An Easy Setup for Parallel Medical Image Processing: Using Taverna and ARC

Xin ZHOU [a,1], Hajo KRABBENHÖFT [b], Marko NIINIMÄKI [a], Adrien DEPEURSINGE [a], Steffen MÖLLER [b] and Henning MÜLLER [a,c]

[a] *Medical Informatics, Geneva University Hospitals and University of Geneva, Switzerland*

[b] *Institute for Neuro– and Bioinformatics, University of Lübeck, Germany*

[c] *Business Information Systems, University of Applied Sciences Western Switzerland, Sierre, Switzerland*

Abstract. Medical image processing is known as a computationally expensive and data intensive domain. It is thus well–suited for Grid computing. However, Grid computing usually requires the applications to be designed for parallel processing, which is a challenge for medical imaging researchers in hospitals that are most often not used to this. Making parallel programming methods easier to apply can promote Grid technologies in clinical environments. Readily available, functional tools with an intuitive interface are required to really promote health grids. Moreover, the tools need to be well integrated with the Grid infrastructure.

To facilitate the adoption of Grids in the Geneva University Hospitals we have set up a develop environment based on the Taverna workflow engine. Its usage with a medical imaging application on the hospital's Grid cluster is presented in this paper.

Keywords. Grid networks, Taverna, medical imaging research, image processing, ARC

1. Introduction

Images are produced in ever–increasing quantities in hospitals. Their automatic analysis and processing is not possible with current computing infrastructures, particularly as few hospitals possess a research computing infrastructure. Parallelization of image processing can help the processing tasks utilise massively parallel infrastructures such as Grid networks [1]. A Grid cluster can consist of a large number of desktop computers that often exist in abundance in hospitals. However, parallel programming environments are not trivial for the medical imaging community that generally focuses on the application itself. A parallel execution infrastructure has to be set up and maintained and the programmers need to have experience in developing parallel applications, which is only rarely the case.

The infrastructure set up questions have been addresses by Grid and batch processing environments (also known as cluster computing systems or Local Resource Manage-

[1] Corresponding Author: Medical Informatics, Geneva University Hospitals and University of Geneva, 24 Rue Micheli–du–Crest, 1211 Geneva 14, Switzerland, xin.zhou@sim.hcuge.ch

ment Systems, LRMS), such as BOINC [2] (Berkeley Open Infrastructure for Network Computing), Condor [3], and others. Remote resources can be allocated through Grid middlewares, for example ARC[2] (Advanced Resource Connector [4]) and gLite [5]. Previous work of the University Hospitals of Geneva includes a simple–to–install local hospital Grid based on the Condor LRMS and the *ARC* middleware inside the Geneva University Hospitals. The idea was to use the computing resources of desktop PCs inside the hospitals using virtualization technologies [6] to clearly separate the computation part of desktops and the part potentially treating patient data. Another approach under development is described in [7]. It contains a pull–based distributed Grid system that does not require the working nodes to have own IP (Internet Protocol) addresses.

This paper focuses on intuitive ways for creating parallel medical imaging applications to reduce the difficulties often due to the lack of parallel programming expertise. Medical imaging research could profit from a layer which masking the difficulties of parallelization. A more classical approach is to use well known libraries, such as *MPI* (Message Passing Interface)[3], *PVM* (Parallel Virtual Machine)[4] or *openMP* (Open Multi–Processing)[5]. Early work in the imaging community has adapted these libraries to perform a low–level parallelization for image processing, for example described in [8]. Despite some efforts these solutions were not widely spread in the imaging community for two reasons: (1) using these approaches was still time–consuming and error prone for image processing developers. A parallelization tool for the image processing community would be acceptable only if it hides all parallelism from the application programmer and suggesting a fully sequential programming model [9]; (2) many proposed solutions were bound to either specific applications or very specific infrastructures.

Using workflow engines such as Taverna for scientific parallel programming has become a new trend in several domains [10]. For medical imaging development, important properties for develop environments are listed in [11]. Several workflow management engines (also called component–based application builders) are compared in this text. Currently, the most widespread workflow engine in the bio–informatics community is probably Taverna[6] [12].

Several benefits of using Taverna can be stated directly:

- It provides a graphical user interface presenting an intuitive overview of parallel task execution.
- The workflow elements are designed independently of applications and are adaptable for different high performance computing (HPC) infrastructures, increasing the potential reusablility of workflows.
- By decomposing the task into workflow elements, both data–decomposition and task–decomposition are possible. This decomposition gives a clear separation of a "sequential" mode and a "parallel" mode. Every workflow element is developed as a sequential program and tested in a sequential environment. The parallelization parameters are only configured in a workflow map.

[2] http://www.knowarc.eu/
[3] http://www-unix.mcs.anl.gov/mpi/
[4] http://www.csm.ornl.gov/pvm/
[5] http://openmp.org/
[6] http://taverna.sourceforge.net/

- A well built workflow can be imported as a workflow element for another workflow, which allows multi–level workflow composition to generate hierarchical programs. This simplifies both program design and development.
- Ready–to–use workflow maps are shared by a large number of Taverna users on the web side MyExperiment.org [7].

Using Taverna for biomedical image processing was already proposed in [13]. In that article, the author proposed to combine Taverna with the EGEE[8] (Enabling Grids in E-Science in Europe) remote grid resources. However, medical images are often subject to legal question preventing them from being sent outside of the hospital for computation. Our application development uses a recent extension (plug–in) [14] of Taverna that allows Taverna workflow to be executed on ARC clusters. This enables us to execute the workflow in the hospital's local ARC Grid as described in [6]. Moreover, since Taverna runs on Windows, the workflow software development can be performed completely on Windows even if all the Grid software is Unix–based. This has undeniable advantages as the entire hospital infrastructure is Windows–based and thus most researchers equally use Windows–based development tools.

The rest of the paper is organized as follows: In Section 2 we describe the methods, thus the basic elements and data we use in this paper. Section 3 presents the results, and thus the main novel parts of our approach. We finalize the article with a discussion and conclusions in Section 5.

2. Methods

In this section the setup for the development is explained. Computing resources and develop environments are presented in details.

2.1. Computing resources available in the Geneva University Hospitals

The Geneva University Hospital like many medical institutions does not have a dedicated research computing infrastructure. On the other hand, a very large number of desktop computers is available that is renewed completely every four to five years. Currently, around 6000 computers are available from a variety of generations, with the slowest one currently containing 1 GB of main memory and a Pentium IV with 2.8 GHz. By mid–2009 a majority of 5'000 PCs will have 2 GB of main memory and dual core CPUs, where virtualisation is foreseen.

For testing purposes 20 old PCs were made available for us to create an inner–hospital Grid (768M RAM and 2.8GHz CPU) A very compact Linux operating system is installed on these PCs in a VMWare Virtual Machine to use them as computing nodes. 50% of CPU time, and 350MB of memory are allocated for the for the virtual machine. The mini-cluster runs Condor on the nodes and ARC as LRMS. 5 other PCs of different generations were also installed in a similar setup to test the influence that such a virtual machine can have on a desktop user when frequently being used for computing image processing jobs.

[7] http://www.myexperiment.org/
[8] http://www.eu-egee.org

2.2. Existing development environment of imaging researchers

Imaging researchers generally use workstations with at least 1GB memory. The most frequently used programming language is Java, for its platform independence and large number of already available tools. Parallel programming has been only rarely used, as most developers do not have much knowledge of parallel environments and using Grid systems is not always an easy task.

2.3. Parallel development environment

The setup for parallel application development proposed in this paper is composed of the following parts:

- the Taverna workflow engine;
- a plug–in provided by the KnowARC project that can be installed using the Taverna plug–in manager and that connects taverna and an ARC resource;
- an environment adapted to the ARC–based Grid of virtual machines mentioned in Section 2.1 to test each component before putting it into the Taverna workflow (using thus the virtual machines with 350 MB of RAM each);
- a web server to upload and share built components and workflow maps.

3. Results

This section presents the main results of this article, consisting of different parts.

3.1. Design approach for our Taverna–based parallelization of medical imaging

The fundamental concepts of Taverna–based application development is decomposition of software systems into components. Two types of decomposition need to be separated: component decomposition (also called task decomposition, concerning the code parts) and data decomposition.

Component decomposition decomposes the application into elementary units, which can be seen as a black box between defined input data and output data. The way to decompose the application often follows these steps:

- decomposition of the application into conceptually independent components for easier reusability of these building blocks;
- decomposition of large but independent components into smaller components, wherever possible, creating check pointing data that can be used if the task needs to be restarted;
- decomposition of each component by separating the input and output parts to facilitate the data structure modification;
- finally, a decomposition of each obtained component into objects with a log strategy to enable debugging and error tracking procedures.

Data decomposition is easy to understand and is related to the decomposed components. A component executed with a part of the data forms a task. Data decomposition has the following goals :

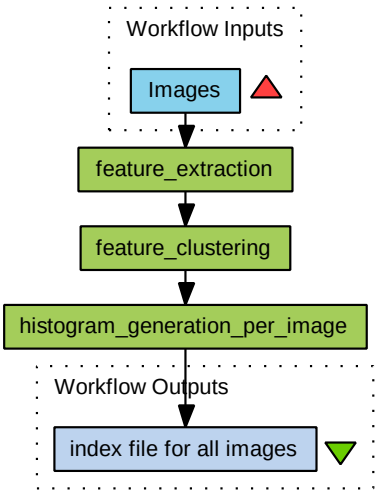

Figure 1. Conceptual decomposition of a medical imaging workflow for feature extraction into basic components.

- tailoring of the intermediary data storage and transition for each task;
- tailoring of the memory usage for each task;
- tailoring of the execution time for each task.

In order to avoid bad parameter selection, the runtime environment (in our case the virtual machine with 350M RAM) is simulated in develop environment (in our case Eclipse). Thus "tailoring" step is performed under develop environment.

3.2. An example parallelization using Taverna and ARC

To try our approach with a concrete application, a content–based image retrieval (CBIR) framework [15], named Parallelized Medical Image Retrieval (ParaMedIR) is developed based on this setup. The data and computationally most intensive part of CBIR is to index databases of images using visual features. From each image, key visual information called features are extracted. This information can be simple color or texture features. In the approach, newly developed for this paper, features are divided into groups called visual key words [16]. First of all, image regions are determined (through a fixed grid, randomly or through interest point detectors), and in neighborhoods of these regions visual features are extracted. The extracted visual features are then clustered into a limited number of similar visual key words. Statistical information on the appearance of those visual key words in images (normally in the form of histograms) is subsequently analyzed for each image to measure the similarity pairs of images. The entire process can be called indexing in this context. It can conceptually be decomposed into three main components: extraction of features from the images, clustering of the features into visual key words, and generation of the histograms that are then stored in a database or other indexing structure.

The initial workflow decomposition for this process is shown in Figure 1. Each conceptually separated component can still be computationally expensive and might re-

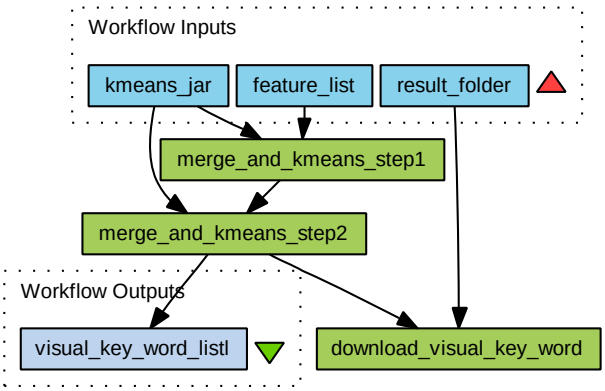

Figure 2. A second decomposition step further details the components of the processing chain.

quire further decomposition. The second decomposition step specifies the check points for large components in the process. In our application, the feature clustering is a good example for such a large component. KMeans clustering is a classical clustering algorithm that we use in this context, which is known to scale badly for very large data sets. A staged parallel execution model [17] was selected to perform the clustering in two steps. Intermediary results such as the collection of features, the cluster centers, and the feature–cluster maps need to be stored. This further decomposition can be seen in Figure 2. Unifying the Input/Output (IO) modules and standardizing the data structures are important requirements for the development as well. A typical design is to generate an IO package of classes for all components. The IO package has to take into account the data parallelization requirements. Data separation should be parameterizable meaning that various file formats should be checked and adapted. Another useful package is the group of classes for log file creation. Some general information, such as time measurement, completeness of calculation percent, error messages are often useful to include. In order to generate IO and log file packages, IO operation and log creation need to be isolated from the original components in the workflow maps. Following this approach, further components are added. Figure 3 is still based on the feature clustering components but adds further components.

3.3. Execution of the developed workflow with Taverna

The execution of the application is started via Taverna and can use a variety of available resources know to Taverna. After workflow design, the researchers need to develop sequential executables for each workflow element and link them to the finished workflow map. Most often such components already exist anyways, as many research tools are simple scripts or components that are combined. Then, the parallel execution needs to be configured for each workflow element. The Taverna user interface is shown in Figure 4. The upper left hand side (Part 1) contains the list of available components of the workflow. The lower left hand side (Part 2) is dedicated to the parallel execution configuration. The right hand side (Part 3), finally, is the workflow window itself. Inside this window the final workflow map is shown.

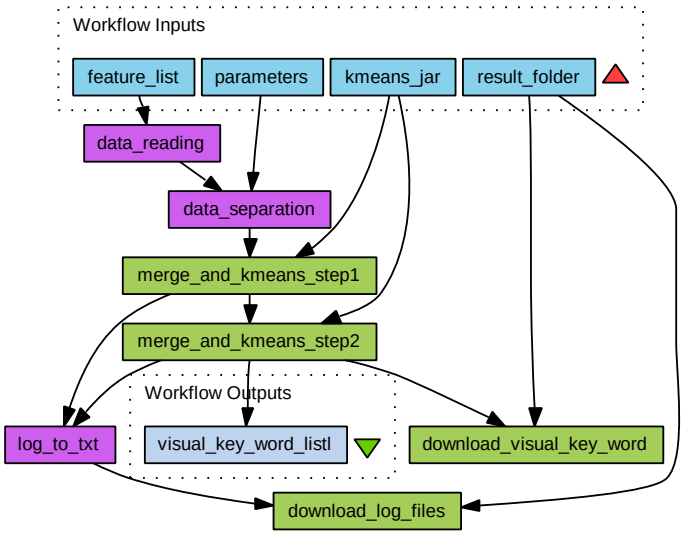

Figure 3. The final decomposition of the workflow including log files and debugging information.

Table 1. Computation times on sever and on local ARC grid.

workflow element	Sequential execution t_s	Parallel execution t_p	Speed–up factor f
feature extraction	291 mins	45 mins	6.5
KMeans clustering	480 mins	30 mins	16 (different algorithm)
histogram generation	61 mins	27 mins	2.2

Table 1 presents the parallel execution time t_p analysis. To compare with sequential execution time t_s, a metric called speed–up factor f is used. The metric, $f = t_s/t_p$ shows how much the execution was accelerated.

It should be mentioned that the KMeans clustering algorithm used is not the same for the sequential and for the parallel execution. Sequential execution takes all the features into account whereas the staged parallel mode divides the features into small groups, and performs the clustering hierarchically. The execution time of clustering is highly related to the quantity of data, which explains the high value of f obtained. The histogram generation requires frequent reading and writing operations and therefore it benefits less from the parallel execution.

4. Conclusions and future work

Hospitals are producers of extremely large amounts of digital information, including images. Computational solutions such as Grids are necessary if these data are attempted to be treated or analyzed to improve the care processes and provide diagnosis aid. Currently the clear tendency for high performance computing is towards parallel processing using standard components. The availability of multi–core CPUs is also accelerating this pro-

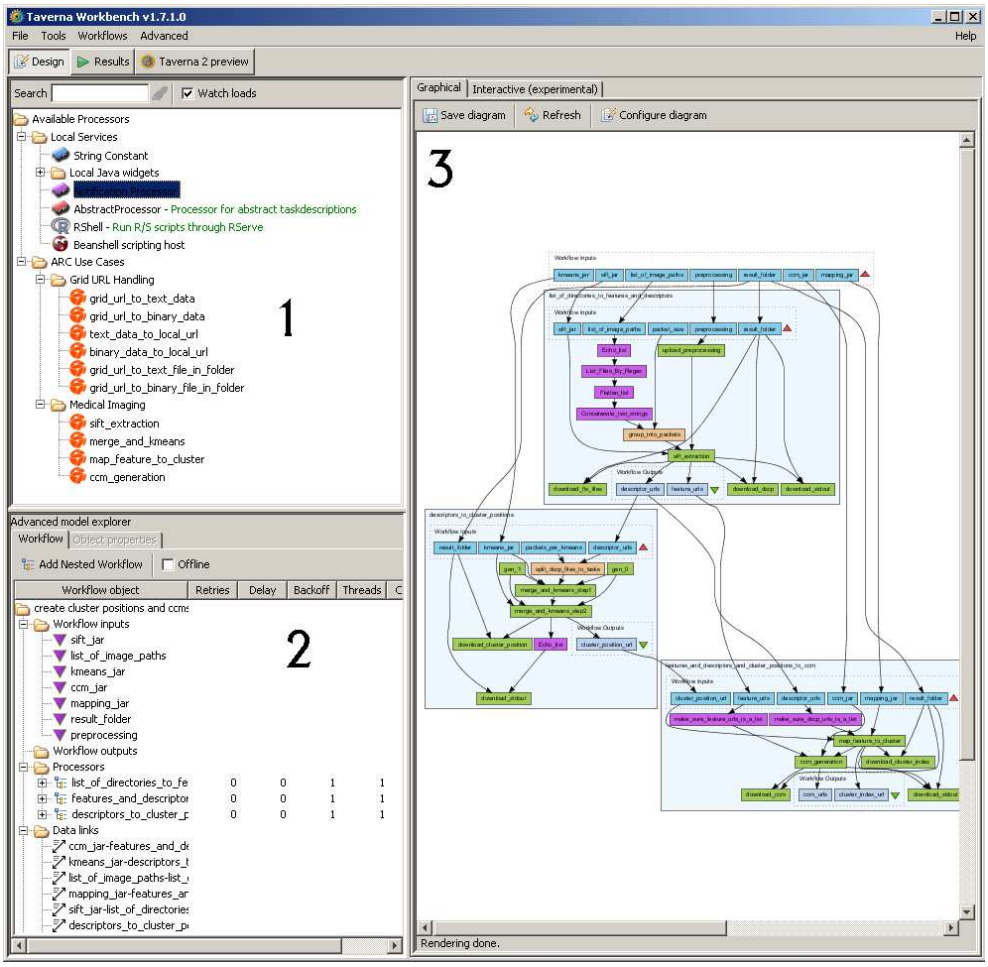

Figure 4. The user interface of Taverna contains three distinctive parts and allows for an easy configuration of resources and workflow for new applications

cess. Most researchers in hospitals have only little experience with parallel development environments and thus only rarely profit from the potentially large computing power that Grid networks can offer.

In the University Hospitals of Geneva, a small test Grid using ARC and Condor was implemented and using this test bed several applications were griddified. It became quickly apparent that griddification is non–trivial for researchers who usually only have little time and avoid the overhead to learn new technologies. Taverna–based griddification was significantly easier and besides the application described in this paper, several other applications were adapted for Grid use, usually taking have a day for all steps. This encouraged other people to use the small test Grid in the hospitals and could be a general model for the adoption of Grid technologies in environments such as a hospital.

Several further developments are still foreseen in this context, particularly when connection such applications with other components of the web–services–based patient

record.

Acknowledgements

This work was partially supported by the EU 6th Framework Program in the context of the KnowARC project (IST 032691) and by the Swiss National Science Foundation (FNS) in the context of the Talisman–2 project (project 200020 118638).

References

[1] Müller, H., Pitkanen, M., Zhou, X., Depeursinge, A., Iavindrasana, J., Geissbuhler, A.: Knowarc: Enabling Grid networks for the biomedical research community. In: Healthgrid 2007, Geneva, Switzerland (2007) 261–268

[2] Anderson, D.P.: BOINC: A System for Public-Resource Computing and Storage. In: Fifth IEEE/ACM International Workshop on Grid Computing (GRID'04). (2004)

[3] Litzkov, M., Livny, M., Mutka, M.: Condor — a hunter of idle workstations. In: Proceedings of the 8th international conference on distributed computing, San Jose, California, USA (1988) 104–111

[4] Ellert, M., Grønager, M., Konstantinov, A., Kónya, B., Lindemann, J., Livenson, I., Langgaard Nielsen, J., Niinimäki, M., Smirnova, O., Wäänänen, A.: Advanced resource connector middleware for lightweight computational Grids. Future Generation computer systems **23** (2007) 219–240

[5] Laure, E., Fisher, S.M., Frohner, A., Grandi, C., Kunszt, P.Z., Krenek, A., Mulmo, O., Pacini, F., Prelz, F., White, J., Barroso, M., Buncic, P., Hemmer, F., Di Meglio, A., Edlund, A.: Programming the grid using glite. COMPUTATIONAL METHODS IN SCIENCE AND TECHNOLOGY **12** (2006) 33–45

[6] Niinimäki, M., Zhou, X., Depeursinge, A., Geissbuhler, A., Müller, H.: Building a community grid for medical image analysis inside a hospital, a case study. In Olabarriaga, S.D., Lingrand, D., Montagnat, J., eds.: Medical imaging on grids: achievements and perspectives (Grid Workshop at MICCAI 2008), New York, USA (2008) 3–12

[7] Orellana, F., Niinimaki, M., Zhou, X., Rosendahl, P., Müller, H., Waananen, A.: Image analysis on gridfactory desktop grid. In: 4th International Conference on Grid and Pervasive Computing (GPC'09), Geneva, Switzerland (2009 – submitted)

[8] Squyres, J.M., Lumsdaine, A., Stevenson, R.L.: A toolkit for parallel image processing. In: Proceedings of the SPIE Conference on Parallel and Distributed Methods for Image processing, San Diego, CA (1998) 69–80

[9] Seinstra, F.J., Koelma, D.: User transparency: a fully sequential programming model for efficient data parallel image processing: Research articles. Concurr. Comput. : Pract. Exper. **16** (2004) 611–644

[10] Zhao, Y., Raicu, I., Foster, I.: Scientific workflow systems for 21st century, new bottle or new wine? In: IEEE International Conference on Services Computing (SCC) 2008, Honolulu, Hawaii (2008) 467–471

[11] Olabarriaga, S.D., Snel, J.G., Botha, C.P., Belleman, R.G.: Integrated support for medical image analysis methods: from development to clinical application. IEEE Trans Inf Technol Biomed **11** (2007) 47–57 (Special Issue on IMAGE MANAGEMENT IN HEALTHCARE ENTREPRISES).

[12] Oinn, T., Addis, M., Ferris, J., Marvin, D., Senger, M., Greenwood, M., Carver, T., Glover, K., Pocock, M.R., Wipat, A., Li, P.: Taverna: a tool for the composition and enactment of bioinformatics workflows. Bioinformatics **20** (2004) 3045–3054

[13] Glatard, T., Montagnat, J., Pennec, X.: Grid–enabled workflows for data intensive medical applications. In: 18th IEEE Symposium on Computer–Based Medical Systems, Dublin, Ireland (2005) 537–542

[14] Krabbenhöft, H.N.N., Möller, S., Bayer, D.: Integrating ARC Grid middleware with taverna workflows. Bioinformatics **24** (2008) 1221–1222

[15] Müller, H., Michoux, N., Bandon, D., Geissbuhler, A.: A review of content–based image retrieval systems in medicine – clinical benefits and future directions. International Journal of Medical Informatics **73** (2004) 1–23

[16] Moghaddam, B., Biermann, H., Margaritis, D.: (Defining image content with multiple regions of interest) 89–93

[17] Kraj, P., Sharma, A., Garge, N., Podolsky, R., McIndoe, R.: Parakmeans: Implementation of a parallelized k-means algorithm suitable for general laboratory use. BMC Bioinformatics **9** (2008) 200