

The Laboratory Bench: Distributed Computing for Parametised Simulations

D. Abramson †
R. Susic †
J. Giddy ‡
M. Cope §

†School of Computing and Information Technology
Griffith University
Kessels Rd, Brisbane,
Queensland, 4111
{davida, susic}@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

§ Victorian Environment Protection Authority
546 Collins St,
Melbourne, 3001
mcope@tuan.cse.rmit.edu.au

Abstract: This paper concerns the use of distributed computers for solving large scientific modelling problems. In many fields, a designer may wish to explore a set of alternative scenarios. If numerical simulation is used as the experimental process, then this means executing a number of independent jobs and then aggregating the results in some way. There are currently two main ways of organising the execution on multiple computers, namely the use of remote job execution systems, or building distributed applications. This paper explores both of these, and proposes a user interface based on a *laboratory bench* metaphor. It then describes a prototype system with the advantages of both current job generation methods. The system is built on top of the Distributed Computing Environment from the Open Software Foundation. A real world example, photo chemical pollution modelling, is used as a sample application.

1994 Parallel Computing and Transputers Conference, Wollongong, Nov 94, pp 17 27.

1. INTRODUCTION

High performance computers are being used to allow scientists and engineers to simulate complex systems. Simulation allows *many* different trials to be conducted in the time required for *one* real laboratory experiment, thus increasing the productivity of the researchers. Further, it allows experiments which are either too dangerous or expensive to be simulated using advanced mathematical models. There are many examples of these systems being used routinely in industry, such as the design of aircraft, automobiles, buildings and mine sites. Also, such models are being used for performing environmental impact statements covering a range of scenarios.

Such computational simulations require very high levels of performance if they are to provide meaningful results in a timely manner. To date, supercomputers have been employed to accelerate the execution of any one simulation run; if multiple simulations are

required to evaluate a range of options then they tend to be run sequentially one after another. However, the proliferation of high powered workstations has allowed users to execute multiple independent simulations concurrently, thus reducing the time through parallel execution. This scheme is much cheaper than using supercomputers, however, the management of such jobs can be tedious. Consequently, a number of remote execution systems have been developed for queuing jobs to remote workstations. Even given the availability of such queuing systems, the management of massively parallel distributed computations is difficult.

A further disadvantage of the current queuing systems is that they do not provide a user interface which is related to the application code, but rather, they provide a view of the computing network and its resources. In this paper we discuss the concept of providing the user with an abstraction of a *laboratory bench* on which the experiments are performed, and the underlying computational resource is displayed in terms of the experiment. This model differs from current systems which are technology oriented rather than user oriented.

In line with the dramatic increase in the number of networked workstations, there has been a rise in *open distributed processing* technology. There are now a number of commercial systems for building distributed applications, i.e. applications that execute across many machines. However, to date, there are very few simulation programs which make use of this level of technology because it requires major restructuring of existing codes.

In this paper we discuss the functionality of existing systems that provide remote job execution. We also discuss the functionality of *open distributed systems* and propose a set of extensions to current remote execution packages which bridge the gap between remote job execution and open distributed processing. We then discuss how a distributed computer can be effectively used in engineering design. Finally, we illustrate the power of such a hybrid system through its application to a study of various air pollution scenarios.

2. PARAMETISED ENGINEERING COMPUTATIONS

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. This approach has been more than adequate for large mainframes, vector supercomputers and even modern workstations. 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.

For many design processes the user wishes to study the behaviour of some of the output variables against a range of different input scenarios. The following brief list gives an indication of the types of parametric studies that might be performed:

- 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.
- An environmental engineer may wish to perform an impact study on varying the amount of waste sent into a dump site.

Using existing programs to perform such studies is 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. There may be insufficient disk space to store all of the input files on one file system, so they may need to be generated incrementally. Finally, some of these modelling programs take hours to execute to produce one point in the output visualisation; if tens or hundreds of runs are required it may not be possible to perform the experiment in a reasonable time. Further, it is often not possible to tie up an expensive supercomputer for the necessary time, making accurate simulations infeasible. In the next section we will discuss the technologies which are available for providing the necessary computational resource, and how the parametric experiments can be managed.

3. 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 large. 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 that those in tightly coupled parallel machines, they cannot be used for general parallel computing unless the tasks which are executed are allocated in very large units.

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.

3.1 Remote Job Execution

There are now a number of software packages which help manage queues of work to remote computer systems. Some examples are Condor, DQS, DJM, LoadLeveler, LoadBalancer, LSF, CODINE and NQS/Exec [2, 3, 4, 5, 6, 7, 8, 10]. Whilst Condor, DQS and DJM are in the public domain, the others are commercial products. They all provide the basic service which 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 them trace their jobs as they move through the system.

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, and the replication occurs through the use of multiple jobs with different input data. The disadvantage is that users must concern themselves with the task of generating 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, 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.

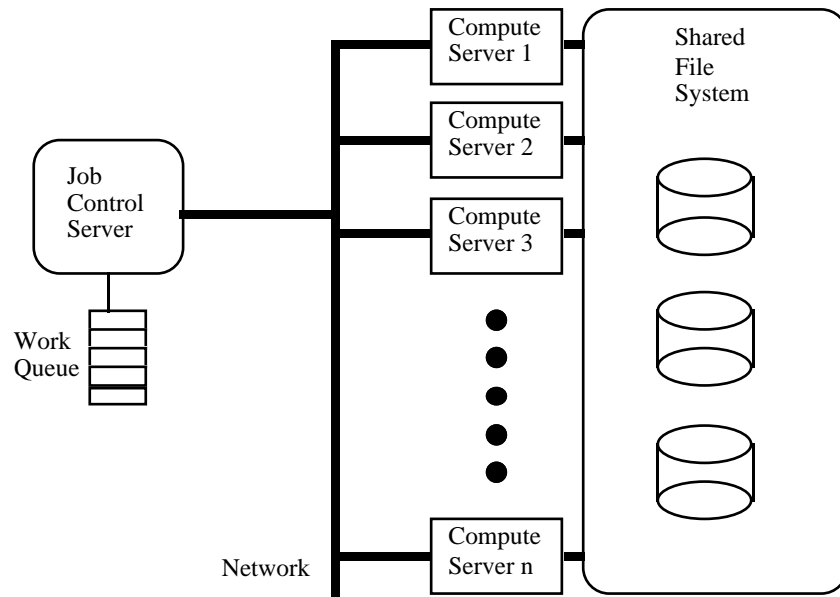


Figure 1 - a generic remote job execution framework

3.2 Distributed Applications

Distributed applications are ones in which the *programmer* is aware of the distributed nature of the computational platform when the program is written. One of the most common mechanisms provided for writing distributed computations is the *remote procedure call*. A remote procedure call has similar semantics to a conventional procedure call, except that the execution of the procedure occurs on a computer different from the caller. Parameters are passed into the called procedure and results are returned when the procedure terminates using a copy in-copy out routine. Figure 2 shows an example of a distributed application. It shows a program running on Machine 1 which acts as a *client*, and which requests services of 3 *servers* through remote procedure calls. The code on Machine 2 acts as a server for the machine 1 client as well as a client of servers running on machines 3 & 4.

This approach has the advantage that the user interface can be tailored to take advantage of the distributed mode of computation, and the metaphor of a *laboratory bench* can be provided. In this scheme, users enter the important parameters for the experiment. The work is automatically generated, distributed and the results are gathered and presented. This metaphor will be discussed further in Section 4.

A number of commercial systems are available for performing remote procedure calls within conventional programs. The Distributed Computing Environment (DCE) from the Open Software Foundation (OSF) is an open interface which is being supported by a number of computer vendors [11]. DCE allows a user to write a program which performs its work on more than one processor by executing a number of concurrent remote procedure calls. Since the calls are embedded in existing sequential languages, and since procedure calls block until the called code completes, a *threads* package is supplied with DCE. This allows a program to fork into a number of concurrent lightweight processes, each of which performs a remote procedure call and waits for the results. This arrangement is summarised in Figure 3.

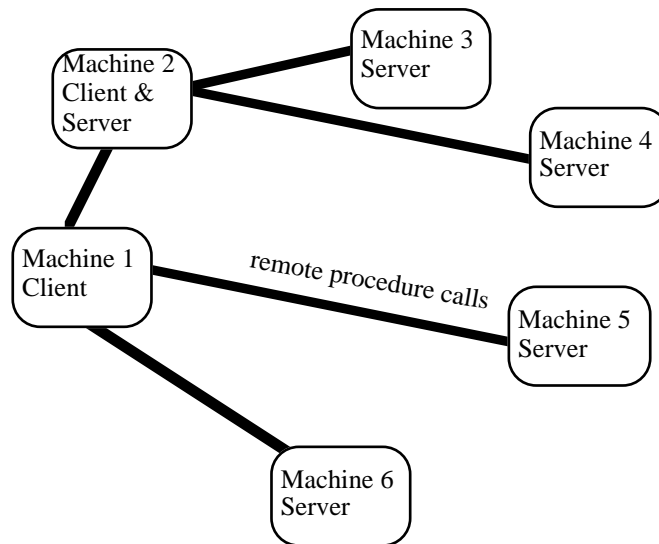


Figure 2 - sample distributed application

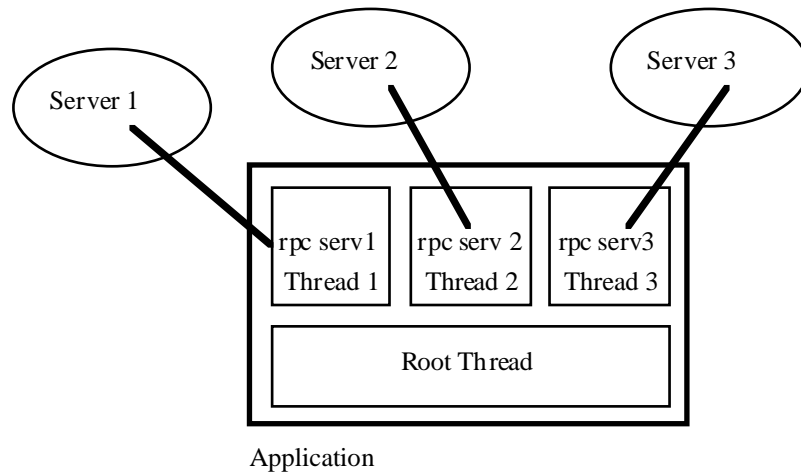


Figure 3 - Executing multiple concurrent RPCs within a distributed application.

The DCE approach is very attractive because it makes it possible to hide the underlying concurrency from the user whilst taking advantage of many idle workstations. However, it requires significant changes to the application code, which may be difficult or impossible. Many existing engineering applications are distributed as executable binaries and cannot be modified. Another disadvantage of DCE is that it has been designed to interface with C. Most high performance scientific and engineering codes are written in FORTRAN, and it would be very difficult to make use of DCE Threads and RPCs from within a FORTRAN program.

4. A NEW DISTRIBUTED REMOTE JOB CONTROLLER

4.1 A Laboratory Bench Metaphor

As discussed in the previous section, there are two main ways of using distributed supercomputers for parametric modelling experiments. The remote job execution systems have the advantage that they do not require any changes to the modelling programs. However, they provide a very technically oriented user interface and are structured in terms of the computational resources rather than the modelling exercise. Further, the user is responsible for setting up the appropriate directories with the files for each job and managing the distribution to machines, which can be a laborious and error prone process.

On the other hand, using distributed computing systems like DCE allow the user program to present a view of the experiment which matches the overall requirements. Further, the code which is necessary to access the computational resources is hidden in the program. The application is responsible for managing the addition of new resources and withdrawal of others, and this can, to a large extent, be hidden from the user. However, it has a major disadvantage that the application code must be modified to incorporate the DCE constructs. Given the large base of existing scientific code, the cost of modifications might be prohibitive.

To provide a better environment for parametric simulations, we are currently developing a user interface and underlying modelling system which presents the user with a *laboratory bench* metaphor. This metaphor provides the advantages of both the remote queuing system and distributed computing environments. In this scheme, as illustrated in Figure 4, jobs are first prepared, processed and then the results are aggregated and visualised for interpretation. The computational resources required to perform the experiment are hidden from the user, who is responsible only for specifying the input and output requirements.

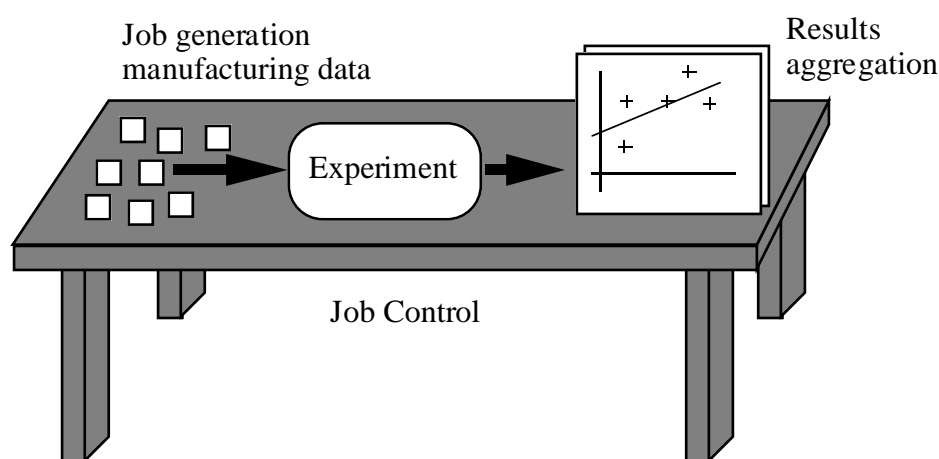


Figure 4 - A metaphor for a laboratory bench

The bench is built in a way which allows it to take advantage of distributed systems like DCE, whilst not requiring modification to the application code. It contains three separate functions, namely, job generation, job control and result aggregation. Job generation and job control make use of graphical user interfaces to provide the user with an application view of the experiment, and use DCE to perform the work distribution. However, the application code is treated as a stand alone program executed via a *remote execution server*. Result aggregation has the knowledge of the distributed nature of the computation, but is separate from the original application code.

4.2 Job Generation

Jobs are generated as a by-product of the different parameter settings. The cross product of all legal parameter values is generated by the system, and each element of the resulting set is processed as an independent job. Figure 5 shows a job generation screen from the tool that we are constructing, containing examples of the different parameters. In this example, the simulation has been designed to model the process of photochemical pollution generation. The screen contains parameters which control the choice of model, the emissions inventory data base, and the number of simulations to run. The latter is controlled by specifying that a number of simulations are required for different levels of the species in the emissions inventory, in this case Nitrous oxides (NOX) and Reactive Organic Compounds (ROCs). This is discussed further in Section 5.

Most modelling applications make use of files for input and output of information. The job generation phase is responsible for sending the input files to the target computer, and

arranging for execution of the code via the job control mechanism. It does this via a remote file transfer server on the target system. Further, it launches the application with the particular parameter settings which correspond to the job, thus the user is free of the responsibility of guaranteeing that the correct files are available for the particular instance of a job.

To test and verify the laboratory bench metaphor, we have built a prototype job generation interface for performing parametric studies of photochemical pollution modelling. We are currently building a generic tool which will generate jobs together with a user designed graphical user interface.

4.3 Job Control

If the jobs are to be distributed to multiple work stations or supercomputers concurrently, then a queuing system must be capable of generating jobs, possibly created on demand to save disk space. As discussed in section 2.1 many queuing systems exist. However, the current systems suffer from two disadvantages:

- They do not use an open distributed processing framework, and are mostly designed for UNIX based operating systems
- They are not integrated into a laboratory bench metaphor, thus users are too concerned with the operational aspects of the computational experiment, such as job generation and result display.
- Jobs are tracked by a job identifier which may have no relationship to the various parameters

We have built a job queuing system which is based around a DCE client for dispatching jobs and a DCE remote execution server for each available machine. Because the DCE specific information is contained in this module, the application can execute as though it is a batch process, and is unaware of its exact location.

Figure 5 shows a sample screen which displays the status of each of the jobs generated. Global statistics, such as the number of jobs which are completed is shown, as well as the status of each job. A job is either waiting for execution, running or complete, as indicated by different icons. In this example, the display is organised so that the two parameters are represented along the X and Y axes of the display, thus it is possible to see which simulations are complete. By clicking on individual job icons, more information can be displayed.

Different job selection algorithms are available, so it is possible to either execute jobs in the order of generation, or in a random order. The latter is useful because it allows aggregation of results to be performed before all jobs have completed, and a partial result can be displayed. In the example considered in this paper, this has the advantage that some preliminary results can be displayed without waiting for all of the jobs to complete. Accordingly, the user may terminate the remaining jobs and begin another experiment.

4.4 Result aggregation

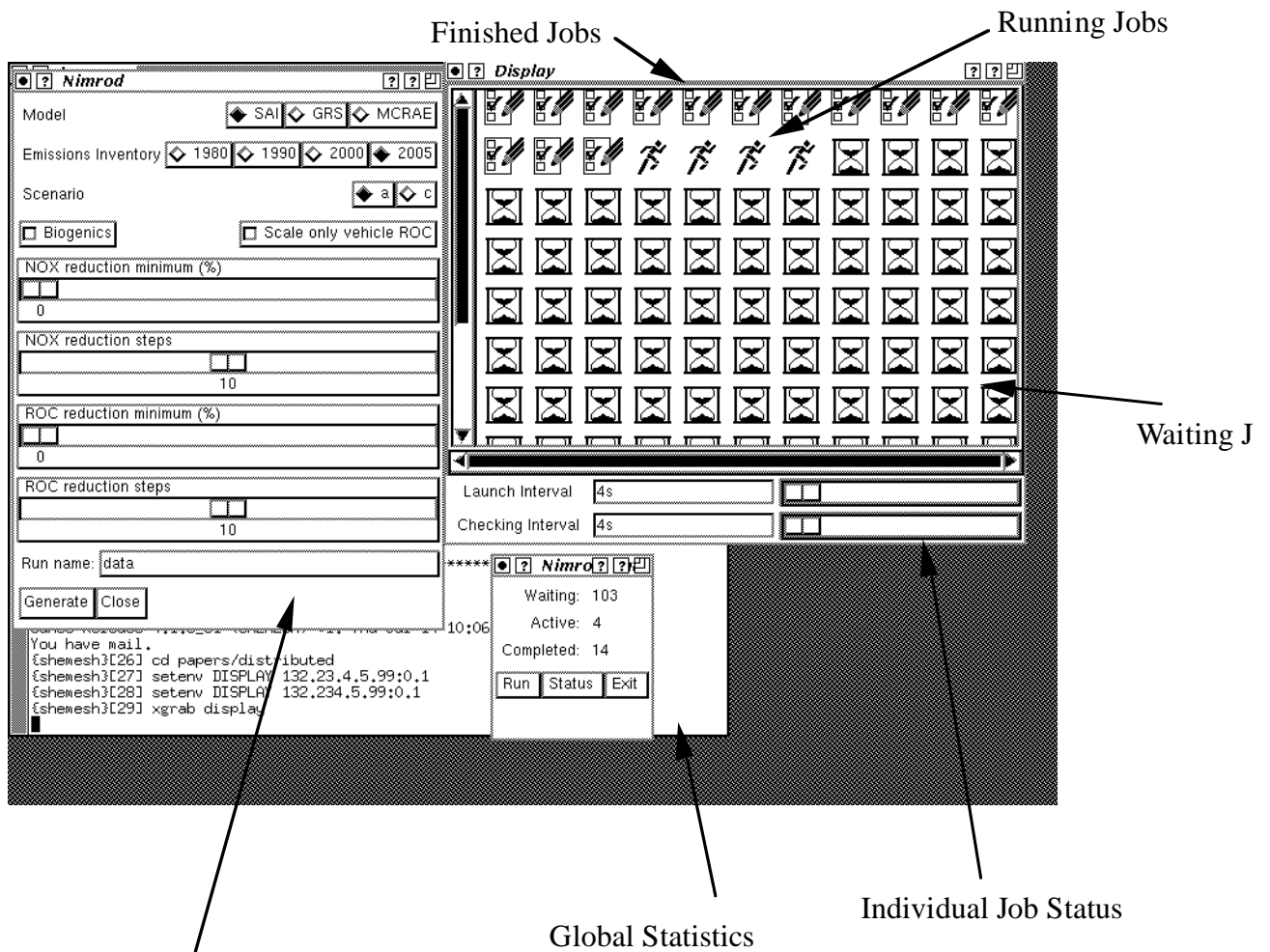
When all of the simulations are completed, the results must be presented in an appropriate manner. The output format for any particular experiment varies depending on the results and how they are to be interpreted, and thus a large amount of flexibility is required in the way that results are presented. For example, in the pollution modelling work we have performed, a simple two dimensional contour diagram is sufficient. However, for some experiments, a three dimensional colour visualisation may be the best way of presenting the results. Accordingly, the framework we have built allows the user to supply a script which is executed, and which displays the data in an arbitrary way.

5. PHOTO-CHEMICAL POLLUTION MODELLING - AN EXAMPLE

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 work was conducted in 1992 as part of a collaboration between the Victorian EPA, the CSIRO Division of Information Technology and the Royal Melbourne Institute of Technology, and serves as an illustration of the power of the laboratory bench metaphor described in this paper.

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.



Job Generation Screen

Figure 5 - Sample generation screen and status screens

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 (NO_x) and other reactive organics (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.

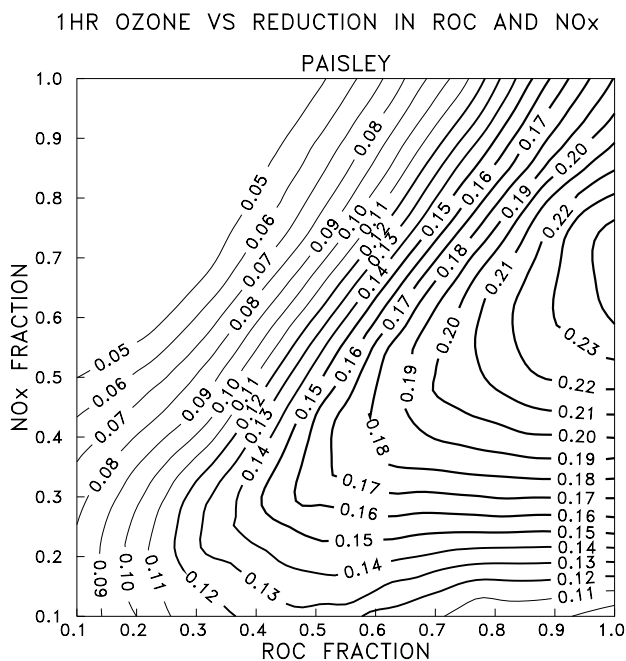


Figure 6 - Sample ozone contours

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 NO_x and ROC control, as shown in Figure 6. 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 requires 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 workstations, physically distributed between Melbourne and Canberra. This provided about 30 Sun Sparc Stations, and 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 6 overnight.

The project described in this section developed an early version of the laboratory bench system described in this paper. This system contained a user interface similar to the one shown in Figure 5, but made use of UNIX remote execution commands rather than DCE. The study indicated that the laboratory bench metaphor was extremely valuable, and allowed the EPA to evaluate a number of alternative pollution reduction strategies. The success of the project lead to the concept of a generic tool which could be applied to many

different modelling applications. More details of the photo chemical pollution work can be found in [1].

6. CONCLUSIONS

Distributed computers have the potential to provide an enormous computational resource, which can be used for performing parametrised numerical simulation experiments. However, the resource must be presented and managed in a way which relates to the original problem if it is to be used effectively by non-experts. In this paper we have discussed the state of the art in distribution of such models, and have proposed an extension to existing job distribution systems.

The laboratory bench metaphor proposed in this paper has been developed using the Distributed Computing Environment. This is an open interface for writing distributed systems. Consequently, the system can make use of a wide range of platforms transparently. The system utilises both of the current methods of performing distributed supercomputing, and thus gains the advantages of both schemes.

We are presently using the system as part of a photochemical pollution modelling study and thus the job generation facility is tailored for this application. The next phase of the project will develop a generic interface which will allow a wide range of engineering applications to be executed.

Additional issues such as security, checkpointing and job migration will be addressed in future research.

7. 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. We also acknowledge the contribution of Rowen McKenzie of RMIT for his work on a version of the laboratory bench system.

8. 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] Bricker, A., Litzkow, M., Livny, M., "Condor Technical Summary", University of Wisconsin - Madison, October 1991.
- [3] Ferstl, F, "CODINE Technical Overview", Genias, April 1993.
- [4] Green, T. P., Snyder, J. "DQS, A Distributed Queuing System", Florida State University, March 1993.
- [5] International Business Machines Corporation, "IBM LoadLeveler: User's Guide", Kingston, NY, March 1993.
- [6] Kaplan, J., Nelson, M. , "A Comparison of Queueing, Cluster and Distributed Computing Systems", NASA Technical Memorandum 109025.
- [7] 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.
- [8] Litzkow, M., Livny, M., "Experiences With The Condor Distributed Batch System", Proceedings of the IEEE Workshop on Experimental Distributed Systems, Huntsville, AL, October, 1990.

- [9] 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.
- [10] Revor, L. S., "DQS Users Guide", Argonne National Laboratory, September 1992.
- [11] Rosenberry, W., Kenney, D., Fisher, G. "Understanding DCE.", O'Reilly & Associates, Inc. Sebastopol, California. 1992.