Running climate models on grids using G-Rex

By D. A. Bretherton*, J. D. Blower, K. Haines and G. C. Smith

Reading e-Science Centre, Environmental Systems Science Centre (ESSC), University of Reading, 3 Earley Gate, Reading RG6 6AL, UK

Compute grids are used widely in many areas of environmental science, but there has been limited uptake of grid computing by the climate modelling community, partly because the characteristics of many climate models make them difficult to use with popular grid middleware systems. In particular, climate models usually produce large volumes of output data, and running them also involves complicated workflows implemented as shell scripts. A new grid middleware system that is well suited to climate modelling applications is presented in this paper. GRID REMOTE EXECUTION (G-Rex) allows climate models to be deployed as Web services on remote computer systems and then launched and controlled as if they were running on the user’s own computer. Output from the model is transferred back to the user while the run is in progress, to prevent it from accumulating on the remote system and to allow the user to monitor the model. G-Rex has a representational state transfer (REST) architectural style, featuring a Java client program that can easily be incorporated into existing scientific workflow scripts. Some technical details of G-Rex are presented, with examples of its use by climate modellers.

Keywords: grid middleware; climate model; Nucleus for European Modelling of the Ocean (NEMO); Web service; representational state transfer (REST)

1. Introduction

Computer models of the oceans and atmosphere are important tools in the study of the Earth’s system. The Nucleus for European Modelling of the Ocean (NEMO; http://www.lodyc.jussieu.fr/NEMO) model (Smith & Haines in press) is a state-of-the-art ocean model that is being used as a European community model, for operational ocean forecasting in France and the UK, and it is also being developed for climate modelling. At Reading, we have been developing data assimilation algorithms for NEMO in order to reconstruct the past state of the ocean climate. On a typical modern cluster, a 1 year global ocean simulation at 1° resolution takes about 3 hours when running on 40 processors and, when assimilating observational data, produces roughly 20 GB of output as 50 000 separate files. For past ocean climate studies, we run the model for 50 year simulations, during which the model resubmits as a new job after each year. Running NEMO relies on a set of complicated shell scripts and command utilities

* Author for correspondence (dab@mail.nerc-essc.ac.uk).

One contribution of 24 to a Discussion Meeting Issue ‘The environmental eScience revolution’.

Published online 16 December 2008

One contribution of 24 to a Discussion Meeting Issue ‘The environmental eScience revolution’.

This journal is © 2008 The Royal Society
to carry out scientific workflows involving meteorological forcing and data pre-and post-processing prior to job resubmission. Models such as NEMO are typically run on expensive national high-performance computing (HPC) resources. NEMO offers an excellent example of a climate model with large numbers of users who could benefit from the choice to run their experiments on a grid of shared cluster resources such as the UK Natural Environment Research Council (NERC) Cluster Grid, which will be described later.

Scientists can benefit in a number of ways from the ability to run models on clusters outside their own institute. For example, runs can be performed when the cluster at their own institute is busy or unavailable, or a task that is too large or lengthy to be performed on a single cluster can be accomplished by distributing the computational load across several different clusters, as in the GCOMS project (Holt et al. 2009). The purpose of grid ‘middleware’ is to provide a uniform interface between the users and remote HPC resources. To be effective for climate modelling applications, grid middleware must address two issues in particular: the large volume of output data, which must not be allowed to accumulate on the remote system; and the complicated scientific workflow scripts, with which the middleware must be able to integrate easily. These are the two challenges that the middleware described in this paper was specifically designed to tackle.

### 2. Grid middleware for climate modelling applications

There are many grid middleware systems currently available. Early middleware systems were criticized for being difficult to install, maintain and use (Chin & Coveney 2004), and so new middleware emerged, based upon Web services architectures (e.g. GLOBUS TOOLKIT 4, Foster 2005; GridSAM, http://gridsam.sourceforge.net). From the climate modeller’s perspective, these newer systems still suffer from usability problems, such as the requirement to construct complex job submission documents (e.g. in JSDL; Anjomshoaa et al. 2005), making it difficult to integrate the grid into the user’s existing scripts and workflows. GRID REMOTE EXECUTION (G-REX) is pure-Java grid middleware that has been developed to address the above limitations. The client component of G-REX is a command-line program called `grexrun`, which behaves in the same way as the model would if it were running on the user’s own computer, without the need for the user to create an accompanying job description document or a batch job submission file (as explained later). The `grexrun` program is therefore very easily integrated with the users’ existing scripts and workflows. Output files can be transferred to the user during the model run, allowing the user to monitor the progress of a long-running model. Once files have been transferred, G-REX can delete them from the server, again during the model run, preventing the accumulation of large quantities of data on the remote server. Although some of these features are provided by other middleware solutions, no existing solution combines all of these features while retaining G-REX’s ease of use.

G-REX is most similar in concept to the APPLICATION HOSTING ENVIRONMENT (AHE, http://www.realitygrid.org/AHE/), but is designed for reduced complexity and greater ease of installation (Turilli 2008) and use. The complexity of G-REX is kept to a minimum by concentrating on the specific needs of climate modellers, rather than attempting to solve all problems associated with remote
execution. For example, G-REX does not provide solutions to the problems associated with building and launching a model on a range of different platforms; these are problems that have to be addressed by the grid administrator when deploying a model as a G-REX Web service on a new HPC resource for the first time. G-REX also does not have facilities to aid selection of the most appropriate (e.g. least loaded) resource for a particular model run. This task is not easily automated and therefore is left to the G-REX user, but grids such as the NERC Cluster Grid usually provide other tools for making this easier.

3. G-REX architecture and operating mechanism

G-REX shares much of its design with the earlier STYX GRID SERVICES (SGS) middleware system (Blower et al. 2006), which used the STYX (http://www.vitanuova.com/inferno/papers/styx.html) architecture for distributed systems. SGS addressed many of the usability issues associated with other middleware systems, but still suffered from performance and scalability issues when applied to large climate models. G-REX’s architecture is based upon HTTP, which is more suited to large file transfer operations and allows for greater interoperability with third-party tools. The G-REX architecture is illustrated in figure 1. The G-REX server is a standard Java Web application that runs inside a servlet container\(^1\) such as APACHE TOMCAT, and is therefore easy to install and maintain by system administrators. G-REX exposes remotely installed climate models as Web services using the representational state transfer (REST)\(^2\) architectural style (Fielding 2000), in which the services and their resources (including their configuration, status of individual jobs and input and output files) are all identified by URLs, and the client communicates with the server using standard HTTP operations such as GET and POST. This RESTful approach allows client programs to be very simple and lightweight and allows users to interact with model runs using only a basic HTTP client (such as a Web browser or the \texttt{curl}\(^3\) utility) if they wish. This design allows for new client interfaces to be developed in other programming languages with relatively little effort. Alternative clients need not include all the features available in \texttt{grexrun}.

For example, the G-REX server provides a basic, read-only Web browser interface that allows users to browse the available services, check the status of batch jobs and download individual output files.

Easy integration of \texttt{grexrun} with the scientists’ existing workflow scripts is one of G-REX’s most important features. G-REX achieves this by automatically generating two documents: the job description document and the batch job submission file. The job description document is generated by \texttt{grexrun} in an XML format that is never seen by the user. (This document is roughly equivalent to a JSDL document as used by systems such as GRIDSAM and indeed G-REX could be adapted to emit JSDL.) It contains details specific to one particular

---

\(^1\) A servlet container is a Web server that supports the execution of servlets, Java programming language elements that are commonly used to provide dynamic (i.e. user interactive) content in websites.

\(^2\) A set of software design principles that lies behind the operation of the World Wide Web.

\(^3\) A command-line tool for transferring files with URL syntax; http://curl.haxx.se/.

*Phil. Trans. R. Soc. A* (2009)
model run, including the parameters used by the remote executable file(s) and the patterns used to identify input and output files. This information is supplied by the user in the form of `grexrun` command-line arguments that are validated against the Web service configuration information held by the G-REX server. The server uses a configuration file in XML format that defines each application service in general terms, with information about all the possible execution parameters (optional and required) and the type of input and output files to expect. For some applications, the names and the number of input and output files may be fixed, but usually these will be different from one run to the next. For example, the service configuration file may state that a particular model produces output files with names of the form `${RUN_ID}/C3.dat`", where `RUN_ID` is a character string supplied by the user. All files matching this pattern would then be transferred back to the client as soon as they appeared in the model’s working directory. Files not matching any of the output file patterns are not transferred, and are deleted at the end of the model run along with all the input files that were uploaded by `grexrun` before the start of the run.

The batch job submission file, containing information about the model run in a format that can be understood by the batch job management system, is generated by the G-REX server before the job is launched. Currently, only SUN GRID ENGINE (SGE, http://gridengine.sunsource.net/) is supported, but there are plans to add support for the PORTABLE BATCH SYSTEM (PBS, http://www.openpbs.org/) and CONDOR (http://www.cs.wisc.edu/condor/) in the near future. It is not necessary for the G-REX user to know how to write job submission files for all these different systems. The G-REX server submits the file to the job management system by executing the relevant command (e.g. `qsub submit_file.sh` in SGE or PBS). The job submission file contains the path to model launching scripts provided by the grid administrator, containing commands (such as `mpirun`) used to start the model running. After launching the model, the G-REX server checks the job’s status at frequent intervals by querying the job management system.
It is important to note that the G-REX system requires the climate models to be pre-installed on all remote clusters. Proper installation of these models is a complex and specialist task and therefore not easily automated. G-REX cannot submit a climate model run to a cluster that does not have the model already installed and configured. A model installation comprises the model’s binary executables, the launch script and job management system parameters such as queue names, which are specified in the G-REX server configuration file. The model’s binary executable files can be provided in one of two ways: the grid administrator must either build the model to provide a fixed set of binary executables, or else provide a model build service to accompany the model execution service. The latter approach is more appropriate in situations where users need to make frequent changes to a model’s source code. Building the model then becomes part of the scientific workflow, where the output from the build service, one or more executable files, forms part of the input to the model execution service. The build service can be considered to be part of the model installation because it uses cluster-specific parameters such as environment variables, library paths and compiler options.

4. Current applications of G-REX

G-REX is employed as the middleware for the NERC Cluster Grid, a 1600 processor grid of HPC clusters belonging to the following NERC research institutes: the Environmental Systems Science Centre at the University of Reading; the British Antarctic Survey; the National Oceanography Centre; the Proudman Oceanographic Laboratory; and the Plymouth Marine Laboratory. The two model run services currently installed are NEMO (Smith & Haines in press) and POLCOMS (Holt et al. 2009), which are exposed as Web services via G-REX. The NEMO 1° model was presented in the introduction, and is the service around which G-REX development has been tested. It was noted that, with data assimilation, NEMO has particularly large requirements for both input and output data, which the G-REX system has been able to handle efficiently. The NEMO 1/4° model with data assimilation actually places less demands on G-REX because of the slower production of output, and it also has been successfully deployed. The POLCOMS model will play an important part in the GCOMS project (Holt et al. 2009), which aims to simulate the world’s coastal oceans, and this has also been deployed under G-REX on part of the NERC Cluster Grid. We plan soon to install the Hadley Centre’s HadCM3 model on the Grid, for use in the decadal climate prediction project GCEP (Haines et al. 2009).

A typical use of G-REX by a scientist to run a climate model on the NERC Cluster Grid proceeds as follows. (i) The scientist prepares input files, including forcing fields, on his or her local machine. (ii) Using the information provided by the Cluster Grid’s Ganglia (http://ganglia.sourceforge.net) monitoring system, the scientist selects an appropriate compute resource. (iii) The scientist runs the relevant workflow script on his or her local machine. This is unmodified except that calls to run the model (e.g. with `mpirun`) are simply replaced with calls to `grexrun`, the desired compute resource being identified by a URL supplied as a command-line argument. (iv) The G-REX middleware automatically handles the uploading of the input files to the remote resource and the downloading of output.
files back to the user. (v) The scientist monitors the output files, using familiar analysis and visualization tools on his or her own local machine. If the run appears to have a problem, the scientist can terminate it to prevent unnecessary resource usage. (vi) When an output file has been downloaded, it will be automatically deleted from the server; this happens during the model run, preventing accumulation of output files on the remote resource. (vii) When the run finishes, the G-Rex system automatically deletes any remaining files associated with the run that are still left on the server.

5. Discussion

G-Rex is well suited to climate modelling because it addresses many of the usability issues that have led to limited uptake of grid computing by climate scientists. It is a lightweight, low-impact and easy-to-install solution that is currently designed for use in relatively small grids such as the NERC Cluster Grid. In order to apply G-Rex to larger production grids, two main limitations must be addressed. (i) Currently, all file transfers pass through the G-Rex server. A more scalable, though more complex, solution would be to support third-party file transfers using, for example, GridFTP. G-Rex's RESTful design will allow this enhancement to be made with very little impact upon existing clients. (ii) G-Rex's current security model (based upon HTTP digest authentication) may not be sufficiently secure or flexible to satisfy some resource providers, who may, for example, mandate the use of digital certificates and encryption. However, G-Rex has been designed to allow different authentication and authorization mechanisms to be plugged in (through the use of Acegi (http://www.acegisecurity.org/) security abstractions), and so it would be relatively straightforward to address this issue. A current topic of research is the use of G-Rex as an easy-to-use front-end to larger-scale grid resources such as the UK National Grid service.

The authors would like to thank Dr J. While at ESSC, for testing and providing feedback on the use of G-Rex. This project was funded by the NERC through the Reading e-Science Centre contract (ref. R8/H12/36) and the GCOMS project (contract ref. NE/C51601X/1, J. Holt et al.).

References


