Virtual EZ Grid: A Volunteer Computing infrastructure for scientific medical applications

Mohamed Ben Belgacem[1], Nabil Abdennadher[2], Marko Niinimaki[2]

[1]University of Geneva, Switzerland
`mohamed.benbelgacem@unige.ch`
[2]University of Applied Sciences Western Switzerland, hepia Geneva, Switzerland
`{nabil.abdennadher, markopekka.niinimaeki}@hesge.ch`

Abstract. This paper presents the *Virtual EZ Grid* project, based on the XtremWeb-CH (XWCH) volunteer computing platform. The goal of the project is to introduce a flexible distributed computing system, with (i) a non-trivial number of computing resources infrastructure from various institutes, (ii) a stable platform that manages these computing resources and provides advanced interfaces for applications, and (iii) a set of applications that take benefit of the platform. This paper concentrates on the application support of the new version of *XWCH*, and describes how a medical image application MedGIFT utilises it.

1 Introduction

Nowadays, volunteer computing (VoC) and grid computing are a well-established paradigm of distributed computing. The term "volunteer computing" is used for all scenarios where a low priority guest application can run on unused remote resources without significantly impacting high priority host applications. In volunteer computing individuals donate unused or idle resources of their computers to distributed high performance computing applications.

On the other hand, Grid computing is the combination of computer resources from multiple administrative domains applied to a common application that requires a great number of computer processing cycles or the need to process large amounts of data.

There are several characteristics that distinguish the volunteer computing from Grid [1]:

- The number of volunteer nodes in VoC systems may range from less than 10 to hundreds of thousands.
- Volunteered resources are owned and managed by regular people, not by IT professionals
- Volunteers are anonymous, and those who misbehave cannot be fired or prosecuted

- Volunteered resources are often behind network firewalls that do not allow incoming connections
- Volunteer computing is asymmetric: volunteers supply resources, and not the other way round.

Grid and VoC platforms are organised with the help of middleware systems.

The most known grid systems are gLite [2], ARC [3], Globus [4], Unicore [5], Condor [6] and GridMP [7].

Berkeley Open Infrastructure for Network Computing (BOINC) [8] is the most widely used middleware in volunteer computing. XtremWeb (*XW*) [9] is a VoC middleware providing a framework and a set of tools to assist in the creation of volunteer computing projects.

XtremWeb-CH (*XWCH:* www.xtremwebch.net) [10], developed by the authors of this paper improves *XW* through the usage of peer-to-peer concepts. *XWCH* is an upgraded version of (*XW*). Major improvements have been brought to it in order to obtain a reliable and efficient system. Its software architecture was completely re-designed. The first version of XtremWeb-CH (*XWCH1*) is composed of three kinds of peers: the coordinator, the workers and the warehouses. Several applications have been deployed on *XWCH1* [11]. [12] details the limits of the *XWCH1* middleware. To overcome these drawbacks, a second version (*XWCH2*) was developed. This version is currently being used to deploy several desktop grid and VoC infrastructures such as *Virtual EZ Grid* [13] and *From Peer-to-Peer (From-P2P)* [14]. One of the main objectives of these two projects is to deploy scientific medical applications. Three applications are being gridified within these projects, but for the sake of brevity we only discuss one of them.
This paper is organised as follow: the next section presents the new features of *XWCH2*. Section 3 details the architecture of the *Virtual EZ Grid* infrastructure while section 4 presents one example of a medical application deployed on the *Virtual EZ Grid* platform: *MedGIFT*. Finally, section 5 gives some perspectives of this research

2 XtremWeb-CH2 (XWCH2)

The new version of *XWCH* features improved support for parallel distributed applications. The extensions carried out are application driven. They were deduced from experiments carried out and lessons learned during the gridification and the deployment of several applications [15]. In more detail, the main improvements of *XWCH2* can be summarized as: dynamic task generation, flexible data sharing (data replication) and persistent tasks.

This paper will only detail the "dynamic task generation" aspect.

We shall also show improvements in the way the user communicates with the system, through its application programming interface (API) in section 2.2.

2.1. The *XWCH2* architecture

Figure 1 illustrates the principal changes in the *XWCH2* architecture. Job submission is done by a flexible API, available for Java and C/C++ programs. The interfaces of the coordinator now contain user service and worker services, both of which are web services, implemented using WSDL [16].

Like in the earlier version of *XWCH*, the basic architecture of *XWCH2* consists of a coordinator, a set of worker nodes and at least one warehouse node [10]. However, contrarily to *XWCH1*, jobs are submitted to the coordinator by a "client node" which executes a client program that calls the services provided by *XWCH2*. The coordinator schedules jobs and pre-assign them to the workers. The workers retrieve the executable files and input files from warehouses (or other workers), compute the jobs, and store their outputs in the warehouses. The coordinator and at least one of the warehouses are assumed to be available to the workers involved in the execution of jobs created by the same client program. Communication between the coordinator and the workers is always initiated by the workers (*Work request, Work Alive* and *Work Result* in Figure 1); thus workers can receive tasks and send results even if they are run in "out-bound connectivity only" environments, like NAT sub-networks.

Since *XWCH2* can be used as a volunteer computing platform, it is not reasonable to assume that any two workers can always communicate directly with each other. Organizations have firewalls and NAT (Network Address Translation) sub-networks that protect the organization's network by limiting connectivity to/from the outside world. In *XWCH*, the workers can communicate directly with each other whenever it is possible, otherwise through a warehouse.

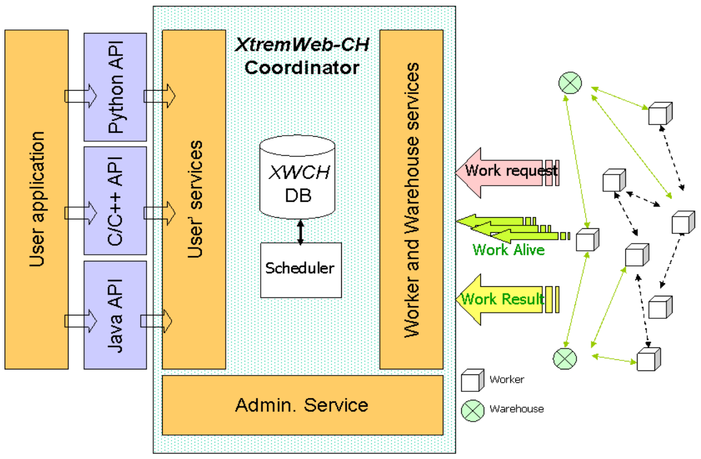

Figure 1: *XWCH2* architecture

An *XWCH* application is composed of a set of communicating jobs and can be represented by a workflow. The number of jobs and the structure of the workflow cannot be known in advance. Jobs are created by the client program by calling a specific service of *XWCH2*.

2.2 Dynamic tasks generation

In what follows, we give a brief summary of the Java API functions (Table 1) that allows user to create *XWCH* jobs according to his/her needs. The entire API documentation is available at the *XWCH* web site www.xtremwebch.net.

XWCHClient (java.lang.String *serverendpoint*, java.lang.String *datafolder*, java.lang.String *clientID*)	This method creates a "connection" with the coordinator. *Serverendpoint* refers to the URL of the user services in Figure 1. *Datafolder* is a local folder (client node) where the binaries and input files exist.
AddApplication (java.lang.String *app_name*)	This method adds an application to the coordinator. It returns an *application_id.*
AddModule (java.lang.String *module_name*)*	This method adds a "module" to the coordinator and returns a *module_id*. A module is as set of binary codes having, in general, the same source code. Each binary code targets a specific (OS, CPU) platform.
AddModuleApplication (java.lang.String *module_name*, java.lang.String *binaryzip*, PlatformType)	Adds an executable binary file to a given module. This is "per platform" basis, i.e. different binary files can be added for each of the platform (MS Windows, MacOS, Linux, Solaris, etc.).
AddData (java.lang.String *app_name*)	Adds an input file to the application *app_name.* This method is often used to upload the input data of one application (one execution) into the *XWCH* system.
AddJob (java.lang.String *jobname*, java.lang.String *app_name*, java.lang.String *module_name*, java.lang.String *command_line*, java.lang.String *inputfiles*, java.lang.String *outfilespec*, java.lang.String *outfilename*, java.lang.String *flags*)	This method submits a job to the coordinator. A job ID is returned. *app_name* and *module_name* refer to the application and the module to which the job belongs. *command_line* is the command that invokes the binary with parameters (in the worker). *inputfiles* represent the set of input files of the given job. o*utfilename* refers to a name that will be given to the compressed output file. By "flags" the programmer can pass specific distribution related options to *XWCH2* (replicate output data on several warehouses, execute a set of tasks on the same worker, execute one task on a given worker, etc.).
GetJobStatus (java.lang.String *Job_ID*)	Gets the status of the job.

GetJobFileOutName(java.lang. String *Job_ID*)	Gives the “reference” (identifier) of the Job's output file.
GetJobResult (java.lang.String *Job_ID*, java.lang.String *outfilename*)	Gets the output file (output data) of a given job.

Table 1: *XWCH2* Java API

An example of using the API to create and execute three communicating jobs: *Job1, Job2* and *Job3*. *Job2* and *Job3* are using the output of *Job1* as input data.

```
// Initialisation of the connection
c = new XWCHClient(ServerAddress, ".", IdClient,1,9);
c.init();

String appid = c.AddApplication("Hello World application");
String ModuleId1 = c.AddModule("Module1");

String refWind = c.AddModuleApplication (ModuleId1,
BinaryPath_Module1_win, PlateformEnumType.WINDOWS); //Windows binary
String ModuleId2 = c.AddModule("Module2");
String refProcesswindows = c.AddModuleApplication(ModuleId2,
BinaryPath_Module2_win, PlateformEnumType.LINUX);   //Linux binary
. . .

String job0 = c.AddJob        ("First Job",          //Job description
                              appid,                 //Application ID
                              ModuleId1,             // Module identifier
                              CmdLine_for_job0,      //Command line
                              frefjob0.toJobReference(),
                              liste_files_out_job0,
                              file_out_id_job0,
                              "" );

//Wait until job0 ends
String status = "";
while (!status.equalsIgnoreCase("COMPLETE")) status =
                              c.GetJobStatus(job0).toString();


// Retrieve the reference of the output file of job0
String inputforJobs_1_and_2 = c.GetJobFileOutName(job0);
String job1 = c.AddJob        ("second Job",        //Job description
                              appid,                //Application ID
                              ModuleId2,            // Module identifier
                              CmdLine_for_job1,     //Command line
                              inputforJobs_1_and_2,
                              liste_files_out_job1,
                              file_out_id_job1,
                              "");

String job2 = c.AddJob        ("Third Job",         //Job description
                              appid,                //Application
                              identifier
                              ModuleId2,            // Module identifier
                              CmdLine_for_job2,     //Command line
                              inputforJobs_1_and_2,
                              liste_files_out_job2,
                              file_out_id_job2,
                              "");
//Wait until job1 and job2 end, by using "GetJobStatus" method (table 1)
String status = "";
while (!status.equalsIgnoreCase("COMPLETE")) status =
                              c.GetJobStatus(job1).toString();
status = "";
while (!status.equalsIgnoreCase("COMPLETE")) status =
                              c.GetJobStatus(job2).toString();

GetJobResult (job1, file_out_id_job1);
GetJobResult (job2, file_out_id_job2);
```

This client program does not show the different features supported by *XWCH2*. Nevertheless, it details how *XWCH2* handles communication and precedence rules between jobs. Although this program does not show it, calls to *XWCH2* services can take place in loops and tests controls. This means that the number of jobs and the structure of the graph representing the application are not known in advance.

3 The Virtual EZ Grid project

This section presents a desktop Grid infrastructure called *Virtual EZ Grid* (http://www.xtremwebch.net/Projects/Virtual_EZ_Grid/EZ_Home.html). This platform, based on *XWCH2* middleware, is mainly used to deploy and execute three scientific medical applications. This section presents the architecture of the *Virtual EZ Grid* project while section 4 presents only one application: *MedGIFT*.

3.1 *Virtual EZ Grid* in brief

The aim of *Virtual EZ Grid* is to establish a sustainable desktop Grid platform across several institutions. Three main goals are targeted by the project:

1. Construct a desktop grid infrastructure with non dedicated desktop PCs to provide harvested CPU power for scientific research projects.
2. Implement a reliable platform by using virtual environments to support secure computing and remote check-pointing. Virtual EZ Grid aims at providing a better control over environmental issues and energy consumption by running only the necessary PCs and shutting down unused PCs at night and during holidays. The proposed platform should give a non-intrusive experience to the PC users.
3. Evaluate the two first objectives in a real world setting with the several medical applications.

The *Virtual EZ Grid* architecture is shown in Figure 3.

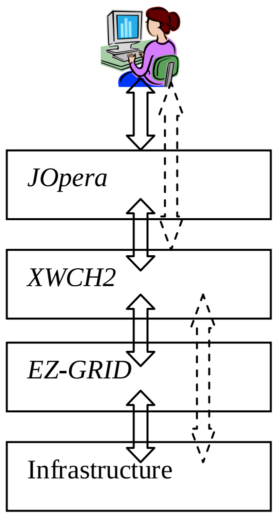

Figure 3: The *Virtual EZ Grid* architecture

The three main supported tools of *Virtual EZ Grid* are: *XWCH2*, *JOpera* and *EZ-Grid*.

JOpera (http://www.jopera.org) is an open grid workflow management system. It provides a visual environment based on the Eclipse platform for modelling grid workflows as a collection of jobs linked by control and data flow dependencies.

EZ Grid: it's a PC grid infrastructure based on the concept of virtualization. On top of this fundamental layer, other functionalities are also considered, such as job check-pointing, restarting and migration. These features are necessary in order to offer a flexible environment with minimal disturbances for both the owner of the PC and the owner of the job.

The user can submit his/her application via the workflow management system *JOpera* or directly through the *XWCH2* middleware. *XWCH2* can be deployed natively or as a virtual machine (Figure 3).

4 The MedGIFT application

One of the applications gridified and deployed on *Virtual EZ Grid* is MedGIFT. Content-based image retrieval is increasingly being used as a diagnostic aid in hospitals [17]. However, hospitals produce large amounts of images -- for instance the University Hospitals of Geneva radiology department produced about 70 000 images per day in 2007 [18]. Preparing these images in such a way that they can be used in diagnosis is a challenging task due to their volume. Content-based image retrieval systems typically use image features like properties of textures and colours [19]; here, we call the extracting features from images *indexing*.

The well-known GIFT, or Gnu Image Finding Tool, software is a content-based image indexing and retrieval package was developed at University of Geneva in the late 1990's. GIFT utilizes techniques common from textual information retrieval and uses a very large collection of binary-valued features (global and local colour and texture features) [19]. GIFT extract these features and stores them in an inverted file. In a typical desktop PC, the speed of indexing is about 1 or 2 images per second.

The history of the ImageCLEFMed image collection can be summarized as follows: ImageCLEF started within CLEF (Cross Language Evaluation Forum) in 2003. A medical image retrieval task was added in 2004 to explore domain-specific multilingual visual information retrieval [20]. The ImageCLEFMed2007 used in this report consists of about 50 000 images, originally from radiological journals Radiology and Radiographics. The images are originals used in published articles. Indexing a set of images can be seen as an "embarrassingly parallel" problem, i.e. "a problem in which little or no effort is required to separate the problem into a number of parallel tasks. This is often the case where there exists no dependency (or communication) between those parallel tasks." [21] Therefore indexing a large set (S) of images can be done by dividing S into small subsets, sending the subsets together with processing instructions into processing nodes, and then combining the output of the processing nodes.

The workflow for MedGIFT can be summarized as follow:

- Process PACKETGENERATOR runs in a client computer, using a directory of the ImageCLEFMed2007 sample as its input.
- PACKETGENERATOR generates packages that consist of the GIFT indexer program (executable), a batch file containing instructions of how to run it in the worker nodes, and an input zip file containing 1000 images (except for the last package).
- After each package has been generated, PACKETGENERATOR submits it to XWCH as a task.
- When all the tasks have been submitted, PACKETGENERATOR examines their statuses. When a status indicates that the task has been finished, PACKETGENERATOR downloads its output.

The process of executing PACKETGENERATOR (=the entire packaging/submission/result retrieving process) took 4 hours 53 minutes 58 seconds (=17638 seconds). This figure is comparable with those achieved by the ARC Grid middleware in [18]. Individual execution for packages in the worker nodes are shown in Figure 4. The short execution time of the last package is because it contained only 25 images. The average of the execution times was 1006 seconds (=16 min 36 seconds) and the sum of the execution times 51316 seconds. The figure of 51316 seconds (ca 14 hours) would thus roughly correspond with executing the whole task on a single CPU.

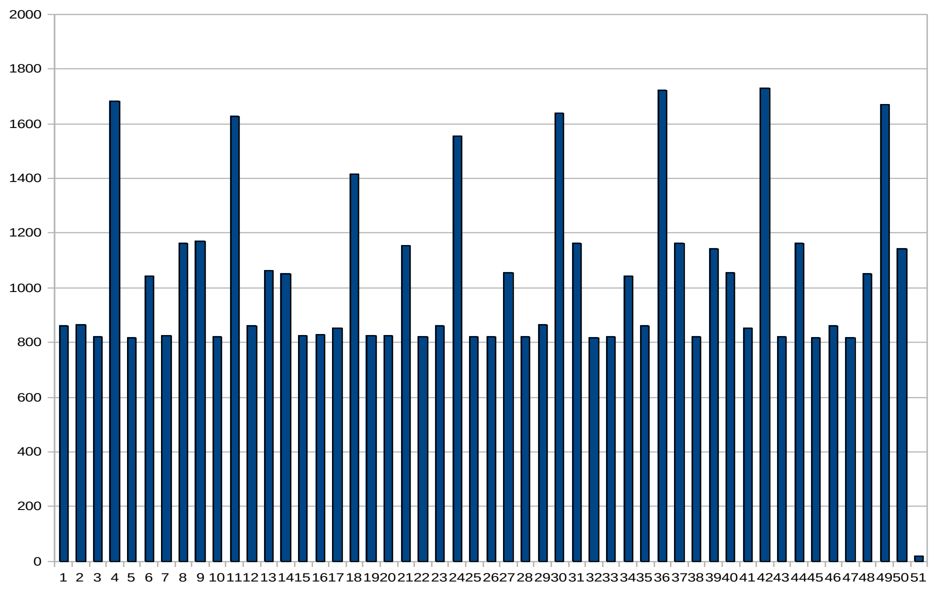

Figure 4: MedGIFT package execution times.

5. Conclusion

This paper has presented the new version of the volunteer computing environment *XtremWeb-CH* (*XWCH2*), used for the execution of high performance applications on a highly heterogeneous distributed environment. *XWCH2* is used in the *Virtual EZ Grid* project, where the computing nodes are provided from different institutes, and applications are built so that they utilise the *XWCH2* API directly, or by a JOpera workflow engine interface.

We have presented the details of the new features of *XWCH2*, in particular dynamic task creation. MedGIFT image indexing, a distributed application, is used as an example of a software that utilises the *Virtual EZ Grid* platform.

Acknowledgements

We gratefully acknowledge that:

- The *Virtual EZ Grid* project is in part funded via AAA/SWITCHand
- The right to use the ImageCLEFMed2007 was granted by Henning Muller on the basis that Marko Niinimaki was still a member of UNIGE's MedGIFT team in 2009.

References

[1] D.P. Anderson. Opinion - Volunteer computing: Grid or not Grid?. iSGTW newsletter, July 4, 2007
[2] gLite. http://glite.web.cern.ch/glite/
[3] Advanced Resource Connector (ARC). http://www.nordugrid.org/middleware/
[4] Globus. http://www.globus.org/
[5] Unicore. http://www.unicore.eu/
[6] M. Litzkow, M. Livny and M. Mutka: Condor - A Hunter of Idle Workstations. In Proceedings of the 8th IEEE Distributed Computing Conference, 1988.
[7] GridMP. http://www.univaud.com/hpc/products/grid-mp/
[8] D. P. Anderson: BOINC: A system for public-resource computing and storage. ACM International Workshop on Grid Computing, 2004.
[9] F. Cappello, S. Djilali, G. Fedak, T. Herault, F. Magniette, V. Neri and O. Lodygensky: Computing on Large Scale Distributed Systems: XtremWeb Architecture, Programming Models, Security, Tests and Convergence with Grid. FGCS Future Generation Computer Science, 2004.
[10] N. Abdennadher and R. Boesch: Towards a peer-to-peer platform for high performance computing, Proc. Eighth Intl. Conf. On High-Performance Computing in Asia-Pacific Region, 2005.
[11] N. Abdennadher and R. Boesch: A Scheduling algorithm for High Performance Peer-To-Peer Platform. CoreGrid Workshop, Euro-Par 2006, Dresden, Germany, August 2006 .

[12] N. Abdennadher, C. Evéquoz and C Bilat: Gridifying Phylogeny and Medical Applications on the Volunteer Computing Platform XtremWeb-CH. HealthGrid 2008, vol 134, IOS Press, 2008.
[13] Virtual_EZ_Grid. Http://www.xtremwebch.net/Projects/Virtual_EZ_Grid
[14] From-P2P. http://bilbo.iut-bm.univ-fcomte.fr/fromP2P/
[15] Nabil Abdennadher, Using the Volunteer Computing platform XtremWeb-CH: lessons and perspectives. ACSE'09, March 2009, Phuket (Thailand), 2009.
[16] D. A. Chappell and T. Jewell: Java Web Services, O'Reilly, 2002.
[17] H. Mueller, N. Michoux, D.Bandon, and A. Geissbuhler: A review of content-based image retrieval systems in medicine – clinical benefits and future directions. International Journal of Medical informatics, 73:1-23, 2004.
[18] M. Niinimaki, X. Zhou, A. Depeursinge, A. Geissbuhler and H. Mueller: Building a Community Grid for Medical Image Analysis inside a Hospital, a Case Study. MICCAI Grid Workshop, New York University, September 2008.
[19] D. M. Squire, W. Mueller, H. Mueller, T. Pun: Content-based query of image databases, inspirations from text retrieval: inverted files, frequency-based weights and relevance feedback. Technical Report 98.04, Computer Vision Group, Computing Centre, University of Geneva, 1998.
[20] H. Müller, J. Kalpathy-Cramer, C. E. Kahn Jr., W. Hatt, S. Bedrick and W. Hersh: Overview of the ImageCLEFmed 2008 Medical Image Retrieval Task, Evaluating Systems for Multilingual and Multimodal Information Access, LNCS Volume 5706/2009, 2009.
[21] I. Foster. Designing and Building Parallel Programs. Addison-Wesley, 1995.