Changes in Computational Load

The user zooms in on a particular area of the model via the steering interface, requesting that the area be shown in more detail. This change increases computational load in the Mesh component and results in a violation of the constraint established within Mesh. A mesh computation takes 30 ms per image, but only 20 ms are allotted. The Mesh component is notified of this violation and, in response, invokes a user-specified adaptation strategy. This strategy might apply the following methods in sequence, until the computation is returned to its specified behavior:

1. *Improve resource quality:* Seek to improve the quality of the compute resources allocated to Mesh, for example, by obtaining additional processors, increasing working set size, or increasing time slice or priority. This process may involve negotiating with a local resource broker and perhaps restructuring the Mesh computation to use additional resources.
2. *Delegate improvement:* Ask other pipeline components whether they can reduce their time requirements by applying an appropriate adaptation strategy. For example, the FE modeling component might be able to acquire additional processors, or one of the network links might be able to ask for more bandwidth. This process involves cooperation and coordination with other system components.
3. *Relocate computation:* Attempt to find other, more powerful compute resources on which to execute Mesh. This process involves locating compute resources with sufficient networking bandwidth and connectivity, scheduling those resources, migrating the Mesh computation, and notifying other nodes in the pipeline of the new location.
4. *Reduce resolution:* If user performance requirements do not specify absolute constraints on mesh resolution, apply domain-specific techniques to reduce the total number of mesh points while maintaining increased resolution in the specified area of interest.
5. *Inform user:* Notify the user of performance violation.

Changes in Networking Bandwidth

Assume an environment in which bandwidth reservations are expensive. Hence, the application, observing low network traffic on link B, initially chooses to rely on best-effort service. Later, the network load increases, causing the communication time to increase from less than 10 ms to 100 ms, violating the constraint on communication times established initially. The FE modeling program is notified of this violation and, in response, invokes an adaptation strategy that applies the following methods:

1. *Control error:* Upon encountering an increase in error or loss rates, temporarily invoke forward error correction [61, 67].
2. *Improve resource quality:* Seek to improve the quality of the network resources allocated to link B by making a bandwidth reservation on the current network.
3. *Delegate improvement:* Ask other pipeline components whether they can reduce their time requirements by applying an appropriate adaptation strategy. For example, the FE modeling component might be able to acquire additional processors, or one of the network links might be able to ask for more bandwidth.
4. *Relocate communication:* Attempt to find other, higher-bandwidth network resources connecting the two end points.
5. *Apply compression:* If sufficient computational resources exist at the end points, apply compression: either lossless or lossy, if an appropriate lossy compression module has been provided.
6. *Inform user:* Notify the user of performance violation.

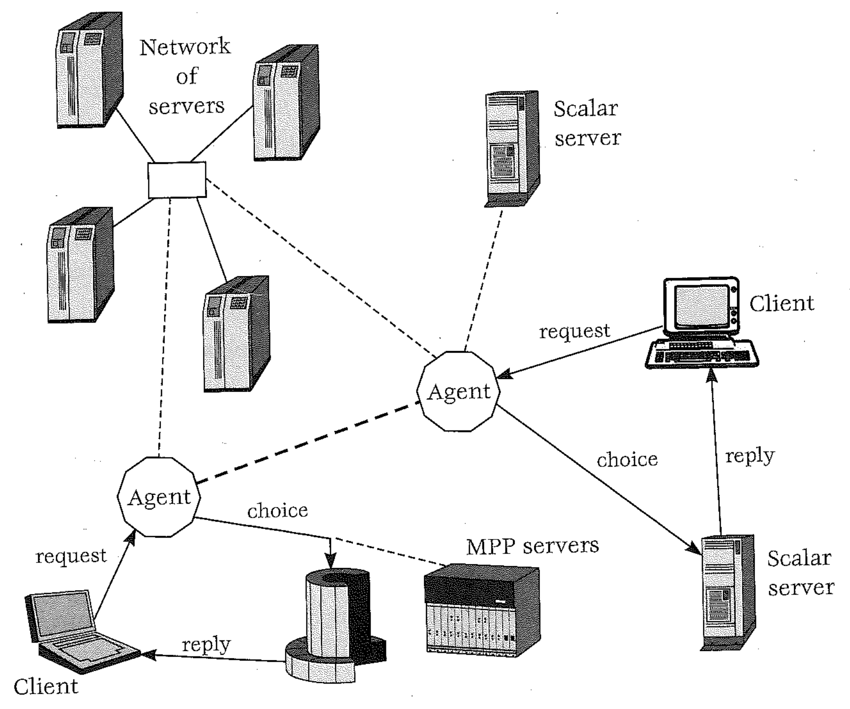
7.3 CASE STUDY: NETSOLVE

We now present case studies of two application-specific tools. In this section we focus on NetSolve, a network-enabled solver; in the next section, we discuss SCIRun, a computational steering system. The purpose of these case studies is to illustrate how the issues raised in the preceding sections are addressed in a system context.

NetSolve is a system designed to allow the use of network resources for numerical computing [104] by providing convenient access to problem-solving techniques (software) in addition to managing computational resources.

Figure 7.4 shows the structure of NetSolve, which consists of the following:

- A set of *client APIs* that allows functions defined in various numerical libraries to be called as remote procedure calls from programs written in MATLAB, C, Fortran, or Java



7.4 FIGURE Organization of the NetSolve system.

- A set of *servers* that encapsulates remotely accessed numerical libraries
- One or more *agents* that match requests for services with servers

Requests to the NetSolve system are submitted to a NetSolve agent as the result of a call to the client API. The NetSolve agent processes these requests, selecting the most suitable server for each request. Finally, servers perform the requested computation. As shown in the figure, the computational servers can run on single workstations, networks of workstations that can collaborate for solving a problem, or parallel systems.

For the duration of this section, we will use the term *resource* (or *computational resource*) to refer to a NetSolve server running on a particular computer (the hardware resource) that provides access to some numerical software (the software resource). We will now discuss, in turn, the NetSolve language, agents, servers, and support for fault tolerance.

7.3.1 NetSolve Language

The NetSolve application-specific language is defined by the client API, which is designed to work with the languages normally used to solve numerical problems, either a high-level mathematics package, such as MATLAB, or a low-level sequential programming language, such as C, Fortran, or Java. In each case, the NetSolve system is invoked by using a special, network-enabled version of the call usually used to invoke the desired numerical routine. For example, the MATLAB code to perform a matrix multiply is

```
c = a * b
```

while the equivalent NetSolve code is

```
c = netsolve('matmul',a,b)
```

As a result of the `netsolve` call, a request is dispatched to a NetSolve agent, which locates an appropriate resource to perform the specified computation. The Fortran, C, and Java APIs are similar.

The MATLAB API for NetSolve also provides an asynchronous (i.e., non-blocking) version of the NetSolve function:

```
request = netsolve_nb('matmul',a,b)
   :
   :
c = netsolve('wait',request)
```

like split-C?

which allows a program to continue computing while a remote call is being computed, and/or to dispatch several remote calls concurrently.

The user interacts with the NetSolve system at a relatively low level, explicitly specifying when NetSolve resources are to be used. Nevertheless, once the `netsolve` call has been made, grid resources are transparently managed by the NetSolve system.

7.3.2 The NetSolve Agent

Fundamental to the design of NetSolve is the idea that a domain-specific scheduling agent can provide robust and high-performance application execution in a grid environment. The NetSolve scheduling agent performs two main functions:

1 ✦ It maintains an index of the available computational resources and their characteristics.

2 ✦ It accepts requests for computational services from the client API and dispatches them to the best-suited server.

A NetSolve system can contain several instances of the NetSolve agent. It is possible to start a NetSolve agent on each "local" network, so that user requests always go to the "closest" agent for processing. Different instances of the NetSolve agent can then have different views of the set of computational resources, reflecting the fact that certain clients are closer to certain computational resources.

Keeping track of what software resources are available and on which hardware resources they are located is perhaps the most fundamental role of the NetSolve agent. Computational resources register with the NetSolve agent when they start up, indicating their capabilities. The agent then monitors the status of the resource, noting if it becomes unavailable for any reason.

For each incoming request the NetSolve agent is responsible for choosing the best-suited computational server. The current NetSolve scheduling policy is designed to optimize resource utilization rather than the performance of any individual application. Scheduling decisions are based on information about the computation as well as static and dynamic information maintained on the available resources. The request submitted to the agent contains computation-specific information such as the size of the input data and of the problem to be solved. Resource-specific information is collected by the NetSolve agent via an assortment of protocols and heuristics. The current NetSolve agent was developed in the absence of any grid services or infrastructure. Clearly, the development of the NetSolve agent would have been greatly simplified if standardized services had been available for resource location, resource allocation, resource monitoring, remote process creation, and communication (e.g., see Chapter 11).

Several other researchers are investigating agent-based approaches to resource scheduling. For example, the Application Level Scheduler (AppLeS) system (Chapter 12) focuses on the development of scheduling agents for parallel applications. The Network-Based Information Library for high-performance computing (Ninf) [366] also uses an agent-based design. The agent in Ninf is called a *meta-server* and is responsible for the request assignment to the different servers. A common interface between NetSolve and Ninf is currently being prototyped, with the goal of achieving a seamless integration that allows either the NetSolve or Ninf interfaces to be used to perform computations on servers administered by either system.

the more complicated
the larger systems

7.3.3 NetSolve Servers

Within NetSolve, computational resources are represented by NetSolve servers. NetSolve servers were designed with three goals in mind:

SSI

✦ *Uniform access to software:* Users should feel like they are using a single scientific package that can perform any computation.

✦ *Configurability:* The servers should not be limited to any particular software package and should extend to new numerical applications at will.

✦ *Preinstallation:* The software present on servers should be ready to use, already compiled for the target architecture, and either already installed or not dependent on the software-hardware binding policy.

The implementation of NetSolve servers that meet the above goals has been facilitated by the use of a general, machine-independent *description language* to describe each separate numerical function to be provided by a computational server. Descriptions written in this language can be compiled by NetSolve tools into actual computational modules. Description files have been written for numerous numerical libraries, including FitPack, ItPack, MINPACK, FFTPACK, LAPACK, BLAS, QMR, and ScaLAPACK. These numerical libraries cover several fields of computational science, including linear algebra, optimization, and fast Fourier transforms.

In addition to providing machine independence, the description language promotes increased collaboration between research teams and institutions, since description files can be exchanged and downloaded at will to start new NetSolve servers. Each time a new description file is created, the capabilities of the entire NetSolve system are thereby increased.

7.3.4 Fault Tolerance in NetSolve

elastic

The NetSolve system was designed to be resilient to failures of both agents and servers, ensuring that a user request will be completed unless every single resource has failed. Agents and servers are designed so that they can be arbitrarily started and stopped without jeopardizing the integrity of the system. Each agent is responsible for keeping track of what servers are currently operational. A fault detection service provided by the Globus toolkit (Chapter 11) is used for this purpose. Multiple agents can coexist within the NetSolve system, providing resilience to agent failure as well.

The basic fault tolerance mechanism used is *retry*. The system maintains sufficient state that if a computational server malfunction (server unreachable, server stopped, failure during computation, etc.) is detected, it can resubmit the request to an alternative server. Furthermore, as mentioned in Section 7.2.3, NetSolve can reduce the possibility of host failure by using a Condor pool (see Chapter 13) or a highly reliable end system (see Chapter 17).

7.4 CASE STUDY: SCIRun

As a second system-level case study of an application-specific toolkit, we consider the SCIRun environment for computational science. This environment provides a tightly integrated framework for performance-critical applications. SCIRun has been used for research within the domains of bioelectric field studies in cardiology and neuroscience, as well as for computational geophysics and other computational field problems. SCIRun does not currently use remote computing resources, but is being modified to do so, as discussed in Section 7.4.3.

SCIRun is a computational steering problem-solving environment [91, 551, 438] that allows the interactive construction, debugging, and steering of large-scale scientific computations. The goal is to enable the scientist "to spot visual anomalies while computing and immediately steer, or modify the calculations, to test out new theories. Interactivity puts the scientist back in the computing loop; the scientist gets to drive the scientific discovery process" [152]. SCIRun enables scientists to modify geometric models, to interactively change numerical parameters and boundary conditions, and to modify the level of mesh adaptation needed for an accurate numerical solution. As changes in parameters become more instantaneous, the cause-effect relationships within the simulation become more evident, allowing the scientist to develop more intuition about the effect of problem parameters, to detect program bugs, to ask what-if questions, to develop insight into the operation of an algorithm, or to develop a deeper understanding of the physics of the problem(s) being studied.

When the user changes a parameter in any of the module user interfaces, the module is reexecuted, and all changes are automatically propagated to all downstream modules. The user is freed from worrying about details of data dependencies and data file formats. The user can make changes without stopping the computation, thus "steering" the computational process. As opposed to the typical "offline" simulation mode—in which the scientist manually sets

like "make"

Opinion: This should be done by a simulation system designated to a more closed parallel machine ... Swarm, Speedos ...

input parameters, computes results, visualizes the results via a separate visualization package, then starts again at the beginning—SCIRun closes the loop and allows interactive steering of the design, computation, and visualization phases of a simulation. SCIRun has been used to solve problems in computational medicine [291, 292] and geophysics and will soon be used in computational fluid dynamics and combustion simulations.

7.4.1 SCIRun Visual Programming Language

SCIRun provides a visual programming language for application development. The motivation for using a visual programming language is ease of use for scientists and engineers; it extends the familiar model of visualizing datasets and manipulating aspects of the computation as it proceeds, through the use of three-dimensional interactive manipulation probes (*widgets*).

The SCIRun system permits an application writer to compose a data flow network, or program, out of existing parts called *modules*. Within each module are the underlying application-specific algorithms written in Fortran, C, and/or C++. These modules are selected from a palette, placed on a canvas, and linked together by using a visual programming language. Selected modules appear on the network editor canvas as boxes. Then, an application writer connects the modules together with *data pipes*. Data flows from module to module via data pipes that connect an output port (through which data descends) to an input port of one or more downstream modules.

7.4.2 Aspects of the SCIRun Environment

An example of the SCIRun system interface is shown in Plate 12. The center of the picture displays a graphical representation of the data flow network. The boxes represent computational algorithms (modules), with the lines representing data connections between the modules. Each module may have a separate user interface, such as the matrix solver interface at the left, that allows the user to control various parameters. The window in the upper right-hand corner is an interactive three-dimensional viewer that combines visualization output with three-dimensional widgets for exploring the data and model. Displaying three-dimensional widgets for manipulating data and computation within rendered geometries more closely couples the changes in problem parameters to modeling, simulation, and the resulting visualization.

SCIRun provides general class libraries for use by scientists or programmers who are building applications or domain-specific components to be used