Scalable Tool Infrastructure for the Cray XT Using Tree-Based Overlay Networks

Philip C. Roth, *Oak Ridge National Laboratory and*
Jeffrey S. Vetter, *Oak Ridge National Laboratory and*
Georgia Institute of Technology

ABSTRACT: *Performance, debugging, and administration tools are critical for the effective use of parallel computing platforms, but traditional tools have failed to overcome several problems that limit their scalability, such as communication between a large number of tool processes and the management and processing of the volume of data generated on a large number of compute nodes. A tree-based overlay network has proven effective for overcoming these challenges. In this paper, we present our experiences in bringing our MRNet tree-based overlay network infrastructure to the Cray XT platform, including a description of proof-of-concept tools that use MRNet on the Cray XT.*

KEYWORDS: XT, scalability, overlay networks, tools

1 Introduction

Performance, debugging, and system administration tools are critical for the effective use of parallel computing systems. Performance tools help users understand why their programs are not performing as they desire, and debuggers help users understand why their programs are not working at all. System administration tools help system administrators configure the system and monitor its health to make it available to users.

As system scale increases, these activities—performance diagnosis, program debugging, and system administration—become more critical for effective use of the system due to increases in the number of entities that must be configured, monitored, and controlled. Unfortunately, tools that work well in small-scale environments often fail to scale well as systems and applications get larger. Providing scalable tools that support these activities becomes more challenging for the same reason that the activities become more important: increases in system and application size result in more complex interactions between a larger number of application and system components. In the rest of the paper, we focus on performance and debugging tools for parallel applications. However, the material also applies to system administration tools with minor adaptations.

The functionality of a parallel performance or debugging tool can be divided into two categories: data collection and analysis, and process control. These activities are performed by one or more components of the parallel tool. Data collection is done by tool *back-end* components (often called tool daemons) that run on the nodes of a parallel system. The user interacts with the tool's user interface, implemented in a tool *front-end* component. Control of application processes (e.g., for single-stepping within a debugger) is done by tool back-end components attached to application processes. Data analysis may be performed by tool back-end components, the front-end component, or both. Figure 1a shows the organization of a typical parallel tool, with tool back end components running on the same nodes as application processes.

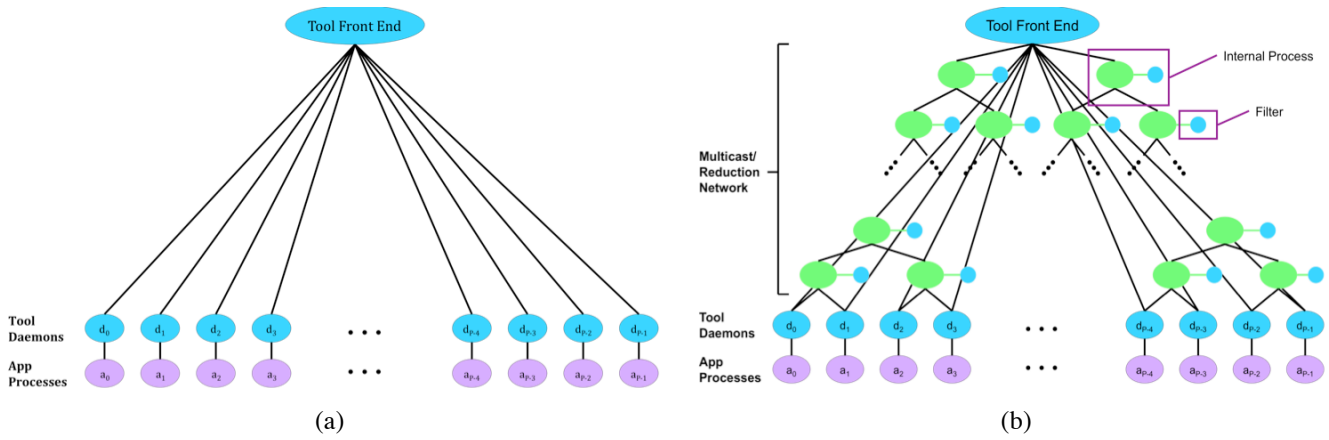

Figure 1: Parallel tool organizations, showing the typical parallel tool organization (a) and the organization of an MRNet-based parallel tool (b).

All tool communication and computation activity comes at a cost. When an activity's cost exceeds the available capability of the underlying parallel system, that activity limits the tool's scalability. Tree-based overlay networks can be used to reduce tool computation and communication costs by providing support for flexible, scalable multicast and data reduction activities. In this paper, we give a brief overview of tree-based overlay network concepts and a description of our tree-based overlay network implementation (Section 2). We then present our experiences in porting this implementation to the Cray XT platform (Section 3), followed by a brief description of a tool under development for the Cray XT that uses our implementation (Section 4). We conclude the paper by summarizing our experiences (Section 5).

2 Tree-Based Overlay Networks

A tree-based overlay network is a collection of tool processes, connected in a tree organization, interposed between a tool's front-end and its back-ends (see Figure 1b). The processes comprising the tree provide separate, independent threads of control that can compute and communicate in parallel. For instance, consider the situation when the tool front-end broadcasts a control message to all of its back-ends. Whereas communication between the tool front-end and its back-ends must be serialized when using a tool organization such as that depicted in Figure 1b, multiple processes within the tree-based overlay network can be broadcasting the message simultaneously to tool back-ends to which they are directly or indirectly attached.

MRNet [7] is our initial implementation of the Tree-Based Overlay Network concept. MRNet has been used to implement scalable performance tools [8], scalable debugging tools [2], and as a runtime for a programming model for data intensive applications [3].

MRNet-based tools transfer data between tool front-end and back-ends on logical data flows called streams. MRNet processes between the tool front-end and back-ends are called internal processes or internal nodes. MRNet internal processes use filters for synchronizing and manipulating data as it passes through the internal node. Using filters, MRNet can efficiently compute common data aggregations like averages and sums on data transferred across the overlay network. MRNet filters can also be used to implement complex, non-traditional data aggregations such as time-aligned performance data aggregation [7] for the Paradyn performance tool [5]. Paradyn's back-end processes produce timestamped performance data samples, but because the system node clocks are not synchronized on most modern parallel systems, samples taken from different application processes may represent different intervals of the program run. Given input sequences of unaligned performance data samples from several application processes, the time-aligned performance data aggregation filter aligns the performance data in time, and then aggregates the performance data to produce a single output sequence of performance data samples.

MRNet supports flexibility in MRNet network topology. When an MRNet overlay network is instantiated, a process topology specification is provided to the MRNet library in the tool's front-end process. The MRNet library creates and connects MRNet processes to match the provided topology.

3 Porting MRNet to the Cray XT

In this section, we describe our experiences in porting the MRNet tree-based overlay network implementation to the Cray XT platform. In this section, we denote this port MRNet/XT when describing characteristics specific to the Cray XT platform. All our MRNet/XT porting efforts

were done using Cray XT systems deployed in the National Center for Computational Sciences (NCCS) at Oak Ridge National Laboratory (ORNL).

When the NCCS Cray XT systems were first deployed at ORNL, the systems ran the lightweight Catamount operating system kernel on their compute nodes. In this time frame, we attempted to port MRNet to the Cray XT platform but were thwarted by mismatches between MRNet's implementation and the support provided by Catamount. For instance, because MRNet was originally implemented to support a wide variety of cluster-like parallel systems, MRNet processes communicate over TCP/IP sockets. Within the MRNet process tree, the MRNet front-end and internal node processes act as parent processes to a collection of child processes. When the MRNet process network is instantiated, each parent process creates a listening socket and accepts connections from its child processes. Catamount supported the client side of the TCP/IP socket API but not the server side, so MRNet processes running on XT compute nodes running Catamount could not create sockets listening for child connections. We made several abortive attempts to abstract the data transfer functionality from the rest of the MRNet implementation so that we could implement MRNet data transfer between compute nodes using the low-level Portals data transfer facility, but retaining the TCP/IP connection approach for traditional Linux clusters while adding a Portals data transfer facility proved challenging.

Eventually, the NCCS Cray XT systems were updated to run Linux on their compute nodes. This change enabled the MRNet implementation to use the same TCP/IP-based approach for data transfer on the XT and on more traditional compute clusters, because MRNet processes acting in a parent role could now create sockets listening for connections from child processes.

Support for server side TCP/IP sockets is necessary for MRNet process connection, but it is not sufficient. The MRNet process creation strategy in the initial MRNet implementation uses `rsh` (or `ssh`) to create processes on the nodes of a parallel system (Figure 2). This process creation occurs in waves by tree level. First (Figure 2a-b), the front-end at the root of the process tree uses `rsh` to create the processes at the first level of the MRNet tree. Once created, these processes connect back to the front-end using host and port information provided on the command line about the listening socket in the tool's front-end. When each child connects, the front-end delivers the topology specification to the child and the child then creates and connects its own collection of child processes (Figure 2c). This approach continues recursively until all network processes are created and connected (Figure 2d). Because each child can create its own children processes independently of each other, network startup is more scalable than if the front-end had to directly connect to each back-end process.

Although the Cray XT compute nodes now use a Linux kernel, they do not allow tools running on behalf of unprivileged users to ssh to compute nodes to create processes remotely. Instead, tools run by unprivileged users must use the system's parallel process launcher (`aprun`) to create application processes. This requirement produces an ordering problem for MRNet's traditional process creation/connection strategy: because all MRNet processes except the tool front-end are created at the same time, there is no opportunity for a process acting as an MRNet parent to create its listening socket, extract its port number, and provide the hostname and port number to the processes that will connect as its children in the MRNet tree as in the traditional MRNet process creation approach.

In addition to complications caused by differences in the available process creation mechanisms between the Cray XT environment and the traditional cluster environment, the Cray XT resource manager's restriction on co-locating processes presents another barrier to using the traditional MRNet process creation strategy on the Cray XT. To support on-line tools like parallel debuggers and parallel performance tools that attach and monitor application processes through third party interfaces (as opposed to being linked into the application's executables and monitoring the application's processes as a first-party entity), MRNet be able to place back-end processes on the same system nodes that are running application processes. On the Cray XT, however, after a parallel application has been launched on a collection of compute nodes the Cray XT resource manager does not allow a subsequent invocation of `aprun` to create processes on that same set of nodes.

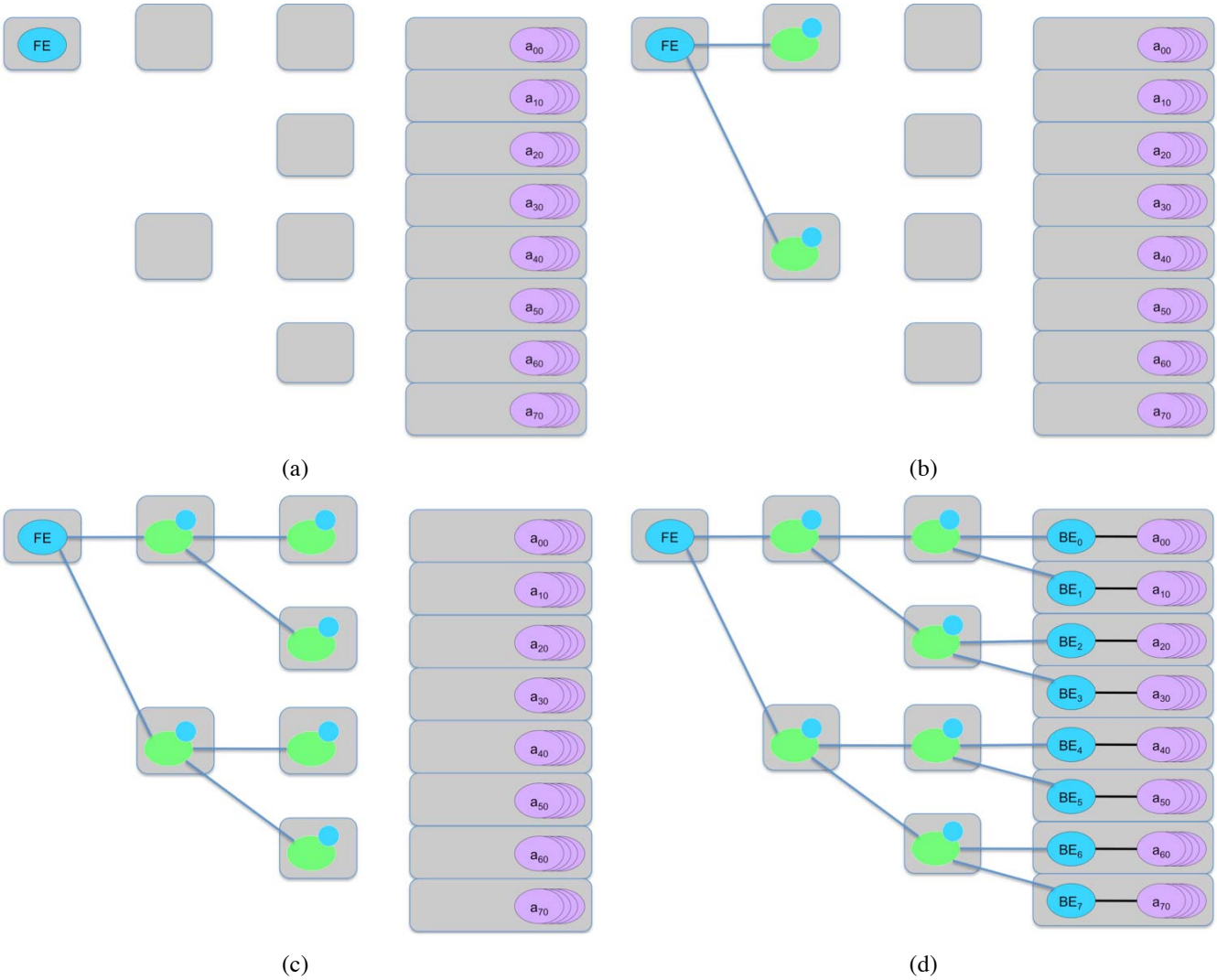

Figure 2: Traditional MRNet process tree instantiation.
A small-scale binary tree topology used for a debugger-like tool is shown as an example. Gray boxes indicate system node boundaries. First, the FE uses the system's parallel job launcher to start application processes (a). Next, the FE uses ssh to launch Level 1 MRNet processes which connect back to the FE and obtain the process tree topology specification (b). Next, each Level 1 process launches its Level 2 children, which connect back and obtain the process tree topology specification (c). Finally, the Level 2 processes launch the tool BE processes, which connect back to their parent processes but also attach to the application processes.

Another challenge facing the MRNet/XT port involves support for an MRNet use case requested by several parallel tool developers. In traditional compute cluster environments, MRNet developers have advocated he use of topologies that place MRNet internal processes on system nodes distinct from those running application processes (Figure 3a) to avoid perturbing application behavior with tool computation and communication activity. However, because leadership class systems like the ORNL Cray XT systems are scarce resources, access to the systems is managed using a highly competitive allocation process. When a user runs a job on such systems, their allocation is decreased based on the number of system nodes used and for how long. Some tool developers are loath to ask users to pad the number of nodes requested to provide "extra" nodes for running MRNet internal processes, and may not be concerned about the potential for perturbing application process behavior because of the type or timing of tool activities. For instance, a post-mortem debugging tool may not be concerned with perturbing application behavior because the tool's activity occurs after the application has finished running. Developers of such tools requested support for "flattening" the MRNet tree topology onto the same set of nodes that run the application processes (Figure 3b)

instead of requesting additional nodes for MRNet processes.

To address these challenges and new requirements, the MRNet/XT port uses a combination of conventions and Cray XT-specific tool support during its process tree creation and connection phase. MRNet/XT can use two mechanisms to create tool processes. For creating any MRNet processes that will not run on the same nodes as application processes, the MRNet library in the tool's front end uses `fork()` and `exec()` to invoke the Cray XT `aprun` command. The `aprun` process placement option (`-L`) is used to control the location of processes created with this mechanism. For creating any processes that are co-located with application processes, MRNet/XT uses a Cray-specific tool helper library. This library provides functions for determining the identity of nodes on which a particular application's processes are running, for spawning tool processes on those nodes, and for staging files to those nodes for use by the tool processes.

To address the problem of lack of information about listening sockets in parent MRNet processes, MRNet/XT separates the connection of MRNet processes into two phases: propagation of network topology and establishment of connections for data transfer. At the start of the MRNet network instantiation (Figure 4a), the only process with topology information is the tool front-end process. All other MRNet/XT processes create a listening socket for propagating topology information. By convention, these processes bind to a well-known port number. The tool front-end connects to each of its child processes at the first level of the MRNet process tree and pass along the network topology information for the subtree rooted at that child (Figure 4b-c). The front-end also delivers the port number of its listening data transfer socket (with system-assigned port number), and each child establishes a data connection to this socket. Although MRNet/XT could rely on well-known port numbers for its data connections as well as its topology propagation connections, the strategy we currently use allows us to share the same data connection logic for the traditional MRNet implementation as MRNet/XT. Once each child receives its part of the topology specification and connects back to its parent, the child propagates the topology to any of its own children within the desired MRNet process topology ((Figure 4d-e).

Using the convention of well-known port numbers during the MRNet/XT topology propagation phase presents a problem when more than one MRNet/XT process are to be placed on the same node, such as with a "flattened" tree topology (Figure 3b) where both internal processes and tool back-end nodes are placed on the same nodes with application processes. The problem is that not all MRNet processes on the same system node can bind to the same well-known port number. To address this problem, MRNet/XT slightly modifies its process creation strategy so that `aprun` or the Cray XT tool helper library creates exactly one MRNet process on each node (Figure 4b). This process accepts the topology specification from its MRNet parent process, determines which other MRNet processes should be created on the local node, and then uses `fork()` and `exec()` to create as many additional MRNet processes as needed for the desired topology (Figure 4c-f). To simplify the process creation logic, the first process on a node with co-located processes is always created to be an MRNet internal node process. If, after receiving the MRNet topology, the process determines that it should be a tool back-end process (i.e., if it is the only MRNet process on the node), it simply `exec()`s itself to become a back-end process.

As an example, consider a parallel debugger that uses MRNet/XT. Assuming the application and its arguments are specified to the debugger on its command line when the debugger front-end process is started, the debugger can use `fork()` and `exec()` to spawn the `aprun` process that launches the application processes with the desired command line arguments. The debugger front-end passes a special flag (`-P`) on the `aprun` command line that indicates the file descriptors of two pipes across which the debugger front-end process communicates with the `aprun` process to control the execution of the parallel application processes; the presence of this flag indicates to `aprun` that it should create the application processes but leave them blocked at a synchronization barrier, to be released after the debugger has completed its initialization. The debugger front-end then uses the tool helper function to determine the nodes on which application processes were placed, and constructs an MRNet topology specification using those nodes. The front-end instantiates the MRNet process network using an MRNet API function, including debugger back-end processes that attach to the paused application processes using the standard Linux ptrace or proc filesystem interfaces. Once the tool's back-ends are attached, they report their status to across MRNet to the tool's front-end and the tool front-end writes a command across its pipe connection to `aprun` causing it to release the application processes from their start up barrier.

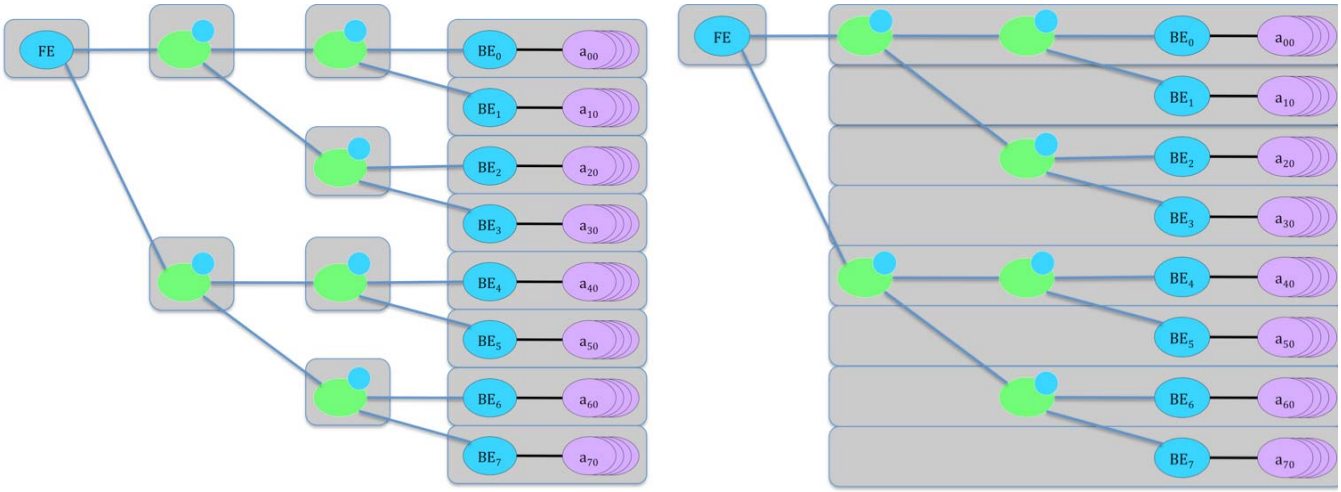

Figure 3: Process placement options for MRNet-based tools.
System node boundaries are indicated by gray boxes. The tool front-end process (labelled FE) runs on a login service node in both scenarios. Tool back-end processes (labelled BE_i) are co-located with application processes in both scenarios. In the "additional node" scenario (a), MRNet internal processes are placed on additional system nodes distinct from those running application processes. In the "flattened tree" scenario (b), one or more MRNet internal node processes may be placed along with a tool back-end process on each system node running application processes.

4 MRNet/XT Example Tool: mpiP

MpiP [1, 11] is a lightweight profiling library for programs that use the Message Passing Interface API [4, 9]. Using the PMPI interface provided by any MPI standard-conformant implementation, mpiP interposes instrumented versions of the communication and I/O functions of the MPI application programming interface. To control performance data volume, mpiP only gathers statistics about each MPI function call such as the maximum number of bytes transferred and the average operation latency. We recently augmented mpiP to collect communication topology data for point-to-point communication operations (e.g., see Figure 5). This capability helps a user understand the communication patterns and intensity used by an application, and thus may expose opportunities for optimizing communication performance with judicious mappings of application processes to system nodes to take advantage of the underlying system's interconnection network.

As an mpiP-instrumented program runs, all mpiP data collection activity is restricted to the local MPI process. MpiP only communicates between processes when it produces a report of observed program behavior. To produce this report, the mpiP library in each application process delivers the statistics it collected about the process' MPI activity to the mpiP library in the mpiP collector process (usually the rank 0 application process) using point-to-point MPI operations. The rank 0 process receives these statistics as they arrive and aggregates the statistics across all application processes for the generated report.

To reduce the cost of aggregating mpiP statistics and communication topology information, we are investigating the use of MRNet for off-loading the aggregation activity from the mpiP collector rank process into filters running in the MRNet overlay network. For communication statistics, each filter instance in an MRNet internal process aggregates the statistics collected from MPI processes reachable by that internal process via the MRNet process tree. By induction, the filter instance running in the MRNet front-end node produces aggregated statistics for the entire application.

Unlike communication operation statistics, communication topology data is not aggregated across application processes because its goal is to specify which pairs of processes are communicating and how much data they are transferring. However, MRNet can still benefit mpiP's collection of communication topology data to the mpiP collector rank by making individual communication operations more efficient. For situations like this, MRNet provides a data concatenation filter that concatenates smaller messages into a larger message as data is transferred across the MRNet process tree toward the tool front-end. On most systems, these larger, concatenated messages can be transferred more efficiently than the smaller messages containing communication topology data for only one MPI process.

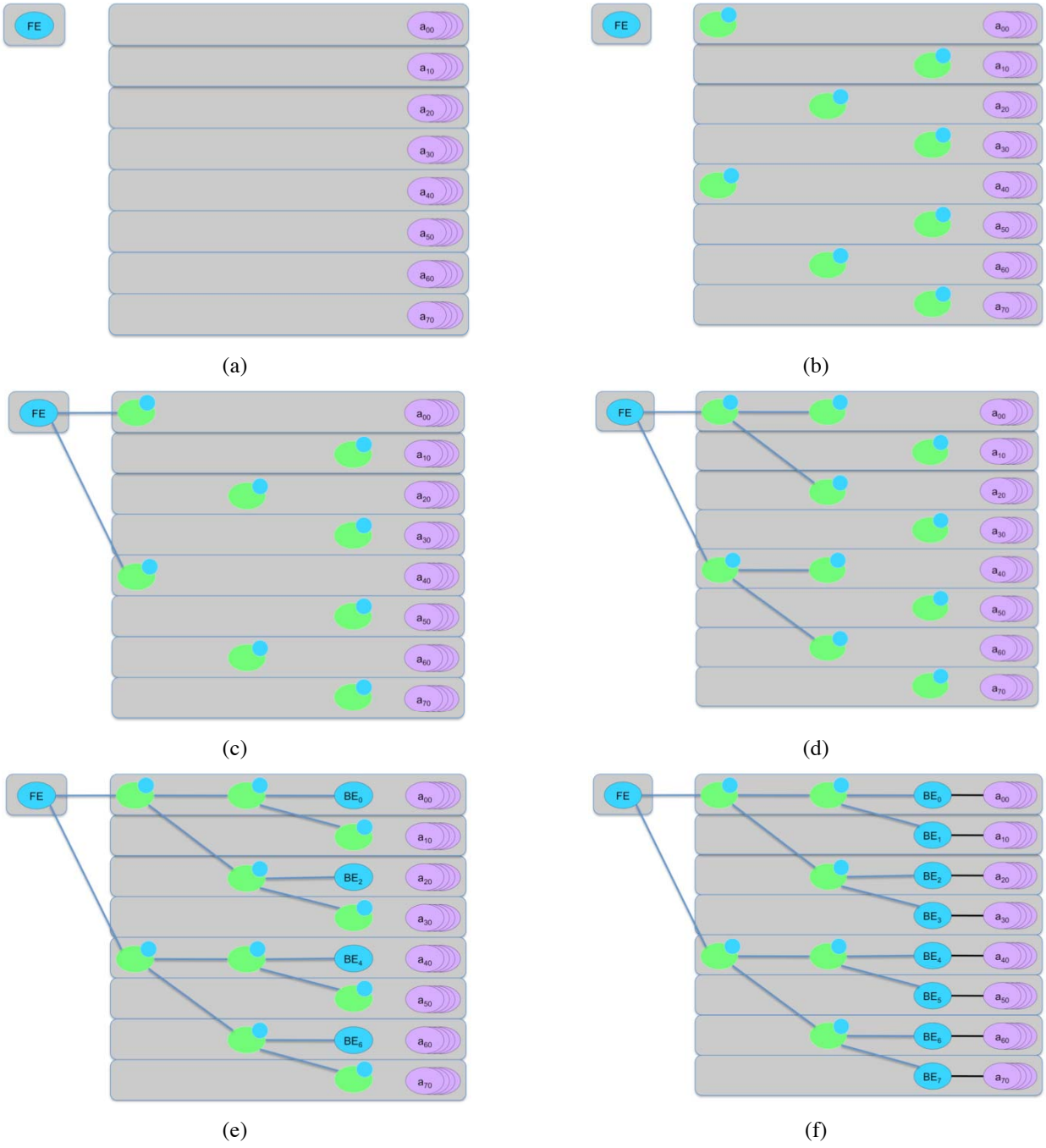

Figure 4: MRNet/XT process tree instantiation.
A small-scale binary tree topology used for a debugger-like tool is shown as an example. The flattened tree topology is used in the example. System node boundaries are indicated by gray boxes. First (a), the FE launches the application processes. Next (b), the FE creates the first MRNet process on each system node. Next (c), the FE connects to the Level 1 processes and provides them with the MRNet topology specification. Then (d), each Level 1 process spawns any co-located Level 2 processes, connects to its Level 2 child processes, and delivers the topology specification to its Level 2 children. Next (e), each Level 2 process creates any co-located Level 3 BE processes, connects to its Level 3 children, and delivers the topology specification to its Level 3 children. Finally (f), any Level 3 processes that are supposed to be BE processes become BE processes, and all BE processes attach to their co-located application processes.

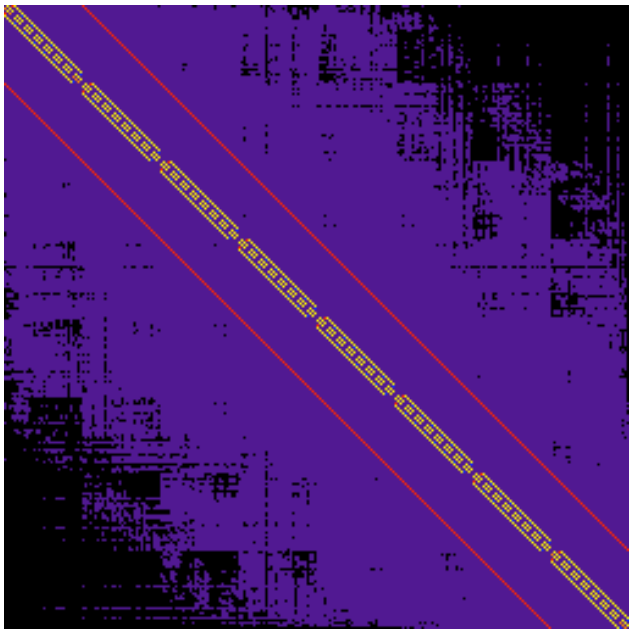

Figure 5: Visualization of point-to-point communication topology for AMG2000 collected with mpiP.
In the matrix, each item represents the amount of data transferred using point-to-point operations between a specific sending process (on the y-axis) and receiving process (on the x-axis). "Hot" colors like yellow and orange represent large communication volumes whereas "cool" colors represent smaller volumes. The observed diagonal pattern of hot-colored cells indicates a large point-to-point communication component, but the smearing of purple cells away from the diagonals indicates a substantial amount of communication also occurs with non-neighboring processes.

By design, mpiP supports profiling of MPI programs. However, some application developers are using programming models other than MPI like a Partitioned Global Address Space (PGAS) language such as Unified Parallel C [10] or Co-Array Fortran [6]. We are investigating how to apply the mpiP infrastructure mpiP to support programs using these alternative programming models. We call this more generic profiling infrastructure xP. Because xP may be used to profile applications that do not use MPI, the xP profiler cannot assume that an MPI implementation is available for aggregating performance data for program reports. MRNet provides a suitable alternative communication and data reduction infrastructure for use by xP that is not tied to any particular programming model used by the application.

5 Summary

Tree-based overlay networks have proven to be effective for overcoming several barriers to tool scalability by providing scalable support for multicast and data reduction operations. We have ported the MRNet tree-based overlay network implementation to the Cray XT platform. As part of the port, we added support placing MRNet internal network processes on the same compute nodes as tool back-end processes and application processes. We are now investigating the use of MRNet for implementing scalable performance and debugging tools on the Cray XT platform.

Acknowledgments

The authors would like to thank Mike Brim and Barton Miller of the University of Wisconsin, and Bob Moench of Cray, Inc. for assistance and advice in porting MRNet to the Cray XT platform. In particular, we thank Mike Brim for debugging assistance and for suggesting the approach for instantiating MRNet processes co-located with application processes.

This research is sponsored by the Office of Advanced Scientific Computing Research; U.S. Department of Energy. The work was performed at Oak Ridge National Laboratory, which is managed by UT-Battelle, LLC under Contract No. De-AC05-00OR22725. Accordingly, the U.S. Government retains a non-exclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes.

This research used resources of the National Center for Computational Sciences at Oak Ridge National Laboratory, which is supported by the Office of Science of the Department of Energy under Contract DE-AC05-00OR22725.

About the Authors

Philip C. Roth is a computer scientist in the Computer Science and Mathematics Division at Oak Ridge National Laboratory, where he is a founding member of the Future Technologies Group. His research interests include performance analysis, prediction, and tools with special emphases on scalability and automation; systems software, and storage for large-scale systems. He earned his Ph.D. from the University of Wisconsin—Madison in 2005. Roth can be reached at Oak Ridge National Laboratory, PO Box 2008 MS 6173, Oak Ridge, TN 37830-6173, USA, Email: rothpc@ornl.gov.

Jeffrey S. Vetter is a computer scientist in the Computer Science and Mathematics Division of Oak Ridge National Laboratory, where he leads the Future Technologies Group and directs the Experimental Computing Laboratory. Dr. Vetter is also Joint Professor in the College of Computing at the Georgia Institute of Technology, where he earlier earned his Ph.D. He joined

ORNL in 2003 after four years at Lawrence Livermore National Laboratory. Vetter's interests span several areas of high-end computing—encompassing architectures, system software, and tools for performance and correctness analysis of applications. He can be reached at Oak Ridge National Laboratory, One Bethel Valley Road, P.O. Box 2008 MS 6173, Oak Ridge, TN 37831-6173, USA, Email: vetter@computer.org.

References

[1] *mpiP: Lightweight, Scalable MPI Profiling*, http://mpip.sourceforge.net/.

[2] D.C. Arnold, D.H. Ahn, B.R. de Supinski *et al.*, "Stack Trace Analysis for Large-Scale Debugging," Proc. IEEE International Parallel and Distributed Processing Symposium, 2007.

[3] D.C. Arnold, G.D. Pack, and B.P. Miller, "Tree-Based Overlay Networks for Scalable Applications," in *11th International Workshop on High-Level Parallel Programming Models and Supportive Environments (HIPS 2006)*. Rhodes, Greece, 2006.

[4] W.D. Gropp, R. Thakur, and E. Lusk, *Using MPI-2: Advanced Features of the Message Passing Interface*, 2nd ed. Cambridge, Massaschusetts: MIT Press, 1999.

[5] B.P. Miller, M.D. Callaghan, J.M. Cargille *et al.*, "The Paradyn Parallel Performance Measurement Tools," *IEEE Computer*, 28(11):37-46, 1995.

[6] R.W. Numrich and J. Reid, "Co-array Fortran for parallel programming," *SIGPLAN Fortran Forum*, 17(2):1-31, 1998.

[7] P.C. Roth, D.C. Arnold, and B.P. Miller, "MRNet: A Software-Based Multicast/Reduction Network for Scalable Tools," Proc. 2003 ACM/IEEE Conference on Supercomputing, 2003.

[8] P.C. Roth and B.P. Miller, "On-line automated performance diagnosis on thousands of processes," Proc. Eleventh ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, 2006.

[9] M. Snir, W.D. Gropp, S. Otto *et al.*, Eds., *MPI—the complete reference*, 2nd ed. Cambridge, Mass.: MIT Press, 1998.

[10] UPC Consortium, "UPC Language Specifications, v1.2," Lawrence Berkeley National Laboratory LBNL-59208, 2005.

[11] J.S. Vetter and M.O. McCracken, "Statistical scalability analysis of communication operations in distributed applications," Proc. Eighth ACM SIGPLAN Symposium on Principles and Practices of Parallel Programming, 2001.