ZIB Structure Prediction Pipeline: Composing a Complex Biological Workflow through Web Services

Patrick May, Hans-Christian Ehrlich, Thomas Steinke

Zuse Institute Berlin, Takustr.7,14195 Berlin, Germany,
{patrick.may,ehrlich,steinke}@zib.de

Abstract. In life sciences, scientists are confronted with an exponential growth of biological data, especially in the genomics and proteomics area. The efficient management and use of these data, and its transformation into knowledge are basic requirements for biological research. Therefore, integration of diverse applications and data from geographically distributed computing resources will become a major issue. We will present the status of our efforts for the realization of an automated protein prediction pipeline as an example for a complex biological workflow scenario in a Grid environment based on Web services. This case study demonstrates the ability of an easy orchestration of complex biological workflows based on Web services as building blocks and Triana as workflow engine.

1 Introduction

In the post-genomics era protein structure prediction is still one of the major challenges in bioinformatics research, because the full understanding of the biological function of proteins requires knowledge about its three-dimensional (3D) structure [1]. Although experimental methods are providing high-resolution structure information, they are still expensive in costs and duration. On the other hand, fully automated computational structure prediction tools have made rapid progress over the last years (Critical Assessment of Structure Prediction, CASP [2] and CAFASP [3]). Protein structure prediction is a process which typically involves multiple data processing and decision steps, iterations, as well as the parallel execution of time-consuming applications. In comparison with sequence homology searches with, for example, Blast [4] structure prediction is a much more complex scenario.

Web services provide a well-defined, standardized access to methods independent from its implementation and programming platform. As pointed out in [5], Web services are an emerging technology paradigm for distributed computing. Problem solving environments with standardized workflow description languages (e.g. BPEL4WS [6]) are providing solutions to these problems. Suitable workflow engines support the orchestration [7] of workflows with Web services as building blocks. Complex workflows contain compute and/or storage intensive tasks.

Regarding compute intensive tasks, the support of parallel execution models, e.g. task farming or MPI parallelized programs, are therefore an imperative pre-requisite. We selected Triana [8] for the following reasons: it easily integrates Web services, and provide a graphical user interface allowing an easy workflow orchestration. Furthermore, it can represent workflows as non-DAG and the workflow engine can be interfaced with selected Grid services which is an important pre-requisite for the next step towards the realization of our workflow in a Grid environment.

There are many initiatives pushing biological applications towards the use of workflow, Grid and/or Web service technologies. Gao et. al. [9] describe a drug discovery data-mining system using Web services. Mattoso et. al. [10] built MHOLline, an automated workflow for comparative modelling with legacy applications using Web service technology. They used BPEL4WS for defining the workflow and IBM BPWS4J 1.0.1 [11] as workflow engine. PROSPECT-PSPP [12] is a fully automated structure prediction pipeline using SOAP for remote procedure calls. Hence, the problem of consistency in data integration projects, which combine common information from different data sources, is still a major obstacle for obtaining unique information sets and data quality in secondary biological databases. There are successful data integration (data warehouse) projects, for example, MSD [13] or Columba [14] with their focus on structural data. The Helmholtz Open BioInformatics Technology initiative (HOBIT) [15] is dedicated to build a technology platform for concatenating applications and resources together with an efficient communication tier for bioinformatics resource access based on Web services.

The scope of this paper is to show that fully automated workflows with Web service components are able to integrate heterogeneous applications and data into a standalone, demanding biological application scenario. One can expect that Web services are one starting point for the realization of a collaborative e-science infrastructure in Grid environments. In this paper we use protein structure prediction as a paradigm for complex biological problems.

The organization of the article is as follows. In the next section we describe the ZIB structure prediction pipeline. Section 3 presents the Web services, followed by workflow definition in Triana. As the most interesting result, we compare the overhead timings of the traditional monolithic workflow using a PERL implementation of the workflow engine with the Web service based workflow using the Triana workflow engine in section 4. Finally, section 5 gives a summary and an outlook towards future work.

2 ZIB Structure Prediction Pipeline

The ZIB structure prediction pipeline has been designed and implemented for the 6th CASP experiment in 2004 [2, 16]. In order to provide a fully automated protein prediction tool, the pipeline integrates various prediction and analysis steps. The whole pipeline is designed modular, so that improved methods can

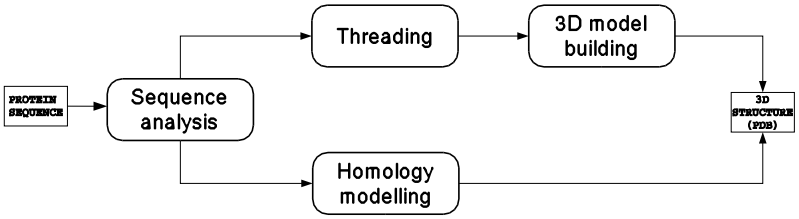

Fig. 1. Schematic representation of the ZIB structure prediction pipeline.

be substituted in, as they become available. Fig. 1 shows the global pipeline architecture.

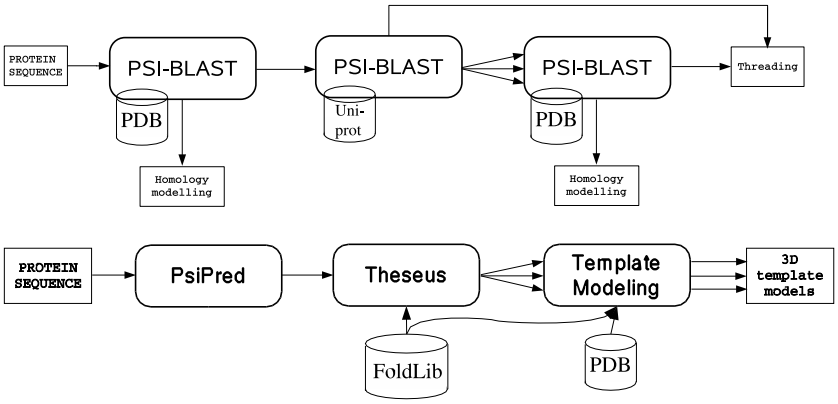

Fig. 2. Sub-Workflows: (top) Sequence analysis, (bottom) Threading

The first step in the workflow is the identification of suitable template structures for homology modelling (Fig. 2, top). A sequence analysis sub-workflow is passed to search for homologous sequences with known structures. Successive PSI-Blast searches are performed in order to find suitable templates. If no template structure has been found in the PDB (Protein Data Bank [17]) database, a second PSI-Blast search in the Uniprot [18] database is initiated followed by parallel PSI-Blast searches in the PDB database starting from the Uniprot hits. If a structural template has been found, an atomic structural model will be generated with MODELLER [19]. If no suitable structural template is detectable, the structure will be predicted by our protein threading implementation. The

threading procedure (Fig. 2, bottom) starts with a secondary structure prediction using PsiPred [20]. PsiPred provides a 3-state prediction (helix, strand, loop) together with a reliability score for every sequence position. THESEUS [21] is an MPI-parallelized implementation of a protein threading based on a multi-queue branch-and-bound search algorithm to find the optimal sequence-to-structure alignment through a library of template structures [22]. From the highest scoring template structures the most probable template is selected and submitted to the loop modelling procedure where different 3D models are generated in parallel. Here, MODELLER is used to model the loop regions and the sidechain atoms of the given template structure. At the end, a full atom structure for the target sequence is provided.

The most time consuming step in the sequence analysis procedure is the PSI-Blast search against the Uniprot database (minutes to one hour of CPU time). The prediction of a 3D protein model by threading typically takes many minutes to hours, the modelling steps with MODELLER some minutes to few hours. The types of data to be exchanged and processed are protein sequences, structures and alignments. Data formats are either application specific, e.g. the PDB format for protein structures or Blast-XML for PSI-BLAST, or in-house developed XML schemes for a standardized data exchange.

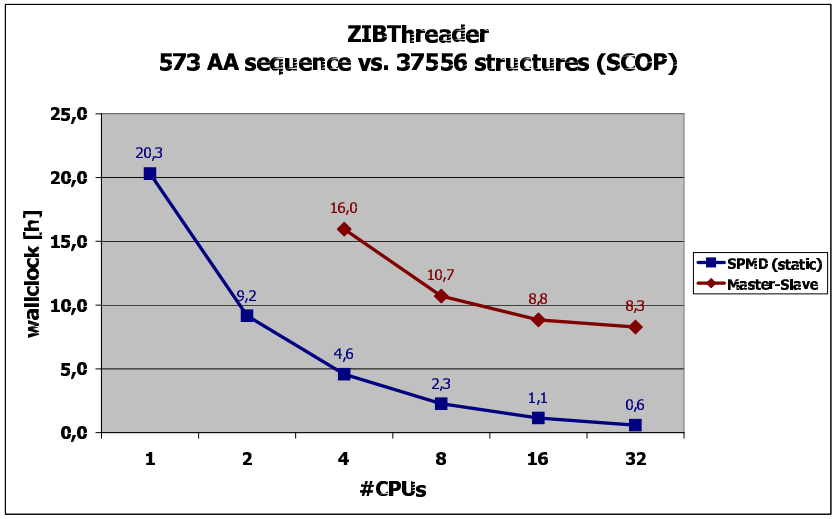

Fig. 3. Performance of the two parallel threading architectures.

Fold recognition by threading can be parallized by assigning each of a subset of template structures to a different process. Our parallel threading core is implemented in C++ and uses either MPI for message passing or POSIX threads. Two kind of parallel architectures are designed: a Master-Slave (MS) version,

and a Single-Program-Multiple-Data (SPMD) version. In the MS architecture the central component is the MySQL database. A master process or POSIX thread distributes each outstanding template structure to a slave process waiting for work. Based on a first-come-first-serve protocol a dynamic load balancing scheme can be realized. In the SPMD architecture the content of the MySQL template structure database is dumped into a binary file which is cloned on each compute node on a Linux cluster. The template structures are distributed in a static scheme amongst the MPI processes, i.e., each MPI process performs its own subset. Having all template structures processed, one MPI process gathers all results from the remaining concurrent MPI processes. The SPMD approach is significant faster over the MS architecture (shown in Figure 3: the red line indicates the MS and the blue line the SPMD architecture). The drawback of the MS architecture is the time determing database connections: the central database server can not timely satisfy the requests from all the slave processes. The SPMD architecture has the extra advantage of parallel I/O. To show the time efficiency of our implementation, we can process a protein sequence consisting of 573 amino acids against 37556 templates structures representing the whole SCOP template database in about 36 minutes on 32 cpus on a IA32 Myrinet Linux cluster .

3 Workflow Implementation with Web Services

3.1 Compute environment

The implemented pipeline runs on compute resources locally available at our site. Web service applications can either run on a compute cluster complex consisting of an IA32 Myrinet Linux cluster and a Cray XD1 system, or on desktop machines. The resources of the compute cluster complex are managed by a job management system providing a single point of control (job submission and job control). The Triana workflow engine runs either on local desktop machines or on the cluster front-end node. More technical details of the hardware and software configuration can be found elsewhere [23].

3.2 Web Service Implementations

For the ZIB structure prediction pipeline the following applications, part of them are legacy codes, were wrapped into Web services:

- A local Blast program package including the standard sequence analysis tools BLAST and PSI-BLAST as well as FastaCMD for retrieving FASTA formatted sequences. The analysis tools are implemented with standard options (e.g. database, *E*-value). Input is a protein sequence.
- A local PsiPred version which requires as input a protein sequence.
- The in-house developed parallelized threading program THESEUS which needs a protein sequence, the predicted secondary structure from PsiPred and a position-specific scoring matrix from PSI-Blast as input.

– A local MODELLER version, which requires the template identifier and a sequence as input and optional the threading model in the loop modelling case.

The Web services are designed asynchronous, because of high computational demands of the applications. They provide methods for submitting the job and for collecting the results. A generic polling Web service has been implemented which monitors the job status on the local batch system and informs the workflow engine process that a job has finished and results are available. Parallelization over data is achieved by handling lists as data structure in Web services.

Data, either XML or unfiltered file contents, are transferred through the body of the SOAP message. The Web services were implemented using two different languages: the Blast, PsiPred, and the MODELLER Web services are written and deployed with Java and Apache Axis [24], the THESEUS and the polling Web service are written in Python.

3.3 Workflow Definition with Triana

Our structure prediction workflow is defined and executed by Triana [25]. Triana allows the user to build and execute workflows consisting of Triana units and Web services. Triana is written in Java and supports the implementation of self-written Triana units easily. The Triana GUI provides a Unit Wizzard for generating a skeleton Triana unit code, an editor and an interface for compiling the code. Fig. 4 shows as an example the source code snippet of our `makeFastaCMDrequest` unit:

– The unit has one input port (line 4).
– The unit has two output ports (lines 19 and 20).
– The input for the unit is a Blast result in a XML document.
– The XML document is parsed for possible Blast hits (line 12:`BlastXML.Hit`).
– The output ports send a request string to the FastaCMD Web service with a list of corresponding hit identifiers that are needed by FastaCMD to fetch the corresponding protein sequences, together with the input XML document, which is needed in the further workflow.

Web services can directly be imported into Triana canvas from its WSDL description. By specifying the URI of the WSDL document the Web service is known to Triana and usable as Triana unit. Input and output object types are given by the WSDL description.

Triana supports the loop as control element in its workflow description. Every resource-intensive application has to be submitted to the local batch system. Then, our polling Web service method `pollStatus` determines the status of a given job (queued, running, finished). The orchestrated Triana sub-workflow unit `pollStatusLoop(PDB)` includes Triana's `LOOP` unit that initiates either the next polling cycle or exits the loop. The decision is made depending on the output of the `pollStatus` Web service: if the stop condition is send (meaning job is finished) the job identifier (being the input of the `pollStatus` Web service) is

```
1  /* provides a list of sequence ids for FastaCMD  */
2   public void process() throws Exception {
3     //get the input from the triana module node
4     BlastXML.BlastFtObj BlastXmlObj = (BlastXML.BlastFtObj) getInputAtNode(0);
5
6     StringBuffer IDs = new StringBuffer(); // array of sequence ids
7     Iterator hitIter = BlastXmlObj.HitStorage.keySet().iterator();
8.....
9     while (hitIter.hasNext()){
10      String tmpKey = (String) hitIter.next();
11      //get each hit
12      BlastXML.Hit tmpHit = (BlastXML.Hit) BlastXmlObj.HitStorage.get(tmpKey);
13      //extract the hit accession and generate FastaCMD request string
14      if (requestCount != BlastXmlObj.HitStorage.size()-1)
15        IDs.append(tmpHit.getHit_accession()).append("\n");
16      else
17        IDs.append(tmpHit.getHit_accession());
18    }
19    outputAtNode(0, IDs.toString());
20    outputAtNode(1, BlastXmlObj);
21  }
```

Fig. 4. Example of a Triana unit: `makeFastaCMDrequest`

passed to the next step in the workflow, which is usually a `getResults` Web service method.

The data driven parallelization of sub-workflows (high-throughput computations) like the invocation of a series of PSI-Blast searches against the PDB database in the sequence analysis sub-workflow (see Fig. 2) is implemented through lists. The Blast Web service works on lists of protein sequences as input data, i.e. the Web service method `runBlast` submits all input sequences to the batch system and returns a list of job identifiers that can be handled by the polling Web service.

4 Web services vs. "Scripting"

In this section we compare the performance of two different implementation scenarios of our protein structure prediction pipeline focussing on the associated overhead costs. The "traditional" approach uses a specifically written workflow engine implemented in PERL (scripting approach). The second implementation is based on the Triana workflow engine with Web services as described in the previous sections. In both implementations the time-consuming bioinformatics applications ran as jobs scheduled via the local batch system, i.e. in both cases the steps (1) job submission to compute nodes of the cluster, (2) monitoring the job status (polling), and (3) delivery of results after job termination were identical. Data analysis steps either implemented into Web services or as Triana units have their counterparts in the PERL implementation as well. The Triana workflow engine process was either started on the cluster front-end (TRIANA/Linux, in Table 2) or on a desktop machine connected through a switched 100 Mb/s Ethernet network to the cluster front-end (TRIANA/Windows). The PERL script ran on the front-end only (PERL/Linux).

Additionally, we have implemented a pipeline version ZIB-jws including the BLAST Web service from DNA Data Bank Japan (DDBJ)[1] for searches against the PDB instead of our local PSI-BLAST installation (see Fig. 2). The DDBJ provides their Web services also synchronous as well as asynchronous. Therefore, the DDBJ BLAST Web service could be directly plugged into our request and response with polling architecture. For the PERL implementation we used the SOAP::Lite library.

Three experiments were performed to estimate the overhead costs:

1. PDB sequence analysis + homology modelling (Homology(PDB)),
2. PDB sequence analysis + UNIPROT sequence analysis + parallel PDB sequence analysis + homology modelling (Homology(UNIPROT)),
3. PDB sequence analysis + UNIPROT sequence analysis + PsiPred prediction + Threading + loop modelling (Threading).

To maintain the desired partial workflow, for every experiment a specific protein target sequences was used:

1. a sequence that had a significant identity ($> 40\%$) to a sequence in the PDB,
2. a sequence that showed no significant identity to a PDB sequence but where sequences with sufficient similarity were detectable in the PDB through iterative search against the Uniprot database sequences with, and
3. a sequence with no similarity detectable in PDB and Uniprot.

All experiments were repeated 10 times for the normal ZIB prediction pipeline (ZIB) as well as for the ZIB-jws version to have a minimal representative set of results. Timing information is based on the `gettimeofday` system call.

In all experiments, the total workflow execution times (wall-clock time) as well as the time spent in the execution of non-application steps, i.e. the workflow overhead execution time, were recorded. In the later case, the wall-clock is fetched before and after the invocation of an asynchronous Web service. The mean values for the total workflow times (Table 1), and the mean values of the overhead times (Table 2) over all approaches are summarized. Note that the total workflow execution time depends heavily on the resource usage of the compute complex, since it includes the job waiting time in a batch queue. Fortunately, for that study these numbers are more of a formal interest since our main focus is to estimate the overhead costs of our workflow design with Web services compared to the scripting approach. The 3D structures obtained by the different approaches were validated to manually obtained reference data in order to ensure the correctness of any workflow implementation.

As expected, the overhead times in the Triana/Web service implementation is by an order of magnitude larger compared to the monolithic PERL approach (Table 2). Overall, the total execution times for the ZIB-jws pipeline are slightly better than those for the in-house version. This is because the DDBJ Web service can only execute simple Blast runs, whereas our in-house implementation uses the more time consuming PSI-BLAST. This implies that any workflow engine

[1] http://xml.nig.ac.jp/wsdl/index.jsp

Table 1. Typical total workflow execution times (wall-clock, in seconds) for the two workflow versions

experiment	ZIB	ZIB-jws
Homology(PDB)	134.1	133.5
Homology(UNIPROT)	358.1	358.0
Threading	503.2	503.1

Table 2. Mean workflow overhead execution times (in seconds) for the two workflow versions

Implementation/Platform	Homology				Threading	
	PDB		UNIPROT			
	ZIB	ZIB-jws	ZIB	ZIB-jws	ZIB	ZIB-jws
TRIANA/Linux	0.120	0.120	0.300	0.300	0.301	0.300
TRIANA/Windows	0.123	0.120	0.305	0.302	0.305	0.303
Perl/Linux	0.012	0.009	0.080	0.078	0.075	0.071

invokes the Blast result polling services less frequently than in the in-house scenario. Furthermore, the overhead execution time did not include the time for data transfers between the workflow engine process and a Web service. Compared to the total workflow execution (wall-clock) times, the overhead times in both workflow versions of the Web services based workflow implementation are about four orders of magnitude lower and therefore practically negligible (less than 0.1%).

5 Summary and Outlook

We have presented the implementation of a protein structure prediction pipeline as Web service-based workflow using Triana. We have demonstrated that Web services are a versatile technology to integrate various, heterogeneous methods into one stand-alone, fully automated and biological demanding application scenario.

The design of such complex workflows with Web services as buildings blocks are well supported by the Triana problem-solving environment. Additionally, Triana supports the workflow design and the development of self-written Triana units. Within the Triana framework, the processing of workflows with Web services is characterized by an additional, but expected performance overhead. Fortunately, these additional "costs" are usually negligible for workflow scenarios where the time-dominating factors are compute-intensive tasks. Such a coarse-grain segmentation of workflows is the appropriate approach for taking the advantages of Web service technology in real-world scenarios. Moreover, we see today the overall benefit of using the Web service approach in the modular design of the workflow, the improved maintainability, and the more intuitive

plug-in of new modules accessible as Web services. Those modules may run locally or are provided by external service providers. The "only" concern of the end user is the functional interface and the corresponding input and output data.

Having a Web service based workflow in place fulfills an important precondition for moving the application scenario into a Grid environment. As long as all services are statically defined in the workflow any flexibility for improving the throughput performance is missing. The next step is to apply brokering services at runtime to select appropriate compute and storage resources for compute and/or storage intensive workflow steps. This approach will allow the transparent use of geographically distributed resources for the workflow processing. It enables the implementation of high-throughput pipelines for solving complex biological questions.

This work constitutes the base for further developments towards a workflow system for protein structure prediction based on Grid services in a Grid environment. Several additional pre- and post-processing steps to further improve the quality of the predicted models will enhance the ZIB structure prediction pipeline. This development is also part of the German MediGRID [26] project.

Acknowledgement

This work is partly funded by BMBF (Germany), grant no. 031U209A (Berlin Center for Genome Based Bioinformatics, BCB) and BMBF (Germany), grant no. 01AK803F (D-Grid/MediGRID-Ressourcenfusion für Forschung in Medizin und Lebenswissenschaften). We would like to thank Falko Krause and Jonas Maaskola for their implementation of local Web services, and the Triana group at Cardiff University for their support.

References

1. D. Baker, A. Sali, Protein structure prediction and structural genomics, Science 294 (2001) 93–96.
2. J. Moult, A decade of CASP: progress, bottlenecks and prognosis in protein structure prediction, Curr. Opin. Struct. Biol. 15 (2005) 285–289.
3. D. Fischer, C. Barret, K. Bryson, A. Elofsson, A. Godzik, D. Jones, K. Karplus, L. Kelley, R. MacCallum, K. Pawowski, B. Rost, L. Rychlewski, M. Sternberg, CAFASP-1: critical assessment of fully automated structure prediction methods, Proteins 3 (1999) 209–217.
4. S. F. Altschul, T. L. Madden, A. A. Schaffler, J. Zhang, Z. Zhang, W. Miller, D. J. Lipman, Gapped BLAST and PSI-BLAST: a new generation of protein database search programs, Nuc. Acids Res. 25 (1997) 3389–3402.
5. S. Majithia, M. Shields, I. Taylor, I. Wang, Triana: A graphical web service composition and execution toolkit, in: IEEE International Conference on Web Services (ICWS'2004), 2004.
6. F. Curbera, T. Andrews, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, S. Weerawarana, Business Process Execution Language for Web services, V.1.0, Available via http://www-106.ibm.com/developerworks/webservices/library/ws-bpel.

7. F. Leymann, Web Service Flow Language (WSFL), version 1.0.
8. Triana, Available via http://www.trianacode.org.
9. H. T. Gao, J. H. Hayes, H. Cai, Integrating biological research through web services, Computer (2005) 26–31.
10. M. C. Cavalcanti, R. Targino, F. A. Baião, S. C. Rössle, P. M. Bisch, P. F. Pires, M. L. M. Campos, M. Mattoso, Managing structural genomic workflows using web services., Data Knowl. Eng. 53 (1) (2005) 45–74.
11. IBM BPWS4J, Available via http://www.alphaworks.ibm.com/tech/bpws4j.
12. J. Guo, K. Ellrott, W. J. Chung, D. Xu, S. Passovets, Y. Xu, PROSPECT-PSPP: an automated computational pipeline for protein structure prediction, Nucleic Acid Res. 32 (Web Server Issue) (2004) W522–W525.
13. S. Velankar, P. McNeil, V. Mittard-Runte, A. Suarez, D. Barrell, R. Apweiler, K. Henrick, E-MSD: an integrated data resource for bioinformatics, Nucleic Acids Res. 33 (Database issue) (2005) D262–265.
14. S. Trissl, K. Rother, H. Muller, T. Steinke, I. Koch, R. Preissner, C. Froemmel, U. Leser, Columba: an integrated database of proteins, structures, and annotations, BMC Bioinformatics 6 (1) (2005) 81–92.
15. HOBIT (Helmholtz Open Bioinformatics Technology) project, Available via http://hobit.sourceforge.net.
16. E. Michalsky, A. Goede, R. Preissner, P. May, T. Steinke, A distributed pipeline for structure prediction, in: CASP6 Methods Abstracts, 6th Meeting on the Critical Assessment of Techniques for Protein Structure Prediction, Gaeta, Italy, 2004, pp. 112–114.
17. H. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. Bhat, H. Weissig, I. Shyndyalov, P. Bourne, The protein data bank, Nucl. Acids Res 28 (2000) 235–242.
18. A. Bairoch, R. Apweiler, C. Wu, W. Barker, B. Boeckmann, S. Ferro, E. Gasteiger, H. Huang, R. Lopez, M. Magrane, M. Martin, D. Natale, C. O'Donovan, N. Redaschi, L. Yeh, The universal protein resource (uniprot), Nucleic Acids Res. 1 (33) (2005) 154–159.
19. M. Marti-Renom, A. Stuart, A. Fiser, R. Sanchez, F. Melo, A. Sali, Comparitive protein structure modeling of genes and genomes, Annu. Rev. Biophys. Biomol. Struct. 29 (2000) 291–325.
20. L. McGuffin, K. Bryson, D. Jones, The PSIPRED protein structure prediction server, Bioinformatics 16 (2000) 404–405.
21. P. May, T. Steinke, THESEUS - protein structure prediction at ZIB, ZIB Report 06-24.
22. R. H. Lathrop, A. Sazhin, Y. Sun, N. Steffen, S. S. Irani, A multi-queue branch-and-bound algorithm for anytime optimal search with biological applications, Genome Informatics 12 (2001) 73–82.
23. BCB-Cluster, Available via http://elfie.bcbio.de.
24. Apache Axis, Available via http://ws.apache.org/axis.
25. I. Taylor, I. Wang, M. Shields, S. Majithia, Distributed computing with triana on the grid, Concurrency and Computation:Practice and Experience 17 (2005) 1–18.
26. MediGRID, Available via http://www.medigrid.de/.