Proc. Third Int. Conference on Parallel Processing and Applied Mathematics, PPAM99,
Kazimierz Donly, Poland, September 1999, pp. 17-31

Metacomputing: Parallel Computation Over the Internet

Patrick H. Fry and Boleslaw K. Szymanski
Department of Computer Science
Rensselaer Polytechnic Institute
Troy, NY 12180-3590, USA
{fryp, szymansk}@cs.rpi.edu

Abstract

This paper describes metasystems designed for parallel computing over the Internet. We classify such systems into two categories and describe in detail basic features of four of them. We present also our framework for distributed parallel computing of computationally intensive applications using idle cycles and the Internet as the interconnection network. We applied this framework to computing a distribution of twin primes and Brun's Constant and report the most recent results here. We outline also how our framework can be extended to allow participation of mobile computers, therefore supporting *nomadic computations*.

1 Introduction

Up until about 10 years ago, high performance computing relied almost exclusively on supercomputers and massively parallel machines. However, the exponentially increasing processing power of relatively inexpensive PC's and workstations has made Network of Workstation (NOWs) a feasible and highly economical platform for high performance parallel computing [1]. Libraries, like PVM [21] and MPI [18], are now commonly used by the HPC community for cluster-based computing.

Another type of parallel computing that has recently become popular is web-based metacomputing. Metacomputing systems, also referred to as metasystems, are designed to use the idle CPU cycles of machines using the Internet as the interconnection network. While cluster-based systems may have tens or hundreds of nodes, metasystems are designed to support hundreds, thousands, or even millions of computers. Cluster-based systems are used for parallel execution of applications on a singly administered network (e.g., educational institution or corporation). Metasystems target long running applications of universal interest, like cryptography, mathematics and computational science [23]. Because of the high latency and low end-to-end throughput of the Internet, metasystems are best suited for parallel applications in which each subcomputation needs only a limited or constant volume of data to initiate and/or synchronize execution and reports a constant amount of data as a result. For example, the results might be binary values for searches or a few numbers for factoring, enumerations and computation of moments of data

distributions. Metasystems face many challenges not normally present in a parallel execution environment. Below, we list the most important of these challenges.

Scalability: Metasystems are usually designed to support hundreds, thousands, or even millions of machines. These machines are interconnected using the Internet which is high latency and has low end-to-end throughput.

Ease of Use: In order to achieve participation on the order of thousands or millions of nodes, it must be simple for "Joe User" to add their machine to the computation. Complex installation procedures or configurations are unacceptable. Ease of application development is also important. The metasystem should either support existing parallel languages or provide simple functions for basic operations as well as use object inheritance to hide the gory details of the implementation.

Execution time: Metasystems are used to execute programs which may take months or even years to finish. Such a long run requires frequent state changes or checkpointing to ensure minimum loss of work in the event of a system failure (e.g., power outage or server shutdown).

Adaptive Parallelism: The set of participating machines will grow and shrink during the computation. The metasystem must be able to gather information about its current state dynamically and *adapt* its behavior appropriately at any time during execution, not just at its startup phase. The system must be able to identify a node failure and reassign the work to another participant.

Heterogeneity: In order to get as many participating nodes as possible, most metasystems have been designed to run on multiple architectures. For this reason, most metacomputing environments are Java based.

Security: Unlike cluster-based computing, there is no single system administrator to manage all participating nodes in a metasystem. Users of the participating machines need guarantees that the metacomputing system will not make unauthorized changes or accesses to their machines (e.g., file and resource control). Issues include authorization, authentication, and encryption. Running the metasystem client as a Java applet in a web browser helps alleviate some of these concerns.

The remainder of this article is organized as follows. In Section 2, we provide a short description of four emerging metasystems: Legion [15], Globus [12], Charlotte [3], and Javelin [6]. Section 3 describes a distributed system we constructed for enumeration of twin primes and calculation of Brun's constant using the farmer–worker paradigm [13]. In Section 4 we outline an extension of this system which will support participation of mobile computers in metasystem computations by allowing *disconnected–mode* operation. Section 5 provides a summary and conclusions.

2 Metasystems

In this section we present four metasystems: Legion [15], Globus [12], Charlotte [3], and Javelin [6]. At this point there are two main types of metasystems being developed. The first, at least

Problem	Tools Available
Writing parallel applications	PVM, parallel C++, Fortran wrappers
Multiple separate file systems	Federated file system for transparent file access
Heterogeneous resources	Automatic scheduling, binary selection and migration, application specific scheduling tools
Multiple resource owners	Owner control of resource consumption, detailed resource consumption accounting
Debugging parallel programs difficult	Post-mortem playback using off-the-shelf debuggers
Host/network failures	Automatic system reconfiguration and limited application fault tolerance

Table 1: Common problems facing metasystems and Legion's proposed solutions [15].

historically, are those that rely on extensions to existing parallel programming libraries and cluster management systems like PVM [21], MPI [18], CC++ [5], LoadLeveler, and Condor [8]. Legion and Globus fall in this category. One major difficulty for this type of system is maintaining current and correct binaries of the application codes. Each supported computer architecture needs its own set of binaries for each parallel application (and the underlying metasystem). Access control is another issue. Most of these systems require user access to each participating machine.

Charlotte and Javelin fall in the second category: Java based metasystems. Up until recently, Java Virtual Machine (JVM) performance was a serious hindrance for computation intensive applications. However, recent improvements in the execution environment for Java applets (e.g., just-in-time (JIT) and dynamic compilation) allow for more efficient execution. The Java applet environment inside a web browser provides some "no extra cost" features which are *extremely* useful for metasystems, like automatic source dissemination, strict security, and support from all modern mainstream operating systems.

There are some restrictions placed on Java applets which limit an environment for metasystems. Consider network communication. A Java applet, by default, is only allowed to communicate with the HTTP server from which it was downloaded. Direct peer-to-peer communication is not possible. This type of indirect communication must be passed through the server which can become a serious performance bottleneck. One option is to run the metasystem task as a Java *application* instead of running it as an applet, but then some other useful security restrictions, such as local file access, are lifted.

Section 2.5 describes some other metasystems which may be of interest. All are similar to one or more of the four systems described below. For brevity of space we do not discuss them in detail, but provide short descriptions and references for interested readers.

2.1 Legion

Legion [15], begun in 1993, is a metasystem software project at the University of Virginia. Legion is an ambitious project whose goal is nothing less than creating a "worldwide virtual computer" providing location transparent access to resources available in the system. Legion is still in the early stages of development. The designers of Legion have the following objectives:

Site autonomy: Legion will consist of resources spread across spheres of system control, each system controlling the use of its own resources by the Legion system (e.g., how often and when a resource is available to Legion and what kind of access is allowed).

Extensible core: The core system must allow users to "construct their own mechanisms and policies to meet specific needs."

Scalable architecture: Must be scalable to millions of hosts - no centralized structures and completely distributed management.

Easy to use, seamless computational environment: Compilers and run-time facilities should manage the environment for the user as much as possible.

High performance via parallelism: Support for easy to use task and data parallelism.

Single, persistent name space: A single name space for file and data access across the entire system.

Security for users and resource owners: Provide mechanisms for users to manage their own security needs and do not weaken existing security mechanisms in the host operating system.

Management and exploitation of resource heterogeneity: Legion must support interoperability between heterogeneous hardware and software components. Scheduling decisions and policies should be customized to each type of system.

Multiple language support and interoperability: Support for legacy codes and heterogeneous language application components.

Fault tolerance: In such a large system, it is obvious that several participating hosts may be temporarily down or disconnected. Legion must dynamically reconfigure itself in the presence of these failures with minimal loss of work.

The first prototype of Legion, the Campus Wide Virtual Computer (CWVC), was released in 1995. It consists of more than 100 workstations and an 18-processor IBM SP2. No other metacomputing project is as ambitious as Legion. The CWVC at present does not realize all the objectives for Legion. This prototype has tools for integrating parallel applications written in PVM, parallel C++, and FORTRAN wrappers. At present, the core Legion object model, incorporating mechanisms for security, fault tolerance, application-directed scheduling, autonomy, and scalable binding is being built. These services are not provided in the initial prototype. It may be some time before Legion becomes a reality, but the design and portions of the system may be useful for other metacomputing projects. Table 1 lists some common issues facing metasystem designs and how the designers of Legion propose to address them.

2.1.1 Programming Model

Legion is being designed to provide support for existing parallel programming languages like PVM and CC++. The goal of Legion's authors is to build the underlying metasystem so it is easy to

Service	Name	Description
Resource Management	GRAM	Resource allocation and process management
Communication	Nexus	Unicast and multicast communication services
Security	GSI	Authentication and related security services
Information	MDS	Distributed access to structure and state information
Health and status	HBM	Monitoring of health and status of system components
Remote data access	GASS	Remote access to data via sequential and parallel interfaces
Executable management	GEM	Construction, caching, and location of executables

Table 2: Core Globus services [12].

"plug in" existing parallel languages by writing libraries which translate parallel procedures in the parallel language into calls to the Legion core system. As new parallel languages are created, they can be supported by Legion with limited effort. System support for "off-the-shelf" debuggers like `dbx` is also planned.

Legion will have a single name space for resources, including application objects, files, and hardware resources. This single naming scheme will make it easier for programmers to identify and locate these items.

2.1.2 System Architecture

Legion is still in the early stages of development, so there are still many architecture issues to work out. Legion will not require "root" access to participating systems. However, some form of user access is required for the system daemons to run in. To support heterogeneous resources, Legion's system philosophy is to build a translation layer which allows Legion built applications to access resources in a uniform manner. For example, Legion has a federated file system. Legion applications requesting access to files will send the request to a Legion server (may be a daemon running on the same machine as the caller) which will translate this call into a native call for the underlying file system in which the file is stored. Legion also provides automatic binary selection. For support of heterogeneous resources, Legion must provide an application binary for each system architecture under which the application will run.

2.2 Globus

The Globus Project [12] is a research program for developing "large-scale high performance distributed computing environments, or *computational grids* that provide dependable, consistent, and pervasive access to high-end computational resources." The Globus authors are interested in creating middleware to support dynamic resource allocation, heterogeneous and dynamic computation environments, and process level security. The primary goal of the Globus project is to combine clusters of high performance systems such as the supercomputers at NCSA with those at Argonne National Labs. For this purpose, GUSTO (Globus Ubiquitous Supercomputing Testbed), a grid prototype has been created for which Globus provides low level services, like a communication within a grid, resource identification, and authentication.

The middleware portion of Globus is called the Globus Metacomputing Toolkit which provides services for security, resource management, and communication. Table 2 lists the core services provided. Globus provides *translucent* interfaces to the heterogeneous resources. In other words, the interfaces manage the resources in a way which provides detailed information about the resource to the applications if desired.

2.2.1 Programming Model

Like Legion, Globus is designed to support existing parallel programming languages. Currently listed as supported or under development for support are: MPI, CC++, HPC++, PAWS, CORBA, and RPC. For MPI, a grid enabled MPI was created which uses Nexus for communication, GRAM for resource allocation, and GSI for authentication. Users can write standard MPI programs which function in a variety of parallel environments, like clustered heterogeneous workstations or MPP machines or a mix of both.

2.2.2 System Architecture

As with the programming model, Globus "grid enables" existing system architectures. For resource management, GRAM provides an interface between Globus and local resource managers, among which the following are currently supported: Network Queuing Environment (NQE), EASY-LL, LSF, LoadLeveler, Condor [8], and a "fork" daemon.

For security, the Globus toolkit currently focuses only on authentication with the thought that the other security issues, such as authorization and privacy will eventually be built on top of the authentication layer. This is an especially difficult issue because the the system must support *N-way security*, or any-to-any security, in contrast with traditional client-server application authentication.

2.3 Charlotte

Charlotte [3] is a metasystem based solely on Java without any native code. Clients are run *entirely* as Java applets within a web browser.

- A user can execute a parallel program on a machine he/she does not have an account on.
- Neither a shared file system nor a copy of the program on the local file system is required.
- Local hardware is protected from programs written by "strangers".
- Any machine on the Web can join or leave any running computation, thus enabling dynamic resource utilization.

The use of Java applets makes many of the difficulties present in Legion and Globus disappear. Portability, heterogeneity, and security are no longer a concern. However, Charlotte also suffers from the restrictions of the applet environment. Currently, Java performance is *at least* 40% slower than machine code applications written in C, C++ or FORTRAN. Communication is

another issue. Direct client-to-client communication is not currently supported, so the communicating clients must use the server as a "pass through" intermediary. Such indirection can cause unacceptable delays for communication intensive parallel applications. Another issue is idle usage management. Similar to a screen saver, it is desirable for the metasystem to use processor cycles when the machine is idle but the metasystem application must immediately shutdown, not just suspend, when other activity ensues on the machine. This type of automatic resource control is not possible from within an applet.

Another issue is file access. The applet environment is not allowed access to the local file system. Hence, there is no possibility of directly storing temporaries in files or accessing data files. Any such activity must be done through the server. An example where local file access would be useful is for our twin primes application discussed in Section 3. We currently use a 3MB file containing all prime numbers up to the 10^8. In an applet environment, each time the applet is started this file would have to be either downloaded from the web server or the data would have to be regenerated by the client.

2.3.1 Programming Model

Unlike Legion and Globus, Charlotte provides its own parallel programming model consisting of a set of classes and interfaces using a distributed shared memory paradigm. A Charlotte program consists of alternating sequential and parallel steps. The sequential step is programmed in a standard sequential Java code. The computationally intensive parts are done in parallel steps in one or more *routines*. A routine is analogous to a thread which can execute remotely. Parallel steps start and stop with the keywords `parBegin` and `parEnd`. Figure 1 shows a sample code snippet for matrix multiplication using a class which inherits from Charlotte's `Droutine`.

2.3.2 System Architecture

Charlotte uses *eager scheduling* for work allocation, assigning a job repeatedly to multiple workers until this job is completed by at least one of them. This allows new workers to join the computation at any time, even in the middle of a parallel step. Slower machines are bypassed by faster ones, thus providing fault tolerance. Slow machines also ask for work less frequently, thus providing automatic load balancing. However, this scheme for work allocation does not scale well. A Charlotte "server" process registers itself by placing a URL on a web page. Users wising to add their machines to the computation click on this URL from the Java-capable web browser. The code is then loaded to the machine and it joins the ongoing computation. In the future, Charlotte will be extended to automate this process by using "manager lookup services."

2.4 Javelin

Like Charlotte, Javelin's [6] workers, called *hosts*, are executed entirely as a Java applet. Javelin focuses on ease of use not only from the host's perspective, but also from the machine which is submitting work to the system, called *clients*. However, many difficulties are not addressed at all or are insufficiently addressed in Javelin, for example this system does not support any form of fault tolerance. Any work replication or rescheduling must be written by the user into the application. In addition, Javelin provides no inherent parallel library support. The system

```
public class MatrixMult extends Droutine{
  public static int Size = 500;
  public Dfloat a[][] = new Dfloat[Size][Size];
  public Dfloat b[][] = new Dfloat[Size][Size];
  public Dfloat c[][] = new Dfloat[Size][Size];

  public MatrixMult() {
     ...
  }

  public void drun(int numTasks, int id) {
    int sum;
    for(int i=0; i<Size; i++) {
      sum = 0;
      for(int j = 0; j<Size; j++)
        sum += a[id][j].get()*b[j][i].get();
      c[id][i].set(sum);
    }
  }

  public void run() {
    ...
    parBegin();
    addDroutine(this, Size);
    parEnd();
    ...
  }
}
```

Figure 1: Matrix multiplication in Charlotte [3].

is solely responsible for assigning work to hosts and retrieving the results. There is no message passing or shared memory model.

2.4.1 System Architecture

The Javelin model has three components: clients, hosts, and brokers. Clients submit *tasks*, or a portion of a computation that is looking for a place to be executed. The clients and hosts register with the *broker* who assigns the clients to specific hosts. A user adds their machine to the metasystem as a host by simply pointing their web browser to the URL of a broker. The HTML source of the broker's URL has the user's machine download and execute an applet which "listens" for work from the broker.

The role of client and host may be played by the same computer and applet. For example, a

host may retrieve a task from a client and break the task into subtasks (e.g., ray tracing). These subtasks may either be executed by that host or sent to the broker for execution by some other host.

Load Balancing Each host and client has a *dequeue* (double-ended queue) of tasks. The host takes work from the back of the queue (like a stack model). If there is no work in the queue, the host asks the broker for more tasks. A broker which is looking for work from other hosts will examine the dequeue of its hosts. If there is work in one of them, it will take a task from the front of the dequeue (like a queue). This model is similar to the *work stealing* architecture used in Cilk [7].

Javelin's brokers currently use a very simple load balancing scheme. Brokers can communicate with each other and transfer client work between them based on their host loads. For example, a broker who has more work than hosts to give it out to might transfer not yet assigned work to less loaded brokers.

2.5 Other Metasystems

Other Java based metasystems include Popcorn [22], ATLAS [2], ParaWeb [4], and DOGMA [9]. Popcorn is applet based and uses economic methods for distribution of work. A Popcorn programmer divides the application into subcomputations, or *computelets*. The *Buyer* computer submits these computelets to a *market* web server. *Seller* computers download these computelets from the market and execute them. Sellers earn *popcoins* based on the number of JOPs, Java Operations, performed. The motivation behind Popcorn is to provide an incentive for participation in the system.

Similar to Javelin, Atlas uses the Cilk [7] programming model. Atlas also has a URL based file system implemented as a native library (non Java). The Atlas system architecture has *clients*, *managers*, and *compute servers*. Clients submit work to the system. Managers maintain a directory of their compute servers. The compute servers perform the parallel computation. System scalability is achieved through a hierarchy of managers. Managers may steal work from its manager and other peer managers. Atlas no longer appears to be under active research and development.

ParaWeb has two separate implementations. The first, Java Parallel Class Library, is a set of Java classes which implements message passing and remote thread execution. The second, Java Parallel Runtime System, is a modified JVM which provides a shared memory model and, again, remote thread execution. ParaWeb no longer appears to be under active research and development.

DOGMA is a recent addition to Java based metasystems. Like ParaWeb, DOGMA has two forms of execution, one which runs entirely within a web browser and another which runs as a Java application. However, unlike ParaWeb, the Java application does not run on a modified JVM, it must run as an application for peer-to-peer communication. DOGMA also provides a screen saver version of the application. The primary supported language is MPIJ, a Java based MPI created by this group.

Many metasystems use existing or slightly modified parallel programming methods commonly used in cluster computing. JPVM [11] and MPIJ [9] provide the popular PVM and MPI message passing environments for Java applications without using any native code. Condor [8] is an application scheduling environment for execution of sequential batch programs in a cluster of

heterogeneous resources. Cilk [7] provides multithreaded parallel extensions to ANSI C. Cilk's "work stealing" method of load balancing is used by some Java based metasystems including Javelin and Atlas. The *Virtual BSP Computer* library [19] supports BSP computations over a network of non-dedicated workstations. This system migrates processes (using Condor) from workstations which become unavailable and supports scalability and tightly synchronized parallel computations.

3 Twin Primes Distribution

This section outlines our experiences in applying a distributed heterogeneous environment to computing Brun's Constant and twin primes distribution over a range of integers that is two orders of magnitude larger than any previously traversed [13, 14, 20]. Although we used the well known *farmer – worker* paradigm, this specific application to twin primes computation has led to several novel contributions. We have a relay agent which creates a hierarchical communication structure. Our server, the farmer, is at the top. Intermediate nodes are relays and leaf nodes are clients, or workers. The relay allows nodes behind a firewall to contribute idle cycles. The *SCATTERS* [10] tool is used to start computation on a group of idle nodes at once. We have added new levels of reliability in the farmer to support the years necessary to complete this computation. We have also developed efficient ways to represent prime numbers and efficiently compute a sum of the large number of inverses with very high precision needed for evaluation of Brun's Constant. The average performance of our application is approximately 5 tera-instructions (5 trillion instructions) per week using several hundreds of computers.

In the farmer – worker paradigm, the farmer is responsible for assigning work to be processed by its workers and collecting the results from the workers. In its most general form, the farmer – worker method makes no assumptions about the number of workers or the reliability of the connection between worker and farmer. Our design goals were to use as many processors as possible, and to enable a computation to proceed for many months. We focus on:

Reliability: We allow workers to drop off and join the computation at will. The farmer can fail and even checkpoint files can be lost without stopping progress of the computation.

Portability: We minimize memory, computation and system requirements of the worker and restrict the needed tools to basic standard (e.g., using TCP/IP for communication, ANSI C, and standardizing endian and data format for Unix and Linux, ensuring source code portability to Windows 95/98/NT). Minimal host requirements open the problem up to a greater number of potential contributors.

Similar goals and means were used by the DESCHALL project which cracked the code in the RSA DES Challenge [24].

Our framework is best suited to parallel applications with modest synchronization requirements. The ratio of computation time to amount of data needed to define a portion of the problem domain space should be high. The amount of data returned to the farmer should be small as well.

Our first application of this system is computing twin primes distribution. This application was chosen both because it is a good representative of a class of applications which lends itself well to the farmer – worker paradigm and its importance to number theory. We have already collected

data for an interval an order of magnitude larger than any previously computed and will have another order of magnitude on completion of the project.

A prime number is a positive integer that is evenly divisible by exactly two positive integers: itself and one. Two subsequent odd numbers that are both primes are called twin primes or twins for short. $(3, 5)$, $(5, 7)$, $(11, 13)$, and $(41, 43)$ are all twins. The Twin Primes Conjecture proposes that twin primes occur infinitely often [16]. Although it is still not known if the set of twins is infinite, Brun proved that the sum of the reciprocals

$$B = (\frac{1}{3} + \frac{1}{5}) + (\frac{1}{5} + \frac{1}{7}) + (\frac{1}{11} + \frac{1}{13}) + (\frac{1}{17} + \frac{1}{19}) + \cdots$$

is convergent; unlike the divergent sum of the reciprocals of all individual primes. B is known as Brun's Constant. This value has been estimated by summing the inverses of twin primes over ever increasing ranges [20] with the current upper limit of 10^{14}. For this application, our goal is to refine this estimate by enumerating twin primes two orders of magnitude further, up to 10^{16}. In addition to counting the number of twin primes and computing the sum of their inverses, we also find the maximum distance between twins and the location of these gaps.

This is a formidable computation which would take over 100 years on a single workstation. Parallelism and careful algorithm optimization are necessary to obtain results in more timely manner. The computation is highly parallel and there is no need for synchronization or communication between processes except for providing initial data and gathering results. We have shown that a distributed environment in which idle processor time is utilized to run this computation is not only feasible but is also a highly efficient and economical means of obtaining the results.

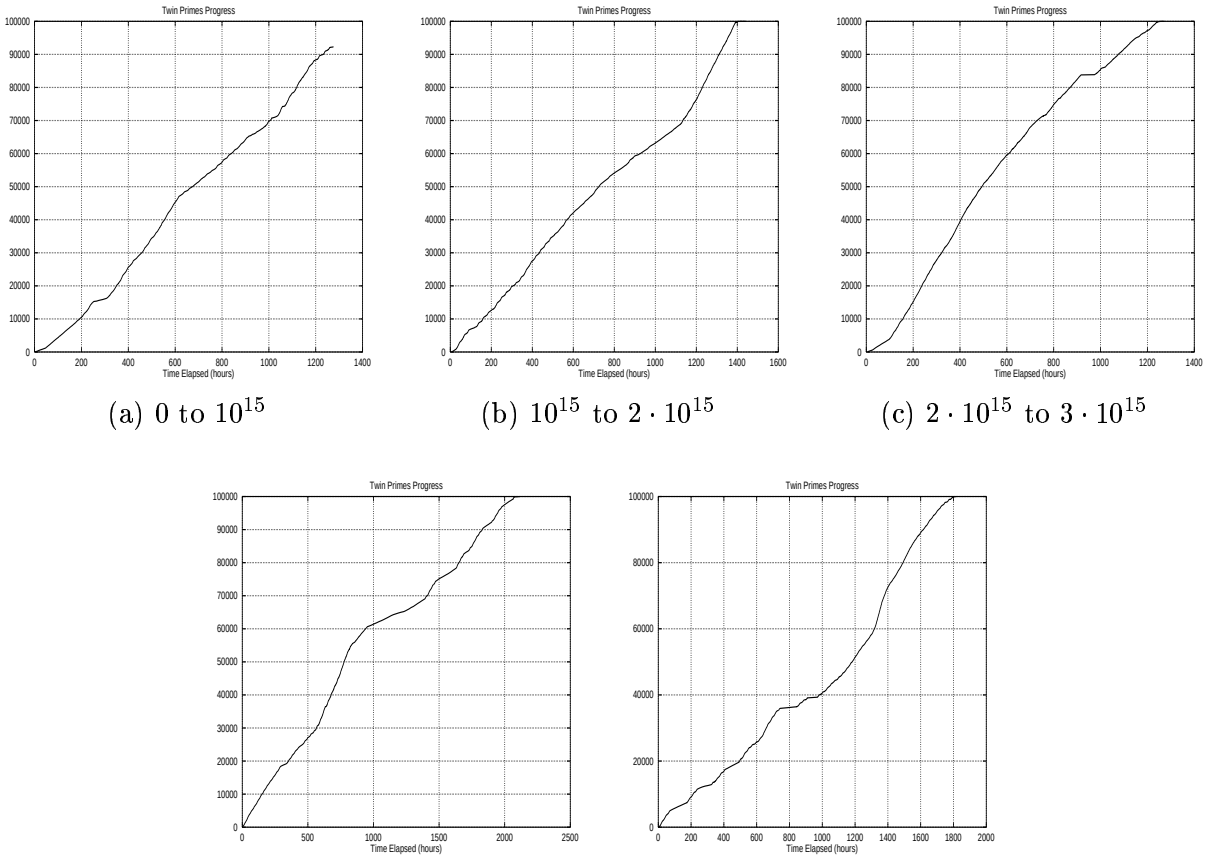
Figure 2 shows the progress of our computation for the first year. Each chart shows the performance for an interval of 10^{15}. The X-axis is the running time of the farmer in hours. The Y-axis is the intervals (10^{10} each) completed. An interval takes 20 to 180 minutes depending on its location (in the range of numbers being searched) and computer speed. Our system has averaged approximately 70 intervals per hour and has found and processed over *5 trillion* twins.

4 Nomadic Metacomputing

The increasing numbers and processing power of laptops and other portable processing devices is as yet an untapped resource for metasystems. A mobile resource should be able to connect to the system from one domain (e.g., at the user's office), retrieve work to be processed, disconnect from the network, and reconnect at a later time from a different domain (e.g., user's home via a modem) at which time it returns results and collects more work. The following are issues unique to nomadic computer support in metasystems and are not addressed in current metasystems under development.

4.1 Identification

Network or geographic specific identifiers cannot be used for mobile resources. The system must provide an identification service which provides meaningful identifiers in a location independent manner. Our twin primes system uses unique worker id's for each independent session with the farmer. The twin primes farmer uses a combination of IP address and a farmer generated



(a) 0 to 10^{15} (b) 10^{15} to $2 \cdot 10^{15}$ (c) $2 \cdot 10^{15}$ to $3 \cdot 10^{15}$

(d) $3 \cdot 10^{15}$ to $4 \cdot 10^{15}$ (e) $4 \cdot 10^{15}$ to $5 \cdot 10^{15}$

Figure 2: Progress of twin primes computation (time in hours vs. number of intervals, 1 interval $= 10^{10}$).

unique integer. This allows multiple workers per IP address, for multiprocessor workstations or supercomputers. However, such identifier is still dependent on the network specific IP address and, of course it, must be replaced by an address-independent identifier.

4.2 Temporary storage

The metacomputing system running on a mobile machine must support execution even when disconnected from the rest of the system. This includes both temporary storage of results from its portion of the computation and storage of the work which was assigned while connected to the system. Our twin primes farmer stores data in a checkpoint file which can be used to resume computation with minimal loss of work in the event of a farmer or system failure.

4.3 Delayed response

The rest of the metacomputing system must be robust enough to allow for significant delays in the return of results for work assigned to mobile resources. Our twin primes system uses time

limits on workers to determine failures in workers. Further study is required to determine efficient failure detection schemes for workers which may be disconnected for extended periods of time before responding.

4.4 Docking

Mobile computers can be assigned a *docking station* which is responsible for storing any information that is addressed to the mobile computer while the mobile computer is unavailable. Docking stations are used in AgentJava [17], a system for mobile agents. Docking plays a similar role to our twin prime computation relay; they both are a proxy for a client. The relay is responsible for forwarding messages between workers and the farmer and appears to the farmer as a representative of each of the relay's supported workers. In addition to this function, in a mobile environment, such a proxy needs also a temporary storage to queue farmer-to-worker messages.

5 Conclusion

Web-based computing, or metacomputing, has recently become a popular method for high performance parallel computing. Metasystems are designed to use the idle CPU cycles of machines using the Internet as the interconnection network. While cluster-based systems may have tens or hundreds of nodes, metasystems are designed to support hundreds, thousands, or even millions. Metasystem designers face many issues which were previously of little or no concern for other types of parallel systems. Most of these issues stem from the characteristics of the Internet: a vast amount of geographically dispersed, autonomous, heterogeneous resources connected through an insecure, unreliable, high latency, low end-to-end throughput network. In contrast, most cluster systems consist of dedicated homogeneous machines connected through a high speed network with one administrator or team of administrators. Current metasystem research issues include scalability, ease of participation and programming, idle-resource usage, load balancing, fault tolerance, system information, heterogeneity, authentication, and encryption.

In this paper we have presented four popular metasystems: Legion, Globus, Charlotte, and Javelin. Those metasystems represent the current state-of-the-art in metacomputing. There are two major "camps" in metasystem design. The first camp builds its metasystem by extending existing cluster-based parallel environments. Legion and Globus fall in this category. For example, Legion is being designed to support parallel languages including CC++ and PVM. Globus is designed to interconnect existing parallel supercomputers and clusters to create a global supercomputer. Each "local" parallel computing resource can micro-manage its own resources. Globus allows *work* to be transferred to other clusters through its translations mechanisms.

The advent of Java is the impetus for the second camp, represented by Charlotte and Javelin, which use Java's inherent support for computing over the Internet. These types of metasystems are created by adding parallel computing Java classes or modifying Java's Virtual Machine. A user contributes machine cycles to the metasystem by running the metasystem application as an applet in a web browser. This approach guarantees that the "untrusted" program cannot gain access to confidential data on the machine. The challenge for the metasystem designer is to build an efficient parallel execution environment under these restrictions.

In Section 3 we describe a distributed parallel execution environment we have created. Our first application using this system calculates twin primes distribution and Brun's constant. The application is currently using several hundred geographically dispersed heterogeneous computers.

The search for more ways to have computers participate in metasystems is an active research field. We have been investigating ways in which mobile computers can contribute cycles to metasystems. In Section 4, we show how our twin primes system can be extended to create a metasystem which supports mobile resources. Mobile computing is a heavily researched area, but mobile metacomputing, or *nomadic metacomputing*, is still in its initial stages. We plan to build a nomadic metasystem which enhances an existing Java based metasystem, like Charlotte or Javelin, with distributed agent technology, like AgentJava.

References

[1] T. E. Anderson, D. E. Culler, and D. A. Patterson. A case for NOW (Network Of Workstations). *IEEE Micro*, 15(1):54–64, February 1995.

[2] J. Eric Baldeschwieler, Robert D. Blumofe, and Eric A. Brewer. ATLAS: An infrastructure for global computing. In *Proceedings of the Seventh ACM SIGOPS European Workshop on System Support for Worldwide Applications*, 1996.

[3] Arash Baratloo, Mehmet Karaul, Zvi Kedem, and Peter Wyckoff. Charlotte: Metacomputing on the Web. To appear in Future Generation Computer Systems.

[4] T. Brecht, H. Sandhu, M. Shan, and J. Talbot. ParaWeb: Towards world-wide supercomputing. In *Proceedings of the Seventh ACM SIGOPS European Workshop on System Support for Worldwide Applications*, 1996.

[5] Compositional C++ (CC++). http://globus.isi.edu/ccpp.

[6] Bernd Oliver Christiansen, Peter Cappello, Mihai F. Ionescu, Michael O. Neary, Klaus E. Schauser, and Daniel Wu. Javelin: Internet-based parallel computing using Java. *Concurrency: Practice and Experience*, 9(11):1139–60, 1997.

[7] The Cilk project. http://supertech.lcs.mit.edu/cilk.

[8] Condor: High throughput computing. http://www.cs.wisc.edu/condor.

[9] DOGMA: Distributed object group metacomputing architecture. http://zodiac.cs.byu.edu/DOGMA.

[10] Garance A. Drosehn. SCATTERS - a Simple Cool Admin Tool To Everywhere Run Something. http://www.rpi.edu/~drosehn/Projects/SCATTERS.html.

[11] Adam J. Ferrari. JPVM: Network parallel computing in Java. Technical Report CS-97-29, Department of Computer Science, University of Virginia, 1997.

[12] Ian Foster and Carl Kesselman. The Globus project: A status report. In *Proceedings of the Seventh Heterogeneous Computing Workshop*, pages 4–18, 1998.

[13] Patrick H. Fry, Jeffrey Nesheiwat, and Boleslaw K. Szymanski. Experiences with distributed computation of twin primes distribution. To appear in *Parallel and Distributed Computing Practices*.

[14] Patrick H. Fry, Jeffrey Nesheiwat, and Boleslaw K. Szymanski. Computing twin primes and Brun's constant: A distributed approach. In *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing*, pages 42–49, Chicago, IL, July 1998. IEEE Computer Society.

[15] Andrew S. Grimshaw and Wm. A. Wulf. The Legion vision of a worldwide virtual computer. *Communications of the ACM*, 40(1):39–45, January 1997.

[16] Richard K. Guy. *Unsolved Problems in Number Theory*. Springer–Verlag, 2 edition, 1994.

[17] David Kotz, Robert Gray, Saurab Nog, Daniela Rus, Sumit Chawla, and George Cybenko. Agent Tcl: Targeting the needs of mobile computers. *IEEE Internet Computing*, 1(4):58–67, July/August 1997.

[18] Message Passing Interface (MPI). http://www.mcs.anl.gov/mpi.

[19] Mohan Nibhanupudi and Boleslaw K. Szymanski. Runtime support for virtual BSP computer. In *Proc. Workshops at 12th Intern. Parallel Processing Symposium*. Springer Verlag, 1998.

[20] Thomas R. Nicely. Enumeration to 1e14 of the twin primes and Brun's constant. *Virginia Journal of Science*, 46(3):195–204, March 1996.

[21] Parallel Virtual Machine (PVM). http://www.epm.ornl.gov/pvm.

[22] Ori Regev and Noam Nisan. The POPCORN market – an online market for computational resources. In *Proceedings of The First International Conference on Information and Computational Resources*, 1998.

[23] Luis Moura Silva, Paulo Martins, and Joao Gabriel Silva. Merging web-based with cluster-based computing. In *Computing in Object-Oriented Parallel Environments*, volume 1505 of *Lecture Notes in Computer Science*, pages 119–126. Springer, 98.

[24] Rocke Verser. The $10,000 DES challenge. http://www.frii.com/~rcv/deschall.htm.