GLAD: a system for developing and deploying large-scale bioinformatics grid

Yong-Meng Teo1,2,*, Xianbing Wang1,2 and Yew-Kwong Ng1

1Department of Computer Science, National University of Singapore, Singapore 117543 and
2Singapore-MIT Alliance, Singapore 117576

Received on January 4, 2004; revised on August 16, 2004; accepted on September 10, 2004
Advance Access publication September 16, 2004

Databases and ontologies

1 INTRODUCTION

Bioinformatics encompasses the methodologies for operating on biological information in order to facilitate research in molecular biology. Common operations on biological data include analysis of protein structures, comparison of genome sequences, visualization of sequence alignment results and placement of sequence databases.

The volumes of biological information stored in bioinformatics databases hosted by genome research centers, such as the National Center for Biotechnology Information (NCBI, http://ncbi.nlm.nih.gov) and Genome Institute of Singapore (GIS, http://www.gis.nus.edu.sg/), are enormous. It must be noted that, in the context of bioinformatics, the term database refers to a large set of cataloged sequences, and does not encompass the capabilities of standard data management systems, such as databases and hashing. Each sequence database file is in the range of gigabytes of data. In a distributed environment such as a wide-area grid, it may be necessary for the bioinformatics programs that are executed in the grid to access sequence data in bulk from several geographically disparate databases.

The essential problem in bioinformatics today is the lack of adequate support tools to bridge the gap between information technology and the life sciences comprehensively. There are many prolific examples of bioinformatics applications that are capable of providing solutions extensively to specific problems in life sciences research (Altschul et al., 1990; Fratini et al., 1982; Laskowski et al., 1993). Extravagant efforts were made to integrate existing bioinformatics resources to promote knowledge sharing among molecular biologists on a wide area. Some of these software projects have bore fruit after years of research and development, and their suite of products utilized by the world’s leading bioscience consortiums (Bosson and Riml, 2003, http://artedi.ebc.uu.se/course/overview/swedish_industry.html). The goals of the individual projects differ, however, depending on the mission of their developers. Table 1 provides a list of some successful bioinformatics projects.

However, bioinformatics applications typically require a high-performance orientation, and insufficient work has been carried out to provide an environment for the development and deployment of flexible, modular bioinformatics software solutions that can be parallelized for execution on a large scale, such as a wide-area grid environment. Grid computing has the overwhelming potential to apply supercomputing power to address a vast range of bioinformatics problems. Problems composed of non-trivial algorithms and operating over large datasets, such as fragment assembly of DNA molecules (Pevzner et al., 2001), three-dimensional structure protein analysis and prediction (Rost, 1998), and genetic linkage analysis (Terwilliger and Ott, 1991) can all be deployed for parallel execution on grids by adopting a divide-and-conquer approach, decomposing the problem into smaller task granules that can be distributed to compute resources. The difficulty posed in grid deployment stems from implementing the parallelization steps of each algorithm component, which is typically tedious, error-prone and, consequently, consumes extensive developmental time. This is especially crucial for large-scale bioinformatics problems, which usually comprises several algorithmic stages that involve rigorous computations. Furthermore, efficient grid computing middleware is necessary in order to facilitate the partitioning and dissemination of tasks to the available compute resources for execution.

The close associations attributed to grid computing and the life sciences in recent years do seek a greater demand for more flexible integrated systems that can support the creation of medium- to large-scale bioinformatics and biomedical grids. This is a crucial step toward globalization of the life sciences industry, as it enhances the

*To whom correspondence should be addressed.
means to share biological knowledge on a worldwide basis, thereby providing opportunities to foster productive collaborations between bioscience enterprises.

In this paper, we present GLAD (Grid Life sciences Applications Developer), which is a Java-based programming environment for bioinformatics problems on grids. GLAD is built on the ALiCE (Adaptive scLable Internet-based Computing Engine) (Alice, 2000, http://www.comp.nus.edu.sg/~teoym/alice.htm) grid core middleware, and sets out to provide bioscience researchers with an efficient workbench to implement distributed bioinformatics applications for deployment on a grid. GLAD provides the underlying mechanisms to handle the extraction and shipment of biological data across the grid. The toolkit comprises an extension layer, which encapsulates a set of commonly used bioinformatics algorithm components that can be adopted in the development of large applications.

This paper discusses the design and implementation of the GLAD application toolkit, demonstrating how an extensible library of bioinformatics algorithm components and a set of Java-based constructs can facilitate the development and deployment of medium- to large-scale bioinformatics problems on grids. The toolkit architecture hides the underlying ALiCE grid infrastructure, remote sequence data fetching and parsing mechanisms, and task communications from the user, thereby enabling user to concentrate on mapping the problem into the GLAD environment. It also allows the previously developed GLAD applications to be added to the extension library and reused as components in the development of future applications.

It shows how bioinformatics problems that involve regular and semi-regular parallelization patterns (Trells, 2001) operate over huge biological sequence databases, and can be efficiently deployed for scalable execution on homogeneous and heterogeneous grid environments with the adoption of task level parallelism. Case studies of applications developed using GLAD include distributed sequence comparison (regular parallelization) presented in this paper and distributed progressive multiple sequence alignment (PMSA) (semi-regular parallelization).

## 2 ALiCE MIDDLEWARE

### 2.1 ALiCE architecture

ALiCE is a grid-computing core middleware developed at the National University of Singapore, Singapore. Unlike Globus (Foster and Kesselman, 1997), which is a collection of fundamental grid construction tools and focuses on low-level services, ALiCE is a grid-computing system designed to aid the implementation of general-purpose applications and focuses on application programming models for grid environments. The current Globus Toolkit version 3.2 is a reference implementation of the evolving OGSa grid standard. We plan to align ALiCE to conform to OGSa once it is finalized. The ALiCE grid architecture consists of three main layers, supported by a set of existing Java technologies, as shown in Figure 1.

Fig. 1. ALiCE grid architecture.
### ALiCE runtime system framework

A framework to keep track of vital statistical information of the grid environment, such as the utilization of each of the participant resources.

ALiCE Extensions encompasses the ALiCE runtime support infrastructure for application execution, and provides the grid application developer with an API. Finally, the ALiCE Applications and Toolkits layer is a non-exhaustive collection of grid applications and programming models developed using the ALiCE programming API. This is the only layer of the ALiCE grid architecture that is visible to application users.

#### 2.2 ALiCE runtime system

The ALiCE runtime system is an integration of the Compute Grid Services and ONTA components from the ALiCE Core layer in the grid architecture. It consists of the following four processes (Fig. 2):

- **Consumer** is the users’ entry point to an ALiCE grid for submitting applications to execute and collect the corresponding results.
- **Producer** runs on any machine that volunteers their compute cycles to an ALiCE grid. It retrieves tasks generated from ALiCE applications, executes them and returns the results to the consumer.
- **Resource broker** coordinates and controls the scheduling of applications and tasks, ensuring that all concurrently executing applications can complete within satisfactory turnaround times, and that all the producers within the grid are well utilized. It also registers, manages and allocates producers and other grid resources.
- **Task farm manager** generates tasks for ALiCE applications. ALiCE supports task farm managers for developing languages other than Java, and this relieves the resource broker from the problem of handling non-Java application codes.

A consumer and a producer can be simultaneously run on a common machine. However, resource brokers and task farm managers must be run on separate nodes. The ALiCE runtime system is robust and has the capability to extend to wide-area grid environments, which would involve multiple heterogeneous resources.

#### 2.3 ALiCE API

To support the development of grid applications and domain-specific application programming toolkits, ALiCE programming template provides an API that adopts a TaskGenerator-ResultsCollector programming model, encompassing the following four extensible classes:

- **TaskGenerator** generates the application’s tasks on a task farm manager machine. It implements a method process that generates a new task and sends a reference to the task back to the resource broker machine for scheduling.
- **Task models** a task object generated by an application, and comprises a series of computations to be performed by a producer machine.
- **Result models** a result object returned from a producer machine to the resource broker.
- **ResultCollector** is the entry point class of the ALiCE grid system, handling application user administrative issues such as data input and results visualization at a consumer machine. It implements by retrieving a result object from the relevant resource broker.

An application, with its codes encapsulated in a .jar file, is submitted at a consumer machine to the resource broker, which distributes the codes to a task farm machine, where the TaskGenerator is run. The generated Task objects are then disseminated to the producer machines for execution, returning the computation results in a Result object to the resource broker. In the interactive mode, the ResultsCollector running on the consumer machine retrieves the Result objects. In the batch mode, the resource broker stores the results.

The ALiCE API enables grid application developers to exploit the distributed nature of the ALiCE grid without needing to know about the technologies for communications and dynamic code linking, by providing the technical functionalities at an abstract level.

#### 3 GLAD TOOLKIT ARCHITECTURE

Although ALiCE provides the runtime environment and programming APIs for grid applications, there are still lots of works to be carried out for developing various grid applications by using ALiCE middleware. For example, how to design user portal, how to parallelize jobs, how to generate biological tasks, etc., as these issues are related to the underlying implementation of ALiCE. For different applications in the same fields, these issues can be reused. Thus, we develop GLAD to provide programming assistance for bioinformatics grid applications.

GLAD is a bioinformatics application workbench on the ALiCE grid architecture for successfully deploying large-scale bioinformatics applications on a wide-area grid environments. It provides programming assistance for developing Java-based distributed bioinformatics applications by reducing the burden posed in the implementation of the parallelization, and allows user to invoke reusable basic components. The GLAD workbench is portable to any hardware platform that runs a Java Virtual Machine and supports the ALiCE runtime system.

#### 3.1 GLAD architecture

The GLAD architecture comprises four constituent layers (Fig. 3):

- **3.1.1 Execution control layer** An effective parallel execution control system should minimize the overhead incurred from data communications and ensures that parallel execution results in a reasonable and significant speed up. These issues are especially important while considering the grid deployment of bioinformatics applications, which may process operations up to tens of gigabytes of biological sequence data from several foreign sources. Thus, good performance and scalability can be achieved.
GLAD: a system for large-scale bioinformatics grid

3.1 GLAD application structure

3.1.2 Tools library A wide variety of long-existing bioinformatics algorithms are commonly used in the development of larger applications. For instance, the maximum parsimony method (Fitch, 1977) is a fundamental character-state method in the area of evolutionary science, while the Fitch and Margoliash (1967) approach is a well-established weighted distance-matrix method for the construction of deep phylogenetic trees. The Smith–Waterman dynamic programming algorithm (Monge and Elkan, 1996), on the other hand, is commonly used for the computation of similarity scores of sequence pairs.

To simplify the work of a large-scale, multi-stage bioinformatics application developers, a library of such algorithms as provided, so that the programmers can invoke the appropriate tools as they need. This potentially reduces the developing time and therefore greater attention to be placed on the more complicated portions of the problem. This approach also promotes modularity and code reusability, which are the essential features of grid systems. The tools library in GLAD is extensible and non-exhaustive, thereby enabling newly implemented bioinformatics algorithms, including even GLAD applications, to be added to the set.

3.1.3 Applications development The development of applications using the GLAD toolkit is supported with the help of a Problem Description Template (PDT), which is really a Java-based programming template for users to model their bioinformatics applications. This template itself was implemented using the ALiCE programming template for grid deployment facility. It allows for development at a certain level of abstraction by keeping the details of grid programming transparent to the programmer, leaving the programmer with the simple job of filling up a set of predefined methods.

The developmental framework also permits the programmer to customize the application’s user interface by offering a number of standard visualization components for the graphical illustration of DNA and protein sequences, the textual display of results, the development of user interfaces for general parametric inputs associated with bioinformatics computations and the presentation of statistical information regarding the application’s dataset. However, the programmer is free to implement his/her own visualization without the aid of any of these provisions.

3.2 GLAD application structure

Figure 4 shows the conceptual model of the structure of a GLAD bioinformatics application and its mapping into ALiCE. In the GLAD execution paradigm, a bioinformatics application is viewed as a composition of one or more successive biological stages, and comprises two parts: the user portal and the problem description.

The user portal enables interactions between the application user and the application itself, obtaining parametric inputs for the problem during the initialization phase, and reporting the results upon completion of the application’s execution in a manner subject to user’s customization. The user portal runs on the consumer machine in the ALiCE runtime system, and its underlying implementation is based on the ALiCE ResultsCollector process.

The problem description is a sequence of biological stages that constitute the bioinformatics problem modeled in the application. The stages are executed chronologically, and each stage can be either executed sequentially or deployed for parallel execution in a distributed environment using the ALiCE runtime system. In the latter case, the stage is comprehensively decomposed into biological tasks that are responsible for solving subparts of the problem associated with it. The task partitioning is performed by the parallelization routine for that stage. The parallelization routines required for all the biological stages involved in the application are collated using the ALiCE TaskGenerator process running at the resource broker. Biological tasks are scheduled for execution at the producer machines, where the task routines are processed, which may involve
4 GLAD LIBRARY IMPLEMENTATION

The GLAD library is implemented based on Java, and the underlying distributed execution engine is supported by the ALiCE runtime system. In the GLAD paradigm, a bioinformatics problem can be modeled using several biological stages, and each stage generates numerous biological tasks during parallel execution. This can be readily described by the object-oriented programming approach that is being advocated using the Java language. Furthermore, the parallelization and task execution routines can be simply translated into ALiCE, which provides very generic constructs for all kinds of grid applications. It is this element of simplicity that enables ALiCE to be an outstanding infrastructure tool for developing domain-specific programming models and toolkits, such as GLAD.

The GLAD library is a structured composition of two different types of classes, namely developmental classes and control classes. Developmental classes are the ones that model the problem, while control classes are responsible for the underlying mechanisms that support the application’s execution. GLAD’s PDT provides the environment in which the application programmer can model the bioinformatics problem.

4.1 Developmental classes

GLAD supports distributed bioinformatics applications development with five main classes, as illustrated in Figure 5. Each class provides a set of primitives that can be manipulated in the course of implementation, and a list of abstract methods that are to be filled in by the programmer to dictate the different execution routines involved.

The BioStage class models a biological stage in the given application, describing the algorithms involved in that stage and providing parallelization capabilities that are kept transparent from the developer. However, the developer can override the parallelization routines when sophisticated parallelization strategies are required to address the problem.

The BioTask, BioGenerator and BioVisualizer classes are inherited from the three major classes in the ALiCE programming template. BioTask models a biological task that is to be scheduled for processing at the ALiCE producer machines. BioGenerator runs the biological task generation codes at the task farm manager. BioVisualizer is the visualization framework for the GLAD application. The BioResult class, on the other hand, is an extension of ALiCE’s Result class, and is used in the propagation of results produced from the computation of a BioTask.

4.2 Problem description template

A distributed bioinformatics application can be implemented using the PDT provided by the GLAD library. This programming template, shown in Figure 6, comprises the four developmental classes described above.

PDT essentially highlights the methods that the GLAD application developer has to implement in the corresponding subclasses in order to successfully deploy a distributed bioinformatics application on a grid system. The developer will typically declare problem-specific attributes, data structures as well as subroutines in the subclasses. In general, the developer is required to provide the following five basic items:

- Linkage of the user interface and visualization components.
- Application execution logic in terms of stages.
- Code for generating the biological tasks for each problem stage.
- Logic for computation of each type of biological task.
- Code for processing of results from the execution of each stage.

The application’s execution commences from the BioVisualizer running on the ALiCE consumer machine, with the entry and exit points of the distributed computation occurring at the StartApplication() method. One or more BioStage processes will be created and activated sequentially. Each BioStage spins off a parallel computation in the ALiCE runtime environment, in which BioTask
objects are generated by the BioGenerator and disseminated to the producer machines for processing. The results of each task computation are written to a BioResult object that is returned to the consumer machine in the course of each biological stage. The BioGenerator class is a subclass of the TaskGenerator class in the ALiCE library, and can be used to spawn different types of BioTask objects for the various stages in the problem.

4.2.2 BioTask template The BioTask objects must be created centrally before they can be distributed to the pool of producers for computation. This is achieved by the BioGenerator class, which generates them appropriately according to the task partitioning schedule received from the application’s visualizer running on the consumer machine in the course of each biological stage. The BioGenerator class is a subclass of the TaskGenerator class in the ALiCE library, and can be used to spawn different types of BioTask objects for the various stages in the problem.

4.2.3 BioGenerator template The BioTask objects must be created centrally before they can be distributed to the pool of producers for computation. This is achieved by the BioGenerator class, which generates them appropriately according to the task partitioning schedule received from the application’s visualizer running on the consumer machine in the course of each biological stage. The BioGenerator class is a subclass of the TaskGenerator class in the ALiCE library, and can be used to spawn different types of BioTask objects for the various stages in the problem.

4.2.4 BioVisualizer template GLAD provides a visualization module in the form of the BioVisualizer class, to enhance the analysis of application results. It also allows the application users to specify problem-specific and execution parameter values prior to execution through a customizable graphical user interface. The BioVisualizer class is inherited from the ResultsCollector class in the ALiCE programming template. Besides handling the collection of BioResult objects in the course of the application’s execution, the BioVisualizer also coordinates the formatting of the visualization frames and windows, and controls the manner in which the results will be presented to the application user.

4.3 Control classes

The execution of GLAD applications is driven by a number of classes that provide the underlying mechanisms supporting remote access to biological databases, interpretation of biological data, as well as monitoring of application performance. Unlike the GLAD developmental classes, these control classes are not inherited from the ALiCE core or extensions layers, and provide the modularity that separates the various functional components in the GLAD execution kernel from the PDT. The detailed implementation and capabilities of the control classes are transparent to application developers.

4.3.1 Handling biological data Biological data essentially comprises structured sequences that may be hosted on the local machine or sited on remote databases that are geographically distant from the ALiCE machine executing the GLAD application. The administration of biological data is performed by the Data Parser and BDA engine components in the GLAD architecture, which involve the following classes:

- DataParser provides an interface between the application and a sequence file, comprising methods to extract the sequence descriptors, actual sequence bodies and information regarding the biological classification of the sequences, such as whether a particular set of sequences are chromosomes or amino acids.
- SequencesClient represents the client end of the BDA engine, running on a compute producer. Whenever a set of sequences hosted on a remote database are required for computation, it establishes a TCP/IP connection with its counterpart process, the SequencesServer, running on the remote machine. The relevant sequences are then downloaded across the network and passed to the DataParser object on the local producer for translation.
SequencesServer provides an interface from which the developer can invoke the various biological data and distributed object communications. Separated from the mechanisms for task decomposition, handling of the workbench for the application developer work upon, are cleanly formatics application. The developmental classes, which provide library modularizes the various functional components of a bioin-
toolkit is presented in Figure 8. The implementation of the GLAD application structure of the Java classes that constitute the GLAD application execution of the application.

Typical protein databases are in the range of gigabytes of data. The BDA engine classes provide efficient transportation of biological data across the network by downloading only the required sequences. Figure 7 shows the collaboration between the above-mentioned classes in handling biological data.

4.3.2 Statistical and performing monitoring It may be important to determine certain meta-features of the biological dataset used by the application prior to execution proper. The Statistics class enables the developer to retrieve information such as the cardinality of the dataset, the mean length of all the sequences in the dataset, the corresponding standard deviation of sequence length and the number of updates to a particular database since it was last accessed by the same GLAD runtime environment. This information not only helps to determine the total number of tasks that would be generated for a given execution, but also provides the user with an overview of the features of the dataset to be used.

The Statistics class also monitors the performance of the GLAD application execution by keeping track of the turnaround time at the BioVisualizer, the average task processing time for each BioStage in the grid environment and the actual amount of time taken for each execution of the application.

4.4 Implementation structure The structure of the Java classes that constitute the GLAD application toolkit is presented in Figure 8. The implementation of the GLAD library modularizes the various functional components of a bioinformatics application. The developmental classes, which provide the workbench for the application developer work upon, are cleanly separated from the mechanisms for task decomposition, handling of biological data and distributed object communications.

The library also includes a special class, LibraryLoader, which provides an interface from which the developer can invoke the various algorithmic components stored in the GLAD tools library for the application’s use. For instance, a developer faced with a problem that involves the Huffman Encoding (HME) algorithm in one of its stages may invoke methods in the HME tool directly, without worrying about the implementation of the complete algorithm as one of its stages. This simplifies coding and reduces programming time, as the developer is capable of manipulating commonly used small- and medium-scaled bioinformatics algorithms at an abstract level.

Newly implemented GLAD applications can also be added to this set of bioinformatics tools by placing the class codes in the relevant subdirectories under the tools directory in the GLAD library, and replicating the codes across all systems in the GLAD runtime environment.

5 BENCHMARK APPLICATIONS The GLAD library has been used to develop and deploy two different applications on grid systems. The first is the distributed sequence comparison, which has a relatively straightforward algorithm, a regular computational pattern, and is both compute- and data-intensive. The second is the distributed PMSA, which comprises multiple algorithmic stages, involves a semi-regular computational pattern and lots of heavy computations but does not typically involve large datasets. The different level of task dependencies between the two problems allows us to study the different approaches considered to parallelize and deploy each of them for grid execution.

For illustration, we introduce how to map the distributed sequence comparison problem onto the GLAD developmental framework. The map of the distributed PMSA onto the GLAD is available in Alice (2000) (http://www.comp.nus.edu.sg/~teoym/alice.htm).

Sequence comparison is one of the most important primitive operations in computational biology, serving as a basis for many other more sophisticated manipulations. In laymen terms, it involves discovering the similarity of parts of two sequences. The end result would be the provision of an optimal alignment for the pair of protein or DNA sequences. Two main concepts are involved in this problem: the similarity and the alignment of the two sequences. The similarity of two sequences is a metric that dictates how syntactically matching they are. The alignment of two sequences is a way of placing one sequence above the other in order to clarify the correspondence
between residues and portions of the sequences. Gaps can be inserted in arbitrary locations along the given sequences so that they end up with the same length, thus, enabling them to be comparable with each other.

Sequence comparison has a regular computation pattern, and the entire problem can be implemented in one biological stage. Figure 9 shows the implementation of the application using the GLAD library.

The SeqCompVisualizer provides the entry and exit points of the application, handling the parametric inputs for similarity computation, spawning of the AlignmentStage process and the visualization of alignment results. AlignmentStage coordinates the flow of the similarity computation and sequence alignments across the queries and dataset, by generating the task partitioning schedule for the SeqCompGenerator. The SeqCompGenerator, in turn, creates the AlignmentTask objects for dissemination to the producer machines. Each AlignmentTask is responsible for the alignment of a query with a specific number of database sequences.

For example, if we assume a task size of 2000 sequences, then each AlignmentTask is responsible for determining the top-scoring sequences among an exclusive partition of 2000 sequences in the database, and returning the score and relevant sequences in the form of an AlignmentResult object. The SeqCompVisualizer updates the highest score from the entire dataset as the AlignmentResult objects are retrieved, and eventually displays, for each query, the highest scoring sequences in the dataset and the optimal alignments thus derived (Fig. 10). The flow diagram in Figure 11 illustrates this procedure for a task size of 2000 comparisons.

6 CONCLUDING REMARKS

We have developed a bioinformatics application toolkit, GLAD, which is implemented using the ALiCE paradigm. GLAD enables the researcher to work with a set of primitives and constructs, without specific technical knowledge of the means in which parallelization and biological data processing are being handled by the system. GLAD is scalable and supports the development and deployment of applications involving regular and semi-regular parallelization patterns and operating over huge biological datasets distributed over several databases on the network.

GLAD provides a grid-based bioinformatics applications development workbench that is designed for the deployment of medium-to large-scale applications to facilitate extensive research in life sciences. The main objective of GLAD is to support the implementation of parallel bioinformatics applications on a variety of grid environments without being handicapped by the lack of human expertise in the construction of computational grids and the hassle of manually coding the treatment of biological datasets involved in the individual executions.

We have also demonstrated the use of GLAD in the development of two grid-based bioinformatics applications: distributed sequence comparison and distributed PMSA, and deployed them on homogeneous and heterogeneous cluster grids separately for our performance scalability experiments.

Users can contact the authors, if he/her can point to anything that could be improved in GLAD, have suggestions, or need benchmark applications and the GLAD toolkit.

ACKNOWLEDGEMENT

The authors would like to thank Dr Li Kuo-Bin from the Bioinformatics Institute of Singapore for the many useful discussions and helpful suggestions.

REFERENCES


National Center for Biotechnology Information (NCBI) (2004), National Library of Medicine, MD.

802