

A Flexible IO Scheme for Grid Workflows

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

Abstract

Computational Grids have been proposed as the next generation computing platform for solving large-scale problems in science, engineering, and commerce. There is an enormous amount of interest in applications, called Grid Workflows in which a number of otherwise independent programs are run in a “pipeline”. In practice, there are a number of different mechanisms that can be used to couple the models, ranging from loosely coupled file based IO to tightly coupled message passing. In this paper we propose a flexible IO architecture that provides a wide range of mechanisms for building Grid Workflows without the need for any source code modification and without the need to fix them at design time. Further, the architecture works with legacy applications. We evaluate the performance of our prototype system using a workflow in computational mechanics.

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 [11][12]. 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 [23].

There is an enormous amount of interest in applications, called Grid Workflows in which a number of otherwise independent programs are run in a “pipeline”; so much interest that this is currently being addressed by a working group with the Global Grid Forum [22][21]. Using this

approach, it is possible to process data from an arbitrary source ranging from databases to real time data from scientific instruments. Workflows can be connected in a flexible and dynamic way to give the appearance of a *single application* spanning multiple physical organizations, with access to a wide range of data running on very high performance platforms. Grid workflows have been specified for a number of different scientific domains including physics [14], gravitational wave physics [9], geophysics [31], astronomy [3] and bioinformatics [29].

There is also a great deal of interest in reusing legacy applications within the workflow rather than rewriting new programs. Obviously, this is because we wish to leverage the enormous amount of complex scientific software that already exists. However, many of these existing programs have been written in a range of legacy languages such as Fortran and C and use conventional file IO operations like READ and WRITE. Further, many of them are quite old and are not well suited to modification. As a result, almost all of the Grid Workflow effort assumes that sub components will read and write local files, and that the data will be copied from one stage of the pipeline to the next, as is the case in Nimrod [1] and APST [6]. Whilst powerful, we are interested in supporting a wide range of IO mechanisms ranging from local files to message passing.

In this paper we propose a flexible IO architecture that provides a wide range of mechanisms for building Grid Workflows *without the need for any source code modification and without the need to fix the mechanisms at design time*. Our system is interposed between the application and conventional Grid middleware services like Globus, thus the components of the workflow believe they are executing in a conventional file system, but the system takes advantage of the flexibility and performance of the Grid.

In section 2 we discuss the type of IO operations we want to perform, and in section 3 we discuss the design of a module called a File Multiplexer (FM) that performs these. Section 4 discusses the implementation considerations, followed by an evaluation of the performance of the FM using two case studies.

This work is part of a large project called GriddLeS which aims to support the development of **Grid** applications using **Legacy Software** [17].

2 What sort of IO do we want to support?

Figure 1 shows hypothetical pipeline and illustrates the diversity in IO operations that may be required. Suppose we build a workflow that consists of three main computational phases, executed on three distinct computer systems. In this example, Phase 1 takes data from both a database, and a scientific instrument, processes it, and produces some output in a file. Phase 2 takes that output file as input, but also reads some input from one of a number of replicated data files. It chooses the copy that is as close as possible to Machine 2, where the work is being performed. Phase 2 writes some output files, some of which form the input to Phase 3, which is executed on another machine again.

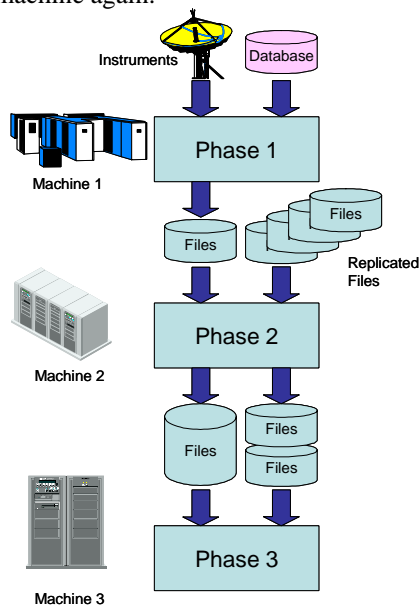


Figure 1 – Sample workflow

Clearly there are a number of ways that this pipeline can be implemented. Most likely, the Phase 1 code will need to be modified so that it can read data directly from the instrument. Likewise, Phase 1 will need code to access the database, which may be local or distributed onto other platforms. Since Phase 2 is executed on a different computer system to Phase 1, it is either necessary to copy files from Machine 1 to Machine 2, or there must be some sort of shared file system across these machines. Phase 2 also reads a file that is replicated on the Grid. In order to do this, it must first locate the most appropriate source,

using some system like the Globus Replica Catalogue [32] or the Storage Resource Broker (SRB) [30], and then it must either copy the file to Machine 2 or use a remote access method like those provided in the SRB or Global Access to Secondary Storage (GASS) [19]. Further, in deciding which file to use, it might need to look at the dynamic bandwidth between systems using something like the Network Weather Service (NWS) [36]. Similarly, when Phase 3 executes on Machine 3, the output files from Phase 2 must be copied to the machines.

Alternatively, it may be beneficial to overlap the computations of Phases 1, 2 and 3 to form a genuine pipelined execution. To achieve this, we would need to remove the IO operations in the programs and replace them with something like those provided by MPICH-G2 [25].

This example is useful because it illustrates the variety of IO that we would like to be able to support. These can be summarized as:

1. Local file IO
2. Local file IO with files copied from one machine to another
3. Remote file IO
4. Remote replicated IO
5. Local replicated IO (i.e. find replica then copy)
6. Direct message passing

Moving between any of these different access methods would almost certainly require source code modification, something we are keen to avoid. Further, it may not be possible to decide which modes to use until the actual workflow is executed. For example, it may not be known whether data files are replicated, or how much bandwidth is available, until the programs are run. Likewise, the computations cannot be scheduled until the loads on various machines are known, so it may not be possible to decide whether a file is local or not before hand. Finally, deciding whether computations should be tightly coupled using message passing may depend on the the characteristics of the communication channels (bandwidth and latency) that are available between the machines that are chosen. Accordingly it is actually desirable to delay the choice of IO method until the application is configured and scheduled. Unfortunately, many of the IO modes discussed above have different APIs and access methods, and this would normally require source code modification. Notably, many of the workflow systems that are currently being discussed assume that files will be copied from one system to another, restricting the effective IO modes dramatically.

3 Flexible IO using the File Multiplexer

As discussed in the previous section we would like to be able to support a wide range of IO methods without modification of the source program. At one end of the spectrum, we want the program to perform local file operations as originally intended. At the other end of the spectrum we wish to connect readers and writers together with a direct TCP connection, without the need to write a file at all. Further, the decision about which mode to use needs to be delayed as late as possible, and may even be changed whilst the programs are running.

3.1 The File Multiplexer Architecture

The key to providing a flexible IO system is to interpose a library between the application and the Grid. We have called this library the File Multiplexer because it makes a dynamic choice about which methods to implement. Figure 2 shows the architecture of the FM.

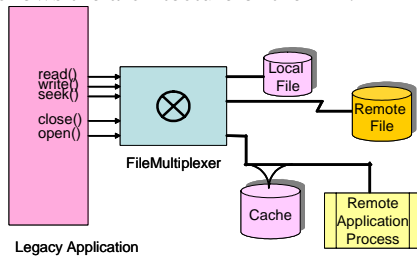


Figure 2 – The File Multiplexer (FM)

The FM intercepts all file operations as specified in the legacy application. When the program performs an OPEN operation, the FM determines which mode to use, and sets up the appropriate pathways. Each OPEN operation makes an independent choice, thus one file may be located locally and another may be remote.

If local file accesses are required, then all of the IO operations are simply diverted to the conventional IO library. Remote file operations can be supported in two ways. First, the remote file can be copied to the local machine, and then local operations can be performed. If the file is modified it can be copied back when it is CLOSED. Second, the FM can access the file on the remote machine using a proxy file server. This type of file operation is supported by systems like Condor. The choice of mode should be based on information about the access patterns and the file size. For example, if an application reads a small fraction of the remote file, it may not warrant copying it to the local file system. Further, if the file is very large, it may not be possible to copy it, and remote accesses might be more efficient. On the other hand, if a file is small and the latency to the remote system it high,

then it is more efficient to copy the file, especially if a latency hiding method like GridFTP is available [19].

If a remote file is replicated, the FM needs to decide which one to access. A range of heuristics can be used to make this decision. For example, if dynamic information such as the network bandwidth and latency is available, then the most *efficient* pathway can be chosen. Further, if a file is opened in read-only mode, then the FM can actually change the mapping dynamically during the execution, allowing it to adapt to changing network conditions. A range of systems can provide replica information, such the Globus Replica Catalogue and the Storage Resource Broker. As with other remote IO, the FM has the choice of copying the file to the local file system or using a remote proxy server.

In some cases a downstream reader application may be overlapped with the writer. For example, in many scientific computations, data is written for each time step, and this is used immediately by a down stream computation. In this scenario, it is desirable to connect the writer directly to the downstream reader using a socket or message buffer. As blocks are written they are copied to the remote system and stored in a buffer. As they are read they can be removed from the buffer. Whilst it is actually possible to place the buffer at either the writer end or the reader end, it is usually more efficient to place it at the reader end, and by using an asynchronous write operation the latency between the systems can be masked. On the other hand, if a writer is broadcasting to a number of readers, such as when a file is used by more than one down stream process, it may be necessary to place the buffer at the reader end of the channel. Normally, such direct communication only works when both the writer and the reader perform sequential IO. However, by caching the file buffer with a cache file, it is possible to allow the reader to re-read blocks from the input stream, and even perform arbitrary seeks. Synchronization of such operations requires the creation and maintenance of a structure that records which blocks have been written, and whether a copy resided in the cache file or not. If a block has not been written, the reader must wait for it to arrive. Accordingly, the read operation can be blocked until the data is written. Similarly, if a reader wishes to seek back to the start of a file and re-read records, it can fetch these the second time around from the cache file. Figure 3 summarizes the use of direct connections.

Interestingly each of the methods discussed in this section have different performance characteristics, and support different concurrency models. Because the FM makes the decision about which method to use at run time, the overall performance of the system can be optimized.

3.2 Configuring a Workflow

When a file is opened, the FM needs to decide which access mode to use. This decision is based on two sorts of dynamic information. First, the designer needs to indicate the way they want the workflow configured. Specifically, they need to decide whether files are accessed locally or remotely, whether a file is replicated, and whether direct buffer connections are required. Most of this information needs to be specified after the components have been written and tested, but before the system is executed. Second, the system needs to obtain information like the machine loads and network performance statistics, and use these to tune where files are located. This information is highly dynamic and may change during the execution of the application.

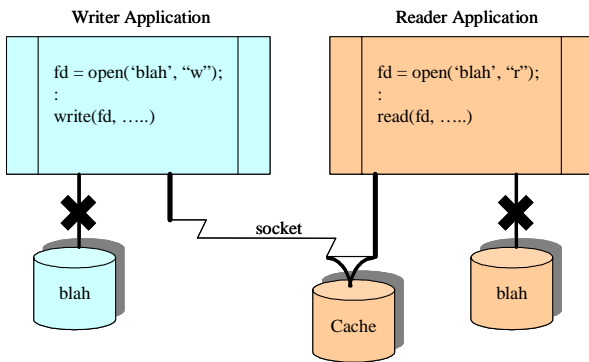


Figure 3 – Direct socket connections

Configuration information is held in a special database, which is accessed using a server call the GriddLeS Name Service (GNS). Each workflow may have its own GNS, or a single GNS may hold configuration information for a number of workflows. When an OPEN call is executed, the GNS is consulted. The GNS matches the names of the machine on which the code resides, and the full path name of the file in the OPEN call, and returns information to the FM about how to configure the IO. If local IO is required, it returns the name of the local file and instructs the FM to use the local file library. If remote access is required, information about the location of the remote file is returned. Further, remote replica information can be resolved at this time.

When a direct connection is requested, 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. Once matched, the system returns the identity and location of the buffer.

3.3 Handling Heterogeneity

Because the Grid consists of machines with differing architectures and operating systems, it is important that the File Multiplexer operates in a heterogeneous environment. The most significant issue is the different byte ordering which has long since been a problem when binary data is taken from one machine to another. Traditionally, applications developers have solved this problem by using formatted ASCII data files when data needs to be written and read on different systems. However, it is possible to use unformatted binary data, and if the record structure is known the bytes can be reordered either before or after they are moved from one machine to another. Currently the File Multiplexer handles formatted ASCII data and binary data only if the two end points have the same byte ordering. However, we are experimenting with a scheme for describing the record structure so that the FM can reorder the bytes dynamically. The data would then be mapped into a neutral form as is done in XDR [18].

3.4 Related Work

A number of projects share features in common with GriddLeS and in particular the File Multiplexer. Intercepting file operations like this is not new, and many systems have used the same method to provide more flexible IO. The Legion file system copies files in before an application is run and out afterwards [35], as does Nimro/G [1]. On the other hand, Condor provides such a mechanism for accessing files in the home file system even when an application is run on a remote node [33]. Condor uses a library called BYPASS, which traps all file system traffic and directs it to a remote shadow process. BYPASS has been used in a number of projects, including The Pluggable File System (now known as Parrot) [16]. Many remote file systems have been built over the years, including NFS, AFS and more recently Grid file systems like the Globus GASS [20]. Many database systems support distributed access in which clients can address one or more database servers. Similarly, the Storage Resource Broker provides a variety of APIs for accessing archived replicated data transparently. All of these systems make it possible to access data from a number of machines, although the level of synchronization support varies between them.

There is much work on developing interoperability among high-performance scientific components by defining an interface definition language that supports scientific abstractions [13][4][15]. However, these techniques still require significant source code

modification, and are better suited to developing new codes rather than legacy file oriented applications.

None of the other work discussed here are as flexible as the approach being proposed in this paper. We are targeting six different IO mechanisms which can be reconfigured dynamically without the need to alter program source code.

4 Implementation Issues

Figure 4 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 TCP connection (accordingly).

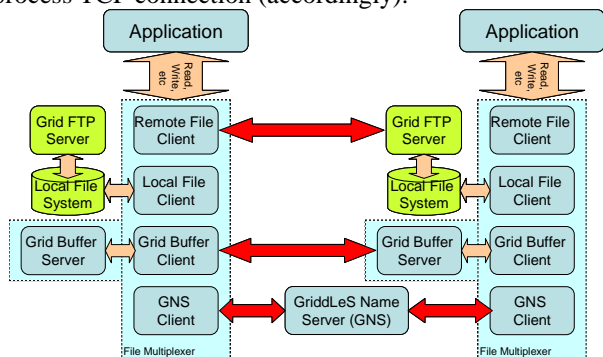


Figure 4 – Architecture of GriddLeS

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 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 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 GridBuffer service acts as a sink for WRITE operations and a source for READs. In order to support random read and write operations, data is stored in a hash table rather than a sequential buffer. The Grid Buffer Service is actually implemented using Web Services, and is accessed by SOAP messages. This implementation leverages the enormous effort in Web Services and also resolves difficulties experienced when direct TCP connections cannot be routed through firewalls.

The Bypass library discussed in section 3.4 provides a framework for intercepting the file system calls and also for accessing data at a remote site. We only utilized the code responsible for intercepting IO operations, and did not use any of the remote server code in our implementation. This is because we wanted to support more services than provided by Bypass.

5 A Case Studies

In this section we consider two computationally intensive workflows and illustrate the benefits of the File Multiplexer.

5.1 The Testbed

In these case studies we utilised a number of machines shown in Table 1, in four countries (AU, US, JP and UK). The machines consisted of different speed Intel Linux systems, connected by varying network speeds.

5.2 Mechanical Engineering

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 [24]. 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 [7]. Figure 6 shows the stress distribution in the plate for a particular hole shape.

Name	Address & Details	Country
dione	dione.csse.monash.edu.au Pentium 4, 1500 MHz, 256 MB, Redhat Linux 7.3	AU
freak	freak.ucsd.edu Athlon Processor, 700 MHz, 256 MB, i386, Debian	US
vpac27	vpac27.vpac.org Pentium 3, 997 MHz, 256 MB, Red Hat Linux 7.3	AU
brecca	brecca-2.vpac.org Intel Xeon, 2.8 GHz, 2048MB, Redhat Linux 7.3	AU
bouscat	Bouscat.cs.cf.ac.uk Pentium 3, 1544 MB, 1 GHz, Red Hat Linux 7.2	UK
jagan	jagan.csse.monash.edu.au Pentium 3, 350 MHz 128 MB, Redhat Linux 7.3	AU
koume00	koume00.hpcc.jp Pentium 3, 1024 MB, 1400MHz Red Hat Linux 7.3, i686	JP

Table 1 – Machine List

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

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 File Multiplexer and GNS are flexible enough to map some files to the local file system, whilst linking writer-reader file chains into direct TCP connections.

We performed a number of experiments as shown in Table 2. We recorded the times that each component completed, and the total execution time for the application

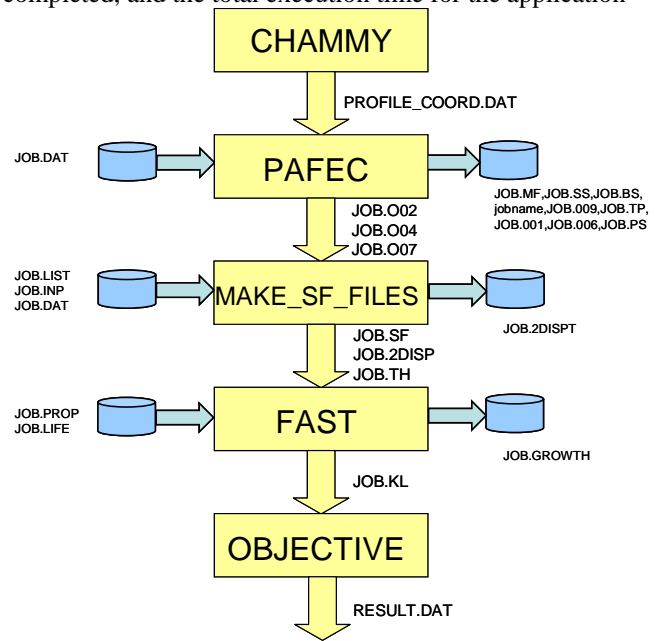


Figure 5 – Durability Pipeline

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 2 – details of the computational experiments performed

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 direct connections 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 using buffers, 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.

5.3 Atmospheric Sciences

In this case study we consider a workflow consisting of three atmospheric models, a Global Climate Model called C-CAM, a linking model called cc2lam and a regional weather model called DARLAM [27][28][34]. Traditionally, C-CAM and DARLAM have been executed on the same computer system, and data is passed between them using conventional files. However, the computational Grid provides an ideal framework for executing these models on different machines that are physically distributed. Importantly, there are many models in use by the atmospheric sciences community, and the Grid provides an opportunity to build genuine multi-organizational models from components that are “owned” by different partners.

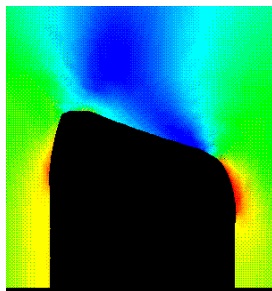


Figure 6 – Stress distribution for one particular hole shape

Importantly, most of the computation is performed by C-CAM and DARLAM, and cc2lam provides simple data manipulation and filtering between the two codes. In this section we describe an experiment in which C-CAM, cc2lam and DARLAM are coupled by Grid Buffers, that is, the output of C-CAM is streamed into DARLAM (via cc2lam). This configuration is shown in Figure 6. This is important because C-CAM and DARLAM are legacy codes written in Fortran, and we did not wish to make modifications to their structure.

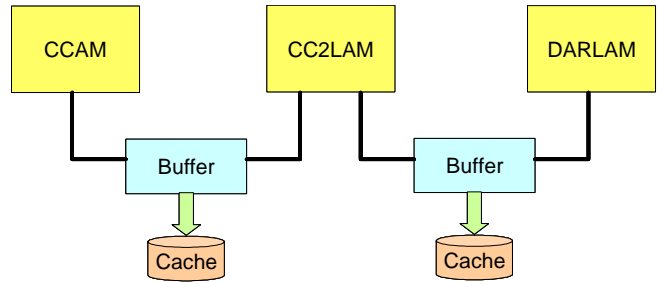


Figure 6 – Linking models using Grid Buffers

When the first writer application (C-CAM) writes a block of data (typically of 4096 bytes) using a write statement, the data is transferred to the Grid Buffer service using the client component of the Grid Buffer service. Once the data arrives, the reader application (in this context cc2lam) uses the Grid Buffer client to obtain the block of data and writes to the other Grid Buffer in a similar way. DARLAM behaves in a similar way, reading from the Grid Buffer and writing its output to conventional files. In some instances, DARLAM re-reads some of the input data. Because the data has already been deleted from the hash table in the Grid buffer Service, it is read from the cache file instead. This occurs transparently to the DARLAM model.

In this case study, three different experiments were performed.

Case 1: All the models are executed sequentially on dedicated machines and the comparative clock timings are as indicated in the following Table 3 below

Machine	C-CAM	cc2lam	DARLAM	Total
dione	00:28:21	00:00:08	00:13:16	00:41:45
brecca	00:16:34	00:00:08	00:07:46	00:24:24
freak	00:30:31	00:00:30	00:13:38	00:44:39
bouscat	01:07:29	00:00:12	00:31:52	01:39:33
vpac27	01:05:22	00:00:11	00:31:00	01:36:33

Table 3 – Times in hr:min:sec for sequential runs.

Case 2: All models are executed concurrently on the same machine. There are two variations on this experiment. First, the programs read and write conventional local files. The results of this experiment are shown in the “With Files” column in Table 4. Second, the programs use Grid Buffers instead of files. These times are shown in the Buffers column in Table 4. All times are cumulative, and thus the DARLAM time also indicates the total time taken.

The results shown in Table 4 highlight that using buffers is always faster than using files when the codes are run on

the same system. This is interesting because the models are multiprocessing the single CPU on these machines, and thus it suggests that buffers are allowing some overlap of computation and communication. Even more interesting is that most of the runs performed with buffers were actually faster than running the codes sequentially on the same platform, as shown in Table 1, again because of the overlap in IO and computation. The exceptions to this are those run on dione and vpac27, which are presumably because of the relative speed of the computation and the IO on these two machines.

Case 3: C-CAM and DARLAM are executed on different computers at different locations, whilst cc2lam is run on the same machine as C-CAM. Because there are a large number of potential pairings of machines, we have selected a few interesting ones and present the timing results in Table 5. Again all times are cumulative, and thus the DARLAM time also indicates the total time taken. In the case where local files are written we have also included the time taken to copy the files in the cumulative totals.

Machine	Model	Files	Buffers
dione	C-CAM	00:41:18	00:44:10
	cc2lam	00:41:56	00:44:15
	DARLAM	01:08:17	00:49:12
brecca	C-CAM	00:18:13	00:20:05
	cc2lam	00:18:25	00:20:12
	DARLAM	00:27:58	00:22:57
freak	C-CAM	00:34:35	00:35:21
	cc2lam	00:35:26	00:35:33
	DARLAM	00:52:39	00:40:30
bouscat	C-CAM	01:10:22	01:17:51
	cc2lam	01:10:39	01:18:10
	DARLAM	01:55:27	01:29:59
vpac27	C-CAM	01:39:28	01:51:11
	cc2lam	01:40:24	01:52:05
	DARLAM	02:44:49	02:15:15

Table 4 – Cumulative concurrent runs on the same system (time in hr:min:sec)

The results in Table 5 show that under some circumstances it is faster to run the codes sequentially and copy the files between phases of the computation. However, under other circumstances it is faster to couple the codes using buffers. Specifically, when the network connections are fast, and the latency relatively low, it is usually faster to couple the codes using buffers. However, when the network connections have a high latency, for example from Australia to the UK or US, it is actually faster to run the

codes sequentially. We believe this is because the file copy sends larger blocks of data, and thus the performance is less sensitive to network latency. *This case study highlights the main advantage of our approach, namely, we are able to shift from using file copies to buffers by only changing some parameters in the GNS.* Both experiments used exactly the same source code. At the same time, we are investigating whether we can produce a version of the buffer code that is less sensitive to network latency.

5.4 Summary

Both of the case studies have illustrated the ability of the File Multiplexer to dynamically assign file IO to local files or direct connections between processes. They did not demonstrate the use of remote files, however, this has been shown in another paper [26]. *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.*

Machine	Model	Files	Buffers
dione	C-CAM	00:28:21	00:34:20
dione	cc2lam	00:28:29	00:34:32
vpac27	File Copy	00:29:19	
	DARLAM	01:00:29	00:48:47
brecca	C-CAM	00:16:34	00:18:05
brecca	cc2lam	00:16:42	00:18:12
dione	File Copy	00:17:32	
	DARLAM	00:30:48	00:25:10
brecca	C-CAM	00:16:34	00:20:51
brecca	cc2lam	00:16:42	01:05:17
bouscat	File Copy	00:24:12	
	DARLAM	00:56:04	01:10:21
dione	C-CAM	00:28:21	00:35:24
dione	cc2lam	00:28:29	00:35:30
brecca	File Copy	00:29:19	
	DARLAM	00:37:05	00:39:24
brecca	C-CAM	00:16:34	00:18:37
brecca	cc2lam	00:16:42	00:18:44
vpac27	File Copy	00:16:57	
	DARLAM	00:47:57	00:40:43
brecca	C-CAM	00:16:34	00:18:19
brecca	cc2lam	00:16:42	00:33:49
freak	File Copy	00:20:17	
	DARLAM	00:33:55	00:41:45

Table 5 – Cumulative concurrent runs on different systems

6 Conclusion

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 two computationally intensive applications.

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 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 [8]. Importantly, the scheduler needs to take account of whether the workflow is configured to copy files or use direct connections, since both impose different scheduling constraints. For example, if file copies are performed the computations need to be run sequentially. On the other hand, if buffers are used then they need to run at the same time.

We have not yet implemented the replication management within the FM. We plan to make use of either the Globus Replication catalogue or the Storage Resource Broker (SRB) to support data replication, and tools like the Network Weather Service 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]. Finally, we will develop a set of real Grid applications that utilize many resources, and distributed data and software components. Clearly the small case studies chosen in this paper are 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 mechanical engineering case study, and John L. McGregor and Jack Katzfey who provided the Atmospheric Sciences case study. This work

is supported by Australian Research Council and Hewlett Packard under and ARC Linkage grant.

References

- [1] 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.
- [2] 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
- [3] 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.
- [4] 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.
- [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] Casanova, H., Obertelli, G., Berman, F. and Wolski, R. "The AppLeS Parameter Sweep Template: User-Level Middleware for the Grid", Proceedings of the Super Computing Conference (SC'2000).
- [7] Chaperon, P., Jones, R., Heller, M., Pitt, S. and Rose, F., "A methodology for structural optimisation with damage tolerance constraints", Engineering Failure Analysis, 7, pp 281-300, 2000.
- [8] Condor DAGman, <http://www.cs.wisc.edu/condor/dagman/>
- [9] Deelman, E., Blackburn, K. et al., "GriPhyN and LIGO, Building a Virtual Data Grid for Gravitational Wave Scientists," 11th Intl Symposium on High Performance Distributed Computing, 2002.
- [10] 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).
- [11] Foster I. and Kesselman C., Globus: A Metacomputing Infrastructure Toolkit, International Journal of Supercomputer Applications, 11(2): 115-128, 1997.

- [12] Foster, I., and Kesselman, C. (editors), *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann Publishers, USA, 1999.
- [13] 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, 1998
- [14] GriPhyN, www.griphyn.org
- [15] <http://www.cca-forum.org/>
- [16] <http://www.cs.wisc.edu/condor/pfs>
- [17] <http://www.csse.monash.edu.au/~davida/griddles>
- [18] <http://www.faqs.org/rfcs/rfc1014.html>
- [19] <http://www.globus.org/datagrid/gridftp.html>
- [20] <http://www.globus.org/gass/>
- [21] http://www.isi.edu/~deelman/wfm-rg/wfm_charter.pdf
- [22] <http://www.ggf.org>
- [23] http://www-fp.mcs.anl.gov/sc2000_netchallenge/entries.htm
- [24] 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.
- [25] Karonis, N., Toonen, B. and Foster, I. "MPICH-G2: A Grid-Enabled Implementation of the Message Passing Interface", *Journal of Parallel and Distributed Computing (JPDC)*, Vol. 63, No. 5, pp. 551-563, May 2003..
- [26] Kommineni, J. "Grid Applications: A Seamless Approach with Web Services, The APAC Conference on Advanced Computing, Grid Applications and eResearch, Royal Pines Resort, Gold Coast, Queensland Australia. 29 September - 2 October, 2003.
- [27] McGregor, J. L., Nguyen, K. C., and Katzfey, J. J. Regional climate simulations using a stretched-grid global model. In: *Research activities in atmospheric and oceanic modelling*. H. Ritchie (ed.). (CAS/JSC Working Group on Numerical Experimentation Report; 32; WMO/TD - no. 1105) [Geneva]: WMO. p. 3.15-3.16, 2002.
- [28] McGregor, J.L., Walsh, K.J. and Katzfey, J.J. Nested modelling for regional climate studies. In: A.J. Jakeman, M.B. Beck and M.J. McAleer (eds.), *Modelling Change in Environmental Systems*, J. Wiley and Sons, 367-386, 1993.
- [29] NPACI, "Telescience, <https://gridport.npaci.edu/Telescience/>."
- [30] Rajasekar, A., Wan, M., Moore, R., Kremenek, G. and Guptill, T. "Data Grids, Collections and Grid Bricks", 20th IEEE/ 11th NASA Goddard Conference on Mass Storage Systems & Technologies (MSST2003) San Diego, California, April 7-10, 2003.
- [31] Southern California Earthquake Center's Community Modeling Environment, "<http://www.scec.org/cme/>."
- [32] 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.
- [33] 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 1-4, 2000.
- [34] Walsh, K.J. and McGregor, J.L. January and July climate simulations over the Australian region using a limited-area model. *J. Climate*, 8 (10), 2387-2403, 1995.
- [35] White, B., Grimshaw, A., Nguyen-Tuong, A. "Grid-Based File Access: The Legion I/O Model", Ninth IEEE International Symposium on High Performance Distributed Computing, Pittsburgh, Pennsylvania, Aug 1-4, 2000.
- [36] 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 5-6, pp. 757-768, October, 1999.