The SCIRun Problem Solving Environment: Implementation within a Distributed Environment

Michelle Miller, Charles D. Hansen, Christopher R. Johnson
Department of Computer Science
3190 MEB
University of Utah
Salt Lake City, UT 84112
{mmiller, hansen, crj}@cs.utah.edu

Extended Abstract

Introduction Building systems that alter program behavior during execution based on user-specified criteria (computational steering systems) has been a recent research topic, particularly among the high-performance computing community. To enable a computational steering system with powerful visualization capabilities such as SCIRun to run in a distributed computational environment, a distributed infrastructure (or runtime system) is required. This infrastructure permits one to harness a variety of machines to collaborate on an interactive simulation. Building such an infrastructure requires devising strategies for coordinating execution across machines (concurrency control mechanisms), mechanisms for fast data transfer between machines, and mechanisms for user manipulation of remote execution.

Interactive scientific visualization and computational steering require low-latency and high bandwidth computation in the form of model generation, solvers, and visualization. Latency is particularly a problem when analyzing large datasets, constructing and rendering three-dimensional models/meshes, and allowing a scientist to alter the parameters of the computation interactively (thus "steering" the computation). However, large-scale computational models often exceed the system resources (memory and storage) of a single machine, motivating closer investigation of meeting these same needs with a distributed computational environment comprised of many machines.

SCIRun, a scientific problem solving environment (PSE), provides the ability to interactively guide or steer a running computation. Initially designed for a shared memory multiprocessor, SCIRun is a tightly integrated, multi-threaded framework for composing scientific applications from existing or new components. High-performance computing is needed to maintain interactivity for scientists and engineers running simulations. Extending such a performance-sensitive application toolkit to enable pieces of the computation to run on different machine architectures all within the same computation proves useful for some scientists. Not only can many different machines execute this framework, but also several machines can be configured to work synergistically on computations (*e.g.,* farming off compute-intensive pieces to the "big iron").

Distributing SCIRun Large-scale scientific computations require multiprocessors of some form due to the vast amount of data inherent in these applications/simulations. However, the computer architecture of multiprocessors varies greatly, differing in the number of processors, communication mechanism, operating system, and memory model, etc. Using abstraction to hide the underlying details of the memory model (*i.e.,* shared memory versus distributed memory) from the application allows us to take advantage of the range of machine types, from shared memory multiprocessors to massively parallel processors. However, such an abstraction creates extra layers in the software infrastructure that generally result in increased software overhead. One must strike a balance of exposing enough details to efficiently tune the application without losing portability.

To achieve our goal of computational steering of distributed simulations, we first created a mechanism for using SCIRun across multiple machines, then we can solve the problem of running a scientific code on a distributed memory architecture. This discussion focuses on the infrastructure mechanisms needed to run one SCIRun computation across multiple machines while maintaining control of the computation from the user's local workstation. The goal of SCIRun has been to maintain interactive rates for users constructing

and manipulating models and simulations. To be successful with a distributed version of SCIRun, we must not impair interactive speeds by incurring excessive network latency.

To meet the demand for growing computational problem sizes, we have extended the existing communications infrastructure within SCIRun to include cross-machine communications and execution. In this way, we can preserve existing system functionality for shared-memory executions, as well as providing more architectural flexibility. In constructing a distributed system, our goal is to provide SCIRun users with the illusion of running on a single computer, while they are actually utilizing several computers, which could be large distances away, each with potentially very different characteristics. For example, we could utilize various Grid resources in geographically separate areas to take advantage of highly specialized resources.

Software Architecture for Distributed SCIRun The model for running SCIRun on multiple machines calls for the scientist to initiate the computation (*i.e.,* start SCIRun) on a graphics-capable machine, to compose an application from existing SCIRun components (*modules*), to specify which of these should run remotely, and then to start execution of the computation (*dataflow network*). Interacting with modules, viewing the rendered visualization, and interacting with three-dimensional data probes (*widgets*) within the visualization all occur on the initiating machine, which we denote as the Master. The compute-intensive dataflow subnetwork executes on the remote machine, which could be a multiprocessor machine.

We abstract away the differences between running on one or more than one machine by modifying the scheduling, inter-module communication, user interface mechanisms, and other "internal" portions of SCIRun that an application writer would typically never see. Making modifications to the infrastructure allows programs that have been designed for the SCIRun problem solving environment to function in either a multi-machine, networked environment or a single-machine, shared memory environment.

Distributed Control Infrastructure The SCIRun scheduler controls the execution order of modules and tracks and signals modules that need to run again. SCIRun is based on a hybrid form of a dataflow network where data passes from upstream modules to downstream ones. To ensure concurrency control, we use one scheduler for all machines working collaboratively on a SCIRun computation. A master-slave configuration yields a single point of control from which module scheduling decisions can be made for the entire network consisting of both local and remote modules. For each remote module, the Master SCIRun instance, running on the user's workstation, instantiates proxy modules to keep the internal module state necessary for scheduling and control. Thus, the single scheduler model is efficient and effective.

Remote Data Transfer Since we must maintain the dataflow metaphor within one SCIRun application program (or dataflow network), we provide a mechanism for data transport from modules running on one machine to downstream modules running on different machines. We accomplish this remote dataflow by opening a communication channel, in this case a socket, for every data pipe linking two modules running across machine boundaries. The data are then bundled into messages and sent across the channel from the module's output port. These data channels are instantiated and torn-down when the user makes, or destroys, a connection between modules which cross machine boundaries. Connected modules running on the same machine, even though remote, use the current SCIRun shared memory data communication.

Remote User Interaction and Steering SCIRun succeeds in allowing a range of intuitive user interactions with parameters of the model, choice of simulation techniques and their parameters, and the resulting visualization. When we move to a distributed environment, careful attention must be paid to maintaining close coupling between user action and system response. Further, providing visual feedback to the user about the runtime performance traits helps the user choose appropriate modules to execute remotely.

We have built a remote communication mechanism into our Tcl wrapper calls used by the GUI. These are based on a combined *push* and *pull* strategy. Modules always check the most current user-selected parameters before using them, and remote modules do the same. Similarly, a progress bar, which indicates the percentage of the computation completed, is updated on the proxy module residing on the Master by the corresponding Slave module. Finally, if a user changes a parameter in one of the steering windows, the module is rescheduled in both the single-machine and distributed versions of SCIRun.