Adding Grid Computing Capabilities to an Existing Modelling Framework

[1]Davis, G.P., [1]R.J. Bridgart, [1]T.R. Stephenson, and [1]J.M. Rahman

[1]CSIRO Land and Water, CRC for Catchment Hydrology, E-Mail: geoff.davis@csiro.au

Keywords: *Grid Computing; Modelling frameworks.*

EXTENDED ABSTRACT

Modelling frameworks are constantly being developed and expanded to take advantage of new computing technology. One such framework undergoing expansion is The Invisible Modelling Environment (TIME) (Rahman *et al* 2003). TIME is a software development framework for creating, testing and delivering environmental simulation models.

With increasing demands being placed on computer systems by complex models, a solution was needed to address the problem of ever increasing execution times.

A number of options were available to reduce long runtimes in TIME. Considering the options, a Grid Computing solution was chosen as the most viable because it could provide the greatest return on investment.

A computational grid can be formed by the networking of existing personal computers and the installation of specialist control software. Existing local area and wide area networks can also be utilised as part of a grid with no changes to hardware configuration.

The nature of some of the processing being done within the TIME engine lends itself to performance optimisation through grid computing. Tasks to be run on a grid need to be able to be broken down into small sub-problems and each of the sub-problems needs be a computationally intensive operation rather than a data intensive operation. The sub-problems also need to be able to run in parallel rather than in series as is generally the case for applications that run on a single machine.

Although a number of grid computing technologies exist, the Alchemi .NET Grid Computing Framework was the framework of choice for TIME. Alchemi is an open source software framework that has been specifically designed for the fast and efficient harnessing of the computational power of networked machines. Although only in the early stages of development when first adopted, its flexibility and potential made it the best choice for integration into TIME.

The Distributed Invisible Modelling Environment (DIME) is the extension to TIME that allows users to transparently distribute model execution over a network. It is the component which links TIME and Alchemi.

DIME includes support for the distribution of global optimisation routines present in TIME and for the batch processing of models.

The performance of the DIME component was analysed at each functional stage of development with particular attention to load and performance testing.

The integrity of the grid was maintained through load testing. Load testing involved modifying the grid composition during execution and simulating network failures to ensure stability.

The performance tests demonstrated the effectiveness of distribution under certain conditions. These kinds of tests are essential in producing guidelines for when a grid should be used for model processing. Although incomplete at this stage, our preliminary performance results look very promising.

1. INTRODUCTION

In recent years there has been considerable activity in the development of model building frameworks that take advantage of new technologies. One such framework is The Invisible Modelling Environment (TIME) (Rahman *et al* 2003).

TIME is a software development framework for creating, testing and delivering environmental simulation models. TIME differs from other modelling frameworks in a number of ways, particularly in its use of metadata to describe and manage models. This gives flexibility to components that manage data and models, recognising that one approach does not necessarily fit all applications. TIME includes a number of tools, which operate generically on models, including an automatic user interface generator and various model tools. TIME is currently being used to develop a range of modelling applications, including a library of rainfall runoff models (Perraud *et al* 2003) and a stochastic climate library (Srikanthan *et al* 2005).

A solution was needed to address the problem of long execution times, particularly during the numeric optimization and sensitivity analysis of model parameters.

There were a number of options available to reduce long runtimes. Considering the options, a Grid Computing solution was chosen as the most viable because it could provide the greatest return on investment. Another important advantage is that a Grid would enable other external organisations using TIME to create their own Grid to increase performance. This would not have been possible if, for example, our solution to long runtimes was to purchase a supercomputer exclusively for the CSIRO.

Grid computing uses the resources of many separate computers connected by a network to solve large scale problems. These are either for large computationally intensive problems where the grid is known as a Computational Grid or grids to manage large amounts of distributed data known as Data grids. Throughout this paper the term Grid or Grid computing refers to Computational Grids (Satoshi *et al* 2005).

This paper will discuss the alternatives considered, the implementation details of the chosen solution and the benefits that this new technology has brought to the TIME framework.

2. PROBLEM DEFINITION

As model complexity increases so to does the demand placed on computer systems which implement them. This increase in demand inevitably leads to longer runtimes and the slower processing of model data. In this section we describe some of the options available to developers to increase the performance of their modelling frameworks. We then detail the option we selected to implement in the TIME framework.

2.1. Increasing performance

There are several alternative approaches when attempting to increase the speed at which computational tasks are executed. These approaches fall into two broad categories—software and hardware. Software solutions include the choice of programming language, the compiler used and algorithmic optimisation. Hardware optimisation can take place at multiple levels; within a single machine—CPU speed, Front Side Bus (FSB) speed, RAM speed and onboard cache size all have the potential to increase performance. It is also possible to connect multiple machines together to provide an increase in performance for certain tasks. Supercomputers, Clusters and Grids are examples of hardware architectures designed to address the problem of performance improvement through hardware optimisation.

The grid architecture (Foster and Kesselman 1999) has several advantages over the supercomputer and cluster, the foremost of which is cost. A computational grid can be formed by the networking of existing personal computers and the installation of specialist control software. Existing local area and wide area networks can also be utilised as part of a grid with no changes to hardware configuration. The number of computers connected to a grid is dynamic and can be changed at any time. Machines on the grid need not be dedicated either—many grid systems are designed to execute jobs using otherwise unused clock cycles on client machines. The more computers connected to the grid and the faster they are, the more powerful the grid becomes.

2.2. TIME and Grid Computing

The nature of some of the processing being executed within the TIME engine lends itself to performance optimisation through grid computing. Tasks to be run on a grid need to be able to be broken down into small sub-problems and each of the sub-problems needs to be a computationally intensive operation rather than a data intensive operation. The sub-problems also need to be able to run in parallel rather than in series as is generally the case for applications that run on a single machine.

Various global optimisation routines are implemented in TIME and some are suitable for distribution on a grid. Multi-start routines which consist of a number of independent searches can easily be executed across a grid, with each search being sent to a different machine for execution rather than one processor sequentially executing N-searches.

TIME has the concept of physical models which consist of a series of variables (inputs, constants, parameters and outputs) and a time step method which, when invoked, advances the model's state by one time step. Being completely independent, models of this type are highly suitable for batch processing across a grid.

Sensitivity analysis, which involves numerous independent model runs, is another area in which distributed computing would decrease execution times.

With the power of a computational grid, large numbers of individual independent tasks can potentially be executed with vast improvements in processing speed. This has two obvious benefits in that individual jobs can be executed faster or more jobs can be executed in a given timeframe.

A coarse-grained approach to parallel processing has been adopted in the TIME environment. This kind of task separation takes place at the application level, meaning that the program itself is specifically designed to utilise the grid. Fine-grain approaches attempt to break up the task for parallel execution at a much lower level. This is more difficult but advantageous, as it is not bound to a specific application, thus can be used for a wider range of tasks with little or no change to the application.

3. EXISTING GRID TECHNOLOGIES

A number of grid computing technologies exist, each using differing technology in all aspects of their design from network communication to job scheduling. Due to the availability of varying technologies, it was deemed unnecessary to develop an entirely new grid computing framework for TIME. Two distribution frameworks were identified as potential candidates for use; Condor (2005) and Alchemi (2005).

3.1. Condor

Condor is a specialized workload management system for compute-intensive jobs. It is a well established product whose roots stem from the Remote-Unix (RU) (Litzkow 1987) project which has evolved into a cross platform distribution framework. It offers a job queuing mechanism, scheduling policy, priority scheme, resource monitoring, and resource management capability. In spite of its features and reputation, Condor was not chosen. Its underlying technologies and native language were seen as less suited for integration with TIME than Alchemi's.

3.2. Alchemi

The Alchemi .NET Grid Computing Framework was the grid computing framework of choice for TIME. Alchemi is an open source software framework that has been specifically designed for the fast and efficient harnessing of the computational power of networked machines. Although only in the early stages of development when first adopted, its flexibility and potential made it the best choice for integration into TIME.

The framework is written in the C# programming language which is the same language predominantly used in TIME. The Alchemi software is designed for application level task definition which is tied into applications through the use of the object oriented programming concept inheritance. The interface provided is extremely simple to utilise in any existing C# application. The architecture is such that jobs are sent to a manager machine whose responsibility it is to handle scheduling and manage the available resources on the grid. Client machines, known as executors, connect to the manager at will and in doing so make themselves available for use by the manager for executing jobs. By default, the manager makes scheduling decisions based on the availability of CPU resources of the executor machines. The manager is also responsible for handling exceptional circumstances such as when executor machines unexpectedly disconnect either on purpose or due to network failure. The Alchemi framework is capable of handling all scheduling and monitoring processes by default unless otherwise specified by the client application.

Microsoft .NET Remoting is the transport technology utilised by Alchemi to remotely execute code on machines in the grid. Remoting is a high level abstracted network communication protocol designed to talk between application domains, either within a machine, or over a network. Another important feature of Alchemi is its support for multi-clustering. This means that multiple independent grids can be connected and act as one at will, vastly increasing the available power of the grid.

4. DIME

The Distributed Invisible Modelling Environment (DIME) is the extension to TIME that allows users to transparently distribute model execution over a network. It is the component which links TIME and Alchemi.

The most significant decision when developing DIME was where DIME could best divide processor intensive areas of TIME. It needed to be at a point where the overall problem could be broken down into small sub-problems. Each of the small sub-problems needed to be a computationally intensive operation rather than a data intensive operation and the sub-problems also need to be able to run in parallel.

4.1. Distributing model execution

The underlying logic of the TIME framework resides within TIMECore. TIMECore provides the Model class from which all the models are extended as shown in Figure 1. In order to run models a ModelRunner class is generally used. TIMEShell and VisualTIME provide the ability to execute a model from the command line and a GUI, respectively. Both TIMEShell and VisualTIME use the ModelRunner class to execute the models. Figure 1 uses a UML diagram to illustrate the basic layout of the important classes used for model execution.

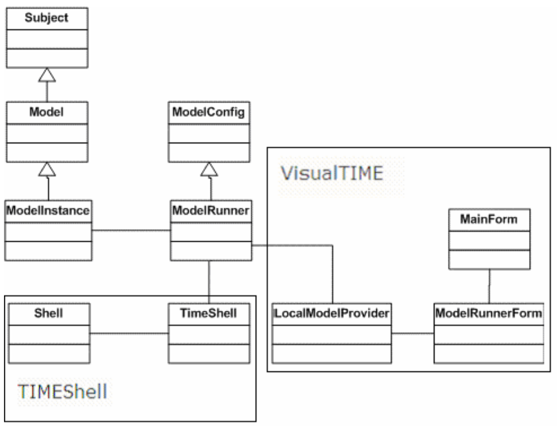

Figure 1. UML diagram of original TIME model structure

ModelInstance represents a particular instance of a developed model which inherits from the base class Model such as a Rainfall Runoff model. The basic model execution was chosen as a starting point for adding distribution. The ModelRunner class was the best point to add in the functionality since it is a common class used by most applications when running models. Directly adding the distribution here would mean that TIME would become tightly coupled with the Alchemi Grid API since the ModelRunner would have to inherit from the GridThread which is in Alchemi. This is a significant drawback.

We decided to implement an interface that is common to both ModelRunner and a grid based equivalent. This would help to make the program easier to understand and code. The interface class implements all of the "signatures" for the methods currently contained in ModelRunner and ModelConfig.

The new GridModelRunner class inherits from GridThread within Alchemi and instantiates an instance of the standard ModelRunner. The GridModelRunner also implements the interface IModelRunner as shown in figure 2.

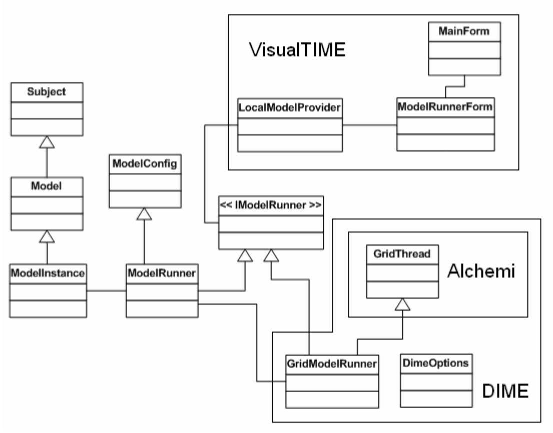

Figure 2. UML diagram of the addition of DIME to TIME

This design implements the Alchemi GridThread functionality through the ModelRunner without making the ModelRunner dependant on Alchemi.

This solution means that the TIME framework is in no way coupled to Alchemi and the DIME functionality can be loaded at runtime through its dynamic linked libraries.

In order to set the options for the Alchemi manager some metadata was added to a new class called DimeOptions (lower right corner in Figure 2). This information will be extracted using reflection in the VisualTIME Configuration GUI and integrated into the standard configuration screen.

4.2. Distributing model optimisation

A second area that would potentially benefit from distribution was model optimisation.

To understand how an optimiser works within the TIME framework it is important to understand how a model is run. A model takes a set of input data and a set of parameters and produces output based on all of the inputs and the parameters used. An optimiser takes a set of input data and a set of output data and attempts to determine which set of parameters would best produce the output data, given the input data.

In doing so, the optimiser must perform a multitude of model runs, testing different parameters during each run to find the optimal parameter set. The optimisation process is statistical in nature— the more optimisation model runs that can be completed, the higher the probability of obtaining the optimal parameter set. Figure 3 shows the IOptimiser interface class being executed by the Calibration Manager to run optimisations locally.

When designing the distributed optimisers, three realistic solutions to the problem of distributing optimisers were found. The simplest and least efficient would be just to use a GridModelRunner object in place of the ModelRunner used by OptimiserModelConfiguration, as shown in Figure 3. The advantage of this design is that it would be very simple to implement and would work with minimal effort. The downside is efficiency and speed. This design would have caused a new job to be sent off for each model run, which would have resulted in a large amount of network traffic for each model optimisation.

4.3. Categorise Search Optimisers

We estimated that conducting the distribution at a higher level would minimise the network traffic and increase the overall speed of the optimisation process.

Given that there are two major types of optimisers within TIME, single search optimisers and multi-start optimisers, we considered splitting each of the start points contained in a multi-start optimiser into a single job. This would be possible as each start point of a multi-search algorithm is independent.

Our final design was an architecture that allowed for the distribution of multiple optimisation runs over the grid. This enabled both single search and multi-search algorithms to be distributed over the grid. Instead of running a single IOptimiser under CalibrationManager; numerous IOptimisers are added to a GridOptManager class which handles all Alchemi grid functionality, shown in Figure 3. This is a superior implementation and minimises network traffic. The number of IOptimisers added to the class is defined by the user in the GenericCalibrationForm. Figure 3 shows the integration of the new DIME architecture into the TIME calibration system.

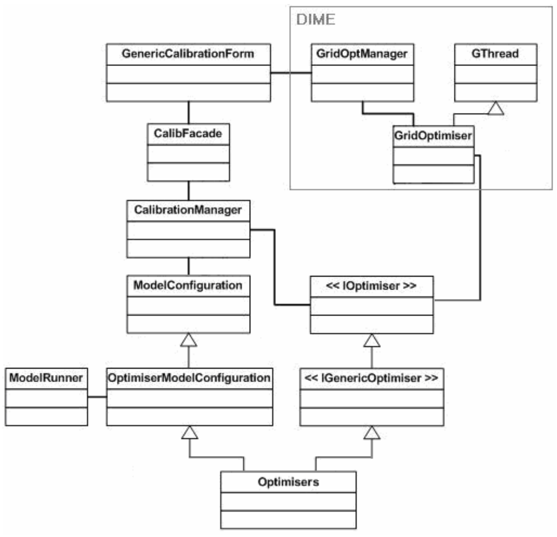

Figure 3. UML diagram of the DIME addition to the TIME Calibration system

Under the new architecture, when the optimisation is being distributed, the IOptimisers are passed to the GridOptManager which then puts them into the GridOptimiser class. The GridOptimiser class is used as a shell to hold the IOptimiser and inherits from GridThread (which enables the distribution of objects in Alchemi). Once loaded with IOptimisers, the GridOptManager then proceeds to distribute all of its GridOptimsiers and await the results. When the results are returned, the GridOptManager sorts the results, and passes the IOptimiser with the best parameter set back to the GenericCalibrationForm for display.

4.4. Batch Runner

It was necessary to create a specialised batch runner to handle the interaction with DIME and Alchemi, since the ability to send a large number of models to be run in parallel would become common functionality, necessary for a number of different applications such as sensitivity analysis.

The batch runner was set up to run many models at once, either locally or over the grid. Its specific requirements were to:

- allow fields to be set with lists or values
- determine combinations of model runs
- minimise network traffic.

The BatchRunner class was able to use IModelRunner, rather than ModelRunner or GridModelRunner, which enabled the BatchRunner to be run locally or across the grid.

The main problem in developing BatchRunner was in the class implementation. BatchRunner needed to be able to operate on any model. As models can have any number of inputs and parameters it was a problem finding all possible combinations of model inputs and parameters. We wrote a recursive function to iterate through all combinations of model inputs and parameters.

This provided an elegant solution to the problem. The function ensures that we can determine all possibilities of any number of changing variables.

The BatchRunner will fit into the exiting architecture as shown in figure 4.

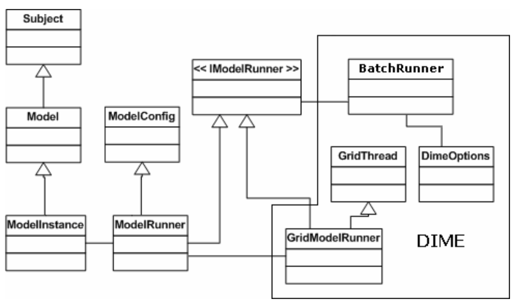

Figure 4. UML diagram of BatchRunner design

TIMEShell was modified to use the BatchRunner and a new interface was written to allow VisualTIME to use the BatchRunner and distribute batch jobs.

5. ANALYSIS OF PERFORMANCE

The performance of the DIME component was analysed at each stage of the development. Each release was reviewed with particular attention to load and performance testing (Bridgart *et al* 2004).

5.1. Load testing

The load testing involved having multiple users and executors connecting to the one manager submitting jobs, either completing or cancelling jobs, disconnecting and reconnecting from the grid, and running large jobs on all the available executors. The largest grid assembled for testing was a 32.1 GHz grid with 12 desktop machines that were in use by the majority of their owners while the load testing was taking place.

5.2. Performance testing

The purpose of the performance testing was to quantify the level of improvements gained by executing jobs over the grid.

Before performance testing began it was necessary to have a consistent testing environment. A small scale grid with five computers, not in use for any other purpose, was set up as the testing grid. The testing grid was in a standard working environment with a total of 8.613 GHz capacity (Davis 2004). There were five executors running with the Alchemi manager running on a separate machine. These tests were done using the BatchRunner through TIMEShell.

The model used to test the batch runner system was purpose built, designed purely to use excessive CPU resources and reference small data sets. The graph in Figure 5 shows the number of model runs that were run both over the grid and locally.

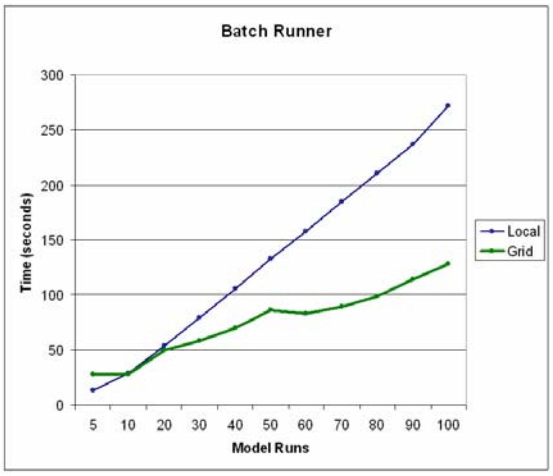

Figure 5. Performance Results (Davis 2004)

As can be seen in Figure 5, when more than 20 model runs were required, the initial overhead of using the grid was outweighed by the benefits of parallel processing gained by the grid. When executing a low number of runs, the overhead of using the grid exceeds the benefits, and as a result it was more efficient to run low numbers locally (Davis 2004).

The performance gains from using the grid were significantly greater than first expected across all tests. This includes the performance tests run on the grid optimisers which, unlike the batch runner, used real-life models and data. Gains were seen with a lot fewer runs than predicted, thus the overhead of using the grid was not as high as first anticipated.

It is worth noting that in spite of these promising results, the system has not yet been fully tested over a wide range of models. Further analysis will be required to develop strong guidelines for modellers wishing to use the grid capabilities in TIME regarding when it is appropriate to distribute work.

6. CONCLUSION

The resulting extension to the TIME framework, DIME, proved to be an effective and efficient distribution system which was able to be successfully deployed. The Alchemi API was easily able to be utilised, due to its flexibility and simple design. Given the nature of TIME and its various components, the use of a computational grid proved effective in reducing operational runtimes. The system created not only addresses the needs identified in this paper but has the potential to be expanded into other areas of the framework. Additions to the system can be made as the framework is expanded and other areas are identified as being computationally intensive.

The system has successfully grid-enabled TIME without tightly coupling TIME to the Alchemi API. The grid was shown to be robust and non-intrusive when deployed under standard operating conditions, allowing for the seamless decrease in execution times for effected TIME applications. This kind of project and research into distributed computing presents opportunities for future scientific applications.

Ongoing developments in these areas continually increase the potential impact of environmental simulations. Cost effective harnessing of existing computer infrastructure enables an increase in computational power, for larger and more complex models. This can be a valuable resource which can be utilised by the modeller.

7. ACKNOWLEDGMENTS

The DIME project was initially developed by Software Engineering students from the University of Canberra at the Commonwealth Scientific and Industrial Research Organisation (CSIRO) for the Cooperative Research Centre for Catchment Hydrology (CRCCH). The distribution framework is provided by Alchemi which is an open source project developed at the University of Melbourne. The DIME project is now being run by software engineers at CSIRO.

8. REFERENCES

Alchemi (2005), Alchemi .NET Grid Computing Framework. http://www.alchemi.net/ Last Accessed August 6, 2005.

Bridgart, R.J., G.P. Davis, and T.R. Stephenson, (2004), *DIME: Final Report*, Honours report, University of Canberra, www.toolkit.net.au/dime/dimeFinalReport.pdf Last Accessed August 9, 2005.

Condor (2005), Condor: High Throughput Computing http://www.cs.wisc.edu/condor/ Last Accessed August 9, 2005.

Davis, G.P. (2004), *TIMEShell: Final Report*, Honours report, University of Canberra, www.toolkit.net.au/dime/TIMEShellFinalReport.pdf Last Accessed August 9, 2005.

Foster, I. and C. Kesselman, (eds.) (1999), *The* Grid*: Blueprint for a Future Computing Infrastructure*. Morgan Kaufmann: San Francisco, CA.

Litzkow, M.J. (1987), *Remote UNIX – Turning Idle Workstations into Cycle Servers.* USENIX Conference Proceedings, summer, 1987, pp. 381-384.

Manish, R. and A. Craif, (2005), *Scanning the Issue: Special Issue on Grid Computing*, Proceedings of the IEEE, Vol. 93, No. 3, March 2005.

Perraud, J.-M., G.M. Podger, J.M. Rahman, and R.A. Vertessy, (2003), A new rainfall-runoff software library, Proceedings of MODSIM 2003, (4), 1733-1738.

Rahman, J., S. Seaton, J-M. Perraud, H. Hotham, D. Verrelli, and J. Coleman. (2003), *It's TIME for a New Environmental Modelling Framework*, MODSIM 2003.

Satoshi, M., S. Sinji, A. Mutsumi, S. Satoshi, U. Hitohide, and M. Kenichi, (2005), *Japanese Computational Grid Research Project: NAREGI*, Proceedings of the IEEE, Vol. 93, No. 3, March 2005.

Srikanthan, R., F.H.S. Chiew and A.J. Frost, (2005), Stochastic Climate Library User Guide, CRCCH Toolkit, www.toolkit.net.au/scl, Last Accessed August 2005.