Comparing Grid Computing Solutions for Reverse-Engineering Gene Regulatory Networks

Martin Swain, Johannes J. Mandel, and Werner Dubitzky

School of Biomedical Sciences, University of Ulster, Coleraine BT52 1SA, UK
mt.swain@ulster.ac.uk

Abstract. Models of gene regulatory networks encapsulate important features of cell behaviour, and understanding gene regulatory networks is important for a wide range of biomedical applications. Network models may be constructed using reverse-engineering techniques based on evolutionary algorithms. This optimisation process can be very computationally intensive, however its computational requirements can be met using grid computing techniques. In this paper we compare two grid infrastructures. First we implement our reverse-engineering software on an opportunistic grid computing platform. We discuss the advantages and disadvantages of this approach, and then go on to describe an improved implementation using the QosCosGrid, a quasi-opportunistic supercomputing framework (Qos) for complex systems applications (Cos). The QosCosGrid is able to provide advanced support for parallelised applications, across different administrative domains and this allows more sophisticated reverse-engineering approaches to be explored.

1 Introduction

Computational grids are able to virtualise distributed, heterogeneous processing and storage resources in order to create a single, integrated infrastructure with great power. Such grids are able to provide computing capacity greater than that of advanced supercomputers – but only for certain applications that typically consist either of independent tasks or tasks that are pleasantly parallelisable because highly parallel applications cannot run efficiently on the grid's distributed infrastructure. However, by reducing the gap between grid infrastructures and supercomputers through the development of quasi-opportunistic supercomputing middleware the QosCosGrid project aims to provide a suitable grid infrastructure for the simulation of complex systems such as gene regulatory networks [1].

In this article we compare two grid infrastructures and show how they can be used to reverse-engineer models of gene regulatory networks by discovering model parameters that generate specific behaviour. Parameter estimation is an important task for many complex systems applications and evolutionary algorithms are a commonly used approach. There are various different implementations of distributed evolutionary algorithms, with different parallelisation approaches and patterns of communication. It is an aim of the QosCosGrid

M. Bubak et al. (Eds.): ICCS 2008, Part I, LNCS 5101, pp. 106–115, 2008.
© Springer-Verlag Berlin Heidelberg 2008

project to develop middleware for parallelised applications, and to provide a toolkit for evolutionary computing. Here we describe how we have grid-enabled existing reverse-engineering software, called Evolver, which has a basic distributed evolutionary algorithm implementation. We then compare an implementation of Evolver, based on Active Objects using ProActive Java [2] and the QosCosGrid, with a batch-processing method, based on the DataMiningGrid [3] using Globus [4] and Condor [5]. Finally we outline how we plan to take advantage of the further functionality which will become available in the QosCosGrid.

2 The Problem: Reverse-Engineering Gene Regulatory Networks

2.1 Gene Regulatory Networks

Gene-regulatory networks (GRNs) are important for understanding an organism's complex dynamical behaviour. Within a GRN, genes and their products interact with one another - genes code for proteins that may in turn regulate the expression of other genes in a complex series of positive and negative feedback loops [6]. Thus gene expression, as regulated by the network, is essential in determining the functional state or physical characteristics of an organism. GRNs may be viewed as a complex cellular control system in which information flows from gene activity patterns through a cascade of inter- and intracellular signalling functions back to the regulation of gene expression [7].

The dynamic activity of GRNs have been revealed by microarray time series experiments that record gene expression [8] and it has been hypothesized that a GRN's structure may be inferred from this, and related, time-series data. This is a reverse-engineering problem in which causes (the GRNs) are deduced from effects (the expression data) [9].

2.2 Evolutionary Computing

Evolutionary computing is inspired by the process of natural evolution, by which new organisms emerge due to changing environmental pressures. Genetic algorithms [10] are an optimisation technique, whereby solutions to a particular problem are represented by individual organisms: the *genotype* of an individual refers to the encoding of a possible solution to the problem, and the *phenotype* represents the possible solution in a form that can be evaluated.

The gene pool is optimised through evolutionary processes, such as mutating and combining genotypes, and evaluating and removing the weakest phenotypes. This is an iterative process, with stochastic mechanisms controlling the mutation and combination of different genotypes. Phenotypes are evaluated using a fitness function, representing the environment, and the evolutionary process proceeds until an individual eventually emerges that represents a suitably accurate solution to the optimisation problem.

For example, an individual's genotype may represent the parameters of a complex system simulation, and the evaluation of an individual's phenotype would

represent how well the output of the simulation meets certain criteria, such as its time-dependent behaviour. It is therefore possible to use evolutionary computing to discover or reverse-engineer complex system simulations with specific properties.

2.3 The Evolver Software Package

The software used in this study was the Evolver component from the Narrator gene network modelling application [7]. While GRNs may consist of hundreds of genes, Narrator is typically used to model networks of about 10 genes.

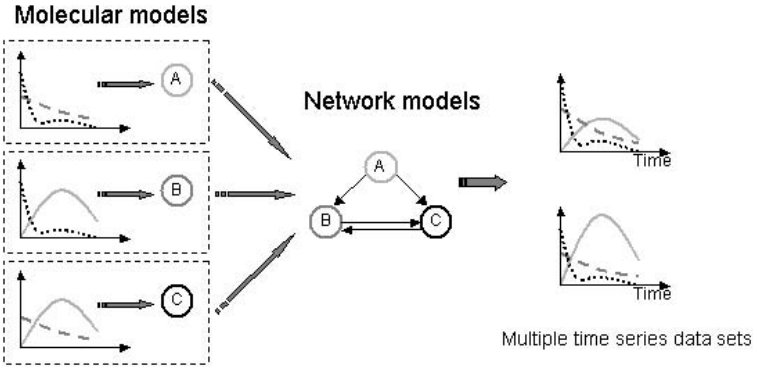

Fig. 1. Showing how molecular models involve each gene being individually optimised using the other genes' expression levels as input; and how network models are optimised to fit the full set of time-series data, which may describe gene expression under different conditions

In a common reverse-engineering scenario where Evolver is used it is assumed that the basic network topology is known, but it is not known if one gene inhibits or excites an other gene. Three different systems of differential equations can be used with Evolver, and through the process of reverse-engineering the parameters of these equations it is possible to determine typical network features such as positive and negative feedback loops. The more detailed and complete the data sets, the more accurate the results.

Evolver is specifically designed to perform gene network feature discovery using a two stage evolutionary optimisation approach, as shown in Fig. 1:

1. **Molecular modelling:** In this stage individual genes are optimised. Initially a population of genes is created using random parameters for each gene. The time-series expression levels of the other genes are used as input to the differential equations used to model the gene's dynamics, and the reverse-engineering algorithm is used to predict the parameters need to fit the output of the gene model being reverse-engineered to its expression levels.

2. **Network modelling:** The most promising models from the molecular modelling stage are combined to form a population of genetic networks, and these are optimised to fit the expression level time series data sets.

To grid-enable Evolver it was necessary to create three packages, two corresponding to the molecular and network modelling stages given above, and a third package for result processing. On the grid, each modelling package can be used to create multiple jobs, for example the optimisation of a ten gene network can be performed with thirty grid jobs: each of the ten genes is optimised three times in order to avoid problems with local minima, so there are 30 executions of the molecular modelling package, which may be performed simultaneously if sufficient grid resources are available. The data processing stage is used to collect the output of these thirty jobs and format it in preparation for the following stage.

3 Deploying Evolver on Grid and Distributed Computing Frameworks

3.1 Implementation of Evolver on the DataMiningGrid

The DataMiningGrid is described in detail elsewhere [3]. In summary, the DataMiningGrid was built using the Globus Toolkit [4], which was extended with a number of enhancements, for example to the execution environment in order to utilise the full functionality of Condor's Java Universe. Users usually interacted with the system via the Triana workflow editor [11]. A test-bed was created, based at three sites in the UK, Slovenia and Germany and at each site Globus components were used to interface with local Condor clusters. Condor is an opportunistic, high throughput computing system that uses cycle-stealing mechanisms to create a cluster out of idle computing resources [5].

Central to the grid infrastructure was an Application Description Schema (ADS) created specifically by the DataMiningGrid and which was used in all aspects of the grid system: for example to describe and register applications on the grid; dynamically configure the workflow editor; provide application technical requirements to resource brokering services; and to enable complex parameter sweep applications. Before deploying Evolver to the DataMiningGrid it was necessary to create an ADS template, which could then be instantiated during a workflow execution and eventually passed to the DataMiningGrid's resource broker service. The resource broker service is able to manage all aspects of the execution process and it is able to distribute the Evolver jobs to Condor pools in different administrative domains.

Three ADS were defined for two Evolver classes. The first Evolver class implements the evolutionary algorithm and this could be called in two different modes of operation: the molecular modelling stage, and the network modeling stage. Certain parameters were hard-coded into the ADS to differentiate between these stages, and two different ADS were created to make the application easier to use

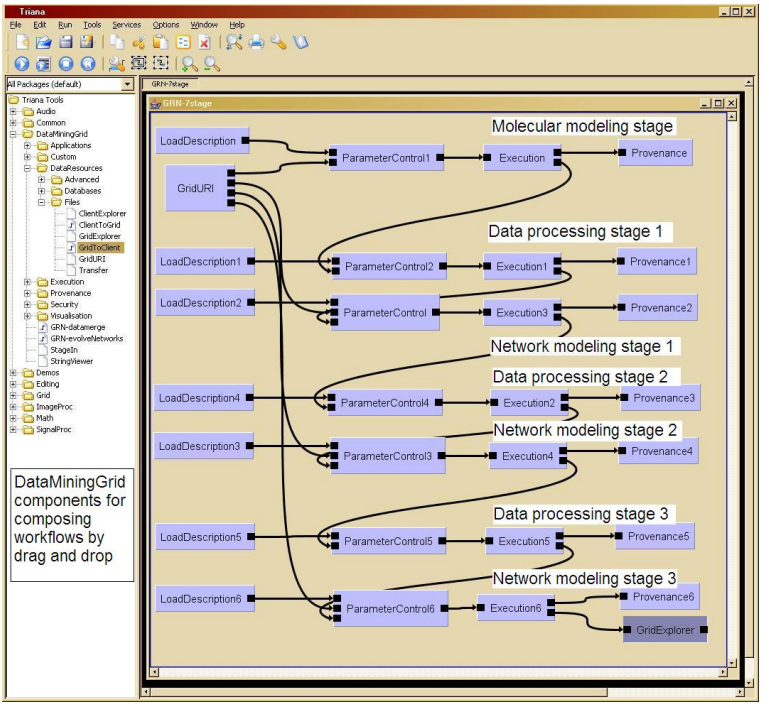

Fig. 2. Screen shot of a DataMiningGrid workflow for reverse-engineering genetic regulatory networks. The black lines show the input and output connections between components. The workflow stages are marked with white labels.

and clearer to understand. The second Evolver class processes the results generated by either modelling stage and it generates new sub-populations by merging the fittest individuals from evolved island populations.

The three ADS correspond to three workflow stages: *molecular modelling stage*; a quick *data processing stage*, here a number of the fittest parameter sets for each gene are collected on a storage server by the DataMiningGrid's resource broker service and combined in different ways to form populations of networks; and a *network modeling stage*.

In Fig. 2 we show how the workflow components can be easily combined together to provide workflows that allow individuals to be exchanged between island populations. This means that fewer generations are needed at each stage of the evolutionary process resulting in a faster overall execution time. In this example the data processing and network modelling stages are repeated three times (and are paired in Fig. 2), while the first or molecular modelling stage is performed just once. A workflow component specifying the location of input data is connected to each molecular modelling and network modelling stage. After the first network optimisation the results are again processed and island populations are regenerated from the fittest networks; this process is repeated

once more before the final reverse-engineered networks are obtained, viewed and downloaded.

In Fig. 3 we show the individual times taken for 90 test jobs to complete: these times are based on the time from when the DataMiningGrid's resource broker service first schedules the job until the job is finished. It is important to see in Fig. 3 that one job is an outlier, and takes almost twice as long than the other jobs, which all complete within 35 minutes. For this application such single outliers were encountered fairly regularly and had a serious effect, both on the performance of the algorithm in terms of execution times, and in terms of overall resource usage as most of the Condor pool is idle while waiting for this single job to complete. The cause of this outlier was not clear as it did not occur consistently, but is probably due to misconfigured software on the corresponding machine node.

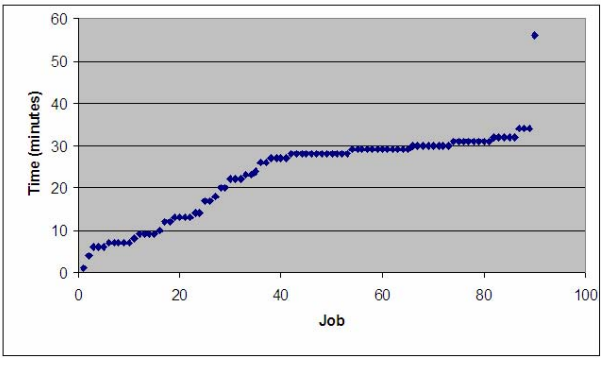

Fig. 3. This shows the time for each individual job to complete, ordered (or ranked) by length of time, for the execution of 90 jobs in the molecular modelling stage

Despite these occasional problems with the Condor pools, an advantage of the DataMiningGrid was the ability to simultaneously distribute Evolver jobs over different administrative domains and thus gain access to greater quantities of computational resources. The workflow editor was also useful, as this allowed easy experimentation and testing of variations of the basic workflow.

However, in this approach Evolver implemented the island-distributed evolutionary model by using files to migrate individuals between islands in a centralised manner. This is awkward: a better solution would be to migrate individuals from node to node by streaming the individuals' genotypes from one Java virtual machine to another, without writing any data to the file system at all. The centralised approach must be synchronised, so that every island must finish before any migration is possible, and this can cause problems when many nodes are used, especially if one of those nodes is unreliable. Misconfigured or unreliable machines can cause significant delays to the execution of the entire workflow.

3.2 The Quasi-opportunistic Supercomputing Approach: The QosCosGrid

Most grids, including the DataMiningGrid, are characterised by opportunistic sharing of resources as they become available. The quasi-opportunistic approach is different in that it aims to provide a higher quality of service by offering more dependable resources to its users. In addition, the QosCosGrid is specifically designed for parallel, distributed applications and it supports two important technologies for distributed applications, namely OpenMPI [12] and ProActive Java [2]. Two other important features of the QosCosGrid's functionality include the co-allocation of distributed resources, with fault-tolerance so that resource requirements can be guaranteed even in grids where resource availability is constantly changing; and the support of synchronous communications according to specific communication topologies, as typically required by various different complex systems simulations.

Although it was hoped that Evolver could be used on the QosCosGrid without any significant alterations, it had to be modified to ensure that Evolver objects were serialisable and so able to work with ProActive's Active Objects.

A master-worker architecture has been implemented with ProActive and is shown in Fig. 4. A master Active Object is used to control and monitor the worker Active Objects. Each worker uses an evolutionary approach to optimise either individual genes or networks of genes, according to the two-stage approach outlined in Sect. 2.3. The process is as follows:

1. The master sends out various input files to the different worker nodes, which are situated in different administrative domains, and calls a method at the worker nodes to begin the molecular modelling stage.
2. The workers notify the master on completion of a molecular modelling run and the master counts the number of such notifications. When all workers have completed, the master executes the data processing stage by gathering the fittest individuals from each worker node and sending out, to the workers, a list of all fit individuals that are to be compiled into a network populations.
3. The master node then activates network evolution by the workers.

When the workers complete their network modelling task, notification is performed by calling a master method: and the master, after performing the data processing task, calls the network modelling method on the workers. This process continues for a fixed number of iterations until the reverse-engineering process is complete.

The advantages of using ProActive are already evident. The iterative approach means that individuals can be exchanged between subpopulations more frequently. While this still currently relies on file-processing, the amount of data in the files is small (less than 100 Kb). To implement data exchange using messaging between Active Object involves implementing modifications to the Evolver source code, and this is something we have been avoiding as we expect to make a thorough redesign of the reverse-engineering approach in the future.

The problem of long-running machines can be overcome with the ProActive approach: the master node is notified when worker nodes complete their tasks,

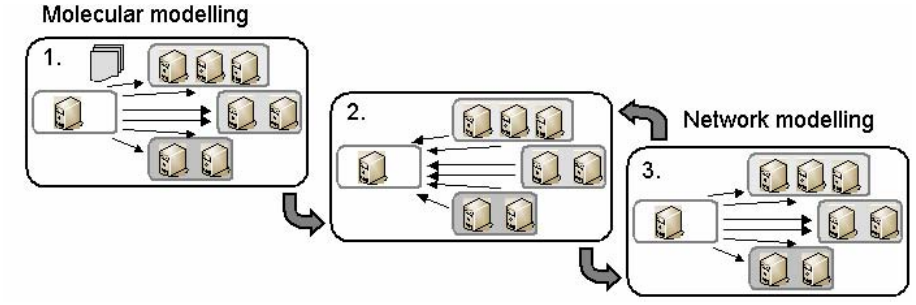

Fig. 4. The design of Evolver using ProActive. Looping occurs between stages 2. and 3. Each group of worker nodes is in a different administrative domain, the master is the single node.

and it counts the number of completed workers. Currently it waits until all workers are complete before initiating the next stage of the application, but it can be changed so that it only waits until 90% of all workers have completed.

It is possible to adapt the evolutionary algorithm parameters for each worker node. This can be very important in a heterogeneous environment, as different machines will run at different speeds. If the master node has data on the technical specification of each worker node, then by reducing population sizes or number of generations it can ensure that all workers complete at much the same time (unless they find a perfect solution and so exit before running all generations).

4 Discussion

There are many reverse-engineering methods for GRNs reported in the literature [13]. While a number of these are Java applications which may be distributed over local clusters, see for instance JCell [14], few have been designed for grids distributed over multiple, independent organisations, although one notable effort in this direction comes from the DARPA BioSPICE community [15]. An advantage of using grids is that the evolutionary optimisation process is easily parallelised and thus able to take advantage of distributed computing resources, greatly reducing the overall runtime.

A particular advantage of using the QosCosGrid infrastructure is due to its sophisticated control of communication, between clusters and nodes within clusters, and across administrative domains. This supports the development of more advanced applications, when compared to traditional grid and high-throughout computing systems such as the DataMiningGrid. For optimisation based on evolutionary algorithms, the QosCosGrid is suitable for fine-grained models such as cellular evolutionary models.

Table 1 summarises the features of Evolver, when deployed in each of the two grid computing platforms.

Table 1. Comparison of Evolver features, when implemented on two grid platforms

Platform	DataMiningGrid	QosCosGrid
Administrative domain	Multi	Multi
Opportunistic	Fully	Quasi
Service guarantee	no	yes
Fault tolerance	no	yes
Parallelisation	Coarse	Coarse or Fine
Island migration	Files	Active objects
Syncronisation	Synchronous	Synchronous

5 Future Work

The QosCosGrid is due to be completed early in 2009, by which time an extensive test-bed will have been deployed. This test-bed will have computational resources sufficient to explore more time-consuming GRN reverse-engineering tasks such as larger networks or collections of networks (i.e. cells). For these scenarios it is important that the evolutionary optimisation process is as efficient as possible. Hence we plan to use a purpose-built evolutionary algorithm toolkit, such as ParadisEO [16] which has a distributed version compatible with OpenMPI. Such toolkits include provide a quick way to test many different parallel implementations of evolutionary algorithms, and combine well with the QosCosGrid functionality.

6 Conclusions

In this paper a computationally intensive application for reverse-engineering gene regulatory networks has been implemented on two different distributed computing platforms. By comparing these technological approaches the advantages of quasi-opportunistic supercomputing, as implemented in the QosCosGrid, have been highlighted. These include support for different parallelisation strategies which allow sophisticated reverse-engineering approaches to be developed.

Acknowledgments. The work in this paper was supported by EC grants DataMiningGrid IST FP6 004475 and QosCosGrid IST FP6 STREP 033883.

References

1. Charlot, M., De Fabritis, G., Garcia de Lomana, A.L., Gomez-Garrido, A., Groen, D., et al.: The QosCosGrid project: Quasi-opportunistic supercomputing for complex systems simulations. Description of a general framework from different types of applications. In: Ibergrid 2007 conference, Centro de Supercomputacion de Galicia (GESGA) (2007)

2. Baduel, L., Baude, F., Caromel, D., Contes, A., Huet, F., et al.: Programming, Deploying, Composing, for the Grid. In: Grid Computing: Software Environments and Tools, Springer, Heidelberg (2006)
3. Stankovski, V., Swain, M., Kravtsov, V., Niessen, T., Wegener, D., et al.: Grid-enabling data mining applications with DataMiningGrid: An architectural perspective. Future Gener. Comput. Syst. 24, 259–279 (2008)
4. Foster, I.T.: Globus toolkit version 4: Software for service-oriented systems. J. Comput. Sci. Technol. 21, 513–520 (2006)
5. Litzkow, M., Livny, M.: Experience with the Condor distributed batch system. In: Proc. IEEE Workshop on Experimental Distributed Systems, pp. 97–100 (1990)
6. Wolkenhauer, O., Mesarovic, M.: Feedback dynamics and cell function: Why systems biology is called systems biology. Molecular Biosystems 1, 14–16 (2005)
7. Mandel, J.J., Fuss, H., Palfreyman, N.M., Dubitzky, W.: Modeling biochemical transformation processes and information processing with Narrator. BMC Bioinformatics 8 (2007)
8. Arbeitman, M.N., Furlong, E.E.M., Imam, F., Johnson, E., Null, B.H., et al.: Gene Expression During the Life Cycle of Drosophila melanogaster. Science 297, 2270–2275 (2002)
9. Swain, M., Hunniford, T., Mandel, J., Palfreyman, N., Dubitzky, W.: Reverse-Engineering Gene-Regulatory Networks using Evolutionary Algorithms and Grid Computing. Journal of Clinical Monitoring and Computing 19, 329–337 (2005)
10. Holland, J.H.: Adaptation in Natural and Artificial Systems. University of Michigan Press (1975)
11. Taylor, I., Shields, M., Wang, I., Harrison, A.: The Triana Workflow Environment: Architecture and Applications. In: Taylor, I., Deelman, E., Gannon, D., Shields, M. (eds.) Workflows for e-Science, pp. 320–339. Springer, New York (2007)
12. Coti, C., Herault, T., Peyronnet, S., Rezmerita, A., Cappello, F.: Grid services for MPI. In: ACM/IEEE (ed.) Proceedings of the 8th IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2008), Lyon, France (2008)
13. Arbeitman, M.N., Furlong, E.E.M., Imam, F., Johnson, E., Null, B.H., et al.: Gene Expression During the Life Cycle of Drosophila melanogaster. Science 297, 2270–2275 (2002)
14. Spieth, C., Supper, J., Streichert, F., Speer, N., Zell, A.: JCell—a Java-based framework for inferring regulatory networks from time series data. Bioinformatics 22, 2051–2052 (2006)
15. McCollum, J.M., Peterson, G.D., Cox, C.D., Simpson, M.L.: Accelerating Gene Regulatory Network Modeling Using Grid-Based Simulation. SIMULATION 80, 231–241 (2004)
16. Cahon, S., Melab, N., Talbi, E.G.: Building with paradisEO reusable parallel and distributed evolutionary algorithms. Parallel Comput 30, 677–697 (2004)