

ANALYSIS OF GRID COMPUTING AS IT APPLIES TO HIGH VOLUME DOCUMENT PROCESSING AND OCR

By: **Dmitri Ilkaev, Stephen Pearson**

Abstract: In this paper we analyze the concept of grid programming as it applies to document imaging and processing. The paper mostly focuses on OCR processing mapped to a simple grid configuration. Alchemi .NET grid framework was used as the grid engine. Basic dependences of the OCR engine behavior in the grid configuration are discussed and the increase of the document processing rate is demonstrated. Based solely on the small sample and test regimen employed, we demonstrate that throughput in excess of 150% can be achieved even in this CPU intensive process by employing grid strategies.

1.0 Introduction

Grid computing (or the use of a *computational grid*) means applying the resources of many computers in a network to a single problem at the same time - usually to a scientific or technical problem that requires a great number of computer processing cycles or access to large amounts of data. As you can see from this definition, a task that involves multiple computers, multiple parallel transactions or other events and processes that repeat again and again in the system and happen concurrently, is a good candidate to be run using the computing grid.

In this paper we will try to apply the concept of grid computing to one of the typical tasks in the document processing, that of OCR. Before doing that, let's look at the typical high level configuration of the document processing system shown below.

2.0 High Volume Document Processing System

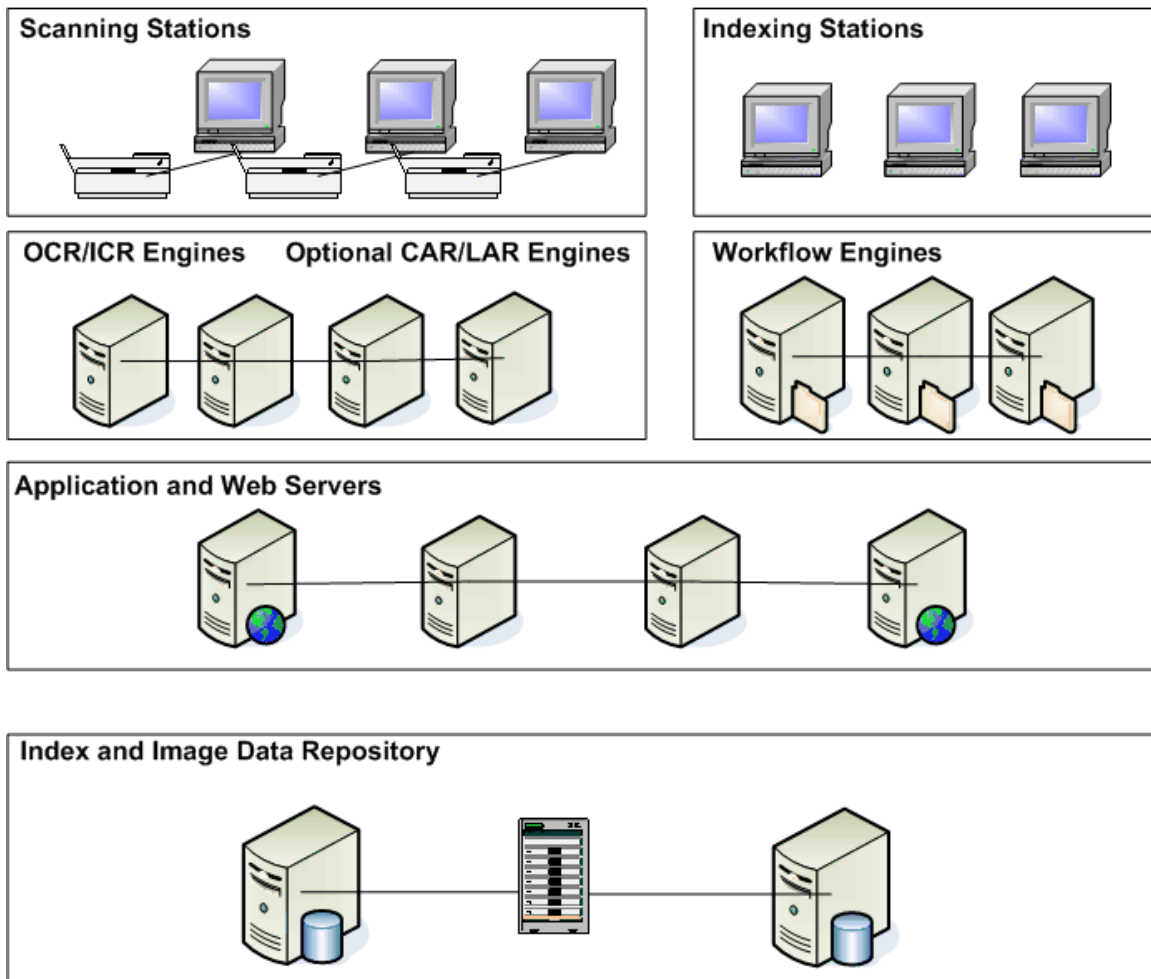


Figure 1. Conceptual Architecture of High Volume Document Processing System

There are multiple scanning stations deployed to be able to handle the high daily loads of the documents to be processed as well as multiple indexing/keying workstations where operators are looking at the document images and doing manual entry data of the index fields. OCR (Optical Character Recognition) is a powerful mechanism to increase the processing throughput especially in the case of the machine typed forms and documents with the defined layout. OCR probably is the slowest process in the document processing (that's why we pay a special attention to it later on) and in order to be able to OCR multiple scanned documents, the system will require multiple instances of OCR engines. These engines can be enhanced with the handwriting recognition (ICR), different OCR engines with the voting mechanisms of results evaluation to improve accuracy of the recognition results and CAR/LAR (Courtesy and Legal Amount Recognitions) engines for the check processing. Additional features of form recognition, bar-code evaluation and language differences are also involved. Another important part of the system is the workflow engine which checks the status of the document and based on the nature and values of the recognized data fields will put the document into the next queue or route it to the right person or next step in the workflow. Thus, to be able to process a high number of the documents in parallel, we need to have multiple workflow engines. Another multi server layer, shown on this picture contains application servers (the rest of the business logic, image retrieval, archival, etc.) and Web Servers (Web based access to the document processing functions). Each layer or module of the system can be scaled independently and of course there is always a choice between horizontal and vertical

scaling: we can add more servers or just more CPU to the existing servers or can try to run multiple instances of the engines on the same box.

Without going into too much detail regarding the design of the document imaging system, we present this information only with the purpose to show that this is complex environment which consists of multiple servers hosting different parts of the application and these modules (and engines) are different in their nature, speed and regimes of their work.

3.0 Grid Computing System

The General grid computing system is well described in [1] below, and in a wide variety of other sources. Typically, users utilize the API's and tools to interact with a particular grid middleware to develop grid applications. When they submit a grid application for processing, units of work are submitted to a central controller component which coordinates and manages the execution of these work units on the worker nodes under its control.

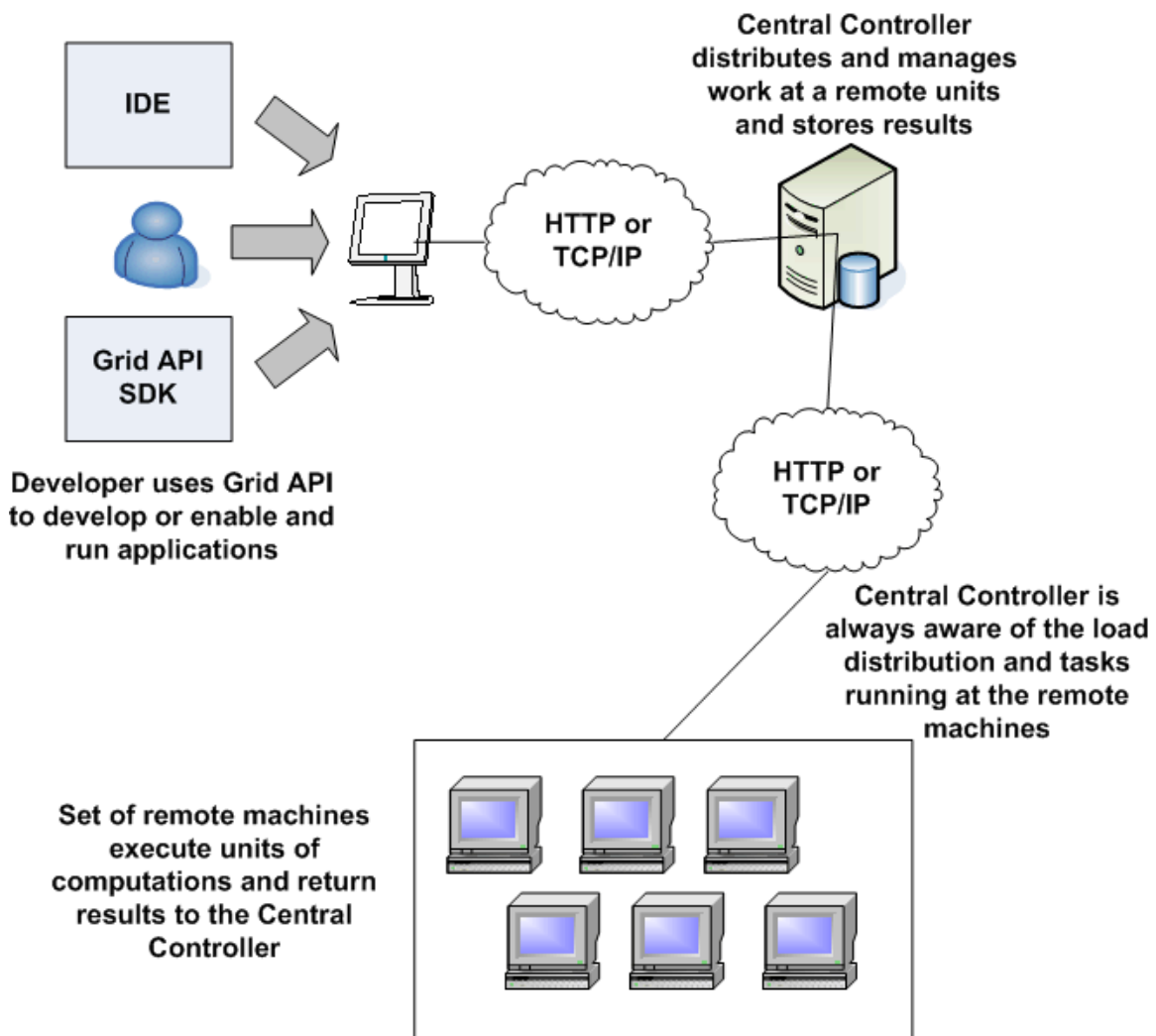


Figure 2. General Approach to Grid Computing

There are multiple grid engines and toolkits, and for the purposes on this article we selected the grid technology from Alchemi (www.alchemi.net) mostly because of its open

source nature and .NET as the base technology for the grid services and interfaces. At the end of the paper we'll reference to our experience with other grid vendors.

Alchemi grid framework architecture is well described in [1] and [2] below, so here we present the basic information about the technology and the way it works. Alchemi follows a paradigm in which a central component dispatches independent units of parallel execution to workers and manages them. This smallest unit of parallel execution is a grid thread, which is conceptually and programmatically similar to a thread that wraps a "normal" multitasking operating system thread. A grid application is defined simply as an application that is to be executed on a grid and that consists of a number of grid threads. Grid applications and grid threads are exposed to the grid application developer via the object-oriented Alchemi .NET API. Figure 3, below represents the high level architecture of the Alchemi's grid framework.

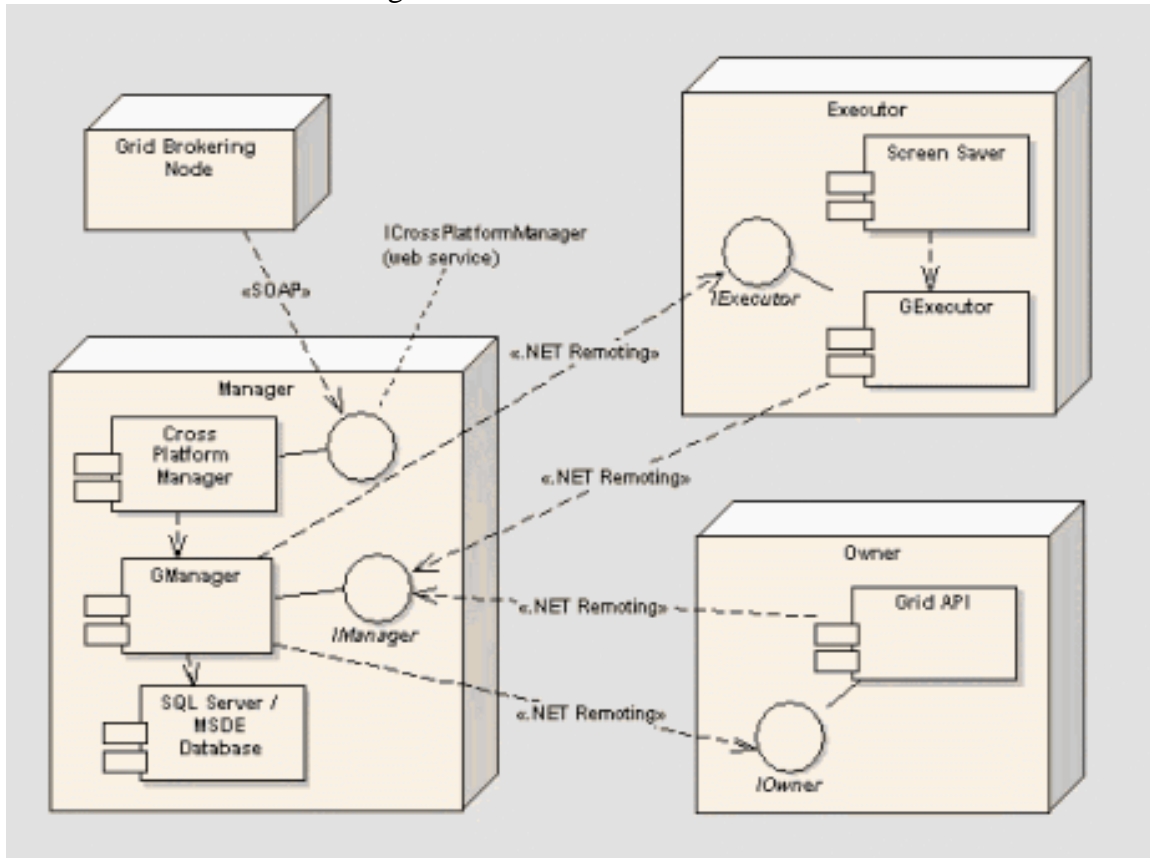


Figure 3. High Level Architecture of Alchemi's Grid Framework

There are three main components:

The Manager manages the execution of grid applications and provides services associated with managing thread execution. The Executors register themselves with the Manager which in turn keeps track of their availability. Threads received from the Owner are placed in a pool and scheduled to be executed on the various available Executors. A priority for each thread can be explicitly specified when it is created within the Owner, but is assigned the highest priority by default if none is specified. Threads are scheduled on a Priority and First Come First Served (FCFS) basis, in that order. The Executors return completed threads to the Manager which are subsequently passed on or collected by the respective Owner.

Alchemi simplifies the development of grid applications by providing a programming model that is object-oriented and that imitates traditional multi-threaded programming. Developers deal only with application and thread objects and any other custom objects, allowing them to concentrate on the grid application itself without worrying about the "plumbing" details. Furthermore, this kind of abstraction allows the use of programming language constructs such as events between local and remote code. All of this is offered via the Alchemi .NET API. The two central classes in the Alchemi .NET API are **GThread** and **GApplication**, representing a grid thread and grid application respectively. There are essentially two parts to an Alchemi grid application. Each is centered on one of these classes:

- code to be executed remotely (a grid thread and its dependencies) and,
- code to be executed locally (the grid application itself).

The syntax and the logic of the API is described in the Alchemi SDK which can be downloaded from the Alchemi's Web site. The grid framework SDK also comes with the command line interface allowing to manually submit jobs to the grid processing.

4.0 Approach

As we mentioned above in the modern systems of document processing the OCR engine normally is the slowest component in the whole architecture and may become a bottleneck in the processing of the high volume of the documents through the system. Our approach aimed to demonstrate the application of grid computing to the work of OCR engine, ignoring many of the issues regarding imaging workflow such as synchronous and asynchronous workflow, mainframe integration, and multiple image presentations for indexing and exception processing. We understand that the configuration of the document imaging system varies from installation to installation and the number of servers and set of the engines depends on many different factors like nature of the processed documents and their types, percentage of the machine typed documents and form based documents as well as form layout specially designed to be OCR'd. We look at our approach and results mostly as the proof of the concept illustrating the benefits of the merge of the OCR with the grid computing, the way it can be achieved and some processing results. In our results discussion we are less focused on the absolute numbers (will it take 1 or 5 seconds to recognize the page of text), since they depend on the characteristics of the servers (CPU and RAM) as well as additional factors. We pay much more attention to the changes observed in the performance, trends of these changes and general understanding of how certain processes can be improved by the grid computing. We also hope that the description of our approach will also help other people to map their current image processing challenges to the benefits provided by the grid frameworks.

The architecture of our "proof of concept" environment is shown in Figure 4., below.

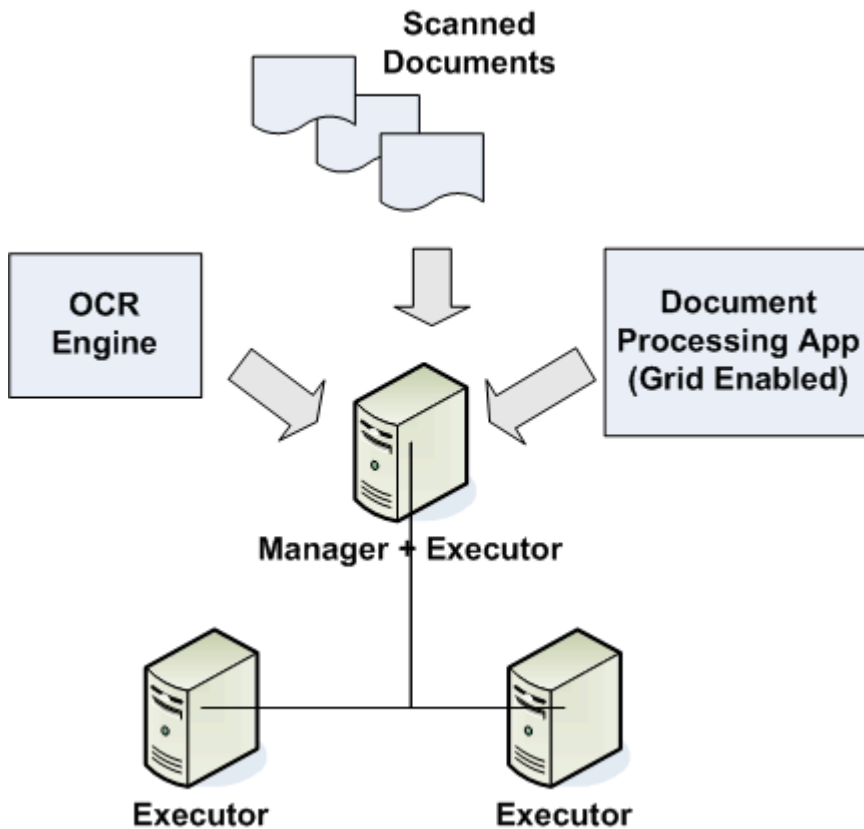


Figure 4. Environment Setup

We have the main machine with Alchemi's Manager and Executor being setup on it (MS SQL Server was used to set up the Manager as well). We also have 2 additional workstations with the Executors which can be on and/or off to try different configuration of this simple grid. All machines are on the same subnet and use regular TCP/IP connection in order for the Executors and manager to communicate. The main workstation contains the collection of pre-scanned documents (tif images), the OCR engine (we had tried Caere and ABBYY) and our custom application taking images in a single or batch mode and running OCR processes against them. We also added timing stamps to the end of OCR results to measure the performance of these operations.

The Alchemi Console shows the state of the Executors running on the grid (see Figure 5., below) and performance of the grid services (will be shown later).

The screenshot shows the Alchemi Console interface. The title bar reads 'Alchemi Console'. Below the title bar, there are tabs for 'System', 'Applications', 'Executors', and 'Users'. The 'Executors' tab is selected. Below the tabs is a table with the following data:

executor_id	host	port	usr_name	is_connected	is_dedicated	cpu_max	cpu_totalusage
676d3172-0044-4835-97eb-6a5fe0d80a4b	UAZL010364	9001	executor	True	True	1598	0.00181994
9bbaeb72-b7a9-454c-bd2b-6ff0263bac4e	DG7QL41	9001	executor	True	True	2392	1.32889e-005

Figure 5. Console Showing the Current Grid Configuration

5.0 Results And Discussion

We ran the typical image processing application in a different regimes (single image and batch mode) on a grid-less and grid-enabled environments. The typical Console screenshots are presented below in Figures 6 and 7. As we discussed above we will be less focused on the absolute numbers of the engine performance and more focused on understanding of the mechanisms and grounds for the optimization.

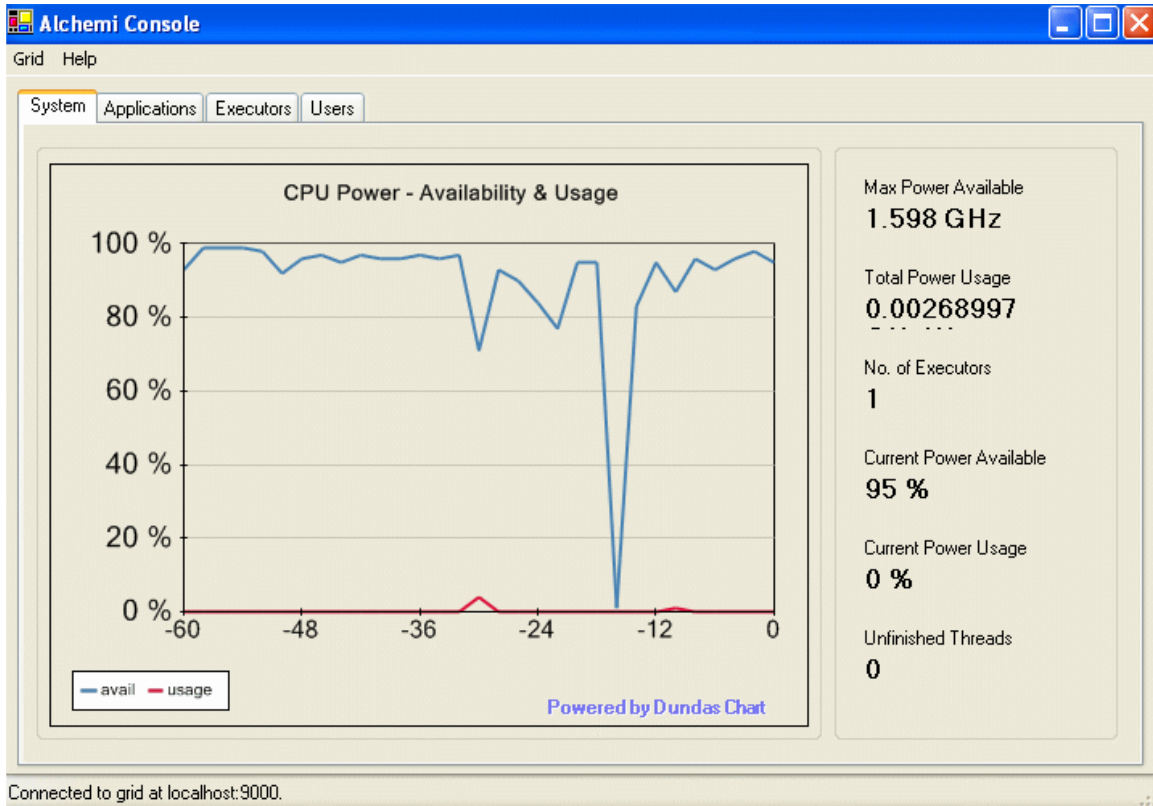


Figure 6. Single Image, One Executor

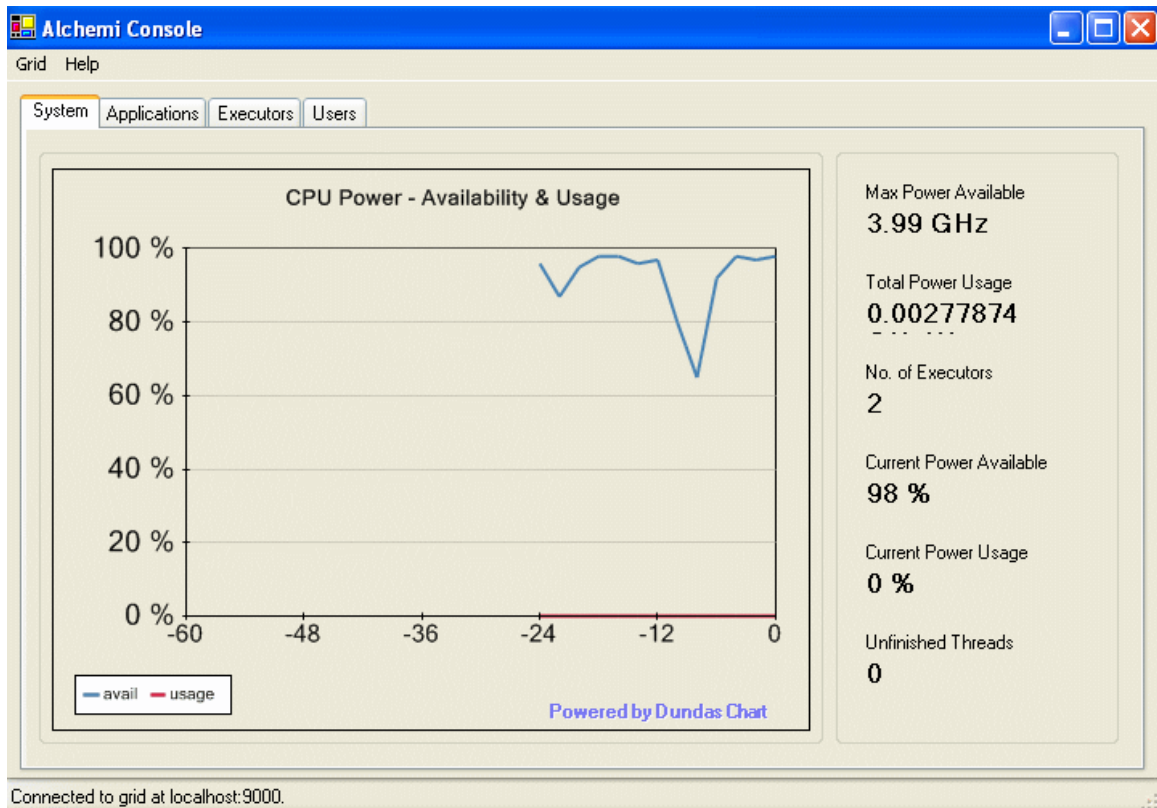


Figure 7. Single Image, Two Executors

While system behavior shows less load on the CPU for the 2 Executors (Grid really worked taking some load off the main machine and doing processing at the second one), the performance of the OCR engine was quite discouraging: for 2 Executors to process a single image it took even more time than for the configuration with the single machine. At the same time, thinking in the terms of parallelization (what's the grid for!) it's easy to understand such behavior: when processing a single image, there is not too much space for such parallelization and the application spends more time trying to take some work, move it over the network, process it and then return to the main machine. You can also see that the grid services (red line on the graph) are not really utilized much. To prove this explanation we also disabled Executor on the main machine but left it connected on the other station. The graph of the system behavior now looks as following:

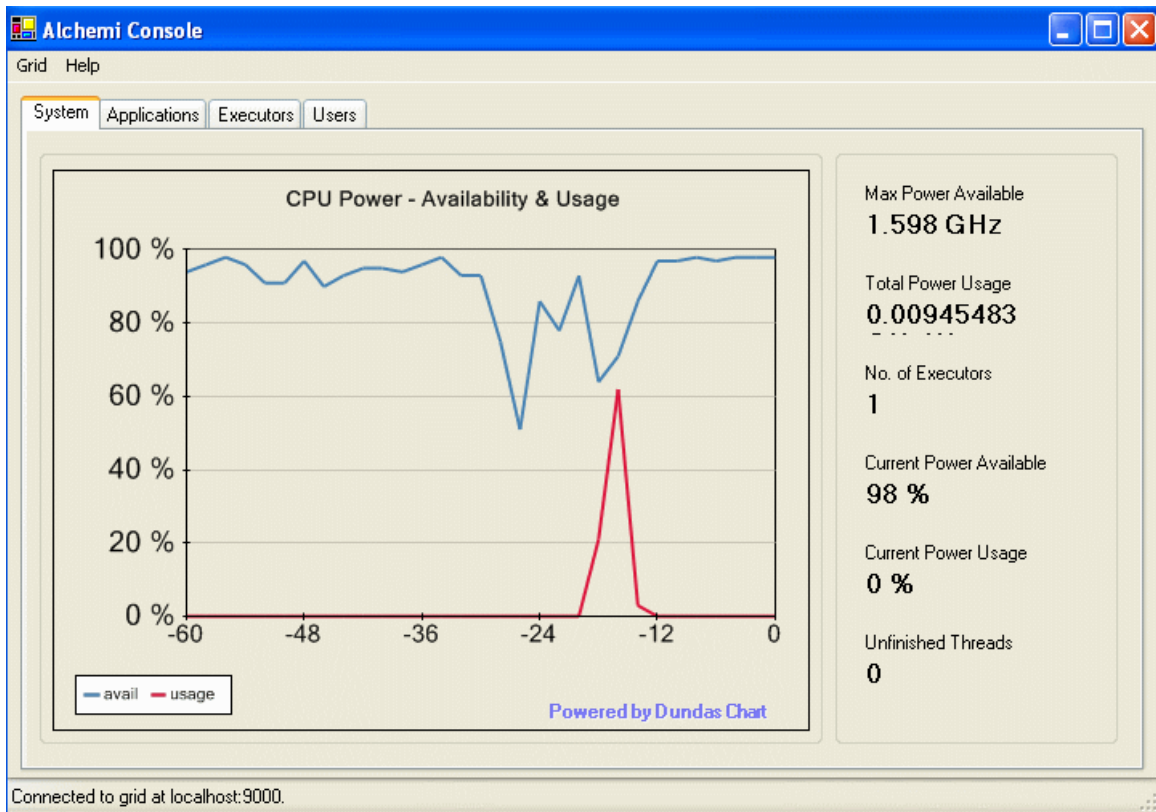


Figure 8. Single Image, One Executor (on a different machine)

This change did not affect the throughput of the OCR engine, however we see that it resulted in the more processing from the grid service (red line) which was required to push the work from the main machine where there was not Executor to a different workstation.

These dependences become totally different when we start processing images in the batches (even as small as 3 or more images in the batch). Now, we can guess that the level of parallelization is high: the grid engine constantly receiving new images which, in order to be processed, have to go to the next available machine. This activates grid processing service which results in the more effective image recognition rate. On the Figures 9 and 10 below we show the system behavior for the batch of 5 images for a different number of Executors. The total speed of the batch processing is higher for the grid configuration with higher number of Executors.

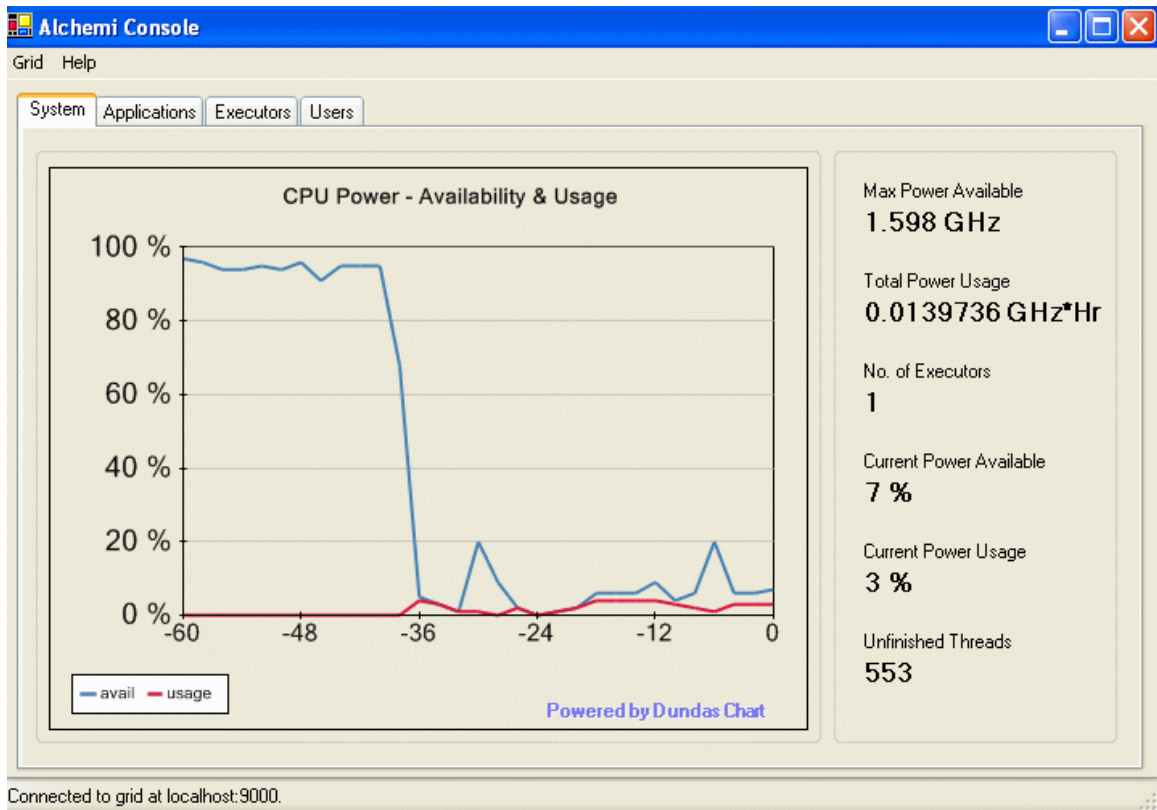


Figure 9. Image Batch, One Executor

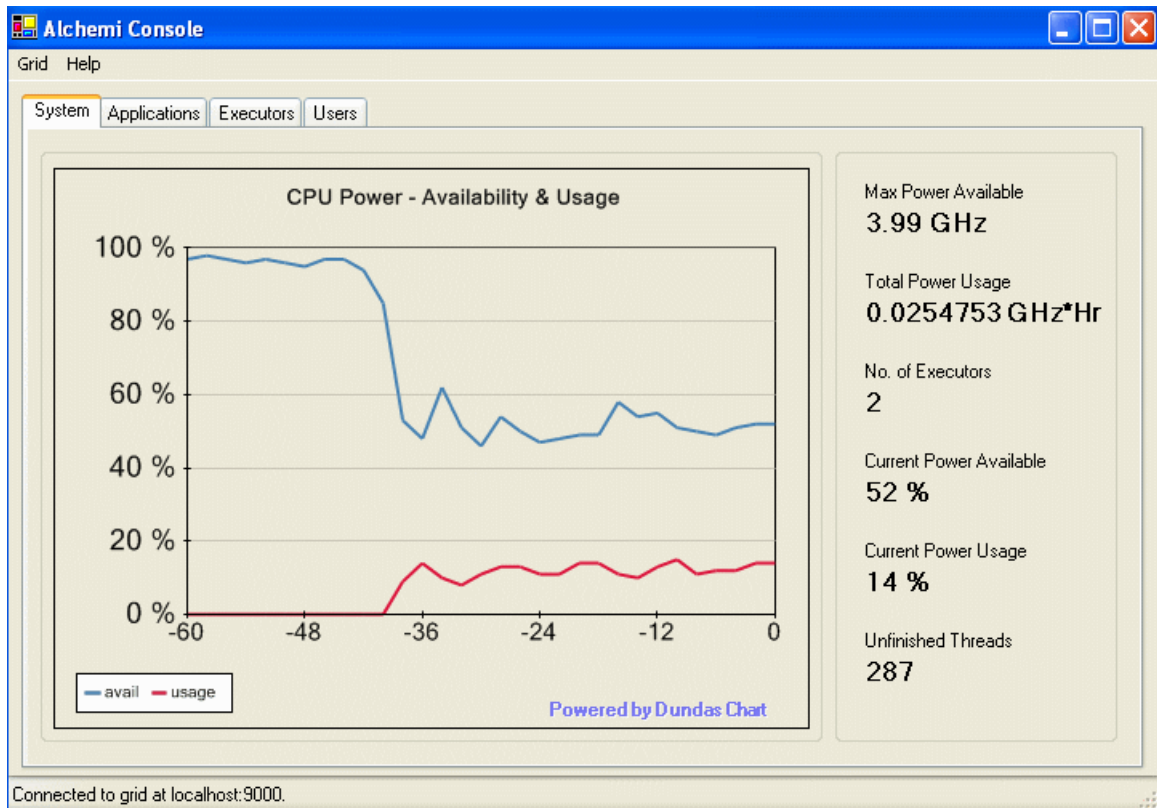


Figure 10. Image Batch, Two Executors

The recognition rate (in relative numbers with ratio to the rate on a single machine) dependence on the number of Executors is presented below.

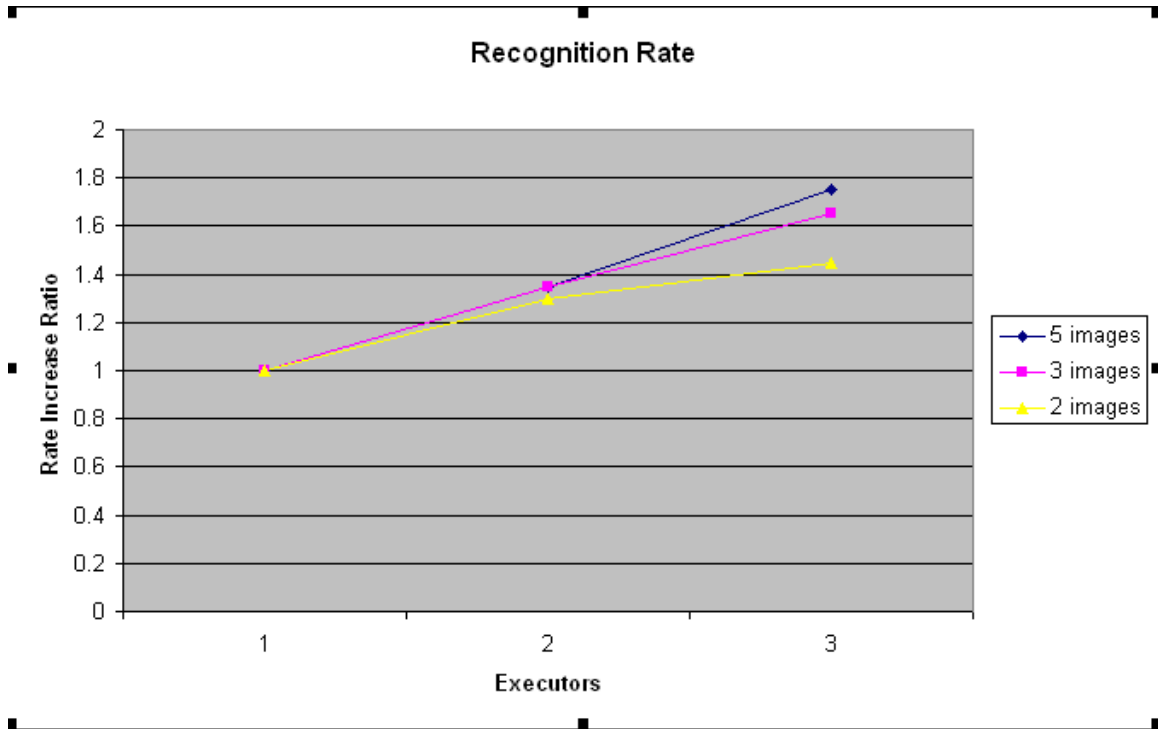


Figure 11. Recognition Rate Increase With Number Of Executors and Different Batch Size

We do observe a recognition rate increase and with the larger number of images in the batch (larger level of parallelism for the grid engine) the higher the gain in the recognition rate. We should mention here the effect that we call a “localization” effect for the grid engine. It’s easy to understand that by blind increase of the number of Executors you can not guarantee a performance increase in your processing. The level of parallelism (the number of concurrent tasks) should be sufficient to trigger the work of at least several Executors even taking into account the “losses” when the part of the work travels back and forth across LAN to be processed on remote machines. We call this a “localization” effect due to the fact that the speed of the processing on a single machine should be evaluated against the speed on how quickly the next job (next image or even batch) will be submitted to the grid engine. If the new task will be submitted to the grid with the speed slower than the processing speed of the Executor, the Executor will be done with its current job and will immediately pick the next one, so there will be no need for remote Executors to participate in this work – it will be all done (localized) by the first few Executors. So, for the real production configuration we recommend to perform some capacity analysis and additional tests to identify the right configuration of the grid and reduced number of processing engines.

While all of our tests were performed using Caere OCR, we also tried to replace it with the ABBYY engine and did not see any noticeable changes. We assume that as long as OCR engines work in more or less the same way the general benefits of the grid processing will remain valid.

Naturally we wanted to confirm that our recommendations in using the grid computing for the purposes of document processing would be of a general character with less dependency on the grid engine. To validate our approach we took another grid engine, MyGrid, another open source project, see <http://mygrid.sourceforge.net/>

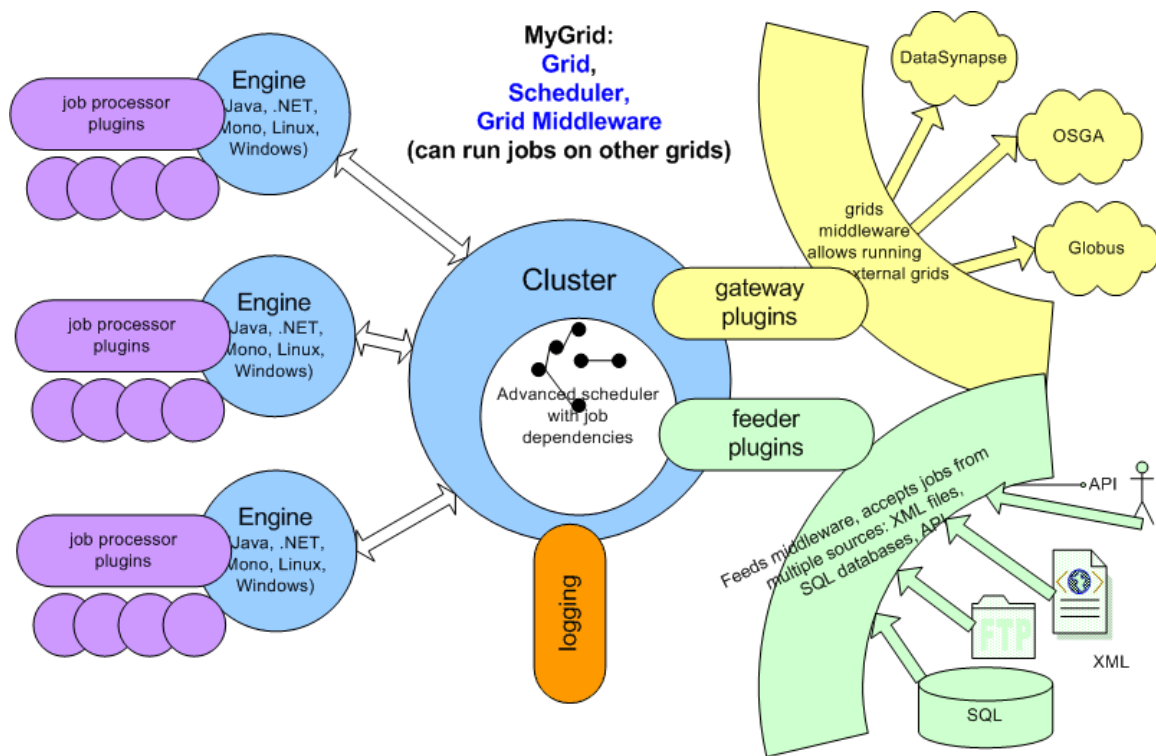


Figure 12. MyGrid Architecture

It operates with the slightly different terms: Cluster, Job Processor and comes with the SDK in order to grid enable the application. The Typical screen of the grid set up is presented in Figure 13, below..

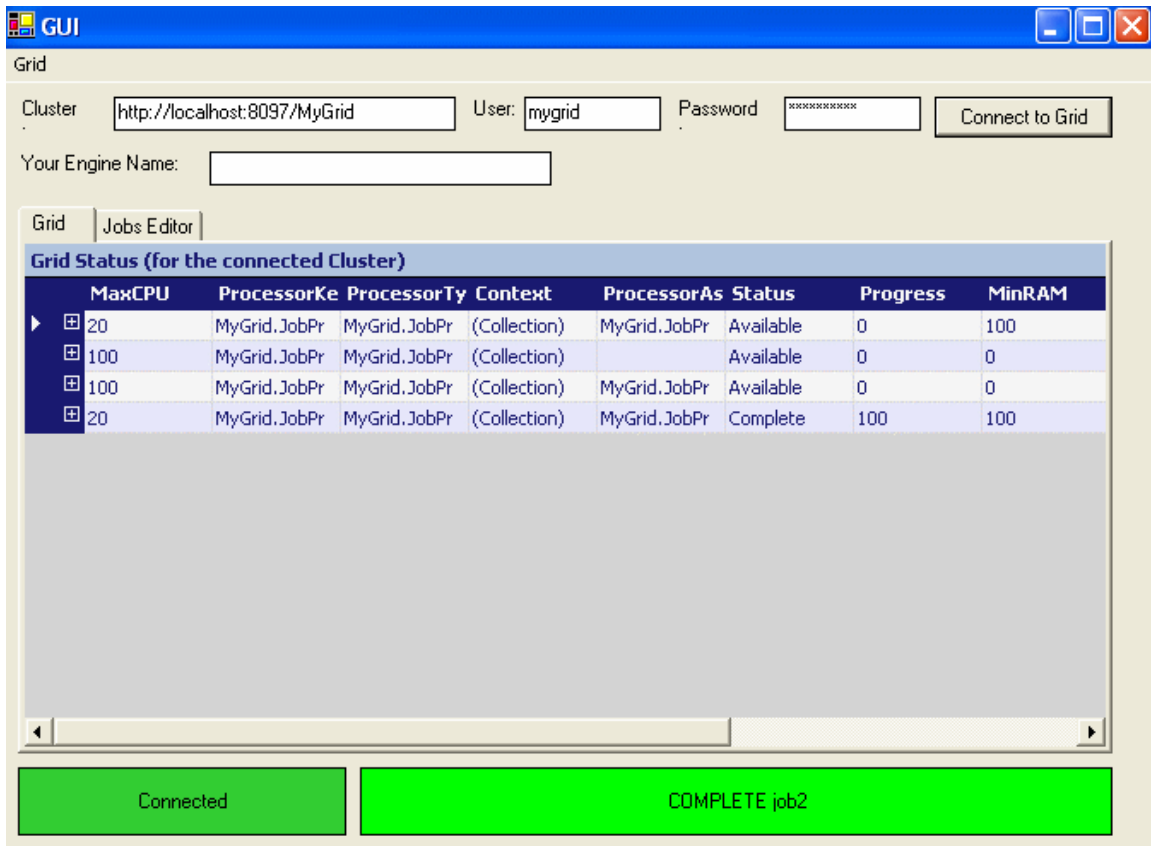


Figure 13. MyGrid Setup Screen

Together with MyGrid SDK and grid API which can be used for the grid enabling of the applications, the grid engine has a simplified way of submitting jobs through the JobFeeder.xml files which describe the batch files or executables and their arguments to be called from the Job Processors.

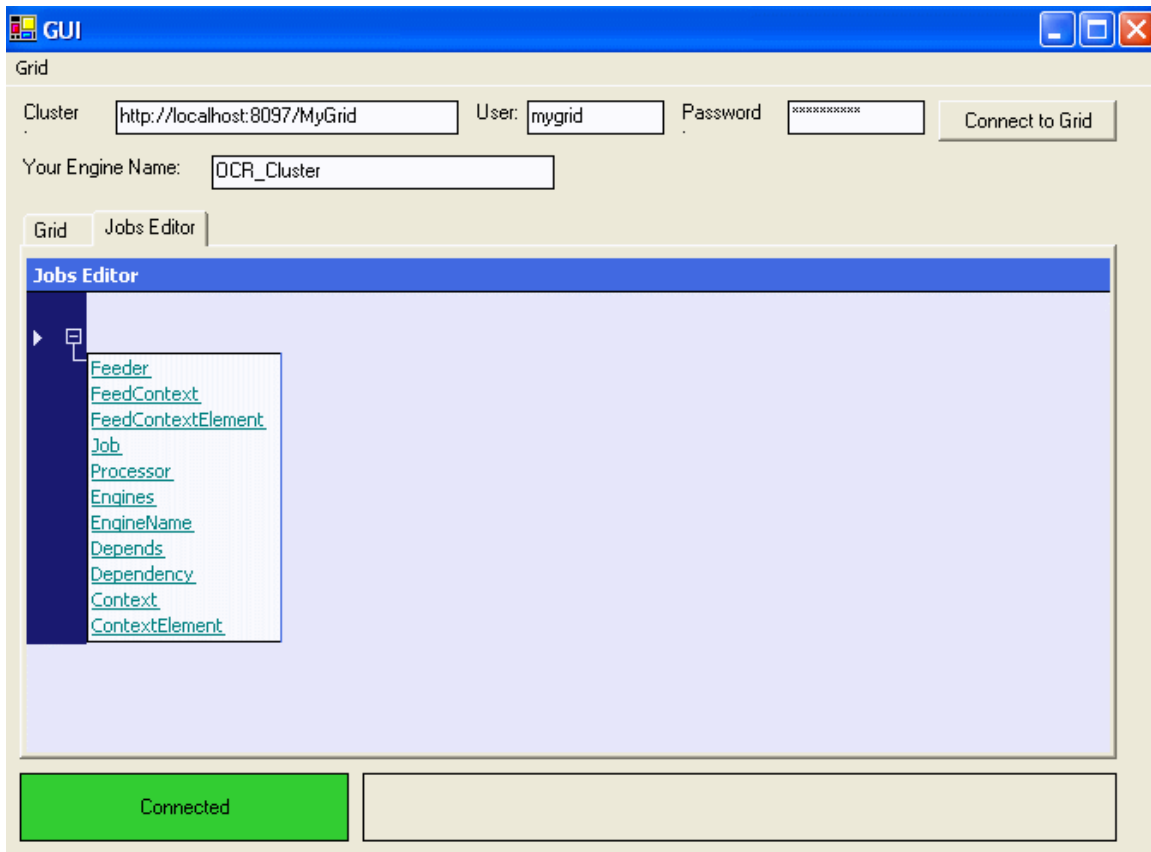


Figure 14. MyGrid Job Submission Screen

We were quite pleased to obtain the same dependencies with the Alchemi and MyGrid engines.

One of the key issues regarding any imaging workflow design is the network bandwidth available for the movement of the images across the network, in many cases, multiple times. This paper does not address this issue and a particular bandwidth requirements should be defined on a case by case basis depending on a specific environment setup and projected load of transactions.

6.0 Conclusion

We had proposed to apply the concept of grid computing to the area of document processing with the focus on OCR functions. In order to prove the concept we built a simple grid environment using Alchemi's grid framework and tested our custom application which performed standard some standard OCR functions. Obtained results demonstrated the gain in the OCR performance being applied to the grid configuration. More detailed analysis revealed additional specifics in the need to achieve certain degree of the tasks parallelization in order to feel the benefits of the grid and we also conformed the "localization" effect where due to the high processing power of the grid nodes the work remains localized on the few initials nodes and the rest of the grid doesn't not participate in the processing. We also confirmed that the dependencies discussed in the paper are of the general character and do not depend on a specific OCR engine or grid framework. The proposed approach and reviewed dependencies will help to optimize the

configuration and throughput of the typical imaging systems used in the modern document processing

References:

[1] Peer-to-Peer Grid Computing and a .NET-based Alchemi Framework
Akshay Luther, Rajkumar Buyya, Rajiv Ranjan, and Srikumar Venugopal see
<http://www.alchemi.net/publications.html>

[2] Alchemi: A .NET-based Grid Computing Framework and its Integration into Global Grids
Akshay Luther, Rajkumar Buyya, Rajiv Ranjan, and Srikumar Venugopal see
<http://www.alchemi.net/publications.html>

About the Authors:

Dmitri Ilkaev has almost twenty years of experience in software and technology development. He holds Ph.D. in Computer Sciences from Moscow Institute of Physics and Technology. Dmitri is the Chief Technologist at Tier Technologies.
(<http://www.tier.com>). He can be reached at DILkaev@tier.com

Stephen Pearson has over 15 years of imaging workflow and OCR/ICR experience including FileNet, Eastman, Open Text and IBM. He has been active as a programmer, team lead, designer and development manager on major implementations in the food service, health care, banking, air freight, government services and pension industries. Currently he works as freelance consultant and can be reached at s_pearson@mindspring.com