The 4th IEEE Symposium on High Performance Distributed Computing, Virginia, August 1995.

Nimrod: A Tool for Performing Parametised Simulations using Distributed Workstations

D. Abramson †
R. Sosic †
J. Giddy ‡
B. Hall §

†School of Computing and Information Technology
Griffith University
Kessels Rd, Brisbane,
Queensland, 4111
{davida, sosic}@cit.gu.edu.au
Phone: +61-7-875 5049
Fax: +61-7-875 5051

‡ Co-operative Research Centre for
Distributed Systems Technology
jon@cit.gu.edu.au

§ School of Science,
Griffith University

ABSTRACT

This paper discusses Nimrod, a tool for performing parametised simulations over networks of loosely coupled workstations. Using Nimrod the user interactively generates a parametised experiment. Nimrod then controls the distribution of jobs to machines and the collection of results. A simple graphical user interface which is built for each application allows the user to view the simulation in terms of their problem domain. The current version of Nimrod is implemented above OSF DCE and runs on DEC Alpha and IBM RS6000 workstations (including a 22 node SP2). Two different case studies are discussed as an illustration of the utility of the system.

1 INTRODUCTION

A wide range of scientific and engineering experiments can be solved using numeric simulation. Examples include finite element analysis, computational fluid dynamics, electromagnetic and electronic simulation, pollution transport, granular flow and digital logic simulation. Accordingly, some very large codes have been written over the years, mostly in FORTRAN and mostly with primitive user interfaces. A typical FORTRAN simulation program may consist of 50,000 lines of code and usually performs its input and output by reading and writing files. Users typically generate a set of test input files and submit the jobs to a batch queue for execution. Post processing is used to display and visualise the results. This approach has been adequate for large mainframes, vector supercomputers and even modern workstations.

In parametric studies a range of different simulations are calculated using the same program. Each simulation, which may takes hours to run, computes output variables for one particular set of input conditions. The following brief list gives an indication of the types of parametric studies that might be performed:

- An environmental engineer may wish to perform an impact study on varying the amount of waste sent into a dump site;
- An architect may wish to study the effect of varying the rigidity of a particular beam in a building design and produce a solution which optimises both cost and safety;
- An aerospace engineer may wish to alter the shape of an aerofoil and observe the effect on drag, lift and cost;

- An electronics designer may wish to vary component tolerances and observe the effect on the performance of some analogue circuit;

- An aerial designer may wish to alter the shape of an antenna and observe its gain.

Existing techniques for controlling such studies are extremely time consuming and can involve very complex and long job initialisation phases. Typically, the user must create a set of input files for each of the different parameter settings, run the modelling program against each set of files, collate and merge the output files and finally produce some form of condensed output. For example, the output may simply be a low dimensional plot (often two or three dimensions) against the input parameters. For any one run, Pancake [9] has measured that users spend in the order of 10% of time performing job setup; for a multiple scenario experiment this may consume even more time because the operation of parametisation and control are more complex. If the job set up is performed manually, great care must be taken to ensure that the input and output files are kept correlated.

Parametised simulations can require enormous amounts of processor time. It is not uncommon for such models to require in the order of hours to evaluate any one set of parameters. If only three variables are explored, with 4 values per variable, 64 different simulations must be performed, and this can amount to days of workstation time. Distributed workstations have the potential to provide the necessary computational resource. They are inexpensive, and are often idle for long periods. However, managing the multiple jobs can be complex and time consuming, and thus tools are required in order to harness their power effectively, and to provide a seamless computational platform.

In this paper we discuss a tool, called Nimrod[1], for managing the execution of parametised simulations on distributed workstations. Because each point in the search space is independent, it is possible to execute the tasks concurrently on separate machines. Nimrod is targeted at the application engineer or scientist rather than at a systems programmer. It provides a high level view of the experiment being conducted whilst utilising a wide range of computing platforms to perform the underlying work.

The paper begins with a discussion of distributed supercomputers and surveys the available tools. It then discusses the design and implementation of Nimrod which is built on top of OSF's Distributed Computing Environment (DCE). The paper gives some case studies as examples of the use of Nimrod.

[1] Our work is inspired by "CONDOR: A Hunter of Idle Workstations " [7]. The biblical character Nimrod was another accomplished hunter.

2 DISTRIBUTED SUPER COMPUTERS

Traditional super computers achieve their performance either through vector hardware or tightly coupled multiprocessors. Providing a program exhibits a high degree of parallel or vector activity, the performance of these systems can be high. However, they are expensive compared to workstations because they employ special hardware and/or software.

Distributed Super Computing refers to the use of a large number of loosely coupled machines to perform one task. Because the speed of the interconnection network between workstations is slow compared to those in tightly coupled parallel machines, they cannot be used for general parallel computing unless the tasks to be executed are allocated in very large units. The class of problems being considered in this paper exhibit moderate levels of concurrency (of the order of 50-100 independent model executions) and any one model run may require several hours of compute time. Distributed supercomputers represent an ideal platform for performing such work because the ratio of computation time to job setup time is high, and thus the communication costs of LANs and WANs can be ignored. Further, it is often possible to secure overnight in the order of 100 workstations which would otherwise be idle.

Until recently there have been two main ways of using distributed supercomputers. One, through remote job execution of multiple jobs, and the other through genuinely distributed applications.

2.1 Existing Remote Job Execution Systems

A number of software packages exist which help manage queues of jobs to remote computer systems. Some examples are Condor, DQS, DJM, LoadLeveler, LoadBalancer, LSF, CODINE and NQS/Exec [5, 6, 7, 8, 9, 10, 11, 14]. Whilst Condor, DQS and DJM are in the public domain, the others are commercial products. They all provide the basic service that is summarised in Figure 1. Users generate a number of *jobs* and then submit them to the queue management system. The jobs are run on available machines and the results returned to the controlling machine. Most of the systems allows users to access files on the originating machines through networked file systems such as NFS and AFS. Some of the systems provide graphical user interfaces to help tracing the jobs as they move through the system.

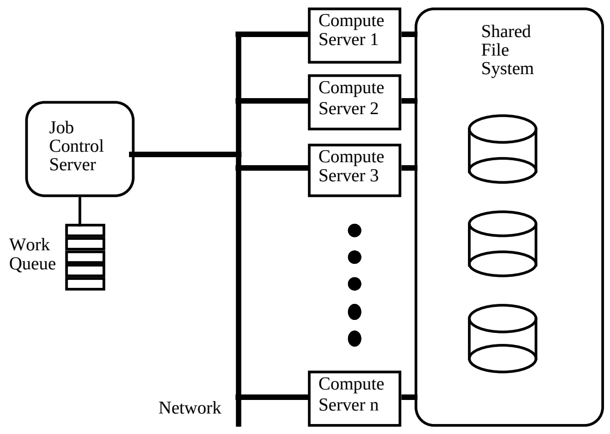


Figure 1 - a generic remote job execution framework

An important feature of this type of distributed supercomputing is that the jobs are unaware that they have been executed in a distributed manner. This has the clear advantage that the programmer need only generate a sequential task. The disadvantage is that users must concern themselves with the task of generating and distributing multiple jobs. This process can be complex and error prone, especially when some of the jobs fail to execute because they have terminated early. Unless special software is written to generate the jobs, the user cannot monitor the progress of the jobs in a way which is meaningfully related to the original parametised request. For example, in parametric strudies, the user requires the computation of a number of different data sets, but the remote job execution system deals in terms of jobs and remote machines.

2.2 Distributed Applications

Distributed applications are built with the knowledge that they will execute on multiple platforms. Thus, an application which performs parametised simulation must be capable of setting up simulations for each of the parameter sets and then executing them using multiples processes. It must handle the initialisation of the processes as well as the aggregation of the results.

There are many tools and environments for building distributed applications. The paradigms vary from remote procedure calls through to co-operating sequential processes. In the remote procedure call system, a program executes a procedure call which causes the routine to execute on another processor. The call looks like a normal procedure call except that parameters are copied in and out rather than passed by reference or value. Since the call blocks until the procedure terminates, multiple calls must be performed in concurrent threads in order to execute more than one simulation at the same time. Examples of remote procedure calls are OSF's DCE [15] system and SUN's RPC [18]. If co-operating sequential processes are used, then the individual simulations can be spawned on other processors, and the parameter information can be distributed through message passing. Unlike RPCs which block, it is possible for the master process to start a number of slaves without waiting for one to complete before starting the next. Examples of portable, multi-lingual, message passing systems include PVM [19] and the P4 system from Argonne National Laboratories [4].

The major advantage of this approach over using a queue manager is that the user interface can be tailored to the problem domain of the application, and all of the details of job generation and resource information can be hidden from the user. Further, the progress of a job through the system can be easily traced in terms of its original parameter settings. Thus, it would be conceivable to build an application which requests a number of parameters from the user, and then automatically generates and distributes the work across a number of workstations. However, the major disadvantage of this approach is that the original application must be modified to take account of distribution. This can be a time consuming task as it may require significant alterations to the program source code. For example, issues such as fault tolerance and restarting of jobs must be addressed from within the application. Further, the source of the application may not even be available to allow the required modifications.

3 NIMROD: A Tool for Parametised Simulations

Nimrod is a tool that combines the advantages of remote queue management systems with those of distributed applications. It provides the user with a problem oriented view of their application without requiring any modifications to their simulation code. Figure 2 shows the overall structure of Nimrod. The user is presented with a generation screen which gives the various parameters to the simulation. After the user specifies the simulation parameters, Nimrod takes the cross product of these parameters and generates a job for each set. It then manages the distribution of the various jobs to machines, and organises the aggregation of results.

By implementing log facilities Nimrod can be restarted at any time throughout an experiment. This attribute is important because a large modelling application may run for days. Over this time various workstations may fail, including the one executing Nimrod. It is also possible to run multiple copies of Nimrod concurrently, allowing simultaneous execution of experiments with different parameter settings, or even different modelling applications.

Rather than assuming a shared file system, Nimrod copies files between systems before and after execution of the program. Thus, when a job is started, the data files are copied to the target system. It is also possible to copy in the executable image for the application, thus it is not necessary to prepare the target system prior to use. When the program terminates, the output files are copied back to the host system.

Most of the Nimrod system is common across all applications. However, the graphical user interface and the control scripts vary depending on the application and parameters. Accordingly, a mechanism is provided for building a new application from a simple description of the parameters and the necessary scripts for running the code. The latter scripts are sufficiently general to make it possible to send and compile the source for the simulation code on the target system.

Figure 4 shows some sample applications of Nimrod, which will be discussed in more detail in Section 5. In this screen dump, Nimrod is being used to control two separate applications. For each application, a generation screen is used to specify the values of parameters. In the laser experiment these are the properties of the model (Non-doppler/Doppler, Isotype1, Isotype2, Linear and Circular), the Detuning range and the Laser Intensity range. In the pollution model these are the model name (SAI, GRS or MCRAE), the emissions inventory (1980, 1990, 2000 or 2005), the scenario (a or c) and the NOX and ROC control steps. The status of all of the pollution jobs can also be seen as well as the status of an individual job. The icons represent the status of the job, such as waiting, running, suspended or finished. The status screen shows 30 jobs out of 403 complete, with 8 currently executing.

Having given an overview of the operation of Nimrod, the next section discusses some of the implementation details.

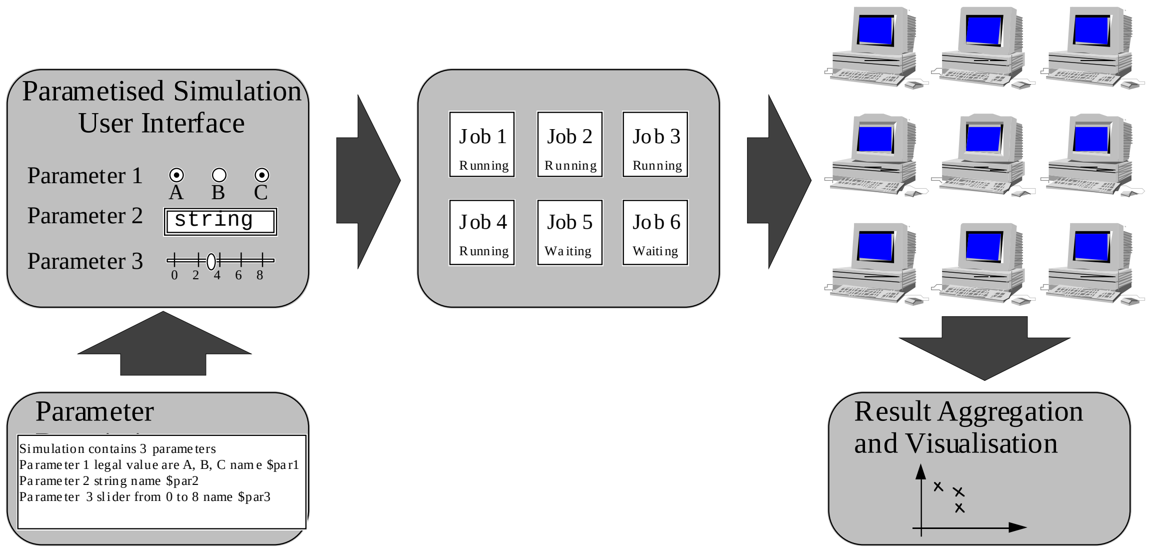

Parametised Simulation User Interface
Parameter 1
A B C
Parameter 2
string
Parameter 3
0 2 4 6 8
Job 1 Running
Job 2 Running
Job 3 Running
Job 4 Running
Job 5 Waiting
Job 6 Waiting
Parameter
Simulation contains 3 parameters
Parameter 1 legal value are A, B, C name $par1
Parameter 2 string name $par2
Parameter 3 slider from 0 to 8 name $par3
Result Aggregation and Visualisation

Figure 2 - Overall structure of Nimrod

4 IMPLEMENTATION DETAILS

The structure of the Nimrod implementation is shown in Figure 3. A vertical broken line divides the model into client and server sides. There are two separate servers , one providing file transfer (FT) between the client host and the server host, and the other allowing remote execution (RX) of processes on the server host.

Control of a run begins on the client side at the Job Organisation Tool (JOT). This module provides the user interface, allowing the user to create, monitor, and direct jobs running on many hosts. The Job Distribution Manager (JDM) queues the jobs that are to be run. The JDM interacts with the two servers on each remote host. The client communicates with the servers through the use of user agents. User agents are client side modules working on behalf of remote servers.

When a job is started on a remote host, any required files (such as executables and input files) are transferred to the host. The remote execution server is used to start a shell on the server host and this shell subsequently starts the executable program. On completion, output files are transferred from the remote host to the originating host.

Currently, communication across the network is provided by the Remote Procedure Call (RPC) facility of

DCE. It is possible to replace DCE with another transport mechanism, like PVM by replacing the logic marked in grey in Figure 4.

4.1 File Transfer

As discussed above, before a job can be executed, the remote system must hold a copy of the executable image and any input files. In a parametised run, some of these files may be different for each job, while other files may be the same for all jobs. Nimrod can generate specific data files for each job based in the selection of parameters for that job. There are several potential methods for making files accessible to a remote host. The simplest method is to restrict the use of remote hosts to those that share a file system through a mechanism such as an NFS mount. However, this scheme is restrictive and lacks generality.

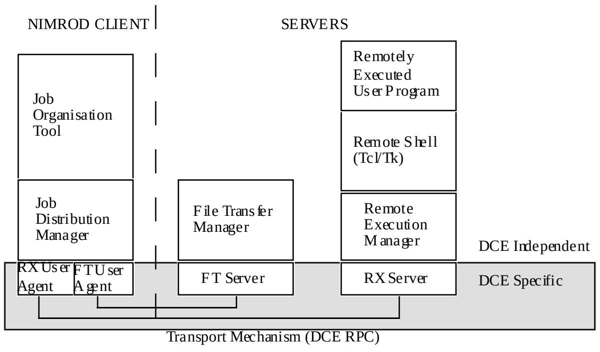


Figure 3 - Structure of Nimrod Client and Servers

A second method involves routing I/O calls (as well as most other system calls) back to the originating host. In this case only the executable must be copied to the remote host. A disadvantage of this scheme is that the overhead in system calls may increase network load unnecessarily. For example, even intermediate temporary files are accessed on the originating host, when they could be handled more efficiently on the remote host. Also, this method requires that the executable is relinked with a customised system library and that the client machine stays up for the entire time the remote job is executing. This latter requirement imposes a serious restriction when jobs run for a long time and when the network is unreliable.

A third method uses a file transfer server and requires explicit copying of the executable and all data files to the remote host. This has general applicability since any host running a file transfer server can accept jobs. In addition, if necessary, an NFS mount can be used for efficiency reasons. Nimrod uses this file transfer method because it allows efficient use of the network and provides resilience to network failures.

The file transfer service consists of three modules. The File Transfer Manager contains the server side functions that perform server activity related to file transfer. The File Transfer Server performs server initialisation and listens for requests. The File Transfer User Agent provides a high-level interface to the functions of the File Transfer Server.

The procedures in the File Transfer Manager are independent of DCE whils the FT Server is DCE dependent. In the current DCE implementation the procedures for transferring files to and from remote hosts make use of DCE pipes for data transfer. DCE pipes provide an efficient way of transferring large amounts of data through RPCs [16]. However, they require the use of callback procedures for placing data on the pipe, for removing data from the pipe, and for allocating buffer space for received data. Every time a file transfer is to be performed, a pipe is created, the callback procedures are registered with the DCE runtime library, and the pipe is passed as a parameter to the RPC. This requires that the code calling the RPC is aware of DCE pipe processing. At the same time, the callback routines used for file transfer are static. They always perform the same basic operation of data transfer between the DCE pipe and a local file.

To reduce the complexity of the Nimrod client, the File Transfer Server interface is abstracted by a File Transfer User Agent which hides the use of DCE as the transport mechanism. DCE binding handles are replaced by an abstract file server handle. The use of pipes in the RPC interface is replaced by a naming scheme for the file's source and destination. This allows applications using the File Transfer server to use other transport mechanisms simply by replacing the FT User Agent and FT Server modules. For example, to make use of a distributed file system, the FT User Agent can be rewritten to call DFS functions instead of issuing RPCs to a FT server.

4.2 Remote Execution

The remote execution (RX) server is responsible for starting a process on a remote host, and is partitioned using the same principles as the FT server. Processes can be started with arbitrary command arguments and execution environments. Terminal I/O can be redirected to a file, obtained through further RPCs, or ignored completely. Processes can be queried for status by clients such as termination and suspension. The server allows forced termination, suspension and continuation of processes providing these functions are supported by the local operating system.

In order to control the load on any given machine, a server only accepts jobs until various thresholds are exceeded. For example, if the system load is too high then the server may reject execution requests. This scheme caters for uni-processors, on which only one job may be run, and multi-processors, which can accept many jobs concurrently.

The facililities for File Transfer and Remote Execution provide the basic infrastructure required by Nimrod. The remaining modules are associated with the particular tasks of distributing the jobs in an intelligent manner and providing capabilities to monitor and control the jobs.

4.3 Remote Shell

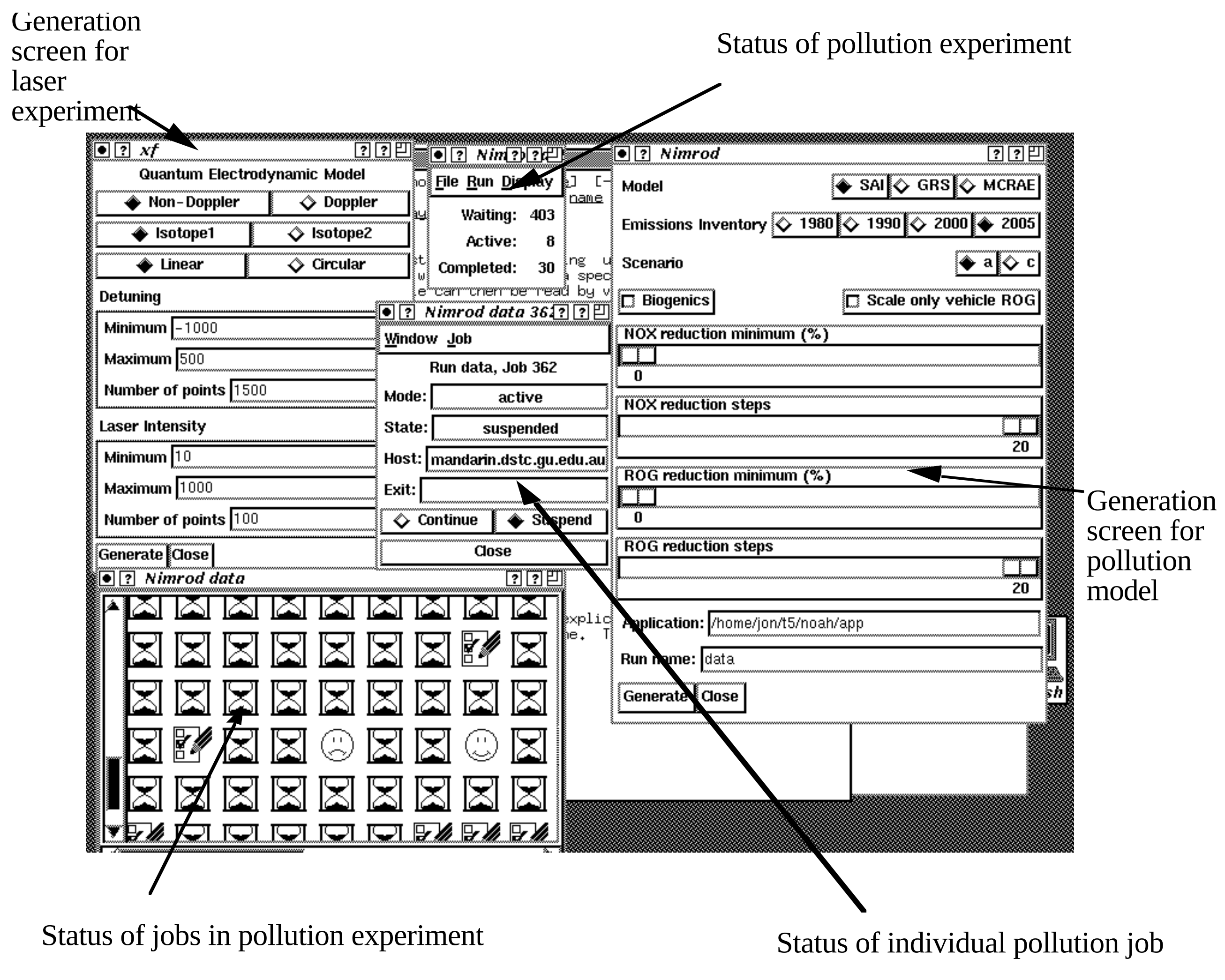
The remote execution server provides some basic information about the status of the job being executed. A remote shell is used to provide more detailed status information to be obtained than that provided by the RX server.

When a new job is started the RX server is used to run a remote shell, which is controlled using calls to the RX server's I/O RPCs. The shell can be used to start the job and wait for the termination of the job. In addition, a customised shell can monitor the intermediate progress of the job. A simple way of doing this is to monitor the size of a particular output file. A more complex, but possibly more useful method is by directly accessing variables in the job's process space. We are currently experimenting with providing this information by using a portable debugging library [17]. Thus, using this scheme, users can observe how far a job has progressed by looking at internal variables of the code, such as time step and loop counters.

4.4 Job Distribution Manager

The Job Distribution Manager (JDM) performs a similar task to that provided by existing systems such as Condor and LSF. Given a set of jobs, it runs them on under-utilised hosts on the network, and provides basic status monitoring (e.g. job is running / suspended / terminated normally or abnormally).

The task of distributing the jobs can be broken down into a number of subtasks. For each job, the subtasks to be performed are:


Generation screen for laser experiment
Status of pollution experiment
Generation screen for pollution model
Status of jobs in pollution experiment
Status of individual pollution job
xf
Nimrod
Nimrod data 362
Nimrod data
File Run Display
Window Job
Generate
Close
Continue
Suspend

Quantum Electrodynamic Model
◆ Non-Doppler ◇ Doppler
◆ Isotope1 ◇ Isotope2
◆ Linear ◇ Circular
Detuning
Minimum -1000
Maximum 500
Number of points 1500
Laser Intensity
Minimum 10
Maximum 1000
Number of points 100

Waiting: 403
Active: 8
Completed: 30

Run data, Job 362
Mode: active
State: suspended
Host: mandarin.dstc.gu.edu.au
Exit:

Model ◆ SAI ◇ GRS ◇ MCRAE
Emissions Inventory ◇ 1980 ◇ 1990 ◇ 2000 ◆ 2005
Scenario ◆ a ◇ c
☐ Biogenics ☐ Scale only vehicle ROG
NOX reduction minimum (%) 0
NOX reduction steps 20
ROG reduction minimum (%) 0
ROG reduction steps 20
Application: /home/jon/t5/noah/app
Run name: data

Figure 4 - Using Nimrod

1. Find a suitable remote host;

2. Perform any setup necessary to allow the job to run on the host (typically file transfer);

3. Start the job running on the remote host;

4. Wait for job termination;

5. Perform any cleanup necessary (copy output files to the originating host and remove created directories).

The task of finding suitable hosts is accomplished by connecting to a trading service [3] and specifying the requirements for a suitable host. Typical requirements are for a host of a particular architecture with a low current CPU load. The trading service then returns the names of hosts. This means that it is not necessary to know which hosts are present before Nimrod is started. Hosts may become available and unavailable during the simulation.

One of the problems of job queuing systems in general is how to specify problem-specific variables such as the files to be transferred from or to a remote host, the resource requirements of a remote host, and the execution command and environment. Most early queuing systems specified their own format for such information (Condor, NQS) and more recent entries have provided compatibility with one or more established formats, often with extensions to support additional functionality (LoadLeveler, DJM). The JDM uses Tcl [12] as a general scripting language. Tcl was chosen because it provides a thorough and clean interface to the C language, is easy extended to provide new commands, and is available on a wide variety of platforms. In this way, the user writes small Tcl scripts which specify the exact details of job setup and close down.

4.5 Job Organisation Tool

Most of the functionality of Nimrod is provided by the Job Organisation Tool. This module controls generation of a run. It provides a user interface which is related to the problem being solved, rather than to computational issues. The user selects the domains over which parameters of the problem will vary, and the system generates one job for each combination of parameters.

Once the jobs are generated and the run is started, the Job Organisation Tool allows monitoring of the progress of the run. Possible views of progress are:

1. a summary view - the main control window for the system is small enough to be left open while other work is being performed on a workstation. It provides a summary of the number of jobs waiting, active, and completed. In this way the researcher sees the current experiment which is being performed.

2. a job view - an array of icons, each icon representing one job. The shape of the icon indicates the state of the job (waiting, running, suspended, terminated normally or abnormally). More details about an individual job can be viewed by clicking on an icon. If only two parameter ranges are specified, then the position of the icon in the two dimensional display can be used to deduce the parameter values assigned to it.

3. an event view - this is a chronological list of major events that have occurred, including the launching and completion of jobs. This view is of more use to system administrators than normal Nimrod users.

The Job Organisation Tool allows the run to be directed by the user. This includes changing the priorities of jobs, suspending and continuing jobs, and terminating and restarting jobs. The Job Organisation Tool is also responsible for selection of parameter domains and controlling scripts, both of which are dependent on the problem being solved. A customised JOT is generated for controlling a particular system. We are currently developing a tool that allows flexible specification of the job parameters and automatic generation of the graphical user interface.

5 SOME EXAMPLES

In this section we illustrate the use of Nimrod for two different parametric experiments.

5.1 Photo-Chemical Pollution Modelling

This section concerns the distribution of some large scientific modelling programs which are used to compute the transport and production of photochemical smog within an urban airshed. The programs allow scientists in the Victorian Environment Protection Authority of Australia to simulate the smog production in Melbourne and to experiment with pollution reduction strategies. The same technology is currently being applied to other Australian cities.

Simulation has major advantages over direct physical experimentation. It is possible to study many more scenarios than would be physically possible, and it is also possible to measure and assess the effect of control strategies without the enormous expense of implementing them.

Photochemical smog modelling poses some interesting challenges. Such models consume enormous amounts of processor time and memory, and must often be performed within strict time constraints and budgets. Parallel and distributed computing technology has the potential to provide realistically priced platforms for performing such experiments.

One of the major uses of photochemical airshed models is to compute oxidant concentrations. Oxidants, such as ozone, are generated as a result of the chemical interaction between various precursors such as oxides of nitrogen (NOx) and other reactive organic compounds (ROCs) in the presence of ultra-violet radiation. Ozone is of particular importance because of its health related side effects; in Melbourne, Australia, the peak ozone levels have been observed to exceed 0.12 ppm in recent years, which is a widely adopted health standard level.

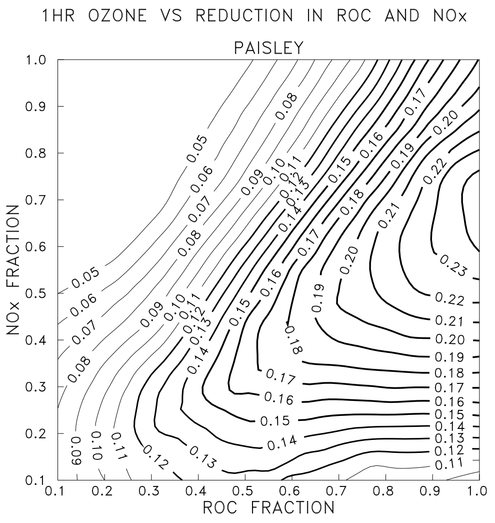


Figure 5 - Sample ozone contours

One way in which the model is used is to determine the sensitivity of the photo chemistry to various input parameters. For example, it is common to plot the peak ozone concentration as a function of NOx and ROC control, as shown in Figure 5. These diagrams then allow the modeller to determine the parameters that most affect the ozone level. The experiments are performed for a number of distinct physical locations. Unfortunately, the amount of work that is required to perform any one contour chart is enormous; if a single scenario takes about 8 hours of workstation time, then one contour chart of 50 independent runs requires 400 hours to perform. If the charts are prepared for up to 10 different weather conditions and emissions data bases, then 4000 hours of compute time are required. Thus, generation of the contour charts require access to supercomputing technology in order to produce timely results.

Rather than using conventional supercomputers to perform this work, we used a large collection of IBM RS6000 and DEC Alpha workstations, physically distributed across Brisbane (The RS6000s were part of an IBM SP2 parallel supercomputer). This utilised machines which would otherwise have been idle overnight. Thus, the work was performed without disturbing the owners of the workstations. Using this resource, it was possible to produce one ozone contour chart like the one shown in Figure 5 overnight. More details of the photo chemical pollution work can be found in [1].

5.2 Laser Atom Interaction Modelling

A detailed understanding of the collision processes between atoms, electrons and ions is of great interest in the atomic physics community. This knowledge is important in the explanation of laboratory and astrophysical plasmas, spectroscopic and surface collision physics, and scattering dynamics. Applications include fluorescent lamp and gas laser technology, surface science and atmospheric physics [2].

Of particular interest is the investigation of electron collisions with a short lived laser excited target atom. Currently there are two experimental methods for exploration of the electron-excited atom collision process. The first is the electron-photon coincidence method. A fluorescence photon from the electron excited state is detected after polarisation analysis in coincidence with the inelastically scattered electron which was responsible for excitation. The second method is the electron-superelastic scattering technique. An atom is optically prepared by a laser of known polarisation in an excited state and scattered electrons, which gain energy by collisionally de-exciting the atom, are detected.

The second method, the electron-superelastic technique, requires a detailed understanding of the laser-atom interaction as a function of laser intensity, laser polarisation and laser/atom detunings. It is possible using Quantum Electrodynamic (QED) theory, to generate equations of motion for atomic operator elements representing atomic populations in the ground and excited state, optical coherences formed between the ground and excited state by the laser and excited state coherences formed by the laser. The QED model generates closed sets of coupled, first order, linear, homogeneous differential equations. These equations are solved using numeric integration, which can be time consuming.

Once the dynamics of the atomic operators are known, it is theoretically possible to predict the line polarisation (K) for linearly polarised excitation, or the optical pumping parameter (K') for circularly polarised excitation, as shown in Figure 6. It is how these parameters vary as a function of laser intensity and detuning that is of particular interest to physicists. Introducing integration over the Doppler profile of the atomic beam introduces another complexity which further lengthens the computing time needed.

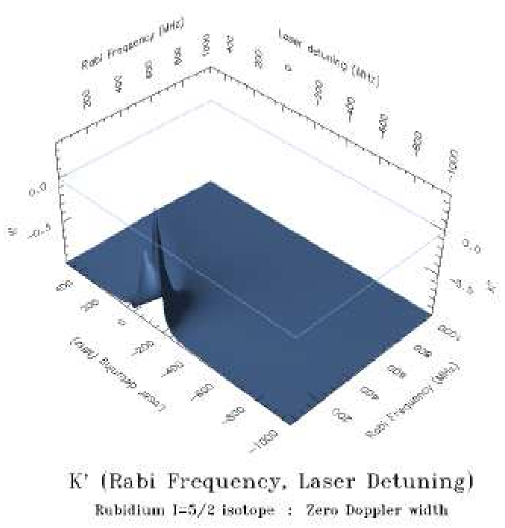


Figure 6 - Output of Laser detuning experiment

Using Nimrod, it is possible to plot K and K' as a function of laser intensity and laser detuning at the same time. A third variable, the Doppler width of the atomic beam, can be introduced and with the current software three dimensional plots can now display K and K' as functions of laser intensity, detuning and atomic beam Doppler width. This will allow the most detailed presentation of all possible data produced by these computations which covers all experimental conditions currently under investigation. As in the photo chemical pollution example, Nimrod has made it possible to distribute this application across idle machines without any consideration of parallel or distributed programming practices.

6 CONCLUSIONS

Our initial experiences with Nimrod have been very encouraging. We have successfully applied it to two different applications from different disciplines. In both cases the scientists have readily adopted the system for routine experiments. Because Nimrod shields them from the intimate details of scheduling jobs on distributed machines, and such topics as distributed file systems, they have been able to concentrate on the science underlying the experiments. In both case studies no changes were made to the application to allow it to operate in a parametric manner.

Currently, it is necessary to develop a tailored Job Organisation Tool manually using Tcl/Tk. Whilst this is not difficult for programmers with Tcl/Tk experience, it is more complex than necessary. We are currently developing a system for automatically generating JOTs using a simple specification language. This will make it possible to generate a JOT by compiling a simple description of the parameters and their ranges, together with the scripts for copying files in and out of the target system.

Whilst Nimrod is a stand alone system it is be possible to integrate its concept of automatic job generation into products such as LoadLeveler. Nimrod's dependence on DCE is limited to the software layer responsible for communications transport. This can be changed to alternative protocols with relative ease. It is worth noting that it took about one day to port Nimrod from the DEC Alpha to the IBM SP2, including the installation of DCE on the SP2. We regard this as a satisfactory outcome.

Acknowledgments

This work has been performed at Griffith University as part of the Co-operative Research Centre for Distributed Systems Technology (DSTC). The authors wish to acknowledge the support of Mr Melfyn Lloyd for proof reading a draft of this paper. Thanks go to our colleague Martin Cope, from the Victorian Environment Protection Authority and to Andrew Lewis from Griffith University. The work reported in this paper has been funded in part by the Cooperative Research Centres Program through the Department of the Prime Minister and Cabinet of the Commonwealth Government of Australia

References

[1] Abramson, D.A., Cope, M. and McKenzie, R. "Modelling Photochemical Pollution using Parallel and Distributed Computing Platforms", Proceedings of PARLE-94, pp 478 - 489, Athens, Greece, July, 1994.

[2] Anderson, N, Gallagher, J.W. and Hertel, I.V., "Physics Reports (Review Section of Phys Rev Lett)", 165, Nos. 1 & 2, 1-188, 1988.

[3] Beitz, A. and Bearman, M. "A ODP Trading Service for DCE", Proceedings of the First International Workshop on Services in Distributed and Networked Environments (SDNE), June 1994, Prague, Czech Republic, IEEE Computer Society Press, pp 42-49, 1994.

[4] Butler R. and Lusk E., "Monitors, Message, and Clusters: The p4 Parallel Programming System", PARCOMP, Vol 20, pp 547-564, 1994.

[5] Bricker, A., Litzkow, M., Livny, M., "Condor Technical Summary", University of Wisconsin - Madison, October, 1991.

[6] Ferstl, F, "CODINE Technical Overview", Genias, April 1993.

[7] Green, T. P., Snyder, J. "DQS, A Distributed Queuing System", Florida State University, March 1993.

[8] International Business Machines Corporation, "IBM LoadLeveler: User's Guide", Kingston, NY, March 1993.

[9] Kaplan, J., Nelson, M. , "A Comparison of Queuing, Cluster and Distributed Comuting Systems", NASA Technical Memorandum 109025.

[10] Litzkow, M., Livny, M. and Mutka, M. Condor - A Hunter of Idle Workstations". Proceedings of the 8th International Conference on Distributed Computing Systems. San Jose, Calif., June, 1988.

[11] Litzkow, M., Livny, M., "Experiences With The Condor Distributed Batch System", Proceedings of the IEEE Workshop on Experimental Distributed Systems, Huntsville, AL, October, 1990.

[12] Ousterhout, J. K. "Tcl and the Tk Toolkit", Addison-Wesley, April, 1994.

[13] Pancake, C., Cook, C., "What Users Need in Parallel Tool Support: Survey Results and Analysis", Proceedings of 1994 Scalable High-Performance Computing Conference, May 23-25, Knoxville, Tennessee, 1994.

[14] Revor, L. S., "DQS Users Guide", Argonne National Laboratory, September, 1992.

[15] Rosenberry, W., Kenney, D., Fisher, G. "Understanding DCE.", O'Reilly & Associates, Inc. Sebastopol, California,. 1992.

[16] Shirley, J. "Guide to Writing DCE Applications", O'Reilly & Associates, Inc,. 1992.

[17] Sosic, R."Dynascope: A tool for program directing", Proceedings of SIGPLAN '92 Conference on Programming Language Design and Implementation, pages 12-21. ACM, 1992.

[18] Sun MicroSystems Inc. "SunOS 5.2 Network Interfaces Programmer's Guide", Part No 801-3927-10, May, 1993.

[19] Sunderam, V, Geist, G, Dongarra, J and Mancheck, "The PVM Concurrent Computing System: Evolution, Experiences and Trends", Journal of Parallel Computing, 20(4), pp. 531 - 546, March, 1994.