Proceedings of the 28th IEEE
EMBS Annual International Conference
New York City, USA, Aug 30-Sept 3, 2006

Pluggable Application Server Framework

H. Wang[1], F. Tan[1], A. Sabnis[3], X. Fu[1], P. Volarath[2], R. Harrison[1, 2]
[1]Department of Computer Sciences, Georgia State University, Atlanta, GA, USA
[2]Department of Chemistry, Georgia State University, Atlanta, GA, USA
[3] Department of Biology, Georgia State University, Atlanta, GA 30303, USA

Abstract— **Building a system based on variants of disparate individual components/programs is usually a challenging task. The components/programs are not designed to communicate with each other but the whole system construction does require a seamless collaboration among them. In this paper, targeting at protein structure prediction, a pluggable application server framework is presented. The framework is capable of combining various existing programs into an efficient unit and the design is devoted to provide a model which is able to integrate heterogeneous components/programs into the system quickly without modifying their codes. Based on the model, different components can be plugged into the system with easy configuration, which would lead to a self-configurable and adaptive system. A protein structure prediction server implementation was developed by applying the design model and the real implementation emphasizes the efficiency and simplicity of the system construction. The method and model are generic and can be applied to other system design as well.**

I. Introduction

Many computation problems in biology use standalone applications. They can be connected onto the construction of application servers. In this paper, we concentrate on presenting a pluggable distributed framework for building application server.

Although the framework can work for different applications, this paper describes designing a protein structure prediction server. Protein structure prediction is one of the most important problems in Bioinformatics. Comparative modeling is the most successful and widely-used approach to model protein structure. Comparative modeling exploits the structural similarities between proteins by constructing a three dimensional structure based upon the known structure(s) of one or more related proteins. The basic steps of a comparative modeling can be outlined in fig. 1.

The modeling process can require intensive human interaction and will always consume computational resources. Removing human interaction and human vainability became the driving force behind the development of automatic protein prediction server.

Although there are many protein structure prediction servers [1,2], they each implement a single algorithm for molecule modeling. Most of the currently prevailing algorithms assume certain approximation which reflect authors' particular viewpoint. If the input data lies outside of the range where these approximations are valid then the server will generally fail. Therefore, designing a protein structure prediction server that can use several alternative algorithms is a worthwhile task.

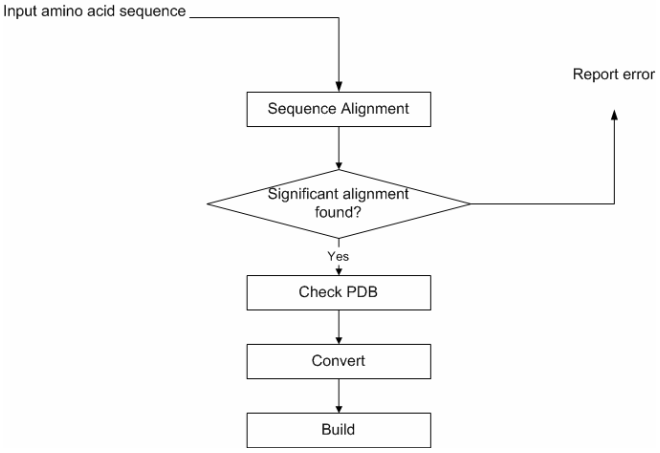

Fig.1 Basic procedure for comparative modeling

In this paper, we propose an application server framework which can automatically plug in and switch potential inherent heterogeneous programs without code modification. To make the framework transparent to disparate programming, we propose the concept of a wrapper daemon which would construct the bridge between the system and underlying programs. A wrapper daemon is actually a dormant component residing in the server which is activated when it receives the kind of input it is programmed to act on. The working of these daemons and their interdependency on underlying programming is a critical feature in the architecture of our structure prediction server framework.

The introduction of daemon is successful in decoupling the system architecture and support programming. An experienced user can choose any of the available programs and integrate them into the system based on his knowledge. The parameters and running order of each program can also be configured, all of which lead to a framework with configurable running paths. The fact that the framework allows the use of different molecular modeling programs, along with managed data integration, results in the ability to test performance of different programming in different circumstances. The test results can be further used to develop an intelligent prediction server: capable of choosing the right component and path to process the analyzed input.

This work was partially supported by NIH1 P20 GM065762-01A1, the Georgia Research Alliance and the Georgia Cancer Collation. Dr. Robert Harrison is a Georgia Cancer Collation Distinguished Scholar.

II. SYSTEM DESIGN AND METHOD

The system mainly consists of two parts: databases and daemons. This section will introduce their designs and roles in the entire system architecture.

A. Database Definition

A comparative modeling practice involves large data access. The data basically belongs to 2 sources:

- PDB Database [3]. The archive of experimentally-determined, biological macromolecule 3-D structures.
- FASTA Database [4]. FASTA format sequence data.

Although above two databases are maintained by non-profit organizations and are accessible publicly through certain interfaces, for performance and stability concerned, we build a dedicated in-house database for the system.

In addition, our design has a user database which keeps track of user submits and their statuses.

B. Wrapper daemon and its communication method

From system point view, wrapper daemon is designed to build up communication channel between system and underlying programming. Building up a universal daemon is inherently challenging. The diversity of program developments has been long demonstrated. A grogram can be coded by any language for any platform. It can take the variety of inputs and generate mixture of outputs.

By general definition, the task of a wrapper daemon is to retrieve information from the system and modify its format to fit as an input for a dedicated program and convert the program output to something that system can recognize. In our design, a wrapper daemon can be used to hook a collection of programs. As a result, we have a much more flexible daemon architecture. First of all, we introduce the concept of program pool, which is a container of a collection of programs sharing certain interfaces. Each program in the program pool is indexed and accessible by the wrapper daemon; secondly, a configuration file is used to resolve the connection between wrapper daemon and program pool. By design, the configuration file specifies the link to one of programs in the programming pool. After reading the configuration file, wrapper daemon knows which underlying program it needs to talk to. The basic architecture is depicted in Fig. 2.

The most challenging task is to build up a general channel between daemon and its underlying program. As the design specification requires the wrapper daemon to act on the underlying programs, we first need to define the activities the daemon should have. As we know, the activities of wrapper daemon are primarily dominated by the nature of its underlying programs, which demonstrates huge diversity. However, focusing only on their input/output can restrain the range of response activities that the daemon can have. The exploit of potential input/output suggests that designed daemon may need following activities:

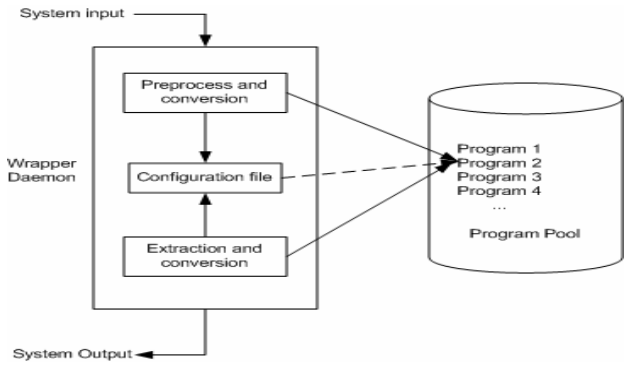

Fig.2 A wrapper daemon architecture in the system

1. Input data provision: A wrapper daemon needs to prepare all required data for underlying programming, and fed it with recognizable input formats.
2. Output data extraction: A wrapper daemon needs to capture all kinds of outputs, obtain related information and transferred back to system.
3. Simulation of human interaction: For a console program designed to talk directly with human being, a wrapper daemon needs to act on behalf of human being.

Activities 1 and 2 can be accomplished by equipping wrapper daemon with two modules: input preprocessing and output postprocessing. For each program, the detail of input and output is listed and readable by above two modules. In case input data is needed and not provided locally, remote database visit will be activated by the module to retrieve the required data. By reading the format description of input and output, modules can make the data conversion and extraction on the fly.

Activity 3 is needed when a console program is not designed to feed all commands within one input. The behavior of that program is similar to `csh` or `bash` in a sense that they can process any predefined commands in any order and may be just in an idle state of waiting for new command. o work with this type of program, wrapper daemon needs to maintain communication channel throughout the whole program life cycle. It is well known that each console program contains three streams in common: standard input, standard output and standard error output. As default, standard output and standard error output are flushed to the monitor, and keyboard typing is directed into the standard input. However, these streams can be reassigned and in the system design, we take this feature to keep a live communication channel between wrapper daemon and console program. The basic idea can be illustrated by figure 3. At first, in wrapper daemon, three buffers corresponding to three standard streams were allocated. Then while launching program, the daemon specifically directs each standard stream to one of the allocated buffers. Subsequently, by accessing these three buffers the simulation module, which we defined to handle activity 3, keeps real time connection with console program. The simulation module parses the buffer data to recognize the console output, and input can be fed just by direct bytes into the related buffer.

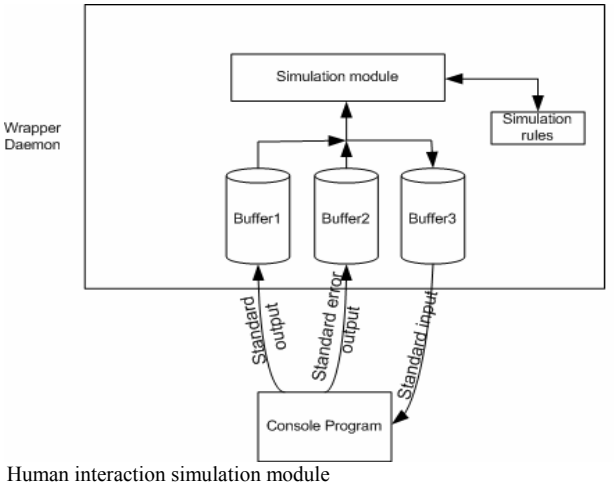

Fig.3 Human interaction simulation module

Roughly speaking, the calculation based on comparative modeling requires two steps: 1) sequence alignment 2) molecular modeling programming. As a result, in the design of system architecture, we define a two daemon structure for molecular calculation. These two wrapper daemons handle the two steps required by the calculation. As previously introduced, daemons are free to choose related programs through program pool. We can simply name two daemons as sequence alignment daemon and molecular modeling daemon. It should be noted that the different names only reflect the difference of their handling purpose. The structure and implementation of these two daemons are exactly same.

```
<?xml version="1.0" encoding="utf-8" ?>
+ <!--  -->
- <MoleculeModeling>
  - <package name="AMMP">
    - <target name="ammp">
        <exec dir="c:/PredictionServer/Program/" />
      </target>
      <SequenceAlignment />
    - <steps totalSteps="3">
      + <step stepID="1" desc="PDB Retrieval">
      + <step stepID="2" desc="PDB Template">
      - <step stepID="3" desc="Conversion">
        - <target name="preammp">
            <exec dir="c:/PredictionServer/Program/" />
          </target>
        </step>
      - <step stepID="4" desc="AMMP">
        - <target name="AMMP">
            <exec dir="c:/PredictionServer/Program/" />
          </target>
        - <parameters>
            <parameter type="InputFile" name="AMMPInput" value="predicted.ammp" />
            <parameter type="OutputFile" name="PDBFile" value="predicted.pdb" />
            <parameter type="ScriptFile" id="1" value="read.ammp" />
            <parameter type="ScriptFile" id="2" value="tsearch.ammp" />
            <parameter type="ScriptFile" id="3" value="polish.ammp" />
            <parameter type="ScriptFile" id="4" value="output.ammp" />
          </parameters>
        </step>
```
Fig.4 A molecule modeling XML configuration file

The pluggable feature of wrapper daemon is achieved by a two layer configuration files. For each daemon, there is a configuration file indicating current setup. It tells the current wrapping program's name and corresponding working directory, etc. In each program's working directory, there exists an exclusive configuration file which elaborates every aspect of that program. The two layer configuration files are actually XML files that follow predefined XML schema. By parsing configuration files, wrapper daemon is well aware of its underlying program (Fig. 4).

- Sequence Alignment Daemon: In a sequence alignment processing, input is simply a valid protein sequence. The whole sequence alignment usually does not require human interaction as long as necessary data and parameters are specified. Two most popular packages to handle sequence alignment are BLAST and FASTA [5,8,9]. Therefore, as a sequence alignment wrapper daemon, the activities mainly involve launching a binary program and providing necessary user defined parameters. In addition, some sequence alignment algorithm, like PSI-BLAST, performs several iterations. As a result of which during each iteration, the daemon needs to wait until the program exits, then retrieve necessary intermediate output and continue with the next iteration if needed.
- Molecular Modeling Programming Daemon: In contrast to sequence alignment processing, molecular modeling programming needs several steps and normally requires human interaction. The steps and interaction requirement are reflected in the configuration file. Thus, wrapper daemon knows the exact path and is able of launch simulation model to handle human interaction as needed.

III. IMPLEMENTATION

An implementation of protein structure prediction server based on PSI-BLAST sequence alignment and AMMP package [6] was developed. Although currently the implementation deals with only these two algorithms, they cover most of the difficulties mentioned in the paper.

The implementation server takes two specific daemons for handling these two steps, both of the daemons are coded using Java language. For the XML parsing, we use Java API for XML Processing (`JAXP`) interface and chose `Xerces2` Parser implementation [7].

- Sequence Alignment Daemon: the daemon reads the associated XML file to obtain the input sequence, the location of PSI-BLAST package, and parameters to run the search. Because PSI-BLAST may run several iterations, the wrapper daemon used Process class in Java to launch the program, and call `wait()` member function to be notified of each iteration. After the execution, the output of PSI-BLAST is saved into a text file and the wrapper daemon is going to extract needed sequence alignment information from that file.
- Molecular Modeling Programming Daemon: AMMP only recognizes its own language, hence, the wrapper daemon needs to make several preprocessing jobs before finally calling the program. (1) PDB template generation. Based on the sequence alignment result, the daemon will access in-house database for retrieving known PDB file, after that, a consistency check and necessary modification are given between the sequence alignment and PDB file. The program `newho` in `AMMP` package was implemented to make that conversion, and this program is called by the daemon. (2) Chemical properties generation. The PDB file does not specify all of the information needed to

perform a molecular mechanics calculation. In particular, force field specific information, the bond length and force constants must be merged with the coordinate data in order to build a model. Additionally, information must be supplied for atoms missing from the PDB coordinates. This step is performed by the `presp4` program in the `AMMP` package. The daemon is in responsible of calling the program smoothly. While it can be driven by scripts, `AMMP` is often run interactively. In the implementation, `AMMP` program is wrapped into a Java process and its standard streams capture can be done by calling Process class member function: `getInputStream()`, `getOutputStream()` and `getErrorStream()`. In order to avoid blocking the computing unit when processing streams, a threaded based class StreamGrabber is developed and all capture streams will be passed into this class, and are further directed to Buffer-Reader class. `StreamGrabber` uses `BufferReader` to parse underlying program output and determine the right interaction.

The implementation was designed to be reliable and scalable. The architecture (Figure 5) was designed to use two layers: the user interface layer and the computing service layer. The user interface layer is the only part that is directly visible to the internet and passively accessed by the computing unit via the database access interface. The isolation will ensure that the server is relatively immune to internet based attaches. The user interface layer acts as an interface between the clients and the computing unit, it is responsible for extracting the sequence from the database. Computer service layer is the place where predicting calculation really occurs. The system is intended to have a pool of computing units, which is the basic unit to perform a calculation task. The exact number of computing units will be scaled based on the volume of user request in a standard time period.

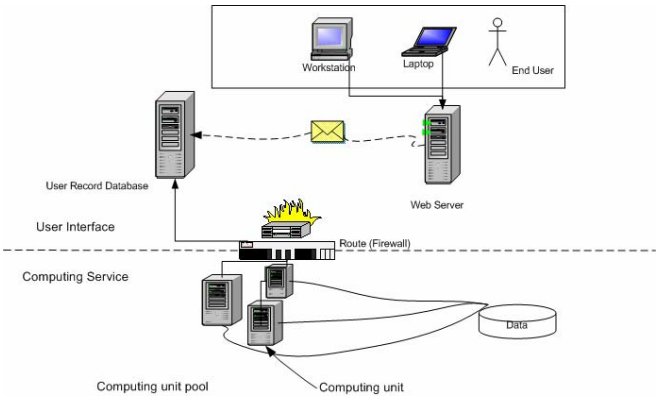

Fig.5 System architecture

The whole prediction server system (http://bmcc2.cs.gsu.edu/PredictionServer) takes different running environment. It reflects our security belief that unless a hacker is good at both operating systems, it would be hard for him to comprise our entire system. The user interface part was completely built up under Windows system. ASP.NET was used to develop web server, and web site is designed to run under Internet Information Server 5.1.

IV. CONCLUSION

In this paper, the design of a prediction system framework is presented. The whole design is primarily targeted to provide a platform which is pluggable to disparate programs for protein structure prediction. The framework can be configured by the end users to reflect their preferences and knowledge. The framework provides the unique up-to-date database support which is isolated from underlying programs used for prediction. In a field like protein structure prediction involving many heuristic algorithms, the framework, along with the configurable pathway, can be used as a live bench to determine the performance of different package combinations in terms of input sequence. The performance result can be further used to build up an intelligent protein prediction server which is self adaptive to varying inputs based on its benchmark knowledge base. The basic idea of the design can be applied to other system as well.

REFERENCES

[1] Bujnicki JM, Elofsson A, Fischer D, Rychlewski L " LiveBench1: continuous benchmarking of protein structure prediction servers" .Protein Science 10:352–361,(2001)

[2] Fischer D, Elofsson A, Rychlewski L, Pazos F,Valencia A, Rost B, Ortiz AR, Dunbrack RLJr CAFASP2: "The second critical assessment of fully automated structure prediction methods".Proteins 45 (Suppl 5):171–183, (2001b)

[3] PDB Documentation http://www.rcsb.org/pdb/info.html

[4] FASTA database file. http://www.ensembl.org/info/data/ftp_files.html

[5] Carlos Setubal, Joan Meidanis "Introduction to Computational Molecular Biology" PWS Publishing Company, Boston, USA

[6] Harrison, R. W., Reed, C.C. and Weber, I.T. "Analysis of Comparative Modeling Predictions for CASP2 Targets 1, 3, 9 and 17" Proteins: Struct. Funct. & Genet., Suppl. 1.68-73, (1997)

[7] Java API for XML processing http://java.sun.com/webservices/jaxp/index.jsp

[8] Pearson WR. "Rapid and sensitive sequence comparison with FASTP and FASTA" Methods Enzymol. 183:63-98 (1990)

[9] Stephen F. Altschul, Warren Gish, Webb Miller, Eugene W. Myers and David J. Lipman "Basic Local Alignment Search Tool" J. Mol. Biol. 215: 403-410 (1990)