RAPIDS 4.0: A versatile performability analysis tool for distributed real-time systems

1 Objectives

An important step in the development cycle of any system is validation. Before a system is put into "action" in its working environment, it is necessary to test the system thoroughly for its conformance to the system specifications. A motivating factor for developing RAPIDS 4.0 came from the failure to find a suitable tool that can be used to analyze both the performance and the reliability of flight prototypes to be built by the REE Project. RAPIDS 4.0 is intended to provide a complete testing framework for REE.

2 Approach

The performance of the target system is analyzed through detailed monitoring of the applications as they are running on the system in a properly controlled environment. The reliability of the system is analyzed through fault injection and detailed monitoring of the fault recovery process. Apart from validation, the tool also allows the user to explore various "what-if" scenarios within the scope of the resources available. The user can test the impact of modifying various configurations and system parameters until the required performance is obtained. The challenge is to achieve all this with minimal intrusion into the target system and the applications.

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, a separate Main Display Node (MDN) serves as the central location for the collection and display of monitoring information from all the ACNs. It is also the main controlling entity that accepts user information and either carries out the task or discharges the task to a local controlling module that is present at each of the ACNs. We now briefly describe the main software components in RAPIDS 4.0.

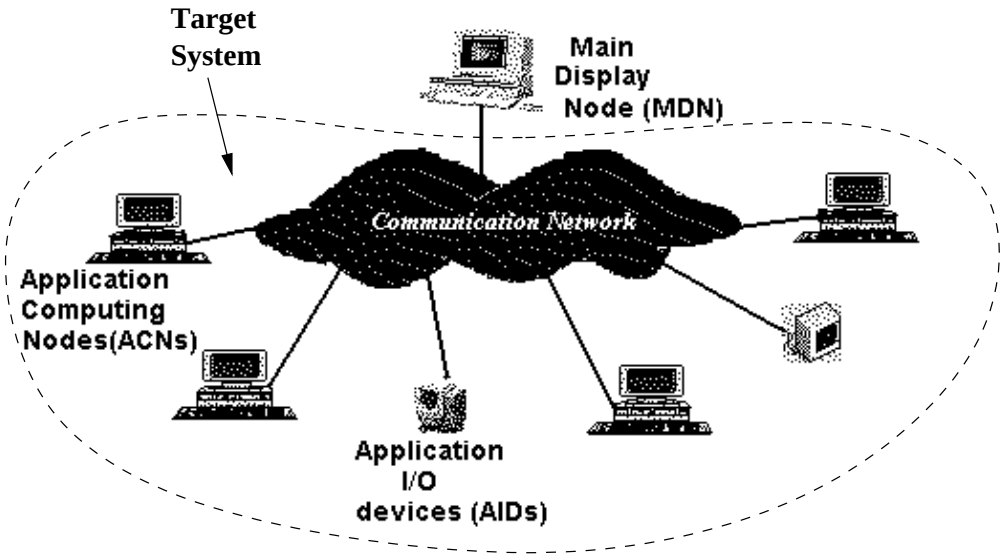

Figure 1: The RAPIDS setup

2.1 Main Components of RAPIDS 4.0

Figure 2 shows the process model and the various components involved in the design of RAPIDS 4.0. We assume that the applications running on the system consists of a number of subtasks that communicate with each other using MPI. 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 provides a good indication of the amount of communication between the nodes and the application processes but also gives a fairly good idea of the amount of time each application process spends in computation. Important MPI calls are wrapped by system calls that send relevant information to the MDN. 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. 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.

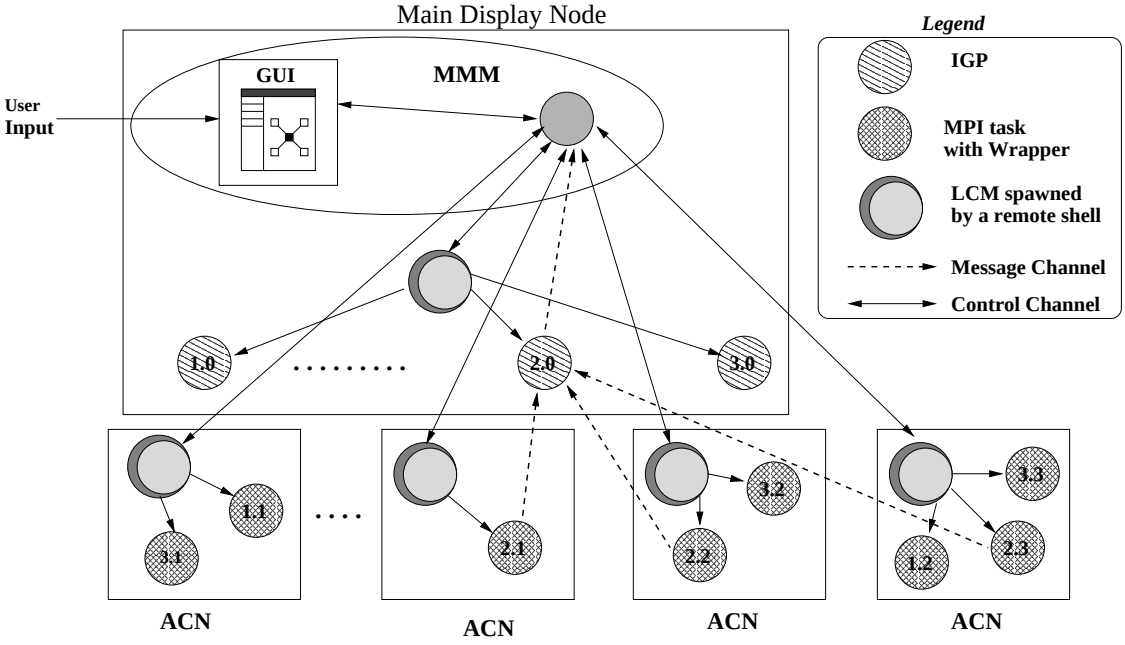

Figure 2: The RAPIDS process model

The Information Gathering Process (IGP) is responsible for collecting monitoring messages sent via the MPI Wrapper from all the subtasks corresponding to one application. Since multiple applications can run on the target system, there is one IGP per application. We have chosen to make the IGP a MPI subtask to enhance the portability of the tool. 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 receiving monitoring messages and this can help 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 supplies various input information and views a detailed pictorial view of the various events happening in real-time at each of the ACNs. The IGPs and the MMM are located in the MDN, another step towards reducing 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. After the user provides the required input through the GUI and starts the analysis, the MMM spawns a Local Controlling Module (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 scheduling (whenever possible). The LCM spawns the tasks and synchronizes with the other LCMs and the MMM to start the execution of the applications. The LCM sends and receives information directly from the MMM through a bidirectional *control channel*. The other most important piece of information that is passed on this channel is information regarding fault injection and recovery.

In theory, the LCM could use any of the fault injectors that are available. As a first step, we integrated JIFI, a software implemented fault injector developed in-house at JPL. We modified JIFI to address some drawbacks related to the overhead of multiple parent processes for injecting faults into multiple child processes and mitigated the requirement to change application source code. We also enhanced JIFI 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 executable was used to determine the location of various entities (such as functions, initialized and uninitialized global variables) and was provided to the user to carry out more focussed fault injection experiments.

2.2 Input and Output of RAPIDS

Apart from simply monitoring an application running on the target platform, RAPIDS also provides the user with the capability of analyzing the impact of certain system parameters on performance and determining their optimal values. RAPIDS aims to make nearly all such changeable parameters that a system or an operating system provides, user-settable. The user input in RAPIDS basically falls into 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, 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.

- **Task parameters:** Integrating applications into RAPIDS is a simple task in that the application just needs to be recompiled with 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. Future enhancements include providing the user with the option of specifying *synthetic tasks*. This is intended for cases when the actual application is not yet available and to simulate ambient (or background) workload of the system. Applications will be fabricated using the specifications and run on the target system.

- **Fault Parameters:** RAPIDS has a detailed user interface for specifying faults. The user can not only specify where the fault should strike, but also how and when. As discussed earlier, the user can specify faults on a per-process basis and indicate which address in the process' address space needs to be affected. Fault location can also be specified using register names, function names and variable names for individual applications. Our fault model consists of random single and multiple bit flips. When to inject the faults can be specified by the user either through a random process specification or a fixed rate. 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.

All the data collected by the MMM is displayed almost in real-time through various windows to provide the user with a detailed pictorial view of the important events during the execution of the applications as they are running. The Task Schedule Window is a very important window in that it displays information regarding the start of tasks, completion of tasks, sending/broadcasting of messages, receiving messages, preemption, checkpointing etc. This window also shows information regarding fault injection and the time of recovery. The Task Schedule Window shows 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 systems 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.

3 Accomplishments

To support the development of RAPIDS, a cluster of workstations was set up. Each node in the system is a Pentium III running RedHat Linux 7.1. They are interconnected through two networks, Myrinet and 100MHz Ethernet. The Myrinet driver installed on the workstations is GM version 1.4. The applications use GM-MPI version 1.2..5. Currently there are five machines in the cluster. Below are some of our achievements/findings:

- RAPIDS 4.0 was first developed to provide a complete performance monitoring tool and then evolved to provide for fault injection and recovery analysis.
- JIFI was ported to the Linux operating system and several enhancements made, as discussed before.
- A clean interface to integrate applications to RAPIDS was defined. Users need simply to change every mention of "mpi.h" in their source files to "rmpi.h" and recompile them with the RAPIDS-MPI library. Three applications have been seemlessly integrated into the tool: OTIS, NGST and RTHT (a real-time benchmark obtained from Honeywell). We perceive that integrating the remaining REE applications into RAPIDS would be just as easy.
- Several output windows of RAPIDS now provide more in-depth and detailed view of the information. In most instances, one can clearly surmise the nature of the application or the type of communication pattern from the schedule window, e.g., OTIS is a compute-intensive application whereas NGST is a communication-intensive application.
- Several configuration parameters have been exercised, e.g., it was found that limiting the number of subtasks from three to four in the case of NGST does not increase the execution time significantly. Such a result can be used towards power savings.

- Dependability analysis of the applications was also carried out using the features provided in the fault table. The advantages of using a fault tolerance scheme such as ALFTD was demonstrated on the OTIS application. A screen shot of the schedule window showing ALFTD in action is shown in Figure 3.

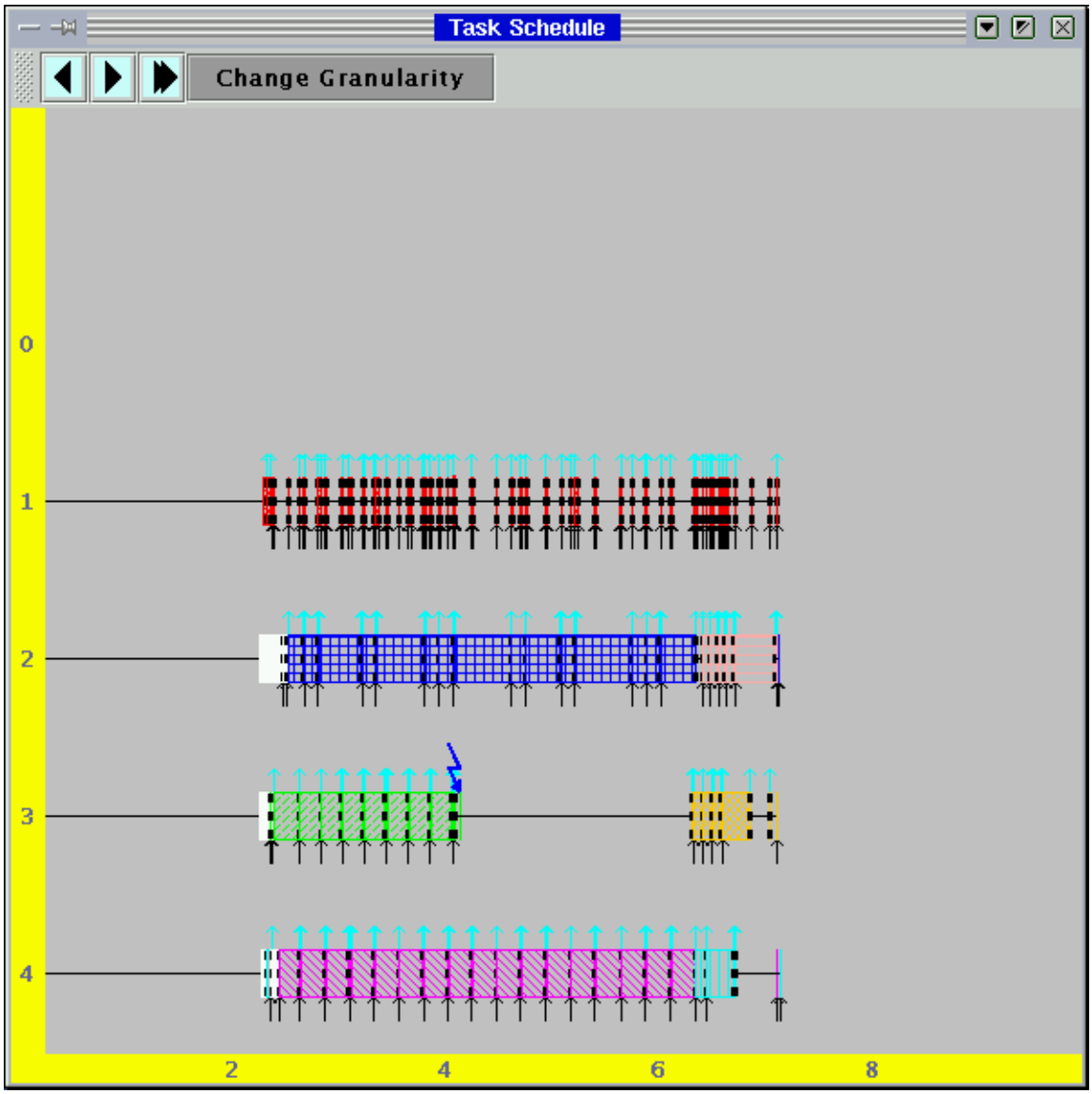

Figure 3: The schedule window of RAPIDS showing the fault injection and recovery in node 3

4 Significance

- RAPIDS provides an integrated platform for the launch and detailed monitoring of real applications on real hardware under a simulated faulty environment. Not only will the collected data be useful in exposing performance and recovery bottlenecks, it may even be used to detect and correct design errors.

- Running the real application on the target platform and having a detailed pictorial view of the important events in the life-time of the application can help us further understand the interaction between the hardware and software. 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.
- Prospective users of the REE spaceborne computer need to be assured of its performance, dependability and availability. RAPIDS 4.0 addresses this need.

5 Future Enhancements

Various components in RAPIDS can be enhanced. The fault injector can be extended to provide for fault injection in the heap and stack without having to make changes in the applications. Dependability analysis of the network can also be carried out by augmenting the fault injector with a network component. This would be useful in analyzing the fault tolerant behavior of the network subsystem. Synthetic tasks can be integrated to provide ambient workload and the GUI can be enhanced to provide multiple levels of monitoring. Performability analysis also needs to be carried out in the presence of SIFT layers such as Chameleon.

The integration of RAPIDS 4.0 emulator and RAPIDS 3.0 simulator will provide an integrated tool that can be used to perform analysis at various stages in the entire design process from specification to pre-deployment. During the initial stages of design when the designer has to work with just a vague idea of the system to be build, 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 designer can use the emulator to obtain real values of key parameters which would help him to refine experiments on the simulator. The designer could also use the simulator to see how well applications scale to larger systems. After a couple of iterations, the emulator can be used to perform a complete performance and reliability analysis on the prototype. It can also be used to study the impact of changes in various configuration and system parameters and obtain their optimum values before the system is deployed. Clearly, such a simulator-emulator combination is of great importance.