Remote Access and Scheduling for Parallel Applications on Distributed Systems*

M.Tehver**, E.Vainikko***, K.Skaburskas† and J.Vedru‡

Center of Technological Competence, Tartu University, Estonia

Abstract. Design of WWW-based frontend for distributed computing systems running scheduled tasks in parallel is discussed. A model implementation for DOUG package (Domain Decomposition on Unstructured Grids) is given. The interface consists of input-output forms, task queueing-scheduling system and administrative tools. MySQL database and PHP scripts are used as building blocks for advanced computational environment accessible for groups working together in the Internet across distinct locations. Various implementational issues are discussed including system security and its testing. An approach to program development of the WWW-interfaces for computational packages is presented.

1 Introduction

The widespread development of computer networks has revolutionised the way in which computations are carried out in large scale problems. Distributed systems and dedicated supercomputers are all making use of some sort of communication between fast processors for solving huge computational problems in parallel. At the same time, from the user's point of view, access to the problem solving environments on computers has become more and more straightforward and easy to use. This is due to availability of various types of user interfaces making the environment more easy to understand, learn and use for performing complex tasks. Most of such interfaces assume user's physical presence within the range of local area network. The internet has brought to the existence a new global range of possibilities for user interactions with computer systems. HTML was developed in the beginning mostly for remote information retrieval. But the movement has been gradually towards more two-way communication between the client (the user) and server. Also, public key encryption technologies have made it possible for various new services to appear online on the Internet that need to maintain privacy, authentication and authorisation. Often these services make use of modern database technology in their core.

* This work was supported by Estonian Science Foundation reserch grant #4449.

** Department of Computer Sciences, Tartu University. E-mail: mark@eeter.fi.tartu.ee

*** Department of Mathematical Sciences, University of Bath. E-mail: masev@maths.bath.ac.uk

† Institute of Experimental Physics and Technology, Tartu University. E-mail: konstan@ut.ee

‡ Institute of Material Sciences, Tartu University. E-mail: vedru@ut.ee

In this paper we investigate possibilities of taking use of the global Internet technology for developing a WWW-interface for the packet DOUG (Domain Decomposition on Unstructured Grids [4]) which has been developed for solving large sparse linear systems in parallel. We use publicly available software tools for developing WWW-environment which is capable of more than solely solving large linear systems – the environment is able to store a series of computer experiments' input and output into a database for later access, analysis and visualisation. We discuss the design and components of such systems together with a general guide for building it up on an example of DOUG.

2 Tools for designing interactive web-pages

There are basically two general classes of web based user interfaces. First dynamic web pages with HTML forms. These web pages are dynamically created (on request) at the server and then sent back to the requesting computer. All processing takes place in the server. The second option is to use Java applets. Java applets are small downloadable binary files that are executed on a client computer. This requires special Java runtime support from WWW-browser and uses more resources on the client side. In recent years a lot of research and development has been done for very large distributed computing systems on the Internet. Complex resource managers and scheduling schemes have been developed like Condor[2], CACTUS [1]. The term 'grid computing' is used for such systems [3]. Java is often seen as a primary development language for 'grid components' as it provides a platform-independent secure operating environment. For relatively small workstation clusters that are homogenous and that are assigned only to computational tasks, grid computing is simply overly abstracted. Resource management and basic scheduling are often already implemented in parallel computational packages.

Although Java applets are more powerful, the interface we were planning to implement did not need their additional functionality and we settled on HTML form based interface. Numerous ways exist for creating dynamic web pages. Probably the most common practice is to use Common Gateway Interface (CGI) on Apache web servers. CGI is not tied to any particular programming language although Perl and C are most widely used. Unfortunately CGI is a low-level interface and requires programmer to know many technical details of HTTP protocol and OS. We wanted to exploit higher level tools that had an abstraction level between the CGI interface and grid components. There are numerous such tools. To restrict the choice, we needed the tools to prove following properties:

1. All software had to be free, actively developed and supported.
2. Tools had to be as platform independent as possible.
3. We preferred tools that were easy to set up and maintain.

We chose PHP scripting language and MySQL database system as key components that satisfied all these requirements. Both PHP and MySQL are reliable, with very large user base and good community support.

2.1 Scripting language PHP

PHP is a programming language specially designed for dynamic web pages [7]. The programming is accomplished by using special tags in HTML files (commonly called 'scripts'). On request PHP scripts are automatically processed in a Web server and as a result a regular HTML file (without any special PHP tags) is sent to the client computer. A clear advantage compared to Java applets (which can be viewed as clients in client-server environment) is that both form generation and processing can be coupled together, making changes in user interface easier to handle.

PHP is not the only language designed for WWW-programming. Other well-known languages are Visual Basic Script and JScript within ASP (Active Server Pages). But PHP stands out with excellent support for different Web servers and platform independence. Besides, being developed by open source community, it is completely free to use and supports most widespread database systems.

2.2 MySQL database technology

MySQL is an open source relational database management system that is based on Structured Query Language (SQL). For our goals MySQL database implementation seemed to fit the best. We needed moderate sized databases with fast access while we did not really need the safe transaction support that is present in larger systems. What made MySQL attractive to us was the ease of maintainance and the very good integration with PHP. MySQL has many small aggregate functions that are handy for formatting output for PHP pages. This reduces PHP code size as a lot of processing can done in SQL.

3 Designing web-interface for DOUG

Our aim was to develop an environment for parallel computations that could be used for wide range of problems with sparse matrices. The application of our choice is DOUG (automatic Domain Decomposition on Unstructured Grids) due to its ability of meeting given needs. We needed a computational package that could be run well in small to moderate sized clusters (4-16 computers). It was intended mostly for a relatively small cluster that was connected into an Ethernet or ATM network. DOUG is currently under rapid development with many fascinating features emerging in the near future. The user interface will add even more value to the list of features.

3.1 DOUG overview

DOUG is a black box parallel iterative solver for large sparse linear systems originating from PDEs with some discretisation. The solver is developed at the Department of Mathematical Sciences, Bath University, United Kingdom. The first public release of the code (DOUG 1) [5] came out in 1997. The release of the new version (DOUG 2) with many advanced features is scheduled to June 2002. Here we give a brief overview of both versions and describe input requirements needed to be able to run the code.

DOUG 1. The first version of DOUG was meant for solving finite element discretisations of elliptic partial differential equations on 2D and 3D domains using unstructured grids. The system is assumed to be given in the form of element stiffness matrices resulting in three required input files:

`info_file` providing information about the number of elements, the number of freedoms, the spatial dimension, the maximum number of freedoms on an element and the number of nodes in the finite element grid.

`data_file` listing for each element the number of freedoms and freedom numbers.

`element_file` providing the element stiffness matrix together with the right hand side vector.

After reading in the elemental matrices the elements are divided into p partitions. Domain decomposition method with small overlap (see *e.g.* [8]), namely the additive Schwarz preconditioner is used for parallel iterative solution of the system. Optionally, for better convergence coarse problem is generated automatically by DOUG. For this user has to provide additional information in the following two files:

`xyz_file` providing spatial coordinates of the nodes in the finite element grid.

`freemap_file` is needed if the numbering of the freedoms differs from the numbering of the nodes.

In addition, various options have to be passed to the solver. These include the type of preconditioner (1 or 2-level Additive Schwarz method), subsolver type, tolerance, method to use in the outer iterations (PCG, BICGSTAB, PGMRES) and others.

Parallelisation of DOUG is based on the Message Passing Interface (MPI) and is using master-slave setup. The code is written in Fortran 77.

DOUG 2. The new version of DOUG supports several other input formats of linear system. Fully assembled matrices can be used, thus enabling it to solve problems with finite difference discretisation. The files can either be in a standard sparse representation or MATLAB sparse format (.mat-files).

Another new feature of DOUG2 is an ability to solve efficiently blocked systems that arise particularly in linearisation of Navier Stokes equations. The list of new features includes also eigenvalue solver for finding a small number of eigenvalues using a class of inverse iteration methods. Part of the new features is supported in the WWW-interface as well.

3.2 WWW-Interface

Most WWW-based user interfaces are based on forms. User has to fill in basic form elements like text fields or checkboxes. For our purposes these basic elements were important but not sufficent as we needed support for uploading and downloading large datasets. Designing input forms for all input data is not a

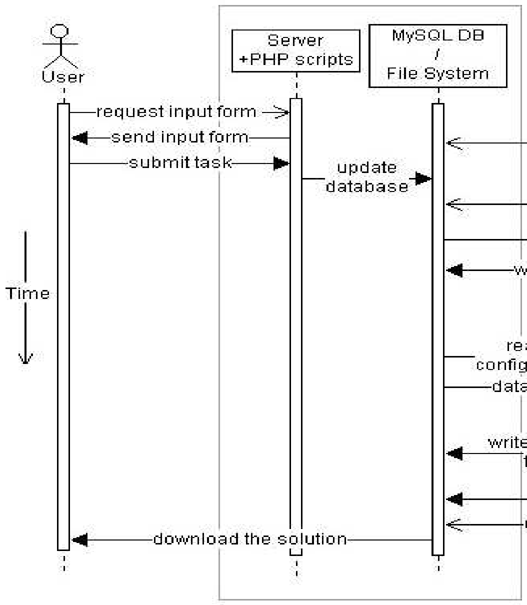

Fig. 1. Dataflow and control sequence diagram in the implementation. (Arrows with bold ends show dataflow. Arrows with regular ends indicate control.)

practical option, as most of the input data comes directly from different programs like MATLAB, Maple or from custom Fortran applications. Fortunately HTTP protocol provides support for both uploading and downloading binary and text files. The interface includes special fields for selection input files that will be uploaded to server on the form submition.

A dataflow and control sequence diagram of our system is presented in Fig.1. The system consists of two parts: input-output interface and task scheduler. MySQL database is used for storage and coordination between parts. It is based on only three entities: `User` entity describes user accounts, `Task` entity describes submitted tasks (and includes fields for keeping different date fields that track solution progress) and `File` entity is used for mapping server side files to tasks (each task may need several input files and produce several output files). Our system also includes an administration module (its functionality is not shown in Fig.1).

Input-output interface design. The central part of our user interface is the task queue. It lists all tasks that have been submitted by showing status (*waiting, running, solved, killed, frozen*), submission date, solving date, description and priority for each task. Queue interface provides direct links (via buttons) to all other scripts of the interface. Queue interface is shown in Fig.2. In the given

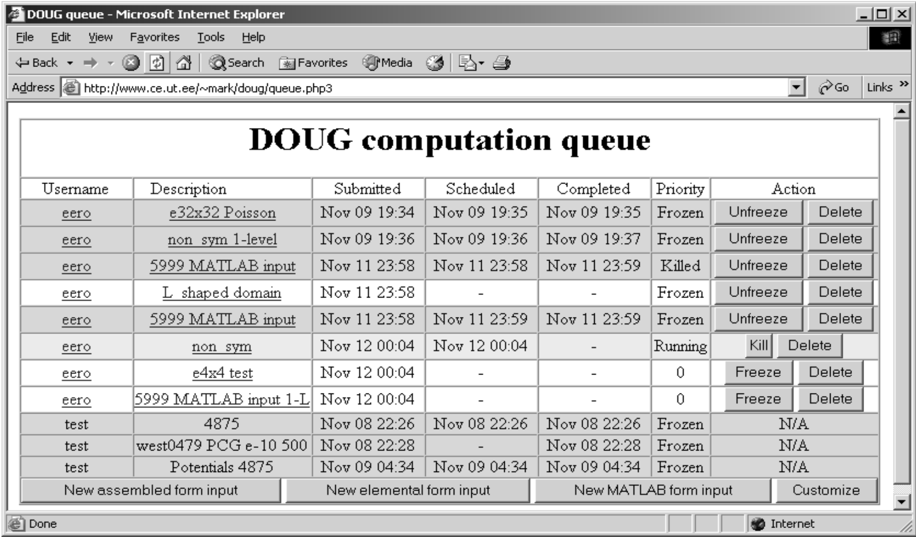

Fig. 2. The main part of the web-interface: the queue.

figure there are eleven tasks in queue, most of which are in *frozen* state (when task is solved, the status is automatically set to *frozen* and *completed* field is set to the current time). One task is currently running and two tasks are waiting. User has option to cancel a running task, or freeze a waiting task. Already solved tasks can be rerun again using the *Unfreeze* button. Note that the three last tasks can not be controlled by the current user as they are submitted by a different user.

While queue interface is completely general to tasks (and did not change after the first version was written), task input and output forms are specific to the computation engine behind the interface and had to be updated several times when new options were added to DOUG. New solving methods were added during the interface development together with new options for displaying the solver output. Currently we have three different input forms for different input data formats (elemental, assembled or MATLAB). Although they are implemented as one script (most options are still common to all formats), several options are specific to each input format.

As forms are generated in the server, client side lacks interactivity. The problem is that task input forms contain many interrelated option dependencies, some options are specific only to a single solution method. To signal this to the user (so that only relevant option combinations could be selected) JavaScript code was embedded into the HTML code. JavaScript is more closely tied to the calling HTML file than it would have been in the case of Java applets. It allows to verify and change forms on client side and is commonly used to make dynamic web pages more user friendly.

While solving a task, DOUG provides some information about solving progress. A lot of this information is not important to end users, though, some of it can be very useful. We extract convergence data from this output and allow users to view graphically the convergence curve while the task is being solved. We use gnuplot utility for graph drawing and redirect its output (which is actually a GIF file) from local file to client computer. So, the user can visually track the solution progress and stop the task solving if convergence is not satisfactory.

We have found the system to be helpful for solving a series of tasks. We have provided support for using input parameters for some previous task as a template for the new task. User can change only relevant parameters and keep the setting of most parameters. For input datasets there is an option for uploading a new dataset or keeping the old. As a result, the user need not upload subsequently the same set of input-files. After submitting the changes, a new task is created and the old one is left unmodified.

Task scheduler. Scheduler is a stand-alone Perl script that searches regularly database for new tasks, dispatches these tasks by launching DOUG when neccessary and updates database when the tasks are solved. Although the basic operation is fairly simple, there are many other operations that scheduler performs. For example, user interface has an option to kill a partially solved task. This request is sent via database to the scheduler which kills the corresponding solver process. Another option is to run multiple scheduler processes in parallel. Running several scheduler processes in parallel can give better utilization of resources.

After a task has terminated (or has been killed) it is possible that more than one process is waiting in the queue. We used a simple cost function that is based on the time that the task has spent in queue and a special priority attribute that is associated with each user and assigned by the administrator. The cost function assures the user that all tasks will be chosen within finite time interval and regular users (users with higher priority) are preferred over "guest" users.

We use the UNIX ‘crontab‘ utility to launch scheduler at regular intervals. When run, scheduler first checks that the number of running scheduler processes does not exceed a given limit, if the limit is exceeded, it exits at once. Otherwise it searches for a new task and tries to solve it using DOUG. After the task has been solved, scheduler looks for a new task and exits if none is found. By using ‘crontab‘ utility, we do not need to manually start our system after the computer has been rebooted. As the scheduler script may not be running when a new task is submitted (or can be busy), there is a latency issue. We have chosen a period of 5 minutes between each scheduler process launch (but the frequency can be changed by system administrator.) Thus it is possible that the user has to wait up to five minutes even if system is not busy. So far this has not been a real drawback as most tasks take longer than five minutes to solve.

Tools for system administration, security issues. One of our goals was to use web-interface for all aspects of our system.We use a similar web interface for the administrator of user accounts. This interface lets us create new user accounts, freeze existing accounts, add comments about users and change user priorities (which are used by scheduler as described above). Setting up a new user account takes virtually no time and the account is ready immediately. We have created a username-password system for our user accounts. When an account is created by administrator, the user is given a password that is required for login. All user scripts are protected by username and password checking. We use the HTTP 'cookies' feature for this; after authentication, the user's username and password hash are stored in a cookie. This cookie is kept only for the current browser session and not stored on hard disk. When the browser is closed the cookie will be erased and the user has to log in again next time.

Although not explicitly required by our system, users should use secure HTTPS protocol for login, uploading and downloading. We have not made this requirement mandatory because even if a password is leaked, the most that can be done is downloading solution file or deleting a task. We have not considered that to be a major security problem as input data should still be present in the client computer.

The real security issue is that the system may not expose system resources that are not associated with task solving. Fortunately PHP is quite proven in that respect if proper programming practices are followed. One basic principle of the server-side web-interface security is that all input from the user may never be directly used as arguments with basic OS utilities like *cat*, *ls*, etc. (By substituting user input to arguments, it could be possible to use the system utilities for unintended purposes.) We have carefully avoided such possibilities by checking all input data coming directly from a user before passing it on to command line utilities.

Because our system allows a much smaller set of possible actions and exposes less resources than OS, we think that creating user interface accounts is far safer than creating real usernames in OS. A more detailed log of user actions can be also stored in the server.

3.3 System testing

Our main goal was to get the WWW-interface working for a wide range of problems in different input formats. We start with an example arising from the field of cardiography using finite difference approximations and conclude with some experiments with matrices in the elemental form.

Foucault cardiography problem. An actual scientific problem, the forward problem of Foucault cardiography (FCG) [9], has been chosen as one of the main problems for verifying the work of the system developed. Simulation models of genesis of the FCG-signal involve the calculation of the distribution of the eddy currents induced by electromagnetic field in the human thorax. Combining the finite differences method with the node voltage method of electrical circuits calculation, the problem can be reduced to a system of linear equations.

For the verification of the WWW-interface we used calculation task with reduced dimensions (4875x4875 matrix with 32109 nonzero elements). The system matrix is sparse, symmetric and positive definite.

Generation of the system matrix, righthand side vector and the node coordinates for the forward problem of FCG were perfomed in the MATLAB environment. The matrices, that should be supplied to the solver, were saved in MATLAB .mat-files. So the "New MATLAB form input" form should be chosen from the "DOUG computational queue" (Fig.2) to submit the calculation task.

The submission sequence of a computational task could be as follows. One has to choose an appropriate solver type to be used (CG), the solution accuracy, max. number of iterations, supply files, choose the format of the solution file to be stored (formatted ASCII / binary (MATLAB v5/6 .mat-file)), choose number of processors and write in the description of the task to be identified by in the queue (Fig.2). Then one has to press the "Process" button. PHP script will process the submitted form and the scheduler will start the calculation task. Following the link in "Description" field of the "DOUG computational queue" table (Fig.2), user can get to the "DOUG output" form. From here it is possible to observe visually the convergence of the solution, insert comments to a protocol and, finally, to download the solution file to the local computer.

When the interface is used from a remote network, most operations are performed at once without any noticeable latency. The slowest operation seems to be uploading of input datasets to the server. In our case this takes from several seconds to a few minutes depending on the size of our datasets. Downloading a solved task took up to one minute in our case. The queue interface is displayed almost instantly when the number of tasks in the queue is less than a hundred.

Other testproblems. We have tested the system on various other problems including a set of examples coming with the DOUG 1 code (see [4]). These include Poisson's Equation on a square, L-shaped domain and on a cube. We have tested the system also on non-symmetric problems.

4 Conclusions

There are several key benefits for having a WWW-interface for a computing package:

1. Better utilization of existing resources. By creating a central WWW-server with backend computing cluster, resources for task solving can be more equally divided among users.

2. Faster learning. WWW-based interfaces are become common and users have knowledge using them. Only the important options can be extracted and presented to the end user.

3. Simple progress tracking. Computing packages usually display a lot of information about solving progress. Unfortunately much of it has no use for an ordinary user. WWW-based interfaces can display only relevant information in a suitable format or even graphically.

4. Better security. Because such interfaces are more restricted compared to user accounts, there are also less potential security holes. User actions can be more easily tracked. System resources can be divided in a more complex way than it is possible with normal user accounts.

5. Availabilty and maintainance. WWW-browser is nowadays available on most computers and no additional software is required.

The implementation of the first version of user interface was conducted in one month. This included time for learning how to use the tools, programming and preliminary testing. We see this as a good result which proves that the chosen tools were at sufficent level. We believe that there is a large set of packages that could use similar user interface. The main future direction is to research how to generalize various parts of the user interface for other computational packages.

References

1. Cactus Code, `http://www.cactuscode.org`.
2. The Condor Project Homepage, `http://www.cs.wisc.edu/condor`.
3. I.Foster, C.Kesselman, S.Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. Intl. J. Supercomputer Applications, 15(3), 2001.
4. M.J.Hagger. Automatic domain decomposition on unstructured grids (DOUG). Advances in Computational Mathematics, 9 (1998) pp. 281-310.
5. M.J.Hagger and L.Stals. Domain Decomposition on Unstructured Grids User Guide v.1.98. Department of Mathematical Sciences, University of Bath, `http://www.maths.bath.ac.uk/~parsoft/doug/userguide/userguide.html`
6. MySQL Reference Manual for version 4.0.1-alpha, `http://www.mysql.com/documentation/mysql/bychapter/`.
7. PHP Manual, `http://www.php.net/manual/en/`.
8. B.F.Smith, P.E.Bjørstad and W.D.Gropp, Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations. Cambridge University Press, Cambridge, 1996.
9. J.Vedru, K.Skaburskas, S.Malchenko. Solving Simply the Forward Problem for Foucault Cardiography. - Proc. XI Int. Conf. Electrical Bio-impedance, Oslo, June 17-21, 2001. - Oslo, 2001. - pp. 565-568.