Running Medical Image Analysis on GridFactory Desktop Grid

Frederik ORELLANA [a], Marko NIINIMAKI [b], Xin ZHOU [b], Peter ROSENDAHL [a], Henning MUELLER [b] and Anders WAANANEN [a]

[a] *Niels Bohr Institute, Copenhagen University, Denmark*
[b] *Medical Informatics, Geneva University Hospitals and University of Geneva, Switzerland*

Abstract.
At the Geneva University Hospitals work is in progress to establish a computing facility for medical image analysis, potentially using several hundreds of desktop computers. Typically, hospitals do not have a computer infrastructure dedicated to research, nor can the data leave the hospital network for the reasons of privacy. For this purpose, a novel batch system called GridFactory has been tested along- side with the well-known batch system Condor. GridFactory's main benefits, compared to other batch systems, lie in its virtualization support and firewall friendliness.

The tests involved running visual feature extraction from 50'000 anonymized medical images on a small local grid of 20 desktop computers. A comparisons with a Condor based batch system in the same computers is then presented. The performance of GridFactory is found satisfactory.

Keywords. Grid, cluster computing, image analysis

Introduction

Over 70 000 images are produced daily by the radiology department of the Geneva University Hospitals [7]. Proper analysis by researchers of such a flow of images is impossible with single desktop PC's. Thus, a batch computing system and programming tools for it are needed. A case used in this paper is content based information retrieval (CBIR) of medical images. In a CPU-intensive process called indexing, features are extracted from images. Based on these features, images can be compared with each other. A typical medical application of this would be to compare if a new image of tissue resembles earlier images of malignant tumors.

GridFactory is a system for aggregating computing resources on local and wide area networks in order to carry out large-scale batch computations.[1] It is currently unreleased and under development. The work described here was carried out with an early pre-release of the software. The GridFactory system consists of (1) a command line user interface, used to submit batch jobs to (2) a server running the Apache [16] httpd server with modules providing the interface for job management, and (3) a Java application

[1] Due to the current requirement that data produced in the hospital does not leave the hospital, we do not discuss inter-organizational computing clusters in this paper; for these, see e.g. [20].

called GridWorker, running on each worker node, that pulls jobs from the server, executes them and uploads results back to the server.

The GridWorker software was installed on a cluster of 20 standard hospital workstations running Windows XP in a hospital seminar room (see [19]). While there are many batch job systems or "Local Resource Management Systems" (LRMS's) available[2] GridFactory has some features that make it well suited for environments like the Geneva University Hospitals (HUG), namely:

- **dynamic pools**; GridWorkers simply periodically request a job. If a job is available for execution, the server checks the requirements of the job with the permissions and capabilities of the requesting GridWorker and (if successful) allocates the job to the GridWorker. If the GridWorker program is interrupted (or the computer running it crashes), the server re-allocates the job to another requester. This is useful since the worker nodes in a hospital environment can contain normal PC workstations that the users can turn off or re-boot occasionally.
- **software repository integration**; jobs can require one or several specific software packages to be available on the execution host. GridWorkers consult a software catalog that contains information on where to download and how to install software packages. If a matching software package is found for a given job, GridWorker downloads and installs it before the job is executed. This is useful in general, since it removes the burden of manual software package installation in the worker nodes.
- **virtual machine provisioning**; a given set of software requirements may only be available on a certain operating system. Jobs can also explicitly require an operating system (typically Linux) that differs from the one hosting GridWorker. In this case, GridWorker can download and boot a virtual machine image that matches the required operating system. The job is then executed inside the virtual machine. The virtualization layer of GridFactory allows any virtualization software to be used. Currently, VirtualBox [23] and QEMU [5] virtual machine images have been tested and are available in our catalog. This feature, naturally, allows one to execute Linux jobs in Windows workstations.
- **stateless, firewall-friendly HTTPS communications** both from the job submitter and job requester to the server. Submissions and requests proceed via standard two-way authenticated HTTPS communication - always initiated by the submitter or requester (not the server). This means GridWorkers can operate on a fire-walled environment (for as long as they can send requests to port 443) and behind network address translation (NAT) sub-networks. The virtual machines do not need dedicated IP addresses, either.
- **virtual organization integration**; all submitters and resources (including execution hosts running GridWorkers) are identified by an X.509 certificate[3]. A virtual organization is defined by a text file on a web server, containing a list of distinguished names of such certificates. On the GridFactory server, permissions can be assigned to such virtual organizations - or to individual distinguished names.

[2] Popular batch job systems that can utilize Windows XP resources include Condor [17] and Sun Grid Engine [22].

[3] For convenience, a default test certificate can be used - effectively rendering the system insecure, which in some situations may be acceptable on a closed network.

Servers and GridWorkers can choose only to support certain virtual organizations. A person submitting a job can choose to allow only members of certain virtual organizations to execute her jobs (and access the input files of the jobs).

In related work, batch job processing systems (also known as cluster computing systems or LRMS's) have been a subject of many studies, including [6,12,17]. Public resource computing, made famous by BOINC [4] uses resources that are maintained by volunteer participants. Virtualization, i.e. separating the job execution environment from the host computer has been promoted in [10]. Grid computing, often characterized as "a system that coordinates resources that are not subject to centralized control, using standard, open, general-purpose protocols and interfaces to deliver nontrivial qualities of service" [11] can combine geographically distributed resources by a Grid middleware.

"Campus grids" or "desktop grids" have been discussed in [15,14]. Practical desktop grid projects include Sztaki [3] and Enabling Desktop Grids for e-Science EDGeS [2]. Both these projects use BOINC software.

An exhaustive comparison of batch systems, public resource computing and virtualization support in desktop Grids would be impossible in the scope of this paper. However, we can state the features of each of these that are present in GridFactory, namely:

- job descriptions and data transfers in the style of a batch system,
- certificate enforced security in all communications,
- initial support to a Grid middleware, namely NorduGrid ARC [9], so that ARC can use a GridFactory cluster as a LRMS resource,
- job execution in a completely isolated virtual machine (that does not communicate through network connections).

The rest of this paper is organized as follows. In Section 1, we give a short presentation of the GridFactory software. In Section 2, we describe the feature extraction task run by GridFactory. Section 3 presents the measurements of job execution times and how they compare to those obtained using two other systems. Finally, Section 4 contains a summary and discussion.

1. GridFactory

The GridFactory system consists of three main components; a command line interface for job submission and manipulation, a server component with queue and spool daemons; and the GridWorker client software. A job definition file is in fact a "shell" executable file with extra tags that are only comments for shell, and interpreted by the components. They define inputs, outputs and requirements for the job.

1.1. Job Description

A compute job is described by a record in the database table. This record contains information on where to download input files and upload output files, the current status of the job, the requirements of the jobs etc. All fields of a job record are readable through a web service.

1.2. Job Submission

Job submission can be done with the provided command line utilities or using the Java API or simply standard HTTPS/WEBDAV directory creation and file uploading. Submitting a job consists in creating a new directory in the job spool directory on the server, served via HTTPS, and subsequently creating a file called “job” in this directory. The file called “job” is a shell script that contains directives in-lined as comments.

1.3. Job Preparation

The spool daemon on the server periodically scans the spool directory and when a new job is found, parses these directives in the “job” file and creates a corresponding record in the database. This daemon also checks that the requested software is available in the software catalog.

1.4. Job Execution

The software catalog allows dependencies among the software packages. These dependencies are resolved by the GridWorker. The resolver may detect that a package requires a different operating system than in the node hosting the GridWorker. In this case, or in the case that virtualization is requested by the job or the GridWorker configuration, the job is run inside a virtual machine. The GridWorker regularly scans the job database on the GridFactory server. When a new job is found, input files are downloaded from the server via HTTPS, the job is executed and output files are uploaded to the server via HTTPS. During such a cause of events, the GridWorker regularly updates the database with information on the job status.

1.5. Job Query and Manipulation

Once a job has been submitted, the only field of a job record that is modifiable by the submitter or the GridWorker is the status field. By modifying this field, the submitter can prompt the worker node daemon to kill, pause or resume the job or upload the current stdout and stderr of the job to the server - from where it can then be downloaded. Modifying this field also allows a GridWorker to request the job and report on its status.

1.6. Example

For our example (see next section), 50 GridFactory jobs were created by an external script. A job contains a job script with in-lined GridFactory directives, as shown below.

```
#GRIDFACTORY -r VM/Linux/Sarge
#GRIDFACTORY -i 1000images1.tar  python-local.py  sources.tar
#GRIDFACTORY -o features.tar.gz
python python-local.py 1000images1.tar
```

The “-r” indicates that the job requires a Linux Sarge execution environment. Input and output are indicated by “-i” and “-o”, respectively. The job is submitted with the command psub.

Technically, the command psub simply issues a “create new directory” command over HTTPS then upload the script and its inputs to this directory. A GridWorker then downloads the files and runs the job. When the job is finished, the GridWorker uploads the output file(s) back to the GridFactory server.

To keep track of and allow GridWorkers to pick up jobs, the GridWorker server uses a MySQL database. All interaction of the GridWorkers with this database proceeds over HTTPS via a specially written Apache module.

In this chain of events, compared to traditional batch systems, the unique features of GridFactory are: (1) Scheduling is done by the GridWorkers. (2) File transfers are done by the GridWorkers and no shared file system is needed. (3) GridWorkers initiate all network connections and need no open inbound ports. (4) All traffic is over HTTPS. (5) Software needed by the jobs is installed by GridWorkers on the fly. (6) In this case, the software in question is a virtual machine image. (7) Jobs are run inside a virtual machine.

2. Feature Extraction with GIFT

The Gnu Image Finding Tool (GIFT, see [1]) is a content-based image retrieval software package [21]. The software extracts features from images and matches them with stored features to find similar images [18]. Though feature extraction (indexing) typically takes only a few seconds per image with current hardware, indexing large image collections like the ImageCLEFMed reference database [8] can take hours (35 hours for a collection of 50’000 images is typical on a modern CPU). To speed up indexing, the feature extraction task can be parallelized in such a way that some of the tasks can be executed with a batch system.

As in previous reports ([20,19]), we have used a database of 50’000 medical images with a simple package generator program. Though the original images are in JPEG format and in varying sizes, they are normalized to 256x256 pixel PPM format, and packaged in sets of 1000 images by the generator. The size of the package is $\sim$400 MB.

The image sets, together with the feature extraction program source code are copied to the execution nodes. The execution on the nodes consists of unpacking the set of images, compiling the feature extraction program and performing feature extraction for each of the images with results stored in a local file. After all jobs have finished and their result files copied to the server, the results can be combined.

Producing a 1000 image set takes between 2 to 10 minutes, and the whole process of producing the 50 sets takes an average of 200 minutes on a modern computer. The extraction of features from a set typically takes 30 to 60 minutes.[4] In an ideal case, the computing nodes readily finish their earlier tasks so that the total execution time is “total image set generation time” plus “total feature extraction computation time” plus “result combination time”. A reliable cluster computing system with fast data transfers should come close to this.

[4] One can deduce from the figures that this task could be optimized by using smaller sets. The size of 1000 is for compatibility for our benchmarking. Smaller sets would increase the overhead, too.

20 CPU local VM cluster with Condor	240 min
20 CPU local VM cluster with GridFactory	270 min

Table 1. Comparison of running times of feature extraction on different clusters

3. Measurements and Results

Previously, we have tested and compared medical image indexing in local and remote Grid clusters [19]. Our sample collection consists of 50'000 medical images. The local cluster's LRMS was Condor running inside a VMware virtual machine ([24]). The local cluster performance was found better, mainly because of the large size of the input files.

Here, we compare the performance of the local cluster running GridFactory with that of the same cluster running Condor.

The local cluster consists of 20 "old" Compaq Evo 510 (Pentium 4, 2.8 GHz, 768 MB RAM) computers.[5] The computers were equipped with the standard hospital installations of Windows XP (with anti-virus software). The network connectivity of these computers was, likewise, the same as for any hospital desktop computer, i.e. they belong to the same sub-network as other desktops in the main hospital area, and get their IP addresses from the same address pool.

For both solutions (Condor and GridFactory), there were more than one computing nodes available for each of the submitted job. In fact, because of the slight over- capacity of the cluster, at least one node was idle at any given time. The results are shown in table 1. It should be noticed that in our previous comparisons, the same analysis in a remote cluster took 537 min, so both Condor and GridFactory in a local cluster can be considered efficient.

The reason for GridFactory/QEMU being slightly slower than Condor/VMware is related to two issues:

- VMware is faster than QEMU. The average job execution time with VMware was 42 minutes while it was 49 minutes for QEMU. In modern computers, the difference would be larger due to the fact that VMware utilizes the virtualization features of modern CPU's quite efficiently.
- GridWorker actually copies the input files twice: it first downloads input files from the server, then copies them into the virtual machine via the virtual network interface.[6] The longer preparation ("staging") time of jobs can be seen, too (Figure 1). The graph indicates that one of the computing nodes failed to finish a job but that it was automatically adopted by another node.

Though in our tests, the Condor/VWware solution performed slightly better than GridFactory, we consider that the benefits explained below compensate this disadvantage.

It should be noted that the fact that GridWorker can run on a host machine and control jobs running inside a virtual machine is a crucial security feature. It means that the job running in the virtual machine can be blocked from having network access, implying

[5] It should be noticed that the processor of these computers does not support virtualization extensions, thus making the virtual machines run significantly slower than they would on more recent hardware; for details about hardware level virtualization support see e.g. [13].

[6] The last step is in fact usually more time-consuming than the transfer from the server.

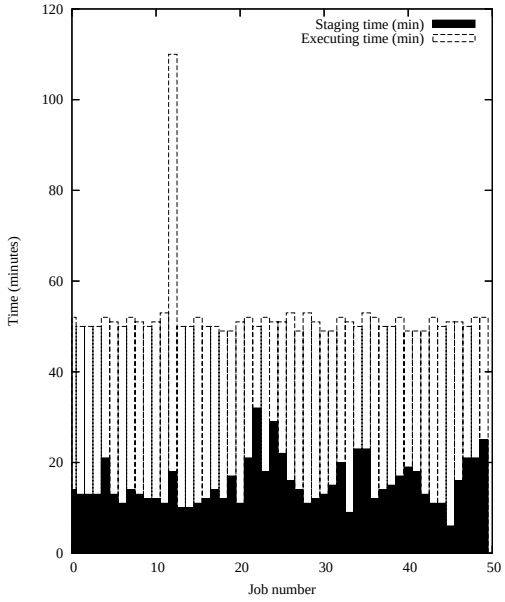

Figure 1. Job staging and execution times.

that a desktop user only has to trust the GridWorker software – *not* the compute job executing inside the virtual machine. GridWorker could perfectly well have been run like the Condor daemons, *inside* a virtual machine, but then the compute jobs themselves would run on a machine with network access; in particular with local area network access - something which is often not acceptable in a production setup in an environment with privacy and security requirements like those of a hospital. For Condor, this situation is aggravated by the fact that the Condor daemons running inside the virtual machines require inbound network ports to be open.

It should also be noted, too, that QEMU virtual machine management functionality (e.g. booting up a QEMU virtual machine disk image) is not part of GridFactory. GridFactory relies on such functionality being provided by software packages in the software catalog. The current software catalog contains QEMU and VirtualBox software packages. In fact the overhead of copying input files into the virtual machine could easily be eliminated by using the shared-folder capabilities of virtualization engines such as VMware and VirtualBox.

Finally it should be noted that the GridWorkers could in fact have been configured to run a job while at the same time downloading input files for the next job. With such a configuration, GridFactory can be expected to achieve total execution times close to the ideal scenario described above.

4. Summary and discussion

We have carried out preliminary performance tests of a new batch system, GridFactory, by using it to carry out feature extraction on a large number of medical images. The run-

ning times were compared to those of previous runs on another Local Resource Management System.

Despite the fact that GridFactory is not yet a finished or mature system, the results were encouraging:

Performance-wise GridFactory compares favorably to other systems tested previously. The tests ran slower on GridFactory than on Condor, but the reasons for this are well understood.

Feature-wise, GridFactory is already a potent system with an integrated software catalog, virtual machine provisioning, dynamic pools of worker nodes, integrated virtual organization support and firewall-friendliness. Although not explored in production yet, in particular the level of security achieved by the novel virtualization approach combined with the virtual organization support is import in a hospital setting, where it must be possible to strictly limit who and which machines can gain access to which files.

References

[1] Gift, GNU image finding tool. http://www.gnu.org/software/gift/, 2005.

[2] Edges. `http://www.edges-grid.eu/`, 2008.

[3] Sztaki desktop grid. `http://www.desktopgrid.hu/`, 2008.

[4] David P. Anderson. Boinc: A system for public-resource computing and storage. In *Fifth IEEE/ACM International Workshop on Grid Computing (GRID'04)*, pages 4–10, 2003.

[5] F. Bellard. Qemu, a fast and portable dynamic translator. In *Proc. Usenix 2005 Annual Technical Conference*, pages 41–46, 2005.

[6] R. Buyya. *High Performance Cluster Computing: Architectures and Systems*. Prentice-Hall, 1999.

[7] Adrien Depeursinge, Henning Muller, Asmaa Hidki, Pierre-Alexandre Poletti, Alexandra Platon, and Antoine Geissbuhler. ImageâĂŞbased diagnostic aid for interstitial lung disease with secondary data integration. In *SPIE Medical Imaging*, San Diego, CA, USA, February 2007.

[8] Thomas Deselaers, Henning Müller, Paul Clough, Hermann Ney, and Thomas M. Lehmann. The CLEF 2005 automatic medical image annotation task. *International Journal in Computer Vision*, 74(1):51–58, 2007.

[9] M. Ellert, M. Gronager, A. Konstantinov, B. Konya, J. Lindemann, I. Livenson, J.L. Nielsen, M. Niinimaki, O. Smirnova, and A. Waananen. Advanced resource connector middleware for lightweight computational Grids. *Future Generation computer systems*, 23(2):219–240, 2007.

[10] Renato J. Figueiredo, Peter A. Dinda, and Jose A. B. Fortes. A case for grid computing on virtual machines. *Distributed Computing Systems, International Conference on*, 0:550, 2003.

[11] I. Foster. What is the grid? - a three point checklist. *GRIDtoday*, 1(6), July 2002.

[12] E. Heymann, M. A. Senar, E. Luque, and M. Livny. Efficient resource management applied to master-worker applications. *J. Parallel Distrib. Comput.*, 64(6):767–773, 2004.

[13] Intel. Intel virtualization technology. *Intel Technology Journal*, 10(3), 2006.

[14] D. Johnson. Desktop grids. In A. Abbas, editor, *Grid Computing – A Practical Guide to Technology and Applications*, pages 75–98. Charles River Media, 2004.

[15] P. Kacsuk, N. Podhorszki, and T. Kiss. Scalable desktop grid system. In *Proc. High Performance Computing for Computational Science - VECPAR 2006*, pages 27–38. Springer, 2007.

[16] B. Laurie and P. Laurie. *Apache: the Definite Guide*. O'Reilly, 3 edition, 2002.

[17] Michael Litzkow, Miron Livny, and Matthew Mutka. Condor - a hunter of idle workstations. In *Proceedings of the 8th International Conference of Distributed Computing Systems*, June 1988.

[18] Henning Müller, David McG. Squire, Wolfgang Müller, and Thierry Pun. Efficient access methods for content–based image retrieval with inverted files. *IEEE Transactions on Image Processing*, (5), 2004.

[19] Marko Niinimaki, Xin Zhou, Adrien Depeursinge, Antoine Geissbuhler, and Henning Müller. Building a community grid for medical image analysis inside a hospital, a case study. In *Workshop on Grids for medical imaging at MICCAI 2008*, September 2008.

[20] Mikko Pitkanen, Xin Zhou, Antti E. J. Hyvärinen, and Henning Müller. Using the Grid for enhancing the performance of a medical image search engine. In *21th IEEE Symposium on Computer–Based Medical Systems (CBMS)*, Jyväskylä, Finland, June 2008.

[21] David McG. Squire, Henning Müller, Wolfgang Müller, Stéphane Marchand-Maillet, and Thierry Pun. *Design and Evaluation of a Content-based Image Retrieval System*, chapter 7, pages 125–151. Idea Group Publishing, 2001.

[22] Sun Grid Engine Team. N1 grid engine 6 administration guide. Technical report, Sun Microsystems, 2005.

[23] Sun Microsystems. Sun xvm virtualbox user manual. Technical report, Sun Microsystems, 2008. Available at http://www.virtualbox.org.

[24] VMWareInc. Vmware – technical white paper. Technical report, VMWareInc, 1999. Available at http://www.vmware.com.