

RAPIDS 4.0: A Simulation/Emulation Tool for Dependability Analysis

Vijay Lakamraju, Israel Koren and C.M. Krishna

Department of Electrical and Computer Engineering

University of Massachusetts, Amherst MA 01003

E-mail: {vlakamra,koren,krishna}@ecs.umass.edu

Keywords: emulation, monitoring, MPI, fault injection, workload generator, real-time benchmarks

Abstract

Dependability evaluation must span the entire design process from specification through pre-deployment. A true estimate of system dependability can only be obtained if testing is carried out on the real hardware and software. Unfortunately, a complete prototype is not available until the fag end of the design cycle. RAPIDS 4.0 is a flexible software environment that helps to validate the performance and reliability of distributed real-time systems, as more software and hardware become available. Such a tool helps to continuously check if the system dependability specifications are being met along the design path.

RAPIDS 4.0 provides an integrated platform for the monitoring and control of applications running on an evolving hardware under a simulated environment. The user can analyze the impact of various system parameters within the scope of the available resources to modify/correct design decisions. The tool is complete with fault injectors, workload generators and GUIs containing detailed pictorial views of task executions and fault recovery in real-time.

INTRODUCTION

One of the most important steps in the development cycle of any system is to ensure that the system truly meets the system specifications that were initially laid out. Before the system is put into “action” in its working environment, it is important to test the system thoroughly for its conformance to those specifications. The testing of distributed systems and real-time systems is in many ways different from the testing of simple uniprocessor systems. In a distributed system,

one needs to have multiple loci of observation points. In a real-time system, where meeting deadlines is the most important design factor, one needs to worry about timing and concurrency. Add reliability and fault tolerance to that and we now have to pay special attention to fault injection, recovery and dependability assessment.

Analytical models may not be able to provide a true estimate of system performance and dependability as traditional techniques have become increasingly difficult to apply with the increasing complexity of computer systems. Simulation has its own drawbacks, not the least of which is that the accuracy depends on the granularity of simulation. Moreover, simulation techniques rely on the accuracy of several key parameters that may not always be easy to obtain. On the other extreme, testing the application while it is running on its target platform in the target environment is the most accurate way to measure system dependability. But this option is probably the least viable, as in most cases, one does not have access to the target environment or the target hardware until the very end of the design cycle. Moreover, discovering that the system does not meet the specifications at this stage can have severe implications. To avoid such a last minute “surprise,” a designer must continuously track the dependability of the system along its design path. The application needs to be run on the evolving system in a properly simulated environment and monitored to ensure that the system is in accordance with its dependability requirements. In this paper, we describe RAPIDS 4.0, a tool that provides a flexible software environment to validate the performance and reliability of distributed (real-time) systems by such a method.

RAPIDS 4.0 is the next in a series of tools developed as part of the RAPIDS tool suite, at the University of Massachusetts, Amherst. Its predecessor, RAPIDS 3.0 [12], is a pure simulation tool that uses an in-depth model for the simulated nodes and network

¹This work was supported in part by CASA, the NSF Engineering Research Center at the University of Massachusetts, Amherst, under award number 0313747.

and simulates the behavior of the system according to this model. The user specifies the topology, workload and information about faults and the system is simulated using events obtained by running the models. The tool has the flexibility of being event-driven as well as execution-driven. No actual applications are run except in the case when it is execution-driven. Even in this case, considerable changes must be made to the application to obtain timing values between important events of interest. These timing values provide the event times that help to keep the simulation going. RAPIDS 4.0 is different from RAPIDS 3.0 in that it is run on the system alongside real applications and very few changes, if at all, have to be made to the applications to provide for testing and evaluating the system and application. RAPIDS 4.0 also provides a framework to change some configuration parameters and algorithms (within the scope of the resources available) in an attempt to determine their optimal values. Such an integrated platform is the first of its kind among those that we are aware of. Most of the tools either concentrate on performance [5] or dependability [4, 16] or run-time monitoring [13].

In the following sections, we describe RAPIDS 4.0 in more detail, first starting with its design and then its input and output interface. We then comment on the intrusiveness of the tool. We conclude with a discussion on previous work.

THE DESIGN OF RAPIDS 4.0

Running a real-time application on a target platform and having a tool to provide a detailed pictorial view of the important events in the life-time of the application can go a long way in further understanding the interaction of hardware and software in the system. To be effective, system testing and validation must be conducted under properly simulated operational conditions, such as component and system failures and other exceptional conditions.

RAPIDS 4.0 provides an integrated platform for the launch and detailed monitoring of real applications running on the available hardware under a simulated environment in the presence of faults. Not only is the collected data useful in exposing performance and recovery bottlenecks, it could potentially be used to detect and correct design errors. The analysis can also help in better understanding the working of the application and making it more efficient. RAPIDS also provides a framework to test combinations of various system parameters and algorithms and this would help in implementing more aggressive designs. The challenge is to achieve all this functionality with minimal intrusion into the system and applications so that the

observations are not distorted.

The tool is set up as shown in Figure 1. Apart from the system under test which consists of the Application Computing Nodes (ACNs), the Application I/O devices (AIDs) and the network that connects all of them together, the Main Display/Control Node (MDN) is a central location for the collection of monitoring information from all the application nodes. It is typically a node that is separate from the ACNs that run the real application(s). The MDN also serves as the main controlling entity that accepts user information, carries out the task or discharges the task to a local controlling module that is present at each of the ACNs. It is the resource consumption and intrusion of this local entity that can cause a great deal of interference with the normal working of the application. Therefore, it is important to design this module with great care. We now describe each of the main components in some detail.

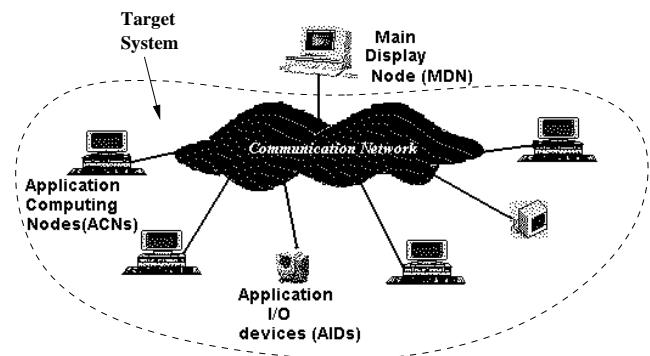


Figure 1. The RAPIDS setup

The Main Components

Figure 2 shows the process model and the various components involved in the design of RAPIDS 4.0. Applications running on a distributed system consist of a number of subtasks that typically communicate using some middleware services. As an example, we chose MPI [18], a message passing library designed for high performance on both massively parallel machines and workstation clusters. Detailed performance analysis requires that every major event happening in the system be reported to the Main Display Node. The amount of overhead and intrusion into the systems is directly related to the amount of logging that needs to be carried out. It is important to strike a good balance between these two aspects in order to provide sufficient monitoring information and an acceptable level of intrusion. As a first step, we chose to monitor various send and receive events at the application. Information about

these events not only provide a good indication of the amount of communication between the nodes and the application processes but also give a fairly good idea of the amount of time each application process spends in computation. One of the design goals of RAPIDS was to make it portable. While a number of other tools use low-level, sometimes OS-specific, features to capture send/receive events, we chose to monitor those events by modifying the middleware layer. This decision was made as a tradeoff between portability and intrusiveness. Important MPI calls are wrapped by system calls that send relevant information to the MDN. It is important that the system calls used be light-weight enough to not greatly impact the performance of the application. Our light-weight *MPI wrapper* attempts to minimize system overhead by using simple asynchronous MPI calls (e.g., `MPI_Isend()`) and also attempts to minimize network overhead by piggybacking monitoring messages as much as possible. Evaluating the overhead of the wrapper is an important issue and is discussed in a later section. Another design goal was to have applications run on the RAPIDS testbed with minimal changes. To meet this goal, the MPI wrapper was combined with the original MPI library to create a new RAPIDS-MPI library. The user now has to either compile his application with this new library or if, dynamic linking is allowed, simply ensure that the application uses this library during runtime.

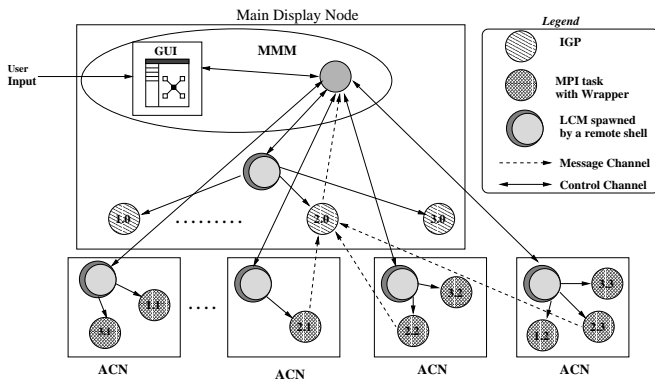


Figure 2. The RAPIDS process model

The Information Gathering Process (IGP) is responsible for collecting monitoring messages from all MPI subtasks corresponding to one application. Multiple applications can run on the target system and so there is one IGP per application. We have chosen to make the IGP a MPI subtask, rather than a normal process, to facilitate the use of MPI calls used in the MPI Wrapper. This also greatly enhances the portability of the systems. One can view all the monitoring messages

pertaining to an application as being sent over a *message channel* and different message channels exist for different applications. Such a view can help the IGPs pre-allocate buffers for receipt of monitoring messages and this can improve the performance of the monitoring system. The IGPs, after some initial processing, pass the messages to the Main Monitoring (and Control) Module (MMM) through some simple inter-process communication. The MMM is a collection center for all the monitoring messages. It also provides the graphical user interface (GUI) through which the user provides various input information and views a detailed pictorial view of the various events happening in real-time at each of the ACNs. Placing the IGPs and the MMM in the MDN is an attempt to further reduce the intrusiveness of the tool.

The MMM is also a control center in that it provides the user with a multitude of options to experiment with. It accepts input from the user through the GUI and performs the task either by itself or with the help of a Local Controlling Module (LCM) present in each node. After the user provides the required input through the GUI and starts the analysis, the MMM spawns a LCM on each node in the system and provides it with all the relevant information particular to controlling its host node. This includes information regarding the various subtasks that must be started on the node and information regarding the fault-prone environment under which the processes need to be executed.

The LCM is a small module that performs low-level control functions such as task spawning, fault injection and even task scheduling¹. Depending on how low one wants to get, the LCM would need to be tied closely with the operating system and portability could become an issue. The LCM spawns the tasks and synchronizes with the other LCMs and the MMM for the signal to start the execution of the applications. After this, the application subtasks send monitoring information through the message channel as usual whereas the LCM can send and receive information directly from the MMM through a bidirectional *control channel*. Information regarding fault injection and recovery is sent on this channel. Since reliability evaluation is an important aspect of the tool, we describe the fault injection interface in some detail in the next section.

Fault Injection

After taking care of spawning the application tasks, the LCM is mainly involved in fault injection and re-

¹Modifying the scheduling algorithm is possible in some real-time OS such as VxWorks and Lynx.

covery monitoring for the rest of the time. In theory, the LCM could use any of the various fault injectors that are available, such as Xception [2] Ferrari [8] and FTAPE [19]. It is also possible to combine these fault injectors to provide multiple fault models, multiple fault triggers and support for multiple targets [16].

As the first step, we have integrated a Software Implemented Fault Injector (SWIFI) which is based on the *ptrace()* system call [8, 21]. This SWIFI uses the process tracing functionality provided by *ptrace()* to stop a process and access its address space. At this point, the tracing process can corrupt any location in the traced process' address space to emulate a fault. This capability can be used to inject faults into the registers of the host processor. We enhanced this rudimentary *ptrace()* SWIFI to provide it the functionality of assessing the sensitivity of specific parts of the process' address space to faults. Static information typically present in the executable² can be used to determine the location of various entities (such as functions, initialized and uninitialized global variables) and this can be used to carry out more focused fault injection experiments. Fault injection can also be carried out on dynamic regions of the process address space (i.e., the heap and the stack) using breakpoints. The fault injection need not necessarily be confined to the host system. Faults can also be injected into I/O devices, as long as sufficient interface functions are provided by the device drivers. One of our projects that has made use of this facility is described in [11].

THE USER INTERFACE

Apart from monitoring an application running on the target platform, RAPIDS also provides the user with the capability of analyzing the impact of various system parameters on performance as well as dependability in an attempt to determine their optimal values. RAPIDS aims to make nearly all such changeable parameters that a system or an operating system provides, user-settable through proper GUIs. The user input in RAPIDS comprises the following three categories:

Configuration parameters: In addition to launching the application on the target platform, the user can override the configuration of the target platform to test the performance of the application under different configurations within the scope of the resources available. For example, in a bus-based system, the user can restrict the application to run on a subset of the nodes attached to the bus. In a large distributed point-to-

point connected system, the user can choose a specific sub-network in which the application must be run. The user can also override the default task management software, i.e., RAPIDS provides the capability of assigning the various tasks of the application to specific nodes in the target platform based on the user's directive or a specific algorithm. Such a capability can help in finding an appropriate system configuration for the application under consideration.

Task parameters: Integrating applications into RAPIDS is a simple task in that the application just needs to run using the RAPIDS-MPI library. The GUI allows the user to specify various options before starting the applications such as the number of subtasks and the command line arguments. The user-specified task allocation algorithm provides the MMM with information to decide which subtasks of which applications are started on which nodes. The tool also provides the user with the option of specifying and generating *synthetic* tasks. Synthetic tasks can be used to simulate the ambient (or background) workload of the system or even the tasks of the application, in cases when the whole application is not yet available. The user is given full control in the description of the synthetic tasks so as to mimic an actual workload. The user can provide task information in terms of a task trace that was generated earlier or through a detailed user interface. If available, the user is also provided the option of overriding the scheduling algorithm at each node in the system. Traditionally, the dependability of a real-time system has been equated to the effectiveness of the fault tolerant mechanisms embedded in the system. Recently, the ability of the system to handle load surges [10] has been pointed out as a better measure for real-time system dependability. RAPIDS provides the user with the capability of simulating load surges and obtaining values of the surge handling capabilities.

Fault Parameters: RAPIDS has a detailed user interface for specifying faults. The user can specify the location, the time and the type of faults that need to be simulated. The user can specify faults on a per-process basis and indicate which address in the process' address space needs to be affected. If symbol information is available in the executable, fault location can also be specified using function names and variable names from individual applications. Both temporary random single bit flips as well as permanent stuck-at-1 and stuck-at-0 faults can be specified. The times at which the fault(s) should be injected can be specified either as a random process or as a fixed rate. Apart from low-level faults such as those affecting the CPU and memory, simulation of various network-related faults such as message corruption, message reordering and message delaying

²The Binary File Descriptor (BFD) library [3] can be used to obtain this information.

are also provided. Each fault specified by the user occupies an entry in a fault table and the MMM sends relevant information to each process at the start of the analysis. By logging important events during the fault recovery process, values of important parameters such as rollback overhead, hardware and software reconfiguration penalty can be obtained.

RAPIDS provides a number of detailed graphical interfaces for viewing the information collected by the MMM. The information is displayed almost in real-time through various windows to provide the user with a detailed pictorial view of important events during the execution of the applications as they are running. Among the many windows that are provided, three windows deserve specific mention. The Task Schedule Window displays information regarding the running of tasks on each node. Events such as the start of an instance of a task, the completion, the sending/broadcasting of messages, the receiving of messages, preemptions, checkpointing etc. are aptly displayed on this window as shown in Figure 3. It is a “sliding” window that displays the required information in real-time. This window also shows information regarding fault injection and the time of recovery. Figure 3 shows the injection of a fault into the task running on node 3 and the recovery obtained through an Application-Level Fault Tolerance (ALFT)-based technique [6]. The Task Allocation Window shown as an inset in Figure 3 provides information about how the tasks are currently allocated and can also be used to see how the allocation changes during reconfiguration. The Performance Windows show various statistics about the working of the system and the applications such as idle times, number of messages sent/received on a per-application basis, amount of communication on a per-node basis etc. All the events are also logged so that they can be played back after the run is complete.

The earlier RAPIDS 3.0 tool took as input system-related parameters such as preemption cost, checkpoint overhead and reconfiguration penalty from the user and so the accuracy of the simulation depends on the user’s experience or knowledge of the target platform. By running the application on the target platform or its evolving version and monitoring important events as can be done in RAPIDS 4.0, more precise values of these parameters can be obtained. Visualizing the data obtained from monitoring a “live” system can also assist in identifying performance and recovery bottlenecks. The designer might then decide to try out a different configuration or modify some parameters/algorithms/mechanisms to ensure the system meets the requirements of the application.

INTRUSIVENESS AND PORTABILITY

Reducing the impact of the monitoring processes is one of the most important design consideration. A number of techniques have been used to factor this aspect into the design. The messages sent out by the Wrapper are piggybacked to circumvent the overhead of sending separate messages. Such a scheme can sometimes impact the real-time display of information on the graphical interfaces at the MDN, but this inconvenience could be ignored in most cases, when weighed in with the decrease in intrusiveness. Message passing between the ACNs and the MDN can cause high network traffic that may affect the average message latency of the application. In order to circumvent this problem, a technique utilizing postponed message delivery is used. Event information can be sent to the MMM during the quiescent state of the network, i.e., when the network traffic is low and the impact of the messages generated by the monitoring tool on the transfer time of the actual application messages will be minimal. It is also possible to use multicasting at the network layer to further reduce the overhead of the extra messages, but this can impact the portability of the system. Table shows the overhead associated with the monitoring and fault injection on the run-time of three space applications. RTHT[20] is an benchmark from the C3I Benchmark Suite, whereas OTIS and NGST have been taken from the REE Application Suite[14]. The numbers reported in the table have been obtained by running the applications on a Myrinet network.

Application	Perc. Increase in Network Traffic	Perc. Increase in Execution Time
RTHT	5.8%	1.2%
OTIS	3.7%	0.8%
NGST	8.2%	1.5%

Table 1. Monitoring overhead of different applications

As with many such tools, portability is another important feature. Our current design assumes that the distributed application use MPI for their communication. Using a MPI wrapper, rather than another method for sending the monitoring messages, allows easy portability to other machines while allowing us to reap the benefits of developments within the MPI group as more platforms are supported. Undoubtedly, if the applications were written using some other middleware layer, we need to extend RAPIDS to involve that middleware. We are concurrently working on supporting FT-MPI, with an intent to increase the usability and applicability of RAPIDS.

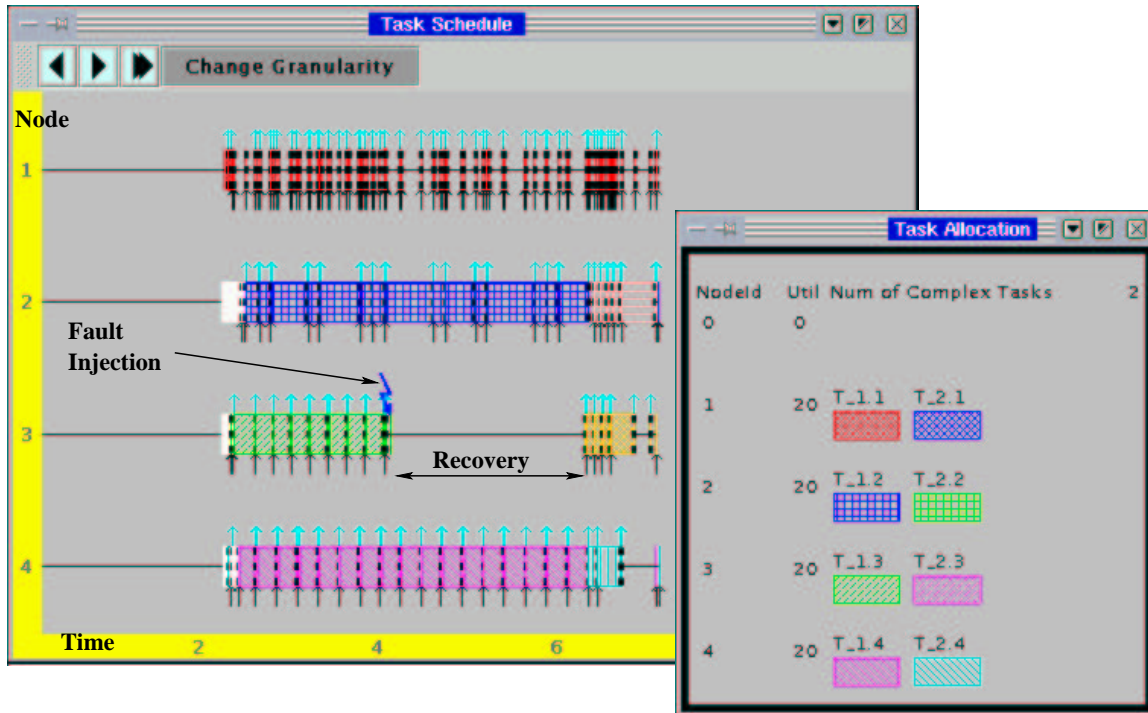


Figure 3. The Task Schedule window showing the running of OTIS, a soft real-time space application. Also shown is the recovery from a fault in node 3.

PREVIOUS WORK

A number of tools for reliability analysis have been reported in the recent past. Most of them concentrate on fault injection and measuring fault tolerance [7, 9, 2, 4, 16, 8] and a few of them are used to validate performance [1, 15, 22, 13, 5]. RAPIDS 4.0 differs from them in that it provides a combined analysis of both performance as well as dependability. Many of the commercially available tools also focus on a specific component of the system such as the CPU, bus, memory etc., while others utilize specific features in the operating system, and hence, are difficult to port to other systems. For example, Xception [2], makes use of the advanced debugging and performance monitoring features of modern, complex processors such as the PowerPC 604. RAPIDS 4.0 attempts to utilize the basic functionality available on most systems and mitigate some of the above deficiencies. RAPIDS can be envisaged as a platform for experimenting with various “what-if” scenarios by allowing to change system parameters dynamically, which is not a common feature available in most tools.

RAPIDS is available by request from the ARTS Lab website [17]. Its ongoing development is expected

to provide new releases in the near future.

SUMMARY AND FUTURE WORK

RAPIDS 4.0 is a flexible, monitoring tool for distributed systems that can be used to validate the performance deliverables of the application while also giving the user the capability of correcting for inefficiencies by changing different parameters within the scope of the resources available on the target platform. Further, it can be used to study the impact of modifications in the hardware and software configuration on the performance of the system.

It is important that dependability evaluation span the entire design process from specification through pre-deployment, and they must address validation at different levels of abstraction. The integration of RAPIDS 3.0 (the simulator) and RAPIDS 4.0 (the emulator) will provide a tool that can be used to perform analysis at various stages in the entire design process. During the initial stages of design when the designer has to work with just a vague idea of the system to be build, she/he can use the simulator to try out various options and get an idea of which configuration and set of algorithms would be appropriate. As some hardware starts becoming available, the designers can get

some key parameter values using the emulator to refine their experiments on the simulator. The simulator can also be used to get a first-level idea of how well applications scale to larger systems. After possibly many such cycles, the emulator can be used to do a complete performance and reliability analysis on the prototype before it is deployed. The combination of the emulator and simulator is being dubbed RAPIDS 5.0.

References

- [1] L. Shi et al. Optimization in a hierarchical distributed performance monitoring system. In *Proc. IEEE International Conf. Algorithms and Architectures for Parallel Processing*, 1995.
- [2] J. Carreira, H. Madeira, and J.G.Silva. Xception: A Technique for the Experimental evaluation of Dependability in Modern Computers. *IEEE Trans. Software Engineering*, Feb 1998.
- [3] S. Chamberlain. *LIB BFD, the Binary File Descriptor library*. Free Software Foundation, 675 Mass Ave, Cambridge, MA, 1991.
- [4] S. Dawson, F. Jahanian, and T. Mitton. ORCHES-TRA: A fault injection environment for distributed systems. Technical Report CSE-TR-318-96, Computer Science and Engineering, University of Michigan, Nov. 1996.
- [5] O. Endriss, M. Steinbrunn, and M. Zitterbart. NETMON-II: a monitoring tool for distributed and multiprocessor systems. *Performance Evaluation North Holland*, 12, 3:191–202, 1991.
- [6] J. Haines, V. Lakamraju, I. Koren, and C. Krishna. Application-level fault tolerance as a complement to system-level fault tolerance. *Journal of Supercomputing*, (16):53–68, May 2000.
- [7] J. Arlat et al. Fault Injection for Dependability Validation: A Methodology and some applications. *IEEE Trans. Software Engineering*, Feb 1990.
- [8] G. A. Kanawati, N. A. Kanawati, and J. A. Abraham. FERRARI: A tool for the validation of system dependability properties. In D. K. Pradhan, editor, *Proceedings of the 22nd Annual International Symposium on Fault-Tolerant Computing (FTCS '92)*, pages 336–344, Boston, MA, July 1992.
- [9] W. Kao and R. Iyer. DEFINE: A distributed fault injection and monitoring environment. In *Proceedings of the IEEE Workshop on Fault-Tolerant Parallel and Distributed Systems*, June 1994.
- [10] Z. Koren, I. Koren, and C. Krishna. Surge handling as a measure of real-time system dependability. *Proc. First Merged Symposium IPPS/SPDP, EHPC Workshop*, pages 1106–1116, April 1998.
- [11] V. Lakamraju, I. Koren, and C. Krishna. Low Overhead Fault Tolerant Networking in Myrinet. In *Proc. International Conference on Dependable Systems and Networks*, 2003.
- [12] M. Allalouf and V. Lakamraju et al. RAPIDS: A simulator testbed for Distributed Real-time Systems. In *Proc. Advanced Simulation and Technology Conference*, 1998.
- [13] R. Rajkumar, F. Jahanian, and S. Raju. Run-time monitoring of timing constraints in distributed real-time systems. *Journal of Real-Time Systems*, pages 163–175, Oct. 1994.
- [14] Remote Exploration and Experimentation (REE) Project. <http://www-ree.jpl.nasa.gov/>.
- [15] S. Lei et al. A software instrumentation technique for performance tuning of message passing architectures. In *Proc. Winter Simulation Conference*, 1997.
- [16] D. Stott, B. Floering, D. Burke, Z. Kalbarczyk, and R. Iyer. NFTAPE: A framework for assessing dependability in distributed systems with lightweight fault injectors. In *Proceedings of the IEEE International Computer Performance and Dependability Symposium*, pages 91–100, Mar. 2000.
- [17] The Architecture and Real-time Systems Lab website. <http://arts.ecs.umass.edu/>.
- [18] The Message Passing Interface. <http://www-unix.mcs.anl.gov/mpi/>.
- [19] T. K. Tsai and R. K. Iyer. Measuring fault tolerance with the FTAPE fault injection tool. *Lecture Notes in Computer Science*, 977:26–??, 1995.
- [20] B. V. Voorst, R. Jha, L. Pires, and M. Muhammad. Implementation and results of hypothesis testing from the C3I parallel benchmark suite. In *Proceedings of the 11th International Parallel Processing Symposium (IPPS'97)*, Apr. 1997.
- [21] V. Sieh. Fault injector using UNIX ptrace interface. *Internal Report IMMDD3, Universitat Erlangen-Numberg*, Nov. 1993.
- [22] T. Yen and W. Wolf. Performance estimation for real-time distributed embedded systems. In *Proc. International Conf. Computer Design*, 1995.