

Interprocess Communication in GriddLeS: Grid Enabling Legacy Software

David Abramson and Jagan Komineni
School of Computer Science & Software Engineering,
Monash University, Clayton
Australia, 3800
{davida, jagan}@csse.monash.edu.au

Abstract

The GriddLeS environment discussed in this paper facilitates the composition of arbitrary grid workflows from legacy software. The underlying belief is that it is possible to take existing programs and grid enable them by providing a high level tool that facilitates the composition of complex systems from smaller, working components. These components are connected by a novel mechanism called GridFiles, devices that behave either as local files, remote (and possibly replicated) files or direct sockets. The paper discussed the type of grid application we envisage followed by a discussion on how GriddLeS allows a user to build such programs without source modification. We then describe our current implementation of GridFiles, which is built on top of Globus, and illustrate its use in a distributed mechanical engineering computation distributed across a small global grid.

1. Introduction

Computational and data Grids couple geographically distributed resources such as high performance computers, workstations, clusters, and scientific instruments. Accordingly, they have been proposed as the next generation computing platform for solving large-scale problems in science, engineering, and commerce [12][13]. Unlike traditional high performance computing systems, such Grids provide more than just computing power, because they address issues of wide area networking, wide area scheduling and resource discovery in ways that allow many resources to be assembled on demand to solve large problems. Grid applications have the potential to allow real time processing of data streams from scientific instruments such as particle accelerators and telescopes in ways which are much more flexible and powerful than is currently available. A number of prototype applications have been built and these demonstrate that the Grid computing paradigm holds much promise [14].

Of particular interest are applications, called Grid Workflows, that consists of a number of components, including: computational models, distributed files, scientific instruments and special hardware platforms (such as visualisation systems) [22]. Importantly, such workflows are interconnected in a flexible and dynamic

way to give the appearance of a single application that has access to a wide range of data, and running on a very high performance platform. Grid workflows have been specified for a number of different scientific domains including physics [23], gravitational wave physics [24], geophysics [25], astronomy [26] and bioinformatics [27].

Much of the effort in Grid computing is being directed towards the construction of new applications, in many cases written from scratch. *We are interested in building new applications, but from legacy components. In particular, we want to leverage the billions of lines of code embodied in existing scientific and engineering codes, by stitching them together into new Grid aware applications.* Over the past 5 years we have constructed a software tool called Nimrod/G, which allows a user to migrate a particular class of applications to the Grid [1][2]. Specifically, it automates the execution of parameter sweep applications (parameter studies) over global computational grids. Nimrod is particularly novel because it supports user-defined deadline and budget constraints for scheduling computations and manages the supply and demand of resources in the Grid using an experimental computational economy [5][6]. *Thus, using Nimrod/G, we have demonstrated that it is possible to build specific Grid application very easily and quickly for a niche class of problems, namely parameter sweeps.* However, Nimrod/G cannot be used to build general grid workflows, as discussed in section 2.

The GriddLeS¹ environment discussed in this paper provides a more general environment than Nimrod, one that facilitates the composition of arbitrary grid applications from legacy software. The underlying belief is that it is possible to take existing programs and grid enable them by providing a high level tool that facilitates the composition of complex systems from smaller, working components. A user of this environment interacts with a visual, graphical manipulation language to describe the interaction between programs, data sources, and IO devices such as shared scientific instruments.

One of the more important aspects of GriddLeS is the mechanism it uses to support communication between

¹ The name stands for “Grid Enabling of Legacy Software”

components. GriddLeS supports the construction of complete applications *without* source modification to the existing components. To achieve this we have overloaded the normal IO primitives in conventional languages so they support interprocess communication as well as file operations. This allows the individual components to behave as though they are operating in a conventional file system, whilst in fact they are sending and receiving data across a distributed grid infrastructure. The mechanism, called GridFiles, is very flexible and can be implemented by a range of different techniques, from file copy to IP sockets.

To date there are very few examples of programming environments that allow legacy applications to be “Grid Enabled”, and thus to date almost all Grid demonstrators have been constructed from scratch. For example, there is much work on developing interoperability among high-performance scientific components by defining an interface definition language that supports scientific abstractions [28][29][30]. However, these techniques still require significant source code modification, and are better suited to developing new codes rather than legacy file oriented applications. A number of current projects share some of the goals of GriddLeS, such as GrADS [4], Uintah [10] and P-Grade [16], but these are work in progress and are by no means mature. Middleware software layers like Globus [12] and Legion [9] are powerful, but they only provide a set of low-level primitives. In fact, we use these to provide the lower level services required by GriddLeS. However, in general, higher-level functions must be composed using the APIs that are provided, and these must be called from within the Grid enabled application. This means that at present, in order to build a general Grid application, it is necessary to modify legacy code.

The paper begins with a description of the type of grid workflow we envisage followed by a discussion on how GriddLeS allows a user to build such programs without source modification. We then describe our current implementation of GridFiles, and illustrate its use to implement a distributed mechanical engineering computation.

2. What do we mean by a Grid Application?

The proposed uses for the Grid are varied, and thus it is difficult to describe the characteristics of all Grid applications [13]. However, the example shown in Figure 1 helps illustrate a realistic scenario in the area of atmospheric modelling. This sample application takes temperature and pressure data from a variety of instruments, such as satellites and airborne and seaborne sensors, and feeds these to a range of different numerical

models. In particular, data is assimilated into a general circulation model of the atmosphere (1), which computes the flow fields across the entire globe. This global model in turn drives the boundaries of a regional weather model (2) which produces more accurate wind vectors and temperature and pressure fields over a limited area. These values are in turn streamed into a variety of pollution models, such as a photo-chemical pollution model (3), a particle dispersion model (4) and a bush fire model (5). Each application addresses some particular aspect of the atmosphere in isolation, but when linked together they interact and provide a rich set of data ranging from weather to pollution. For example, a bush fire generates particles that must be dispersed, and also increases various precursors that affect photo-chemical pollution. If the fire is severe enough, it actually affects the regional weather. Accordingly, the different models need to interchange data at various times.

In the Grid, static data sources, such as pollution inventories and vegetation maps, required by the various computational models might be distributed geographically, but copies may be available at more than one site. This means that when models are scheduled to the various machines in the Grid, the location of the *closest* data also needs to be taken into account.

3. Existing unmodified legacy applications as components

Traditional software engineering has promoted object oriented programming as the basic method of building reusable software components. This methodology incorporates advanced techniques such as inheritance and design patterns to build a rich and powerful semantic model, and it has been applied very successfully to many large-scale projects [19].

In spite of this, such techniques are not commonly used in the development of scientific software, and many programmers prefer to use legacy languages such as Fortran and C. Rather than question the reason that object oriented technology has not been adopted widely, we pose the question as to whether computational models built from legacy languages can form the component base of Grid enabled applications.

A component in a Grid application needs to have a predefined function;

- have a well defined interface;
- have a relatively long computation time compared to the time it take to send data from one component to another and
- act as a building block for the system as a whole.

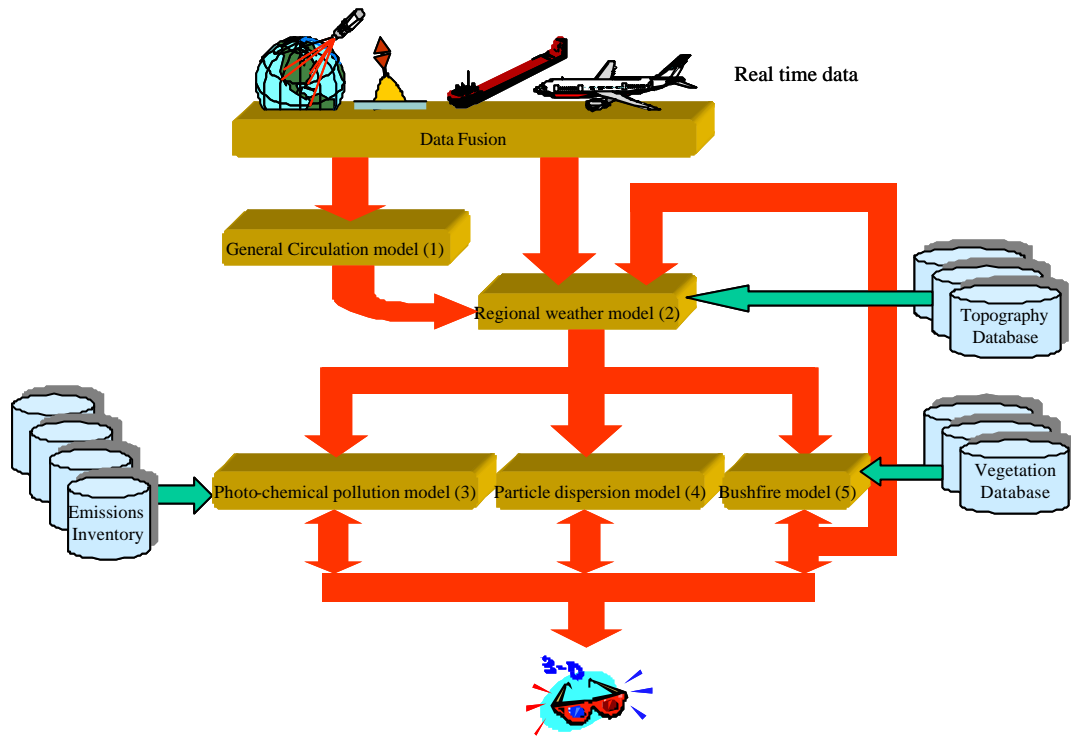


Figure 1 – Sample Grid Application for atmospheric modelling

Clearly, existing application codes possess all these attributes. An existing program has a very clearly defined function and has a well documented interface (typically through files). Further, these programs often run for hours, even on supercomputers, and so the ratio of computation time to communication time is high. This is necessary in a Grid, because the latency between nodes is often high and the bandwidth may be relatively low (compared to a dedicated local network or file system). Finally, the example in Figure 1 suggests that we can build a real application using existing computational models.

4. GridFiles: File based interprocess communication

Whilst there are many different mechanisms for supporting general distributed computing, message passing has become the dominant method for inter-process communication in parallel systems. Accordingly, the MPI standard has been widely adopted by both the hardware vendors and software developers [17]. MPI provides a set of high level, standard procedures for sending data from one process to another and for synchronising events. Not surprisingly, MPI has been adopted as the method of choice for performing communication between components of a Grid application – MPICH-G is an example of a library that

has been built on the Globus system. However, message passing has three main disadvantages when used in this mode. First, it requires the applications to be modified. This demands changes to the source code of the applications, which is inconvenient and sometimes impossible. Second, it requires scientific programmers to learn a new programming methodology, namely message passing. Third, most of the implementations of MPI are not robust, and do not support fault recovery. Thus, if a Grid application is built on a non fault tolerant MPI layer it becomes very brittle.

In [7] we proposed a new method of performing inter-process communication based on a file abstraction called NetFiles. NetFiles allows an applications programmer to write conventional programs using READ and WRITE primitives. As far as the application is concerned it accepts data via conventional READ statements, and emits data via conventional WRITE statements. However, by re-directing the IO traffic to a socket connection instead of the file system, the same statements can support inter-process communication. Thus, a READ statement behaves like a MPI_RECEIVE call, and a WRITE statement behaves like a MPI_SEND call, as shown in Figure 3. Further, if the substitution is performed in the file IO library it becomes transparent to the application.

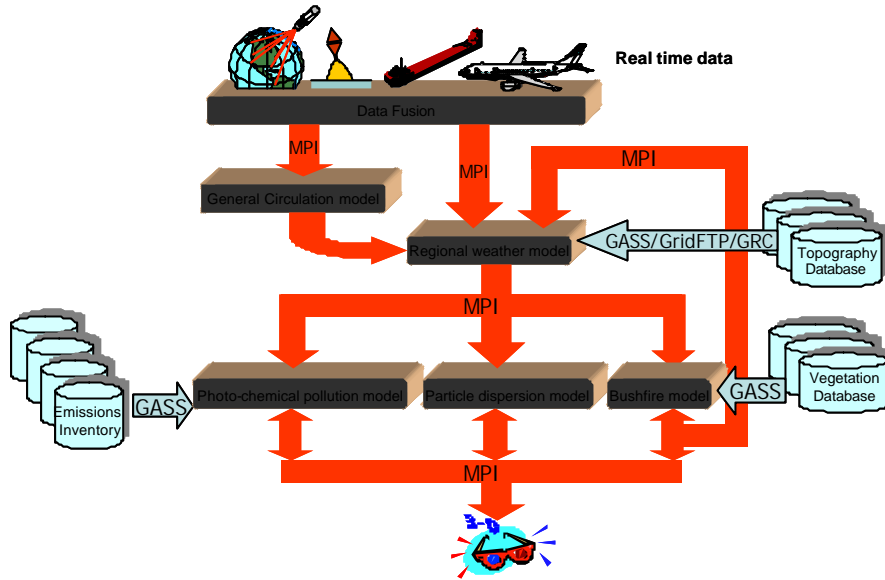


Figure 2 – Implementation using traditional technologies

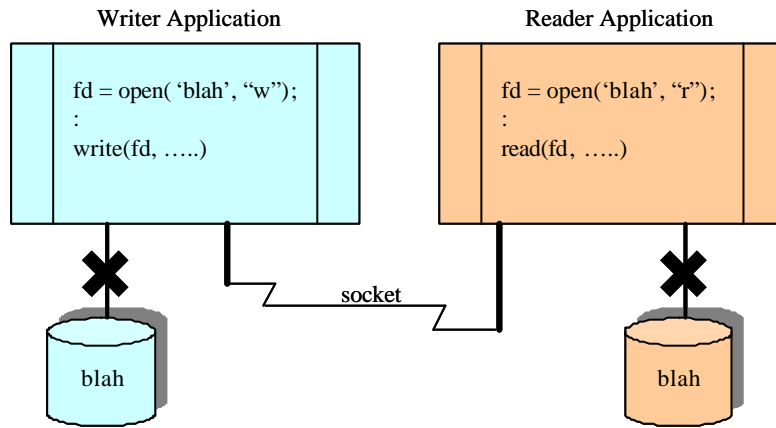


Figure 3 – NetFiles being used for inter-process communication

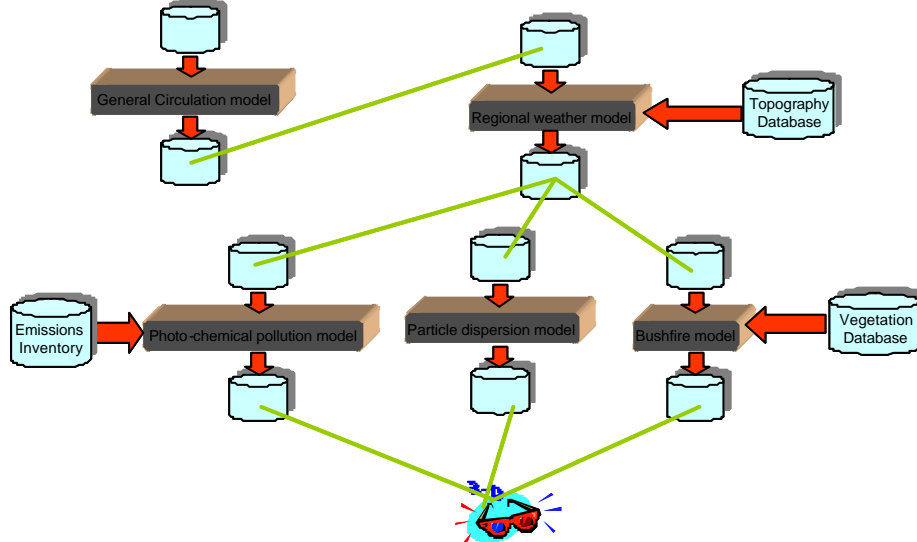


Figure 4 – Using GridFiles to link components

When implemented over a wide area, NetFiles can be extended into GridFiles. Using this approach, the individual applications believe they are operating as shown in Figure 4, reading and writing individual files, but in fact, they are working as shown in Figure 2 using message passing libraries for IPC and accessing remote data files where necessary.

This approach, whilst simple, is very powerful. It means that applications programmers can write and test components in a familiar environment, and can leave the more difficult aspects of distributed computing, such as message passing operations, to system programmers. Further, it means that a given application program can operate either as a stand-alone program, or as a component of a parallel system. We argue that using this same approach in a Grid tool kit will simplify porting of legacy applications.

GridFiles can be implemented in a number of ways. For example, in the most basic form, it is possible to create a real file at the writer end, and copy this to the file system of the reader using a standard file copy protocol like GridFTP. This mode is extremely fault tolerant, because the network need only be operational whilst the file is actually being copied. However, the disadvantage is that it allows no overlap in the reader and writer computations. Thus, a write must finish and close the output file before the reader can start. On the other hand, by mapping the read and write calls to socket connection, traditional message passing can be achieved transparently to the application. This allows complete overlap of reader and writer activity but also requires a reliable network connection even when data is not being transmitted. Finally, a hybrid of the two schemes can be implemented, in which data is sent down a socket, but is simultaneously written to a cache file. In the event that a restart is required, the cache file can be read and piped into the reader application. The location of the cache can also be varied from the writer node to the reader node depending on the requirements. Importantly, all of these configuration changes are transparent to the application.

Any given application needs to manipulate a number of files concurrently, GridFiles makes use of a “File Multiplexer” as shown in Figure 5. This routine replaces the normal file IO library for a particular language, and allows the system to redirect file IO requests dynamically to local files, remote files or remote processes. In the latter case, a file multiplexer on the writer machine is linked with a symmetric file multiplexer on the reader machine. The device handles the synchronisation of readers and writers, and thus supports quite complex interprocess communication patterns. It is also possible to cache the data being

transmitted between components, and this provides a fault tolerance not available using conventional message passing libraries like MPI. In this scenario, if one of the application components fails, or the network connecting them fails, it is possible to restart the down stream computations, reading previous traffic from the cache. Providing the programs have not altered any permanent databases, the replay is possible without informing the down stream components that they have been restarted.

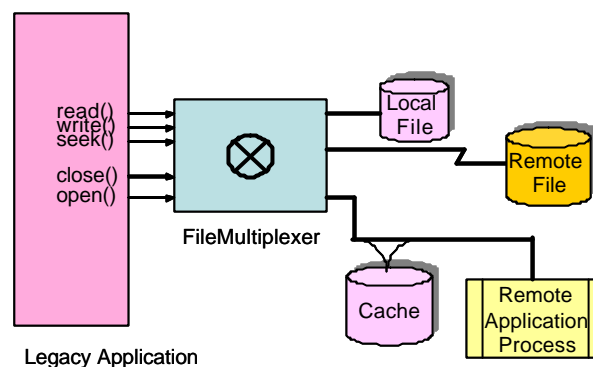


Figure 5 – File Multiplexer Process

An important issue related to the substitution of active processes for files is the file naming convention that is adopted. Typically, an OPEN statement takes a file name at run time and maps this to some sort of local process variable. When an active process is substituted for a static file, the system needs to connect the writer process to the corresponding reader process. To solve this problem we have developed a global naming scheme and have built a manager that recognises when writers and readers are referring to the same information. This service is called the GriddLeS Name Service (GNS).

5. Implementation

Figure 8 provides an architectural overview of the implementation of the File Multiplexer and associated servers. Each application is linked to its own File Multiplexer library, which contains a number of client modules. As discussed, the normal file IO primitives are intercepted by the File Multiplexer, and these are processed either by the Local File Client, the Remote File Client or the Grid Buffer Client depending on whether the file reference is for a local file, a remote file or an inter-process socket (accordingly).

The GNS Client is responsible for resolving the local file names specified in the OPEN calls, and for mapping these to either local files, remote files, remote replicated files or remote processes. The File Multiplexer treats the GNS as a read only database, and matches up multiple OPEN calls. The GNS is loaded by a separate process

responsible for configuring a grid application. Each entry in the GNS indicates what should happen when a particular file is opened on a particular resource. For example, if the file is to remain local to the resource, then the GNS simply stores the local file name. However, if the file is to be read from a remote resource, the full pathname of the remote file is stored in the GNS entry. If a Grid Buffer is required, then the local file name is mapped onto a Grid Buffer identifier.

The Local File Client simply passes the calls onto the local file system, using the file name as resolved by the GNS. The Remote File Client connects to a Grid FTP server [3] on the remote machine, and passes back blocks of the file as required. Note that the GridFTP server is a standard part of the Globus distribution, not a special component of GriddLeS. The Grid Buffer Client is responsible for implementing inter-process communication. It connects to a corresponding Grid Buffer Server on the other host, and sends blocks of data for each local WRITE call. At the other end of the socket, the Grid Buffer Client reads blocks by making calls to the local Grid Buffer Server. A cache file can be stored at either the sending end of a Grid Buffer connection or the receiving end. This provides considerable flexibility when faults occur. The buffer management code is able to support sequential read and write operations, but also if the cache is enabled, it can support out of order operations like seek(). It also supports broadcast operations, where one application may write to the buffer, but many may read the buffer. The buffer management subsystem is implemented using DiNucci's CDS library [11] and the inter-process communication is built on the Bypass library [20], which provides a framework for intercepting the file system calls.

6. A Case Study

This case study considers computer models of thin plates containing holes and subject to cyclical loading. The models assume pre-existing cracks normal to the hole profile and use the Jones method of crack dynamics to estimate the number of cycles required for these cracks to spread from an initial length to some final length [15]. Our aim is to determine the hole shapes that will maximize the life of the worst (least cycles) crack. Previous work has shown that optimizing for life in this way may give different results from optimizing for stress on the hole boundary [8]. Figure 7 shows the stress distribution in the plate for a particular hole shape.

In order to complete the computations, we need to execute a pipeline of 5 programs, as shown in Figure 6. CHAMMY takes a formula for a hole shape, depending on several parameters and generates points on the

boundary of that hole. The programs MAKES_SF_FILES and OBJECTIVE are used to transform data from one phase to the other. PAFEC is a finite element code that computes the stress tensors in the meshed design. FAST is a crack propagation code that computes the number of cycles before a number of independently placed cracks reach a certain length.

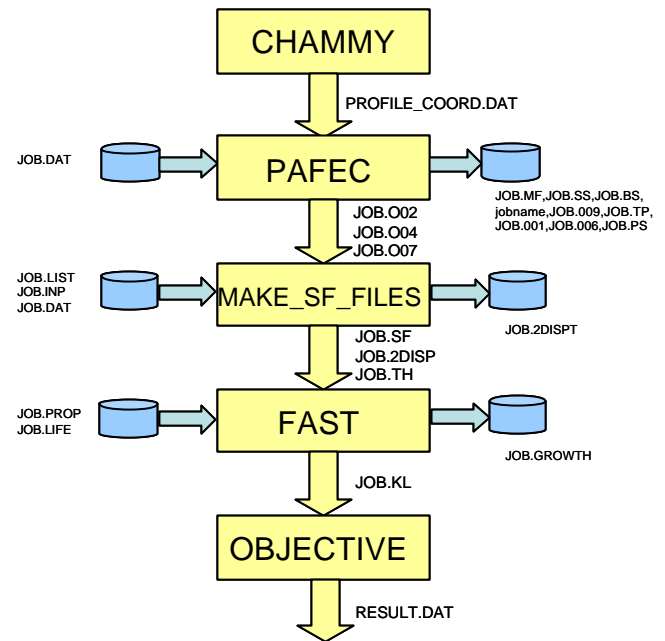


Figure 6 – Durability Pipeline

Traditionally, the entire pipeline has been executed on the one computer, with intermediate results passed using files. Importantly, some files are passed from one phase to another, whereas, other files are simply read from the file system. The final output, RESULT.DAT, contains the value for the life of the design, which is the minimum time for any of the cracks to reach a certain length. This result defines the life of the design. The GriddLeS File Multiplexer and GNS are flexible enough to map some files to the local file system, whilst linking writer-reader file chains into direct socket connections.

To test the flexibility of GriddLeS, we performed a number of experiments as shown in Table 1. In each case, the components were mapped onto different computers in an experimental grid, the details of which are shown in Table 2. We recorded the times that each component completed, and the total execution time for the application.

In experiment 1 all phases of the computation are run on the machine “jagan”, at Monash University, Melbourne Australia. In this case, GriddLeS was configured so that

local files were written and read by each phase of the computation. Accordingly, the next phase in the pipeline cannot start until the previous one is complete. In experiment 2, the programs were again run on the one machine, “jagan”, but GridFiles were used instead of real files. This time the experiment completed 10 minutes faster, mostly because the next phase of the computation could start before the previous one completed. Thus, the phases were overlapped into a pipeline. In experiment 3 all phases were run on different machines, which were physically distributed across three different countries. This experiment completed some 44 minutes faster than experiment 1. The speedup can be attributed to both the overlap in computation provided by GridFiles, and also the machines were of varying speeds. This latter point is a powerful argument in support of such a computational grid, because sometimes faster machines are available at a remote location. *The most important result of these experiments is that the changes in configuration required no modification of the software application, and this was achieved solely by changing entries in the GNS.*

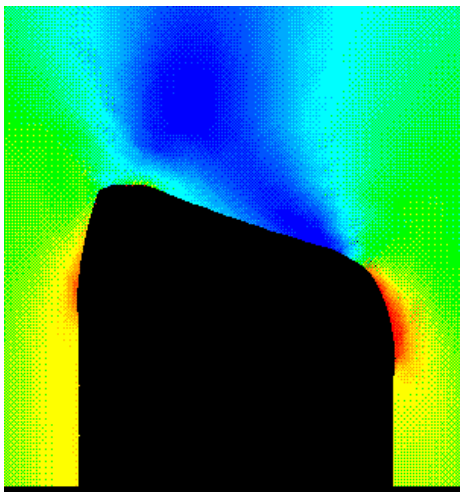


Figure 7 – Stress distribution for one particular hole shape

Exp	Assignment of tasks	Nature of IPC	Total Time mm:ss
1	All programs on jagan...	Files	99:17
2	All Programs on jagan...	GridFiles	89:17
3	Chammy on koume00... Pafrun on jagan... Make_sf_file on dione... Fast on vpac27... Objective on freak...	GridFiles	55:11

Table 1 – details of the computational experiments performed

Machine	Machine Name Location	Details
1	dione.csse.monash.edu.au Monash Univerity Australia	Pentium 4, 1500 MHz, 256 MB, Redhat Linux 7.3
2	jagan.csse.monash.edu.au Monash Univerity Australia	Pentium 3, 350 MHz, 128 MB, Redhat Linux 7.3
3	koume00.hpcc.jp Japan	Pentium 3, 1024 MB, 1400MHz Red Hat Linux 7.3, i686
4	freak.ucsd.edu UCSD United States	Athlon(tm) Processor, 700 MHz, 256 MB i386, Debian
5	vpac27.vpac.org VPAC, Australia	Pentium 3 (Coppermine), 997.488 MHz, 256 MB, Red Hat Linux 7.3

Table 2 – Machine details

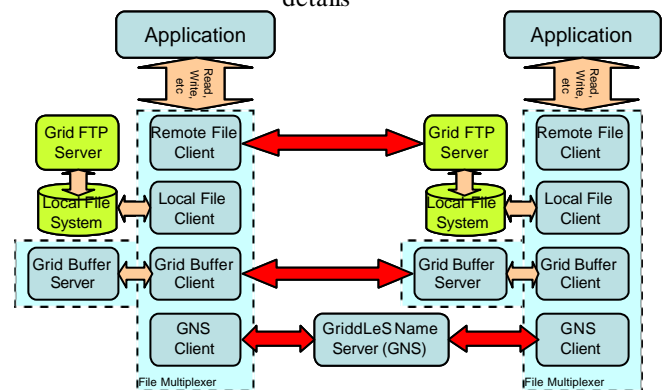


Figure 8 – Architecture of GriddLeS

7. Conclusions

This paper has discussed one aspect of the GriddLeS system, namely the way that it supports the linkage of a number of separate software components into single distributed Grid applications. We described the design and implementation of a File Multiplexer, a module that maps file system primitives into either local or remote files, or allows communication with remote processes. We demonstrated the effectiveness of the system using a small mechanical engineering case study. *The most significant conclusion of the paper is that it is possible to build quite general Grid applications without any source modification of the underlying components,*

which differs from the current state of the art in Grid computing.

GriddLeS makes extensive use of existing middleware layers like Globus to provide the low level services it needs. Thus, it should not be viewed as a replacement for Grid middleware, but rather as a layer that interfaces between legacy code and such middleware. There are many other aspects of GriddLeS that have not been discussed in this paper. For example, tools are required for specifying and composing a new Grid application. We are currently experimenting with systems like P Grade to see if it can perform this task [16]. We have not discussed replication of files, and the heuristics by which GriddLeS chooses a remote file. We have not discussed scheduling of components to resources and assignment of data sources. Clearly, there has been much work in this area, and we should be able to leverage existing schedulers like the Condor DAGman [31]. Importantly, the scheduler needs to take account of whether GriddLeS is configured to copy files or use direct socket connections, since both impose different scheduling constraints. Whilst the current implementation of GriddLeS implies a single centralized GNS, this is not required. In fact, it would be quite possible to have many local GNSs, since these really only store information about particular application configurations. We plan to explore the use of multiple GNSs in future versions of GriddLeS.

We plan to make use of the Globus Replication catalogue [18] to support data replication, and tools like the Network Weather Service [21] to provide up to date information about the relative costs of accessing data. We plan to extend our earlier Nimrod/G work which uses an experimental computational economy to provide user driven quality of service goals [5][6]. The current prototype for GriddLeS requires the user to start processes on remote machines manually using conventional logins. We plan to alter this to use the Globus GRAM [12], as is done in Nimrod/G. Finally, we will develop a set of real Grid application, like the one in Figure 1, that utilize many resources, distributed data and software components. Clearly the small case study chosen in this paper is only the beginning.

Acknowledgements

The authors would like to acknowledge the work of their colleagues, in particular Mr Tom Peachey and Professor Rhys Jones who provided the case study. This work is supported by Australian Research Council and Hewlett Packard under and ARC Linkage grant.

References

- [1] Abramson D., Sosic R., Giddy J. and Hall B., "Nimrod: A Tool for Performing Parametised Simulations using Distributed Workstations", The 4th IEEE Symposium on High Performance Distributed Computing, Virginia, August 1995.
- [2] Abramson, D., Giddy, J. and Kotler, L. "High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid?", International Parallel and Distributed Processing Symposium (IPDPS), pp 520- 528, Cancun, Mexico, May 2000.
- [3] Allcock, W., Bester, J., Bresnahan, J., Chervenak, A., Liming, L., Meder, S. and Tuecke, S. "GridFTP Protocol Specification. ", GGF GridFTP Working Group Document, September 2002. URL
- [4] Berman et al, "The GrADS Project: Software Support for High-Level Grid Application Development", International Journal of High Performance Computing Applications, Winter 2001 (Volume 15, Number 4), pp. 327-344.
- [5] Buyya, R, Abramson, D., and Giddy, J., An Economy Driven Resource Management Architecture for Global Computational Power Grids, International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA), Las Vegas, Nevada, USA, June 26 - 29, 2000.
- [6] Buyya, R., Stockinger, H., Giddy, J. and Abramson, D. "Economic Models for Management of Resources in Peer-to-Peer and Grid Computing", Technical Track on Commercial Applications for High-Performance Computing, SPIE International Symposium on The Convergence of Information Technologies and Communications (ITCom 2001), August 20-24, 2001, Denver, Colorado, USA
- [7] Chan, P. and Abramson, D. "NetFiles: A Novel Approach to Parallel Programming of Master/Worker Applications", HPC Asia 2001, 24-28 September 2001 • Royal Pines Resort Gold Coast, Queensland, Australia
- [8] Chaperon, P, Jones, R., Heller, M., Pitt, S. and Rose, F., "A methodology for structural optimisation with damage tolerance constraints", Engng Failure Analysis, 7, pp 281-300, 2000.
- [9] Chapin, S., Karpovich, J., Grimshaw, A. "The Legion Resource Management System", Proceedings of the 5th Workshop on Job Scheduling Strategies for Parallel Processing, April 1999. (<http://legion.virginia.edu/>)

- [10] DeTar, C., Fogelson, A., Johnson, C., Sikorski, A. "Computational Engineering and Science Program at the University of Utah," in Computational Science - ICCS 2001, San Francisco May, 2001, Lecture Notes in Computer Science, Springer, Ed. by V. Alexandrov, J. Dongarra, B. Juliano, R. Renner and K. Tan, pp. 1176-1185.
- [11] DiNucci, D. "A Simple and Efficient Process and Communication Abstraction for Network Operating Systems", Proceedings of the Workshop on Communication and Architectural Support for Network-Based Parallel Computing (CANPC) '97, San Antonio, Feb 1-2, 1997, published as LNCS volume 1199, Springer-Verlag, pp. 31-45, D. Panda and C. Stunkel Eds.
- [12] Foster I. and Kesselman C., Globus: A Metacomputing Infrastructure Toolkit, International Journal of Supercomputer Applications, 11(2): 115-128, 1997.
- [13] Foster, I., and Kesselman, C. (editors), The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann Publishers, USA, 1999.
- [14] http://www-fp.mcs.anl.gov/sc2000_netchallenge/entries.htm
- [15] Jones, R. and Peng, D. "A simple method for computing the stress intensity factors for cracks at notches", Engineering Failure Analysis, 9 (2002) 683-702.
- [16] Kacsuk, P., Cunha, J.C., Dózsza, G., Lourenco, J., Antao, T., Fadgyas, T., "GRADE: A Graphical Development and Debugging Environment for Parallel Programs", Parallel Computing Journal, Elsevier, Vol. 22, No. 13, Feb. 1997, pp. 1747-1770.
- [17] Snir, M, Otto, S., Huss-Lederman, S., Walker, D and Dongarra. D., "MPI: The Complete Reference", Published by the MIT Press, 1995.
- [18] Stockinger, H., Samar, A., Allcock, B., Foster, I., Holtman, K and Tierney, B, "File and Object Replication in Data Grids", Journal of Cluster Computing, 5(3)305-314, 2002.
- [19] Szyperski, C. "Component Software - Beyond Object-Oriented Programming", Addison-Wesley/ACM Press, 1998 (411 pages), ISBN 0 201-17888-5.
- [20] Thain, D and Livny, M. "Bypass: A tool for building split execution systems", In the Proceedings of the Ninth IEEE Symposium on High Performance Distributed Computing, Pittsburg, Pennsylvania, pp 79-85, August 14, 2000.
- [21] Wolski, R., Spring, N. and Hayes, J., "The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing", Journal of Future Generation Computing Systems, Volume 15, Numbers 56, pp. 757-768, October, 1999.
- [22] Deelman, E., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Vahi, K., Lazzarini, A., Arbree, A., Cavanaugh, R. and Koranda, S. "Mapping Abstract Complex Workflows onto Grid Environments", In Journal of Grid Computing (to appear).
- [23] GriPhyN, www.griphyn.org
- [24] Deelman, E., Blackburn, K. et al., "GriPhyN and LIGO, Building a Virtual Data Grid for Gravitational Wave Scientists," presented at 11th Intl Symposium on High Performance Distributed Computing, 2002.
- [25] Southern California Earthquake Center's Community Modeling Environment, "<http://www.scec.org/cme/>."
- [26] Annis, J., Zhao, Y. et al., "Applying Chimera Virtual Data Concepts to Cluster Finding in the Sloan Sky Survey," Technical Report GriPhyN-2002-05, 2002.
- [27] NPACI, "Telescience," <https://gridport.npaci.edu/Telescience/>.
- [28] Gannon, D., Bramley, R., Stuckey, T., Villacis, J. Balasubramanian, J., Akman, E., Breg, F., Diwan, S. and Govindaraju, M. "Component Architectures for Distributed Scientific Problem Solving," IEEE Computational Science and Engineering, 5(2): 50-63, 19998
- [29] Armstrong, R., Geist, A., Keahey, K., Kohn, S., McInnes, L., Parker, S. and Smolinski, B. "Toward a Common Component Architecture for High-Performance Scientific Computing". 8th IEEE International Symposium on High-Performance Distributed Computing, Redondo Beach, CA, August 3-6, 1999.
- [30] <http://www.cca-forum.org/>
- [31] Condor DAGMan, <http://www.cs.wisc.edu/condor/dagman/>