IMPROVING PERFORMANCE OF BACKGROUND JOBS IN DIGITAL LIBRARIES USING GRID COMPUTING

Marco Fernandes, Pedro Almeida, Joaquim A. Martins and Joaquim S. Pinto
Universidade de Aveiro, Campus Universitário de Santiago, Aveiro, Portugal

Keywords: Grid computing, Digital Libraries.

Abstract: Digital libraries are responsible for maintaining very large amounts of data, which must typically be processed for storage and dissemination purposes. Since these background tasks are usually very demanding regarding CPU and memory usage, especially when dealing with multimedia files, centralized architectures may suffer from performance issues, affecting the responsiveness to users. In this paper, we propose a grid service layer for the execution of background jobs in digital libraries. A case study is also presented, where we evaluate the performance of a grid application.

1 INTRODUCTION

The amount of information available to the public has increased steeply in the last years, with the internet serving as a vehicle for its production and dissemination. Information systems such as digital libraries are responsible for not only storing very large amounts of data but also processing that information for storage and dissemination purposes.

During the process of content submission to a digital library, as with many other systems, it may be required to execute some complex and/or CPU-intensive jobs in the background, such as format conversion and information extraction, whose complexity usually increases when dealing the multimedia resources. Also, many applications require some maintenance tasks to be periodically performed. These jobs – initiated either by an operator of the system or by other jobs –, while important for the proper functioning of the system, should not degrade the overall performance and responsiveness to other, more frequent and interactive, requests.

If the document submission rate to the digital library is high or the maintenance executed at an improper schedule, these background jobs may severely limit the performance of the application, especially if this is centralized in one server. One promising solution is the usage of Grid computing (Buyya, 2007)(CSSE, 2006) to minimize the load placed on the server that deals with these tasks. Grids are generally subdivided into computational grids, data grids, and service grids (Taylor, 2005). In this paper we focused on Computational Grids: a distributed set of resources dedicated to aggregate computational capacity.

With this work, we aim to build a reusable set of complex and/or CPU-intensive services identified as necessary for digital libraries. Due to the high requirements set by these services, they should be run using an available Grid middleware. By distributing the work by several, possibly idle, machines, the server will have more available resources, thus becoming more effective on responding requests to regular users.

This paper is outlined as follows. In section 2 the objectives of the work are briefly discussed. In section 3 we present the Grid framework used. In section 4 we discuss related work. In section 5 a generic architecture for the deployment of the service layer is presented. In section 6 a case study is presented and an evaluation of the Grid performance shown. In section 7 conclusions and future work are presented.

2 OBJECTIVES

The aim of this work is to build a service layer on top of an existing Grid framework. This layer should encapsulate a set of services required for the execution of CPUintensive and time consuming jobs in digital libraries. Also, services should be made available through Web Services interfaces.

3 ALCHEMI

To implement the Grid network, the Alchemi (Luther, 2005) framework was chosen. Since Alchemi was developed with the .NET platform, it allows the creation of Grid infrastructures which can be used in Windows-class machines. However, by providing a Web Service interface, Alchemi also supports interaction with other custom middleware.

To run an Alchemi Grid application, the User node connects to a Manager node and starts the application. The Manager will then create the threads to send to the Executor nodes which registered in the Manager. Results are then collected at the Manager and delivered to the User. Alchemi supports two operation modes:

- Thread Model: the Thread Model (which we will be using) is operated by using the Alchemi API to create the Grid application. The developer must create the "local code" – to be executed in the Manager, responsible for creating Grid threads and collecting the results – and the "remote code" – the application core operations to be executed in each Executor.
- Job Model: jobs and tasks are described by specifying commands and I/O files in XML conforming to Alchemi schemas. This model is present to Grid-enable existing applications and support interoperability with other Grid middleware.

4 RELATED WORK

The use of Grid technology in digital libraries is recommended and being subject of research from work groups such as those from the DELOS Network of Excellence on Digital Libraries (Agosti, 2006).

Although with different application scopes, Grid computing is being applied by other digital library projects such as BRICKS (Risse, 2005) and Diligent (Candela, 2005), with the later being a project which aims to integrate Grid and digital library technologies.

In the context of document processing, Grid technology has been applied Tier Technologies to process high volumes of data. Ilkaev and Pearson (Ilkaev, 2005) have created and evaluated a small Alchemi Grid application developed to distribute the OCR of scanned images, which is one of the slowest document processing tasks.

5 ARCHITECTURE

Figure 1 shows a simplified architecture for the Grid applications to be deployed on the service layer. At the top, applications are developed in two classes: Manager code and Executor code. The Executor code is a standard serializable .NET class which receives a number of arguments, executes a custom code, and returns the produced output. The Manager code is responsible for the Grid threads creation (Initialization block) and any processing required prior to dispatch or after the output is received. The Manager code has two custom blocks encapsulated: Requirements and Decision.

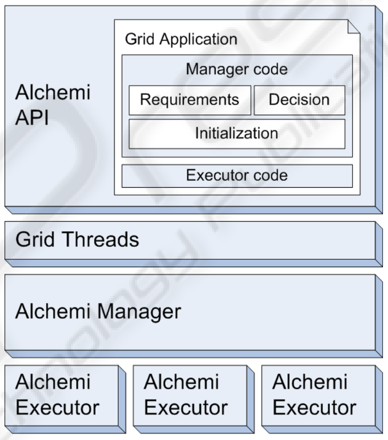

Figure 1: Alchemi Grid application architecture.

5.1 Requirements Block

Some tasks are self-contained, such as mathematical functions which require only some embedded libraries. Others, however, require the interaction with external libraries, applications (such as Office, Acrobat), platforms (.NET, JRE) or services.

These dependencies can be defined in a service deployed in the framework so that upon execution the Requirements block will retrieve the list of available Executors which satisfy the criteria. This operation can itself be implemented by using a small Grid application which will run a number of tests and store the results in a cache at the Manager. Whenever a new dependency is created, a new test must be added to the application.

A second group of requirements is hardware related: even the smallest possible work unit of a grid application may require an amount of free memory or disk space which might not be available in the Executor. Since these parameters vary with time, the Manager must periodically retrieve information from its Executors.

5.2 Decisions Block

Most standalone jobs cannot be directly migrated to a Grid environment, since they were not designed for parallel execution. Changing a job from standalone execution to its Grid equivalent may introduce large overheads and delays:

- it is usually required to pre-process and prepare input data before distributing it;
- large amounts of data may have to be transferred to and from Executors, making bandwidth an important variable;
- output data from different Executors may have to be processed in order to achieve the desired output.

With this in consideration, we find it is important to have a rule-based decision block on our service middleware which functions on top of the Grid framework.

For each service to be deployed, a set of rules and conditions can be supplied to the Decisions block. For instance, if it is expected to have a performance gain only if the number of Executors is larger than X, the decision block can dispatch the task to only one machine if there are not X Executors available. On the other hand, a service may present a performance gain which decreases as we add more Executors. For instance, some services may present a poor performance gain when increasing the number of Executors by one (for a specific value). Since there is a trade-off of using more resources, therefore decreasing availability for other simultaneous requests, a gain threshold should be defined.

Services can then be characterized by defining minimum and maximum threshold values. These thresholds are mainly affected by two variables:

- the computation to communication ratio – due to the high latency of an internet environment usually only applications with a high ratio are suitable for Grid deployment;
- the overhead involved in the creation of a distributed application – modifying a standalone application may require additional processing of input and output data.

6 CASE STUDY

In order to test the proposed architecture in a real word scenario, we developed a Grid application for an integrated digital library and archive (Almeida, 2006).

6.1 The Problem

The digital library in question has several distinct collections, with documents ranging from books and theses, posters and photographs, videos and music. One of the most computationally demanding tasks performed in the system is the submission of a new digital manuscript (book, thesis or dissertation) in the PDF format. Unlike many digital libraries, such as DSpace (Tansley, 2003) this digital library converts such documents into image and text files upon submission. This approach has some advantages:

- the viewer displays one image (page) at a time, making it possible to define granular access rules, constrain page views, and making it harder for malicious agents to crawl and retrieve entire collections of documents;
- users do not have to download the entire documents, which may reach hundreds of megabytes, in order to see them;
- it is easier to perform full-text searches on the document with per-page results;

The disadvantage of this approach is the increased complexity at the submission time. This task alone involves a number of jobs which must be executed sequentially:

- generation of a unique identifier (common to all document submissions)
- conversion of the PDF into a high quality image format;
- resizing of the images into lower resolution files to display on the web
- text extraction from the PDF, either “directly” or by using an OCR process (this can be eventually executed in parallel with the second job);

This task is specially suited for Grid computing, not only because of its complexity but especially because format and image conversions are very CPU intensive. Also, this digital library in case is currently using a single-threaded PDF conversion library, which limits simultaneous PDF submissions to one per CPU.

6.2 The Application

The migration of a standalone job to a Grid environment can be rather complex. For instance, the PDF to images conversion cannot be directly transposed into a Grid application. In order for it to work with more than one Executor, the PDF must first be split into at least N smaller PDF files, where N is the number of Executors and is larger than the number of pages. This introduces an overhead which did not exist in the single Executor scenario.

From the subtasks of the above submission process, only one can be almost directly migrated to our service layer: the image resizing/conversion. This is the task we will evaluate with our grid application. What the application will do is simply divide the number of images to convert by the number of Executors and dispatch them (a more intelligent method could make an uneven distribution according to the Executors processing power, but for simplification purposes this was not implemented).

The Executor code of the application is extremely simple. Each agent receives the images as byte arrays and the library needed for the conversion. Then, the process performs the in-memory conversion of the images and returns them also as byte arrays. The Manager code performs the dispatch of groups of images to the Executors and, upon completion of each, saves the output files in a local directory.

6.3 Evaluation

The application was tested using the images from three different PDF files against a number of Executors varying from one to eight. The application was set to convert PNG images to the JPEG format. Since we did not have eight similar desktop computers, the test was conducted starting by adding the best computers first and the worse later. Since the test was ran with machines connected to the University's intranet, the network latency is small.

Table 1 shows the performance of the application when run from standalone mode to eight Executors. As can be seen, the smaller document presents the least predictable behavior. This is an expected result, since with so few images sent the application is more vulnerable to network fluctuations and variations of available memory and processing power. As we move to the largest document the performance gain becomes more predictable.

Table 1: Time spent in the document conversion with different number of Executors.

Executors	**document 1 24 pages, 1.4 MB**	**document 2 50 pages, 3.2 MB**	**document 3 482 pages, 37.2 MB**
1	28.73 s	64.81 s	402.65 s
2	14.99 s	34.94 s	210.55 s
3	11.99 s	26.76 s	152.66 s
4	9.74 s	21.45 s	146.83 s
5	11.07 s	18.71 s	122.35 s
6	8.20 s	17.91 s	104.08 s
7	9.21 s	13.27 s	91.71 s
8	5.93 s	14.32 s	81.30 s

Figure 2 shows perhaps the most interesting result we can extract from the evaluation. If we normalize the time spent in each stage as a fraction of the time spent in the standalone scenario, we realize the behavior is almost identical with all documents.

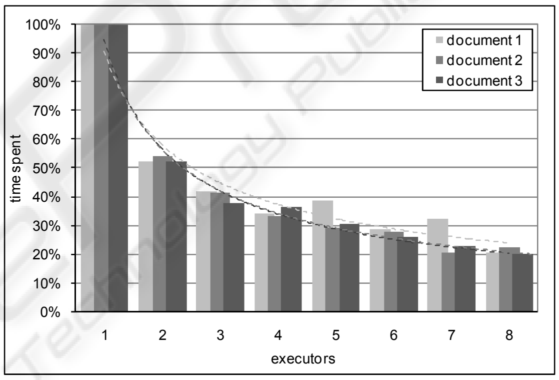

Figure 2: Average task duration with different number of Executors as a fraction of the single machine completion time.

6.4 Analysis

This is one of the simplest application scenarios we could have chosen for our first grid service, since there is almost no processing needed at the manager. There are also no requirements we need to set on the Requirements block: the only library needed for the image conversion is transferred to the Executors at runtime, there is no need for hard disk space, and memory usage will not exceed a few megabytes.

Regarding the Decision block configuration, we must determine if, depending on the document to be converted, there is an optimal number of executors we should aim at. The average time required for the conversion of N images using our grid application with E Executors is the sum of the time spent uploading the files, converting them at the slowest Executor and downloading the output to the Manager:

$$T_{N,E} = T_{up} + T_{conv} + T_{down} = N[(S_{PNG} + S_{JPG})/B + O_{max}/E + K_{OH}E] \quad (1)$$

where S_x is the average size of an image with the format X (for the dimensions used in the conversion), B is the available bandwidth, O_{max} is the maximum time spent by the slowest Executor to convert one image, and K_{OH} is an overhead factor needed to create and consume the threads. This is merely a simplification of the expected behavior, and numerous variables fluctuate over time, mostly due to the network latency.

If we consider the thread overhead to be negligible for small values of E and N, the equation becomes of the form:

$$T_{N,E} = k_{1,N}/E + k_{2,N} \quad (2)$$

which indicates that the optimal performance point is reached when E is maximum, E=N. However, the gain in performance becomes less apparent as we add more Executors (in the case of document 3, there is 48% gain when increasing E up to 2 and only 11% for E=8).

From the above Decision block discussion, we should define a minimum gain for this conversion service. Since it is difficult to perform this calculation with so many variables which fluctuate in an unpredictable manner, we used our set of results. Hence, for a minimum 15% performance gain, we defined a maximum threshold (number of executors) of 6 and no minimum.

7 CONCLUSIONS

We have presented a generic Grid based layer which we will develop in order to offer generic and stateless services to be executed in a parallel manner. We also have shown a simple case study and how the performance of a basic service has been greatly improved has we fully used the Grid capabilities, executing up to five times faster.

Analysis has been conducted for a very simple application scenario. Some work is yet to be done to create robust and intelligent blocks for the Manager. As a future work we plan to define a minimum set of simple application agnostic services required for digital libraries. We will analyze their requirements and define the rules for their Decision blocks. We all these settings well established, we shall implement the applications and create Web Services which will serve as the interface to execute the services.

ACKNOWLEDGEMENTS

This work was funded in part by FCT – Portuguese Foundation for Science and Technology – grant number SFRH/BD/23976/2005.

REFERENCES

Agosti, M. et al, 2006. D1.1.1: Evaluation and comparison of the service architecture, P2P, and Grid approaches for DLs. Technical Report, DELOS – A Network of Excellence on Digital Libraries.

Almeida, P. et al, 2006. SInBAD - A Digital Library to Aggregate Multimedia Documents. In ICIW'06: International Conference on Internet and Web Applications and Services. Guadeloupe, France.

Buyya, R., 2007. Grid Computing Info Centre (GRID Infoware). http://www.gridcomputing.com.

Candela, L., Castelli, D., Pagano, P., 2005. Moving Digital Library Service Systems to the Grid. In Türker, C., Agosti, M., and Schek, H. (Ed.), *Peer-to-Peer, Grid, and Service-Orientation in Digital Library Architectures* (pp. 236-259). Springer, Berlin.

CSSE - Department of Computer Science and Software Engineering of University of Melbourne, Australia, 2006. The GRIDS Lab and the Gridbus Project. http://www.gridbus.org.

Ilkaev, D., Pearson, S., 2005. Analysis of Grid Computing as it Applies to High Volume Document Processing and OCR. Technical Report, Tier Technologies, USA.

Luther, A., Buyya, R., Ranjan, R., Venugopal, S., 2005. *High Performances Computing: Paradigm and Infrastructure*. Wiley Press. New Jersey, USA.

Risse, T. et al., 2005. The BRICKS infrastructure – an overview. In Proc. of 75th Conference on Electronic Imaging, the Visual Arts & Beyond (EVA 2005).

Taylor, I.J., 2005. *From P2P to Web Services and Grids – Peers in a Client/Server World*. Springer-Verlag. London.

Tansley, R. et al., 2003. The DSpace institutional digital repository system: current functionality. In Proceedings of the 2003 Joint Conference on Digital Libraries (JCDL'03), IEEE Computer Society, 87-97.