Journal of Universal Computer Science, vol. 14, no. 18 (2008), 3031-3050
submitted: 1/3/08, accepted: 29/9/08, appeared: 1/10/08

Comparative Aspects between the Cluster and Grid Implementations of BigBatch

Giorgia de Oliveira Mattos
(Federal University of Pernambuco, Recife, Brazil
giorgiamattos@uol.com.br)

Andrei de Araújo Formiga
(Federal University of Pernambuco, Recife, Brazil
andrei.formiga@gmail.com)

Rafael Dueire Lins
(Federal University of Pernambuco, Recife, Brazil
rdl@ufpe.br)

Francisco Heron de Carvalho Júnior
(Universidade Federal do Ceará, Fortaleza, Brazil
carvalho.heron@gmail.com)

Fernando Mário Junqueira Martins
(Universidade do Minho, Braga, Portugal
fmm@di.uminho.pt)

Abstract: BigBatch is an image processing environment designed to process batches of thousands of monochromatic documents. One of the flexibilities and pioneer aspects of BigBatch is offering the possibility of working in distributed environments such as clusters and grids. This paper presents an overview of BigBatch image processing features and analyzes the results of a number of experiments devised to compare its cluster and grid configurations. Although preliminary results were published earlier on, the new data shown here that sheds new lights onto this aspect. The results obtained exhibit almost no difference in total execution times for some grid and cluster configurations, but significant differences for others, indicating that the choice between such configurations must take into account a number of details in order to reach peak performance. Besides those, there are other qualitative aspects that may impact this choice. This paper analyzes these aspects and provides a general picture of how to successfully use BigBatch to process document images employing computers in parallel for this task.

Keywords: Cluster, grid, image processing, load-balancing
Categories: D.1.3

1 Introduction

Digital documents are replacing paperwork in organizations in every corner throughout the world. This step to be effective must encompass the paper legated documents. Thus document digitalization is the way to bridge the gap between past

and present technologies. The paramount numbers involved in document digitalization demand efficient and low cost solutions. Production-line, automatically fed flatbed scanners (such as [Kodak, 1999]) for the digitalization of batches of thousands of documents offer a viable solution to balance the cost-performance binomial. However, this process may introduce into document images a number of undesirable artefacts such as noisy borders, skew, salt-and-pepper noise, that not only damage document readability by humans, but also demands larger storage space, claims for more computer network bandwidth for document transmission and degrades OCR response. Thus, an environment that efficiently removes such artefacts is most desirable. Each digitalized image must be processed by a series of filters to get them ready for storage and later use as digital documents, and this must be done for large batches of documents, which would normally take a long time in a single computer. In general, the number of images scanned by a production line flatbed scanner in a day is several times larger than the time needed to process such images only to remove the artefacts mentioned. The use of OCR to automatically find keywords for instance is a factor about ten the filtering time. However, organizations often have many computers with spare machine cycles available, and this computational power could be used for such tasks. BigBatch [Lins, 2006] can process monochromatic document images in batches, and may make use of all the computers available to document image filtering, either by employing a cluster configuration (mostly restricted to single Local-Area Networks) or a grid configuration distributed over many LANs or WANs. The grid configuration of BigBatch allows the use of the spare processing cycles of machines in an organization for document filtering and indexing.

This paper presents BigBatch, the problems it solves, and how it can be used on clusters and grids. Once an organization decides to employ a distributed environment with many computers to the task of document processing with BigBatch, however, it must select either a cluster or grid configuration for this. This paper is thus concerned with comparing both configurations in quantitative and qualitative aspects, to provide guidelines for systems administrators to take such a decision. These results and considerations were obtained in the context of BigBatch, but can be generalized to any application that processes a batch of documents in distributed systems. In previously published work [Mattos, 2008], the BigBatch tool was presented, along with preliminary results for comparing cluster and grid configurations; the new results are reported in this paper that widens and deepens the previous results obtained, besides making use of more modern machine architectures including multiprocessor ones.

The paper is organized as follows: Section 2 describes the problems that occur in raw digitalized images of documents and that hamper their efficency as digital documents. Next, Section 3 presents BigBatch, a software platform designed to solve those problems for a large numbers of documents, taking advantage of all computational power that an organization can make available for this task, and thus including the possibility of working with clusters and grids of workstations. Then, Section 4 describes a series of experiments designed and executed to assess the performance of cluster and grid configurations. The results are presented and analyzed in Section 5, in which are also considered qualitative aspects of comparison

between both distributed configurations. Finally, Section 6 concludes the paper and presents directions for further investigation.

2 Digitalized Document Images

The direct scanning of a document does not usually produce a useful digital document. The use of a scanner, either manually-fed flatbed or a production-line, automatically-fed one, introduces artefacts to the raw digitalized image that are undesirable in digital documents. Three common problems are the presence of black borders, wrong orientation, document skew, and the presence of salt-and-pepper noise.

Depending on a number of factors such as the size of the document, its state of conservation and physical integrity and the presence or absence of dust in the document and scanner parts, very frequently the image generated is framed either by a solid or stripped black border (Figures 1-4 present some of the most typical kinds of noise border). This undesirable artefact, also known as *marginal noise*, not only drops the quality of the resulting image for CRT visualization, but also consumes space for storage and large amounts of toner for printing. Removing such frame manually is not practical, for it is a time-consuming operation that requires specialized users. Several production-line scanner manufacturers have developed software tools for removing such noisy borders. However, many of these programs [ClearImage, 2003] [Leadtools, 2001] [ScanFix, 2001] [Skyline, 2003] are too greedy and tend to remove essential parts of documents.

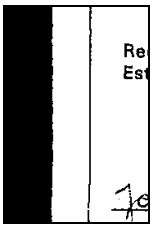

Figure 1: Solid black noisy frame

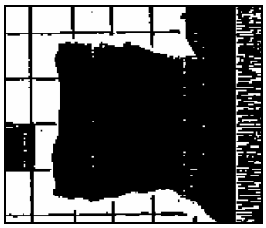

Figure 2: Irregular shape noise border

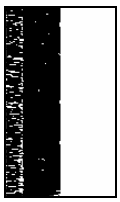

Figure 3: Pattern with stripes

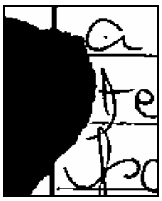

Figure 4: Information linked to noise border

Documents are not always correctly placed on the flatbed scanner, either manually by operators or by the automatic feeding device. This problem yields either incorrectly oriented or rotated images. For humans, badly oriented and rotated images are difficult for visualization and reading. For machine processing, a number of problems arise that range from needing extra space for storage to making more error-prone the recognition and transcription of the image by automatic OCR tools. Thus, orientation and skew correction are present in any environment for document processing. However, the precision of orientation and rotation angle detection, the quality of skew-correction, and the time required for processing in those operations vary widely from one tool to another. Three problems often appear in the rotation of monochromatic images: white holes appear within flat black areas, smooth edges become uneven and full of ripples, and neighboring areas become disconnected. Very often the result of rotating a monochromatic image shows degradation effects such as the ones presented on Figure 5.

The digitalized image may also include some noise in areas that were originally homogeneous, or near contours, especially the kind of noise known as salt-and-pepper. Removal of this kind of noise can often improve document quality and facilitate further document processing, like character recognition using OCR tools.

At a minimum, a tool for processing images of digitalized documents must include filters for at least these three problems. The next section describes BigBatch, a document processing tool that includes these filters.

Figure 5: Word rotated by 45º and -45º by classical algorithm

3 The BigBatch Solution

BigBatch was designed to automatically process thousands of monochromatic images of documents generated by production line scanners. BigBatch opens a batch of documents and, for each one, removes its noisy borders, checks and corrects its orientation, calculates and compensates the skew angle, crops the image standardizing document dimensions, removes salt-and-pepper noise, and finally compresses it according to a user-defined file format. BigBatch includes some of the best recent algorithms for monochromatic document images [Ávila, 2004a] [Ávila, 2004b] [Ávila, 2005a] [Ávila, 2005b] [Lins, 2004]. Figures 6 to 9 present an example on the document processing capabilities of BigBatch.

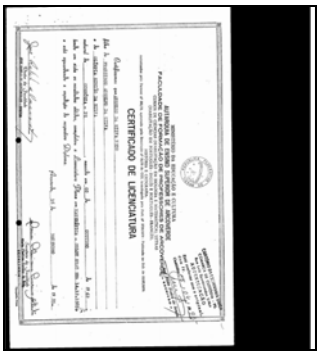

Figure 6: Original Scanned Document Image

Figure 7: Noise Border Removed

Figure 8: Orientation and skew corrected image

Figure 9: Cropped and filtered image

BigBatch may work either in standalone or operator-assisted modes. The operator-assisted mode allows a user to apply each filter and visualize its results to improve a given image. In standalone mode a full batch of documents, specified by the user, is processed, in one of three configurations: sequential (using a single computer), cluster or grid. In cluster or grid mode, BigBatch automatically dispatches document processing tasks from a server to workstations available to the job, collecting the results from each of them afterwards.

The image processing algorithms in BigBatch were implemented in C. The graphical user interface, and support for cluster and grid configurations, were developed using Scala [Odersky, 2005], a language that runs on the Java platform. To test the program, a set of test images were generated using Kodak 0 production line scanners, with 200dpi resolution and compressed in TIFF(G4) file format. The quality of the resulting images after processing with BigBatch was at least as good as the best ones produced by the other commercial tools tested 00000. Sequential processing in BigBatch outperformed all the other tools whenever images of comparable quality were generated (many times some of the other tools demanded less time than BigBatch for border removal, for instance, but their results were unsatisfactory). A previous version of BigBatch, limited to sequential processing of documents in a single node, is described in a paper by Lins, Ávila and Formiga [Lins, 2006].

One innovative feature of BigBatch is the possibility of using cluster and grid configurations for processing large batches of documents. High-performance computing was once restricted to institutions that could afford the significantly expensive supercomputers of the time. Nowadays, more than 70% of the top 500 computing systems in the world are clusters [Top500, 2007]. Besides the adoption of clusters, computational grids – mostly composed of PC workstations – have become prominent in recent years. Clusters are associated with local-area networks (LANs), and are composed of dedicated nodes. On the other hand, grids are traditionally associated with wide-area networks (WAN) and internets, and can include nodes in different domains; the promise of grids is to integrate computational resources that are available across an internet and to bridge organizational barriers for the execution of tasks. Clusters tend to be used in applications where there is communication between nodes, whereas grids are better suited to applications where the work units are independent and there is little inter-node communication.

With BigBatch giving the choice of using either a cluster or grid configuration to take advantage of a number of computers made available for document processing tasks, it is up to the user to select one of them to execute these tasks. For this, the user must know the advantages and disadvantages of each configuration, in both quantitative and qualitative aspects. The primary aspect for comparison is performance: whichever configuration achieves greater throughput is often preferable, as the processing of large amounts of document images may take a long time. But there are also other aspects to consider: how easy it is to setup each configuration, if the computer must be dedicated to the document processing task, what are the requirements to run a cluster or grid configuration, amongst others.

The questions about performance are not clear at first. For instance, it is frequently assumed that the flexibility brought by grids, allowing the use of computers over the Internet or in different organizational domains, would degrade the performance for tasks executed locally, in an environment wholly controlled by a single organization. To determine if this is indeed the case, and if so, what are the performance penalties to pay when using a grid configuration on a dedicated LAN, a set of experiments were designed for comparing the execution of BigBatch tasks over a cluster and a grid. The cluster and grid configurations both included the same number of machines, and executed the same tasks over the same test images. The document processing tasks generated by BigBatch are ideal for distribution over grids and clusters, because they are easily partitioned: each computer can be assigned to process a subset of images from the complete batch, and return the results.

The next subsections describe the configurations, and Section 4 describes the experiments performed. Section 5 presents the results of the performance comparison and considers other aspects, mostly qualitative, about the choice of configurations.

3.1 The Cluster Configurations

There is a wide variety of cluster software libraries and middleware programs that can be used to help managing tasks in a cluster, e.g. openMosix [openMosix, 2002], Condor [Litzkow, 1998] and Microsoft Cluster Server [Microsoft, 2003]. It is more common that applications must be explicitly written with the cluster in mind, incorporating the division of tasks between nodes and the communication between

them. The programming of cluster tasks often uses specialized libraries such as MPI [Snir, 1998] and OpenMP [Chandra, 2000].

Initially, the support for distributing BigBatch tasks to nodes in a cluster configuration was custom written for this application, using the Scala programming language [Odersky, 2005]. Scala is a functional and object-oriented language that was designed to run in the Java Virtual Machine and interoperate with Java libraries and APIs. It was chosen because it was desirable to work with the Java platform, leveraging its portability and the availability of libraries; further, Scala includes good support for distributed programming using Actors [Haller, 2006]. Another reason for having the BigBatch application running over the Java platform is to ease integration with the grid component, as will be explained in the next subsection.

Later, another cluster configuration was prepared, to take advantage of a cluster running MPI [Snir, 1998]. The distribution of tasks was written as a C program that used MPI calls to coordinate between nodes. Thus it is possible to compare cluster software written using MPI with a custom cluster program written in Scala.

Nodes in the image processing application are divided into *worker* nodes and a single *master* node which coordinates the distribution of tasks between the workers. The computer where the main BigBatch application is executed is the master node, while worker nodes must execute a smaller component called the BigBatch Client Module. Communication between the nodes is done by message-passing, always from the master to the workers or from one worker to the master, never between workers. The master dynamically balances the load by distributing tasks to the available worker nodes, maintaining a list of tasks that need to be executed and available worker nodes. Whenever there are pending tasks and workers are available, the master assigns tasks to the workers in some arbitrary order (as nodes are homogeneous, it makes no sense to select one over another for a given task). A worker node that receives a task is marked as busy, and it stays in this state until the task is completed and a message is sent to the master to signal that; the master then marks the node as available again, adding it to the list of available nodes. This process continues until there are no more tasks to be executed. Load-balancing is thus very simple, due to the homogeneous nature of the cluster architecture and the data-parallelism nature of the problem.

3.2 The Grid Configuration

Computational grids are also formed from a collection of general-use computers that are coordinated to the execution of related tasks. The main difference between a grid and a cluster is that the latter tends to be established using dedicated resources that are local to a single organization, while the former may include non-dedicated, non-local computers as nodes. It is common for grid software to take over a workstation computer (that would normally be available to human users) to execute tasks while it is idle. Therefore, grids are a distributed computing environment that features lower coupling than what is expected of clusters.

The low-coupling between nodes and the distributed nature of processing makes the programming of applications over grids more complex and challenging than is the case with clusters. A special case of problems that can be solved with grid platforms are the ones whose sub problems are independent and need no communication between the nodes themselves. This class of applications is commonly called *bag-of-*

tasks applications, and their execution is simpler to manage in grids. Taking advantage of that, a number of software systems have been developed to support the execution of this kind of tasks on computational grids, the so-called grid *middleware* systems. One such system is OurGrid, which was selected to be used in the support for grids provided by BigBatch. Document processing tasks generated by BigBatch fulfill the bag-of-tasks requirements.

OurGrid [Cirne, 2006] is both an open-source grid middleware and a grid infrastructure where sites may make available their computational resources from idle computers; in exchange, a participating site can obtain access to the computational resources from other sites, whenever necessary. To organize the exchange of computational favors, OurGrid establishes a peer-to-peer network between interested sites, in which the "currency of exchange" is computational time. This is done to assure that participation in the grid and allocation of resources is fair, and the peer-to-peer network formed is called a "network of favors" [Andrade, 2004]. Participation in the network of favors is optional and an organization may use the OurGrid middleware only internally, as is the case reported in this paper.

In the OurGrid solution there are three main components: **MyGrid**, the **Peer**, and the **UserAgent**. The MyGrid component is responsible for the management and scheduling of grid tasks – organized in collections called *jobs*. The Peer manages nodes in a site and the exchange of computational resources with other sites. Finally, the UserAgent is a small program that must be installed in each node that will be part of the grid. The grid needs a node executing the MyGrid component and a node executing the Peer component (these two may execute in a single node), in addition to the UserAgent executing in each worker node.

MyGrid is further subdivided into two modules: the scheduler and the replica executor. The scheduler is responsible for receiving new tasks from users and managing them, allocating nodes for their execution; it creates replicas of the tasks (if necessary) and communicates with the Peer requesting nodes for execution of the replicas. The nodes returned by the Peer may be local, or may be obtained from remote sites through the network of favors. The replica executor manages the execution of replicas of tasks and the sending of task results to the scheduler.

Currently, MyGrid works with two scheduling strategies: Workqueue with Replication [Paranhos, 2003] and Storage Affinity [Santos-Neto, 2005]. The first was designed for CPU-intensive applications, while the latter was created to improve the performance of applications that process large data sets.

A collection of tasks related to the same problem is called a job in OurGrid. A job is composed of independent tasks, each one composed of three phases: init, remote and final. These phases are executed on sequence, with the init and final phases being mostly used to transfer files needed for execution of the task; they are thus executed on the MyGrid node. The remote phase is executed in one or more worker nodes (depending on the replication strategy), and comprises the computation needed by the job. While executing a job, MyGrid requests nodes from the Peer to assign tasks to them.

Execution of the job is managed by the MyGrid component, which schedules tasks between the nodes made available by the Peer following the chosen scheduling method. This proceeds until all tasks have been executed. In the case of BigBatch, the BigBatch application creates the job, based on the batch of document images that

must be processed, and communicates this job to the MyGrid component. As both this component and the main BigBatch application run over the Java platform, this communication is easily performed using the Java Remote Method Invocation (RMI) mechanism.

4 The Experiments

A number of experiments were devised and executed to assess comparatively the performance of cluster and grid configurations. Besides the experiments and results presented in [Mattos, 2008], further configurations were tested. For completeness, this section describes all the experiments, including the new ones. In total, three different physical configurations were used, while three different logical configurations were laid over the physical structure in different combinations.

The first physical configuration used to run the experiments with BigBatch was composed by eight 3.2GHz Pentium IV computers with HyperThreading technology and clock at 3.2GHz, 512Mb of RAM, connected in a local-area network by a standard Ethernet connection and a 100Mb/s Ethernet switch; henceforth, this configuration will be called **HT1**. The second physical configuration was composed of similar machines, with eight Pentium IV computers with HyperThreading technology, clocked at 3.2GHz and with 1Gb of RAM; the only difference to the first configuration is the amount of memory. This configuration will be called **HT2** in the rest of the paper. The third configuration employed four 2.66GHz Intel Core 2 Duo computers having 2Gb of RAM, and will be called **CoreDuo**. The logical configurations, and the names they will be referred to in the rest of the paper, were as follows:

- **Grid**: grid configuration using OurGrid. The operating system was Ubuntu Linux 6.06 [Ubuntu, 2006] with a standard desktop installation.
- **Cluster-Scala**: custom cluster configuration using software written in Scala version 2.4.0 [Odersky, 2005]. The operating system was Ubuntu Linux 6.06 [Ubuntu, 2006] with a standard desktop installation.
- **Cluster-MPI**: cluster configuration using software written in C with MPI [Snir, 1998]. The operating system was Microsoft HPC Server 2008.

Table 1 shows the combinations of physical and logical configurations that were used in experiments.

	Grid	Cluster-Scala	Cluster-MPI
HT1			
HT2			
CoreDuo			

Table 1: Combinations of physical and logical configurations used in experiments. Shaded cells represent combinations that were used.

In the **Grid** configuration, the MyGrid and Peer components where executed in the same node, designated as the master node. All other nodes, the workers, executed a copy of the UserAgent component. The image processing tasks were specified to OurGrid by the BigBatch application, including the transfer of files necessary to each task. For the cluster configuration, there were two programs that coordinated the work between nodes, the master program and the worker program. The master program is the main BigBatch application, and was executed in the master node, and each worker node executed a copy of the BigBatch Client Module.

Each image processed is relatively small, in the order of 100Kb. The filtering of each image is independent of the others, which confirms that the application is naturally bag-of-tasks, and suggests a partition of the problem. To minimize network and scheduling overheads, it was decided to assign to each task the processing of an image package, composed of a number N of images each, instead of one task per image. For example, with a hundred images per package and a total of 21,200 images, 212 packages were generated, totaling 212 independent tasks that should be run in cluster and grid configurations. The number of images N was initially fixed as a hundred per package and later varied in subsequent experiments, as explained later.

As both OurGrid and the BigBatch application run on the Java platform, the Java Virtual Machine, version 1.6.0, was used in the nodes for all configurations. The BigBatch application used Scala version 2.4.0 (compiler and libraries). In the **Grid** configuration, OurGrid version 3.3 was used, selecting Storage Affinity as the scheduling algorithm, as the application is clearly data-intensive. The image packages were initially stored in a single node, the master node, so it was necessary to transmit the package file to a worker node prior to processing the task assigned to it. After processing, the worker had to transmit the resulting package of filtered images to the master. In the **Grid** configuration, this was performed in the init and final phases of tasks, while in the **Cluster-Scala** configuration the necessary file transfers over the network were designed into the BigBatch application. For the **Cluster-MPI** configuration, the files were kept in a shared network directory.

Using this basic setup, tasks were executed under different conditions, where the objective was to observe what changed when some variables were changed independently. For this first group of variations, the tasks were executed with N (the number of images per package) equal to one hundred. The first condition observed was the allocation of tasks to the master node: in one configuration, a node was used only as a master node, never a worker; in another, the master node was also a worker node. In cluster configuration, the master node executed the part of the application responsible for scheduling and storage of image packages; in the grid, the master node executed the MyGrid and Peer components, as already discussed. These two possibilities of use of the master node were exercised on both configurations.

The **Grid** configuration included another varying condition: as a computational grid may include non-dedicated nodes, an external computational load was simulated by continuously playing a DVD in the worker nodes, while they were also executing image processing tasks. This was done to assess how an additional computation load, not related to the grid tasks, would affect the performance of a worker node.

As a further experiment, the number N of images per package was varied to assess the impact of the size of tasks in the performance. Complete sets of tasks were generated to process all 21,200 images in packages of N equal to 25, 50, 500 and

1,000 images. These tasks were executed, with the master node being allocated also as a worker node, and their times were noted. The case for N=100 was already measured in previous experiments. This was done to measure how differences in load balancing would affect performance.

In the configurations that used dual-core computers (**Cluster-Scala** over **CoreDuo** and **Cluster-MPI** over **CoreDuo**), the experiments were also executed and measured with the scheduling of two tasks per node, to take advantage of the two cores. This was also tried for the **Cluster-Scala** configuration over computers with Simultaneous Multi-Threading (configuration **HT1**).

In each observed condition, the number of nodes used to compose the cluster or grid was varied, to determine how the application scales up in relation to the number of nodes made available. The number of nodes in the experiments reported here varied from one to eight.

5 Comparison between Cluster and Grid Configurations

In this Section the results from the experiment described in Section 4 are presented to establish a performance comparison between cluster and grid configurations. Later on some qualitative aspects of the comparison between the two configurations are seen.

5.1 Performance: Grid x Cluster-Scala

Table 2 shows the total execution times for the tasks in **Grid** and **Cluster-Scala** configurations, in the conditions detailed in Section 4, with a hundred images per package (N = 100). The physical configuration was HT1.

	Grid			**Cluster-Scala**	
# Computers	Time with non-dedicated master	Time with dedicated master	Time with non-dedicated master and DVD	Time with non-dedicated master	Time with dedicated master
1	10:27	13:24	14:25	11:21	13:42
2	05:20	06:18	06:30	06:19	06:42
3	03:53	03:58	04:43	04:09	04:11
4	03:00	03:07	03:20	03:41	03:46
5	02:26	02:34	02:53	02:55	02:57
6	02:08	02:15	02:22	02:27	02:30
7	01:45	01:59	02:05	01:53	01:54
8	01:24	---	01:53	01:32	---

Table 2: Total execution time (hh:mm) of tasks (N=100) in HT1 configuration.

For the purpose of comparing total execution times between **Grid** and **Cluster-Scala**, Figure 10 shows graphically the results for both configurations when executing tasks with a dedicated master node.

It is clear that there is little difference in total time between configurations, with times for the cluster still a little higher than for the grid. This was not initially

expected, as the cluster software was custom developed for this image processing application. This aspect of the results may be explained by some inefficiency in the cluster application that went undetected during the experiments. However, in both configurations the load is balanced dynamically, the cluster software being in advantage only that its scheduling algorithm may be far simpler and achieving similar results. For this reason, it is to be expected that total times for both cluster and grid be similar, even if the cluster software is improved, as both configurations do essentially the same work. These results indicate that there are no losses when executing bag-of-tasks applications using a grid platform instead of a cluster.

Figure 11 shows a graph similar to Figure 10, but this time without a dedicated master node, that is, the master node also was a worker node and executed image processing tasks. Once again, results for configurations **Grid** and **Cluster-Scala** are very similar.

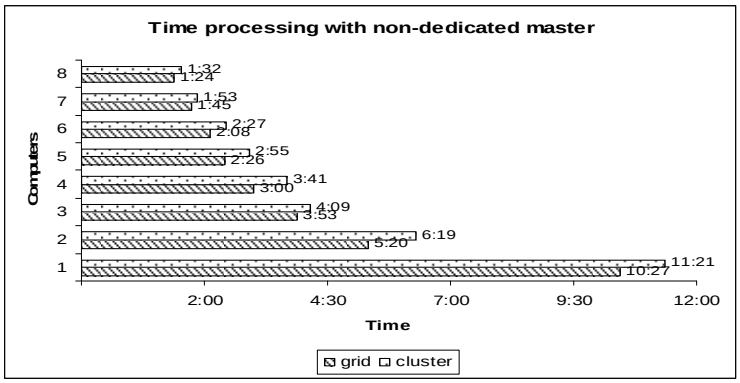

Figure 10: Execution time of tasks in cluster and grid with master processing tasks (N=100), using configuration HT1 (512Mb RAM).

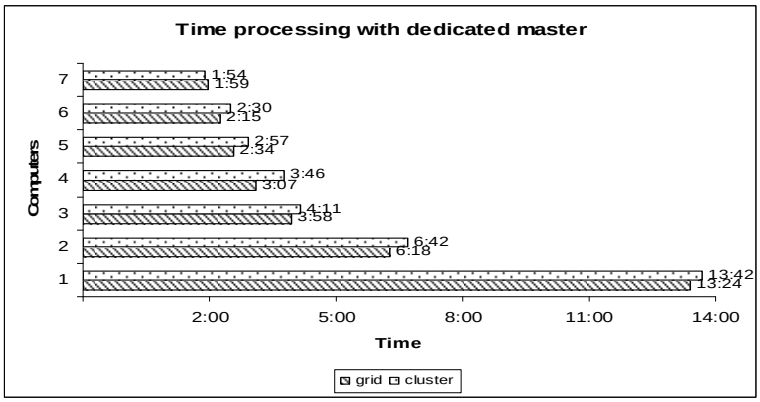

Figure 11: Execution time of tasks in cluster and grid configurations without master processing tasks (N = 100), using configuration HT1 (512Mb RAM).

To observe the effects of varying the conditions established in Section 4, Table 2 shows the total execution times separately for **Grid** and **Cluster-Scala**, respectively, for the case of one hundred images per package. It is clear that dedicating a node to act only as the master, or not, imposes very little difference on total execution times when many worker nodes are present; observed variations are only significant for a small number of worker nodes.

The effect of adding an external computational load in the grid configuration, simulated by playing a video, has similar impact, affecting the total time significantly only for a small number of workers. In a scenario of grid use by an organization, it is expected that the number of local nodes that can be allocated to running tasks is greater than seven or eight, which indicates that external loads can have limited impact to the grid performance. However, it is important to observe that the continuous playing of a video does not realistically simulate external computational loads in nodes that are used both as worker nodes in the grid and user workstations; in a real situation, a user would impose an external load which varies widely in time, possibly executing very demanding applications, sometimes. Another observation is that the computers used included multithreaded processors, which manage the execution of simultaneous tasks better than traditional single-threaded processors. However, as the current trends in computer architecture indicate a growing use of single-chip multiprocessors, which handle simultaneous tasks even better, it is expected that such processors may make the use of grids even more advantageous for an organization, because not only the grid tasks will be less affected by loads imposed by users, but also the user experience will be less affected by the fact that his computer is executing grid tasks at the same time.

Another observation regarding performance is to take under consideration the network protocols. Both configurations used message-passing over a TCP/IP stack, while in cluster applications it is usual to use MPI implementations that incur in lower network overheads. This was considered for further experiments that are detailed in Section 5.2. Another envisaged way to make the cluster more efficient is to perform load-balancing statically and a priori, taking advantage of the fact that the network configuration available to the cluster is known in advance.

Tables 3 and 4 show the results of varying the number of images per package (and thus the size of tasks) when processing all the images, for physical configuration **HT1**. Table 3 shows the results for **Grid** and Table 4 for **Cluster-Scala**.

# Computers	N=25	N=50	N=100	N=500	N=1000
1	10:36	10:34	10:27	10:30	10:30
2	05:30	05:38	05:20	05:28	05:36
3	03:38	04:38	03:53	03:36	03:58
4	02:50	02:57	03:00	02:50	02:56
5	02:14	02:40	02:26	02:21	02:21
6	01:54	01:57	02:08	02:02	02:08
7	01:36	1:38	01:45	01:49	01:56
8	01:25	01:38	01:24	01:30	01:32

Table 3: Execution time of tasks (hh:mm) in grid with differing package sizes (configuration: Grid + HT1).

In both cases, the results clearly show that there is little variation in total times. This is to be expected, as total processing times for each task, for N=100, are more than two orders of magnitude greater than the time taken to transmit the image package over the network. As such, the distribution of the load is not greatly affected by the sizes of tasks.

# Computers	N=25	N=50	N=100	N=500	N=1000
1	11:22	11:23	11:21	11:25	11:27
2	06:24	06:23	06:19	06:20	06:21
3	04:10	04:11	04:09	04:09	04:10
4	03:42	03:42	03:41	03:43	03:42
5	02:57	02:56	02:55	02:58	02:59
6	02:31	02:30	02:27	02:28	02:29
7	01:55	01:54	01:53	01:56	02:01
8	01:35	01:38	01:32	01:33	01:36

Table 4: Execution time of tasks (hh:mm) in cluster with differing package sizes (Cluster-Scala + HT1).

Table 5 and Table 6 show the results for the same experiment of varying package sizes, but for the HT2 physical configuration. The **Cluster-Scala** configuration, in this case, scheduled two tasks per node, to take advantage of Hyperthreading technology.

The results are quite similar to those obtained previously for the **Grid** configuration. The load balancing is not affected by differing package sizes. For the **Cluster-Scala** configuration, the results indicate a reduction of about 25% in processing time when scheduling two tasks per node. This gain must be attributed to the Simultaneous Multi-Threading present in the computers. In **Grid** configuration such scheduling is not possible.

# Computers	N=25	N=50	N=100	N=500	N=1000
1	10:46	10:42	10:36	10:33	10:32
2	05:26	05:25	05:19	05:20	05:38
3	03:38	03:38	03:35	03:36	03:44
4	02:44	02:43	02:41	02:47	02:54
5	02:11	02:10	02:10	02:13	02:18
6	01:49	01:49	01:49	01:56	02:07
7	01:34	01:34	01:34	01:44	01:59
8	01:22	01:22	01:22	01:21	01:31

Table 5: Execution time of tasks (hh:mm) in grid with differing package sizes (configuration: Grid + HT2).

All the experiments presented earlier concern the comparison between **Grid** and **Cluster-Scala** configurations, and is similar to what was presented in earlier work [Mattos, 2008]. The next results permit a comparison between the three logical configurations considered. Tables 7-9 show the results of running the image

processing tasks, with varying package sizes in **Grid**, **Cluster-Scala** and **Cluster-MPI** configurations, respectively. Here the physical configuration used was **CoreDuo**; in both cluster configurations, two tasks were scheduled per node, to take advantage of the two processing cores available.

# Computers	N=25	N=50	N=100	N=500	N=1000
1	8:37	8:35	8:32	8:33	8:34
2	4:18	4:18	4:18	4:18	4:20
3	3:29	3:27	3:28	3:30	3:30
4	2:13	2:12	2:10	2:10	2:11
5	1:56	1:54	1:52	1:52	1:53
6	1:30	1:30	1:31	1:32	1:33
7	1:14	1:13	1:13	1:14	1:14
8	1:05	1:04	1:04	1:05	1:06

Table 6: Execution time of tasks (hh:mm) in cluster with differing package sizes (Cluster-Scala + HT2).

# Computers	N=25	N=50	N=100	N=500	N=1000
1	10:35	10:31	10:25	10:24	10:24
2	05:24	05:22	05:18	05:20	05:30
3	03:38	03:38	03:35	03:37	03:40
4	02:50	02:44	02:42	02:49	02:55

Table 7: Execution time of tasks (hh:mm) in grid with differing package sizes (configuration: Grid + CoreDuo).

# Computers	N=25	N=50	N=100	N=500	N=1000
1	06:53	06:49	06:47	06:46	06:46
2	03:39	03:35	03:33	03:33	03:35
3	02:17	02:16	02:17	02:17	02:19
4	01:38	01:40	01:42	01:44	01:47

Table 8: Execution time of tasks (hh:mm) in cluster with differing package sizes (Cluster-Scala + CoreDuo).

# Computers	N=25	N=50	N=100	N=500	N=1000
1	4:13	4:12	4:11	4:13	4:13
2	1:51	1:51	1:52	1:53	1:53
3	1:18	1:16	1:18	1:19	1:17
4	1:03	1:03	1:04	0:59	0:58

Table 9: Execution time of tasks (hh:mm) in cluster with differing package sizes (Cluster-MPI + CoreDuo).

The results in both cluster configurations indicate an improvement of 35% on average when compared to the **Grid** configuration because the cluster configurations are able to use the dual-core processors by scheduling two tasks per node, thus

reaching better times. Furthermore, the use of MPI presented a significant performance improvement over the custom cluster architecture, in some cases reaching a reduction of almost 40% in running times. A comparison between the results for **Cluster-MPI** and **Grid** show a total reduction in running time of more than 50%. These results indicate further that the custom cluster software may have sources of inefficiency that went undiscovered, and that it may be advantageous to use cluster configurations when running tasks over computers with dual-core processors, or processors with more cores. However, it's likely that the OurGrid middleware will be improved in future versions to allow the scheduling of more than one task per node, which will allow it to reap the benefits of multicore processors. A lesson to be learned here is that the cluster configurations allow for more control over task scheduling, in relation to grid configurations, in which the grid middleware decides over scheduling autonomously.

5.2 Qualitative Aspects

Although performance is probably the main point of comparison for users, there are other aspects which deserve to be considered, and these are often not measurable. This subsection considers some of those aspects.

Ease of use is an important aspect, which includes ease of installation. The main BigBatch application is used in both configurations, so the user sees the same interface. However, additional software must be installed in worker nodes, and for the grid it is necessary to install the OurGrid components. Further, as both the BigBatch application and OurGrid are based on the Java platform, a Java runtime must be installed in all computers involved in the execution of tasks. In summary, for the cluster configuration, it is necessary to install the Java runtime in all nodes, the BigBatch main application in the master node, and the BigBatch Client Module in worker nodes; for the grid, the setup includes the Java runtime in each node, the BigBatch main application in the master node, along with the MyGrid and Peer components of OurGrid, and the OurGrid UserAgent component in each worker node. The grid setup is thus a little more complicated, but not significantly more so. The cluster configuration using MPI requires computers with the MPI libraries installed, and access to a shared network directory. More relevant is the fact that using the grid imposes an additional software dependency in comparison with the cluster, because the grid depends on the OurGrid middleware. This may raise maintenance costs in the long term.

Another aspect is the possibility of using the worker nodes for other tasks concurrently with the document processing, like making the computer available for users. In the grid case this is easily supported; for the cluster configuration, it is usual to use nodes as dedicated cluster computers, but the BigBatch Client Module does not prevent the execution of other applications in the same computer. The two configurations are very similar in this respect.

With regards to expansibility to a greater number of computers, the grid configuration is clearly superior, as grids were designed with this in mind. A user may easily include worker nodes from different domains in the same organization, or even take advantage of the distributed OurGrid infrastructure and get nodes from other domains over the Internet. Of course, this entails, by the network of favors, making some nodes internal to the organization available to external users of the grid, but this

can usually be arranged, at least for a limited number of nodes. However, the distribution of document processing tasks over external nodes may be counter-productive, as each task includes the transmission of a large package of images over the network; the bandwidth over a WAN may make the transmission times greater than the processing times, offsetting general performance. This is a topic for further investigation.

The grid is also more flexible, making it possible, for example, to make computers execute document processing tasks only when idle and not being used by a person – this can be detected by the execution of a screen saver, for instance. Although this can be programmed into the BigBatch Client Module, it is one more feature that is already present on the grid. This feature, together with the ease of including further nodes in the grid, makes one to expect that it is far easier for grid configurations to muster a greater number of worker nodes than in the case of clusters. However, it is important to highlight the fact that sending documents for processing in nodes outside the organization could be a privacy problem. This should be seriously considered if the organization wishes to use external grid nodes.

As shown in the results when considering dual-core computers, the cluster configurations are easier to change to take advantage of scheduling conditions not designed into the grid software. The system administrator has more control over the scheduling of tasks in cluster configurations, and this is an advantage when new technology is being adopted.

To summarize the comparison, there are no significant performance differences between cluster and grid for the BigBatch tasks when running on single-core computers, but there is still a significant difference if multi-core computers are available; furthermore, using MPI for the cluster currently presents the best performance for BigBatch, and MPI itself is a mature and efficient technology, already a de facto standard in the High-Performance Computing community. The cluster is slightly easier to setup, and has fewer software dependencies; however, the grid is more flexible and expandable.

6 Conclusion and Lines for Further Work

This paper analyses the parallel processing capabilities of BigBatch, a software tool designed to process batches of thousands of monochromatic images from digitalized documents. BigBatch can use the computers available to this task in different configurations: a grid or two different cluster configurations. The results presented herein generalize and update for more modern architectures expand the performance data reported in previous work [Mattos, 2008].

To compare the use of these distributed configurations, a benchmark was set and the results show no significant performance differences between a cluster and a grid of similar physical configuration, executing the same image processing application, except when multi-core processors are available. Other qualitative aspects were considered, allowing a user to better decide between a cluster or grid configuration for his tasks. These results were generated by executing the BigBatch application over the two configurations, but can be safely generalized for any application that applies the same processing to a batch of documents in distributed fashion. However, the work

reported here does not cover all useful possibilities of comparison, as there are other variations and interesting variables that can be observed using similar experiments.

One point where the comparison can be made more interesting is to observe that high-performance cluster applications usually do not perform dynamic load-balancing; this is made statically, by dividing the work between cluster nodes during development and before execution. This eliminates all the need for scheduling and file transmission in the cluster, which would certainly reduce the execution time by exploiting the *a priori* knowledge of the configuration that is characteristic of clusters. As the grid uses generic software that was designed to work over varying and heterogeneous physical configurations, it can not take advantage of a fixed configuration and must always balance the load dynamically.

Another interesting possibility would be to make the scheduler smarter, by using properties of the images to algorithmically group them into packages in an attempt to achieve better load-balancing. This once again entails the use of information specific to the application to improve performance. In the experiments reported here, images were arbitrarily packed in packages of 25, 50, 100, 500 and 1000 images each, with no other grouping criteria than scanning order. As a result, some packages are much bigger than others, and this may lead to a sub-optimal load-balancing. To investigate better ways of grouping images into packages, depending on their properties and characteristics, it would be necessary to create models which related these properties and characteristics of the images with the times spent processing them. A simple model would take into consideration the generation of equally-sized (in bytes, not images) packages, for instance.

For the grid, another possibility would be to simulate a more realistic external load, by randomly executing CPU-intensive tasks, for example. This would better predict how the grid behaves in situations were users may affect the execution of tasks. A simulation along these lines may be made more interesting by using recent single-chip multiprocessor (multicore) computers. Also, it would be interesting to investigate the use of external nodes, obtained from other grid sites on the OurGrid infrastructure. It is not clear if the increase in network latency and the decrease in bandwidth would impact on the feasibility and desirability of using external nodes from the Internet.

On the cluster side, another possibility would be to use ready-made cluster software, including packages that run on Linux/Unix and Windows environments.

Finally, the set of image filters applied by BigBatch is not necessarily fixed. Some users may want to incorporate other filters in all their document processing tasks, while others may need to use a specific filter in a specific set of images. BigBatch could be easily fitted to use different filters. Even more interesting would be to make this configurable: the user would specify which filters to run in each batch, and could even provide filters not included in BigBatch.

References

[BlackIce, 2003] BlackIce Document Imaging SDK 10. BlackIce Software Inc. http://www.blackice.com/.

[ClearImage, 2003] ClearImage 5. Inlite Res. Inc.http://www.inliteresearch.com.

[Kodak,1999] Kodak Digital Science Scanner 1500.
http://www.kodak.com/global/en/business/docimaging/1500002/

[Leadtools, 2001] Leadtools 13. Leadtools Inc. http://www.leadtools.com.

[Microsoft, 2003] Microsoft Cluster Server.
http://www.microsoft.com/windowsserver2003/enterprise/clustering.mspx

[openMosix, 2002] openMosix, http://openmosix.sourceforge.net/. Access on 05-23-2007.

[ScanFix, 2001] ScanFix Bitonal Image Optimizer 4.21. TMS Sequoia,
http://www.tmsinc.com.

[Skyline, 2003] Skyline Tools Corporate Suite 7. Skyline Tools Imaging.
http://www.skylinetools.com.

[Top500, 2007] TOP 500 Supercomputer Sites, "Top 500 list", http://www.top500.org/. Access on 06-04-2007.

[Ubuntu, 2006] Ubuntu Linux. http://www.ubuntu.com/

[Andrade, 2004] Andrade, N., Brasileiro, F., Cirne, W., and Mowbray, M. 2004. Discouraging Free-riding in a Peer-to-Peer Grid. Proceedings of the Thirteenth IEEE International Symposium on High-Performance Distributed Computing (HPDC13).

[Ávila, 2004a] Ávila, B.T. and Lins, R.D. 2004. A New Algorithm for Removing Noisy Borders from Monochromatic Documents, ACM-SAC'2004, pp 1219-1225, ACM Press, March.

[Ávila, 2004b] Ávila, B.T. and Lins, R.D. 2004. Efficient Removal of Noisy Borders from Monochromatic Documents, Proc. of ICIAR 2004, LNCS(3212):249-256, Springer-Verlag.

[Ávila, 2005a] Ávila, B.T. and Lins, R.D. 2005. A New and Fast Orientation and Skew Detection Algorithm for Monochromatic Document Images, ACM DocEng 2005.

[Ávila, 2005b] Ávila, B.T., Lins, R.D. and Augusto, L. 2005. A New Rotation Algorithm for Monochromatic Images. ACM DocEng 2005.

[Buyya, 1999] Buyya, R. (ed.). 1999. High Performance Cluster Computing: Architectures and Systems, Prentice Hall.

[Chandra, 2000] Chandra, R., Menon, R. *et al.* 2000. Parallel Programming in OpenMP, Morgan Kaufmann.

[Cirne, 2006] Cirne, W. et al. 2006. "Labs of the World, Unite!!!". Journal of Grid Computing, v. 4, n. 3, pp.225-246.

[Haller, 2006] Haller, P., and Odersky, M. 2006. Event-Based Programming without Inversion of Control. LNCS. 4228, pp. 4-22.

[Lins, 2005] Lins, R.D. and Alves, N.F. 2005. A New Technique for Assessing the Performance of OCRs. IADIS - International Conference on Computer Applications, 1:51-56 IADIS Press.

[Lins, 2004] Lins, R.D. and Ávila, B.T. 2004. A New Algorithm for Skew Detection in Images of Documents, Proc. of ICIAR 2004, LNCS(3212):234-240, Springer Verlag.

[Lins, 2006] Lins, R. D., Ávila, B. T., and Formiga, A. A. 2006. BigBatch: An Environment for Processing Monochromatic Documents. International Conference on Image Analysis and Recognition, LNCS 4142, pp. 886-896.

[Litzkow, 1998] Litzkow, M., Livny, M., and Mutka, M. 1998. Condor – a hunter of idle workstations. In 8th International Conference of Distributed Computing Systems.

[Mattos, 2008] Mattos, G. O., Formiga, A. A., Lins, R. D. and Martins, F. M. J. 2008, BigBatch: A Document Processing Platform for Clusters and Grids. 23rd ACM Symposium on Applied Computing, 2008, Fortaleza. Proceedings of the ACM-SAC 2008. New York : ACM Press, 2008.pp. 434-441.

[Odersky, 2005] Odersky, M. 2005. Scalable Component Abstractions. OOPSLA 2005: pp. 41-57.

[Paranhos, 2003] Paranhos, D., Cirne, W., and Brasileiro, F. 2003. Trading cycles for information: Using replication to schedule bag-of-tasks applicatoins on computational grids. Proceedings of the Euro-Par 2003: International Conference on Parallel and Distributed Computing, Lecture Notes in Computer Science, v. 2790, pp. 169-180.

[Santos-Neto, 2005] Santos-Neto, E., Cirne, W., Brasileiro, F., and Lima, A. 2005. Exploiting replication and data reuse to efficiently schedule data-intensive applications on grids. LNCS, v. 3277, pp. 210-232.

[Snir, 1998] Snir, M., and Gropp, W. 1998. MPI: The Complete Reference, 2nd. Ed., MIT Press.