourier transform. *Proceedings of CLADE 2003*, pages 47–52, 2003.

[13] C. Engelmann, S. L. Scott, and G. A. Geist. Distributed peer-to-peer control in Harness. *Lecture Notes in Computer Science: Proceedings of ICCS 2002*, 2330:720–728, 2002.

[14] G. E. Fagg, A. Bukovsky, and J. J. Dongarra. Harness and fault tolerant MPI. *Parallel Computing*, 27(11):1479–1495, 2001.

[15] G. E. Fagg, A. Bukovsky, S. Vadhiyar, and J. J. Dongarra. Fault-tolerant MPI for the Harness metacomputing system. *Lecture Notes in Computer Science: Proceedings of ICCS 2001*, 2073:355–366, 2001.

[16] FT-MPI at University of Tennessee, Knoxville, TN, USA. Available at http://icl.cs.utk.edu/ftmpi.

[17] G. A. Geist, A. Beguelin, J. J. Dongarra, W. Jiang, R. Manchek, and V. S. Sunderam. *PVM: Parallel Virtual Machine: A Users' Guide and Tutorial for Networked Parallel Computing*. MIT Press, Cambridge, MA, USA, 1994.

[18] HA-OSCAR at Louisiana Tech University, Ruston, LA, USA. Available at http://xcr.cenit.latech.edu/ha-oscar.

[19] I. Haddad, C. Leangsuksun, and S. Scott. HA-OSCAR: Towards highly available linux clusters. *Linux World Magazine*, March 2004.

[20] I. Haddad, C. Leangsuksun, and S. Scott. HA-OSCAR: The birth of highly available OSCAR. *Linux Journal*, November 2003.

[21] Harness at Emory University, Atlanta, GA, USA. Available at http://www.mathcs.emory.edu/harness.

[22] Harness at Oak Ridge National Laboratory, Oak Ridge, TN, USA. Available at http://www.csm.ornl.gov/harness.

[23] Harness at University of Tennessee, Knoxville, TN, USA. Available at http://icl.cs.utk.edu/harness.

[24] D. Kurzyniec, V. S. Sunderam, and M. Migliardi. PVM emulation in the Harness metacomputing framework - Design and performance evaluation. *Proceedings of CCGRID 2002*, pages 282–283, 2002.

[25] D. Kurzyniec, T. Wrzosek, V. Sunderam, and A. Slominski. RMIX: A multiprotocol RMI framework for Java. *Proceedings of IPDPS 2003*, pages 140–145, 2003.

[26] S. Mishra and L. Wu. An evaluation of fbw control in group communication. *IEEE/ACM Transactions on Networking*, 6(5):571–587, 1998.

[27] V. Sunderam and D. Kurzyniec. Lightweight self-organizing frameworks for metacomputing. *Proceedings of HPDC 2002*, pages 113–124, 2002.

[28] B. Whetten, T. Montgomery, and S. M. Kaplan. A high performance totally ordered multicast protocol. *Dagstuhl Seminar on Distributed Systems*, pages 33–57, 1994.