A Gridified Protein Structure Prediction System "Rokky-G" and its Implementation Issues

Shingo MASUDA a, Minoru IKEBE a, Kazutoshi FUJIKAWA a, Hideki SUNAHARA a

a Graduate school of Information Science, Nara Institute of Science and Technology

Abstract. In recent years, simulation using computer systems has been of increasing importance in the life sciences. We have developed a system called “Rokky-G” that facilitates a protein structure prediction strategy called “Rokky” on Grid systems. Rokky-G provides the framework of protein structure prediction on the Grid. In this paper we discuss the architecture of Rokky-G and implementation issues identified in order to obtain highly reliable results.

Keywords. Grid Computing, Protein structure prediction, Trial-and-error, Job Priority, Resource allocation

1. Introduction

Protein structure prediction using large computer systems has been of increasing importance in the life sciences in recent years. Protein structure prediction is intended to determine the three-dimensional structure of a protein (known as the “tertiary structure”) from a given amino acid sequence (known as the “primary structure”). The predicted structure is very useful for other life science and medical fields because a protein’s function is closely related to its three-dimensional structure. However, there is no unique prediction algorithm which can find the correct structure for every protein, although several algorithms have been developed. Developing an effective prediction method is therefore a matter of high priority for researchers in the life sciences and medical fields. Takada et al. have been developing an effective protein prediction system called “Rokky” [4]. Rokky is a web-based fully-automated server that can predict structure from a given amino acid sequence. Although Rokky can provide good prediction results, there are two issues which may be addressed to improve the results. One is that human intervention may be incorporated into the prediction procedure, because it can help with prediction failures. The other is that the current Rokky system only runs on a single computer cluster system. Naturally, we expect that Rokky would provide better prediction results if it could benefit from a Grid environment.

In order to accommodate the these issues, we have been developing “Rokky-G” which is a Grid-oriented protein structure prediction system. With Rokky-G, all simulation programs for structure prediction can be handled as jobs in Grid environments [5]. In developing Rokky-G, we have three design objectives: one is to minimize modification of the original Rokky system. The second is to enable the
straightforward addition of other programs to Rokky-G, and the third is to improve reliability of results.

In this paper, we discuss the implementation methodology of Rokky-G intended to satisfy the above design goals. The rest of the paper is organized as follows. Section 2 describes the original Rokky system. Section 3 describes the architecture of Rokky-G. Section 4 reveals and evaluates a prototype implementation of Rokky-G, and finally Section 5 describes related work and concludes the paper.

2. Rokky and its protein structure prediction algorithms.

Rokky is a web-based fully-automated server that can predict protein structure given an amino acid sequence. Rokky’s performance was benchmarked in a recent world-wide blind test at the CASP6 [7], revealing that Rokky is the second best prediction server among over 50 servers for targets without a template protein. A distinctive feature of Rokky is that the system integrates the standard bioinformatics tools and servers with a fragment assembly simulator called “SimFold” [6, 7].

Rokky consists of three simulation programs and several scripts. There is a master script which invokes other scripts and simulation programs. The three simulation programs are PSI-BLAST, 3D-Jury and SimFold, which are for targets of CM, FR, and NF respectively. Rokky performs the following steps automatically. First, Rokky executes PSI-BLAST [1] and produces some results. Next, Rokky invokes 3D-Jury [2]. When results are returned from 3D-Jury, Rokky computes the results of “Comparative Modeling” and decides whether or not SimFold should be run. If there are unsolved portions in the target sequence, Rokky executes SimFold. Finally, Rokky combines the output of CM, FR and NF in order to generate the final results.

In Rokky, a user can improve the reliability of results using trial-and-error. SimFold adopts a “Monte Carlo/Simulated Annealing” method. In the Monte Carlo method, multiple simulations with different parameters can be executed independently and simultaneously. To get reliable results within a limited time, a user can check the intermediate results during simulation execution and change the parameters in Rokky dynamically.

3. Architecture of Rokky-G

3.1. Design goal

In developing Rokky-G, we have the following three design objectives:

- Minimize modification of the original Rokky programs
- Enable the straightforward addition of other programs
- Enable the user to improve the reliability of results

The first objective is intended to reduce the cost of gridifying the original programs. There are several simulation programs and scripts in Rokky. It is desirable that the original simulation programs and scripts can simply be treated as jobs or tasks in Grid environments.
No one complete algorithm has been established for protein structure prediction. It is therefore sometimes desirable to use other simulation programs to improve the reliability of results. In fact, Rokky previously used “FUGUE” \cite{3} as an FR program. As described in the previous section, executing simulation programs in a trial-and-error manner can often help to improve the reliability of results. Trial-and-error execution of simulation programs should therefore be made straightforward.

3.2. Architecture

In Rokky, several simulation programs and scripts are invoked by the main script, and some parameters and settings are embedded in these scripts. In Grid environments, the computational resources available are not determined in advance. However, simulation programs and scripts must be installed on the computational nodes in Grid environment in advance and this makes changing the parameters embedded in scripts problematic. Furthermore, Rokky’s main script depends on the configuration of other programs and computational resources. Considering this problem, we gridify Rokky as follows. First, we develop two middlewares. One is a workflow management system called “Workman”. Workman can replace Rokky’s main script. Workman provides the functionality to automate the execution of several programs sequentially. The other middleware is a job management system called “Jobman”. Jobman enables dynamic computational resource allocation. It also enables effective job execution and enables more reliable results to be obtained in a given time.

Generally, gridified programs are reconstructed as services working in a coordinated manner with each other. A service consists of several programs (including some middleware) and computer resources (such as cluster computers). It provides abstract methods that may be called by users or other services. Workman and Jobman are services. All executions are performed by invoking services using abstract methods. Program control is achieved through the invocation of services, as shown in figure 1, and a workflow consists of a number of services and their relationships.

![Figure 1: A service](image-url)
When we gridify Rokky, we construct services for each algorithm and include several scripts and programs because it is rare that scientists wish to execute only a part of an algorithm.

We construct a Rokky-Main service with functionality similar to that of Rokky’s main script. However, the Rokky-Main service doesn’t depend on the configuration of the executing environment or other services. While in the original Rokky system, computer resource allocation, location of input/output files and the name of the program to execute are determined statically before execution of a complete prediction sequence, the Rokky-Main script can control these parameters even if the prediction sequence is currently under execution. The functions of the Rokky-Main service are limited to the following:

- Deciding whether or not it is necessary to execute SimFold
- Combining the output of CM, FR and NF to generate final results

The following figure shows the architecture of Rokky-G.

![Figure 2: The architecture of Rokky-G](image)

In using Workman the user can easily define the order of program execution and the relationships among programs. This definition is called the workflow. Workman has the following two components:

1. Workflow design tool
2. Workflow execution server

The User Interface in figure 2 incorporates the workflow design tool. All jobs are invoked by Jobman and the workflow execution server uses Jobman to execute programs.

### 3.3. The service for parallel jobs

In this section, we discuss the service for parallel jobs. In Rokky, SimFold uses a type of Monte Carlo/Simulated Annealing that requires many parallel jobs. The Monte
Carlo method often requires a many execution patterns with different initial values, often generated randomly. Each execution can usually be an independent job, and these jobs can be performed on multiple computers concurrently. When SimFold is executed there are a large number of jobs of this kind. The execution of SimFold is however problematic, because the number of jobs necessary to obtain reliable results is indeterminate. There are often differences in the number of jobs required according to the domain, and it is necessary to assign more computational resources to executions which have a large number of jobs being run.

The best granularity of parallel job services is expressed below. The requirements are as follows:

- Utilize all computer resources
- Enable dynamic computational resource allocation
- Facilitate straightforward service implementation

We can consider services at three levels of granularity. The list below shows what would be performed for a single invocation at each service level.

1. Perform executions with all sets of initial values.
2. Perform executions with some sets of initial values (which may be specified upon invocation).
3. Perform execution with only one set of initial values.

Choice 3 is most appropriate for our requirements.

Suppose we consider SimFold as an example. The type of service with the coarsest granularity is 1. When we construct SimFold services as type 1, all SimFold jobs are executed for each invocation and all processes are usually executed on one cluster system. Frequently however, computational resources are spread over many bases and we cannot make effective use of them. If SimFold services are constructed as type 2 or 3, a single invocation of a service doesn’t perform all the jobs corresponding to the initial values. Therefore, we can distribute jobs over multiple services. Thus, we
can utilize multiple computer systems spread over many bases. There are usually cases when there are several domains which must be executed concurrently. However, a type service cannot perform jobs in multiple domains concurrently because execution using this type in only one domain fills the computational resources of one service with jobs. This situation is shown in figure 3.

When we construct services of type or , we must construct a “Master” service. The Master service is required to make lists of initial values and invoke the services of SimFold multiple times automatically. Additionally, the Master service collects all output data and compiles the ultimate result.

We now explain which of these two types of SimFold service is better. Type is better because we can construct a service straightforwardly. In fact, type services are a super-set of type services. The functionality to make multiple executions is included in type services by invoking type services once. The Master service must have a function to invoke the SimFold services many times in order to utilize services on multiple bases whether using type or . If we select type , we must implement similar functions in the SimFold services and in the Master service.

In protein structure prediction, it is necessary to perform executions in multiple domains simultaneously. For this reason, it is necessary that the Master service has the functionality to decide where to assign service executions in each domain. Consequently, we implement this function in Jobman. The Master service uses Jobman to invoke SimFold services and thus implementing the Master service is straightforward.

As mentioned above, in constructing a service for “Parallel program execution” we should make two paired services. One of the two services is a calculation service which executes one process with one invocation, and the other is a Master service which coordinates the calculation service.

4. A Prototype implementation

In this section, we explain a prototype implementation of the system, and the operations of Jobman and Workman.

4.1. Jobman: The Job management system

We developed a prototype version of Jobman with the following functions:

- Deciding computer resource allocation for a job group according to its priority
- Changing computational resource allocation dynamically according to user decisions and/or process reports

Both and regulate the assignment of jobs to services. Jobman knows the number of jobs which can be executed simultaneously on each service. The interactions between Jobman and the Master service, and the calculation services for Parallel execution are described below.
In order to handle computer resource allocation, Jobman makes groups consisting of multiple jobs. Priorities are assigned to a group and all the jobs participating in the group. All services are invoked by Jobman.

We now explain the operation steps of Jobman using SimFold as an example. As described in the previous section, the Master service invokes SimFold services through Jobman.

First, the Master service makes a request for the construction of a group to Jobman. When the Master service sends the request for a job to Jobman, Jobman doesn’t directly invoke SimFold services. The Master service must send a “Run group” request to start the job. The Master service sends job requests from all domains and information about default priorities before sending “Run group” requests. Jobman decides the number of jobs which execute simultaneously in each group using priority information. For example, suppose there are two SimFold services, named “Sa” and “Sb”. Each service can execute 30 jobs at the same time. Jobman receives requests for 2 groups, named (Ga) and (Gb), each with 80 jobs. When the priority of each group is the same, Jobman decides the number of concurrent jobs is 30 for each group. In many cases, SimFold services perform differently. For this reason, Jobman assigns 15 jobs of (Ga) and (Gb) to each service to prevent a large inequality occurring. If the priority of (Ga) is twice as high as the priority of (Gb), then Jobman decides the number of concurrent jobs in (Ga) to be 40, and 20 jobs are assigned to each service. The number of concurrent jobs in (Gb) is decided to be 20, and 10 jobs are assigned to each service. In this example, if both Sa and Sb have the same computational performance and each job has the same execution time-frame, 40 jobs in (Gb) will not have been executed at the point when all jobs of (Ga) finish executing.

Jobman sends the decided number of “Run Job” requests to the calculation service. When Jobman observes that a job has completed, Jobman sends a request for the next job. Since job requests are sent step by step, the SimFold services have no regard for job priorities and only have to execute the jobs sent.

4.2. Workman: The workflow management system

In this section we describe the workflow management system, Workman. We developed a Workman prototype with has following features:

<\A> Straightforward performance of simulations in trial-and-error manner by the user
\B> Reduced total execution time according to the trial-and-error method

Workman organizes cooperation among multiple services, and executes services in a user defined sequence. The function of Workman can be explained with a simple example. In the workflow system, we call a service that has been invoked “work”, and the whole process consisting of several pieces of “work” is called the “workflow”. Making the workflow is equivalent to defining and ordering the services to be executed. We now consider a case when we want to execute 2 simulation services. First, we define which services to execute and the execution order. We further define the relationships among input/output data. In this example, the two services have the following definitions, and we input the data shown in table 1 into the workflow system using a dedicated user interface.
Service A
Name: Protein-CM-sample
Input data: Filename
Output data1: Filename
Output data 2: Filename

Service B
Name: Protein-NF-sample
Input data: Filename
Output data: Filename

Table 1: Example Service.

First, we define the work by feeding in the names of each service, and then define the execution order. In the Workman user interface, a single box indicates work and an arrow shows the order of execution. For example, “A->B” means “execute work B after work A”. We have to define the input data of “Service A”, but we don’t have to define the output data, because Workman can create the file name automatically. Next, we define the data relationships. If we wish to use Output data 2 from Service A as input data, we define these data with a variable of the same name. At this point Workman may begin the workflow.

After Workman receives a “Start” message Workman invokes the “protein-CM-sample” service through Jobman. Jobman sends a “Run” message along with the definition of the input data to Service A. Service A will receive a file defining its input data and start calculating. When Service A finishes calculating, it sends a “Finished” message and the name of its output data to Jobman. Jobman sends these messages to Workman. Workman copies the output data to user storage. Next, Workman invokes “Service B” through Jobman. In this case, “Output data 2” will be used as the input data of Service B and this file name is generated automatically. Workman thus executes many services automatically.

We now explain how Workman supports the changing of parameters. When a user wants to change parameters while a workflow is running, they must take the following steps:

1. Select the work whose parameters are to be changed from the running workflow using Workman’s user interface
2. Set the new parameters
3. Instruct Workman to “Restart workflow”
When a user has completed these actions, the workflow will be restarted. The advantage of using Workman is that we do not have to consider which services we should restart and which data we should reuse. Workman divides the running pieces of work into work which should be stopped and work which should be continued. Workman then copies data which are reusable and automatically sets the file name of the input data for the first piece of work restarted.

5. Conclusion

There are several projects and researches focused on the use of the Grid System for bioinformatics. In the myGrid[9, 10] project, which is a UK e-Science project funded by the EPSRC involving five UK universities, some middleware and tools have been developed and these are used in the biosciences. Folding@home[11] is a distributed computing project making use of a large number of idle computers around the world. Anyone can easily join this project by downloading a single program and installing it. Such a person can then help research into understanding protein folding, mis-folding, and related diseases.

These projects facilitate the study of proteins according to their unique technology. In our project, we support a trial-and-error method using an original workflow system. Furthermore, by using our job management system scientists can utilize multiple computational resources spread over many bases.

In this paper, we have shown a methodology for gridifying a system for protein structure prediction. We believe that this methodology is applicable to other scientific programs. In addition, we have revealed important factors that improve the reliability of results. Our system works effectively and it has been shown that we can construct a system supporting the straightforward acquisition of reliable results.

References