The Journal of Systems and Software 83 (2010) 1249–1257

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Bioinformatics algorithm development for Grid environments

Fotis E. Psomopoulos *, Pericles A. Mitkas

Department of Electrical and Computer Engineering, Aristotle University of Thessaloniki, Thessaloniki, 54 124, Greece

ARTICLE INFO

Article history:
Received 8 October 2009
Received in revised form 26 January 2010
Accepted 28 January 2010
Available online 4 February 2010

Keywords:
Bioinformatics
Data analysis
Grid computing
Workflow design
Protein classification
Semi-automated tool

ABSTRACT

A Grid environment can be viewed as a virtual computing architecture that provides the ability to perform higher throughput computing by taking advantage of many computers geographically dispersed and connected by a network. Bioinformatics applications stand to gain in such a distributed environment in terms of increased availability, reliability and efficiency of computational resources. There is already considerable research in progress toward applying parallel computing techniques on bioinformatics methods, such as multiple sequence alignment, gene expression analysis and phylogenetic studies. In order to cope with the dimensionality issue, most machine learning methods either focus on specific groups of proteins or reduce the size of the original data set and/or the number of attributes involved. Grid computing could potentially provide an alternative solution to this problem, by combining multiple approaches in a seamless way. In this paper we introduce a unifying methodology coupling the strengths of the Grid with the specific needs and constraints of the major bioinformatics approaches. We also present a tool that implements this process and allows researchers to assess the computational needs for a specific task and optimize the allocation of available resources for its efficient completion.

1. Introduction

Although computational biology and bioinformatics are often confused as the same interdisciplinary field, they do have several distinguishing differences. Bioinformatics is mainly concerned with the analysis and processing of data related to the biology of humans and other species. These data are being produced at huge rates by laboratories all over the world and necessitate the development of novel techniques and theories to solve formal and practical data management problems. On the other hand, computational biology aims to solve specific biological problems, utilizing computers to test and evaluate hypotheses.

Nonetheless, it must be also emphasized that "although bioinformatics and computational biology are distinct, there is also significant overlap and activity at their interface" (BISTIC, 2009). Proteomics is one of the key fields that flourish in that overlapping area. In a nutshell, proteomics is the large-scale study of proteins, ranging from the structural and functional analysis to the construction of protein–protein interaction networks and phylogenetic trees. Proteins are large organic molecules composed of aminoacids arranged in a linear chain and held together by peptide bonds. They constitute an essential part of organisms, participating in all processes within and between cells. For example, proteins catalyze biochemical reactions (enzymes), maintain the cell shape serving as scaffolds, provide the means of signaling between cells, etc. The term proteome denotes the entire complement of proteins expressed by a genome at a given time and under specific conditions. The word itself is a portmanteau of "protein" and "genome".

There has been a recent shift in research activity from genomics to proteomics, because scientists consider proteomics to be the next step in the study of biological systems. The genome of an organism is fairly stable, showing little variation throughout its cells in comparison with the proteome, which is highly differentiated from cell to cell. One of the more significant insights that have emerged from proteomics is the nature of relationship between genes and proteins. The study of the mouse proteome has demonstrated that a protein can be considered as the expression of not one but many genes (Gauss et al., 1999). Correspondingly, a single mutation in a gene can affect many proteins. Moreover, using the yeast proteome, the essential-essential protein interaction network has been proposed to form a generic scaffold around which organism-specific and taxon-specific proteins and interaction coalesce (Pereira-Leal et al., 2005).

The growing demand for automated analysis of large and distributed protein data poses new challenges to the available computational power and dictates the collaboration among scientists of different fields. The typical end user is mainly driven by concepts, issues, tasks and requirements arising from his or her application domain, and is usually not overly concerned with the specific characteristics of the resources available. Besides the formidable yet scarce supercomputing facilities, researchers have access to the Grid, an ever-expanding network of processing nodes. Indeed,

* Corresponding author. Tel.: +30 2310996349; fax: +30 2310996398.
E-mail addresses: fpsom@issel.ee.auth.gr (F.E. Psomopoulos), mitkas@eng.auth.gr (P.A. Mitkas).

0164-1212/$ - see front matter © 2010 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2010.01.051

according to Foster and Kesselman, *"a computational Grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities"* (Foster and Kesselman, 1998).

With the advent of Grid and application technologies, scientists and engineers are building increasingly complex applications to manage and process large data sets, and execute scientific experiments on distributed resources (Yu and Buyaa, 2005). Therefore, many efforts have been made towards the development of workflow management systems for Grid computing. A scientific workflow is the process of combining data and processes into a configurable, structured set of steps that implement semi-automated computational solutions of a scientific problem. A workflow is mainly concerned with the automation of the particular tasks of a scientific process, structuring them based on their control and data dependencies.

A workflow project can be analyzed into four distinct elements: (a) workflow design, (b) workflow scheduling, (c) fault-tolerance, and (d) data movement (Yu and Buyaa, 2005). Workflow design determines how workflow components can be defined and composed. Scheduling focuses on mapping and managing the execution of workflow tasks on shared resources that are not directly under the control of workflow systems. In any process, especially in the case of Grid environments, workflow execution can fail for various reasons, such as network failure, or non-availability of required software components. Thus, fault-tolerance techniques are essential in any workflow project. Finally, the staging of input / output files in large scale computations is a key component, especially for distributed systems.

Currently, there exist several Grid workflow approaches, such as Triana (Taylor et al., 2003), Taverna (Oinn et al., 2004), Pegasus (Deelman, 2003) and Kepler (Ludäscher et al., 2005) among others. Each has a unique approach to the overall workflow implementation; both Taverna and Pegasus utilize a DAG (*Directed Acyclic Graph*) workflow design, as opposed to the Non-DAG approach of Triana and Kepler. Taverna addresses the issue of data management with a centralized approach, by exchanging intermediate data between resources via a central point, whereas Pegasus utilizes a mediated approach, in which the locations of intermediate data are managed by a distributed data management system. Triana offers a decentralized workflow scheduling architecture, whereas the Kepler system is explicitly designed to allow for user extension. However, the majority of workflow systems have been designed to operate outside the Grid environment.

In this paper we present a novel theoretical workflow framework for data analysis in bioinformatics, which can operate entirely within the Grid. The framework aims to assist experts in the design of data analysis applications, and to enhance the application porting process. We also present a semi-automated tool developed to implement the theoretical workflow. The tool operates within the Grid infrastructure as well and is applied to a number of real-world use cases, demonstrating a significant increase in time and resource efficiency.

The structure of this article is as follows. Section 2 provides an overview of Grid computing in conjunction with Bioinformatics and presents the most common issues that associate with this infrastructure. Section 3 describes in detail the proposed approach to combine Grid with Bioinformatics applications, Section 4 presents the complexity analysis of the approach, and Section 5 provides three real-world case-studies. Conclusions and future work are presented in Section 6.

2. Grid computing and bioinformatics

In previous years, genomics and proteomics could only focus on a single gene or protein at a time. The continuous advancement in biosciences coupled with the availability of sufficient computing power have enabled a shift in research from hypothesis-driven to data-driven studies. This shift is presenting scientists with new challenges in data analysis. On one hand, most bioinformatics studies are performed on a limited number N of instances (i.e. samples or records). Usually N ranges from tens to hundreds of cases, as opposed to the thousands or millions of instances in a typical engineering or finance application. On the other hand, high-throughput techniques in life sciences yield several thousand variables per case, in sharp contrast to the traditional data mining scenarios. This problem is known as the curse of dimensionality, or small-N-large-P problem (Berrar et al., 2007). In proteomics data sets for instance, the number of variables-attributes P can be in the order of 10^4, whereas the number of cases-instances N is usually in the order of 10^2. In order to cope with this situation, most machine learning methods focus on specific cases or reduce either the size of the original data set or the number of attributes involved. Grid computing could potentially provide an alternative solution to this problem, by combining multiple approaches in a seamless way.

2.1. Grid computing

Although the term "Grid" has been conflated, at least in popular perception, to embrace anything from advanced networking to artificial intelligence, Grid computing has emerged as an important new field in computer science. In practice, a Grid environment can be viewed as a virtual computing architecture that provides the ability to perform higher throughput computing by taking advantage of several heterogeneous computers geographically dispersed and connected by a network. Grid computing is not equivalent to parallel computing. In a parallel computer many instructions are carried out simultaneously by multiple similar processors and parallelism can exist at different levels: bit, instruction, data, and task. In a Grid environment parallelism manifests itself in the form of computer clusters that are available as resources. However, since the resources on a Grid are connected via a comparatively low bandwidth network, Grid computing shows its true potential when dealing with embarrassingly parallel problems. The problems in this class can be decomposed into a large number of tasks which can run independently of each other. Therefore it can be argued that Grid computing is a superset containing the capabilities of both Parallel computing (utilizing the clusters available) and the ability to use geographically dispersed computer resources.

Since the majority of bioinformatics applications exhibit a high degree of parallelism, they can benefit from the increased availability, reliability, and efficiency of computing resources in a Grid environment. There is already considerable research in progress toward applying parallel computing techniques on bioinformatics methods, such as multiple sequence alignment (Data, 2006), gene expression analysis (Martino, 2006) and phylogenetic studies (Stamatakis, 2006) among others. Moreover, there is a recent trend in utilizing the Grid as a platform for complex approaches to bioinformatics problems, such as reverse-engineering gene-regulatory networks (Swain et al., 2005) and the Grid Basic Local Alignment Search Tool (Grid-Blast) (Krishnan, 2005), and in building specific frameworks over the Grid that target genomics and proteomics issues, such as the Biotechnology Information and Knowledge Grid (BioGrid) (Bata et al., 2002) and the Bioinformatics and Genomics Grid for European Research (BIG-GER) (Cameron, 2003). Moreover, there are several approaches in deploying generic machine learning approaches in a Grid environment, such as N2Grid for distributed neural networks (Schikuta and Weishäupl, 2004), however, they are still in an experimental phase.

It is evident from the ongoing research in the field that there is significant potential in employing the Grid as a means to enhance

most of the current approaches in bioinformatics and proteomics analysis studies. For example, the efficiency and overall performance of proteomics algorithms can be considerably boosted when existing data mining techniques are combined with the distributed environment of the Grid.

2.2. Challenges of Grid

Application programmers that decide to use the Grid need to be aware of a number of challenges, some of which are listed below:

1. *High-latency communications*: One of the main issues in Grid architectures is the loose connection of the geographically distributed resources. The poor inter-node communication performance, however, is mitigated by the prevailing parallel-programming formalisms, such as MPI, PVM and OpenMP, which are largely supported by tightly integrated parallel computers, such as clusters.
2. *Heterogeneity*: One of the distinguishing characteristics of a computational Grid is its heterogeneity. Although this may seem as an inevitable impediment, due to the need for the integration of diverse hardware/software setups and for future extensibility, heterogeneity also allows the execution of tasks on different resources, depending on suitability.
3. *Legacy applications*: Bioinformaticians typically use large collections of software tools and utilities (such as Rice et al. (2000)), designed for execution on traditional architectures. Mainly because such applications work well in their current environment, the users are often hesitant to port or re-implement their programs to make them Grid compatible.
4. *Middleware dependency*: Due to the fact that Grid technologies are still in a stage of infancy, a lot of competing middleware is being developed by different groups. This situation poses portability problems for developers when they have to move to other middleware implementations, and it slows down the development process, as the underlying middleware must be set-up, configured and tested to develop the code.

It is a universal truth that researchers prefer to use software modules they are familiar with, while harnessing as much computational power as they can, without having to worry about implementation details. The goal of this paper is to formalize the process of setting up a bioinformatics application for the Grid and thus to provide a level of abstraction between scientist and Grid. This layer, in the form of a workflow designer, aims to help the scientist (a) organize the execution of a computational experiment, (b) decide on the best allocation of resources, and (c) get a first-order estimate of the speed-up without having special knowledge of the Grid infrastructure. We will refer to this workflow as BADGE (*B*ioinformatics *A*lgorithm *D*evelopment for *G*rid *E*nvironments).

3. BADGE workflow

Five distinct phases emerge during the analysis of a typical bioinformatics process, which we will call an *experiment* from now on. The first phase is the Data Acquisition, where the scientist defines the source of the data involved. These data can be accessed in several ways, such as in the form of a file, through a database system or by means of a web-service interface. In any case, there is no limit to the number of sources included, provided that either their output is of the same format, or a format-converter is available. In practice, the latter case is not an issue, since most of the public databases provide both XML and FASTA formatted output. Moreover, it must be noted that, due to the nature of the Grid, when a job is terminated, the execution node is cleared of all data utilized. This means that when the same data have to be processed by several different jobs, they have to be fetched every time to the corresponding execution node even if that node has been used previously for the same process. In order to address this issue, the usual approach is to store data directly on the Grid as a grid resource. Obviously, the data have to be fetched in this case too, however the staging times will be significantly shorter.

The second phase of the workflow is Data Preprocessing, where the scientist defines whether the input data must be conditioned (cleared, normalized, etc) for the experiment. The third phase is Algorithm Configuration and it takes place only when the selected algorithm must be compiled and configured, before the actual execution. The mode of processing is defined in the fourth phase of the BADGE workflow, Application Execution, where the type of the code execution is set, either explicitly by the scientist, or implicitly by the algorithm compilation parameters. Finally, the fifth phase of the workflow, Evaluation Method, allows the scientist to declare any post-processing methods to be used in order to facilitate the evaluation of the results.

During execution of an experiment some tasks can proceed in parallel while others must run in sequence. In the general case each intermediate process in the experiment workflow can be decomposed into two parts: the serial and the parallel. When attempting to design a workflow, one must take into account the different processing paradigms that may exist in a Grid context:

1. *Single process applications*: This category includes use cases where the application consists of a single computationally intensive process. Programs in this category tend to show only slightly better response when used in a Grid environment, as opposed to a local workstation. However, they are the most commonly used, mainly due to the fact that they present easy Grid deployment for the end user.
2. *Multiple process applications*: This paradigm includes all use cases where the application can be analyzed into a number of repetitive processes, with little to no dependency between each repetition and process. Good examples of this category are the parameter sweep and single-program-multiple-data applications. These applications tend to be embarrassingly parallel in nature and they usually offer maximum flexibility to the end user, allowing for multiple experimental setups to be executed simultaneously. However, they often require some knowledge on the part of the end user for the submission to the Grid.
3. *Parallel applications*: This class includes all use cases that exhibit fine-grained parallelism. Typically they are MPI implementations of parallel algorithms and they require a cluster environment in order to be executed. A characteristic member of this category is the parallel CLUSTALW application. Despite the fact that these applications are the most efficient among the three categories in terms of processing time, they are also the most rarely used. The main reason is that they require special implementation, and therefore special knowledge on the part of the end user. Although there are several algorithms available, end users still have to know how to approach such methods.

It is obvious that the ideal scenario would combine the flexibility of the Multiple Process class, with the efficiency of the Parallel category and the ease-of-use of the Single Process paradigm. Although the whole combination is fairly difficult to achieve, it is within reason to combine flexibility with convenience for the end user. In any case, it can be argued that running several different configurations simultaneously on several remote units instead of sequentially on a local unit, can realize a huge gain.

The objective of BADGE is to simplify the process of submitting bioinformatics applications on a Grid environment for the end

Table 1
BADGE Workflow Phases.

#	Phase	Possible options
1	Data acquisition	a. File (Local Grid) b. Database access (User Public) c. Application access (Web services)
2	Data preprocessing	a. Data de-noise b. Feature selection (e.g. PCA, SVD) c. No preprocessing
3	Algorithm configuration – Compilation step – Configuration step	common compiler parameters
4	Application execution	a. Single process b. Multiple processes c. Parallel (MPI)
5	Evaluation method	a. Testing process b. Iterative process

user, in an effort to increase the overall efficiency. An overview of the main phases in the workflow is presented in Table 1. It must be noted that there is no explicit failure management in the workflow. Since the BADGE workflow is designed to be executed within the Grid infrastructure, the issue of handling failures is delegated to the grid middleware. The most common technique available in the majority of middleware is the combination of *retry* and *alternate resource*. Coupled with the fact that each phase of the BADGE workflow is distinct, failure at any point during execution is addressed by simply retrying or migrating to an alternate resource after a user-defined number of attempts.

The BADGE workflow can be utilized by software designers as a theoretical framework in order to develop Grid-enabled applications for bioinformatics efficiently and with less effort. In addition, an implementation of this approach can be utilized as a tool to facilitate the use of Grid resources by non-experts. The developed BADGE tool follows the same structure of the BADGE workflow, and allows the gradual construction of the necessary files for job submission to the Grid through a series of options (such as input data source, path of the application, etc.).

Listing : Internal Representation of a Bioinformatics Application using the BADGE tool.

```
AppType = "Multiple";
NumCPU = 10;
PreProc = "NaN";
Class = {"Family"};
Algorithm = "weka.classifiers.trees.J48";
LocalData = {"Prosite.arff"};
RemoteData = {"lfn:/grid/see/fpsom/weka.jar"};
Output = {"gClass_models.tar.gz"};
Eval = {{"cross-validation", 10}};
```

The BADGE tool has been implemented as a Grid resource available at the User Interface level of the Grid environment. Although it has been designed as a general purpose tool, the current implementation is dependent on the gLite middleware which is the base of the EGEE Grid Environment Grids for E-sciencE. The BADGE tool is reminiscent of a "wizard" application, where the user sets the preferred parameters for the specific process, and the tool constructs and submits the necessary files for the job. Listing : shows an example of the internal representation of the BADGE tool for a workflow that consists of a multiple process application (*AppType* field) which will be executed simultaneously on 10 CPUs (*NumCPU* field). There is no preprocessing phase required (*PreProc* field), and the specific application is the classifier J48 (*Algorithm* field). Due to the algorithm selected the field *Class* is also required. The data staging is defined by the three fields *LocalData*, *RemoteData* and *Output*. Finally, the user requires also an evaluation phase, thus setting the parameters in field *Eval*.

4. BADGE complexity analysis

In this section we present a first-order complexity analysis for the BADGE workflow. Defining $F_i(\hat{n}_i), i = 1, 2, \ldots, 5$, as the time complexity of each of the five steps in BADGE, the total complexity $F_s(n_t)$ of any given process executed in a single machine, would be equal to:

$$F_s(n_t) = \sum_{i=1}^{5} F_i(\hat{n}_i) \tag{1}$$

where n_t is a function of $\hat{n}_i, i = 1, 2, \ldots, 5$.

However, instead of executing the whole process as a sequential procedure, several steps can be further analyzed to allow for sequential and parallel parts. The complexity $F_1()$ of step 1 (Data Acquisition) can be re-written as follows:

$$F_1(n_1, p_1) = \mathcal{I}_s^d(n_1) + \frac{\mathcal{I}_p^d(n_1)}{p_1} \tag{2}$$

where $\mathcal{I}_s^d(n_1)$ is the part of the Data Acquisition process that is necessarily sequential, $\mathcal{I}_p^d(n_1)$ is the fraction of the process that can be executed concurrently, and p_1 is the number of processors that will be utilized in the parallel case (in the context of data access, n_1 usually refers to the number of data blocks). Obviously, the worst case scenario is when there is no concurrent part (i.e. $\mathcal{I}_p^d(n_1) = 0$) which means that all data blocks n_1 have to be processed sequentially.

The preprocessing step can be decomposed into a serial and a concurrent fraction in a similar way. If $F_2()$ is the complexity of the second step, then the function decomposition could be as follows:

$$F_2(n_2, p_2) = \mathcal{I}_s^r(n_2) + \frac{\mathcal{I}_p^r(n_2)}{p_2} \tag{3}$$

Once again, the worst case scenario is when no part of the preprocessing algorithm can be executed in parallel (i.e. $\mathcal{I}_p^r(n_2) = 0$). Parameter n_2 is generic, due to the fact that it depends on the specific algorithm utilized in this phase.

The Algorithm Configuration phase (step 3) has been extensively studied regarding time complexity. Given n_3 lines of code, an assembler would require nearly $O(n_3)$ time (assuming that symbol table lookup can be done using a standard hashing function). For higher level languages, such as C++ or Java, the process is much more elaborate. Complexity ranges from $O(n_3 \times log(n_3))$ for a fast parsing method (eg single-sweep attribute grammars) and a complete semantic structure in the language, to $O(n_3^2)$ when introducing the graph problems in optimizers. Without loss of generality, the complexity of the third step can be written as follows:

$$F_3(n_3) = f(n_3^2 \times log(n_3)) \tag{4}$$

The fourth step in BADGE is essentially the actual execution of the program in the Grid environment. Although in theory there are three different processing paradigms, most bioinformatics approaches embrace the Multiple Process paradigm. A plethora of bioinformatics algorithms exhibit fine-grained parallelism, but it is common knowledge that the majority of experiment designs require multiple runs of the same algorithm with different parameter setup and the same input. So, despite the fact that the complexity of this step in theory is similar to Eq. (5), the actual complexity $F_4(n_4, M)$ is given by Eq. (6).

$$F_4^{theory}(n_4, p_4) = \mathcal{I}_s^e(n_4) + p_4 \times \mathcal{I}_p^e(n_4) \quad (5)$$

where p_4 is the fraction of the program that can be executed in parallel.

$$F_4(n_4, p_4, M) = M \times (\mathcal{I}_s^e(n_4) + p_4 \times \mathcal{I}_p^e(n_4)) \quad (6)$$

where M is the number of iterations of the whole program execution.

Finally, the complexity of the fifth step (Evaluation phase) can be expressed in a similar way to the Data Preprocessing phase, as shown in Eq. (7).

$$F_5(n_5, p_5) = \mathcal{I}_s^v(n_5) + \frac{\mathcal{I}_p^v(n_5)}{p_5} \quad (7)$$

At this point, one should also take under consideration the overhead that is always present in every task submission to a Grid environment. Although this overhead tends to be diminished as the middleware is constantly improved, it is well known that the overhead time rises linearly with the number of CPUs (p_0) requested (Eq. (8)):

$$F_0(p_0) = \mathcal{I}^o(p_0) \quad (8)$$

By substituting the complexity functions of each step in Eq. (1), the total complexity of any given process in BADGE and on a single machine will be the following:

$$\begin{aligned} F_s(n_t) = {} & \mathcal{I}_s^d(n_1) + p_1 \times \mathcal{I}_p^d(n_1) + \mathcal{I}_s^r(n_2) + p_2 \times \mathcal{I}_p^r(n_2) \\ & + f(n_3^2 \times log(n_3)) + M \times (\mathcal{I}_s^e(n_4) + p_4 \times \mathcal{I}_p^e(n_4) \\ & + \mathcal{I}_s^v(n_5) + p_5 \times \mathcal{I}_p^v(n_5) \end{aligned} \quad (9)$$

where $n_t = g(n_1, n_2, n_3, n_4, n_5), p_1, p_2, p_3$ and p_4 are internal algorithm parameters, and M is a user-defined parameter. On the other hand, the complexity of a given process in BADGE over the Grid would be as follows:

$$\begin{aligned} F_{grid}(n_t, p_{nodes}) = {} & \mathcal{I}^o(p_{nodes}) + \mathcal{I}_s^d(n_1) + \frac{\mathcal{I}_p^d(n_1)}{p_1} + \mathcal{I}_s^r(n_2) \\ & + \frac{\mathcal{I}_p^r(n_2)}{p_2} + f(n_c^2 \times log(n_3)) + \frac{M}{p_{nodes}} \times (\mathcal{I}_s^e(n_4) \\ & + p_4 \times \mathcal{I}_p^e(n_4) + \mathcal{I}_s^v(n_5) + \frac{\mathcal{I}_p^v(n_5)}{p_5} \end{aligned} \quad (10)$$

If $p_{nodes} = M \approx p_1 \approx p_2 \approx p_5$, which is the case in practice, then the speedup will be optimal, with respect to the serial approach. Note that p_{nodes} should not exceed M because in this case some nodes will be idle. On the other hand, if $M = 1$ and there exist no parallel parts in any step of the workflow, then the Grid approach will be marginally worse than the serial one. However, in practice the vast majority of bioinformatics approaches are mainly iterative processes, therefore allowing for a significant gain in execution time.

5. Results

In order to evaluate the workflow discussed in the previous sections, three use cases have been developed and deployed over the EGEE Grid Environment Grids for E-sciencE. The EGEE Grid consists of 41,000 CPU in addition to about 5 PB of disk storage and can potentially maintain 100,000 concurrent jobs. In order to avoid repetition, the tools and algorithms mentioned in the use cases are physically located in Storage Elements on the EGEE and retrieved directly on demand, unless stated otherwise.

The three use cases have been selected so as to include representative examples of all real-world applications, i.e. applications that are necessarily single process but are usually executed several times (*Use Case 1*), applications that are single process by default but can be easily transformed to multiple process (*Use Case 2*), and applications that are parallel and require a cluster (MPI) environment for execution (*Use Case 3*).

5.1. Use case 1–GridBLAST

The first use case involves a very common bioinformatics task, the execution of BLAST for a number of protein sequences. For each sequence, BLAST tries to determine the best alignment (similarity) against the proteins stored in a large reference database. The run for each sequence is independent from all others, so all the runs can be executed in parallel given sufficient hardware. Since BLAST is not available on every Grid node, for every run the program has to be fetched and installed in the corresponding node. The SWISSPROT database has also to be fetched.

In the experiments, BLAST version 2.2.19 was utilized, and 100 protein sequences randomly selected from SWISSPROT, ranging from 500 to 2500 aminoacids, where used for evaluation. Specifically, in each node the FASTA version of the SWISSPROT database was used (Release 57.5, 470,369 entries), which was consequently transformed to the BLAST accepted format, using the *formatdb* command (default parameters). The *blastall* application was executed using the default parameters for the *blastp* algorithm on the subset of the data assigned to the node (in each case data was split evenly among the nodes of the experiment).

Table 2 presents the execution time for the different number of nodes requested. Due to the fact that BLAST is provided as a precompiled binary file, the times presented do not include compile time. Obviously, this type of problem is a perfect candidate for execution on a grid.

5.2. Use case 2–g-Class

5.2.1. Algorithm outline

g-Class is a divide-and-conquer data mining methodology which utilizes existing data mining algorithms in a parallel-enabled environment in order to create a single protein classification model (Polychroniadou et al., 2006). Using motif-based representation of the protein sequences, the goal is to evaluate the effect of data parallelization on the overall performance, accuracy and efficiency of the final classifier. Initially the original dataset containing the known (classified) protein sequences is split into several smaller and disjoint datasets, that maintain the class distribution of the original dataset. Each of these datasets is consequently used as a training dataset for the creation of a classification model. The training process can be performed by any applicable classification algorithm, thus allowing for greater flexibility for the user. Since the construction of each classifier model is independent from each other, the different processes can execute in parallel. In the end, the classifiers are combined into a single model, which is used for the evaluation of the whole methodology and the classification of new and unknown protein sequences.

The g-Class methodology, whose workflow diagram can be seen in Fig. 1, comprises three steps:

(1) Data splitting
Split the original input dataset into n disjoint subsets.
(2) Classifier training
Using the n subsets, utilize a classification algorithm in order to construct n classification models.

Table 2
BLAST execution time.

Number of nodes	Execution time (s)	Speedup
1	4768	1.00
2	2335	2.04
4	1214	3.93
8	608	7.84

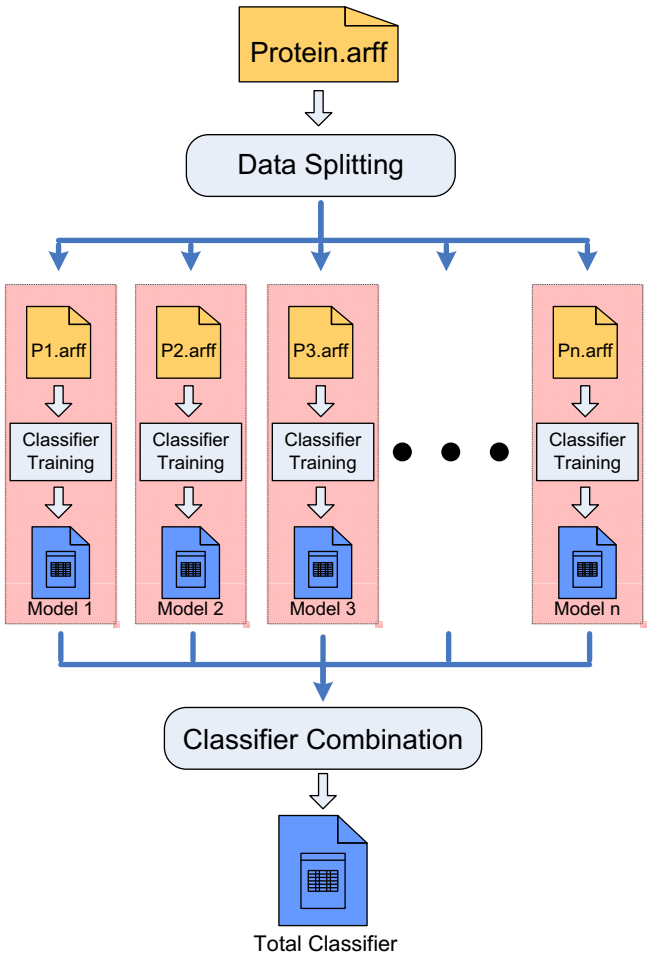

Fig. 1. G-Class experiment design workflow.

(3) Classifier combining
Combine the n models of the previous step, in order to produce the final ruleset.

5.2.2. g-Class complexity

Following the five steps of the BADGE methodology, the overall complexity of the g-Class for the serial and the parallel case can be estimated. The results are presented in Table 3.

In this Table the overall complexity of the serial approach $(O(n+f(n)+g(m)))$ seems to be comparatively lower than the g-Class approach $(O(n\times p+f(\frac{n}{p})+g(m)))$. Since the parallel approach is definitely faster, as verified by experimental results presented later in this section, we have to analyze the performance of the algorithm in more detail. First of all, the number of instances n in a typical dataset is usually in the order of tens of thousands, where the number of processes p used (and each process is a classification model in the case of g-Class) is usually in the order of tens, i.e. $n \gg p$. Moreover, the execution time of the majority of the classification algorithms rises (often exponentially, especially in the case of main memory algorithms) with the number of instances n in the training set, i.e. $f(n) \gg f(\frac{n}{p})$. Based on these observations, it is fairly obvious that the major factor in the overall complexity in both cases is function $f()$, which denotes the execution time of the classification algorithm, thus leading to the conclusion that the g-Class approach presents overall lower time complexity.

Table 3
g-Class complexity comparison

Step	Serial approach	Parallel approach
Data acquisition	$O(n)$	$O(n)$
Data preprocessing	–	$O(n\times p)$
Algorithm configuration	–	–
Application type	$O(f(n))$	$O\left(f\left(\frac{n}{p}\right)\right)+O(n\times p)$
Evaluation method	$O(g(m))$	$O(g(m))$
Overall	$O(n+f(n)+g(m))$	$O(n\times p+f\left(\frac{n}{p}\right)+g(m))$

5.2.3. g-Class results

The first phase in deploying the whole process on the EGEE Grid is to define the input data source. The input dataset is retrieved from PROSITE (Hoffmann et al., 1999), a supervised motif database. In this case, the data are located on a local flat file, reformatted from the PROSITE data format to ARFF (Attribute Relation File Format) used by the WEKA (Witten and Frank, 2005) data mining software package.

In the ARFF representation of the PROSITE database, each motif is transformed to a binary attribute, and all protein families are assigned to a *Family* attribute. Thus, every protein in the database is represented as a binary vector data instance, where a value of 1 in an attribute is translated as the occurence of the corresponding motif in the specific protein sequence, and a value of 0 denotes the absence of the motif.

One of the goals of the procedure is to cover the entire dataset, therefore in the next step no data preprocessing methods are specified. However, the sheer amount of data involved makes it virtually impossible to create a single classification model within a reasonable time limit. This in turn implies that the application should be deployed not as a single process, which is the default choice of standard classification algorithms, but as a multiple process. The choice in application type is reinforced by the fact that PROSITE database contains numerous protein families (approximately 1100), which leads to the creation of multiple classifiers and eventually the combination of their outputs. In order to ensure that all processes have access to the same amount of information, and therefore to increase the efficiency of the combination procedure, the original dataset is automatically split into M disjoint datasets, where M is the number of processes spawned.

Some results, using various subsets of the original data and different number of splits, are shown in Fig. 2. As is evident from the diagrams, a significant gain in execution time can be achieved with the increase of the number of subsets, without any loss in the overall classifier accuracy (Polychroniadou et al., 2006). The specific execution times and speedup in each case are presented in Table 4 (it must be noted that, in the case of the larger datasets, the application could not be executed on a single CPU and therefore the overall speedup is calculated based on the 2-CPU execution times).

5.3. Use case 3-Proteas

5.3.1. Algorithm outline

Proteas is a novel parallel methodology for protein function prediction (Gkekas et al., 2008). Data mining techniques are employed in order to construct a model for each Gene Ontology (GO)[1] term, based on data generated from already annotated protein sequences. The models created can then be used to predict the annotation of

[1] The Gene Ontology project is a major bioinformatics initiative with the aim of standardizing the representation of gene and gene product attributes across species and databases. The project provides a controlled vocabulary of terms for describing gene product characteristics and gene product annotation data, as well as tools to access and process this data. (Ashburner et al., 2000; Gene Ontology)

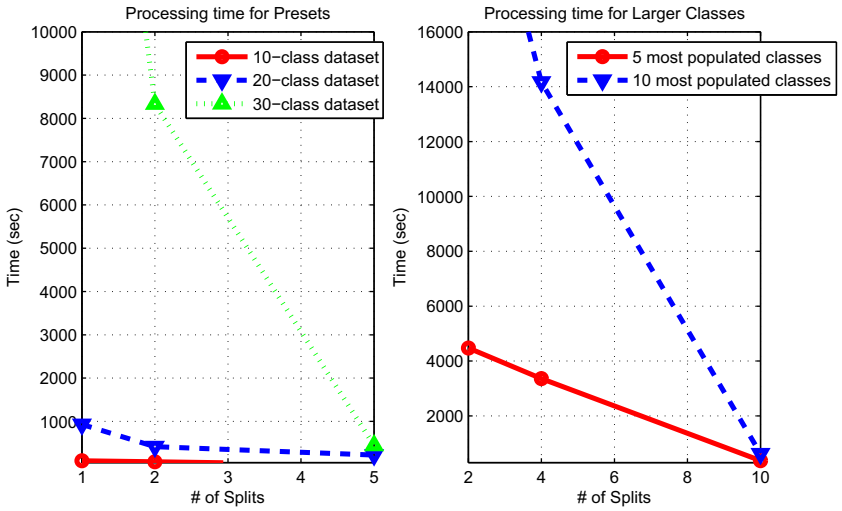

Fig. 2. G-Class experimental results.

Table 4
g-Class execution times

Presets					Larger datasets			
Number of splits	Execution time (s)			Speedup (avg)	Number of splits	Execution time (s)		Speedup (avg)
1	90	930	21600	1.00	2	4474	24406	1.00
2	68	415	8334	2.05	4	3360	14168	1.53
5	2	223	444	32.06	10	370	617	25.82

new protein sequences. In more detail, the motif sequence of the protein under examination is extracted and run through all available Gene Ontology term models. This process generates similarity scores, which constitute an accurate prediction of the protein's annotation.

A set of protein data, in the form of XML files, is used as input in the Proteas methodology. Each file contains information regarding motif sequence, GO terms and other characteristics of a specific protein. These data are retrieved directly in XML format using the InterProScan tool (Zdobnov and Apweiler, 2001; InterProScan Tool) which acts as an interface between users and a number of protein databases. The data are initially split into two disjoint sets, the Training and the Test set. The Training set is used for the construction of one model for each of the available GO terms, while the Test set is used for the evaluation of the whole process.

In order to create a model for a specific GO term, only the proteins in the Training set that contain this term are taken under consideration. These protein instances are then used for the construction of a Prefix Tree Acceptor (PTA). In the next step, the PTA is transformed into a Stochastic Finite State Automaton, which is the objective model of the respective term. The final step is the evaluation of the models using the protein instances in the Test set; for each protein a probability vector is created by assigning each term-model with a probability that the protein sequence may be described by the specific term. By using a threshold filter, the most probable GO terms can be extracted for each protein instance.

The key point in this case-study is the custom MPI-enabled classification algorithm. It has been designed focusing on the protein classification problem, its constraints and idiosyncracies. The main advantage is flexibility in terms of usability and data access. However, it still shares some of the drawbacks of parallel applications, as mentioned previously, which means that its configuration, beyond the basic parameters, requires the end user to have advanced parallel programming knowledge. The workflow diagram of the Proteas methodology is shown in Fig. 3.

5.3.2. Proteas complexity

The Proteas methodology was designed and implemented having in mind the advantages and disadvantages of a Grid environment. Using the steps described in BADGE, the complexity analysis of Proteas is presented in Table 5. For the complexity analysis the following symbols have been used:

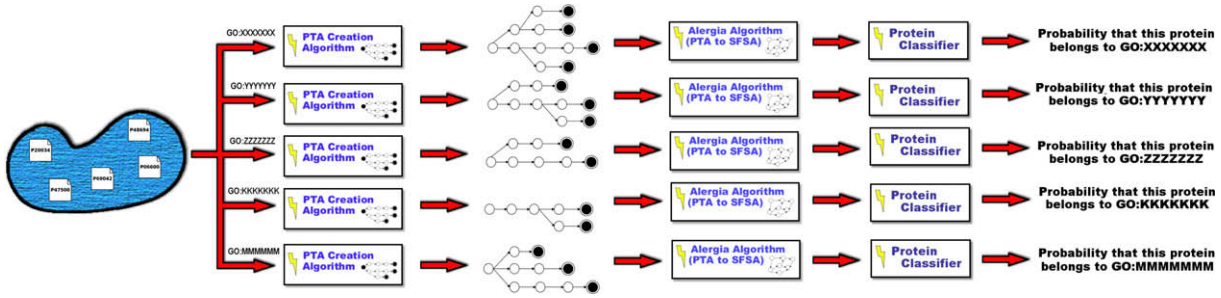

Fig. 3. Proteas workflow diagram.

- n is the number of protein instances.
- g is the number of Gene Ontology terms.
- m is the number of motifs in a given protein sequence.
- I is the total number of motifs that can appear in a protein sequence.

A number of assumptions can be made. First of all, both the number of motifs m in a sequence and the overall number of motifs I can be considered fairly constant across different runs (in practice, I changes with every major database update, and m ranges mostly between 4 and 50 with an average around 20). Also, based on experimental results (Carrasco and Oncina, 1994) the overall complexity of the Proteas methodology can be evaluated as shown in Table 6.

5.3.3. Proteas results

As mentioned previously, Proteas utilizes protein data from all databases available through InterProScan (i.e. ProDom, PRINTS, PIR, PFAM, SMART, TIGRFAMs, PROFILE, PROSITE, SUPERFAMILY, SIGNALP, TMHMM, PANTHER, GENE3D - May 2009). The number of protein sequences available was 39211, across 1712 Gene Ontology terms. For all experiments performed, each protein was compared against all 1712 Gene Ontology term models thus producing a 1712-value probability vector in each case.

Table 5
Proteas worst-case scenario complexity

Step	Complexity
Data acquisition	$O(n)$
Data preprocessing	$O(n \times g)$
Algorithm configuration	–
Application type	$O(n \times m \times I) + O(f(n^3))$
Evaluation method	$O(n \times g \times (2^m - 1) \times e^m)$
Overall	$O(n \times (g + m \times I + g \times (2^m - 1) \times e^m)) + O(f(n^3))$

Table 6
Proteas Complexity

Step	Complexity
Data acquisition	$O(n)$
Data preprocessing	$O(n \times g)$
Algorithm configuration	–
Application type	$O(n) + O(f(n))$
Evaluation method	$O\left(\frac{n \times e^m}{p}\right)$
Overall	$O\left(n \times g + f(n) + \frac{n \times e^m}{p}\right)$

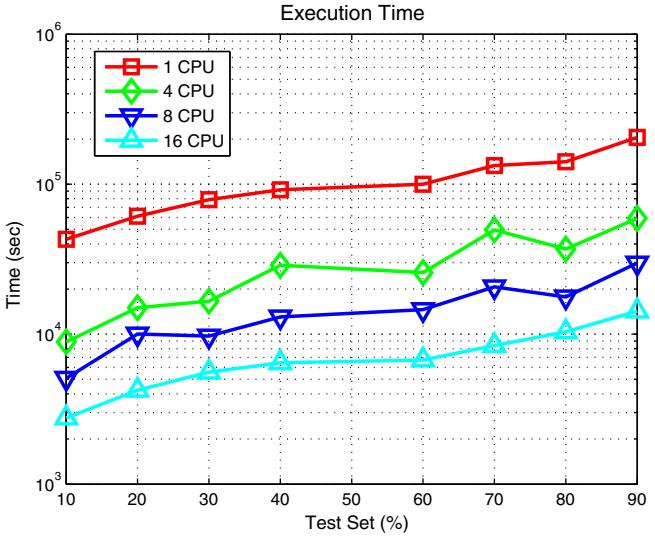

Fig. 4. Proteas execution time.

Using the BADGE methodology for the configuration and submission of the jobs on the Grid, the time efficiency of Proteas was evaluated. The diagram in Fig. 4 shows the processing time over a dataset including approximately 20,000 protein sequences, using different number of CPUs. In each case the same input dataset was used, but different ratios of Training/Test sets. Finally, each experiment was repeated several times, and the mean execution time (including compile time) is shown in Fig. 4.

From the diagram in Fig. 4 it is evident that there is an overall gain in execution time, especially when using 8 or 16 CPUs.

6. Conclusion

The BADGE workflow presented here is only the first step towards bridging the gap between the end user's perceptions and expectations, and the complexity of real world data mining applications over a grid environment. There is considerable ongoing research toward the development of more efficient and accurate workflow paradigms with emphasis on domain-specific solutions. Since the majority of these solutions have a somewhat limited scope, there still remains the goal of unifying the different viewpoints. Admittedly the distance among them tends to shrink, at least in recent years, however, the main issue remains unresolved.

In this paper a novel theoretical workflow framework is studied, which aims to assist experts in the design of data analysis applications for use within the Grid. Moreover, a semi-automated tool which incorporates the results of the study, is presented through a number of real-world use cases, demonstrating a significant increase in time and resource efficiency.

The presented prototype can be perceived as a first step toward making the Grid an easily accessible resource in every bioinformatics study. Despite the fact that the scientific community has realized the need for transparent and, especially, user-friendly access to technologies and methodologies for high-throughput experimentation, progress is hampered by the fact that scientists often work in isolation. Future developments in the area of seamless user interaction with the Grid will certainly overcome such difficulties and present a whole new way in how grids are used in life sciences.

References

Ashburner, M., Ball, C.A., Blake, J.A., Botstein, D., Butler, H., Cherry, J.M., Davis, A.P., Dolinski, K., Dwight, S.S., Eppig, J.T., Harris, M.A., Hill, D.P., Issel-Tarver, L., Kasarskis, A., Lewis, S., Matese, J.C., Richardson, J.E., Ringwald, M., Rubin, G.M., Sherlock, G., 2000. The gene ontology consortium, gene ontology: tool for the unification of biology. Nature Genetics 25, 25–29.

Bata, P., Alessandrini, V., Girou, D., MacLaren, J., Brooke, J., Pytlinski, J., Nazaruk, M., Erwin, D., Mallmann, D., Myklebust, J.F., 2002. BIOGRID-A European grid for molecular biology. In: Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing (HPDC-11), Edinburgh, Scotland.

Berrar, D., Granzow, M., Dubitzky, W., 2007. Introduction to genomic and proteomic data analysis. In: Dubitzky, W., Granzow, M., Berrar, D.P. (Eds.), Fundamentals of Data Mining in Genomics and Proteomics. Springer Science, NY, USA, pp. 1–37.

Working definition of bioinformatics and computational biology, National Institutes of Health, BISTIC (Biomedical Information Science and Technology Initiative Consortium, http://www.bisti.nih.gov/CompuBioDef.pdf (Last Access: Mar 2009).

Cameron, G., 2003. Bioinformatics and genomics grid for European research (BIGGER). In: Proceedings of the First European Health Grid Conference (HealthGrid'03), Lyon, France.

Carrasco, R. C., Oncina, J., 1994. Learning stochastic regular grammars by means of a state merging method. In: Springer Lecture Notes in Artificial Intelligence LNAI (Proceedings of the Second International Colloquium on Grammatical Inference - ICGI'94), vol. 862, Alicante, Spain, pp. 139–152.

Data, A., 2006. Multiple sequence alignment in parallel on a cluster of workstations. In: Zomaya, A. (Ed.), Parallel Computing for Bioinformatics and Computational

Biology, Wiley Series on Parallel and Distributed Computing. Wiley-Interscience, New Jersey, USA, pp. 193–210.
Deelman, Ewa et al., 2003. Mapping abstract complex workflows onto grid environments. Journal of Grid Computing 1 (1), 25–39.
Foster, I., Kesselman, C., 1998. The Grid: Blueprint For a New Computing Infrastructure. Morgan Kaufman Publishers.
Gauss, C., Kalkum, M., Lowe, M., Lehrach, H., Klose, J., 1999. Analysis of the mouse proteome. (I) Brain proteins: separation by two-dimensional electrophoresis and identification by mass spectrometry and genetic variation. Electrophoresis 20, 575–600.
Gene Ontology project, http://www.geneontology.org/.
Gkekas, Christos N., Psomopoulos, Fotis E., Mitkas, Pericles A., 2008. A Parallel Data Mining Application for Gene Ontology Term Prediction, Presented at the 3rd EGEE User Forum, Clermont-Ferrand, France.
Enabling Grids for E-sciencE, http://www.eu-egee.org.
Hoffmann, K., Bucher, P., Falquet, L., Bairoch, A., 1999. The PROSITE database, its status in 1999. Nucleic Acids Research 27, 215–219.
InterProScan Tool, http://www.ebi.ac.uk/InterProScan.
Krishnan, A., 2005. GridBLAST: a globus-based high-throughput implementation of BLAST in a grid computing framework. Concurrency and Computation: Practice and Experience 17 (13), 1607–1623.
Ludäscher, B. et al., 2005. Scientific workflow management and the Kepler system. Concurrency and Computation: Practice & Experience 18 (10), 1039–1065.
Martino, R.L., 2006. Parallel computing in the analysis of gene expression relationships. In: Zomaya, A. (Ed.), Parallel Computing for Bioinformatics and Computational Biology, Wiley Series on Parallel and Distributed Computing. Wiley-Interscience, New Jersey, USA, pp. 265–284.
Oinn, Tom et al., 2004. Taverna: a tool for the composition and enactment of bioinformatics workflows. Bioinformatics 20 (17), 3045–3054.
Pereira-Leal, J.B., Audit, B., Peregrin-Alvarez, J.M., Ouzounis, Christos A., 2005. An exponential core in the heart of the yeast protein interaction network. Molecular Biology and Evolution 22 (3), 421–425.
Polychroniadou, Helen E., Psomopoulos, Fotis E., Mitkas, Pericles A 2006. G-Class: A divide and conquer application for grid protein classification. In: Proceedings of the second ADMKD 2006: Workshop on Data Mining and Knowledge Discovery (in conjunction with ADBIS 2006: The 10th East-European Conference on Advances in Databases and Information Systems, Thessaloniki, Greece, pp. 121–132.
Rice, P., Longden, I., Bleasby, A., 2000. EMBOSS: the european molecular biology open software suite. Trends in Genetics 16, 276–277.
Schikuta, Erich., Weishäupl, Thomas., 2004. Artificial neural networks and the grid. In: Lecture Notes in Computer Science, Computational Science-ICCS 2004, vol. 3036, pp. 486–489.
Stamatakis, A., 2006. Parallel and distributed computation of large phylogenetic trees. In: Zomaya, A. (Ed.), Parallel Computing for Bioinformatics and Computational Biology, Wiley Series on Parallel and Distributed Computing. Wiley-Interscience, New Jersey, USA, pp. 327–346.
Swain, M., Hunniford, T., Mandel, J., Palfreyman, N., Dubitzky, W., 2005. Reverse-engineering gene-regulatory networks using evolutionary algorithms and Grid computing. Journal of Clinical Monitoring and Computing 19 (4–5), 329–337.
Taylor, I., Shields, M., Wang, I., 2003. Resource Management of Triana P2P Services, Grid Resource Management. Kluwer, Netherlands.
Witten, I., Frank, E., 2005. Practical Machine Learning Tools and Techniques, second. Morgan Kaufmann.
Yu, J., Buyaa, R., 2005. A taxonomy of scientific workflow systems for grid computing. SIGMOD Record 34 (3), 44–49.
Zdobnov, E.M., Apweiler, R., 2001. InterProScan: protein domains identifier. Nucleic Acids Research 17, 847–848.

Pericles A. Mitkas received his Diploma of Electrical Engineering from Aristotle University of Thessaloniki, Greece, in 1985 and an MSc and PhD in Computer Engineering from Syracuse University, USA, in 1987 and 1990, respectively. Between 1990 and 2000 he was a faculty member with the Department of Electrical and Computer Engineering at Colorado State University in USA. Currently, Dr. Mitkas is a Professor of Electrical and Computer Engineering at the Aristotle University of Thessaloniki. He is also a faculty affiliate of the Informatics and Telematics Institute of CERTH, where he also directs the Intelligent Systems and Software Engineering Laboratory. His research interests include databases and knowledge bases, data mining, software agents, enviromatics and bioinformatics. Dr Mitkas is a senior member of the IEEE Computer Society. His work has been published in over 180 papers, book chapters, and conference publications. He is the co-author of a book on "Agent Intelligence through Data Mining" by Springer. Dr. Mitkas has served as Primary Investigator or Project Coordinator in several US and European research projects.

Fotis E. Psomopoulos is a Ph.D. candidate with the Electrical and Computer Engineering Department of the Aristotle University of Thessaloniki, Greece, where he received his Diploma in 2004. He is a research associate of the Informatics and Telematics Institute of CERTH, where he is a member of the Intelligent Systems and Software Engineering Lab. His research interests include bioinformatics, data mining and grid computing. He has worked on various National and European projects, including "eTHMMY" and "ASSIST" among others. Also, he is a Member of the IEEE Computer Society and the Technical Chamber of Greece.