Metacomputing – What's in it for me?

Greg Lindahl
Andrew Grimshaw
Adam Ferrari
Katherine Holcomb

University of Virginia Computer Science Department
Thornton Hall
Charlottesville VA 22903
+1 804 982-2293
lindahl@cs.virginia.edu

Introduction

We are often asked by applications programmers, "What exactly is a metasystem, and why should I care?" This paper attempts to answer this question from an applications perspective, pointing out concrete ways in which the current practices of high performance scientific computing can possibly be improved.

What Is A Metasystem?

Before we can answer the challenge posed by the title of this paper, we need to define what a metasystem is. Physically, a metasystem is a collection of geographically separated resources (people, computers, instruments, databases) connected by a high speed network. A metasystem is distinguished from a simple collection of computers by a software layer, often called middleware, which transforms a collection of independent resources into a single, coherent, virtual machine. This machine should be as simple to use as the machine on the user's desktop, and should allow easy collaboration between colleagues located anywhere in the world.

What is the problem with today's collections of computers? A typical researcher using machines at multiple sites faces the problem of slightly or radically different software environments, separate filesystems which require frequent copying of files between sites, security policies which prohibit transfer of files between sites without a human typing in a password, and so forth.

So why don't we have metasystems today? As usual, the fundamental difficulty is software, specifically an inadequate model of "systems software" for the worldwide collection of computer systems. Faced with the eternal rush of new hardware, the computing community has stretched existing models – interacting but autonomous computers – to a level where this model breaks down. The result is a collection of incompatible, incomplete solutions which work well in isolation, but do not work together nor scale for the future.

Our vision of a metasystem[4] is of a system containing thousands of computers and terabytes of data in a loose confederation, tied together by high-speed networks. The user will have the illusion of a very powerful computer on her desk, and will manipulate objects representing data resources such as databases of physical data, applications such as physical simulations and visualization tools, and physical devices such as scientific instruments. To allow the construction of shared workspaces, these objects may be securely shared with other users.

It is the metasystem's responsibility to support this illusion of a single machine by transparently managing data movement, caching, and conversion; detecting and managing faults; ensuring that the user's data and physical resources are adequately protected, and scheduling application components on the resources available to the user.

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

- more effective collaboration, by putting co-workers in the same virtual workplace,
- higher application performance due to parallel execution and exploitation of off-site resources,
- improved productivity through a considerably simpler programming environment.

The next section of this paper will introduce an example application, which we will use in the subsequent section to illustrate the benefits of the major subsystems of the Legion metacomputing system.

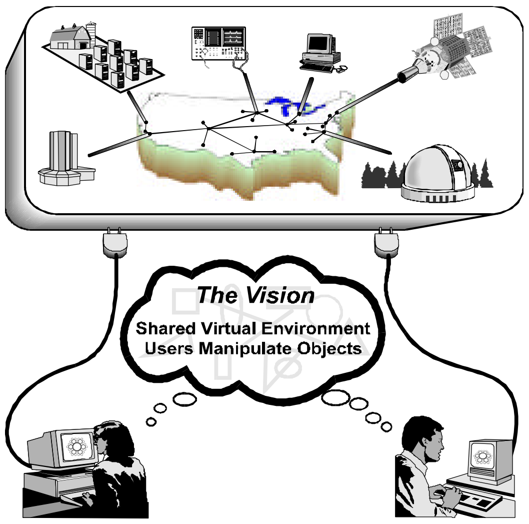


Figure 1. Our vision is to construct a shared, high-performance, secure, and fault tolerant computational environment for scientists and engineers. The resulting environment will enhance the productivity of the scientific community.

Example – Multi-Scale Climate Modeling

Climate modeling is an example of a field that can benefit greatly from metacomputing. Climate modeling has progressed beyond atmospheric simulations to include multiple aspects of the Earth system, such as full-depth ocean models, high-resolution land-surface models, sea ice models, chemistry models, and so forth. Each component model generally requires a different resolution in space and time. We might even wish to couple global and regional models. For example, an El Nino study might involve coupling a global climate code with a regional weather code.

Over the course of simulating a coupled system, the individual models must exchange data, such as temperature, winds, and precipitation, and the execution of the entire system must be kept in synchronization. The models often originate from different research groups around the world and may even be written in different languages. As an additional complication, some models have parallel implementations, often using different parallel toolkits.

With existing tools, coupling these models would be tedious and error-prone at best. Many parallel toolkits are incompatible with each other, and do not support heterogeneous collections of systems. In the metacomputing environment, the models could interoperate on the same or different machines, which need not be in close physical proximity. While this solves some of the problems of coupled climate models, other difficulties remain. Fault tolerance and security are major issues. Many scientific models write restart dump files at regular intervals, and can be restarted by hand after a system failure, but this is only a partial solution. Constant restarting is an aggravation at best; at worst it is a waste of resources and researcher time, as well as an invitation to error. Security problems, which can often be neglected inside a single machine, are also potentially increased by use of far-flung resources.

A final issue is visualization. The larger and more complex the simulation, the more critical is the need for visualization, in order for humans to be able to digest the enormous amount of data generated by high-resolution scientific models. But it can be difficult to couple visualization tools to applications.

We are currently constructing such a multi-scale climate model as part of the NPACI ESS effort, linking together the UCLA OGCM/AGCM code with a regional California code.

So What Can I do With a Metasystem?

Our initial description of a metasystem is rather vague and high-level. The real question for users is: What's in it for me? There are many ways to use the new capabilities that a metasystem provides. They range from relatively simple use of Legion facilities to intricate usages which exploit Legion's capabilities to solve problems currently considered impossible. Below, we sketch out several uses of metasystem technology, and the capabilities that these features add to your research.

Shared Persistent Object (file) Space

The simplest service a metasystem provides is location-transparent access to data files, which is usually called a distributed filesystem. An ideal distributed filesystem allows a user to access a file anywhere in the world without knowing if the file is local or remote, and without involving her systems administrator. Having a shared filesystem significantly simplifies collaboration. NFS is a well-known example of a distributed filesystem[2]. However, NFS requires super-user configuration and has significant security implications, so few users are able to use NFS to access remote data, or collaborate with colleagues in remote locations. The World Wide Web provides a limited (read-only) distributed filesystem. Legion's shared object space provides shared, secure access to data files without super-user configuration.

A more powerful model than shared files is shared object spaces. Instead of just sharing files, all entities – programs, databases, instruments, etc. – can be named and shared between users. This merging of "files" and "objects" is driven by the observation that the traditional distinction between files and other objects is not necessary. Files represent persistent data, and happen to live on a disk, so files are slower than RAM, but persist if the computer crashes. In a shared object space, a file object is any object which supports the standard file operations, such as "read" and "write". In addition, the object interface can also define additional properties such as its persistence, fault, synchronization, and performance characteristics. Not all files need be the same; this eliminates the need to provide Unix synchronization semantics for all files, since many applications simply do not require those semantics. Instead, special semantics can be selected on a file-by-file basis, and even changed at run-time.

Beyond basic sequential files, persistent objects with flexible interfaces offer a range of opportunities, including:

- *Application-specific "file" interfaces*. Instead of just read and write, a "2D array file" may also have functions such as "read_column", "read_row", and "read_sub_array". The advantage to the user is the ability to interact with the file system in application terms – in this example a special object which efficiently reads and writes 2D files – rather than just one-dimensional streams of bytes. The implementations of files can be optimized for a particular data structure, by storing the data in sub-arrays, or by scattering the data to multiple devices to provide parallel I/O. Note that these characteristics can be set on a file-by-file basis, unlike most current parallel filesystems.

- *User specification of caching and prefetch strategies.* This feature allows the user to exploit application domain knowledge about access patterns and locality to tailor the caching and prefetch strategies to the application. For example, the user may know what data she might read minutes in advance.
- *Active simulations.* In addition to passive files, persistent objects may also be active entities. For example, a factory simulation can proceed at a specified pace (e.g., wall clock time) and can be accessed (read) by other objects. Of course the factory simulation may itself use and manipulate other objects.

Transparent Remote Execution

A slightly more complex service is that of transparent remote execution. Consider a user working on a code. After setting up the initial data for a run, she is left with the problem of deciding where to execute the code. She might choose to run it on her workstation (if it has sufficient resources), or on a local high performance machine, or on a remote workstation farm, or on a remote supercomputer. The choice involves many trade-offs. Which choice will result in the fastest turn-around? Today, a user must usually check each potential machine by hand to guess the turn-around time.

Next, there are the inconveniences of using remote resources. Data and executable binaries may first need to be physically copied to a remote center, and the results copied back for analysis and display. This may be further complicated by the need to access input data from a collaborator. Finally, the user must recall how to use the local queuing system; there are 25 different ones in use [3]. These inconveniences are usually so great that most users pick one site and infrequently consider moving. Finally, there are the administrative difficulties of acquiring and maintaining multiple accounts at multiple sites.

In a metasystem, the user can simply execute the application at the command line. The underlying system selects an appropriate host from among those the user is authorized to use, transfers program binaries, and begins execution. Data is transparently accessed from the shared persistent object space, and the results are stored in the shared persistent object space for later use. A queuing system could be used, in order to create a wide-area queuing system, extending today's local queuing systems.

Wide-Area Parallel Processing

Another opportunity presented by metasystems is connecting multiple resources together for the purpose of executing a single application, providing the opportunity to run problems of a much larger scale than would otherwise be possible. Not all problems will be able to exploit this capability, since the application must tolerate the latency involved in crossing a building or crossing the country.

First, let's consider a parameter space study. In a parameter space study, the same program is repeatedly executed with slightly different parameters. The program may be sequential or parallel. For example, a convergence study might involve running the same code repeatedly with different grid sizes, or with slightly perturbed initial values. These sorts of problems are sometimes called "bag of tasks" problems, since all the runs are independent.

Bag-of-tasks problems are well suited to metasystems because they are highly latency tolerant. While one computation is being performed, the results of the previous computation can be sent to the results bag, and the parameters for the next computation can be retrieved from the input bag. Furthermore, the computations can be easily spread to a large number of sites (using Legion's remote execution capability), because the computations do not interact in any way.

.

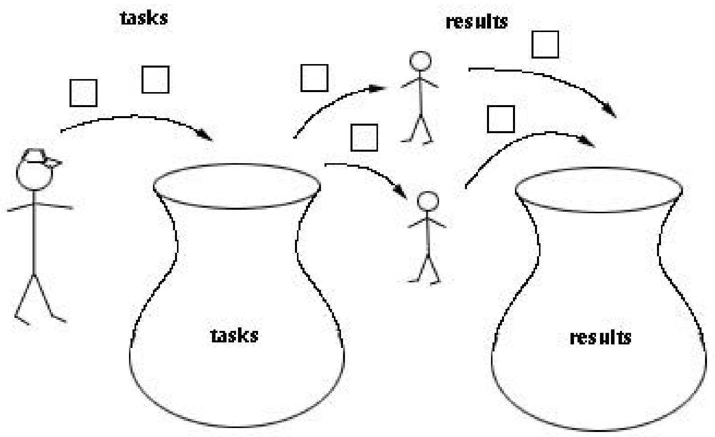


Figure 2. Bag-of-tasks. The master places work units containing parameters for the workers into the tasks bag. Workers take work from the bag, perform the required computation, place the results into another bag, perhaps used by other workers as input, and retrieve another piece of work. This continues until all of the work has been completed. Note that there may be more than one master.

Consider next a more complex class of problems, such as an extremely large ocean model.[1] Suppose that we wish to use two distributed memory MPPs and a visualization system at different sites for a single run. All 3 sites are connected by a fast network running at 155 megabits per second. (Figure 3). Further suppose that the first host has twice as many processors as the second does. Balancing the load requires that the problem be decomposed in such a manner that the first host has twice as much of the data as the second. (In general, the scheduling problem can become extraordinarily complex, so a simple example is used here to illustrate the point.)

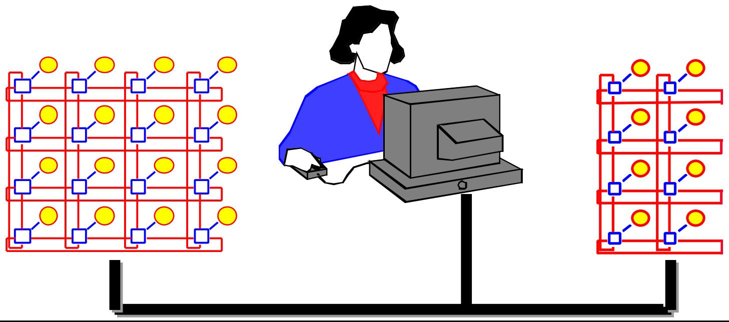


Figure 3. Two geographically separated distributed memory MPP's connected by a high-speed link. The user sits at a visualization station at a third site. The hosts may be the same or have different processors and interconnection networks.

[1] Most ocean codes are 3D, but are decomposed in 2D.

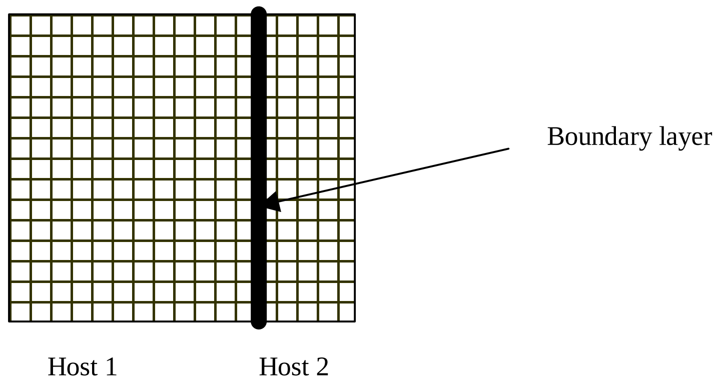
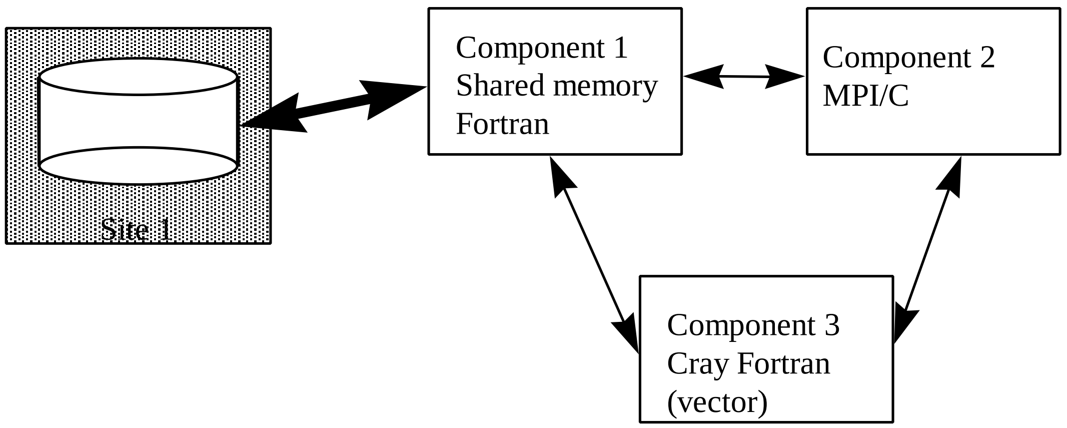


Figure 4. One possible decomposition of a 2D grid between two MPP's. The thick line represents the boundary layers that must be sent between hosts.

Given the decomposition shown above and information on the size of the zones, we can estimate the bandwidth requirements of the communications channel. Suppose that the problem is 10,000 by 10,000 zones, and each zone communicates with its 4 immediate neighbors only once per cycle, and each zone contains 100 bytes of data. Then one megabyte of data must be transferred over the wire in each direction for each cycle. Assuming coast-to-coast communication and a 155 megabit channel, the time to transmit the boundary layer is at least 50 ms. For some applications, such as those which use an implicit solver and transfer information many times during a cycle, that is likely to be too long. But for others, 50 ms is acceptable, especially if communication can overlap with computation.[2]

Meta-Applications

The most challenging class of applications for the metasystem is meta-applications. A meta-application is a multi-component application, many of whose components were previously executed as stand-alone applications.



A generic example is shown above. In this example three previously stand-alone applications have been connected together to form a larger, single application. Each of the components has hardware affinities. Component 1 is a fine grain data parallel code that requires a tightly coupled machine such as an SGI Origin 2000, component 2 is a coarse grain data parallel code that can run efficiently on a "pile of PCs"

[2] Deciding the partition and placement of cells on processors can be difficult to get right if done by hand, but fortunately there are tools for making those decisions [4]. The issue of dynamically changing the partition and placement has not been solved.

machine, and component 3 is a non-parallelized but vectorizable code that "wants" to run on a vector machine such as Cray T90. Component 1 also uses a very large database that is physically located at site 1. Component 3 is written with Cray Fortran extensions, component 1 is written in HPF, and component 2 is written in C using MPI calls.

There are many difficult issues involved in this example. The first is that data is often geographically distributed – it is often stored physically close to the people who collect it, not necessarily the people who use it. Today's coupled models usually require all the data to be copied to a single location. The challenge to the metasystem is to help determine when it makes sense to move the computation to the data, and when it makes sense to move the data to the computation.

Next, scheduling the meta-application onto the hardware is a significant challenge. Consider scheduling our example meta-application on a single distributed memory machine. We would like each component to progress at the same rate, so we might need to assign different numbers of processors to each. Second, the component tasks must be mapped to the processors in such a manner as to reduce the communication load – random placement may lead to communication bottlenecks. Finally, the computational requirements of the components may vary over time, requiring dynamic re-partitioning of resources.

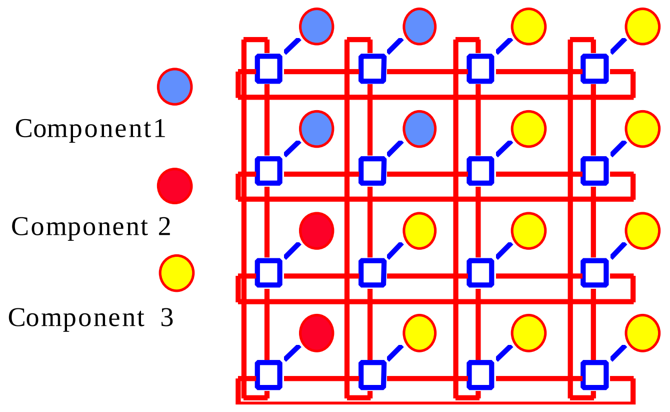


Figure 6. One possible mapping of three data-parallel components onto an MPP.

Now suppose that instead of a fixed number of processors on an MPP to choose from, we must choose between a large number of diverse systems, each connected to the others by networks of widely varying capability[3]. It is easy to see that the scheduling problem is a significant challenge.

As the number of hosts and processors in a computation increase, the mean time to failure falls. Today's large machines are less reliable than the machines used 5 years ago; collections of workstations have always presented a fault tolerance challenge. The metasystem should provide transparent fault tolerance whenever possible.

[3] Just determining network characteristics is non-trivial in a metasystem. Unlike an MPP, which is often not shared, the wide-area network is usually shared, resulting in large variances in both bandwidth and latency. Predicting network performance [5] is a critical component of the metasystem scheduling problem.

Summary

Metasystems technology is rapidly maturing. Three years ago at Supercomputing ’95, the I-Way was a one-time stunt that demonstrated a large number of applications that had been constructed in an *ad hoc* fashion. Today, metasystems testbeds are operational on a full-time basis. As the technology matures further and becomes hardened enough for production use, we hope to see a significant increase in computational scientists’ productivity.

Metasystems will provide users with a transparent, distributed, shared, secure, and fault-tolerant computational environment. With a metasystem, users will be able to share files, computational objects, databases, and instruments. No longer will they have to manually decide where to execute their programs and copy the necessary binaries and data files: the metasystem will do those jobs. New classes of applications, meta-applications, will be enabled by the new infrastructure, further increasing users' efficiency and productivity.

Acknowledgments: This work partially supported 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. The authors would also like to thank Sarah Parsons Wells for her editing assistance.

References

1. A. Grimshaw and W. Wulf, "The Legion Vision of a Worldwide Virtual Computer," *Communications of the ACM*, pp. 39-45, vol. 40, number 1, January, 1997.
2. E. Levy, and A. Silberschatz, “Distributed File Systems: Concepts and Examples,” ACM Computing Surveys, vol. 22, No. 4, pp. 321-374, December, 1990.
3. J.A. Kaplan and M.L. Nelson, “A Comparison of Queueing, Cluster, and Distributed Computing Systems,” NASA Technical Memorandum 109025, NASA LaRC, October, 1993.
4. J. B. Weissman, A.S. Grimshaw, “A Framework for Partitioning Parallel Computations in Heterogeneous Environments “, Concurrency: Practice and Experience, pp. 455-478, Vol. 7(5), August, 1995.
5. R. Wolski, “Dynamically Forecasting Network Performance to Support Dynamic Scheduling Using the Network Weather Service,” *6th IEEE Symposium on High Performance Distributed Computing*, 1996.