Partitionning medical image databases for content-based queries on a grid

Johan Montagnat, Vincent Breton, Isabelle Magnin

Submission to the *Methods of Information in Medicine* journal special issue on *HealthGrid'04*.

Title: Partitioning medical image databases for content-based queries on a grid

Authors and affiliation:

Johan Montagnat

CREATIS, CNRS UMR 5515-INSERM U670

Vincent Breton

LPC, CNRS

Isabelle E. Magnin

CREATIS, CNRS UMR 5515-INSERM U670

Corresponding author :

Johan Montagnat

CREATIS, bât B. Pascal,

20 av. A. Einstein

69621 Villeurbanne Cedex

France

Mail : johan@creatis.insa-lyon.fr

Tél : +33 4 92 94 27 20 / +33 4 72 43 63 87

Fax : +33 4 92 94 28 98

Summary

Objectives: In this paper we study the impact of executing a medical image database query application on the grid. For lowering the total computation time, the image database is partitioned in subsets to be processed on different grid nodes.

Methods: A theoretical model of the application complexity and estimates of the grid execution overhead are used to efficiently partition the database.

Results: We show results demonstrating that smart partitioning of the database can lead to significant improvements in terms of total computation time.

Conclusions: Grids are promising for images content-based retrieval in medical databases.

Keywords: Medical images content-based retrieval, databases search, Grid computing.

Partitioning medical image databases for content-based queries on a grid

Johan Montagnat[a], Vincent Breton[b], Isabelle E. Magnin[a]

[a] CREATIS, CNRS UMR 5515-INSERM U670, 20 av. A. Einstein, 69621 Villeurbanne, France

[b] LPC, Université B. Pascal, 24 av. des landais 63177 Aubière cedex, France

1 Introduction

Medical imaging is data intensive due to the size of medical images. Digital medical images represent tremendous amounts of data for which automated indexing and search tools are increasingly needed [2][3]. With the arising of medical record databases in hospitals, physicians have access to precious datasets containing an history of sample data, diagnoses, medical interventions, and results, that could help in indexing and analyzing new data. However, tools are needed to manipulate and search for relevant data in these databases. Due to the amount of data (each medical image represents MB to GB of data) and the complexity of image analysis algorithms, these tools are both data and computationally intensive [4].

Computer grids are promising to handle such applications. Grids provide massive processing power and a potential for a high level of coarse grain parallelism well suited to tackle queries on full medical image databases in a reasonably short time. However, grid technologies are still in their youth and many problems have to be solved before they can be used in clinical environment. Beside the security issues that are fundamental for this field [5][6], the potential parallelism of grids may be difficult to exploit. In particular, the overhead introduced by the grid middleware and the need to deal with resources distribution may badly impair the performances of some applications.

In this paper we focus on a medical image similarity measurement application. Image similarity measures are used in various medical image registration and recognition algorithms [7][8]. They estimate quantitatively the similarity between the content of two images. The typical usecase for this application is a physician searching for known medical cases close to a case he has to diagnosis: he wants to find in the database all images with a close correlation to a sample image he is studying to be able to confirm his diagnosis by looking at other similar records. To perform efficient queries on a grid, the system need to partition the database in subsets that will be independently processed on different processors. We study the trade-off between distributing a small number of large jobs dealing with large datasets and a large number of small and short jobs.

2 Objectives

2.1 Application description

We assume that this application is running on a grid of standard PC machines connected to a local area network. The medical images are available from a medical data server recording both images and metadata associated to these images. Figure 1 illustrates this application. The user first select a sample image. A dataset of candidate images (i.e. images of the same region body, acquired with the same imager, etc.) is determined by selecting images on their metadata in the database. The candidate images are then transported to the grid for analysis. For each candidate image, a similarity measurement is computed between the candidate and the user sample. This measure results in a score attributed to the candidate. Once all candidates have been processed, the scores are ranked and the user can retrieve the highest score images corresponding to the most similar cases stored in the database.

Figure 2 shows an example of the algorithm execution. The upper left image is the user sample image (a Magnetic Resonance Image of the thorax). The other images are candidates (all classified as thorax MRI in the system) ranked by their similarity score (2 high scores, 2 low scores).

Several similarity measures may be used to measure the differences between images. Although each measurement is not very computer intensive, the comparison of a sample image against a complete database is intractable, in a reasonable time, on a single computer due to the size of medical databases. The actual cost of such a computation depends on several parameters such as the input image size and the computation precision desired as detailed in section 3.1. Moreover, a job may be started for each candidate image but one might want to execute one job computing several scores for a subset of the input candidates, thus loading the system with a reduced amount of jobs. This application is therefore very scalable and the optimization of the computation cost on a distributed system is not obvious.

Executing the application using a grid middleware introduces a significant overhead on the computation cost. The grid reduces the computation cost by introducing parallelism but the counter part is an increase in data manipulation and job scheduling cost. This overhead may be negligible for very long processing jobs but it is not the case when each job computation time is below a given threshold. A trade-off has therefore to be found: either to submit a small number of large jobs, each dealing with a large subset of the input data, or a large number of small jobs, each dealing with a small subset or even a single image. Reducing the number of jobs reduces the system overhead while lowering the potential application parallelism. There are also implications in terms of data replication as described in the next section. Figure 3 shows the typical execution time for a same query executed at different granularities (the X axis representing the granularity level). An optimal value significantly improves the computation time.

2.2 Grid middleware

The *middleware* is the software layer offering basic services to access a grid infrastructure and hiding the system complexity to the user. The middleware and its capabilities have a significant impact on the application computation time. Although grid technologies are still in their youth, the growing activity around grids lead to many middleware or specialized middleware components development. Early hour middlewares focused on embarrassingly parallel applications [9] but more sophisticated approaches are being proposed to deal with different problems [10]. An important component for medical applications is the data manager. Our application manipulates both data and associated metadata. All medical data are sensitive and their access should be strictly controlled on a grid, which is something difficult to achieve on a distributed environment.

Our application has been developed on our own middleware layer, after earlier testing on the European DataGrid project middleware [9]. Our middleware is designed to be very light weighted in order to remain easy to use and maintain. It was designed to access cluster of PCs available in laboratories or hospitals. Therefore, it does not make any assumption on the network and the system installation except that independent hosts with their private CPU, memory, and disk resources are connected through an IP network and communication is possible on one port with each machine. This middleware is currently in an early development phase although it provides the basic functionalities needed for such an application. It is written in C++, C doubtlessly being the ideal language for system programming. It is built on a few standard components such as the OpenSSL library [11] for authenticated and secured communications, and the MySQL C interface to access the MySQL database server [12].

Our middleware includes a data manager and a job controller. A daemon is running on each host (later on referred to as *grid node*) to receive data and job related requests. A farm manager is the entry point in the system. The user can connect to the farm manager from any host through a programmable user interface or command line tools. If she is properly authenticated, her request is transmitted to a grid node for processing. The client then make direct connection to the grid node for data or job information exchange to release the farm manager. As can be seen, this solution is not a real grid implementation: it is not scalable yet as it is centralizing access on a single farm manager. This is meant to evolve in future developments. However, the farm manager is developed to remain as light weighted as possible, delegating every possible actions to the grid nodes. The middleware is working on an active policy. Each time a worker node is started or finishes a job, it declares itself to the farm manager. The farm manager keep track of the busy and ready nodes. Once a node becomes ready, it can receive requests from the farm manager.

Authentication. Each user is authenticated through an X509 certificate. The certificate is signed by a certification authority. A farm manager or a grid node will only accept valid and signed certificates. All communication between the user and the nodes are encrypted using the OpenSSL public key interface. Each grid node also authenticates itself with a certificate. The client interface

or the farm manager will not accept communication with unauthenticated hosts for security reasons.

Data management. Since we do not make any assumption on each host file system, each host is supposed to dispose of its own storage resources, not necessarily visible from the other hosts. At creation time, a grid node declares its available space to the farm manager and will send frequent updates of this value. The farm manager holds a catalog of all files known to the middleware. Each file is described by a unique grid wide identifier. Thus the user can refer to a file without needing to know its physical location. A file becomes known to the grid once it has been registered: the user transfers the file from its local machine or any external storage through the middleware interface. The file is registered (its identifier is written in the farm manager files table) and a physical copy is stored on a grid node. The farm manager holds a table giving associations between the file identifier and its physical replicates. Several replicates may exist on several nodes for a file. Indeed, when an host is responsible for executing a job, it needs to access a set of files manipulated by this job. Since all job files are not necessarily located on a single node, they are first copied onto the target node before the job is started. These multiple instances of a file are then kept on the nodes, unless disk space is lacking, for caching in case of subsequent use. This replication of files causes an obvious problem of coherence that is not handled in the current implementation: the user is responsible for creating a new file in case of modification. Our middleware controls the access to file authorization through the user certificate subject. The subject string is stored in the farm manager table on file registration allowing the system to control file access at each user level.

Job control. The farm daemon is also responsible for assigning jobs to grid nodes. When the user submit a job, he can specify some system requirements and the files needed for this job to execute. The farm manager will search for possible target nodes matching the system requirements. It sorts the possible candidates list on (1) their readiness, (2) the amount of data that has to be transferred before starting the job, and (3) their processing power. After this basic scheduling, the job is assigned to the first host that becomes ready in the list. The farm manager thus orders automatic replication of files on need. Replication is actually done directly between the grid nodes owning and receiving a replica.

2.3 Optimization problem

Given the middleware registration procedure, the execution time for a job includes: (1) the file registration time if the job requires file from outside the grid, (2) the job scheduling and queuing time t_{sch}, (3) the time for files replication when needed t_{rep}, and (4) the job execution time t_{job}. In this paper we will ignore the job registration time that depends on external components and we will consider that all needed files have been pre-registered.

Let us now consider the image similarity measurement application where N candidate images should be tested against the user sample image. For optimizing the computation time, we might want to process data by subset of k images, resulting in N/k jobs to be processed. Each job is the sequential execution of k similarity measures and its total execution time is therefore kt_{job}. Assuming that a sufficient amount of resources is available to process all jobs in parallel, the total parallel computation time will be kt_{job} while the sequential computation time would be Nt_{job}.

The grid overhead is the time needed to schedule the jobs and replicate the files for each of them: $N/k(t_{\text{sch}} + t_{\text{rep}})$ where t_{sch} is a function of the number of jobs to process, N/k, and t_{rep} is a function of the number of images to transfer, k, and the images size, n. Our purpose is to find the optimal value for k such that:

$$k = \min_{k} \quad \frac{N}{k}\left(t_{sch}\left(\frac{N}{k}\right) + t_{rep'}(n,k)\right) + kt_{job} \qquad (1)$$

In order to estimate the optimal value for k we need to estimate t_{sch}, t_{rep}, and t_{job}. t_{job} can be derived from the insight of the algorithm as shown in section 3.1. t_{sch} and t_{rep} are dependent on many parameters and we have been measuring suitable values from testing runs.

3 Methods

3.1 Similarity measures complexity

The complexity of all similarity measures depends on the size and the dynamic range of the medical images processed. Furthermore, the complexity of computations varies from one measure to another. We give here a brief overview of the similarity measures implemented for this test and we estimate the computational complexity. All similarity measures proposed first need the computation of the joint histograms of the images to compare. Then the similarity measure itself is estimated.

Let I and J denote the two images to compare. Both images are supposed to have the same size with length l, height h, and number of slices s. Thus the total number of voxels per image is $n = l \times h \times s$. Medical images are gray level images usually coded using 8 or 16 bits. Let $r \in [2^8, 2^{16}]$ be the dynamic range of the image gray level values.

We denote n_{ij} the number of voxels with intensity $i \in [0, r]$ in the first image and $j \in [0, r]$ in the second image (*i.e.* the cell *(i,j)* of the joint histogram). Let us define: $n = \sum_{ij} n_{ij}$ the total number of voxels, and $p_{ij} = \sum_{ij} \frac{n_{ij}}{n}$ the normalized number of voxels in cell *(i,j)* of the joint histogram. We further define:

- $p_i = \sum_i p_{ij}$ and $p_j = \sum_j p_{ij}$ the lines and columns normalized sum

- $m_I = \sum_i p_i$ and $m_J = \sum_j p_j$ — I and J means
- $s_I^2 = \sum_i (i - m_I)^2 p_i$ and $s_J^2 = \sum_j (j - m_J)^2 p_j$ — I and J variances
- $m_{J|i} = \frac{1}{p_i} \sum_j j p_{ij}$ and $m_{I|j} = \frac{1}{p_j} \sum_i i p_{ij}$ — the conditional means
- $s_{J|i}^2 = \frac{1}{p_i} \sum_j (j - m_{J|i})^2 p_{ij}$ and $s_{I|j}^2 = \frac{1}{p_j} \sum_i (i - m_{I|j})^2 p_{ij}$ — the conditional variances

Using the above notations, we have implemented 6 common similarity measures:

- The sum of differences *SD*: $SD(I,J) = \sum_i \sum_j p_{ij} |i - j|$
- The sum of squared differences *SSD*: $SSD(I,J) = \sum_i \sum_j p_{ij} (i - j)^2$
- The coefficient of correlation *CC*: $CC(I,J) = \sum_i \sum_j \frac{(i - m_I)(j - m_J)}{\sqrt{s_I}\sqrt{s_J}}$
- The woods criterion *Woods*: $Woods(I|J) = \sum_j \frac{s_{I|j}}{m_{I|j}} p_j$
- The ratio of correlation *RC*: $RC^2(I|J) = 1 - \frac{1}{s_I^2} \sum_j s_{I|j}^2 p_j$
- The mutual information *MI*: $MI(I|J) = -\sum_i \sum_j p_{ij} \frac{p_{ij}}{p_j}$

For cost estimation, we will consider a time unit c roughly corresponding to the time needed for a executing floating point operation on the microprocessor (*i.e.* c is in the order of few nanoseconds on a 1GHz processor).

3.2 Joint histogram computation cost

The joint histogram is a r x r sparse matrix. Its construction means the computation and storage of all p_{ij} values. The joint histogram can be stored in a 2 dimensions array if r is small enough. However, $r = 2^{16}$ would imply a 2^{32} cells histogram which is too large to fit in most machines memory. Therefore, we store the large sparse histograms as an array of r lines, each made of a linked list of non null column cells. In practice, we set the switch threshold to $r = 2^{12}$: for $r = 2^{12}$, the joint histogram is a 2 dimensions matrix while for $r > 2^{12}$, the joint histogram is a vector of lists. Therefore, the joint histogram computation cost and access time depends on r.

The joint histogram construction involve the initialization of the cells, the images parsing to compute the n_{ij} and the normalization to compute the p_{ij}.

Joint histogram for $r = 2^{12}$

The joint histogram computation using an r x r array involves:

- The initialization (allocation and affectation) of the r^2 matrix cells. The unitary cost for allocating and initializing a cell is estimated to $5c$ from empirical measurements.
- The parsing of all image voxels (n retrieval and additions). The unitary cost is estimated to $12c$.
- The normalization of all coefficients (r^2 retrieval and divisions). The unitary cost is estimated identical as above: $12c$.

This sums to:

$$H(n, r = 2^{12}) = 5\, r^2 c + 12\, nc + 12\, r^2 c = (17\, r^2 + 12n)c \qquad (2)$$

Joint histogram for $r > 2^{12}$

The joint histogram computation using a sparse matrix involves:

- The initialization (creation of r empty lists).
- The parsing of all n image voxels and the histogram update.
- The normalization.

The unitary costs for all these operations are to some extent implementation dependent and difficult to determine theoretically due to the compiler optimizations while generating code. Therefore, we have made measurements of the average costs in our code, leading to the following estimates:

- Each list creation costs $20c$.
- The cost of each update in the histogram depends on the pixel value (all lists are not evenly balanced) averages to $5000c$.
- The normalization of each histogram row averages to $4000c$.

This sums to:

$$H(n, r > 2^{12}) = 20\boldsymbol{rc} + 5000\boldsymbol{nc} + 4000\boldsymbol{rc} = (4020\boldsymbol{r} + 5000\boldsymbol{n})\boldsymbol{c} \qquad (3)$$

Assembling equations 2 and 3, the construction cost is therefore:

$$H(n,r) = \begin{cases} (17r^2 + 12n)c & if \quad r \le 2^{12} \\ (4020r + 5000n) & if \quad r > 2^{12} \end{cases} \qquad (4)$$

3.3 Similarity computation cost

The similarity measures computation cost is highly dependent on the cost for accessing the p_{ij} histogram value. Based on the above mentioned assumptions, this cost is estimated to be $12c$ for $r<2^{12}$. In this case, the computation cost of each similarity measure may be estimated.

Let us first estimate the computation cost for the statistical values p_i, m_I, and s_I. Let $C()$ designate the cost function:

- set of p_i, $\forall i$: C(p_i) = r C($\sum_i p_{ij}$) = r (r x 12c) = 12r^2c
- m_I or m_J: C(m_I) = C(m_J) = C(? $_i$ ip_i) = r C(ip_i) = r(c + 12rc) = 12r^2c + rc ? 12r^2c
- s_I or s $_J$: C(s_I) = C(s $_J$) = C(? $_i$ (i - m_I)^2 p_i) = rC((i - m_I)^2 p_i) = 3rc
- subsequently, the computation time sums up to: C(m_I, m_J, s_I, s $_J$) = 24 r^2c + 6rc ? 24 r^2c

Given the above statistics computation cost, it is now possible to estimate the similarity measures cost. For instance, in the case of the Coefficient of correlation:

$$C(CC,r) = C\left(\sum_i \sum_j \frac{(i-m_I)(j-m_J)}{\sqrt{s_I}\sqrt{s_J}}\right) + C(m_I, m_J, s_I, s_J) = r^2(7c) + 24r^2c = 31r^2c \quad (5)$$

The total computation cost for a similarity measure M applied over two images with n voxels and dynamic range r = 2^{12} is therefore:

$$C(\mathrm{M}, n, r) = H(n, r) + C(\mathrm{M}, r) \quad (6)$$

where $H(n, r)$ is defined in equation 4 and C(M, r) is computed as in equation 5.

For r > 2^{12}, the theoretical cost for similarity measures is made difficult due to the use of the sparse matrix. Indeed, the sparse matrix fullness depends on the actual images gray level dispersion. This matrix fullness has a direct effect on the p_{ij} retrieval time and therefore on the overall computation time. On one hand, the sparse matrix structure increases the time for the retrieval of a p_{ij}, but on the other hand, as we only consider non zero values of the histogram, the number of operations involved in similarity measures is often far less than n or n^2 in practice.

In our experiments on 16 bits voxel images (see Table 2), it appears that the similarity computation time is much smaller than the joint histogram computation time and that it is in the order of magnitude of the computation time for 12 bits voxel images. We will therefore make the approximation:

$$C(\mathrm{M}, r = 2^{16}) \approx C(\mathrm{M}, r = 2^{12}) \quad (6)$$

3.4 Experimental validation

We have used 3 sets of images for validating the computation cost model as summarized in Table 1. Each images gray level range may be undersampled prior to processing. This loss of precision brings an improved computation time. Following experiments are therefore using undersampled version of the original images to 8 and 12 bits when possible.

Table 2 shows the measured computation time (in seconds) for the similarity measures (excluding image I/O and undersampling) on pairs of the 3 above mentioned image datasets, the 6 similarity measures proposed, and every possible undersampling to 8, 12, and 16 bits. The times where measured on a 800MHz Intel Pentium III processor.

Table 2 may be compared to the theoretical values computed using equations 4 to 11 with c = 20ns. Results shown in Table 3 are consistent with Table 2 and the model may be used for predicting computation time in an optimized partitioner.

3.5 Submission cost

The job submission cost is more difficult to model as it depends on a large number of parameters and inter-system interactions. Figure 4 shows two curves measured at run time that we are using to estimate the submission parameters. On the left is shown the time needed for files replication. On the right is shown the scheduling time curve.

The file replication cost linearly depends on the image size n. The larger the job granularity is, the higher the number of files to transfer. Therefore, we make the assumption that the replication cost linearly depends on k and we estimate the scheduling time such that:

$$t_{\text{rep}}(n,k) = n(p_1 k + p_2) \tag{12}$$

where p_1 and p_2 are estimated by linear regression on the data shown in Figure 4. We found p_1=5.19 10^{-7} and p_2=1.572 10^{-6} in this case.

The scheduler time cost is clearly non-linear. It depends on the number N/k of jobs the scheduler has to deal with. Given the shape of the scheduling time curve, we match it with a quadratic function:

$$t_{sch}\left(\frac{N}{k}\right) = \max\left(0, p_3\left(\frac{N}{k}\right)^2 + p_4\frac{N}{k} + p_5\right) \tag{13}$$

A non-linear regression iterative procedure lead to: p_3=1.02, p_4=0.08, and p_5=-54.6.

4 Results

Figure 5 shows the measured computation time (plain line) and the estimated time (dashed line) in function of the granularity k for two experiments. The former involves 144 tridimensional images of size n = 181x217x181 and the later 100 tridimensional images of size n = 150x160x65. The 16 bits images where undersampled to 12 bits (r = 2^{12}) and the coefficient of correlation similarity measure was used: t_{job} = $48r^2c$+$12nc$. Inserting this value into equation 1 lead to the dashed-line estimates shown in Figure 5.

All experiments have been lead on a farm of 8 Pentium III 1GHz PCs with 1GB of RAM and 10GB of free disk space. The estimated curves are rather approximate but sufficient to choose a reasonable value of k. However, the model is sensitive to the sizes of the processed images: for much smaller (*e.g.* 2D) images, t_{rep} and t_{sch} need to be estimated from different experiments.

5 Conclusions

Application granularity control is important to optimize the performance of parallel applications with relatively short jobs in grids. However, the control at application level requires an in-depth study of the algorithm complexity that might prove to be difficult for some algorithms. Further work is needed to determine whether this could be done at the middleware level by monitoring the running applications and estimating the computation cost from measured data.

Acknowledgements

This work is partly funded by the French ministry for research ACI-GRID program [1] and the French region Rhône-Alpes RAGTIME project.

References

1. MEDIGRID, French ministry for Reseach ACI-GRID project. http://www.creatis.insa-lyon.fr/MEDIGRID/.

2. Tweed T. and Miguet S. “Distributed indexation of a mammographic database using the grid”. *International Workshop on Grid Computing and e-Science*. 17th Annual ACM International Conference on Supercomputing. San Francisco, USA, June 2003.

3. Montagnat J., Duque H., Pierson J.M., Breton V., Brunie L., and Magnin I.E., “Medical Image Content-Based Queries using the Grid”. *Proceedings of the first European HealthGrid conference*, pp 142-151, Lyon, France, January 2003.

4. Montagnat J., Breton V., Magnin I.E., “Using grid technologies to face medical image analysis challenges”, *Biogrid'03, proceedings of the IEEE CCGrid03*, pp 588-593, May 2003, Tokyo, Japan.

5. Claerhout B. and De Moor G., “From GRID to HealthGRID: introducing Privacy Protection”, *Proceedings of the first European HealthGrid conference*, pp 152-162, Lyon, France, January 2003.

6. Seitz L., Pierson J.M., and Brunie L., “Key management for encrypted data storage in distributed systems”. *Second International IEEE Security in Storage Workshop* (SISW). Washington DC, USA, October 2003.

7. Penney G.J., Weese J., Little J.A., Desmedt P., Hill D.L.G., and Hawkes D.J., “A comparison of Similarity Measures for Use in 2D-3D Medical Image Registration”, In *Medical Image Computing and Computer-Assisted Intervention* (MICCAI), volume 1496 of LNCS, pp 1153-1161, Cambridge, USA, October 1998. Springer.

8. Roche A., Malandain G., Pennec X. and Ayache N., “The Correlation Ratio as a New Measure for Multimodal Image Registration”. In *Medical Image Computing and Computer-Assisted Intervention* (MICCAI), volume 1496 of LNCS, pp 1153-1161, Cambridge, USA, October 1998. Springer.

9. EDG. European DataGrid IST project. http://www.edg.org/.

10. Caron E., Desprez F., Lombard F., Nicod J.-M., Quinson M., and Suter F., “A Scalable Approach to Network Enabled Servers”. Proceedings of the 8th International EuroPar Conference, volume 2400 of LNCS, Paderborn, Germany, pp 907-910, August 2002. Springer-Verlag.

11. OpenSSL. Secured Socket Layer. http://www.openssl.org/.

12. MySQL. SQL database. http://www.mysql.org/.

Figure 1. Content-based query on medical images

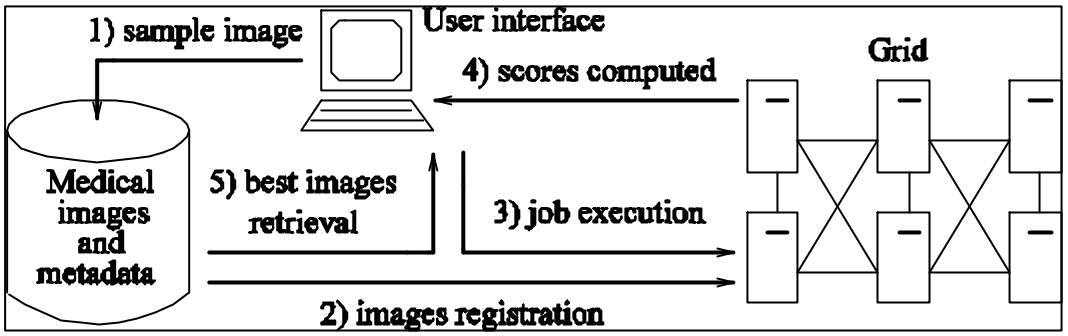

Figure 2. From up to down and left to right: a sample image and ranked candidate images with high and low similarity scores

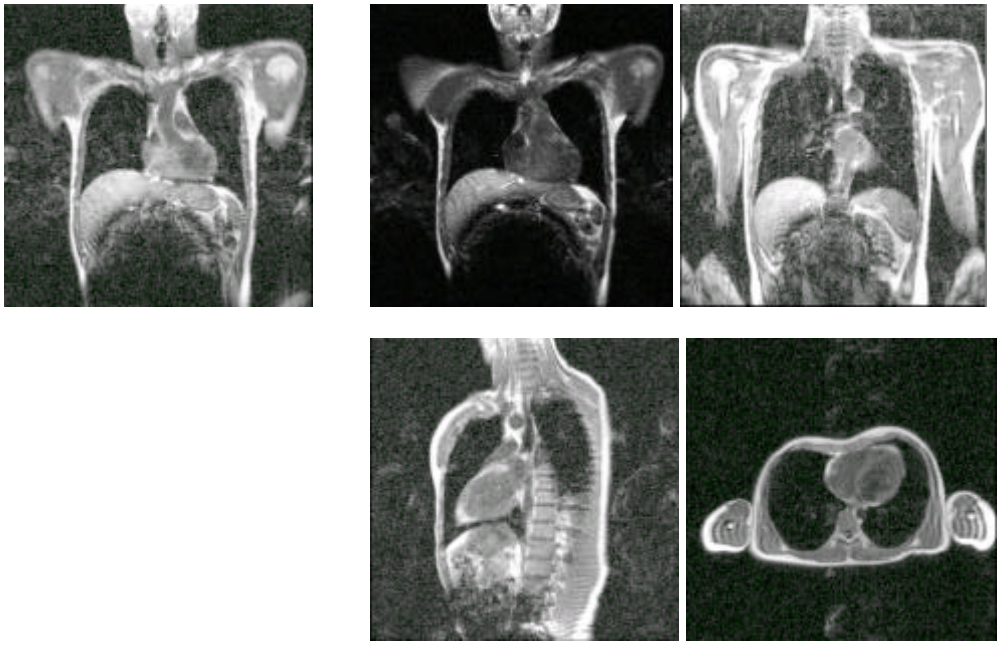

Figure 3. Time (in seconds) needed to answer a query depending on the application granularity

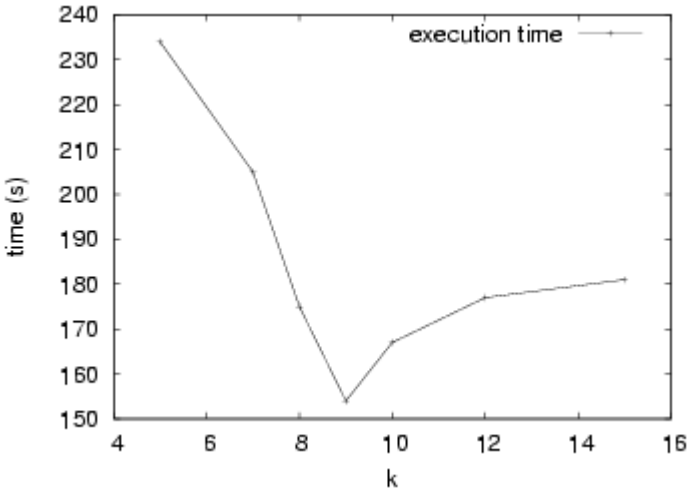

Figure 4. Left: replication time, right: scheduling time.

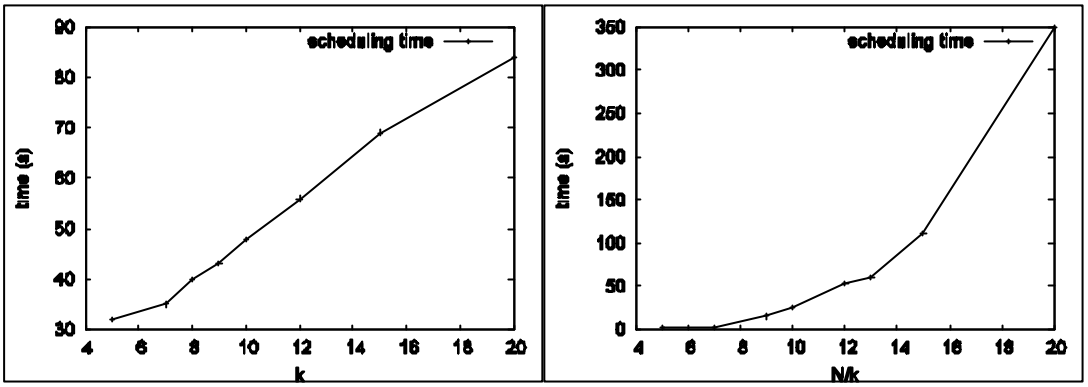

Figure 5. Estimated computation time versus measured computation time

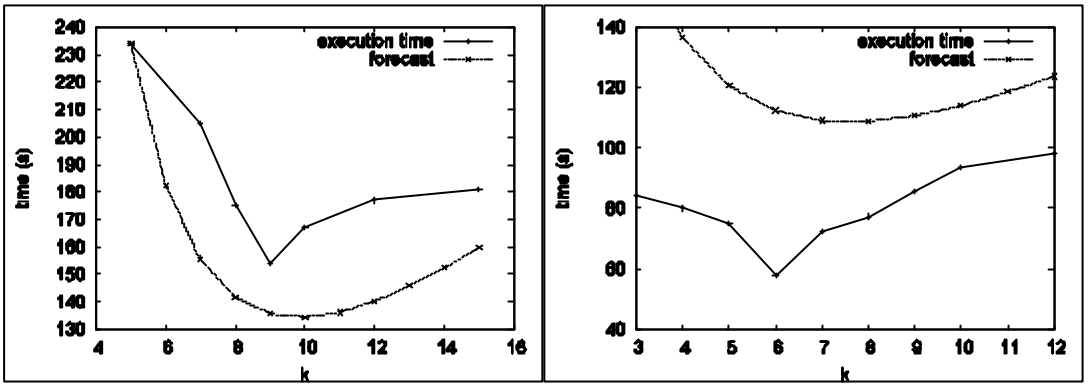

Table 1. Test images size and gray level ranges.

	Set 1	Set 2	Set 3
Number of images	124	238	456
Dimensions	256x256	181x217x181	181x217x181
Size (n)	65536	7109137	7109137
Precision (r)	2^9	2^{16}	2^{12}

Table 2. Computation times (in seconds) for joint histogram and similarity measure computation.

n	r	Joint histogram	SD	SSD	CC	RC	Woods	MI
Set 1	2^8	0.030	0.016	0.020	0.033	0.030	0.031	0.040
65536	2^9	0.063	0.041	0.047	0.076	0.070	0.071	0.087
Set 2	2^8	0.778	0.017	0.020	0.032	0.030	0.032	0.035
7109137	2^{12}	7.897	5.895	6.562	8.277	8.150	7.972	8.029
	2^{16}	693.0	4.308	4.405	7.321	9.644	9.709	11.27
Set 3	2^8	1.021	0.017	0.025	0.041	0.036	0.037	0.071
7109137	2^{12}	8.943	5.031	5.600	9.058	8.677	8.670	9.946

Table 3. Theoretical computations times (in seconds) computed from 4 to 11 with c = 20ns.

n	r	Join histogram	SD	SSD	CC	RC	Woods	MI
Set 1	2^8	0.038	0.020	0.020	0.041	0.037	0.037	0.050
65536	2^9	0.073	0.051	0.051	0.105	0.095	0.095	0.129
Set 2	2^8	1.728	0.020	0.020	0.041	0.037	0.037	0.050
7109137	2^{12}	7.410	5.033	5.033	10.40	9.395	9.395	12.75
	2^{16}	716.0	5.033	5.033	10.40	9.395	9.395	12.75
Set 3	2^8	1.728	0.020	0.020	0.041	0.037	0.037	0.050
7109137	2^{12}	7.410	5.033	5.033	10.40	9.395	9.395	12.75