The Nimrod Computational Workbench: A Case Study in Desktop Metacomputing

David Abramson† Ian Foster‡ Jon Giddy§ Andrew Lewis#
Rok Sosič† Robert Sutherst£ Neil White£

†School of Computing and Information Technology
Griffith University
Brisbane, Qld. 4111
Australia
{D.Abramson,R.Sosic}@cit.gu.edu.au

‡Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, IL 60439
U.S.A.
foster@mcs.anl.gov

§Co-operative Research Centre for Distributed Systems Technology
Level 7, Gehrmann Laboratories
University of Queensland
St. Lucia, Qld.
Australia
jon@dstc.edu.au

Queensland Parallel Supercomputing Facility
Griffith University
Brisbane, Qld. 4111
Australia
andrew@qpsf.edu.au

£Co-operative Research Centre for Tropical Pest Management
Level 5, Gehrmann Laboratories,
University of Queensland
St. Lucia, Qld.
Australia
n.white@ctpm.uq.edu.au

The Nimrod Computational Workbench: A Case Study in Desktop Metacomputing

Abstract

The coordinated use of geographically distributed computers, or *metacomputing*, can in principle provide more accessible and cost-effective supercomputing than do conventional high-performance systems. However, we lack evidence that metacomputing systems can be made easily usable or that large numbers of applications are able to exploit metacomputing resources. In this article, we present work that addresses both these concerns. The basis for this work is a system called Nimrod that provides a desktop problem-solving environment for parametric experiments. We describe how Nimrod has been extended to support the scheduling of computational resources located in a wide-area environment and report on an experiment in which Nimrod was used to schedule a large parametric study across the Australian Internet. The experiment provided both new scientific results and insights into Nimrod capabilities. We relate the results of this experiment to lessons learned from the I-WAY distributed computing experiment and draw conclusions as to how Nimrod and I-WAY–like computing environments should be developed to support desktop metacomputing.

1 Introduction

A *metacomputer* is a distributed collection of computers, potentially located at physically distant sites, that can be assembled to form a logical parallel computer. This logical computer can be used to access unique resources not accessible at a particular site or to assemble aggregate computational resources superior to that offered by a single site [3]. In principle, metacomputing can both increase accessibility to supercomputing capabilities and provide more cost-effective computing than do conventional high-performance systems.

Experiments such as the I-WAY/GII Testbed [4] have demonstrated serious applications on wide-area networks. However, concerns remain regarding the viability of the approach. Two questions appear particularly troublesome. Is there in practice a large base of applications able to exploit geographically distributed resources connected by networks with high latencies and low bisection bandwidth? Will programmers master the complexities inherent in computing in geographically distributed, heterogeneous, internetworked environments?

The I-WAY [9], Legion [10], and Globe [21] projects are addressing the usability issue by developing system services intended to provide the illusion of a single virtual machine. These efforts build on experience with systems such as PVM [19] that hide machine-specific details. The LSF network operating system [22] also attempts to provide the illusion of a large processor address space. Job management systems such as LoadLeveler [11], NQS, Codine [6], and others [12] map jobs placed in a work queue to physically distributed processors.

While low-level system services are important, they are not in themselves a sufficient solution to the problems of metacomputing applications and usability. We believe that new techniques are required that integrate metacomputer resources seamlessly into the user's desktop, much as network file systems allow users to access files without being aware that the bits are stored on remote servers. In this article we describe a problem-solving system that achieves this goal. This system, called Nimrod, combines a specialized environment with a metacomputer scheduling system. Users interact with a desktop interface to formulate parametric experiments, in which a user-supplied application program is executed for a range of parameter values. This computational problem is then decomposed automatically and scheduled transparently across local or remote computational resources. Hence, Nimrod addresses both usability and applicability concerns: it provides an easy-to-use problem solving environment and allows an important class of problems to access metacomputing resources.

Nimrod was originally developed to provide seamless access to a homogeneous collection of workstations located on a single local area network. In this environment, Nimrod has proved extremely effective in application studies involving users with a wide range of parallel programming skills [13]. In this article, we describe extensions to Nimrod that support its use in a metacomputing environment. These extensions include support for multiple architectures, including parallel supercomputers; new job startup mechanisms and file transfer mechanisms, suitable for wide area network; and alternative authentication mechanisms for job creation at multiple sites. We present the results of a case study used to evaluate the success of these extensions. In this case study, Nimrod is used to apply computers distributed across various Australian High Performance Computing Centers to a challenging scientific problem, namely the execution of a biological model of an important agricultural pest, the cattle tick. The experiment requires that large numbers of relatively fine-grained tasks (jobs averaged 2 minutes) be scheduled across heterogeneous systems (IBM SP2, SGI Power Challenge, DEC Alpha) connected by wide area networks with high latencies.

This first experiment in desktop metacomputing demonstrated the significant potential advantages of this approach to scientific problem solving. Operating from a desktop environment, we were able to solve in 30 minutes a problem that would have required 6 hours on a single workstation. The experiment also revealed areas in which our approach requires further refinement. We combine these lessons with those derived from the I-WAY demonstration and derive a list of future challenges that must be addressed if tools such as Nimrod are to provide desktop access to metasupercomputing in a routine manner. We conclude by indicating how Nimrod can be modified to meet these challenges.

2 The Nimrod Problem-Solving Environment

Nimrod automates the creation and management of large parametric experiments [1, 2]. It allows a user to run a single application under a wide range of input conditions and then to aggregate the results of these different runs for interpretation. In effect, Nimrod transforms file-based programs into interactive "meta-applications" that invoke user programs much as we might call subroutines.

Nimrod can be compared with optimization systems designed to locate local or global minima of user-supplied functions across parameter spaces [5]. It is distinguished from these systems by the fact that it does not require changes to user code. Nimrod also has similarities with job-scheduling systems such as Codine, LSF, NQS, and LoadLeveler, in that it treats individual runs of the user program as independent jobs that can be scheduled to remote systems. However, unlike other job distribution systems, it hides this scheduling and partitioning from the user, who thinks in terms of an experiment run over a parameter space.

Nimrod is also distinguished from other optimization and scheduling systems by its use of declarative experiment templates. As we describe below, parametric studies are defined using a simple declarative syntax, which is translated automatically into a graphical user interface. The user interacts with the experiment via this interface and, in many cases, need not write any additional software. Hence, experiments can be prototyped quickly.

2.1 Defining an Experiment

Nimrod processes a user-supplied declarative *experiment template* to obtain a graphical *control panel* used to initiate, monitor, and control an experiment. The template defines the structure and valid ranges for input parameters, which may be supplied as command line arguments, standard input or general input files. The control panel incorporates sliders for variable values, lists for discrete values, radio buttons for literal values, and so forth.

A template is defined by a file containing `parameter` statements defining input parameters. (The file can also contain `script` statements, defining the actions to be performed in different phases of the computation; we describe these below.) A parameter statement has the following syntax:

```
parameter name label type [ type_argument ... ]
```

where `name` specifies the variable that represents the parameter in the control scripts, `label` identifies the parameter to the user, and `type` indicates how the user will specify the parameter. Valid types are `select`, `switch`, `text`, `list`, and `range`. For each type, further arguments specify subtypes, defaults, and limits for the values.

Figure 1 illustrates the use of the `parameter` statement. The first statement defines a parameter `year` with values specified by a user-defined list; a separate window is opened when the user clicks on the field in the interface. The second parameter, `ndip`, takes at most seven values between 3 and 9; at run time the user chooses the number of elements, and a list is subsequently generated. The third parameter, `description`, takes an arbitrary text string. The fourth, `dtype`, can take any one of four literal constants. Figure 2 shows the control panel generated from the template of Figure 1.

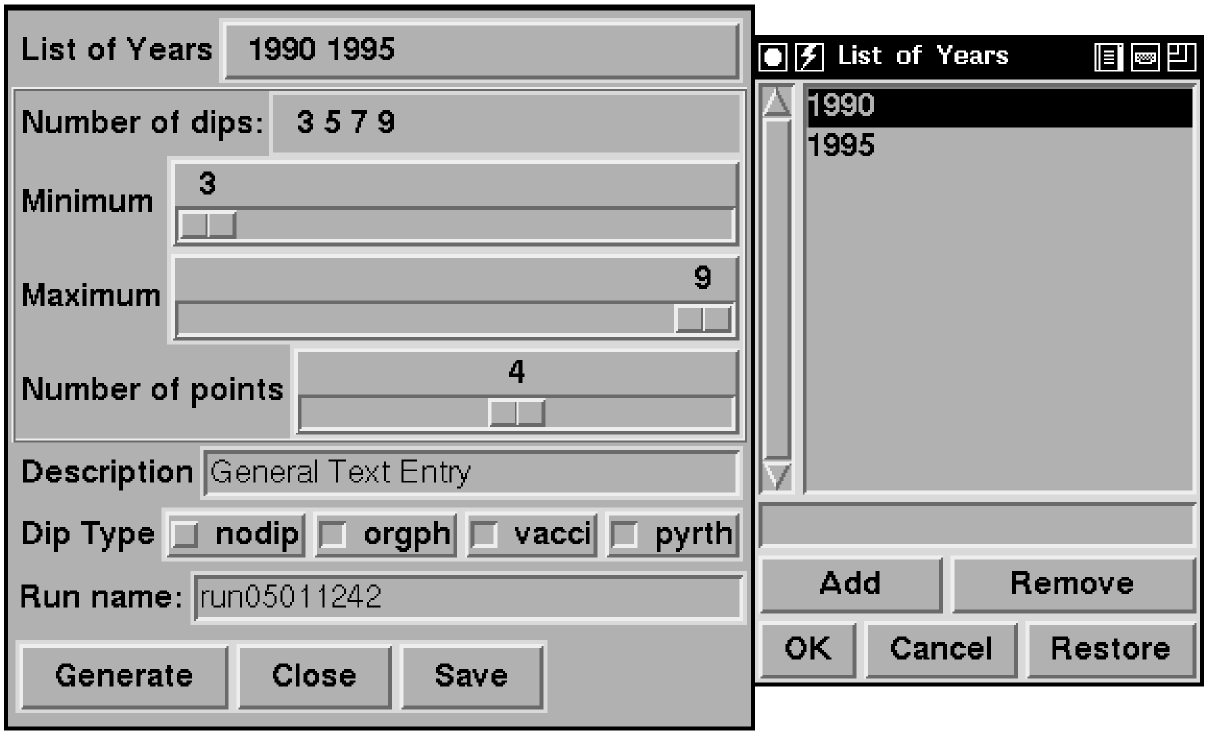
2.2 Running an Experiment

Once a control panel is defined, an experiment proceeds as follows:

1. We use the sliders and other controls provided by the control panel to select the parameter values for which we wish to run our program.

```
parameter year "List of Years" list
parameter ndip "Number of dips" range atmost 7 from 3 to 9
parameter description "Description" text
parameter dtype "Dip Type" select anyof "nodip" "orgph" "vacci" "pyrth"
```

Figure 1: Four example `parameter` statements, defining the input parameters `year`, `ndip`, `description`, and `dtype`, respectively



List of Years: 1990 1995
Number of dips: 3 5 7 9
Minimum: 3
Maximum: 9
Number of points: 4
Description: General Text Entry
Dip Type: nodip, orgph, vacci, pyrth
Run name: run05011242

List of Years
1990
1995

Figure 2: Sample user interface generated by template

2. We request Nimrod to perform the specified runs. During this phase, we can use a second *status panel*, also provided by Nimrod, to track the progress of our experiment.

3. Finally, we examine the results of the experiment, perhaps using a scientific visualization package.

The parameter choices made in the first step determine the number of runs to be performed in the second step. For example, we may specify that parameter X is to take 5 values in the range 0 to 100, while parameter Y takes 10 values in the range –10 to 10. Together, these two specifications request $5 \times 10 = 50$ runs of our program.

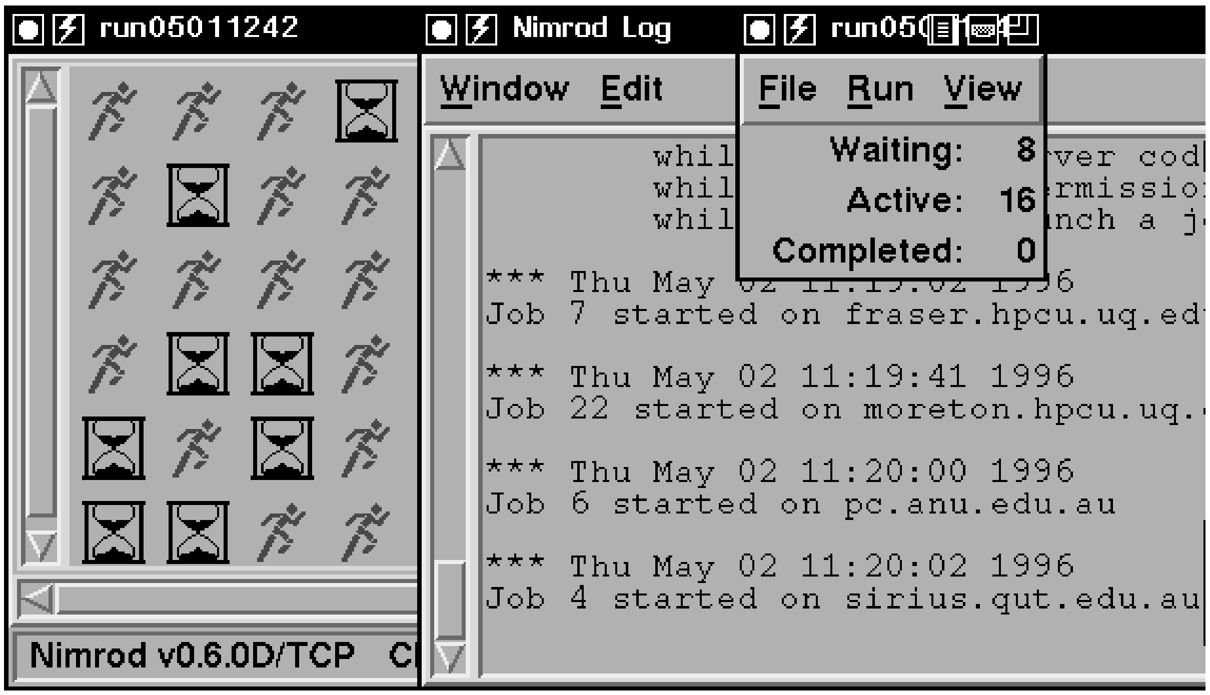
Nimrod distinguishes seven phases in the second step: experiment initialization, job setup, job execution, job cleanup, experiment termination, server initialization, and server shutdown. Most of the actions to be performed in each phase can be specified in the experiment template by using `script` statements. Figure 3 shows script statements for job startup, job cleanup and job execution phases. In this example, file `input` is sent to the remote site before the job is started and file `output` is returned upon termination. The job execution script indicates how to run the program, using a shell script that accepts the file names `input` and `output`. This figure also illustrates the use of parameter substitution. The `$` symbol indicates that parameter substitution should occur when the script is interpreted. For example, `$year` will be replaced by the value of the parameter `year` when the script runs. The `fsubst` command searches the file `input` and replaces any parameters with their actual values. Hence, the application receives different values for each run. The `put` and `get` commands are used to send files to a remote system and to return results, respectively.

```
script job setup {
        fsubst input nimrod.input
        put nimrod.input input
}

script job cleanup {
        get output output.$year.$ndip.$dtype
}

script job execute { ./run_tick1g input output }
```

Figure 3: Example `script` statements for job setup, job cleanup, and job execution

An experiment's status panel (Figure 4) is used to initiate, monitor, and control the experiment. One component of this panel shows the progress of individual jobs. Each icon represents a job and indicates whether it is awaiting execution, is running, or has completed. A special icon is displayed if the job fails, either through an error or because the server connection is broken. The user can click on an individual icon to display job parameters, the name of the machine on which it is running and the exact state of the job: suspended, waiting, or running. The user can also nominate selected application scalar variables to be monitored. Their values are then extracted and displayed along with other parameter information. This feature is useful for long-running jobs because it allows the user to determine the exact progress of the run, for example by monitoring the time step or computed error values within an equation solver. Variables are accessed with the Dynascope library [18, 17], which provides a convenient method for extracting data from remotely executing programs.

The final phase of an experiment is data aggregation and display. This functionality is supported by a user-specified experiment completion `script` that can invoke external data filters and visualization tools. A filter is often used to reduce the computed data, and a visualization package such as AVS used to display the results.



Waiting: 8
Active: 16
Completed: 0

```
whil                     ver cod
whil                     rmissio
whil                     nch a j
*** Thu May 02 11:19:02 1996
Job 7 started on fraser.hpcu.uq.ed

*** Thu May 02 11:19:41 1996
Job 22 started on moreton.hpcu.uq.

*** Thu May 02 11:20:00 1996
Job 6 started on pc.anu.edu.au

*** Thu May 02 11:20:02 1996
Job 4 started on sirius.qut.edu.au
```

Figure 4: A Nimrod status panel

2.3 Nimrod Architecture

Nimrod uses a client-server architecture. A user runs a Nimrod client on their desktop to create and manage experiments. A user can start multiple clients to run more than one experiment concurrently. Each user also runs their own remote execution server (RES) on each machine that can accept jobs. A RES not only manages the execution of the application but also is responsible for transferring files between the client and server machines. Since each user runs their own RES, the servers inherit the access privileges of the given user.

Nimrod does not assume a shared file system or even a global naming system for files. When a job is initiated, the input files required for the run are transferred to the remote system, and output files are returned after each run. A RES builds a unique location in the target file system for each job. This avoids file name space conflicts in the event that a machine accepts more than one job or several machines share a file system. For efficiency reasons, it is possible to share common input files across jobs by using absolute path names rather than local ones, in which case they are not altered by the server.

3 A Case Study

We describe an experiment in which Nimrod is used to evaluate control strategies for cattle tick, a major agricultural pest in Australia. This study is interesting from a scientific perspective because it provides new results in management strategies. From a metacomputing perspective, it is interesting because it involves a large number of relatively short-lived tasks and because we use Nimrod to map these tasks to computers located across Australia. Hence, the experiment provides a challenging test of Nimrod's scalability and performance.

3.1 Experimental Design

Cattle ticks (*Boophilus microplus*) affect about one third of Australia's cattle. The cost of cattle tick management is estimated at $150 million per year, due to lowered production and expensive control measures. Enormous savings are to be made by optimizing the application of control measures and utilization of resistant strains. Optimal control techniques also can help to reduce chemical resistance within tick populations and to minimize residual pesticide levels.

Our experiment uses TICK1 [20, 14], a simulation code developed to study cattle tick ecology. TICK1 is a climate-driven, process-based, discrete time step (weekly) model of the population dynamics of cattle ticks. It incorporates models of various ecological and physiological tick development processes, including on-host survival, competition between ticks, and avoidance behavior in cattle. Process rates are calculated as a function of a number of meteorological, pasture cover, and host-related variables. Herd composition data for Australia, obtained from the Australian Bureau of Resource Economics, are used to derive weighted average herd characteristics and stocking rates. A soil dryness index is derived from meteorological data by using a single-layer soil water model based on [7]. Each simulation uses long-term average climate data and is run to equilibrium (ten years) after an initialization. The model is written in Fortran 77 and uses the NCSA netCDF/HDF data file format [16] to handle climate surface data and simulation output. A 50 km grid is used across Australia (2785 locations). Simulation inputs are obtained from a commercial geographical information system; these comprise climatic (rainfall, maximum and minimum temperature and evaporation), herd composition, stocking rate, and management strategy data. These data do not change across the scenarios considered in this article.

The goal of the experiment considered in this article was to use TICK1 to design a minimal-cost treatment strategy for all of Australia. Hence, we designed an experiment that varied TICK1 input parameters relevant to a cost-effective strategy, namely the timing, treatment interval, and number of treatments. Three treatments were considered: organophosphate dip, pyrethrin dip, and vaccination. Treatment effectiveness was measured in terms of a single metric that combined the costs of reduced live weight gain and treatment.

As discussed in Section 2, Nimrod defines experiments in terms of a template that identifies the model variables that are to be varied. In this case, four parameters are defined: the time of the year in which treatment begins, the number of treatments after that time, treatment frequency, and treatment type. For example, a suitable treatment strategy might entail administering a vaccination at fortnightly intervals, up to five times, starting at week 15 of the year. In the actual experiment we explored the following parameter values:

- Number of treatments = 3, 5, 7, 9
- Starting weeks = 8, 20, 33, 46

- Interval between treatments = 2, 5, 8, 12 weeks
- Treatment type = vaccination, pyrethrin, organic phosphate dip.

This range of values yielded 192 TICK1 runs, each requiring less than 2 minutes to execute on a modern RISC microprocessor. Hence, the experiment would have taken over six hours to run on a single workstation. While relatively short, this run was of interest to the biologists; in fact, as we explain below, it is actually a preliminary run intended to identify strategies to be evaluated in more complex experiments. In addition, it typifies applications in which metacomputing converts a time consuming activity to something that can be performed almost in real time from the desktop.

3.2 Scientific Results

Figure 5 presents some of the results obtained from the Nimrod-managed TICK1 experiment, showing for each grid point the lowest cost obtained over all 192 scenarios. The maximum cost of $2.40 per head is found in northern Australia. The regions in central and South Australia with no cattle tick show a zero management cost. The most striking feature of the analyses is the effect seen in the intermediate zone. These regions represent a cost of about $1.20 per head, and further analysis indicates that this is the cost of the cattle tick alone because no treatment is administered in these regions. This suggests that it is cheaper to leave the ticks untreated than to apply a treatment schedule.

The conclusions of this one experiment need to be considered carefully in the context of the limited information used by the model. In particular, we used only a rudimentary cost model; greater detail is required in terms of the herd structure and stocking rate. For example, a more accurate model would reflect the fact that different types of cattle have different natural resistance to ticks. In addition, we note that the simulations performed here represent only a small proportion of the total experiment space that needs to be explored. The timings presented here suggest that a full-scale experiment would require 225 hours on a workstation, even before the added complication of seasonal climate variation and tactical dipping is investigated. This same experiment could feasibly be explored in about 3 hours on a metacomputer of the type used in our work. This will be the basis of continuing work in the area.

3.3 Computational Approach and Results

The computational platform used for the experiment was constructed from the 78 processors listed in Table 1. Previous Nimrod experiments have used only one type of processor; here, we use three different types of workstations, namely an IBM SP2 and several SGI Power Challenge systems and DEC Alpha workstations. These machines represent quite different microprocessors and computer architectures, testing Nimrod's ability to handle heterogeneous systems.

Nimrod can be used to move both object code and input files to each remote system. However, to minimize remote storage usage, it copies these files once for each job and deletes them upon termination. To reduce communication, we copied constant input files onto remote machines just once. Hence, per-job communication requirements comprised

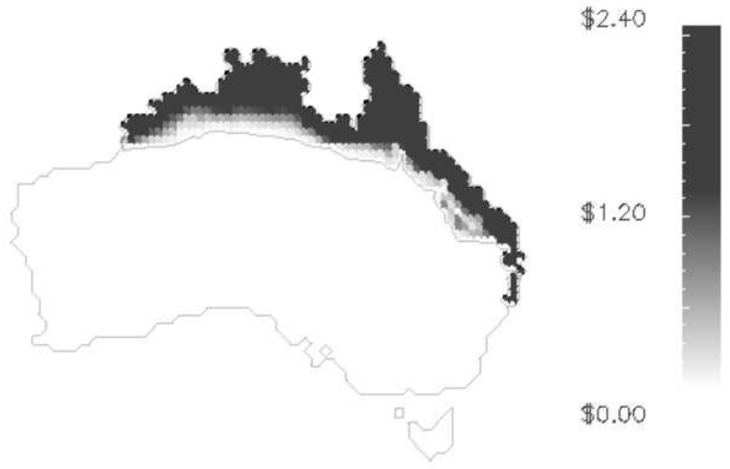


Figure 5: Variation of tick management cost across Australia: Black regions represent a unit cost of $2.40 her head of cattle, and white pixels are zero cost

Table 1: Location and type of machine used in the TICK1 metacomputing experiment

Location	Machine Type	Number of Processors
Australian National University	SGI Power Challenge	8
University of Adelaide	SGI Power Challenge	8
University of Queensland	SGI Power Challenge	16
James Cook University	SGI Power Challenge	8
Queensland University of Tech.	SGI Power Challenge	4
Griffith University (DSTC)	DEC Alpha Workstations	4
Griffith University (QPSF)	IBM SP2	12
University of Queensland (DSTC)	DEC Alpha Workstations	18
Total		78

only the parameter file, of size 400 bytes. This staging of input files could easily be automated. The HDF output files created for each run are each about 10 MB in size. As only a fraction of this data is required for analysis, we used a postprocessor to extract the required data. The data were written to another file, which, after compression, occupied only about 1.5 kB. Hence, communication and storage requirements for our 192 jobs were reduced from 2 GB to about 300 kB.

In the absence of any overheads or load imbalances, the 78 processors listed in Table 1 would have completed our experiment in 7 minutes; in practice, it took about 30 minutes of elapsed time. While significantly better than 6 hours, this is not ideal. We attribute this relatively poor performance to Nimrod's centralized scheduling architecture, the slow speed of the remote job spawning mechanisms used by Nimrod, and the small sizes of TICK1 tasks. Currently, the Nimrod scheduler is completely centralized. Furthermore, since the scheduler is written using a mixture of Tcl and C, scheduling a single task takes a relatively long time: about 3 seconds on a local area network and an average of 10 seconds on the Internet because of network congestion. (Between Brisbane and Adelaide, it took up to 30 seconds to start jobs.) Since a single TICK1 simulation runs quite quickly—in about 2 minutes of CPU time—we would need to spawn a job every 1.5 seconds in order to make effective use of 78 processors.

Nimrod performance can be improved substantially by making relatively minor changes to its scheduling architecture. For example, we can provide a hierarchical scheduler that migrates multiple tasks to subschedulers located at remote sites. To test the effectiveness of this approach, we combined a number of TICK1 tasks together into one script, which meant that each job ran for a much longer time. Using this technique, we were able to schedule all 78 processors before any jobs completed.

3.4 Discussion

Creating the Nimrod experiment template was simple. It took about one hour from the time we started developing the script to the time we had an application capable of running on a distributed platform. This result is quite dramatic when compared with alternative technologies, such as building a parallel program which performed the same task, or using an existing job management system. At all times we were able to monitor the experiment in terms of the parameter values using an automatically generated GUI without concern for the details of the underlying computational platform.

The TICK1 experiment also revealed deficiencies in terms of the techniques used to exploit widely dispersed workstations. In the next section we address these and highlight some general challenges for desktop metacomputing types of applications.

4 Challenges and Technologies for Desktop Metacomputing

cIn the preceding section, we discussed a large-scale experiment in which Nimrod was used to map a moderate-sized parametric experiment across 78 processors located at eight sites connected by the Australian Internet. We are interested in understanding what this experiment tells us about the practicality of integrating large-scale metacomputing systems into desktop applications. Rather than discuss Nimrod in isolation, we place the discussion in the context of the lessons learned from the I-WAY wide-area computing experiment. The I-WAY was designed to support a rather different class of applications to Nimrod: for the most part, tightly coupled applications with demanding network quality of service requirements. Hence, it is interesting to understand how I-WAY and Nimrod requirements correspond and differ.

4.1 The I-WAY Experiment

The I-WAY project [4] was conceived in early 1995 with the goal of providing a large-scale testbed in which innovative high-performance and geographically distributed applications could be deployed. The testbed comprised an ATM network connecting supercomputers, mass storage systems, and advanced visualization devices at 17 different sites within North America. It was deployed at the Supercomputing conference (SC'95) in San Diego in December 1995 and used by over 60 application groups for experiments in high-performance computing, collaborative design, and the coupling of remote supercomputers and databases into local environments. A management and application programming environment, called I-Soft [9], provided uniform authentication, resource reservation, process creation, and communication functions across I-WAY resources.

The I-WAY was successful in demonstrating that large-scale, high-performance metacomputing is feasible and useful. Just as important, it provided the first application-oriented testbed in which to identify the critical issues affecting future progress in this area. In the rest of this section, we review some of the lessons learned from the I-WAY experiment and discuss how these lessons relate to desktop metacomputing systems such as Nimrod. This discussion motivates a number of proposals for Nimrod extensions.

4.2 Network Awareness

Predictably, network latencies in metacomputing systems tend to be both high—roundtrip times of hundreds of milliseconds can be expected on a continental scale—and variable. Bandwidth also tends to be scarce and unpredictable. A clear lesson from the I-WAY experiment was that both applications and tools need to be able to negotiate quality of service (QoS) requirements with a scheduler or network management system. In addition, applications and tools may need to be able to determine network properties such as topology and delivered QoS, so that they can adapt algorithms and protocols to maximize performance [8].

Previous work on wide area scheduling has not really addressed these issues, and it might appear that the coarse-grained tasks typically scheduled by such systems would not be overly sensitive to these factors. However, the TICK1 experiment emphasizes that similar information is important if systems such as Nimrod are to provide robust performance across a range of problem sizes and network topologies. For example:

- Information about machine capabilities can be used to determine when cheaper protocols for job startup are applicable.
- Information about network topology and machine capacities can allow us to stage large input files to intermediate nodes in the network.
- Information about network bandwidth and latency can be used to control task granularity, for example by expanding parameters in the client or the server.

We are currently building a new version of Nimrod that will apply these sorts of optimizations.

4.3 Scheduling

Metacomputing systems tends to be highly heterogeneous. This characteristic leads to the need to maintain multiple code versions, to convert between alternative data formats, and so forth. However, in many respects a more serious problem is that the software and management architectures at different sites are also heterogeneous. For example, different sites typically employ different authentication, file system, and scheduling mechanisms. A metacomputing system cannot impose uniform mechanisms but must interoperate with local solutions.

Scheduling is a major area in which heterogeneity causes problems. While in local environments it is often possible to deploy a single, uniform scheduler, political and technical constraints will typically make it infeasible to provide a single "metacomputer scheduler" to replace the schedulers that are already in place at various sites. I-Soft addressed this problem by adopting a two-part strategy that allowed administrators to configure dedicated resources into virtual machines and allowed users to request time on particular virtual machines. The strategy involved a (1) central scheduler daemon that managed and allocated time on the different virtual machines on a first-come, first-served basis, and (2) a local scheduler daemon communicating directly with the local site scheduler. Local schedulers performed site-dependent actions in response to requests from the central scheduler to allocate resources, create processes, and deallocate resources [9].

Nimrod does not currently use local machine schedulers. In the TICK1 experiment, we disabled the native schedulers (which in most cases was LSF) and required users to start Nimrod execution servers directly. In effect, we negotiated access to remote resources manually—via e-mail or telephone. This technique worked well in the TICK1 experiment, but it is clear that Nimrod must be integrated with native machine schedulers in the future. For example, Nimrod should be able to negotiate with local schedulers to obtain "low-grade" cycles when machines are otherwise idle. As a result of the case study, we plan to build a new set of remote execution servers for Nimrod that make use of the I-Soft scheduling capability.

4.4 Distributed File Systems

Metacomputing systems introduce three related requirements with a file-system flavor. First, users and tools require access to various status data and utility programs at many sites. Second, users running programs on remote computers must be able to access executables and configuration data at many sites. Third, application programs must be able to read and write potentially large data sets. These three requirements have different characteristics. The first requires support for multiple users, consistency across sites, and reliability. The second requires somewhat higher performance (if executables are large) but does not require support for multiple users. The third requires, above all, high performance. It seems likely that these three requirements are best satisfied with different technologies.

In Nimrod, the file system problem is easier, but only because jobs are constrained to fit a simple format in which the set of required files is specified ahead of time by the user. Hence, Nimrod can operate in environments in which a global file system is not available, moving files between between sites explicitly as they are required. Furthermore, traffic can be reduced by staging input files and filtering and compressing output files. These techniques worked well in the TICK1 experiment. For other applications, we can imagine difficulties, for example if the input files required by an application are data dependent. In these situations, distributed file system support could be invaluable as a means of propagating and caching large files and code in a demand-driven manner. We will investigate this issue in future research.

4.5 Security

Security is a major and multifaceted issue in metacomputing systems. Ease-of-use concerns demand a uniform authentication environment that allows a user to authenticate just once in order to obtain access to geographically distributed resources; performance concerns require that once a user is authenticated, the authorization overhead incurred when accessing a new resource be small. Both uniform authentication and low-cost authorization are complicated in scalable systems because users will inevitably need to access resources located at sites with which they have no prior trust relationship.

The Nimrod system used for the TICK1 study adopted a simple approach to security problems. Each Nimrod user is required to have an account (and hence a prior trust relationship) at every site. Execution servers are then run under the id of the user. A magic token is used to ensure that servers accept only authorized requests, in a similar way to X windows (XAuth [15]).

The current Nimrod approach works reasonably well across a wide range of platforms. Because it does not require any privileges not already granted to a normal user, it does not result in any major security difficulties. Its most significant disadvantage is the need for a prior trust relationship. We propose to use I-Soft mechanisms to address these issues in the future.

5 Conclusions

We have described the Nimrod problem solving environment and explained how it can provide transparent desktop access to metacomputer resources for an important class of applications. Users supply an application program and a declarative definition of a parameter study experiment; Nimrod constructs a graphical user interface for the experiment and manages the execution of individual jobs on local and remote resources. Nimrod has been used successfully to map a medium-scale parametric study over 78 processors located at eight sites across Australia; the results of this study yielded new insights into the management of an important natural pest, the cattle tick. Nimrod demonstrates that it is indeed possible to achieve ease of use in a metacomputing environment; it also opens an important class of applications to metacomputing. Because Nimrod is a generic tool, it can be applied to a wide range of projects involving scientific modeling. Prototype versions of Nimrod are currently in use by six strategically chosen users. The level of interest to date has been most encouraging.

Our experiments also identified apparent deficiencies in the Nimrod architecture. In an attempt to identify important issues, we have compared Nimrod requirements with problems identified as important within the I-WAY/GII Testbed wide area networking experiment. This comparison revealed striking similarities between Nimrod and I-WAY requirements. While Nimrod can function effectively with extremely simple solutions to problems of network characterization, scheduling, distributed file systems, and security, it is clear that future Nimrod-like systems can benefit significantly from the techniques proposed for the I-WAY. For example:

- Information about network characteristics can be used to optimize scheduling decisions, hence improving throughput and allowing users to solve more fine-grained problems.
- Interfaces to site schedulers can allow Nimrod to negotiate availability of resources before starting an experiment; this information can be used to size an experiment to meet user turn-around requirements.
- Simple distributed file system support providing high-performance I/O capabilities would allow Nimrod applications to access files in a more flexible fashion.
- Fine-grained authentication schemes that allow access to sites with which a user has no prior trust relationship can expand dramatically the opportunities for remote execution.

We are currently designing extensions to Nimrod that will address these and other issues.

Acknowledgments

This article reports the work of many people and organizations. We are grateful to Griffith University, Argonne National Laboratory, the Co-operative Research Centres for Distributed Systems Technology (DSTC) and Tropical Pest Management (CTPM), and the Queensland Parallel Supercomputing Facility. Silicon Graphics machines were made available courtesy of the Australian National University, the University of New South Wales, the University of Adelaide, the University of Queensland, the Queensland University of Technology and James Cook University. The Nimrod project has been funded by the DSTC since 1994.

References

[1] D. Abramson, R. Sosič, J. Giddy, and M. Cope. The laboratory bench: Distributed computing for parametised simulations. In *1994 Parallel Computing and Transputers Conference*, pages 17–27, Wollongong, Australia, 1994.

[2] D. Abramson, R. Sosič, J. Giddy, and B. Hall. Nimrod: A tool for performing parametised simulations using distributed workstations. In *Proc. of the 4th IEEE Symposium on High Performance Distributed Computing*, 8 1995.

[3] C. Catlett and L. Smarr. Metacomputing. *CACM*, 35(6):44–52, 1992.

[4] T. DeFanti, I. Foster, M. Papka, R. Stevens, and T. Kuhfuss. Overview of the I-WAY: Wide area visual supercomputing. *Intl J. Supercomputer Applications*, 1996. in press.

[5] M. Eldred, D. Outka, C. Fulcher, and W. Bohnhoff. Optimization of complex mechanics simulations with object-oriented software design. In *Proceedings of the 36th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, pages 2406–2415, New Orleans, LA, 4 1995.

[6] F. Ferstl. CODINE Technical Overview. Technical report, Genias, 4 1993.

[7] E. A. Fitzpatrick and H. A. Nix. A model for simulating soil water regime in alternating fallow-crop systems. *Agricultural Meteorology*, 6:303–319, 1969.

[8] I. Foster, J. Geisler, C. Kesselman, and S. Tuecke. Multimethod communication for high-performance metacomputing applications. Preprint, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Ill., 1996.

[9] I. Foster, J. Geisler, W. Nickless, W. Smith, and S. Tuecke. Software infrastructure for the I-WAY high-performance distributed computing experiment. In *Proc. of the 5th IEEE Symposium on High Performance Distributed Computing*. IEEE, 1996.

[10] A. Grimshaw, W. Wulf, J. French, A. Weaver, and P. Reynolds, Jr. Legion: The next logical step toward a nationwide virtual computer. Technical Report CS-94-21, Department of Computer Science, University of Virginia, 1994.

[11] IBM. IBM LoadLeveler: User's Guide. *International Business Machines Corporation*, 3 1993.

[12] J. Kaplan and M. Nelson. A comparison of queuing, cluster and distributed computing systems. Technical Report 109025, NASA, Langley Research Centre, Hampton, Virginia, 23681-0001, 10 1993.

[13] A. Lewis, D. Abramson, R. Sosič, and J. Giddy. Tool-based parameterisation : An application perspective. In *Computational Techniques and Applications Conference*, Melbourne, Australia, 7 1995.

[14] G. F. Maywald, M. J. Dallwitz, and R. W. Sutherst. A systems approach to cattle tick control. In *Proceedings of the 4th Biennial Conference of the Simulation Society of Australia*, pages 131–139, 1980.

[15] Linda Mui and Eric Pearce. *X Window System Administrator's Guide*. O'Reilly and Associates, Inc, Sebastopol, California, 1992.

[16] NCSA. The HDF reference manual version 3.3. Technical report, National Center for Supercomputing Applications., 2 1994.

[17] R. Sosič. Design and implementation of Dynascope, a directing platform for compiled programs. *Computing Systems*, 8(2):107–134, Spring 1995.

[18] R. Sosič. A procedural interface for program directing. *Software–Practice and Experience*, 25(7):767–787, July 1995.

[19] V. Sunderam, A. Geist, J. Dongarra, and Mancheck. The PVM concurrent computing system: Evolution, experiences and trends. *Parallel Computing*, 20(4):531–546, 3 1994.

[20] R. W. Sutherst and M. J. Dallwitz. Progress in the development of a population model for the cattle tick boophilus microplus. In *Proceedings of the 4th International Congress of Acarology*, pages 557–563, 1974.

[21] M. van Steen, P. Homburg, L. van Doorn, A. Tanenbaum, and W. de Jonge. Towards object-based wide area distributed systems. In *Proc. of the International Workshop on Object Orientation in Operating Systems*, pages 224–227, 1995.

[22] S. Zhou, J. Wang, X Zheng, and P Deliale. Utopia: A load sharing facility for large, heterogeneous distributed systems. Technical Report CSRI-257, Computer Systems Research Institute, University of Toronto, Toronto, Canada, M5S 1A1, 1992.