Using the Grid for Enhancing the Performance of a Medical Image Search Engine

Mikko Juhani Pitkanen* Xin Zhou†Antti E. J. Hyvärinen‡ Henning Müller†§
* Helsinki Institute of Physics, Technology Programme <pitkanen@mail.cern.ch>
† University and Hospitals of Geneva <[xin.zhou;henning.muller]@sim.hcuge.ch>
‡ Helsinki University of Technology <antti.hyvarinen@tkk.fi>
§ University of Applied Sciences, Sierre, Switzerland

***Abstract*— In this paper we show how Grid computing can be used to improve the operation of a medical image search system. The paper introduces the basic principles of a content-based image retrieval (CBIR) system and identifies the computationally challenging tasks in the system. For the computationally challenging tasks an efficient design is proposed that uses a distributed Grid computing to carry out the image processing in distributed and efficient way. The algorithms of the search system are executed by using a real medical image collection as input and a Grid computing infrastructure to provide the needed computing power. Finally, the results show how the image processing task that required tens of hours to complete can be processed by using only a fraction of the originally required computing time.**

I. Introduction

The amount of data produced in medical care is increasing on a rapid pace. Managing the large quantity of information is a challenging task even for an information science specialists. Thus, the complexity of managing the information may prevent the professionals of medical care from making full use of the collected information. This motivates to create simple (web-based) applications to manage and make use of data that is collected in several clinical databases. In this paper, we consider how the MedGIFT [1] (Medical GNU Image Finding Tool) application can be used to perform query by example (QBE) searches from large medical image databases. By using MedGIFT, the medical doctors can use the visual material related to a patient such as newly taken X-ray, to search earlier patient cases that had similar image material referenced in the clinical record. These earlier cases may help the medical doctor to find a suitable treatment.

The MedGIFT project hosted at the University and Hospitals of Geneva has investigated the use of content-based image retrieval tools for searching medical data from large databases. The Radiology department alone produced more than 70'000 images per day in 2007. Large numbers of images make the content–based retrieval task challenging, as for instance indexing a database of 70,000 images for efficient search consumes in the order of 20 hours of processing time on a modern server. In this study we use the MedGIFT search engine as an example of a content-based image retrieval (CBIR) system. MedGIFT is used to build an index with the ImageCLEF[2] image database. We show how distributed computing tools can provide a possible solution to high demand for computational power in CBIR systems. As an example of a distributed computing environment we use an advanced resource connector (ARC) [5] middleware–based Grid system. By integrating these two systems we provide a proof of concept design for distributed feature extraction in CBIR systems based on Grid computing.

The MedGIFT project also aims at enabling novel methods and technologies of informatics within the medical domain. To bring these systems into real use in medical care, a research infrastructure has been set up within the medical informatics research group. The initial infrastructure has been built as a part of EU-funded KnowARC[3] project. The research continues by developing more fine-grained (and thus often computationally more expensive) visual descriptors for new use cases. Moreover, the number of images to be processed is constantly increasing, and the project investigates on how to meet the increasing computing and storage requirements.

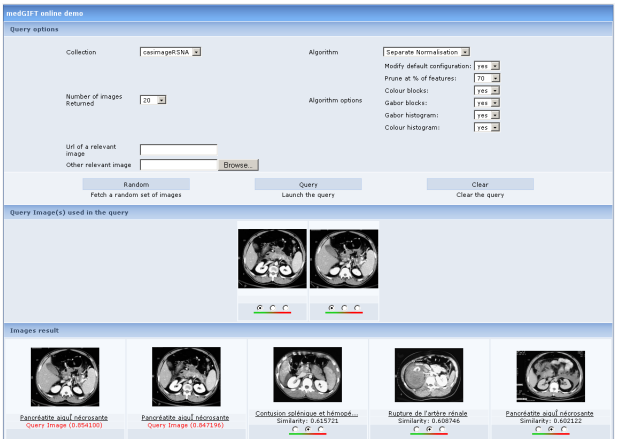

Fig. 1. Web interface of the MedGIFT system.

Figure 1 illustrates the web-based user interface of the MedGIFT application. The user, a medical doctor or medical informatics specialist, starts the search process by providing an example image to be used as a criterion for the query. Based on the given image (or multiple images), the database

[1] http://www.sim.hcuge.ch/medgift/
[2] http://www.imageclef.org/
[3] http://www.knowarc.eu/

is searched for patient cases with similar visual information. The cases containing related images are then shown to the user who evaluates the relevance by assigning a score to each image for further queries.

In this paper, we propose a distributed computing solution which can be used to extract visual features from an image collection. The visual features represent the image in the databases and are used to calculate distances between images. To optimize the algorithms and the parameters used in the extraction task, the medical informatics specialist may need to run the extraction task several times to optimize the feature set for a certain situation. Our solution makes this task significantly more efficient and enables to carry out the processing of the entire image collection several times during a single working day. The proposed solution is general enough to provide an easy-to-use interface to Grid resources, and can be used for other clinical applications too.

Related Work

Earlier work by Montagnat et al. [11] presented a method to partition a database of medical images over a set of distributed resources. The database is partitioned over a set of resources, the query image sent to the resources and the similarity evaluation is carried out for query image and the images stored in the resource. The evaluation is based on using a set of resources within a single site, whereas our study focuses on using heterogeneous resources from multiple administrative domains to analyze images for building a centralized index.

The Globus MEDICUS [13] project has presented an image sharing system that proposes to use the Grid tools for enabling efficient medical image transfer. The system guarantees sufficient security measures to share medical data across different administrative domains.

The computational Grids have been also used to harness large amounts of computational power to identify drug candidates for the influenza A virus [12]. Similarly to our study, they employed Grid infrastructure which extended over diverse sites and the amount of transferred data was large.

Oliveira et al. [17] have shown how the Grid computing can be employed to reduce the computation time in CBIR systems. The authors installed MyGrid architecture in a hospital network and the image analysis libraries were installed to computers in the network. Our solution differs in that we submit jobs over heterogeneous resources and across the organizational boundaries. We also send the minimal analysis libraries with the job and compile the binary in the executing resource.

Breton et al. [15] have proposed the use of Grid technology to transfer patient data over organizational boundaries. Their work discusses many of the challenges in distributing the medical data over multiple sites. Sloot et al. [18] have presented the Grid–based ViroLab decision support system to be used for various challenges in the medical domain.

II. Problem Statement

The of medical information stored in databases within the health care domain is increasing at a rapid pace. The increase in amount of data already renders the task of extracting information intractable for a modern workstation. In this paper, we investigate how Grid computing can be used to help the medical informatics professionals to enhance an image analysis systems.

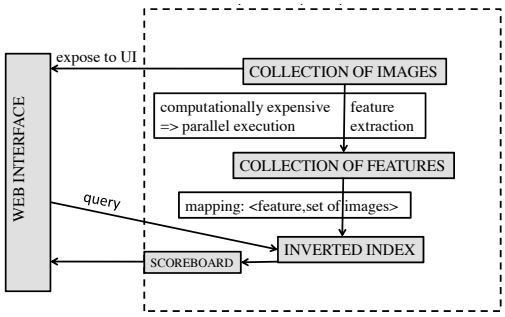

Fig. 2. Functional overview of the MedGIFT system.

Figure 2 illustrates the functionality of the MedGIFT system. The image collection is exposed to the user via a web–interface and the user may browse the collection or start the query with an external example image. Features of every image in the collection are extracted during a computationally intensive task, for which an efficient distributed method is proposed in this study. Afterwards, an index is built to enable the user to perform consecutive queries from a database in an efficient way. The process of building the index requires a large amount of computational power. Moreover, the output of the process (image features) in the index defines how effectively the system can perform a search. Thus, our research problem is defined as minimizing the wall clock time required to extract the features from all of the images in the collection.

III. Methodology

In our implementation, we used the ARC middleware to gain access to computing resources available through the NorduGrid collaboration[4] within the KnowARC project. Our work relied on using the standard way to define the ARC jobs by using *job descriptions*. Moreover, we used a grid job manager to coordinate the distributed execution of jobs.

A. Grid Job for Extracting Image Features

A *Grid job* is comprised of program code together with its input and a task definition. The job is executed sequentially on a remote resource by using a Grid middleware. A typical job has input file(s) and an executable which is used to produce output file(s). A job in the ARC based Grid is defined by writing a *job description* in the extended resource specification language (XRSL). Each job has its own job specification and can be executed independently from other jobs without requiring interaction between jobs. This type of computational tasks are often referred to as *embarrassingly parallel* or *bag of tasks* jobs.

[4] http://www.nordugrid.org/

```
1:  (* executable for the feature extraction *)
2:  ( executable=''python-local.py'' )
3:  (* 1st argument, images for feature extraction*)
4:  ( arguments="imgs-0.tar")
5:  (* inputs, images, extractor source, local coordinator *)
6:  ( inputFiles=(imgs-0.tar "") (src.tar "") (local.py ""))
7:  (* stdout file for storing the results *)
8:  ( stdout=''stdout-gift-0.tar'' )
9:  ( outputFiles=("fts-0.tar" "gsiftp://dn.ch/fts-0.tar"))
10: ( stderr=''geneva-gift0.err'' )
11: (* jobname for easy monitoring *)
12: ( jobname="geneva-gift0")
```

Fig. 3. A job description for distributed execution when using the ARC middleware.

Figure 3 illustrates an example of a Grid job description. Every job sent for execution through the ARC middleware is defined by a corresponding job description. Scheduling is carried out in the ARC client. After the client machine has chosen the remote resource where the job will be executed, the input files defined in row 6 are uploaded to the resource. Row 2 defines the executable to be executed on the remote resource. The file referred to by executable-attribute is also transferred. After successful execution of a job, the Grid middleware writes the results to an output file defined in row 9.

The job description defines the Grid operations and parameters and it is used to choose the remote resource. Once the executable and the input files are uploaded to the remote resource, the executable is started as a process on the remote resource. We have defined an executable that takes care of coordinating the job execution in a sequential manner when started. The executable is a Python script performing a set of operations on input data and as a result produces an output. Then, depending on the job description, the results are either written locally for the retrieval by the client or the Grid middleware is used to upload the results into a remote directory. In the former case the client has to retrieve the job results after the completion, whereas in the latter case the results are immediately uploaded to a resource provided by the user.

B. Grid Job Manager (GridJM) for Distributed Execution

The Grid Job Manager [5] (GridJM), is a system for managing parallel computations in a Grid. The system aims at dynamically adjusting to the available resources from the point of view of a single user. The user is provided with a simple interface to submit jobs and to receive the results of the jobs, whereas details of the distributed infrastructure are hidden. These details include mechanisms to handle Grid-specific operations such as scheduling, reliable processing, submitting, monitoring and downloading the results of a Grid job.

The implementation of GridJM is built on the ARC middleware. We believe that the solution is also applicable to other Grid middlewares, which allow client–side scheduling. Several studies on scheduling in Grid environments exist. For examples, see [14], [16].

[5] http://www.tcs.hut.fi/~aehyvari/gridjm/

While GridJM also offers a simple Grid abstraction for the user, the core of the system is the dynamic per-user view of the remote resources. Whereas in a stable distributed environment, only functions for submitting, monitoring and result retrieval are required for efficient operation, our results indicate that job failures and high variance in execution time constitute a major challenge for obtaining high parallelism in Grids. The unpredictability results from non-trivial inherent properties of Grids, including

- incomplete scheduling information, such as policies in the queues of the remote resources,
- occasional badly configured resources,
- remote hardware failures, and
- communication problems in the network.

Job failures resulting from such reasons are handled by GridJM with a resubmission scheme also known as job migration.

To further decrease the likelihood of failures, GridJM also provides a layer of fault avoidance based on short-term historical observations of previous jobs. The system assumes that if a job has recently executed in a non-optimal manner on a certain resource, it is likely to execute badly in the near future on the same resource. By maintaining the information related to the failed jobs, GridJM avoids submitting jobs to these resources. Resources having been re-configured may again be included as potential scheduling targets after a predetermined amount of time. Maintaining the job failure information in the client application gives the benefit of having the failure information per application. This approach seems to work better with a wide diversity of applications than a centralized approach.

GridJM runs as a daemon process on the client machine. The job manger listens on a network socket and accepts XRSL job descriptions. For each accepted description, GridJM manages the execution of the job on the Grid infrastructure. GridJM immediately reacts to a finished execution of a job by receiving the output and reporting the results back to the socket. The interface simplifies the execution of complex jobs such as processing an entire collection of medical images. In practice, an application developer has only to send n XRSL descriptions to the socket and to wait for n announcements of the completed executions. The task is completed when all the announcements have arrived. This functionality has proven us that GridJM can provide an appealing solution to the management of MedGIFT feature extraction in ARC–based Grids.

IV. Results

This section explains our design for the distributed feature extraction in the Grid. First, we show how the client application creates several sub–collections of the database of images. The images can then be sent in packages with appropriate size to minimize the overhead of scheduling, security operations, and the queuing for the entire database. Several sizes of sub–collections were evaluated. For brevity, we only show the results for the jobs with 1000 images, which was seen as a good batch size. Before sending the images for analysis,

the database is pre–processed locally, and the images are converted to bitmap format (PPM – Portable PixMap), which is required as input for the feature extraction executable. The bitmap format allows compressing the set images to reduce the amount of data that needs to be transferred.

A. Design for a Distributed Analysis of Medical Images

Figure 4 illustrates how a large image collection is prepared for execution with the ARC middleware. The machine hosting the image collection and now working as the Grid client machine starts by searching the image collection's directory tree for the medical images.

```
1: imageCollection = collectionDirectoryTree.walk()
2:
3: batchSize = N
4:
5: for anImage in imageCollection:
6:     currentBatch = []
7:     while(currentBatch.size() < batchSize):
8:           add(anImage, currentBatch)
9:     createJobDescription(currentBatch)
10:    packageAndCompressInput(currentBatch)
```

Fig. 4. Creating image archives for distributed feature extraction.

The images are packaged into sub–collections with configurable size. The size of the sub–collections can be defined to suit the computational and data transfer capabilities of the used computing environment. For a global Grid with high latencies, both during data transfer and queuing, it is reasonable to use rather large sub–collections[6]. Large job sizes lead to less parallelism but introduce less overhead to be caused by the Grid security and scheduling operations. Finally, a job description is created for each sub–collection. The description along with any input can then be submitted to the remote resource.

```
1: extractArchive(images-nnn.tar.gs)
2: extractArchive(extractor-sources.tar.gz)
3: buildExtractorBinary()
4:
5: for aFile in workingDir.walk():
6:       if (recognizedAsImageFile(aFile)):
7:              aFeature = extractFeatures(aFile)
8:              addToFeatureArchive(aFeature)
9:
10: packageAndCompressOutput()
```

Fig. 5. Remote run of a Grid job for feature extraction.

Figure 5 illustrates how the feature extraction is carried out for images of a single Grid job. First, the input files containing the feature extraction source code and the input images are extracted. Then, the source code for the feature extraction is compiled. After this, the files in the local job directory are walked through. When a file is identified as an image file, it is provided as input to the executable that extracts the features from this image. The output of the extraction (feature vector of the image) is added to the output of the job. In the end, the middleware transfers the output away from the resource and cleans the job directory.

[6]A Job with an execution time of half an hour was suggested in a discussion with experienced ARC professionals.

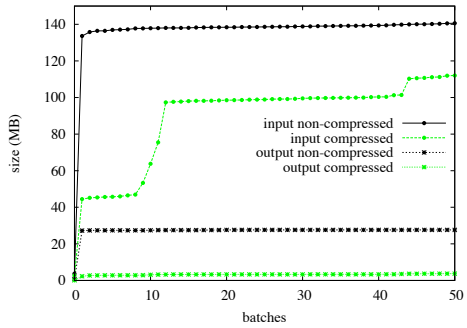

Fig. 6. Image sub–collections of 1000 images and the effect of compression.

Figure 6 illustrates the effect of compression on the size of the input and the output files. Each file contains multiple images or features and is compressed by using the well–known `gzip` application on a Linux machine. The sizes of the compressed and non–compressed files are sorted by increasing order. The compression produces a significant decrease in the file size. As a benefit, smaller size of inputs (and outputs) decreases a time for data transmission and is especially helpful when the communication resources are the bottleneck of the distributed execution rather than the computational resources. Since the time overhead associated with compression and decompression is insignificant compared to the gain in transfer time, we always compress the images before sending the jobs to Grid and decompress the inputs in the remote host before starting the analysis.

Moreover, earlier work [13] has shown how image record (image and metadata) transfer time can be reduced significantly by using compressed records together with Grid data transfer. Similar to our approach, the method is more efficient than direct transfer using the Digital Imaging and Communications in Medicine (DICOM) standard.

B. Measurement — Simple Job Submission

A straightforward approach to using Grids for processing the entire image collection was to create a script that used the ARC client to submit all the jobs to the Grid one after another. After successful completion of the job, the output files were directly uploaded to a GridFTP server on which the client (and MedGIFT search engine) was hosted. However, this approach has two major drawbacks. First, the GridFTP server needed to be installed on the resource hosting the client. The sever requires firewall policies to allow incoming data transfers from multiple resources on which the jobs are executed. Even though the need to open the firewall ports could be circumvented by the client actively fetching the results, this leads to a more complex design in which the client needs to know about the status of the job. Second, the jobs that have failed need to be submitted again. To re–submit a job the client has to take care about monitoring the job states, the progress of jobs and has to record the causes of job failures.

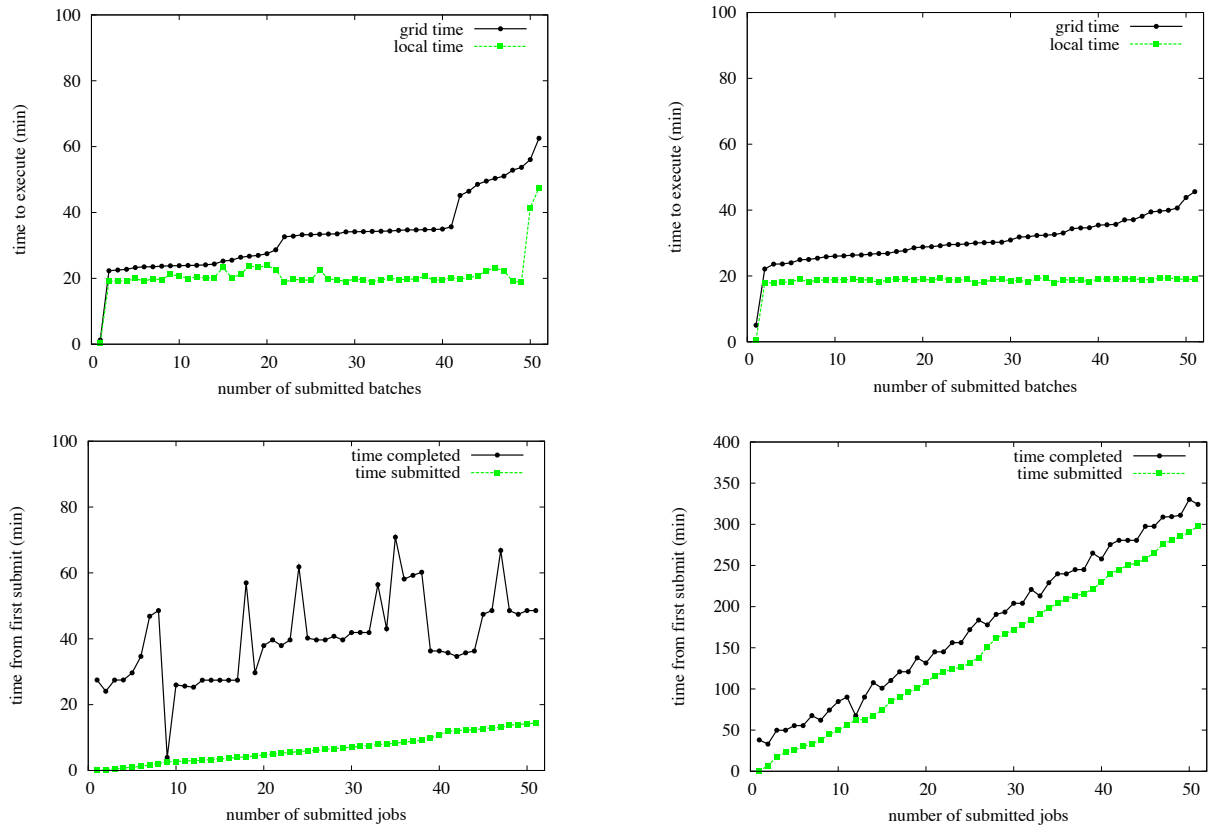

Fig. 7. Client run in Finland (left) and in Switzerland (right), job size 1000 and the job-by job (top-row) and overall (bottom-row) execution.

Number of images in batch	**500**	**1000**
Number of jobs (batches)	101	51
Number of successfully completed jobs	94	37
Average time to return results after submitting the job	12 min 38 s	20 min 11 s

TABLE I
THE RESULTS FROM FIRST TESTS WITH SIMPLE EXECUTION ON THE GRID.

Table I illustrates the success rate of jobs and the time consumed per job when we used the simple sequential job submission. The high failure rate of jobs creates a need to either take care of resubmitting the failed jobs or to create more sophisticated coordination of the job (re–)submission. The lack of results from the failed jobs leads to a situation where part of the image features cannot be included into the search index without first extracting features from them. The extraction could be done either locally or by submitting the job to the Grid. In both cases the overall execution time increases significantly.

C. Measurement — GridJM to Manage Distributed Execution

To overcome the unreliability issues explained in section IV-B we used the GridJM job manager [2]. The grid job inputs created by the method illustrated in Figure 4 were given to the job manager. The manager application then executes the jobs until the corresponding results for each input are downloaded to the client machine.

The top row of Figure 7 illustrates the time used for each individual Grid job that executes the feature extraction. For each job, 'Grid time' shows the time from starting to upload the input until the results have been downloaded to the client machine. Thus, the Grid time includes the overhead components of the distributed execution such as scheduling, stage–in (upload), queuing, keeping the job status up–to–date, stage–out (download), and possible resubmissions in the case of failed jobs. The component 'local time' is measured by using the Linux command `/usr/bin/time` to capture the wall clock time used to extract the features in the remote resources. For the client in Finland, we can observe more diverse execution times for individual jobs than for the client in Switzerland. This depends on the choice of executing nodes made by the scheduling in the client. However, from the application point of view it is more important how fast the entire collection can be executed.

The bottom row of Figure 7 illustrates the overall progress of the extraction task for the entire image collection. The 'time submitted' shows the elapsed time after starting the upload for the first job, which is chosen as the origin of the time axis. The 'time completed' measure shows the amount of time that has elapsed from the start before the results are retrieved for a particular task. The time values are plotted in the order of

submission and each individual Grid job is presented as a point on the x–axis.

Running the client application in two different locations resulted in notably different performance results. When the client is run in Finland, the individual jobs have less uniform execution times, but the entire collection is analyzed in shorter time. The explanation is that it takes a much longer time for the client in Switzerland to upload the results (to computing resource that are in large part in Scandinavia). Even if there would be resources to execute the jobs, the parallelism is decreased because of the bottleneck caused by the network resources.

V. Interpretation of Results and Conclusions

In this paper we presented how Grid computing can be used to address the high computing demands of a medical image search engine. A design for distributed image analysis was proposed, and the results showed how we can execute the analysis task that required tens of hours to execute to run within only a fraction of the original time requirement. This solution greatly helps the task of a medical informatics specialist as (s)he can receive feedback from the analysis task multiple times during a single workday. Furthermore, the growth of the computing capacity enables to design novel types of applications, where analyzing a single image would take up to minutes instead of just a second. These applications then enable better quality of search for the increasingly detailed medical images in different special domains.

One of the most important benefits of the design is that the jobs can be executed on heterogeneous resources. This enabled us to use the resources of the collaborating partners that have been made available through the EC-funded KnowARC project. The large amount of the available Grid resources that were accessible enabled experimenting with efficient computing resources even without having the budget to acquire the resources for the CBIR research. This helps especially when prototyping the distributed execution of the novel medical applications.

References

[1] Henning Müller, Mikko Pitkanen, Xin Zhou, Adrien Depeursinge, Jimison Iavindrasana, Antoine Geissbuhler, "KnowARC: Facilitating Grid networks for the biomedical research community", Proceedings of the 5th HealthGrid Conference 2007, pages 261-268, Geneva, Switzerland, 2007.

[2] GridJM homepage, `http://www.tcs.hut.fi/~aehyvari/gridjm/`

[3] A. E. J. Hyvärinen, T. Junttila and I. Niemelä, "A Distribution Method for Solving SAT in Grids". Proceedings of the 9th International Conference on Theory and Applications of Satisfiability Testing, Seattle, August 12 - 15, 2006

[4] Henning Müller, "User Interaction and Performance Evaluation in Content-Based Visual Information Retrieval", Doctoral Thesis, University of Geneva 2002.

[5] Mattias Ellert et al., "Advanced Resource Connector middleware for lightweight computational Grids", Future Generation Computer Systems, 23 (2007).

[6] V.Welch, F. Siebenlist, I. Foster, J. Bresnahan, K. Czajkowski, J.Gawor, C. Kesselman, S. Meder, L. Pearlman and S. Tuecke, "Security for Grid Services", Proceedings of IEEE HPDC-12, 2003.

[7] NorduGrid Grid Monitor, `http://www.norduGrid.org/monitor/`

[8] Tobias Gass, Antoine Geissbuhler, and Henning Müller, Learning a frequency-based weighting for medical image classification, MIMI 2007, Beijing, China, pages 151-161, 2007.

[9] Henning Müller, David McG. Squire and Thierry Pun, "Learning From User Behaviour in Image Retrieval: Application of the Market Basket Analysis, International Journal on Computer Vision, volume 56(1-2), pages 65-77, 2004.

[10] Henning Müller, Antoine Rosset, Arnaud Garcia, Jean-Paul Vallee, and Antoine Geissbuhler, "Benefits of content-based visual data access in Radiology", Radiographics, volume 25 (3), pages 849-858, 2005

[11] Johan Montagnat, Vincent Breton, and Isabelle Magnin. "Partitionning medical image databases for content-based queries on a grid" in Methods of Information in Medicine (MIM), 44 (2), pages 154-160, 2005

[12] Hurng-Chun Lee , Jean Salzemann , Nicolas Jacq , Hsin-Yen Chen , Li-Yung Ho , Ivan Merelli , Luciano Milanesi , Vincent Breton , Simon C Lin, and Ying-Ta Wu, "Grid-enabled high-throughput in silico screening against influenza A neuraminidase", IEEE Trans Nanobioscience. 2006 Dec ;5 (4).

[13] Erberich SG, Silverstein JC, Chervenak A, Schuler R, Nelson MD, and Kesselman C., "Globus MEDICUS - Federation of DICOM Medical Imaging Devices into Healthcare Grids", Studies in Health Technology and Informatics, IOS Press, Volume 126, p:269-278, 2007

[14] Lingyun Yang, Jennifer M. Schopf, and Ian T. Foster "Conservative Scheduling, Using Predicted Variance to Improve Scheduling Decisions in Dynamic Environments", Proceedings of the ACM/IEEE SC2003 Conference on High Performance Networking and Computing, 2003.

[15] Breton V. Blanquer I., Hernandez V., Legre Y., Solomonidés, T., "Proposing a roadmap for Healthgrids", Stud Health Technol Inform, vol. 120, 2006.

[16] Nicola Tonellotto, Philipp Wieder and, Ramin Yahyapour, "A Proposal for a Generic Grid Scheduling Architecture", Proceedings of the Integrated Research in Grid Computing Workshop, 2005

[17] Costa Oliveira M., Cirne, W., and de Azevedo Marques P., "Towards applying content–based image retrieval in clinical routine", Future Generation Computer Systems, vol. 23, 2007.

[18] Sloot P., Tirado–Ramos A., Altintas, I., Bibak M, and Boucher C., "From Molecules to man: Decision support in individualized e-health", IEEE Computer, vol 39, number 11, 2006.