Metasystems

How they create the illusion of a giant desktop computational environment—transparent, distributed, shared, secure, fault-tolerant.

Andrew Grimshaw, Adam Ferrari, Greg Lindahl, and Katherine Holcomb

Metasystems give users the illusion that the files, databases, computers, and external devices they can reach over a network constitute one giant transparent computational environment. With a metasystem, users can share files, computational objects, databases, and instruments. They need not decide where to execute their programs nor copy the necessary binaries and data files; the metasystem does these jobs. And new classes of applications—meta-applications—are available through the emerging infrastructure, further increasing users' efficiency and productivity.

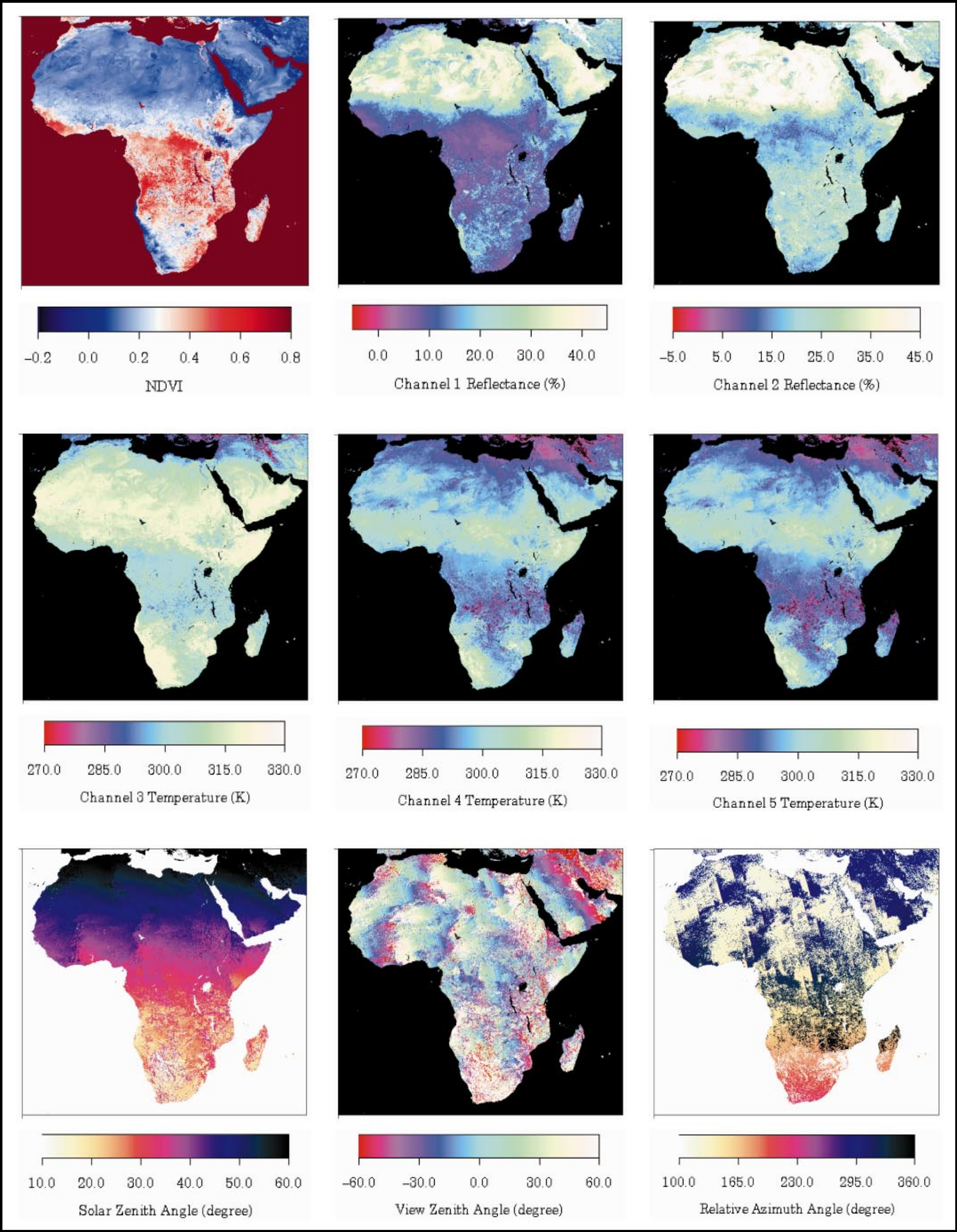
Here we explore the potential of metasystems and the technical problems that have to be solved to realize them, highlighting five metasystem uses: shared persistent object spaces; transparent remote execution; wide-area queueing systems; wide-area parallel processing; and meta-applications. We also cover an example of a meta-application being developed by NPACI—a coupled ocean-atmosphere-weather model.

A metasystem's building blocks are geographically separated resources (people, hosts, instruments, databases) connected by one or more high-speed interconnections. An essential software layer, often called middleware, transforms a collection of independent hosts into a single, coherent virtual machine, giving the user the power of a unified environment. To the user, this virtual machine is a single entity that executes applications, schedules application components, detects and recovers from faults, provides protection and security to users and resource owners, and brings users together through enhanced support for collaboration and sharing.

Why don't we use metasystems? The fundamental difficulty is lack of software—specifically, an inadequate conceptual model for metasystem software design. Faced with accelerating changes in hardware and networking, the computing community has sought to stretch the existing paradigm for sharing resources over a network—interacting autonomous hosts—to a level that exceeds its abilities. The result is a collection of partial solutions, some good in isolation but lacking coherence and scalability, making development of even a single wide-area application at best demanding and at worst nearly impossible.

The challenge to the computer science community is to provide a solid, integrated middleware foundation on which to build wide-area applications. NPACI, in its role as agent for high-end computing, has an additional challenge—to design and deploy metasystems technology providing the high performance needed for the most demanding scientific applications.

As envisioned, the NPACI metasystem contains thousands of hosts and petabytes of data. Users will have the illusion of a very powerful computer on



Land cover of Africa in Plate Caree projection.
(Courtesy, Joseph Já Já and John Townshend, University of Maryland.)

their desks that can manipulate objects representing data resources, such as digital libraries or video streams; applications, such as teleconferencing or physical simulations; and physical devices, such as cameras, telescopes, and linear accelerators. The objects being manipulated may be shared with other users, allowing construction of shared virtual workspaces.

This metasystem will support the illusion of a single machine by transparently scheduling application components on processors; managing data migration, caching, transfer, and coercion, or the masking of data-format differences between systems; detecting and managing faults; and ensuring that users' data and physical resources are protected. Moreover, the technology used by the metasystem will have to scale appropriately.

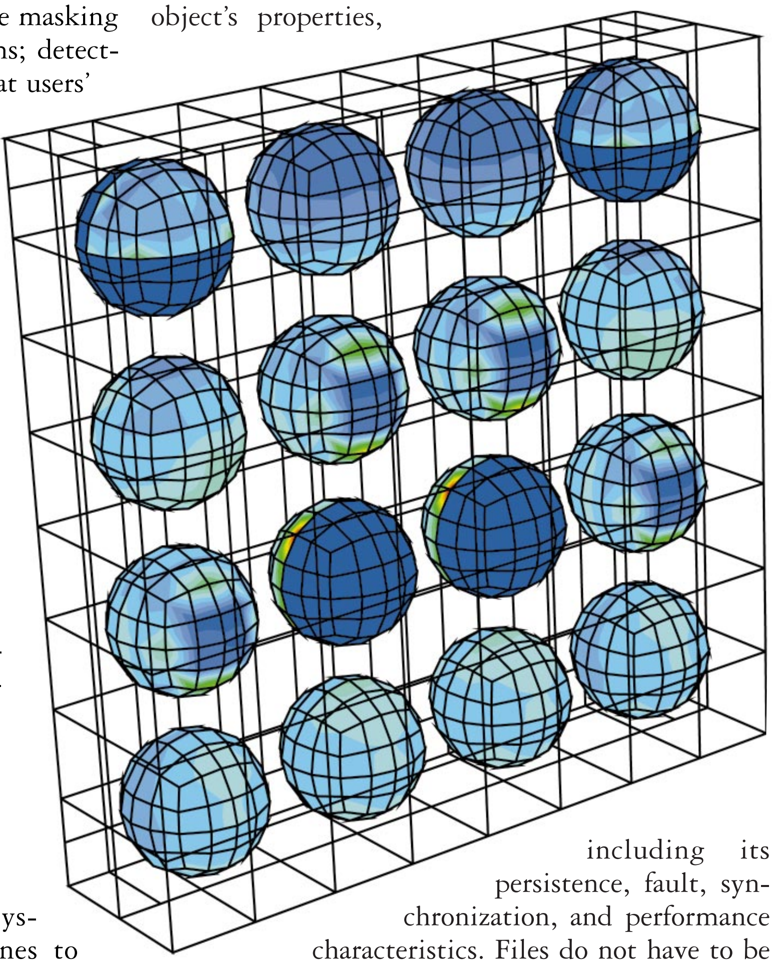
The potential benefits of such a metasystem to the scientific community are enormous, including:

- More effective collaboration, achieved by putting coworkers in the same virtual workplace
- Higher application performance, owing to parallel execution and exploitation of off-site resources
- Improved access to data and computational resources
- Improved researcher and user productivity, resulting from more effective collaboration and better application performance
- A considerably simpler programming environment for applications programmers

There are many ways to use a metasystem, ranging from relatively simple ones to intricate implementations exploiting its abilities to solve currently impossible problems. We sketch five wide-ranging uses to illustrate some of the possibilities.

Shared persistent object (file) space. The simplest service a metasystem provides is location transparency, whereby users gain authorized access to entities without knowing where they reside. For file access, this well-understood type of service is often called a "distributed file system." Well-known distributed file systems include NFS [9] and Andrew [10].

In shared object spaces, instead of just sharing files, all entities (files as well as executing tasks) can be named and shared among authorized users. This merging of "files" and "objects" is driven by the observation that files are merely a special kind of object that happens to live on a disk, so files are slower to access but persist when the computer is turned off. In a shared object space, a file object is a typed object with an interface that exports standard file operations, such as `read`, `write`, and `seek`. The interface can also define an object's properties, including its persistence, fault, synchronization, and performance characteristics. Files do not have to be of the same type.



Beyond basic sequential files, class-based persistent objects offer a range of opportunities, including:

- *Application-specific "file" interfaces.* For example, instead of just `read`, `write`, and `seek`, a 2D-array-file may have additional functions, such as `read_column`, `read_row`, and `read_sub_array`. The advantage is the ability to interact with the file system in

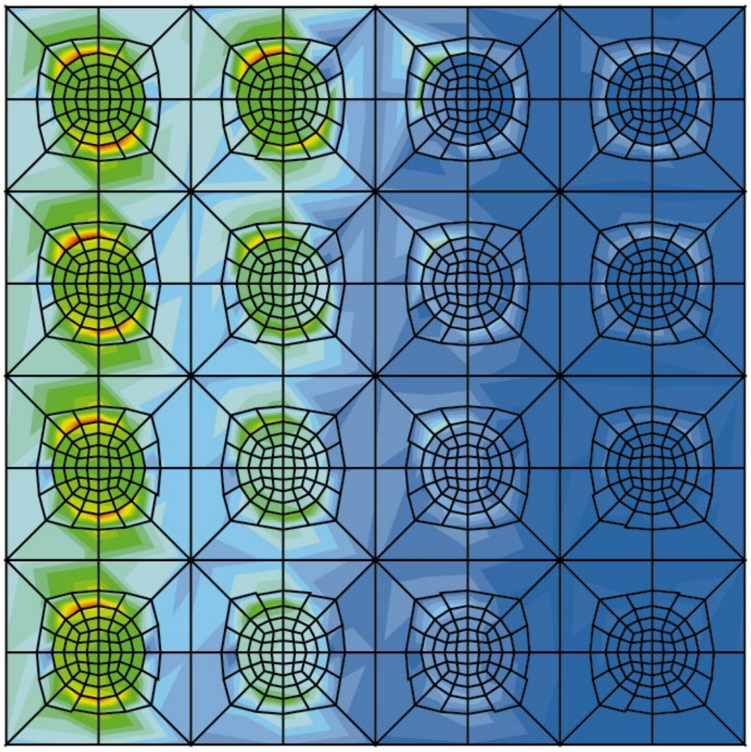
Above: Where damage starts; opposite: Stress in a composite.
(Courtesy, Kumar Vemaganti, University of Texas, Austin.)

The challenge is to provide a solid, integrated middleware foundation on which to build wide-area applications.

application terms rather than as arbitrarily linearized streams of data.

- *User specification of caching and prefetch strategies.* Users can exploit application domain knowledge about access patterns and locality to tailor the caching and prefetch strategies to the application.
- *Asynchronous I/O.* Combined with a parallel object implementation, I/O can be performed concurrently with application computation. The combination of user specification of prefetching and asynchronous I/O can drastically reduce I/O delays.
- *Multiple outstanding I/O requests.* Again, if combined with a parallel object implementation, the application can request remote data long before it is actually needed. The application can then compute while I/O is being processed. By the time data is needed, it may already be available, resulting in almost zero latency.
- *Data format heterogeneity.* Persistent classes may be constructed to hide data format heterogeneity, automatically translating data as it is read or written.
- *Active simulations.* In addition to passive files, persistent objects can also be active entities. For example, a factory simulation can proceed at a specified pace (such as wall clock time) and be accessed (read) by other objects. The factory simulation may itself use and manipulate other objects.

Transparent remote execution. A slightly more complex service is transparent remote execution. Consider the following hypothetical situation: A user is working on a computationally complex sequential code, perhaps written in C or Fortran. After compiling and linking the application, the user has to decide where to execute the code, choosing to run it on the local workstation (assuming adequate space and speed are available), on a local high-performance machine, or at a remote supercomputer center. The choices involve many trade-offs. First, there is the



scheduling issue: Which choice will result in the fastest turnaround? This decision is especially crucial when the user has accounts at multiple centers. How is the user to know which choice is likely to result in the best performance without manually checking each? Next, there are the inconveniences of using remote resources. Data and executable binaries may first have to be copied physically to a remote center and the results copied back for analysis and display. Finally, there are the administrative difficulties of acquiring and maintaining multiple accounts at multiple sites.

In a metasystem, the user can simply initiate the application at the command line. The underlying system selects an appropriate host from among those the user is authorized to use, transfers binaries if necessary, and begins execution. Data is transparently accessed from the shared persistent object space, and the results are stored in a shared persistent object space for later use.

Wide-area queueing system. Queueing systems are part of everyday life at production supercomputer centers. The idea is simple: Rather than interactively starting a job, the user submits a description of the job to a program (the queueing system), and that program schedules the job to execute at some point in the future. The purpose of the queueing system is to optimize an objective function, such as system utilization, minimum average delay, or a priority scheme. There are more than 25 different queueing systems in use today, including NQS, PBS, Condor, and Codine [8].

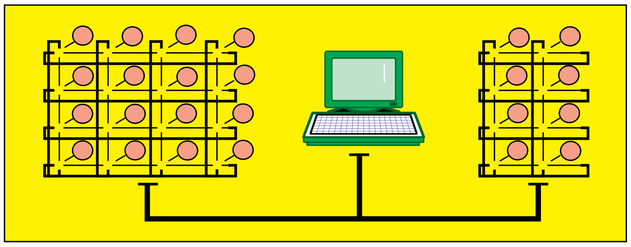


Figure 1. Two geographically separated distributed-memory MPPs connected via high-speed link and a user at a visualization station at a third site. The hosts can be the same or have different processors and interconnection networks.

Just as queueing systems can be used to manage homogeneous resources at a particular site, metasystem queueing systems manage heterogeneous resources at multiple sites. This feature enables submission of a job from any site and, subject to access control, automatic scheduling of the job somewhere in the metasystem. Further, if binaries for the application are available for multiple architectures, the queueing system has a choice of platforms on which to schedule the job.

The advantage to the user over the current state of practice in distributed systems is significant. Users with multiple accounts at different sites often have to shop around, looking for the shortest queue in which to run a job, and may need to copy data between remote sites. This manual metascheduling process consumes the most precious resource—user time—which is better spent doing science.

Wide-area parallel processing. Another opportunity offered by metasystems is to connect multiple resources for the purpose of executing a single application, enabling scientists to tackle mission-critical problems on a much larger scale than would otherwise be possible. However, not all problems can exploit this capability, since the application must be latency-tolerant (it is, after all, at least 30msec from California to Virginia). Examples of applications in this category include "bag-of-tasks" problems and many regular finite-difference methods.

Consider typical bag-of-tasks problems, such as parameter-space studies, Monte Carlo computations, and other simple data-parallel problems. In a parameter-space study, the same program is repeatedly executed with slightly different parameters. (The program may be sequential or parallel.) In a Monte Carlo simulation, many random trials are performed; the trials may be distributed easily among a large number of workers and the results gathered.

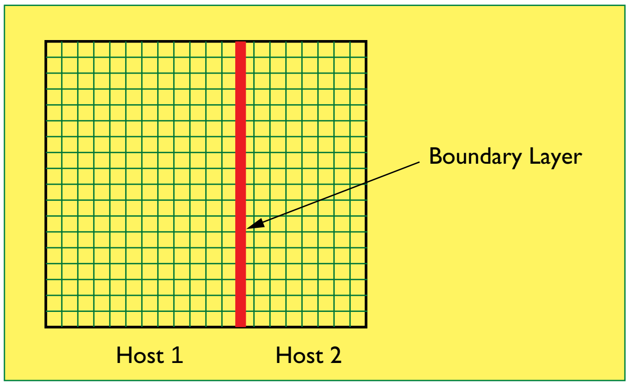


Figure 2. One possible decomposition of a 2D grid between two MPPs

In a simple data-parallel problem, the same operation is performed on each of many different data points, and the computation at each data point is independent of those for all other data points.

These bag-of-tasks problems are well suited to metasystems because the related applications are highly latency tolerant. While one computation is being performed, the results of the previous computation can be sent back to the caller and the parameters for the next computation sent to the caller. Further, the order of execution is unimportant, completely eliminating the need for synchronization among workers, thus simplifying load-balancing significantly. The computations can also be spread easily to a large number of sites because the computations do not interact in any way.

A more complex class of problems might comprise 2D finite-difference applications, such as time-explicit hydrodynamics. Suppose we wish to use two distributed-memory massively parallel processors (MPPs) and a visualization system at different sites, all connected by a fast communication pipe running at 100Mb–1Gb (see Figure 1). Further suppose that the first host has twice as many processors as the second one has. Balancing the load requires decomposition of the problem in such a way that the first host has twice as much of the data as the second.

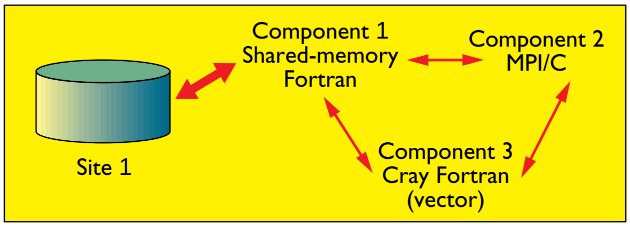


Figure 3. A multicomponent application

Given the decomposition in Figure 2 and information on the size of the mesh points, we can easily compute the bandwidth requirements of the communications channel to determine whether the total latency is acceptable.

Meta-applications

The most challenging class of applications is the meta-application—a multicomponent application whose components may have been previously executed as standalone computations.

The generic example in Figure 3 shows three previously standalone applications connected to form a larger, single application. Each of the components has hardware affinities. Component 1 is a fine-grain data-parallel code requiring a tightly coupled MPP machine, such as an IBM SP2; Component 2 is a coarse-grain data-parallel code that runs efficiently on an inexpensive "pile of PCs" machine; and Component 3 is a vectorizable code that runs best on a vector machine, such as a Cray T90. Component 1 also has a very large database physically located at Site 1. Component 1 is a shared-memory Fortran code, Component 2 is written in C using the Message Passing Interface (MPI) [7], and Component 3 is written in Cray Fortran.

The underlying software layer has to take into account component characteristics, distributed databases, scheduling, visualization, fault tolerance, and security.

Component characteristics. Meta-application components are often written and maintained by geographically separate research groups. The components may be written in different languages or may use different parallel dialects of the same language (such as Fortran components using MPI, Parallel Virtual Machine, or PVM, [4] and High Performance Fortran, or HPF). Since the components often represent valuable intellectual property for their owners, these owners may want to protect their code by keeping it in house. Similarly, software licenses may be available only on a subset of the available hosts. The metasystem has to facilitate intercomponent communication when the components are in different languages, and it has to migrate binaries transparently from site to site.

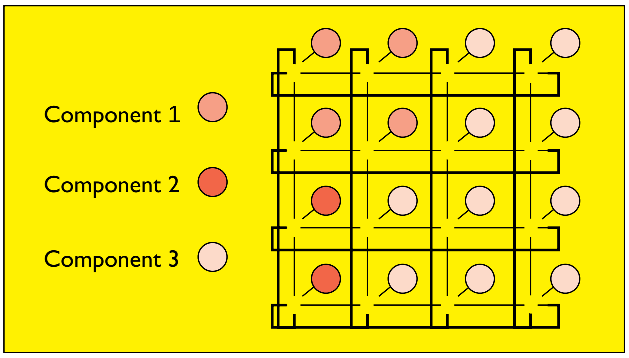


Figure 4. One possible mapping of three data-parallel components onto an MPP

Geographically distributed databases. Data is usually physically close to the research group that collects and maintains it. So before execution can begin, the data often has to be copied to a single location. The challenge to the metasystem is to determine when it makes sense to move the computation to the data or vice versa.

Scheduling. Scheduling meta-applications is another significant challenge. The general scheduling problem of mapping an arbitrary task graph onto more than three processors is known to be an extremely difficult problem. Application-class-specific heuristics that exploit knowledge about the application (the "shape" of the graph) have to be used [1]. Even so, the scheduling problem is quite difficult.

Consider scheduling a simple meta-application with three data-parallel components onto a single distributed-memory MPP (see Figure 4). First, the number of processors allocated to each component must ensure that each component progresses at the same rate; efficiency may require that each component use a different number of processors. Second, the component tasks have to be mapped to the

Metacomputing is indispensable for the success of large, complex simulations.

processors in such a way as to reduce the communication load among them; random placement can lead to communication bottlenecks. Finally, the computational requirements of the components may vary over time, requiring dynamic repartitioning of the available resources.

Now suppose that instead of a fixed number of processors on an MPP, we have to choose among a large number of diverse systems, each connected to the others by networks of widely varying capability. The scheduling problem is certainly a tough one.

Visualization. During computation, users may want to see what is happening, at the global scale and at smaller scales. Further, they may wish to look inside individual models at particular points and perhaps adjust the computation by modifying some parameters.

Fault tolerance. As the number of hosts and processors participating in the computation increases, the probability of a failure increases and the mean time to failure decreases. When the mean time to failure is less than the completion time for a run of the application, the probability that the application will finish successfully is low. Fault tolerance is a necessity under these circumstances.

Security. Security encompasses a range of issues, including authentication, access control, and encryption. Different users have different requirements. Suppose, though, that a meta-application is using databases that are distributed geographically. These databases must be protected against unauthorized access and update, and the distributed components' results must be protected from tampering or destruction. Finally, data transferred among components may be proprietary and need protection against tampering and snooping.

A meta-application, in sum, may be composed of multiple models running at different scales, written by different research groups using different languages or parallel processing tools, using proprietary, geographically distributed databases, on faulty hardware. Such applications require a system that cleanly and easily supports interoperability among components, as well as the plug-and-play incorporation of components into a running program and the scheduling of components onto processing resources (within a large MPP and between hosts). Further, the system must support transparent, secure, and efficient access to remote databases while providing robust integrated visualization.

Metacomputing at NPACI

Metacomputing is one of NPACI's four enabling technology thrust areas. (The other three are also covered in this issue.) The NPACI metasystems plan consists of three broad components: rapid hardening and deployment of leading-edge metasystems software, integration of software and tools developed in the other technology thrust areas, and close collaboration among metasystem thrust teams and pioneering applications teams.

NPACI does not support basic research per se in metasystems or in the other thrust areas. Rather, it supports technology transfer from existing funded research projects to the production computing environment. However, research software is rarely immediately ready for production use. Before such software can be used, it must undergo aggressive testing and elimination of bugs. And the development of documentation, sample codes, tutorials, and other training materials is needed to make the software accessible to mainstream scientific users.

The NPACI hardening effort consists of four metasystems projects: AppLeS [1], Globus [2, 3], Legion [5, 6], and the Network Weather Service (NWS) [11].

AppLeS (Application-Level Scheduler). Directed by Fran Berman of the University of California, San Diego, this project focuses on developing application schedulers for individual metasystem applications. Each AppLeS agent couples with its application to develop and deploy a customized application schedule that can respond to the dynamic and heterogeneous performance characteristics of the underlying metasystem. AppLeS uses application-specific information, dynamic system information (provided by NWS), and predictive models to develop a performance-efficient, time-dependent, and load-dependent schedule.

AppLeS applications can either target networked environments with no additional infrastructure or use the infrastructure provided by other NPACI metasystem projects. AppLeS agents are being developed and deployed for such NPACI applications as protein-docking codes, a synthetic-aperture radar atlas (SARA), and tomography codes.

Globus. Globus is an infrastructure toolkit developed by Carl Kesselman of the University of Southern California and Ian Foster of Argonne National Laboratory with a broad group of collaborators. It provides services in the areas of communication, resource location/allocation, security, information, and data access. Globus has withstood many tests, including a recent one involving battlefield simulations distributed across more than 30 machines and representing the independent activity of more than 100,000 tanks, trucks, and other units.

For each of Globus's five service areas, there is a defined application programming interface (API) and a set of implementation notes explaining the semantics and the implementation. For example, Globus communication services (the Nexus communication library) provide unicast and multicast message delivery services. Globus information services (through its Metacomputing Directory Service) provide a uniform mechanism for obtaining real-time information about system structure and status. Globus resource location and allocation services provide mechanisms for expressing application resource requirements, for identifying resources meeting these requirements, for scheduling resources once they are located, and for initiating and managing computation on these resources.

Legion. Legion is a reflective object-based metasystem being developed by the author Andrew Grimshaw and his group at the University of Virginia. Legion aims to provide a single, coherent, virtual machine addressing scalability, programming ease, heterogeneity, fault tolerance, security for users and resource providers, site autonomy, multilanguage support, and interoperability.

Legion supports a range of services and programming tools, including a shared, secure persistent object space supporting authentication and arbitrary access control lists for objects; parallel programming tools, such as MPI, PVM, the Mentat programming language, and Fortran dataflow; complete site autonomy (a site can decide which users can run which binaries on the site); a complete set of authentication, encryption, and access control services; and user-defined scheduling and resource management.

Network Weather Service. NWS, being developed by Richard Wolski of the University of California, San Diego, dynamically forecasts the performance that various network and computational resources can deliver over a given time interval. It is being used to track and monitor the performance of end-to-end very high-speed Backbone Network Service (sponsored by the NSF). It operates a set of sensors (network and CPU monitors) from which it gathers readings of current conditions. It then uses sophisticated numerical models to generate forecasts of what the conditions will be for a given time frame. The supported forecasting methods treat successive measurements from each monitor as a time series. These methods fall into three categories: mean-based, median-based, and autoregressive methods.

NWS tracks the accuracy of all predictors to generate a forecast, using prediction error as an accuracy measure and selecting the predictor exhibiting the lowest cumulative error measure at any given moment. In this way, NWS automatically identifies the best forecasting technique for any given resource.

Metasystems represent the glue that unites all of the other NPACI projects. Over time, the other enabling technologies and the applications prototypes will exist in a metacomputing environment. For example, KeLP coupling and interpolation and the NetSolve numerical object environment (described in J. Saltz's article in this section) are already being integrated with Legion and Globus. And application integration is moving ahead. An example is the collaboration between the Legion team and an earth science application originating at the University of California at Los Angeles.

Coupled Ocean-Atmosphere Modeling

Global climate modeling is an example of a field that can benefit from metacomputing. Climate modeling has progressed beyond atmospheric simulations to include multiple aspects of the Earth system, such as full-depth world ocean models, high-resolution land-surface models, sea-ice models, and chemistry models. Each component model generally requires a different resolution in space and time. Some components, such as a full model of atmospheric chemistry, can add as much as an order of magnitude to the total processing time.

The models usually originate with different research groups around the world and are written in different languages. An additional complication is that many models have parallel implementations, often using different parallel toolkits. With existing tools, coupling these models would be tedious and error-prone at best; not only do the data formats and protocols have to be implemented by hand, but (for the sake of performance) models have to be scheduled appropriately on the available resources. In addition, databases required for the component models are usually geographically distributed, and if there is a huge quantity of data, the data needed for a desired set of resolutions and models may not fit on the machine on which the simulation is to be performed.

The metacomputing environment will enable the models to run on different, perhaps physically remote, machines, as long as fault tolerance and security are addressed. Security problems are exacerbated by the operation over a network of far-flung resources, because the data and the results of the coupled models are the intellectual property of the researchers who compiled the input data and produced the results. A final issue is visualization. The larger and more complex the simulation, the more critical is the need for visualization—for humans to be able to digest the enormous amount of data generated by high-resolution scientific models.

The Earth System Model (ESM) being developed by NPACI partners R. Carlos Mechoso and Richard Turco at UCLA is a coupled atmosphere and ocean model that can include atmospheric and ocean chemistry to study such problems as the effect on climate of chlorofluorocarbons and aerosols. Work is also under way to couple a regional mesoscale model developed at the Lawrence Berkeley Laboratory to the ESM to enable researchers to focus on the effect of global climate phenomena, especially El Niño and La Niña, on California weather. Future plans call for even smaller scales, down to individual bays and estuaries along the California coast.

The global atmospheric model is currently being run at a resolution of approximately 1.25 degrees of latitude × 1 degree of longitude, with 30 vertical atmospheric levels; such a model can resolve large cyclonic weather systems. Although this is state-of-the-art climate modeling, it is still too coarse a grid to resolve details of smaller individual weather systems. The mesoscale model is run with a resolution of 20km over a region encompassing mainly California and Nevada.

A full system of this nature would be nearly impossible to run on existing individual supercomputers, even those that are massively parallel. ESM alone can consume most of the cycles of an MPP Cray T3E, while the current implementation of the mesoscale model runs most efficiently on machines like a Cray C90 or T90. Therefore, the Legion research team is developing a Legion-based coupling module to manage the data exchange and synchronization among various models running on different hardware. The researchers in charge of the component models would need to add only a few communication calls to their codes; no major changes or rewriting would be required—a major advantage when dealing with complex scientific models.

ESM communicates internally via a native parallel toolkit, such as MPI or PVM. There is no impediment to using Legion in this situation, but we have decided not to alter the current implementation. The ESM master process acts as the intermediary between ESM and the Legion coupler. The Legion coupler receives data from ESM, performs the appropriate transformations from the ESM grid to the regional grid, and sends the transformed data to the regional model. From the perspective of the models, this is little different from writing to or reading from a file. Once full coupling is achieved between ESM and the regional model, a dynamic interface is essential, but much of the communication effort has already been completed by way of a unidirectional interface. Moreover, maintaining a separate "manager" means that resolutions can be changed and models interchanged or added with little difficulty.

This coupling project illustrates clearly the advantage of a metacomputing approach for a demanding, important scientific simulation. The models run optimally on different architectures but must communicate at regular intervals to calibrate one another. Each model has to use a large store of surface data that might be in a format incompatible with the architectures on which the other models are being run. The lengthy runs required for climate simulation make fault tolerance imperative, especially in a coupled environment in which one model could hang hopelessly if another fails. Metacomputing is thus indispensable for the success of large, complex simulations.

Computational Productivity

Metasystems technology is maturing. Three years ago at the Supercomputing95 conference, the I-Way was a one-time demonstration of a large number of applications that had been constructed in a rather ad hoc manner. Today, metasystems testbeds are operational on an almost full-time basis. As the technology matures further and becomes hardened enough for use in production systems, we can expect a significant increase in the computational productivity of the sciences. C

This work is supported in part by DARPA (Navy) contract N66001-96-C-8527; DOE grant DE-FD02-96ER25290; DOE contract Sandia LD-9391; Northrup-Grumman (for the DoD HPCMOD/PET program); DOE D459000-16-3C; and DARPA (GA) SC H607305A.

References

1. Berman, F., and Wolski, R. Scheduling from the perspective of the application. In *Proceedings of the 6th IEEE Symposium on High-Performance Distributed Computing*, IEEE Computer Society Press, Piscataway, N.J., 1996.
2. Foster, I., and Kesselman, C. Globus: A metacomputing infrastructure toolkit. *Int. J. Supercomput. Appl. 11,* 2 (1997), 115–128.
3. Foster, I., and Kesselman, C., Eds. *The Grid, Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, San Francisco, 1998.
4 Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R., and Sunderam, V. *PVM: Parallel Virtual Machine*. MIT Press, Cambridge, Mass., 1994.
5. Grimshaw, A., and Wulf, W. The Legion vision of a worldwide virtual computer. *Commun. ACM 40,* 1 (Jan. 1997), 39–45.
6. Grimshaw, A., Ferrari, A., and West, E. Mentat. In *Parallel Programming Using C++*, G. Wilson, Ed., MIT Press, Cambridge, Mass., 1996.
7. Gropp, W., Lusk, E., and Skjellum, A. *Using MPI: Portable Parallel Programming with the Message Passing Interface*. MIT Press, Cambridge, Mass., 1994.
8. Kaplan, J., and Nelson, M. A comparison of queueing, cluster, and distributed computing systems. NASA Tech. Memo. 109025, NASA LaRC, 1993.
9. Levy, E., and Silberschatz, A. Distributed file systems: Concepts and examples. *ACM Comput. Surv. 22,* 4 (Dec. 1990), 321–374.
10. Morris, J., et al. Andrew: A distributed personal computing environment. *Commun. ACM 29,* 3 (March 1986).
11. Wolski, R. Dynamically forecasting network performance to support dynamic scheduling using the network weather service. In *Proceedings of the 6th IEEE Symposium on High-Performance Distributed Computing*. IEEE Press, Piscataway, N.J., 1996.

Andrew Grimshaw (grimshaw@cs.virginia.edu) is an associate professor of computer science at the University of Virginia.
Adam Ferrari (ferrari@cs.virginia.edu) is a research scientist on the Legion project at the University of Virginia.
Greg Lindahl (lindahl@cs.virginia.edu) is a research scientist at the University of Virginia.
Katherine Holcomb (kholcomb@cs.virginia.edu) is a research scientist at the University of Virginia.