An Atmospheric Sciences Workflow and Its Implementation with Web Services

David Abramson [1], Jagan Kommineni [1], John L. McGregor [2], and Jack Katzfey [2]

[1] School of Computer Science and Software Eng., Monash University, 900 Dandenong Rd, Caulfield East, 3145, Australia
[2] Division of Atmospheric Science, CSIRO, PMB 1, Aspendale,Vic, 3195, Australia

Abstract. Computational and data Grids couple geographically distributed resources such as high performance computers, workstations, clusters, and scientific instruments. Grid Workflows consist of a number of components, including: computational models, distributed files, scientific instruments and special hardware platforms. In this paper, we describe an interesting grid workflow in atmospheric sciences and show how it can be implemented using Web Services. An interesting attribute of our implementation technique is that the application codes can be adapted to work on the Grid without source modification.

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 [4][5]. 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 are currently available. Of particular interest are applications, called "Grid Workflows", that consist of a number of components, including: computational models, distributed files, scientific instruments and special hardware platforms (such as visualisation systems) [3]. 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, running on a single platform. Grid workflows have been specified for a number of different scientific domains including physics [6] and gravitational wave physics [2].

In this paper we describe a Grid workflow for solving problems in atmospheric sciences. The workflow supports the coupling of a number of pre-existing legacy computational models across distributed computers. An important aspect of the work

M. Bubak et al. (Eds.): ICCS 2004, LNCS 3036, pp. 164–173, 2004.

is that we do not require source modification of the codes. In fact, we don't even require access to the source code. In order to implement the workflow we overload the normal file IO operations to allow them to work in the Grid. We also leverage existing Grid middleware layers like Globus [4] [5] to provide access to control of the underlying resources.

In Section 2 we describe an atmospheric science workflow, with some detail of the functions of the various components. Section 3 discusses the implementation techniques, and Section 4 provides some experimental results.

2 An Atmospheric Sciences Workflow

Global climate models: A global climate model is a computer model representing the atmosphere, oceans, land and sea-ice. By solving mathematical equations based upon the laws of physics, a GCM simulates the behaviour of the climate system. The model divides the planet into a number of vertical layers representing levels in the atmosphere and depths in the oceans, and divides the surface of the planet into a grid of horizontal boxes separated by lines which may be similar to latitudes and longitudes. In this way, the planet is covered by a three-dimensional grid of boxes (Figure 1).

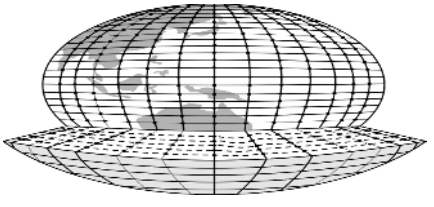

Fig. 1. Representation of the Earth's surface and atmosphere in a typical global climate model.

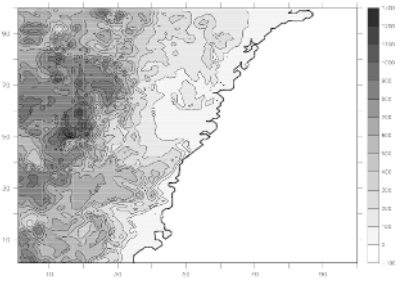

Fig. 2. DARLAM model domain for simulations using a 3 km grid over Sydney.

Global climate models capture large scale features like the deserts and tropics very well, but have difficulty capturing smaller features like cyclones and thunderstorms because they occur at scales much smaller than the grid boxes.

Regional climate models: To improve regional detail in climate models, it is desirable to reduce the spacing between grid points. However, due to the complexity of global climate modeling, computational requirements become prohibitive if the horizontal grid resolution is less than a few hundred kilometers. At this resolution, vitally important small-scale phenomena, like tropical cyclones and cold fronts, are poorly captured. This affects simulated patterns of temperature and rainfall, and hence the ability to realistically simulate observed regional climate features in GCMs.

A computationally feasible alternative to a coarse resolution global climate model is to use a finer resolution model over a small part of the globe. A regional climate model (RCM), with a horizontal resolution of about 100 km or less, is able to simulate

regional weather patterns better than most GCMs [7][8][11]. Part of the reason for the improved climate simulation relative to GCMs is the fact that coastlines and mountains are represented in more detail in RCMs. Since topographic features strongly influence regional temperature and rainfall, more detailed features are likely to give a better climate simulation.

A regional climate model requires meteorological information at its lateral boundaries in order to simulate weather within its boundaries. For climate change studies, an RCM is typically driven at its boundaries by information from a coarser-scale GCM. This is commonly called nesting an RCM inside a GCM. One-way nesting allows information to flow from the GCM to the RCM each simulated day, but the weather simulated by the RCM does not affect the GCM interactively. This means that the RCM can be run after the GCM experiment has been completed.

A Grid Workflow: Traditionally, the GCM and the RCM have been executed on the same computer system, and data is passed between them using conventional files. However, the computational Grid discussed in Section 1 provides an ideal framework for executing these models on different machines that are physically distributed. There are many reasons why one might wish to do this. First, both models may have different computational requirements and be suited to different types of platforms. For example, one may execute well on a vector supercomputer and the other may be efficient on a parallel processor. Second, both models may not have been ported to the same hardware. Thus, it may be time consuming and expensive to couple them on one machine. Third, the models may be "owned" by different organizations. As discussed in [4] the computational grid facilitates the construction of a "virtual organization" in which the models are linked into a single grid application without actually moving the codes to a single organization. Finally, it may be possible to pipeline the computations, providing a quicker solution than if they were run sequentially on one system. This can be achieved if the data files are replaced by communication pipes that allow one program to write data concurrently with a downstream one reading the same data. Figure 3 shows such a grid workflow based around the models discussed above. In this paper we discuss a particular system involving three models – a GCM called C-CAM, a RCM called DARLAM and a data filter called cc2lam.

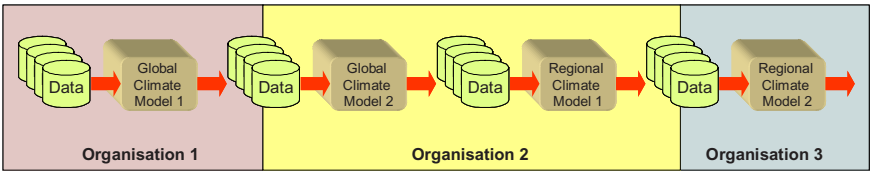

Fig. 3. An Atmospheric Sciences Grid Workflow

3 An Implementation with Web Services

Clearly it is possible to implement the system described in the previous section in a number of ways. For example, the programs could be run unmodified, and some system could be responsible for copying files from one machine to another. This is

effectively the practice that atmospheric scientists have employed manually for some time now. Or, the files could even be shared by a single distributed file system like NFS or AFS, removing the need to explicitly copy them from one local file system to another. The major disadvantage of this general approach is that it does not take advantage of any potential overlap in the computations. Another option is to modify the programs so that they do not read directly from files, but instead they use a message passing library like PVM or MPI to send data from one model to another. This allows the computations to be pipelined, however the programs would need to be modified at the source level. Further, once modified that would no longer work as stand alone codes, limiting the flexibility of the individual components. A third alternative is to modify the file system library so it performs message passing rather than writing to and reading from local files. This achieves the advantages of both of the previous alternatives, requires no source modification (only relinking the application) and does not permanently fix the way the programs operate. For example, by linking the normal file system primitives, the programs behaves as normal, reading and writing local files. However, when the message passing library is linked, the program sends messages for writes, and receives messages for read. In a previous paper we proposed such a mechanism, called NetFiles, for implementing parallel master-slave programs [1]. NetFiles were implemented within a single cluster or parallel machine and could not cross administrative domains. Here we have broadened the approach to support interprocess communication across the computational Grid. Accordingly we have called this mechanism GridFiles. Figure 4 shows how two legacy computations can be coupled using GridFiles.

In the GridFiles approach, the conventional system calls (like open, read, write etc.,) are replaced (transparently) by a call to a module called a “File Multiplexer”. The File Multiplexer is responsible for passing the file system operations onto an appropriate service, and has the flexibility to change the mappings dynamically. Thus, the File Multiplexer can redirect IO requests to local files, local processes, remote files or remote processes. This means that a program can perform a READ operation, and this might read a local file, or a remote one, or even be connected directly to a WRITE operation on a remote machine. The latter mode is the equivalent of connecting the two programs by sockets, and allows complete overlap of IO operations, and is called the buffer service. This structure is depicted in Figure 4. The File Multiplexer is composed of three clients. The Local File Client performs local file operations. The Grid File Client communicates with the Grid Buffer Service, and the GNS Client communicates with the GriddLeS Name Service. The Grid Buffer and Name Services are both implemented using Web Service technologies such a SOAP and XML [9][10].

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. Thus, if a read is issued for a block that has not been written yet, the read waits for the data to arrive. After a block has been read from the hash table it is written to a cache file and then deleted from the hash table. The cache file is provided for two main reasons. First, it allows a block to be reread even after it has been deleted from the hash table. This occurs when the reader seeks back to a previous block. Second, it provides a mechanism for implementing broadcast operations to more than one process. When this happens, the first reader obtains the

data from the hash table, but subsequent readers retrieve the data from the cache file. It is possible to have an arbitrary number of readers using the approach without the need to inform the Grid Buffer Service before hand.

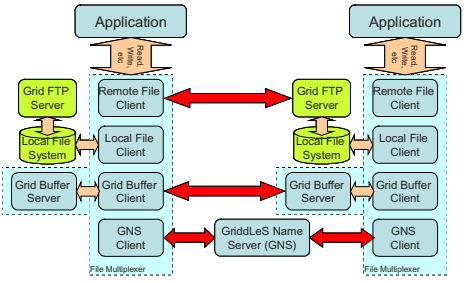

Fig. 4. Using GridFiles to link apps

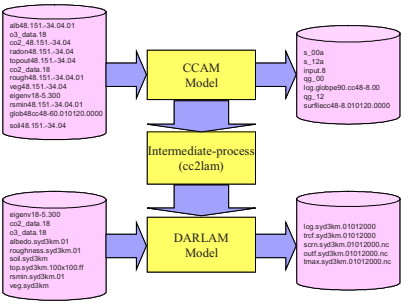

Fig. 5. File dataflow

The GriddLes Name Service (GNS) is responsible for configuring the grid application. Each entry in the GNS indicates what should happen when a particular file is opened on a particular resource. At the time of the deployment the GNS reads the data from the configuration file and keeps it in a hash table. These keys and values correspond to the global naming schema. The reader and writer applications can use different keys which are mapped to the same entry in the GNS, indicating both are linked to either the same file or buffer. Buffers are distinguished from files by the word "buffer" in the value. If an entry in the GNS represents a buffer, then additional configuration information is contained. The GNS can be updated at any time, a this reconfigures the Grid application dynamically.

All the applications and web services can run on the same system or each one can be distributed to the different Grid nodes which are located geographically at different locations.

4 Experimental Results

Figure 5 shows a particular configuration of the three atmospheric models discussed in section 2, namely C-CAM, cc2lam and DARLAM. 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). Because we used a file multiplexer as discussed in section 3*, it was possible to do this without source modification.* This is important because C-CAM and DARLAM are legacy codes written in Fortran, and we did not wish to make significant modifications to their structure. This scheme is very flexible. All models and services can run on a single machine, or on different nodes of a cluster or on different machines in a computational grid. Further, these configuration changes can occur without any modification to the atmospheric models.

When the first writer application (C-CAM) writes block of data (typically of 4096 bytes) using the write statement, then the data need to be transferred over to the GridBuffer service by using the client component of GridBuffer service which is in FMP underneath the application layer without the user interaction. Once the data arrives, the reader application (in this context cc2lam) uses the GriddBuffer client and obtain the block of data and writes into the other GridBuffer in a similar way, so that the other reader application (in this context the DARLAM Model) reads the block of data. In some instances, DARLAM rereads some of the input data. Because the data has already been deleted from the hash table in the Grid buffer Service, it is read form the cache file instead. This occurs transparently to the DARLAM model.

In this experiment we utilised a number of machines shown in Table 1, in three countries (AU, US and UK).

Table 1. Machine List

Name	Address & Details	Country	Name	Address & Details	Country
dione	dione.csse.monash.edu.au Pentium 4, 1500 MHz, 256 MB, Redhat Linux 7.3	AU	brecca	brecca-2.vpac.org Intel Xeon, 2.8 GHz, 2048MB, Redhat Linux 7.3	AU
freak	freak.ucsd.edu Athlon Processor, 700 MHz, 256 MB, i386, Debian	US	bouscat	bouscat.cs.cf.ac.uk Pentium 3, 1 GHz, 1544 MB, Red Hat Linux 7.2	UK
vpac27	vpac27.vpac.org Pentium 3, 997 MHz, 256 MB, Red Hat Linux 7.3	AU	dragon	dragon.vpac.org Intel(R) Pentium(R) 4 CPU 1.70GHz, 1006MB Red Hat Linux 8.0 3.2-7	AU

In this case study, three different experiments were performed.

Case 1: 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 "Files" column in Table 2. Second, the programs use Grid Buffers instead of files. These times are shown in the Buffers column in Table 2. All times are cumulative, and thus the DARLAM time also indicates the total time taken. In this experiment the code was run for 480 and 960 time steps.

The results shown in Table 2 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, 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 2: 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

Table 2. Cumulative concurrent runs on the same system (time in hr:min:sec)

		480 time steps		960 time steps	
Computer	Model	Files	Buffers	Files	Buffers
dione	C-CAM	00:41:18	00:44:10	01:23:57	01:29:59
	cc2lam	00:41:56	00:44:15	01:25:13	01:30:09
	DARLAM	01:08:17	00:49:12	02:19:13	01:35:00
brecca	C-CAM	00:18:13	00:20:05	00:36:29	00:41:21
	cc2lam	00:18:25	00:20:12	00:36:50	00:41:24
	DARLAM	00:27:58	00:22:57	00:56:35	00:44:11
freak	C-CAM	00:34:35	00:35:21	01:22:28	01:28:15
	cc2lam	00:35:26	00:35:33	01:23:15	01:30:01
	DARLAM	00:52:39	00:40:30	02:14:22	01:36:15
bouscat	C-CAM	01:10:22	01:17:51	02:44:03	02:42:42
	cc2lam	01:10:39	01:18:10	02:44:39	02:43:00
	DARLAM	01:55:27	01:29:59	04:14:54	02:54:55
dragon	C-CAM	00:41:25	00:41:28	01:23:01	01:27:57
	cc2lam	00:41:46	00:41:41	01:23:43	01:28:08
	DARLAM	01:06:17	00:46:24	02:13:12	01:32:57

ones and present the timing results in Table 3. 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. The runs were done for both 480 and 960 time steps.

Table 3. Cumulative concurrent runs (Time in hr:min:sec)

		Close Systems					Distant Systems			
		480 Time steps		960 Time steps			480 Time steps		960 Time steps	
Model	Mach	Files	Buffers	Files	Buffers	Mach	Files	Buffer	Files	Buffer
C-CAM	dione	00:28:21	00:34:20	00:55:18	01:08:14	brecca	00:16:34	00:18:39	00:32:32	00:38:14
cc2lam	dione	00:28:29	00:34:32	00:55:40	01:08:25	brecca	00:16:42	01:00:48	00:32:49	01:48:42
File Copy		00:29:19		00:56:52			00:24:12		00:40:25	
DARLAM	vpac27	01:00:29	00:48:47	02:00:59	01:26:04		00:56:04	01:05:03	01:46:19	01:52:58
						bouscat				
C-CAM	brecca	00:16:34	00:18:37	00:32:32	00:38:55		01:07:29	01:18:18	02:29:58	02:22:44
cc2lam	brecca	00:16:42	00:18:44	00:32:51	00:39:01	bouscat	01:07:41	01:30:34	02:30:23	02:34:12
File Copy		00:16:57		00:33:04			01:17:11		02:37:59	
DARLAM	vpac27	00:47:57	00:40:43	01:37:11	01:14:35	bouscat	01:24:57	01:31:55	02:55:02	02:35:20
						brecca				
C-CAM	brecca	00:16:34	00:18:05	00:32:32	00:38:55	brecca	00:16:34	00:18:20	00:32:32	00:37:10
cc2lam	brecca	00:16:42	00:18:12	00:32:48	00:39:01	brecca	00:16:42	00:33:49	00:32:49	01:05:40
File Copy		00:17:32		00:33:59			00:20:17		00:37:14	
DARLAM	dione	00:30:48	00:25:10	00:59:53	00:44:27	freak	00:33:55	00:41:45	01:07:01	01:07:19
C-CAM	brecca	00:16:34	00:18:37	00:32:32	00:39:03	freak	00:30:31	00:36:57	01:16:56	01:14:03
cc2lam	brecca	00:16:42	00:18:40	00:32:48	00:39:18	freak	00:31:01	00:44:08	01:17:23	01:21:07
File Copy		00:17:32		00:33:04			00:34:36		01:21:48	
DARLAM	dragon	00:29:44	00:23:10	01:00:09	00:43:53	brecca	00:42:22	00:45:17	01:38:51	01:22:16

The results show that buffers are always faster for systems which have good network connections than when files are used. Interestingly, the results for 960 time steps are less than double those for 480. This is because the startup overheads are masked in the longer runs, and thus the parallel efficiency is higher.

The results for distant machines tell a different story. Because these machines have poorer networks between them, it is not always faster to use buffers than to copy

the files. For example for the shorter runs of 480 time steps, it is always better to use file copies. On the other hand, buffers are more efficient for some of the longer 960 time step runs. The results shown here highlight the importance of being able to reconfigure the application dynamically because it is not always possible to know which configuration will be more efficient.

Case 3: In this experiment C-CAM and cc2lam are executed on brecca and DARLAM model is run on dragon. We run models for 480 time but data is exchanged at different intervals from 60 to 15 time steps. The results are shown in Table 4.

Table 4. Varying the output frequency (Time specified in hr:min:sec format)

	Amount of Output Data produced by different models in MB				
Print Interval in time steps	CCAM	cc2lam	DARLAM	Total Time with Files	Total time with Buffers
60	72.663	42.485	43.000	00:29:38	00:23:23
30	137.253	80.249	83.599	00:35:50	00:26:30
15	264.428	155.771	163.760	00:38:54	00:26:21

As expected, as the write interval reduces, the magnitude of data produced by each model increases. Even though there is an increase in computation time as the write interval reduces, the total computation time with the buffers is less than with files because more overlap is possible.

Table 5 analyses the degree of overlap in the computations, and therefore the efficiency, for a few different machine configurations. Here we calculate the best theoretical time that could have been achieved assuming no startup costs and perfect networking, and compare this to the actual time. The results indicate that the efficiency can be quite high for low latency, high bandwidth networks, especially for the longer runs.

Table 5. Overlap comparisons (Time specified in hr:min:sec format)

Machines	Time Steps	C-CAM with Files	DARLAM with Files	Total time with Buffers	Best Theoretical	Efficiency
Brecca-2 & vpac27	480	0:16:34	0:31:00	0:40:43	0:31:00	76%
	960	0:32:32	1:04:07	1:14:35	1:04:07	86%
dione & vpac27	480	0:28:21	0:31:00	0:48:47	0:31:00	64%
	960	0:55:18	1:04:07	1:26:04	1:04:07	74%
Brecca-2 & vpac27	480	0:16:34	0:13:16	0:24:58	0:16:34	66%
	960	0:32:32	0:25:54	0:44:27	0:32:32	73%

5 Conclusions

In this paper we have discussed the implementation of a grid workflow that couples two legacy atmospheric science applications, namely a global climate model and a regional weather model. *One of the more significant achievements of the experiment is that we managed to do this without any changes in the source code of the two models.* This is no mean feat since they are legacy codes written in Fortran and were designed without any knowledge of the underlying grid infrastructure. Inter-process communication is provided by a software device called a File Multiplexer, and we have chosen to implement this with web services.

The performance results indicate that there are cases when it is advantageous to couple the models tightly using pipes, and other cases where it is more efficient to write files locally, copy them to the receiving node and read them locally. An important feature of our implementation is that the decision about whether to copy or use buffers can be delayed until *the time that the application is configured* and does not need to be integrated into the source code of the models. We are not aware of other systems with equivalent flexibility.

The case study discussed here is actually a small fragment of a much larger system currently being constructed. Rather than just couple 2 models, we plan to couple a number of atmospheric science models including air pollution codes. We also plan to retrieve data directly from scientific instruments like temperature and pressure sensors. Such an application would form an interesting grid application because it would involve integration of a number of separate computational models, running on different computer systems (possibly owned by different organizations) and taking data from real time scientific instruments.

Acknowledgements. The Australian Research Council and Hewlett Packard support this work under an ARC linkage grant. Kommineni was supported by an Australian Postgraduate Research Award.

References

1. 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.
2. 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.
3. 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”, Journal of Grid Computing, Vol. 1, No. 1, pp 9--23, 2003.
4. Foster, I. and Kesselman, C., Globus: A Metacomputing Infrastructure Toolkit, International Journal of Supercomputer Applications, 11(2): 115-128, 1997.
5. Foster, I., and Kesselman, C. (editors), The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann Publishers, USA, 1999.

6. GriPhyN 2003, www.griphyn.org
7. 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.
8. 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.
9. Robert van Engelen, "The gSOAP toolkit 2.0.", Technical report, Florida State University, http://www.cs.fsu.edu/~engelen/soap.html, 2001.
10. Sun Microsystems, "Web Services Made Easier: The Java APIs & Architectures for XML", 2002, http://java.sun.com/webservices/white/ & http://java.sun.com/webservices/webservicespack.html
11. 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.