Middle-R - A User Level Middleware for Statistical Computing

Rodrigo A. Dias, Alfredo Goldman, Roberto Hirata Jr.

[1]Instituto de Matemática e Estatística - Universidade de São Paulo (USP)
Rua do Matão, 1010, São Paulo - SP, BRAZIL

{rdias,gold,hirata}@ime.usp.br

Abstract. *In this work we present middle-R, a user-level middleware for statistical computing. It has been developed to distribute R (a language and environment for statistics) in commodity computers running Microsoft Windows. The solution is simple because it uses Alchemi, a grid middleware developed using Microsoft .NET technology in the University of Melbourne. The implemented solution uses Alchemi as a low-level middleware and makes it possible to execute R scripts in a grid environment. The solution has been successfully implemented and tested in two different environments: the didactic laboratories of a college and of a high school. The main application reported in this paper is a combinatorial task for finding molecular markers of a few genes to classify correctly samples of acute lymphoblastic leukemia against acute myeloid leukemia.*

1. Introduction

Bioinformatics poses some huge computational problems, for instance: DNA sequence analysis, finding molecular classifiers of few genes to predict cancer, or assessing the confidence of statistical tests (that may require the use of a known statistical technique called bootstrap). The typical time frame to complete one of these tasks in a high-performance computer is measured in weeks. To facilitate this kind of computation, cluster and grid computing fits very well, mainly when the analysis can be broken in thousands of small independent processes with slightly varying inputs, a kind of parallelism usually referred to as "embarrassingly parallel" computing [Foster 1995].

Unix solutions to distribute scientific applications are not new and there are several successful histories based mainly on MPI [Dongarra et al. 1993, Snir and Otto 1998] and PVM [Geist et al. 1994]. However, these solutions are mainly based on proprietary clusters or on grid middleware.

Our work is motivated by the following reasons: To avoid the acquisition costs of large clusters or the possible overhead of using an existent middleware we choose a volunteer computing approach. Moreover, due to the large availability of desktops running Microsoft Windows systems we propose a Windows based solution (which also can also be extended to run on heterogeneous machines).

We have started our own grid project to use the computer facilities of our collaborators, college and high school computing laboratories with commodity computers. Our project have initially faced four boundary conditions: (1) practically all labs' computers run Windows, so should the grid, (2) the software must run without affecting the daily activities of the users, (3) the grid software has to easily communicate with other software,

and (4) the grid has to be able to communicate with other grids. A suitable platform to be used under those conditions is naturally .NET [Platt 2002], mainly if we consider that nowadays the majority of the stations run Microsoft operating systems, and that most grid solutions available are for Unix or Linux platforms.

We firstly considered creating our own .NET grid solution but then we have found a .NET middleware called Alchemi [Akshay Luther and Venugopal 2003, Akshay Luther and Venugopal 2005, A. Luther and Venugopal 2005] that helped us with conditions (1) and (4) and let us concentrate on our bioinformatics problems. Alchemi is a grid middleware entirely developed using Microsoft .NET Framework at University of Melbourne as part of the Gridbus project. We have chosen to use Alchemi as the low level grid middleware because it was developed targeting this kind of demand, increasing the technology acceptance by private companies and common users. For condition (2), the natural way to go is to make Alchemi work as a Windows service that should be easy to install, start, stop and with no necessity of user interaction. Alchemi is an open source project under GNU's LGPL so we had no problem adapting it to our needs. We later contributed this modification back to Alchemi's project and it has been incorporated to the official distribution.

The solution to condition (3) and an application of the solution are the main subject of this paper. An important environment to do bioinformatics nowadays is the R [R Development Core Team 2006] software, an open source implementation of S language and environment. R has been compiled to several platforms as: Linux, Windows, MacOS and Solaris. R can be integrated with several languages (C, Fortran, etc) but has no support for .NET. However, it can communicate with other processes using Microsoft COM [Baier and Neuwirth 2006] technology by using a contributed package called RCOM. To solve our problem of using R in a grid environment, we propose middle-R, a user level middleware that uses Alchemi as a low-level middleware and makes it possible to execute R scripts in a grid environment. The layer has been implemented and used in our bioinformatics activities.

1.1. Related Work

There are several packages to parallelize R processes [Rossini et al. 2003, Yu 2006, Li and Rossini 2006, Markus Schmidberger and Mansmann 2009]. All of them have been implemented and are suitable for Linux but can be adapted to Windows with some limitations. However, they are used better in a small dedicated network of computers (a cluster, for instance) than in a large infrastructure. Besides that, the user has to manage the computer names and distribution of work for each one in the R script. There is no mechanism of logging, ressubmission and others, that are usually found in grid middleware. Condor [Douglas Thain and Livny 2005] is another workload management system to distribute workloads in a network. There are some notices of groups using R under Condor for Windows but there is no paper or report available. A system of volunteer grid computing is an alternative solution recently presented for interpreted applications within BOINC infrastructure [González et al. 2008].

1.2. Paper Organization

After this Introduction, in Section 2 we introduce some of the technologies involved in the solution, in Section 3 we present middle-R, in Section 4 we present an example of

application and in the Conclusion we discuss some future steps we think will improve the work.

2. Involved Technologies

In this Section, we outline the major technologies involved in the proposed solution, i.e., R and Alchemi.

2.1. R

The most popular open software project for statistical analysis and computing nowadays is the R language and environment [R Development Core Team 2006]. The project has been originally developed by Robert Gentleman and Ross Ihaka of the Statistics Department of the University of Auckland and later it has been made freely available under GNU's GPL. The project's source code and binaries to several supported platforms (Linux, Unix, Windows and MacOS) are freely available through the project's website: `www.r-project.org`. As an environment, R offers to the user data handling and storage methods, matrix computation operators, a large set of tools for data analysis and graphics generation, implemented statistical analysis functions such as linear and non-linear modelling, statistical tests, time series analysis, data classification, etc. As a language, R offers a complete and robust programming language with resources like input/output handling, conditionals, loops, and user-defined functions.

Support for extensibility, that is, user's development and integration of packages is another important feature of the environment. There are more than one thousand packages available through the website at the time of this writing. Through these packages, R can be extended by adding features such as database connection support, Internet communication support, signal and image basic analysis, and so on.

R environment is also very popular among biostatisticians because of its support for biological data analysis. In fact, many statistical and computational techniques used in bioinformatics are present in R through community designed packages as an independent project called Bioconductor [Gentleman et al. 2004], making it a very popular tool among these researchers.

2.2. Alchemi

Alchemi is an open source grid middleware entirely developed using Microsoft .NET Framework by Dr. Rajkumar Buyya and his team in the University of Melbourne as part of the Gridbus project [Buyya and Venugopal 2004]. Grid applications can be developed using Alchemi's own API and executed on a desktop grid constructed using Alchemi's runtime facilities. Both development and execution of grid applications are supported by a strong object-oriented model, giving easy access and total control to all of its features to users and developers.

Alchemi's architecture is divided in four well defined components: the **Manager**, that manages execution of grid applications, threads and grid resources; the **Executor**, responsible for dedicated or voluntary execution of grid threads; the **User**, represents the owner of an application and the **Cross-Platform Manager**, an optional component responsible for exposing the functionalities of the Manager to global grids through an

web service interface. Each one of these components are represented by an object on Alchemi's object-oriented programming model.

New resources can easily be aggregated into an Alchemi grid by registering a new Executor in the Manager node anytime, even with an application already running on that grid. The new Executor will start receiving grid threads as soon its Manager's scheduler finds it necessary. The Executors can be in the same network, in adjacent networks or over the Internet behind a firewall or NAT device, the only restriction is that it must be able to access a predetermined port in the Manager.

To make Alchemi compliant with one of our boundary conditions (run without affecting the normal life of the laboratories) some modifications have been made in the Alchemi Executor in order to make it to run as a Windows service. As a service, the Executor can run unattended and outside any user context or interference. To accomplish this, some portions of the `AlchemyCore` library had to be rewritten and the Executor was divided into two separated applications: the Executor Service - the Windows service itself - that contains the same functionalities of the Executor, and the Executor Service Controller - the service GUI - that connects to the Executor Service and have the same functionalities of the Executor GUI, in addition to service connect, disconnect, start and stop functionalities. These changes were later contributed back to Alchemi's project and incorporated into the official distribution.

2.3. RCOM

The Microsoft Component Object Model (COM) technology enables interprocesses communication between software components from different vendors. A key package used in our solution is RCOM package [Baier and Neuwirth 2006], developed by Erich Neuwirth and Thomas Baier as a package for Windows-based R installations. Through RCOM, we can expose R's features to other Windows applications written in any COM compatible programming language via COM interface.

When RCOM is installed on a machine, a COM Server is created enabling COM clients to connect and create objects representing an R instance. Through this instance, one can transfer data (with proper data conversions) between R and a client application, use R's computing engine, access its graphical features and input/output facilities.

3. Middle-R

We start this section with a discussion on .NET and COM interoperability, we continue with a brief explanation on Alchemi's programming models and then we discuss the proposed solution in details.

3.1. .NET and COM Interoperability

COM was designed some year's before .NET initiative. Microsoft consider them complimentary technologies but, in fact, they are incompatible as their programming models differ greatly. The main problem is that COM code is not executed by .NET Common Language Runtime (CLR) [Meijer and Miller 2000] and is not managed by any CLR service (unmanaged code). That is, all components that .NET managed code can access are those managed by CLR.

In our work, we have created a component that uses a Remote Callable Wrapper (RCW) to wrap the RCOM object and do all the conversions between it and Alchemi C# code, making the RCOM object appear to Alchemi just as if it were a native .NET object. This interoperable layer was constructed using `tlbimp.exe`, a tool from .NET SDK for the conversion of public interfaces between COM and .NET using information stored in COM component type library.

3.2. Alchemi's programming models

Alchemi has two object-oriented programming models: (1) a grid thread programming model and (2) a grid job programming model [Akshay Luther and Venugopal 2003, Akshay Luther and Venugopal 2005]. In the grid thread programming model, the atomic processing unit is an object inherited from the class `GThread`, which is serialized by a grid application running under a Manager node and transmitted to an Executor node responsible for its execution. This model gives the developer a closer control of the distributed threads. On the other hand, the grid job programming model has been developed to leverage legacy applications to become grid-enabled. In this latter model, the atomic processing unit is a `GJob` object (a class that extends `GThread` to enable legacy applications to run on the grid) that depends on one or more files that will be copied by the Manager and executed by an Executor node.

Alchemi has two ways of accessing the grid job model: (1) declaratively, by using a task description XML file to describe jobs and its dependencies; and (2) programmatically, by creating an application via Alchemi API that instantiates and configures a `GApplication` object, which contains multiple `GJob` objects.

3.3. middle-R and Alchemi

Due to .NET Runtime design limitations, Alchemi .NET managed code cannot directly access R computation engine and library, written in unmanaged code. For this reason, we needed to design an interoperability layer to enable Alchemi code to access R engine. So, we took advantage of RCOM package to access R engine through its public interfaces published as a COM server object, to design a small .NET program, called `RJob`. The role of `RJob` is to receive R scripts along with Alchemi jobs, evaluates them against R engine and returning the evaluation output as Alchemi job result. To be able to evaluate R commands, `RJob` needs first to initialize the R engine. It is done by evoking `Init(...)` method of `STATCOnnector` class, the interoperability layer between R's COM object and `RJob` unmanaged code. The initialization command is passed to the `STATConnector`, the COM object that interfaces R, which passes again to `RProxy` where it will be evaluated in R engine itself. The engine is now prepared to receive R commands, which will follow the same path as the initialization command. Figure 1 shows the layers involved in `RJob`, while Figure 2 shows the interactions of these layers as a sequence diagram. For several design and conceptual reasons, RCOM server object cannot be serialized and passed as a parameter to an Executor node inside a grid thread object, as expected by Alchemi's grid thread programming model. Therefore, middle-R has been built using Alchemi's grid job model. In our solution, RJob will be copied from the Manager node to the Executor nodes, along with an input file containing the R script we want to run in each node of the grid, as described in Figure 3. In every Executor node, RJob will parse that input file and use the R engine to evaluate the script and generate

an output file containing the execution results of the script. In this version, we have chosen to get textual response for each command, instead of R response objects, due to data conversion costs.

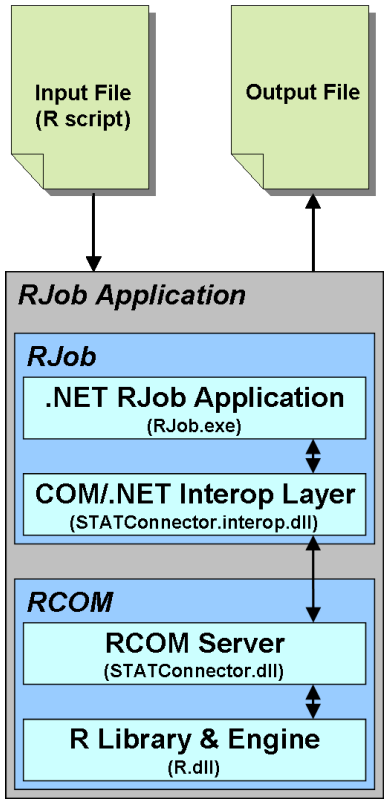

Figure 1. RJob application scheme.

An application in Alchemi is an instance of `GApplication` and contains a Manifest of files which that application depends on, that is, a collection of `FileDependency` objects needed by the application to be executed on remote Executor nodes. The Manifest must contain all the files RJob will need to be executed locally, including `RJob.exe`. The `GApplication` object contains a collection of threads, that is, all the jobs that will be executed. As mentioned above, a job is an object inherited from `GJob` and has an input file collection, an output file collection and a command that will be executed when the job runs on Executor node. Figure 3 shows the scheme for the middle-R in this context

4. Application Example

In this section we outline a possible utilization of the middle-R, a combinatorial bioinformatics problem to finding molecular classifiers using pattern recognition techniques.

One of the important research areas in Medicine, and consequently in Bioinformatics, is the correct prognosis and diagnosis based on the expressions of certain genes in a tissue sample. The expression of a gene is the quantity of that gene that are being expressed in the cell in a certain moment of the cell's life. The idea is to find a set of genes that could have their expression measured (from biopsies tissues, for instance) and

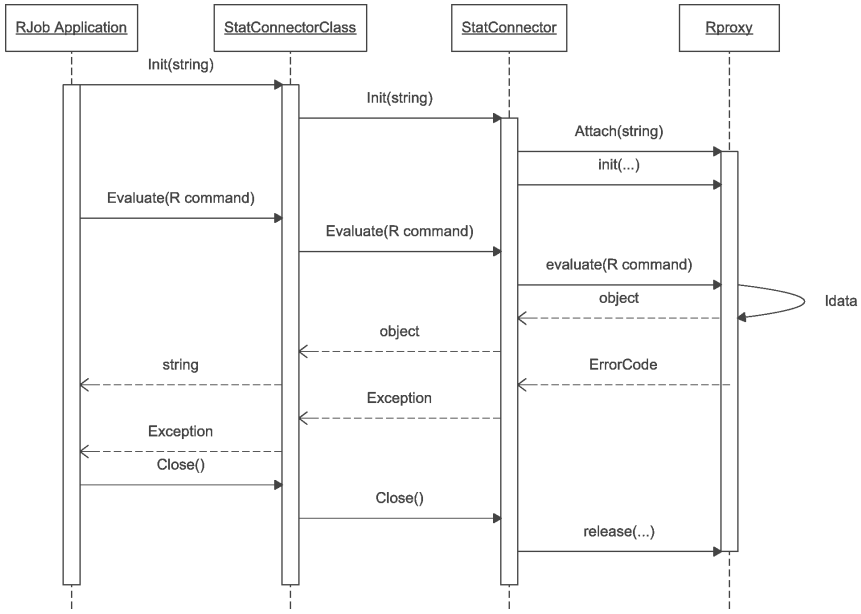

Figure 2. Middle-R sequence diagram.

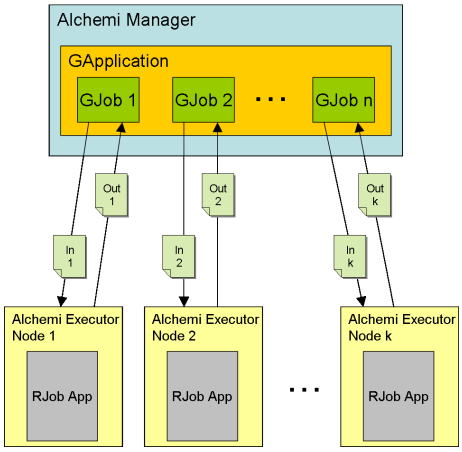

Figure 3. Middle-R scheme.

analyzed to help the diagnosis, or even better, the prognosis of a certain tissue condition [Zweiger 1999].

There are several techniques to quantify the gene expression of the cells of a given tissue [Bowtell 1999]. cDNA microarray and high density oligonucleotide arrays are the most used microarray techniques. The strength of these techniques is that each microarray experiment can assess the expression of thousand of genes at the same time.

For practical purposes, the output of an experiment is a $n \times p$ matrix, where n is the number of genes and p is the number of tissues being assessed. Each matrix element $g_{i,j}$ is the expression of the gene i for the tissue j. The expressions can be absolute or relative to another tissue or pool of tissues expressions. In the case of the dataset used in our work [Golub et al. 1999], the experiment is to compare samples of Acute Lymphoblastic Leukemia (ALL) to Acute Myeloid Leukemia (AML). The expression is measured using Affymetrix high-density oligonucleotide arrays containing $p = 6817$ genes. The dataset

comprises 47 ALL tissues and 25 AML tissues. The dataset was prepared following the steps reported in [S.Dudoit et al. 2002], i.e., (i) thresholding: floor of 100 and ceiling of 16000; (ii) filtering: exclusion of genes with max/min ≤ 5 and max $-$ min ≤ 500, where max and min refer respectively to the maximum and minimum expression levels of a particular gene across mRNA samples; (iii) base 10 logarithmic transformation. The final dataset is a 72×3051 matrix of gene expressions, i.e., after preparation the number of genes is 3051.

There are several techniques in the area of Pattern Recognition to find classifiers by supervised learning. For statistical and computational reasons, we are going to use Fisher Linear Discriminant Analysis (FLDA) [Bishop 2006] to find the pairs of genes that perfectly classify the tissues by cancer types.

Let $\mathbf{x}$ be a point in the plane formed by the expression levels of a pair of genes (g_i, g_j) and y be such that $y = \mathbf{w}\mathbf{x}$, where $\mathbf{w}$ is a weighting vector that better separates the projections of the two cancer types (ALL and AML). The idea behind FLDA is to find, by computing projections to a hyperplane (or a line in our case), the vector $\mathbf{w}$ that maximizes the ratio of the between-class covariance matrix $\mathbf{S}_B$ to the within-class covariance matrix $\mathbf{S}_W$ defined by:

$$\mathbf{S}_B = (\mathbf{m}_2 - \mathbf{m}_1)(\mathbf{m}_2 - \mathbf{m}_1)^T \tag{1}$$

and

$$\mathbf{S}_W = \sum_{n \in C_1} (\mathbf{x}_n - \mathbf{m}_1)(\mathbf{x}_n - \mathbf{m}_1)^T + \sum_{n \in C_2} (\mathbf{x}_n - \mathbf{m}_2)(\mathbf{x}_n - \mathbf{m}_2)^T \tag{2}$$

where: $\mathbf{m}_1 = \frac{1}{N_1} \sum_{n \in C_1} \mathbf{x}_n$ and $\mathbf{m}_2 = \frac{1}{N_2} \sum_{n \in C_2} \mathbf{x}_n$.

Solving this problem is equivalent to inverting matrix $\mathbf{S}_W$ and multiplying by the difference of the means $\mathbf{m}_1$ and $\mathbf{m}_2$. This is not a discriminant but only the direction of the normal vector of the separating line between the two types of cancer. The discrimination point can be found by a naive Bayes approach [Bishop 2006].

To compute one discriminant is very fast (about 0.025s). However, we need to compute $C(3051, 2) = 4652775$ discriminants and check if any pair separates perfectly the cancer samples. To do that, we can split the jobs to the grid (because they are independent) and just check the results. There are 514 pairs of genes that satisfy the requirement of separating the cancer types perfectly.

To perform the experiments, two didactic laboratories have been prepared to run an Alchemi grid. Each laboratory has 20 machines: 20 Intel Celeron D330 2.66 GHz, 256 Mb RAM in one of them (named here Lab1) and 20 Intel Celeron D315 2.25 GHz, 256 Mb RAM in the other (named here Lab2) . Both laboratories run Windows XP SP and at least the following packages are installed in each machine: R (version 2.6.0), RDCOM Server (version 1.8.3) and Alchemi (version 1.0.6). The machines in each laboratory are connected to a ethernet hub (10 Mbps) and both hubs are connected in cascade mode and connected by a switch to the administrative network. During the tests, all machines were dedicated to the experiment, i.e., there were no cycle sharing among other users or applications. Despite of the fact that the middle-R can be used in an opportunistic way, we provided a dedicated setting for the experiments in order to verify the performance.

To assert the stability of the performance of the solution, a subset of the pairs to

be tested has been chosen (5000 pairs of genes selected at random) as part of a job that was tested in three different experiments with one, ten and twenty nodes.

All the experiments had a number of jobs equal to the number of nodes and were repeated 100 times. For a one node grid, the maximum time of execution for this experiment is 2 minutes and 10 seconds, the minimum time of execution is 2 minutes and 7 seconds and the standard deviation is 0.828 seconds. For a ten nodes grid the minimum time of execution was 2 minutes and 8 seconds, the maximum execution time was 2 minutes and 14 seconds and the standard deviation was 1.81 seconds. For a twenty nodes grid the minimum time of execution was 2 minutes and 8 seconds, the maximum execution time was 2 minutes and 16 seconds and the standard deviation was 1.68 seconds.

The performance of the grid application has been compared to a stand alone machine (Lab1) by running the R script to do all the 4652775 tests (same dataset), i.e., no grid infrastructure. The experiment has been repeated using a grid with five, ten, twenty and forty nodes. For all experiments but the last (40 nodes), only Lab1 machines have been used. Initially, the number of pairs to be tested have been divided in 1000 jobs (approximately 4653 tests/job). Table 1 shows the results for each one of the tests. A super-scalar effect can be noticed comparing the time of 5 nodes against 10 nodes.

Nodes	Time
$\star$	43:39:19
5	15:49:22
10	7:43:16
20	4:18:45
40	2:12:14

Table 1. Time comparison between experiments with 5, 10, 20 and 40 grid nodes checking all pairs divided into 1000 jobs. Time column represented as HH:MM:SS format.
($\star$) Time measured outside grid infrastructure.

To reduce the impact of communication between Alchemi and the nodes, a second experiment has been done. This time, the number of jobs is equal to the number of nodes in the experiment and each job tests $4652775/N$, where $N = \{5, 10, 20, 40\}$. On this second experiment, instead of having small jobs and a large amount of communication on the cluster, we used large jobs and N communications.

Table 2 shows the results for each one of the tests. Comparing to Table 1, the total time has been reduced in 0,88% for 5 nodes, 3,74% for 20 nodes and 7,69% for 40 nodes. However, the time of execution for 10 nodes have not shown the super-scalar effect. With the last two experiments it is possible to observe a kind of trade-off between the communication among jobs, and the disk swap effects of large jobs[1].

5. Conclusion

There are several grid solutions for Unix environments and there are some packages to facilitate R applications to be distributed under Unix. Under Windows, there were no

[1]"We performed some additional tests to verify the communication overhead. On all cases but with 10 nodes it corresponds to the time difference. So, we suppose that the difference with 10 nodes comes from disk swap.

Nodes	Time
$\star$	43:39:19
5	15:41:02
10	8:06:33
20	4:09:05
40	2:02:04

Table 2. Time comparison between experiments with 5, 10, 20 and 40 grid nodes checking all pairs divide into N jobs where $N = \{5, 10, 20, 40\}$. Time column represented as HH:MM:SS format.
($\star$) Time measured on R outside grid infrastructure.

"off-the-shelf" solution to distribute R when we started this project. Fortunately, using Alchemi, a grid middleware that uses Microsoft .NET technology, we have designed and implemented middle-R, a user level middleware to enable R applications to be executed in a grid environment. More recently, two solutions are being tested: one under Condor and another under BOINC.

There is a very important difference between our solution and the ones proposed before: since middle-R is integrated with Alchemi, a grid middleware, the grid can scale to a very large network of computers, distributed across different administrative domains, while in the other solutions that would be very difficult or even impossible. The proposed solution has been deployed in computing laboratories (all using commodity computers) of two educational institutions in a way that is not disturbing the didactic activities (users can continue using Windows and do not notice performance degradation).

We have used the grid in some of our bioinformatics problems, mainly microarray statistical analysis to find robust biological classifiers (gene-cliques to distinguish cancer and non-cancer tissues, or to distinguish between types of cancer) and we could solve an important problem related to scientific reproducibility that is controlling the version of R packages in a distributed environment. We have also implemented a .NET component responsible for R communication (that supports serialization) to enable middle-R to use the grid thread model. One advantage of this solution is that we have a better control of the threads. The disadvantage is that it is slower than the COM solution and not fully compatible with R commands.

In economical terms, there are several aspects that we started studying but have to be improved. According to our initial studies, the energy cost per machine is approximately US$0.0135 (considering the kw/h cost approximately US$0.012 per machine) with the monitor in idle status and the CPU in a peak of processing. If we consider that the cost per hour of machine instance is approximately US$0.1 for the EC2 (Amazon's computing grid), the proposed solution is economically viable, even counting other cost variables as maintenance, cost of ownership and etc. The proposed solution is also economically adequate for computing labs because, in the future, the institutions may be able to rent computing time and consequently reduce the cost of ownership, leverage research partnerships and mainly, being a non-profit institutions, reduce the students' tuition.

As a future work we intend to further explore the effects on the jobs size on the total execution time, using different settings as heterogeneous machines and faster net-

works. We also intend to add fault tolerance capacities to the jobs, using a kind of heart beat verification is order to allow the resolution of actual problems.

6. Acknowledgements

The authors thank the SENAC College of Computer Science and Technology and Colégio Rainha da Paz for allowing us to use their computing resources. The authors thank Alexandre Gomes Ferreira, Paula Akemi Nishimoto, Lucas A. M. Perin and Dr. E. Jordão Neves for their participation in the development of the project. The authors thank Dr. Gisele S. Craveiro and Dr. Nina S. T. Hirata for reviewing part of the work. The authors thank the anonymous reviewers for important suggestions to improve this paper. The authors also thank Dr. Buyya and his team for designing and developing Alchemi and Gridbus. Alfredo Goldman and Roberto Hirata Jr. are partially supported by CNPq.

References

A. Luther, R. Buyya, R. R. and Venugopal, S. (2005). Alchemi: A .NET Based Enterprise Grid Computing System. In *Proceedings of the 6th International Conference on Internet Computing (ICOMP'05)*, Las Vegas, USA.

Akshay Luther, Rajkumar Buyya, R. R. and Venugopal, S. (2003). Alchemi: A .NET-based Grid Computing Framework and its Integration into Global Grids. Technical Report GRIDS-TR-2003-8, Grid Computing and Distributed Systems Laboratory, University of Melbourne, Australia.

Akshay Luther, Rajkumar Buyya, R. R. and Venugopal, S. (2005). *High Performance Computing: Paradigm and Infrastructure*, chapter Peer-to-Peer Grid Computing and a .NET-based Alchemi Framework. Wiley Press, New Jersey, USA.

Baier, T. and Neuwirth, E. (2006). *R COM Client Interface and internal COM Server*. R package version 1.4.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.

Bowtell, D. D. L. (1999). Options Available – from Start to Finish – for Obtaining Expression Data by Microarray. *Nature (Genetics Supplement)*, 21:25–32.

Buyya, R. and Venugopal, S. (2004). The Gridbus toolkit for service oriented grid and utility computing: an overview and status report. In *Grid Economics and Business Models, 2004. GECON 2004. 1st IEEE International Workshop on*, pages 19–66.

Dongarra, J. J., Hempel, R., Hey, A. J. G., and Walker, D. W. (1993). Mathematical Sciences Section: A PROPOSAL FOR A USER-LEVEL, MESSAGE PASSING INTERFACE IN A DISTRIBUTED MEMORY ENVIRONMENT.

Douglas Thain, T. T. and Livny, M. (2005). Distributed computing in practice: The condor experience. In *Concurrency and Computation: Practice and Experience*, volume 17, pages 323–356.

Foster, I. (1995). *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering*, chapter 1.4.4. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R., and Sunderam, V. (1994). *PVM Parallel Virtual Machine, A User's Guide and Tutorial for Networked Parallel Computing*. MIT Press, Cambridge, Mass.

Gentleman, R. C., Carey, V. J., Bates, D. M., Bolstad, B., Dettling, M., Dudoit, S., and-Laurent Gautier, B. E., Ge, Y., Gentry, J., Hornik, K., Hothorn, T., Huber, W., Iacus, S., Li, R. I. F. L. C., Maechler, M., Sawitzki, A. J. R. G., Smith, C., Smyth, G., Yang, L. T. J. Y. H., and Zhang, J. (2004). Bioconductor: Open software development for computational biology and bioinformatics. *Genome Biology*, 5:R80.

Golub, T. R., Slonim, D. K., Tamayo, P., Huard, C., Gaasenbeek, M., Mesirov, J. P., Coller, H., Loh, M. L., Downing, J. R., Caligiuri, M. A., Bloomfield, C. D., and Lander, E. S. (1999). Molecular Classification of Cancer: Class Discovery and Class Prediction by Gene Expression Monitoring. *Science*, 286(5439):531–537.

González, D. L., de Vega, F. F., Trujillo, L., Olague, G., Cárdenas, M., Araújo, L., Castillo, P., Sharman, K., and Silva, A. (2008). Interpreted applications within boinc infrastructure. In *Ibergrid 2008. 2nd Iberian Grid Infrastructure Conference Proceedings*, pages 261–272.

Li, N. and Rossini, A. J. (2006). *rpvm: R interface to PVM (Parallel Virtual Machine)*. R package version 1.0.1.

Markus Schmidberger, Martin Morgan, D. E. H. Y. L. T. and Mansmann, U. (2009). State-of-the-art in parallel computing with r. Technical report, Institut für Statistik.

Meijer, E. and Miller, J. (2000). Technical overview of the Common Language Runtime. Technical report, Microsoft Corp.

Platt, D. S. (2002). *Introducing Microsoft .Net, Second Edition*. Microsoft Press, Redmond, WA, USA.

R Development Core Team (2006). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0.

Rossini, A., Tierney, L., and Li, N. (2003). Simple parallel statistical computing in R. *UW Biostatistics working paper series*. Paper 193.

S.Dudoit, Fridlyand, J., and Speed, T. P. (2002). Comparison of discrimination methods for the classification of tumors using gene expression data. *Journal of the American Statistical Association*, 97(457):77–87.

Snir, M. and Otto, S. (1998). *MPI-The Complete Reference: The MPI Core*. MIT Press, Cambridge, MA, USA.

Yu, H. (2006). *Rmpi: Interface (Wrapper) to MPI (Message-Passing Interface)*. R package version 0.5-3.

Zweiger, G. (1999). Knowledge discovery in gene-expression-microarray data: mining the information output of the genome. *Trends in Biotechnology*, 17(11):429–436.