

**Abstract**

The goal of a metacomputing system is to increase job throughput by using all available resources in a grid, while transparently providing for resource heterogeneity, network heterogeneity, and a security model for the multiple administrative domains that correspond to the autonomous sites comprising the underlying grid. The initial vision of metacomputing was to harness idle cycles of a local network, and grew to encompass those in geographically distinct government laboratories. Such systems were not without problems: each cluster had different hardware, resulting in non-portable and non-scalable software and preventing the realization of infinitely-scalable computing. In recent years several new systems have been developed with a focus on developing working systems that provide a comprehensive set of resources. We examine a representative subset of these systems in this paper, as well as some of the conceptual problems stemming from the inhomogeneities in a metacomputer.

# A Perspective on Metacomputing

Abhishek Agrawal, Craig Donner, Neil Jones, Ritu Mahajan

December 1, 2001

## 1 Introduction and Background

Consider the spectrum of parallel computing systems, with a tightly coupled Massively Parallel Processor on one end and a loosely-coupled “grid” of computers on the other. A metacomputing system would fall somewhere near the middle of such a spectrum; these systems are often touted as the “next great computing platform.” We examine this claim from the vantage point of impartial observers. In this paper we introduce the essential features of a metacomputer and explore the challenges involved in the construction of such a system.

For the purposes of this paper, we will refer to the abstract notion of work—a sequence of algorithms, data, and the software components that accomplish a particular computational task—as a *job* or an *application*. We will refer to anything that is used in a job as a *resource*, such as CPU cycles, data, primary/secondary/tertiary storage, specialized software packages, or specialized devices. We will define a *node* to be a provider of a resource, and a *site* a collection of geographically colocated nodes. A *grid* is a collection of autonomous sites. A *Metacomputer*, then, is a middleware platform to enable a job to be mapped to nodes in a grid. This mapping is determined in large part by a scheduler subsystem that compares the resource requirements of the job to the resources provided by each site. Although this mapping problem is a difficult and intrinsically interesting research question, it is beyond the scope of this paper. We point out that there is no commonly accepted definition of a metacomputer, which itself poses a significant difficulty in metacomputing research. We introduce the notions above in order to accurately describe the systems that we investigate below.

The goal of a metacomputing system is to increase job throughput by using all available resources in a grid, while transparently providing for resource heterogeneity, network heterogeneity, and a security model for the multiple administrative domains that correspond to the autonomous sites comprising the underlying grid. Metacomputers are scalable in that new resources may join the grid, applications may immediately take advantage of these resources, and the new resources do not cause inefficiencies in resource allocation and scheduling

---

\*We would like to thank Dr. Henri Casanova for his input to the research for this paper.

algorithms. Metacomputers do not attempt to provide low-latency calculations, nor will one consistently improve the performance of an inherently sequential application by executing it in a metacomputing environment; the intent of a metacomputer is simply to increase the resources available to a parallelizable computational problem and to efficiently overlap multiple jobs to minimize the price-to-performance ratio of hardware.

Metacomputing began in a very basic form at NCSA in the late '80s as the questions that scientists posed outgrew the boundaries of a single super-computer, sometimes by several orders of magnitude[11]. Noticing that such resource boundaries were often caused by lack of quantity rather than lack of quality, engineers began to consider the possibility of connecting multiple machines to accommodate these increasing computational demands. The first few systems consisted primarily of vector processors and massively parallel processors coupled to a common filesystem through a fast network, but quickly grew to incorporate microprocessors and other devices. The initial vision was to harness idle cycles of a local network, and grew to encompass those in geographically distinct government laboratories. Such systems were not without problems: each cluster had different hardware, resulting in non-portable and non-scalable software and preventing the realization of infinitely-scalable computing. Programming tools and environments lacked support for these ad-hoc clusters of supercomputers, and though the scientific questions could be feasibly answered, the resulting systems were not particularly efficient. In recent years several new systems have been developed with a focus on developing working systems that provide a comprehensive set of resources. We examine a representative subset of these systems in this paper.

Section 2 of this paper discusses our view of the software requirements of a metacomputing middleware system, while section 3 examines several important metacomputing systems in the light of these requirements. Finally sections 4 and 5 present and discuss our findings.

## 2 Software Requirements

The requirements that we set forth below for a metacomputing environment serve a dual purpose: to distinguish a metacomputer from the conceptual notion of grids and from MPPs, and to enumerate the problems that a metacomputing system should address.

Though a metacomputer requires an underlying grid to operate, it represents a more comprehensive set of services to manage the scheduling and security requirements of a job and of the grid itself. In this way, a metacomputer exceeds the basic requirements of a grid by providing facilities for a job to expand beyond the capacity of a single node. Because a metacomputer arises from a loosely coupled collection of nodes, it also differs from tightly-coupled parallel processors. For example, the fastest computational node in a grid might not be the best node for a given algorithmic step if the node is connected via a slow link and the algorithm requires a large amount of data transfer.

Some of the most challenging problems associated with a metacomputer include scalability, resource discovery and management, security, interoperability with existing operating system and language platforms, and a constantly changing hardware configuration. In order for a metacomputing system to properly compensate for these problems, it must exhibit the characteristics that we list below. If a metacomputing system fails to provide any of these services, applications that would use the system will necessarily provide their own implementation of them, leading to the inherent unreliability and non-portability of an ad-hoc solution. In general, we simply require that a metacomputing system provide *some* implementation of the following, but we do not state exactly what that implementation might be.

**Resource Discovery and Management** The substrate grid contains a set of resources that changes over time. When a new node joins the grid it must advertise the availability of some or all of its resources, while nodes already extant in the grid must be able to discover these new resources and make use of them.

**Resource Specification** A metacomputing environment must provide some facility for a job to enumerate the resources that it will require during execution. For example, a system could provide a resource definition language that is used by the scheduling facility. A more elegant, though difficult, solution would be to provide a language environment that would recognize and define these requirements implicitly. Both the resource requirements for a job and the available resources on the grid may be used as input to determine which jobs map to which resources, though the exact methods for determining this mapping are the subject of active research and are beyond the scope of this paper.

**Security, Authentication, Authorization** Since the grid is composed of autonomous sites that each contain a different set of users, a metacomputing system must provide safeguards against both malicious users and malicious nodes. A node  $X$  must be able to authenticate a user  $U$  and provide  $U$  with the authority to use some amount of resource  $R$ . Similarly,  $U$  should be able to verify that  $X$  really is providing  $R$ , and not some interloping node  $Y$ .

**Fault Tolerance and Data Replication** An MPP has a static set of nodes and resources on which to run a computation. In a metacomputer, however, a node may join or leave the substrate grid without disrupting a running computation. When a node that is performing some portion of a job leaves the grid or crashes, the metacomputing system must provide some recovery mechanism for the job. Additionally, persistent data in a metacomputing system must be replicated coherently across sites as needed.

**Parallelization and Packaging** In order to develop applications for a metacomputing system, one must be able to describe parallel algorithms using a

set of both sequential and parallel primitive operations. The metacomputing environment must specify exactly what those primitives are, as well as the granularity of parallelization: are entire subroutines run sequentially but separate subroutines parallelizable, or are modules parallelizable?

We point out that though these are necessary requirements they may not be sufficient; political and economic pressures have historically dominated the metacomputing arena. An attractive feature of a metacomputer would be support for the execution of legacy applications with only minor modification, given the total tonnage of code that domain scientists have already written. Because this sort of requirement does not substantially change the nature of a metacomputer, we do not concern ourselves with it further.

### 3 Comparison of Systems

We briefly describe several systems that have been recently developed that provide some subset of the requirements listed above. The features of each are summarized in table 1.

#### 3.1 Globus

The goal of the Globus [7] project is to elucidate the requirements for grid-enabled applications and to develop the essential technologies required to meet them. The central element of the Globus system is the Globus Metacomputing Toolkit which provides the low-level mechanisms used to implement higher level services.

Within the Globus metacomputing toolkit, modules define interfaces and provide implementations of those interfaces. The Globus Resource Allocation Manager (GRAM) provides resource management and a standard network-accessible interface to all constituent systems. Communication mechanisms provided by Globus allow an application to use a variety of primitives, such as message passing, remote procedure call, multicast or a distributed shared memory metaphor. These primitives take into account factors such as latency and bandwidth. Users of Globus are supplied with real-time information about the metasytem's status and configuration by means of the unified resource information service. The authentication interface in Globus regulates the security of data and resources; implementations include mechanisms for the validation of user and resource identity, and authorization and protection of data.

Access to resources external to the metacomputer, such as relational databases or file storage systems, is managed by a data access module through CORBA. In addition, the Globus team has defined a universal access mechanism to retrieve information at any layer of abstraction: the metacomputing directory service (MDS) consists of a framework in which data can be represented in diverse distributed computing applications via the lightweight directory access protocol (LDAP).

## 3.2 Condor

The Condor project [13] began in the mid 80's at the University of Wisconsin-Madison and was one of the first platforms to embody the notion of metacomputing. A Condor cluster consists of identical Unix workstations connected to each other by a LAN. The principal idea behind Condor is that not all workstations are busy at all times, so CPU cycles on idle machines can be used for jobs that would otherwise be stalled. The Condor grid contains one machine that acts as a central resource manager responsible for allocating jobs to other idle Condor nodes. All other Condor nodes run two daemon processes, one responsible for negotiating with the central manager to allocate local jobs to the cluster, and the other for advertising to the central manager when the node becomes idle.

It is obvious that Condor is a modest platform when compared with Globus or Legion in terms of the possible size of the substrate grid, the range of applications that may run on Condor, and the generality of the system. It does, however, make for an interesting case study because it shows the limitations of a metacomputer that only provides CPU cycles as a resource and does not allow for heterogenous nodes to be attached to the grid.

## 3.3 Ninf/Netsolve

Both Ninf[18] and NetSolve[4] target computational applications that link to well-known mathematical libraries, such as ScaLAPACK[1]. A programmer interfaces with Ninf and Netsolve using a series of RPC function calls designed to replace functions in existing libraries; it is possible with very little code modification to recompile applications to use these systems. Because the two systems are so similar, a "bridge" has been developed that converts between the systems' internal data formats allowing the two systems to use each other's resources [14, 2]. Both systems have also recently become capable of interfacing with other metacomputing environments, such as Globus and Condor systems, which is the first evidence we have seen of collaboration between metacomputing projects to date. As both Ninf and NetSolve have progressed, they have become increasingly similar, leveraging on each other's features.

Ninf and NetSolve make scheduling decisions based on the dependencies of the input data: any groups of remote calls are analyzed for dependency relationships, and those that may be executed without waiting for another call to finish are immediately sent off to a remote resource. Although this marks a major step in the evolution of the RPC interface, there have as yet been no implementations of automatic code transport; Ninf and NetSolve are still limited to numerical applications.

Scheduling in Ninf and NetSolve is completely transparent to the end user. Although a programmer can define blocks of related function calls, the actual selection of a remote resource and the migration of data is handled by the Ninf Metaserver[15] and the NetSolve Agent[3], which we refer to collectively as *schedulers*. The user provides a problem description to the scheduler, which

	Resources	Security	Replication	Fault Tolerance	Parallel	Legacy
Globus	good	good	good	moderate	moderate	good
Condor	poor	poor	none	good	poor	good
Ninf/Netsolve	moderate	poor	none	moderate	moderate	good
Legion	good	good	good	poor	good	moderate
Seti@Home	none	none	none	good	good	none

Table 1: Recent Metacomputing systems

tries to match as closely as possible the correct resources to the submitted job, based on a database of resource loads and performance. Should a server fail to complete a task, the task is simply rescheduled to a new resource. Newer implementations of Ninf and NetSolve attempt to do resource prediction and task farming[5] to simplify the programming interface and reduce the number of function calls. Tasks aid in recovery from failures and preparation for migration by performing application-specific checkpointing of data, as there are no automatic checkpointing facilities[17].

Because libraries are installed and maintained by local administrators, an unscrupulous user could potentially waste resources by continually submitting useless jobs to remote machines, or intercept sensitive data en route to or from remote locations. Ninf has begun using SSL encryption and certificates, where administrators may create policies deciding what users may use what libraries on what machines[19]. All schedulers are, however, implicitly trusted by each other; any malicious user could try to overload a remote machine, or at least its scheduler, with requests.

### 3.4 Legion

The Legion system [12] provides a set of programming tools, an interactive environment, and an execution environment for applications requiring a meta-computer. The distributable components in a Legion application are objects, generally written in the Mentat [10] dialect of C++. Objects that are marked as “mentat” objects become the equivalent of computational servers and may be bound to a particular site or group of sites. Invocations of methods on these objects are handled by a traversal of the data-dependency graph at runtime, such that several method invocations may be outstanding at a given time, but only the invocations that contain a full set of parameters will be activated. Any non-mentat objects are colocated with the mentat object that depends on them. The mentat language analyzes code prior to compilation to exploit any hidden parallelism not explicitly stated by the programmer.

The Legion project developed a rich object model to provide the metacomputing services we mention above. Because each class in the object model can be used as a base class, site administrators have complete flexibility in redefining the policies used to perform computation to tailor the operation of legion to their own site. For example, a scheduling object for a particular application

can be derived from the default implementation to create a new scheduling policy; this new scheduling policy interacts with a derivable site magistrate that enforces authentication and authorization at both the user level and the object level.

### 3.5 Seti@Home

The Seti@Home[20] project has, to date, achieved incredible success. Total statistics since its launch on 13 May 1999 claim more than 800,000 *years* of CPU time, or greater than  $10^{20}$  floating point operations. Seti@Home, however, takes the inverse approach to the other metacomputing paradigms we have seen so far: individual users/computers actively request a small (250KB) data packet to work on, rather than passively accepting data and/or tasks from a scheduler. Seti@Home is particularly suited to this sort of philosophy since it is “embarrassingly parallel”; none of its data units depend on any other. In addition, Seti@Home supports long times between the distribution of data and the return of results, as well as fault recovery by simply scheduling a particular data packet for multiple users (the particular heuristic is that if 60% of the returned results for a given data packet are identical the results are considered valid).

Although projects like Seti@Home are only one class of distributed applications, other massively parallel projects could potentially take advantage of the huge user base with CPU cycles to spare by finding ways to break their computation into small, mutually exclusive pieces.

## 4 Discussion

It is clear that none of the systems that we have described satisfy all of our requirements for a metacomputing system. The projects that developed these systems had differing goals: Ninf/Netsolve try to solve a specific practical problem facing numerical analysis applications, while Globus is an engineering experiment to determine the best organization of software for metacomputing middleware. Nonetheless, each of these systems has shown practical benefits. For example: Legion has been used to study protein folding problems, Ninf and Netsolve have been used to study cellular microphysiology, Globus has been used to investigate meteorological and quantum mechanical simulations, and Condor has been used to solve mixed integer optimization challenge problems.

Even if a perfectly complete metacomputing system existed, we assert that there would still be significant limitations to what applications could benefit from using a metacomputer. In theory, any parallel application could run on a metacomputer, and proponents of grid computing often state that by harnessing all available idle cycles on a network one can improve a job’s performance[12]. In addition, we claim that the primary purpose of a metacomputer is not actually to increase the *speed* of a particular job, but to provide *access* to distributed resources. Idle cycles alone are not sufficient to improve the performance of a

job; many small, slow computers connected through a grid will generally not improve the performance of an application that might run faster on a single moderately capable sequential computer. We believe that realizing any benefit from a perfect metacomputing platform will be difficult unless the following hold:

- The job consists largely of parallel sub-problems
- The job is relatively fault-tolerant, or can be scheduled in such a way that fault-susceptible parts of the job are run on reliable nodes
- The amount of data transfer and coordination required between compute nodes is small compared to the amount of computation to compensate for the high latency of networks
- The substrate grid is relatively reliable over the lifetime of the application (eg, cell phones do not make good candidate nodes for a grid)
- The total number of jobs present in the metacomputer does not exceed some fraction of the metacomputer's CPU resources or memory
- The metacomputer has a sufficient quantity of resources critical to the job's execution
- The overhead associated with the metacomputing middleware is negligible relative to the application's total computation time

Because of these application requirements, one questions the usefulness of a metacomputer for typical personal or home use in the near future, in contrast to the popularized notion of a metacomputer in [8]. Certainly, a great variety of scientific problems fall into this regime, and one expects that many enterprises have applications that behave in this way. Of the applications that fall into this class, one would expect that a metacomputer could provide special value to jobs that required access to a limited or rare resource attached to a node in the substrate grid, or for jobs that required more resources than could be built economically into a single integrated hardware platform. For example, some scientific visualization applications require moderately fast access to terabytes of data, which is difficult to achieve with conventional disk systems. It would be faster to use secondary storage at remote sites than to use tertiary storage such as DAT tape at the local site, especially when access to the data is fairly random.

An interesting effect of a metacomputer is that it encourages users with less powerful computers to submit jobs consisting of large and complex tasks. If a grid is composed primarily of low-end workstations and desktops but contains a few very high-end MPPs, it is possible that the high-end computers will become flooded with job requests that only they can fulfill. In order for this to be equitable for the providers of high-end computers, a commensurate quantity of embarrassingly parallel applications (i.e., those similar to Seti@Home) should be scheduled in the metacomputer, or each individual job must be strictly larger

than all of the high-end machines. Thus, it may be the case that very large computational problems that cannot efficiently be solved on high-end machines may actually help the load distribution in a metacomputer.

One of the primary problems underlying all metacomputing projects is the lack of theoretical support for the development of applications that use metacomputing platforms. In particular, the models used in parallel algorithm development generally do not hold in the metacomputing environment: networks cease to be reliable and low-latency, memory access is decidedly non-uniform, and the operations that the metacomputing middleware performs are not necessarily constant-time. It is difficult for an engineer to make informed decisions about the use of a particular algorithm when there are few commonly accepted quantitative techniques to explore the algorithm's asymptotic behavior while realistically accounting for the nonuniform communication latencies and error rates. Several researchers [16, 6] have attempted to rectify this situation to varying degrees of success, but no all-encompassing model has been proposed. Valiant [21] notes that this problem also affects the development of applications targeted for MPPs, and that it partially explains why sequential programming still dominates the computer science field despite the fact that it is generally not scalable.

This problem manifests itself in another significant way: without defining an abstract model that maps every parallel algorithm to a metacomputer, one cannot define the primitives that such a machine should provide. Without conceptual agreement between different projects on these primitives, it is much more difficult to develop an implementation-independent "standard" that addresses the challenges of metacomputing.

## 5 Conclusions

In this paper we informally introduced the notion of a metacomputer, listed the difficulties and requirements of a metacomputing system, compared several existing metacomputing systems and examined the current state of metacomputing from a philosophical viewpoint. We have also explored the difficulties faced in building such a system.

Metacomputers have been advertised as the platform that will harness the combined untapped power of personal computers, supercomputers, televisions, and toasters[9] as a single virtual supercomputer. We believe that this particular marketing message is not an accurate reflection of the current state of the art. We have described two areas where theoretical work could help bridge this gap, and observed that even with a perfect system, a metacomputer is not universally useful. Practical experience has shown that metacomputing platforms are enormously useful in answering scientific questions, but the commonly used metaphor of a "power grid of computing" seems strained at best, and perhaps untenable. In addition we have found that the lack of a theoretical model that accurately describes a metacomputer inhibits the design and implementation of metacomputing systems.

## References

- [1] L. S. Blackford, J. Choi, A. Cleary, E. D’Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. ScaLAPACK: a linear algebra library for message-passing computers. In *Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing (Minneapolis, MN, 1997)*, page 15 (electronic), Philadelphia, PA, USA, 1997. Society for Industrial and Applied Mathematics.
- [2] H. Casanova. Enabling collaboration between netsolve and ninf, 1997.
- [3] H. Casanova and J. Dongarra. Netsolve’s network enabled server: Examples and application. *IEEE Computational Science and Engineering*, 5(3):57–67, September 1998.
- [4] Henri Casanova and Jack Dongarra. NetSolve: A network-enabled server for solving computational science problems. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(3):212–223, Fall 1997.
- [5] Henri Casanova, MyungHo Kim, James S. Plank, and Jack Dongarra. Adaptive scheduling for task farming with grid middleware. In *European Conference on Parallel Processing*, pages 30–43, 1999.
- [6] Franck Cappello et al. Hihcohp - toward a realistic communication model for hierarchical hyperclusters of heterogeneous processors (extended abstract).
- [7] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128, Summer 1997.
- [8] I. Foster and C. Kesselman. Computational grids. In *The Grid: Blueprint for a New Computing Infrastructure*, page Chapter 1. Morgan Kaufman Publishers, 1998.
- [9] A. Grimshaw and W. Wulf. Legion — a view from 50,000 feet. In *5th Intl. Symp. on High Performance Distributed Computing*, August 1996.
- [10] The Mentat Group. Mentat 2.5 programming language reference manual. Technical report, University of Virginia, Charlottesville, VA, 1995.
- [11] <http://archive.ncsa.uiuc.edu/Cyberia/MetaComp/MetaHome.html>. Meta-computing website.
- [12] M. Lewis and A. Grimshaw. The core legion object model. In *5th IEEE Symposium on High Performance Distributed Computing*, pages 562–571. IEEE Computer Society Press, 1996.

- [13] M. J. Litzkow, M. Livny, and M. W. Mutka. Condor - A hunter of idle workstations. In *Proceedings of the 8th International Conference on Distributed Computing Systems (ICDCS)*, pages 104–111, Washington, DC, 1988. IEEE Computer Society.
- [14] H. Najada, S. Matsuoka, and S. Sekiguchi. Bridging ninf and netsolve, 1997.
- [15] Hidemoto Nakada, Hiromitsu Takagi, Satoshi Matsuoka, Umpei Nagashima, Mitsuhisa Sato, and Satoshi Sekiguchi. Utilizing the metaserver architecture in the ninf global computing system. In *HPCN Europe*, pages 607–616, 1998.
- [16] J. Nieplocha and R. J. Harrison. Shared memory NUMA programming on I-WAY. In *Proc. of the Fifth IEEE Int'l Symp. on High Performance Distributed Computing (HPDC-5)*, 1996.
- [17] James S. Plank, Henri Casanova, Micah Beck, and Jack Dongarra. Deploying fault-tolerance and task migration with netsolve. In *PARA*, pages 418–432, 1998.
- [18] Mitsuhisa Sato, Hidemoto Nakada, Satoshi Sekiguchi, Satoshi Matsuoka, Umpei Nagashima, and Hiromitsu Takagi. Ninf: A network based information library for global world-wide computing infrastructure. In *HPCN Europe*, pages 491–502, 1997.
- [19] Satoshi Matsuoka; Hidemoto Nakada; Mitsuhisa Sato; Satoshi Sekiguchi. Design issues of network enabled server systems for the grid. In *Grid Computing - GRID 2000, Springer-Verlag, LNCS 1971*, pages 4–17, 2000.
- [20] SETI. Seti@home: The search for extraterrestrial intelligence. Technical report, University of California at Berkeley, Space Sciences Laboratory, University of California at Berkeley. URL: <http://setiathome.ssl.berkeley.edu/>, 1999.
- [21] Leslie G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, 1990.