A Web Based System Enabling Distributed Access and Intelligent Processing of Bioinformatics Data in Grid Environments

Ilias Maglogiannis[1], John Soldatos[2], Aristotelis Chatzioannou[1],
Vasilis Milonakis[2], Yiannis Kanaris[1]
[1]*University of the Aegean,* [2]*Athens Information Technology,*

e-mail: {imaglo@aegean.gr, achatzi@eie.gr, jsol@ait.edu.gr, vmil@ait.edu.gr, icsdm05004@icsd.aegean.gr}

Abstract This paper presents a Web based portal, which enables intelligent processing of biological data in Grid environments. The deployed software aims at creation of tools for processing data from microarray experiments over the Hellenic Grid infrastructure. Emphasis is given on user interface and access issues, while the paper describes also the data parsing and parallelization of the microarray data processing. The description of the system is oriented to Grid developers and users, since it focuses on the customization and use of the microarray applications over the Grid. Apart from supporting the high performance and economical execution of microarray experiments, the proposed system endeavors to provide access to a distributed repository of experiments information and results.

1 Introduction

The completion of the Human Genome Project and the emergence of high-throughput technologies at the dawn of the new millennium, are rapidly changing the way biomedical research is performed. Microarray experiments permit a genome-scale evaluation of gene functions and are therefore among the most topical and prominent developments of biomedical research. A microarray

experiment may produce great amounts of biological digital data that require further processing towards their exploitation. To alleviate the high cost of computing equipment required to support microarray experiments, researchers can leverage the computing power and the quality of service provided by Grid computing infrastructures. The amount of power offered by Grid infrastructures has been already exploited in the scope of a significant number of e-science applications, in particular applications with stringent computational and storage requirements. Bioinformatics applications in general and microarray experiments in particular are perfectly tailored to Grid infrastructures due to the need for high computing power and storage capacity [1]. Motivated by this fact, the aim of this research is to 'Gridify' and put on the Grid a selected number of microarray analysis, normalization and processing applications for cDNA arrays. The target Grid infrastructure is the Hellenic portion of the pan-European Grid infrastructure developed for e-science in the scope of the EGEE (Enabling Grids for E-Science in Europe) project and its successors [2]. The proposed platform is called hereafter HECTOR or EKTORAS in Greek, since it is funded by the Greek Secretariat of Research and Technology under a project named EKTORAS [9].

Figure 1 provides an overview of the application components comprising the EKTORAS application environment. At a high level the microarray experiments are conducted, processed and analyzed based on the following steps:

- Selecting a particular experiment among the pool of available normalization and clustering methods for cDNA microarrays. This selection task is performed by end users.
- Providing the microarray input files, which are usually structured according to formats that are standard for the microarray bioinformatics community. Microarray input files are specified by the end user and can be either files provided by the researchers themselves or even files residing in microarray public databases (i.e. European Bioinformatics Institute Database http://www.ebi.ac.uk/).
- Pre-processing the input files as so to render them usable by the range of algorithms available. The results of this pre-processing will be directed to the Grid's storage elements (SE).
- Parallelizing the application and distributing the parallel chunks & jobs to various nodes-processors of the Grid. Accumulating, storing and post-processing the results. This step contributes to create a large-scale virtualized database of microarray experiments.

The architecture presented in Figure 1 supports the above steps through a variety of software elements that are placed either within the Grid infrastructure, or as part of the access portal supporting interaction with the end users as described in the remainder sections.

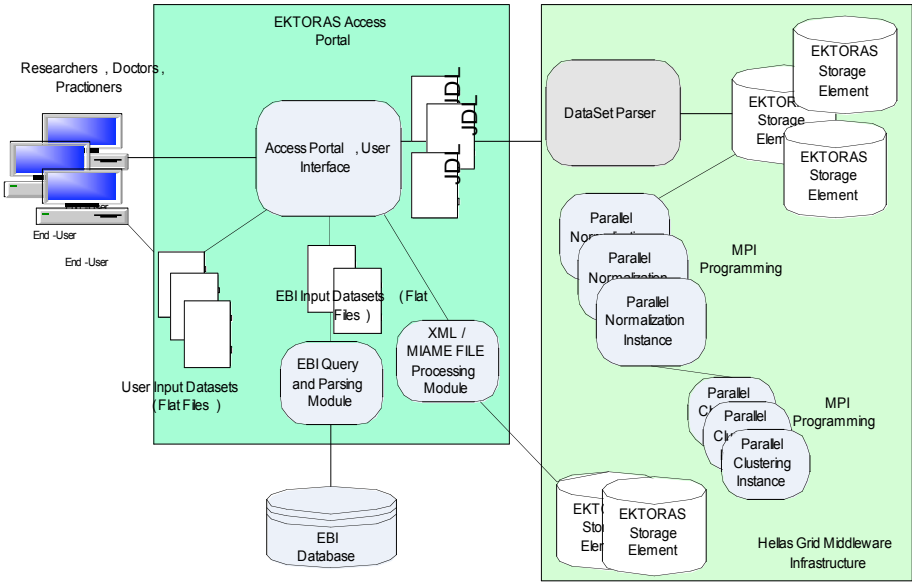

Fig.1. Overview of the HECTOR application architecture

2 User Interface - Access Portal

Users accessing the Web Interface through the implemented portal are given the ability to submit their experiments, retrieve their results and also compare them with formerly submitted experiments. Since the portal is set up on Hellas Grid User Interface (HG-UI), users have the ability to actually access the whole Grid infrastructure, consisting of many grid nodes. Access to services is enabled by parsing input files and accordingly activating the 'gridified' algorithms for processing the microarray experiments. Both data parsing operations and launching of experiments are specified as Grid jobs, using the Job Description Language (JDL). JDL is a high-level language, which is used to describe jobs and to aggregate jobs with arbitrary dependency relations. In the scope of the Grid, JDL is used to specify the desired job characteristics and constraints, which are used by the match-making process to select the best resource to execute the job. A job description is a file (called JDL file) consisting of lines having the format: attribute = expression; and terminated by a semicolon. Expressions can span several lines, but only the last one must be terminated by a semicolon.

Users of the EKTORAS portal are not required to be familiar with the procedure of creating the JDL files. They just have to set up a few parameters from the Web Portal that describes their experiment as shown in Figure 2, and to either upload their input files consisting of gene description, or even select them from a third party library-database of microarray files (e.g., the EBI Database). Accordingly the portal dynamically produces a JDL file specifying:

- Commands for parsing the inputs files and producing output files enabling the execution of the experiments. The parsing process is illustrated in the following section 3.
- Commands for activating the experiments over the Grid infrastructure. The later commands exploit the MPI capabilities of the Grid [3-4], while also leveraging appropriate data files produced after the parsing process.

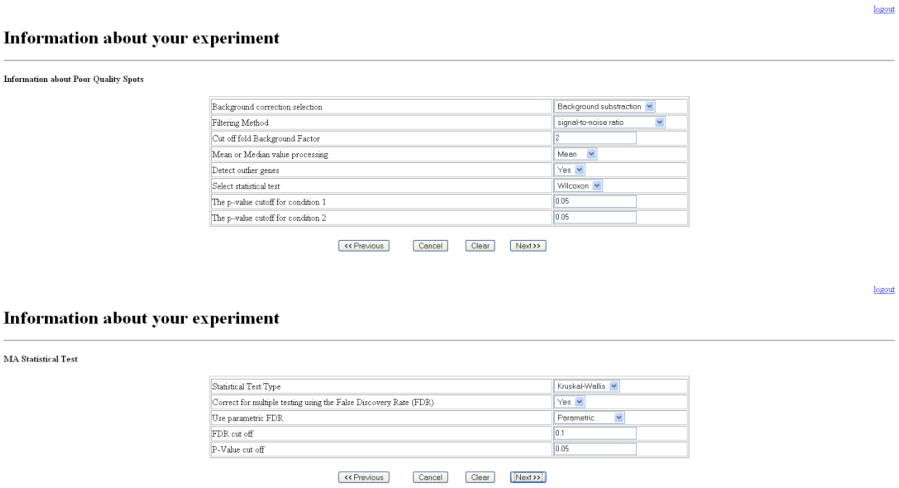

Fig. 2. Some steps from the input of parameters for the Experiment in the EKTORAS Portal

Following the dynamic production of the JDL file based on input files and configuration parameters, the portal submits this file to the Workload Management System (WMS) of the Grid for execution. The EKTORAS portal is accessible over the World Wide Web, through the URL http://www.icsd.aegean.gr/ektoras/. From an implementation perspective the portal is implemented using JSP (Java Server Pages) technology over a Tomcat/Jakarta infrastructure. The Jakarta infrastructure is hosted in a machine that is owned by the University of Aegean. Nevertheless, this machine becomes part of the Grid, in the form of a Grid UI. Implementing the portal within a Grid UI ensures that the access part of the EKTORAS application can directly leverage middleware services (e.g., security, reliability, resource management) of the underlying Grid infrastructure.

3 Input Data Parsing and Storage

Input data parsing constitutes an expedient pre-processing step to running microarray experiments. As outlined in the previous section and depicted in Figure 3, a data set parser module undertakes the transformation of input data files to other data storage structures (e.g., Matlab/Octave project files) that can be executed in conjunction with the gridified application. In order to maximize the impact and utility of EKTORAS services, we have tried to support a broad range of input files

formats. The development of the EKTORAS data parsing functionality supports the most popular formats, starting with the Imagene format [5-6]. The ImaGene format is the format specified and used within the ImaGene microarray image analysis software; a popular software for quick, automated measurement and visualization of gene expression data from spotted arrays. The ImaGene software extracts and quantifies spotted data from any 16-bit TIFF image file, and exports processed data in either text or XML files. In addition to Imagene other popular software tools are supported including QuantArray, GenePix, TIGR Spotfinder and the generic tab-delimited format used by EBI for storing data. By supporting these entire different formats EKTORAS platform is compliant with the vast majority of experiment data produced. Parsers are implemented in Python scripts, and manage to convert any of these aforementioned formats to Octave ASCII workspace files, storing the data into certain structures needed by the Octave code to run.

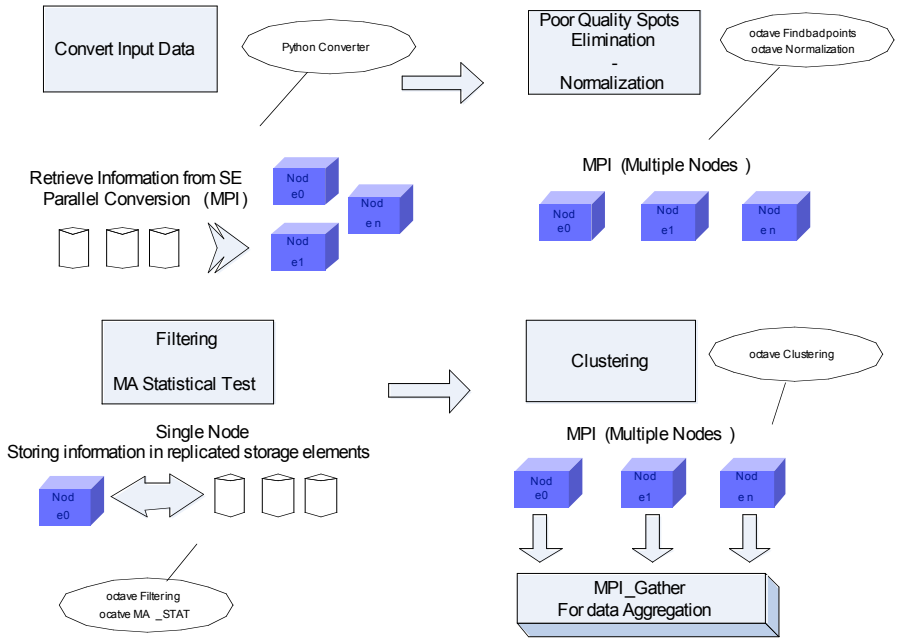

Fig. 3. Graphical representation of code parallelization

The parsing process produces a set of output files, which accordingly are stored within the Grid. Storage of output files is implemented using the Storage Elements (SEs) of the EGEE. SEs are services which allow a user or an application to store data for future retrieval. Currently, data storage within SEs is not subject to policies for space quota management. Moreover, all data in a SE are considered permanent and it is user responsibility to manage the available space in a SE (e.g., removing unnecessary data, moving files to mass storage systems etc.). As a result, the EKTORAS system stores outputs of the parsing process to one or more storage elements that are allocated to the project. These files are therefore available to the

experiments code, which also manages these files based on commands of the LCG File Catalog (LFC) and using the following abstractions:

- Logical File Name (LFN), which is an alias created by a user to refer to some item of data, e.g. "lfn:cms/20030203/run2/track1"
- Globally Unique Identifier (GUID), which is a non-human-readable unique identifier for an item of data, e.g., "guid:f81d4fae-7dec-11d0-a765-00a0c91e6bf6"
- Site URL (SURL) (or Physical File Name (PFN) or Site FN), which represents the location of an actual piece of data on a storage system, e.g., "srm://pcrd24.cern.ch/flatfiles/cms/output10_1" (SRM) and "sfn://lxshare0209.cern.ch/data/alice/ntuples.dat" (Classic SE)
- Transport URL (TURL), which corresponds to the temporary locator of a replica along with its access protocol: understood by a SE, e.g., rfio://lxshare0209.cern.ch//data/alice/ntuples.dat.

4 Grid Enabling Applications

The EKTORAS microarray processing algorithms have been initially provided by NHRF (National Hellenic Research Foundation) as a set of MATLAB libraries. However, no nodes of the Hellenic Grid Infrastructure provide support for MATLAB execution. Furthermore MATLAB is a commercial product, which raises intricate licensing issues when it comes to installing it in the Grid and makes it unlikely to become available in the near future. Therefore, we have investigated possible alternatives, the most prominent being the use of Octave Forge, which is the GNU open-source alternative to MATLAB. As a result, the first Grid application development step involved the conversion of the MATLAB code to (results) equivalent Octave Forge code.

Accordingly, we dealt with the task of parallelizing the (Octave Forge) microarray application and accordingly making it appropriate for use over the Grid. Initially, emphasis was given in placing the existing system into the grid environment. This task can be generally achieved through "wrapping" the existing code. The wrapping process has to audit the existing applications for their appropriateness for the Grid based on their following characteristics [7]:

- Their inter-process communications between jobs, without high speed switch connection (for example, MPI). In general, multi-threaded applications need to be checked for their need of inter-process communication.
- Their level of job scheduling requirements depending on data provisioning by uncontrolled data producers
- Unresolved obstacles to establish sufficient bandwidth on the network.
- Strongly limiting system environment dependencies for the jobs.
- Requirements for safe business transactions (commit and roll-back). At the moment, there are no standards for transaction processing on grids.
- High interdependencies between the jobs, which expose complex job flow management to the grid server and cause high rates of inter-process communication.

- Unsupported network protocols used by jobs, which may be prohibited to perform their tasks due to firewall rules.

Accordingly, we examined the application and job flows. Application flows is the flow of work between the jobs that make up the grid application. The internal flow of work within a job itself is called the job flow. During the grid enablement of the microarray processing algorithms we classified application flows into the following three basic types:

- Parallel flow: If an application consists of several jobs that can all be executed in parallel, a grid may be very suitable for effective execution on dedicated nodes, especially in the case when there is no (or a very limited) exchange of data among the jobs. From an initial job, a number of jobs are launched to execute on pre-selected or dynamically assigned nodes within the grid. Each job may receive a discrete set of data, fulfill its computational task independently and deliver its output. The output is collected by a final job or stored in a defined data store. For a given problem or application, it would be necessary to break it down into independent units. To take advantage of parallel execution of the microarray application in a grid, it is important to analyze tasks within an application to determine whether they can be broken down into individual and atomic units of work that can be run as individual jobs. This parallel application flow type is well suited for deployment on the EGEE grid. Significantly, this type of flow can occur when there are separate data sets per job and none of the jobs need result from another job as input. Classical examples of parallel flows are mathematical calculations, where the commutative and associative laws can be exploited.
- Serial flow: In contrast to the parallel flow is the serial application flow. In this case, there is a single thread of job execution where each of the subsequent jobs has to wait for its predecessor to end and deliver output data as input to the next job. This means that any job is a consumer of its predecessor, the data producer. In this case, the advantages of running in a grid environment are not based on access to multiple systems in parallel, but rather on the ability to use any of several appropriate and available resources. Note that each job does not necessarily have to run on the same resource, so if a particular job require specialized resources, this can be accommodated, while the other jobs may run on more standard and inexpensive resources. The ability for the jobs to run on any of a number of resources also increases the application's availability and reliability. In addition, it may make the application inherently scalable through the ability to utilize larger and faster resources at any particular point in time. Nevertheless, when encountering such a situation, it may be worthwhile to check whether the single jobs are really dependent on each other, or whether, due to their nature, they can be split into parallel executable units for submission on a grid. A prominent example of serial application flows are iterative scenarios (for example, convergent approximation calculations) where the output of one job is required as input to the next job of the same kind, a serial job flow is required to reach the desired result.
- Networked flow: In this case certain jobs within the application are executable in parallel, but there are inter-dependencies between them. In the scope of application with a networked flow we will exploit loose coupling, which implies a need for a job flow management service to handle the synchronization of the

individual results. Loose coupling between the jobs avoids high inter-process communication and reduces overhead in the grid. In the case of such experiments we will analyze how to best split the application into individual jobs, with a view to maximizing parallelism.

During grid programming we also introduced a hierarchical system of sub-jobs. Specifically, it is likely that a job could utilize the services of the grid environment to launch one or more sub-jobs. For this kind of environment, an application can be partitioned and designed in such a way that the higher-level jobs could include the logic to obtain resources and launch sub-jobs. This can facilitate large applications to isolate and pass the control and management of certain tasks to the individual components. Since the microarray processing algorithms deal with multiple replicates of data and the main process is independent for each replicate, they are highly parallelizable. So the elimination of poor quality spots and the normalization process can be executed in parallel. This parallel flow makes them appropriate for execution over a grid environment, towards meeting the economic goals set in the introductory paragraph of this report. In implementing the parallelization and exploiting multiple processors within the Grid, we exploited the MPI programming model. The following table provides pseudo code illustrating the use of MPI to launch parallel Octave forge instances executing normalization-related functions for the microarray instances.

Table 1. Sample MPI Pseudo code

```
#include <mpi.h>
int main(int argc, char *argv[])
{
... //----initializing MPI
MPI_Init(&argc, &argv);
//----learn node number
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
// load files        (in parallel for each file)
// Start Find Bad Points  (in parallel for each experiment)
[exptab,TotalBadpoints]=FindBadpoints(datstruct,t,exprp,imgsw);
// Normalize Data    (in parallel for each experiment)
[DataCellNormLo]=NormalizationLO(exptab,exprp,t,gnID);
   Gather processed data  (On node 0)
for (i=1;i<NumOfExperiments;i++)
   MPI_Recv(rcvbuf,count,datatype,i,tag,MPI_INT, MPI_COMM_WORLD,
&status);
/* MPI shutdown MPI */
MPI_Finalize();
//Filter Replicates
[DataCellFiltered]=FilterReplicates2(DataCellNormLo);
//Statitical Test
[DataCellStat]=MA_StaTestExp_New_total(DataCellFiltered,DataCellNormLo)
;
//CLUSTERING
……..
}
```

Table 1 refers to a simple parallelization of the microarray processing application, based on the number of input files/slides and their subsequent normalization processes. This is the most straightforward and simplistic

parallelization approach for the available microarray applications. A thorough analysis and structuring based on the above-mentioned jobs-subjobs hierarchical approach can in a later stage boost the application to much better performance levels. Figure 3 shows a more detailed view of the steps executed by our code and how it is parallelized in more nodes.

The aggregation of the results as it is depicted in Figure 3 runs in single node since it requires information from all experiments and it is not heavy computationally. Finally the clustering of the results runs in parallel on many nodes since it can run independently for each experiment and has increased computation complexity. Following the initial interaction of the end user with the access portal, there are is virtually no essential on-going interaction between user and grid application. During the course of the application's execution users limit themselves to monitoring the status of their job submissions.

5 Distributed Elements for Experiments Storage and Virtualization

Apart from supporting the high performance and economical execution of microarray experiments, the EKTORAS system endeavors to provide access to a repository of experiments information and results. Specifically, following the completion of an experiment over the EKTORAS application infrastructure, the experimental results are stored within appropriate SEs. These results are provided to the SE along with experiment meta-data specified according to MIAME (Minimum Information About a Microarray Experiment) XML format [8]. These metadata are requested by end-user through the access portal, prior to the execution of the experiment. Accordingly the portal structures a MIAME XML file. During implementation we tried to conform to microarray data standards, which are developed by the Microarray Gene Expression Data (MGED) Society (http://www.mged.org/). MIAME (www.mged.org/miame) is a prominent such standard, which outlines the minimum information that should be reported about a microarray experiment to enable its unambiguous interpretation and reproduction. Following the structuring of experiment meta-data as MIAME files, and their storage in SEs, we also developed a browsing mechanism that navigates across all the distributed SEs that contain experiments' information. This mechanism exports a browsing interface, along with a query/search interface to the access portal, in order to allow the end user query, search and access experiments' information and results. The distributed browsing mechanism is implemented as a C wrapper over the Grid data management functions and APIs. The latter APIs are exploited to deal with:

- Heterogeneity, since experiments' data will be stored on different storage systems using different access technologies
- Distribution, since experiments' data is likely to be stored in different locations, while also data needs to be moved between different locations.
- Different Administrative Domains, since data is likely to be stored at places where different access policies are applied and hence the browsing

mechanism will have to deal with the relevant security and auditing implications.

The EKTORAS access portal allows end-users to retrieve experiment files from the EBI microarray library. To this end the access portal provides an adapter to the EBI database system, which allows EKTORAS user to view, browse and select EBI files.

6 Conclusions

In this paper we strived to underpin the importance of grid computing for DNA microarray experiments. Accordingly, we have described the main components comprising the proposed 'gridification' of EKTORAS microarray data processing applications, along with the key technologies that support the implementation and integration of these components. We have also elaborated on a set of structuring principles for building the EKTORAS Grid Portal. This work serves as a starting point for building a more complete and integrated Grid enabled microarray experimentation environment.

Acknowledgment: This Research work is funded by the Greek Secretariat of Research and Technology under the Grant "Distributed Biological Data Processing of existing Open Public Databases (EKTORAS)".

References

1. I. Foster, 'What is the Grid? A Three Point Checklist', GRIDToday (http://www.gridtoday.com), July 20, 2002.
2. I. Foster, C. Kesselman, S. Tuecke, 'The Anatomy of the Grid: Enabling Scalable Virtual Organizations', International Journal of Supercomputer Applications, Vol.15, No.3, 2001.
3. Yukiya Aoyama et. al. 'RS/6000 SP: Practical MPI Programming', IBM redbook, August 1999.
4. Hellas Grid Training Material, available at: http://grid-training.ekt.gr/
5. http://www.biodiscovery.com/index/imagene
6. M. Kapushesky, P. Kemmeren, A.C. Culhane, S. Durinck, J. Ihmels, C. Kr, M. Kull, A. Torrente, U. Sarkans, J. Vilo and A. Brazma, 'Expression Profiler: next generation-an online platform for analysis of microarray data', Nucleic Acids Research, July 2004, Volume: 32: 1362-4962.
7. Luis Ferreira et al. 'Introduction to Grid Computing with Globus', IBM redpaper, December 2002.
8. A Brazma et al, 'Minimum information about a microarray experiment (MIAME)-toward standards for microarray data', in the Proc. Nat Genet. 2001 Dec;29(4):365-71.
9. I. Maglogiannis et al., "An Application Platform Enabling High Performance Grid Processing of Microarray Experiments", 20th IEEE Conf on Computer Based Medical Systems CBMS2007, Slovenia