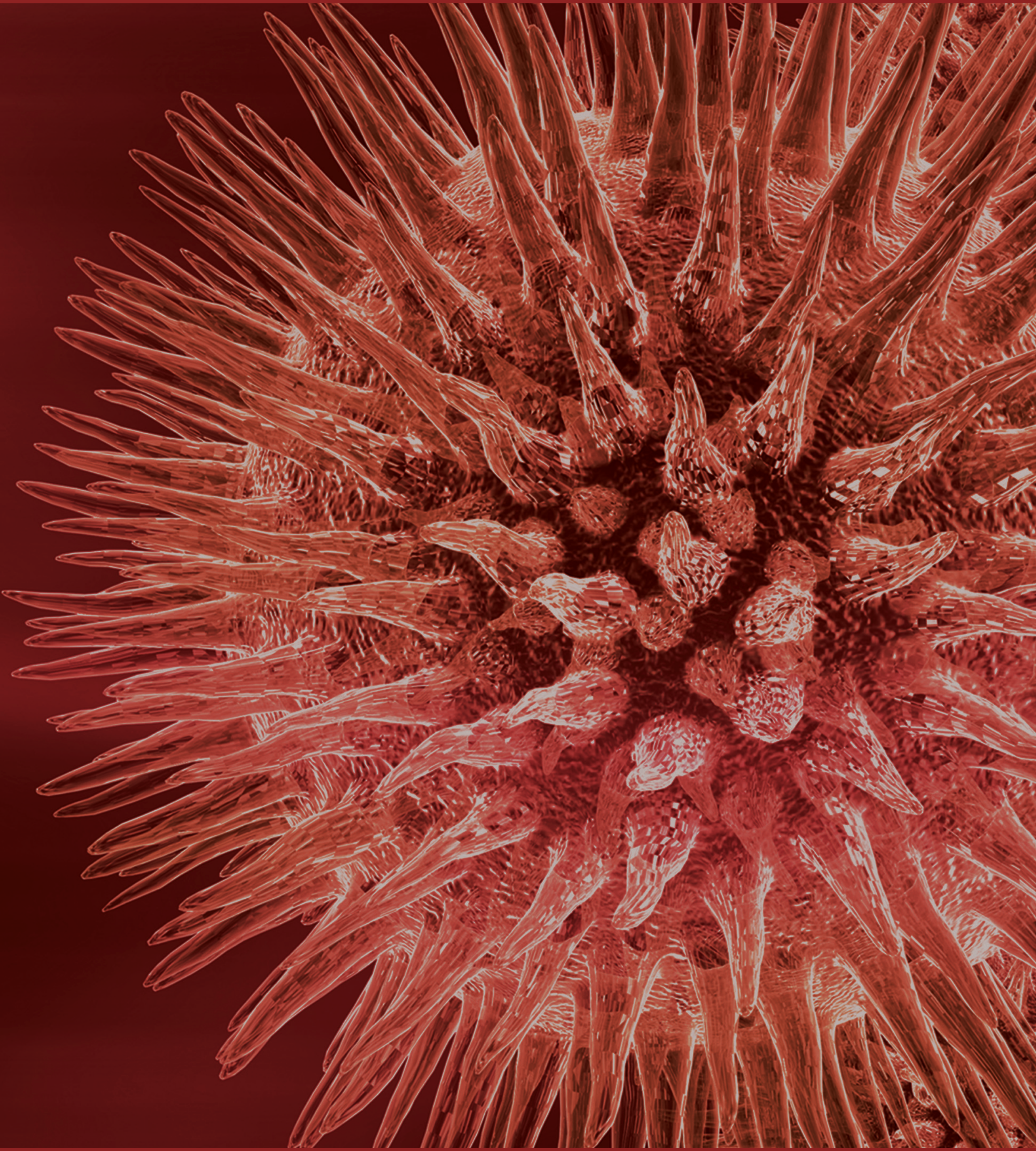
BioMed Research International

Biocloud: Cloud Computing for Biological, Genomics, and Drug Design

Guest Editors: Ching-Hsien Hsu, Chun-Yuan Lin, Ming Ouyang, and Yi Ke Guo



Biocloud: Cloud Computing for Biological, Genomics, and Drug Design

BioMed Research International

Biocloud: Cloud Computing for Biological, Genomics, and Drug Design

Guest Editors: Ching-Hsien Hsu, Chun-Yuan Lin, Ming Ouyang, and Yi Ke Guo

Contents

Hindawi Publishing Corporation
BioMed Research International
Volume 2013, Article ID 909470, 3 pages
http://dx.doi.org/10.1155/2013/909470

Editorial

Biocloud: Cloud Computing for Biological, Genomics, and Drug Design

Ching-Hsien Hsu,[1] Chun-Yuan Lin,[2] Ming Ouyang,[3] and Yi Ke Guo[4]

[1] *Department of Computer Science and Information Engineering, Chung Hua University, Hsinchu 30012, Taiwan*
[2] *Department of Computer Science and Information Engineering, Chang Gung University, Taoyuan 333, Taiwan*
[3] *Department of Computer Engineering & Computer Science, University of Louisville, Louisville, KY 40292, USA*
[4] *Department of Computing, Imperial College London, London SW7 2AZ, UK*

Correspondence should be addressed to Ching-Hsien Hsu; chh@chu.edu.tw

Received 5 September 2013; Accepted 5 September 2013

1. Introduction

Cloud computing has emerged rapidly as an exciting new paradigm that offers a challenging model of computing and services. Leveraging cloud computing technology, bioinformatics tools can be made available as services to anyone, anywhere, and through any device. The use of large biodatasets, its highly demanding algorithms, and the hardware for sudden computational resources makes large-scale biodata analysis an attractive test case for cloud computing.

This special issue aims to foster the dissemination of high quality research in any new idea, method, theory, and technique related to cloud computing and bioinformatics and to showcase the most recent developments and research in cloud computing for biological, genomics, and drug design, considering genomics and drug design on the cloud, biological tools on the cloud, biodatabase on the cloud, cloud-based biocomputing, and all kinds of successful applications. The research papers selected for this special issue represent recent progresses in the aspects, including theoretical studies, practical applications, new analysis and modeling technology, programming methodologies, and experimental prototypes. All of these papers not only provide novel ideas and state-of-the-art techniques in the field but also stimulate future research in the biocloud environments.

2. Cloud-Based Biological Service

Large-scale scientific experiments have an ever increasing demand for High Performance Computing (HPC) resources. The paper by R. De Paris et al. "*wFReDoW: a cloud-based web environment to handle molecular docking simulations of a fully flexible receptor model*" proposes a cloud-based web environment, called web Flexible Receptor Docking Workflow (wFReDoW), which reduces the CPU time in the molecular docking simulations of FFR models to small molecules. It is based on the new workflow data pattern called Self-adaptive Multiple Instances (P-SaMI) and on a middleware built on Amazon EC2 instances. P-SaMI reduces the number of molecular docking simulations while the middleware speeds up the docking experiments using a High Performance Computing (HPC) environment on the cloud. The experimental results show a reduction in the total elapsed time of docking experiments and the quality of the new reduced receptor models produced by discarding the nonpromising conformations from an FFR model ruled by the P-SaMI data pattern.

On the other hand, as bioinformatics is embracing cloud computing, the paper by L. Kaján et al. entitled "*Cloud prediction of protein structure and function with PredictProtein for Debian*" reports the release of PredictProtein for the Debian operating system and derivatives, such as Ubuntu, Bio-Linux, and Cloud BioLinux. The PredictProtein suite is available as a standard set of open source Debian packages. The release covers the most popular prediction methods from the Rost Lab, including methods for the prediction of secondary structure and solvent accessibility (profphd), nuclear localization signals (predictnls), and intrinsically disordered regions (norsnet). The authors also present two case studies that successfully utilize PredictProtein packages for high performance computing in the cloud.

3. High-Performance Biological Computing

Although the computer science technologies can be used to reduce the costs of the pharmaceutical research, the computation time of the structure-based protein-ligand docking prediction is still unsatisfied until now. The paper by J.-L. Chen et al. entitled "*A high performance cloud-based protein-ligand docking prediction algorithm*" presents a novel docking prediction algorithm to accelerate the docking prediction. The proposed algorithm works by leveraging two high-performance operators: (1) the novel migration (information exchange) operator is designed specially for cloud-based environments to reduce the computation time; (2) the efficient operator is aimed at filtering out the worse search directions. The simulation results illustrate that the proposed method outperforms the other docking algorithms compared in this paper in terms of both the computation time and the quality of the end result.

The proteome-wide analysis of protein-ligand binding sites and their interactions with ligands is potentially an important source of information in structure-based drug design and in understanding ligand cross-reactivity and toxicity. The paper by C.-L. Hung and G.-J. Hua entitled "*Cloud computing for protein-ligand binding site comparison*" develops a cloud computing service, called Cloud-PLBS, combining SMAP and Hadoop framework, and it is deployed on a virtualization cloud computing platform. Cloud-PLBS takes advantage of the MapReduce paradigm as means of management and parallelizing tool under massive number of protein-ligand binding site pairs compared under the experiment. Cloud-PLBS provides both a web portal and scalability for biologists to address a wide range of compute intense questions in biology and drug discovery. The performance experiment shows that it is desirable for molecular biologists to investigate the protein structure and function analysis under reasonable time constraints by using our cloud service.

An understanding of the activities of enzymes could help to elucidate the metabolic pathways of thousands of chemical reactions that are catalyzed by enzymes in living systems. The paper by C.-C. Huang et al. entitled "*Enzyme reaction annotation using cloud techniques*" proposes the enzyme reaction prediction (ERP) method as a novel tool to deduce enzyme reactions from domain architecture. We used several frequency relationships between architectures and reactions to enhance the annotation rates for single and multiple catalyzed reactions. The deluge of information which arose from high-throughput techniques in the postgenomic era has improved our understanding of biological data, although it presents obstacles in the data-processing stage. The high computational capacity provided by cloud computing has resulted in an exponential growth in the volume of incoming data. Cloud services also relieve the requirement for large-scale memory space required by this approach to analyze enzyme kinetic data.

4. Big Data Intelligence

The rate of accumulation of biomolecular data is increasing astonishingly. This information explosion is being driven by the development of low-cost, high-throughput experimental technologies in genomics, proteomics, molecular imaging, amongst others. Success in the life sciences will depend on our ability to rationally interpret these large-scale, high-dimensional data sets into clinically understandable and useful information, which in turn requires us to adopt advances in informatics. The paper by J. Chen et al. entitled "*Translational biomedical informatics in the cloud: present and future*" demonstrates the utility and promise of cloud computing for tackling the big data problems. The authors outline their vision that cloud computing could be an enabling tool to facilitate translational bioinformatics research. Biomedical cloud, given the proper architecture, could integrate all the petabytes of available biomedical informatics data in one place and process them on a continuous basis. In this way, we would continuously observe the connections between genotypic profiles and phenotypic data. We can envision that the cloud-supported translational bioinformatics endeavours will promote faster breakthroughs in the diagnosis, prognosis, and treatment of human disease.

Based on the concepts of resources on demand and pay as you go, scientists with no or limited infrastructure can have access to scalable and cost-effective computational resources. However, the large size of next generation sequencing (NGS) data causes significant data transfer latency from the client's site to the cloud, which presents a bottleneck for using cloud computing services. The paper by S. A. Issa et al. entitled "*Streaming support for data intensive cloud based sequence analysis*" provides a streaming-based scheme to overcome this problem, where the NGS data is processed while being transferred to the cloud. The proposed scheme targets the wide class of NGS data analysis tasks, where the NGS sequences can be processed independently from one another. This study also provides the elastream package that supports the use of this scheme with individual analysis programs or with workflow systems. The experiments presented in this paper show that the proposed solution mitigates the effect of data transfer latency and saves both time and cost of computation.

5. GPU Technologies

With the endeavor to narrow performance overhead, the virtualization technology expands its coverage from cloud computing to high performance computing such as biological computation. Recently, biological applications start to be re-implemented into the applications which exploit many cores of GPUs for better computation performance. The paper by H. Jo et al. entitled "*Exploiting GPUs in virtual machine for BioCloud*" proposes a BioCloud system architecture that enables VMs to use GPUs in cloud environment. The proposed system exploits the pass-through mode of PCI express (PCI-E) channel. By making each VM to be able to access underlying GPUs directly, applications can show almost the same performance as when those are in native environment. The proposed scheme multiplexes GPUs by using hot plug-in/out device features of PCI-E channel. By adding or removing GPUs in each VM in on-demand manner, VMs in the same physical host can time-share their GPUs. The performance results showed that this prototype is highly effective for biological GPU applications in cloud environment.

The Smith-Waterman (SW) algorithm searches for a sequence database to identify the similarities between a query sequence and subject sequences. However, this algorithm is prohibitively high in terms of time and space complexity. The paper by S.-T. Lee et al. entitled "*GPU-based cloud service for Smith-Waterman algorithm using frequency distance filtration scheme*" presents a novel Smith-Waterman algorithm with a frequency-based filtration method on GPUs rather than merely accelerating the comparisons yet expending computational resources to handle such unnecessary comparisons. A user friendly interface is also designed for potential cloud server applications with GPUs. Experimental results indicate that reducing unnecessary sequence alignments can improve the computational time by up to 41%.

6. Conclusions

All of the above papers address either big data intelligence issues in cloud or cloud-based biological service or propose novel application models in the various cloud and ubiquitous fields. They also trigger further related research and technology improvements in application of Biological computing. Honorably, this special issue serves as a landmark source for education, information, and reference to professors, researchers, and graduate students interested in updating their knowledge about or active in biological computing, biocloud services and management, and novel application models for bioCloud services and computing systems.

Acknowledgments

The guest editors would like to express sincere gratitude to numerous reviewers for their professional effort, insight, and hard work put into commenting on the selected articles which reflect the essence of this special issue. We are grateful to all authors for their contributions and for undertaking two-cycle revisions of their manuscripts, without which this special issue could not have been produced.

Ching-Hsien Hsu
Chun-Yuan Lin
Ming Ouyang
Yi Ke Guo

Hindawi Publishing Corporation
BioMed Research International
Volume 2013, Article ID 140237, 11 pages
http://dx.doi.org/10.1155/2013/140237

Research Article

Enzyme Reaction Annotation Using Cloud Techniques

Chuan-Ching Huang,[1] Chun-Yuan Lin,[2] Cheng-Wen Chang,[3] and Chuan Yi Tang[4]

[1] *Department of Computer Sciences, National Tsing Hua University, Hsinchu 300, Taiwan*
[2] *Department of Computer Sciences and Information Engineering, Chang Gung University, Taoyuan 333, Taiwan*
[3] *Department of Applied Chemistry, National Chiao Tung University, Hsinchu 300, Taiwan*
[4] *Department of Computer Sciences and Information Engineering, Providence University, Taichung 433, Taiwan*

Correspondence should be addressed to Chuan Yi Tang; cytang@pu.edu.tw

Received 7 December 2012; Revised 12 July 2013; Accepted 19 July 2013

Academic Editor: Ching-Hsien (Robert) Hsu

An understanding of the activities of enzymes could help to elucidate the metabolic pathways of thousands of chemical reactions that are catalyzed by enzymes in living systems. Sophisticated applications such as drug design and metabolic reconstruction could be developed using accurate enzyme reaction annotation. Because accurate enzyme reaction annotation methods create potential for enhanced production capacity in these applications, they have received greater attention in the global market. We propose the enzyme reaction prediction (ERP) method as a novel tool to deduce enzyme reactions from domain architecture. We used several frequency relationships between architectures and reactions to enhance the annotation rates for single and multiple catalyzed reactions. The deluge of information which arose from high-throughput techniques in the postgenomic era has improved our understanding of biological data, although it presents obstacles in the data-processing stage. The high computational capacity provided by cloud computing has resulted in an exponential growth in the volume of incoming data. Cloud services also relieve the requirement for large-scale memory space required by this approach to analyze enzyme kinetic data. Our tool is designed as a single execution file; thus, it could be applied to any cloud platform in which multiple queries are supported.

1. Introduction

Enzymes are biochemical agents that efficiently catalyze the conversion of substrates into products in organisms. Enzymes are essential to the metabolic activity of living systems, and they share 3 features: catalytic power, specificity, and regulation [1]. Catalytic power is the ratio of the rate of an enzyme-catalyzed reaction to the rate of the uncatalyzed reaction. Enzyme-catalyzed reactions provide faster rates than traditional biochemical processes because enzymes reduce the energy required for biochemical reactions. Enzymes perform specific actions, and their selection should be specific to the desired reaction; thus, the use of enzymes can avoid competing reactions from producing side products. Consequently, enzyme applications are increasingly being employed in industrial applications. Enzyme activities can be optimized to provide metabolic reaction rates that are appropriate to cellular requirements.

The catalytic power and specificity of enzymes can enhance productivity in industrial applications. A recent study published by the BBC research group estimated that the global market for industrial enzymes was at \$3.3 billion in 2010 and was expected to reach \$4.4 billion by 2015 [2]. Enzymes involved in digestion, such as lipase, protease, and amylase, are classed as hydrolases. The Nomenclature Committee of the International Union of Biochemistry and Molecular Biology (NC-IUBMB) classified enzymes into 6 groups: oxidoreductases, transferases, hydrolases, lyases, isomerases, and ligases. According to the NC-IUBMB scheme and the Enzyme Commission's (EC) system, an enzyme reaction is assigned a 4-numerical-block number [3]. The method presented in our study can facilitate enzyme annotation, and is also valuable in followups to biochemical studies and applications, including metabolic process investigations and drug discovery.

There are 3 main types of enzyme reaction annotations: sequence similarity, chemical structure comparison, and domain architecture fingerprint. Certain annotation methods, such as profils pour l'identification automatique du métabolisme (PRIAM) [4] and Catalytic Families (CatFam) [5], are based on protein sequences. These methods generate high-level profiles from sequences to represent and determine protein catalytic functions. The EnzymeDetector [6] annotation method uses sequence similarity analysis and a comprehensive enzyme database, BRaunschweig ENzyme DAtabase (BRENDA) [7], which is manually extracted from the literature. The Enzyme Function Inference by Combined Approach (EFICAz) [8] method adopts and combines various independent sequence-based methods.

The second type of enzyme reaction annotation is based on chemical structure comparison because the conversion of a particular reactant into a product with a specific molecular structure in an uncatalyzed chemical reaction can often be achieved by enzyme catalysis in an organism. Problems are frequently encountered when an enzyme catalyzes several reactions and when the same reaction is catalyzed by different enzymes. Several reported computational methods exist for assigning EC numbers that use the physicochemical and topological properties of reactants, products, and bonds involved in the reaction [9–12].

Domain architecture fingerprint is the third type for enzyme reaction annotation. Substrates bind to an enzyme at its active site, where they undergo reaction. An enzyme reaction is intimately linked to the compact protein structure of a domain. As a general rule, enzymes of similar domain architectures catalyze similar reactions; this creates a difficult mapping problem from the architecture space into the reaction space. Various machine-learning methods have been applied to the mapping problem, including the association rule algorithm [13], the decision tree method [14], support vector machines [15], neural networks [16], and other classification schemes including domain teams [17], probabilistic rule-based models [18], and a weighted domain architecture comparison tool, the Feature Architecture Comparison Tool (FACT) [19].

The advent of genomics technologies, including next-generation sequencing and mass spectrometry-based flow cytometry [20, 21], creates an exponential growth in the volume of data. Cloud technologies provide large computing capacity, and this allows for the integration of distributed large-scale facilities for managing user requests and providing cost-efficient responses. Platform as a Service (PaaS) is provided by several companies, including Google, Microsoft, and Amazon. Microsoft's DryadLINQ execution engine and its application to the Alu clustering problem and an expressed sequencing tag (EST) assembling program in Apache Hadoop are extensions of the Google MapReduce platform [22]. Our proposed scheme requires large-scale computer memory for estimating and ranking each subset based on the domain architecture enumeration phase measurements. The results of queries when using this scheme are efficiently managed by the cloud's distributed architectures. Because adopting cloud technology enables annotation schemes to provide new architecture, the global enzyme market is expected to benefit from the increases in production capacity made available by the new architecture.

2. Materials and Methods

Proteins comprise polypeptide chains that form several compact, occasionally loosely connected, global units called structural domains. Regarding the protein structure, structural domains are considered fundamental units of protein function, folding, and evolution [23]. It is reasonable to consider a protein as one type of domain architecture consisting of a set of domains. The SUPERFAMILY structural domain database, integrated into the InterPro database (release 33.0), is adopted for constructing the domain architectures of proteins. For example, a Q5VT25 protein consists of the domain architecture with the SUPERFAMILY domains SSF50729, SSF56112, and SSF57889, such that the set {SSF50729, SSF56112, SSF57889} is considered to represent Q5VT25. Moreover, different proteins may share the same domain architecture of {SSF50729, SSF56112, SSF57889}, such as Q9BZL6 and E0W264. A particular reaction may be catalyzed by different enzymes, and an enzyme can often mediate more than one reaction. The resulting complex relationship between the set of domain architectures and the set of enzyme reactions remains a difficult problem, even after simplifying by considering a protein as one type of domain architecture. In this study, we identified proteins and recorded their corresponding domain architectures and enzyme reactions in our database.

2.1. Data Sets.

From the viewpoint of protein function, enzymes are agents of metabolic function, which control the rate of biochemical activities in living organisms [1]. The first block of the EC number indicates to which of these 6 groups an enzyme belongs. The second and third blocks indicate subclass and sub-subclasses according to the enzyme's associations with the chemical features of the reactants and products of the reaction system. The final block is a sequential number. Enzymes are collected based on their corresponding EC numbers from the UniProt Knowledgebase (UniProtKB), such as Q5VT25 associated with EC 2.7.11.1, Q9BZL6 with 2.7.11.13, and E0W264 with 1.3.1.74 and 2.7.11.13.

UniProtKB [24] is a comprehensive protein sequence and annotation resource. It comprises UniProtKB/Swiss-Prot and UniProtKB/TrEMBL sections. The literature-based records in the Swiss-Prot section are manually annotated and analyzed computationally by curators. The TrEMBL section contains records that are annotated automatically, using qualitative computational analysis methods. Enzyme reactions described by either UniProtKB/Swiss-Prot or /TrEMBL are collected. The InterPro [25] database is an integrated resource of protein signatures in which protein domains held in different member databases are cross-referenced. We used the SUPERFAMILY member database [26] to investigate the relationship between domain architectures and enzyme reactions. All enzymes assigned EC numbers were collected from the Swiss-Prot and TrEMBL sections of UniProtKB (release 2011_07). We extracted the proteins that (1) had specific EC numbers and (2) were cross-referenced to SUPERFAMILY (version 1.73) in the InterPro database (release 33.0). Based on the integrated material we gathered from the UniProtKB and SUPERFAMILY databases, there are totally 1,664,839

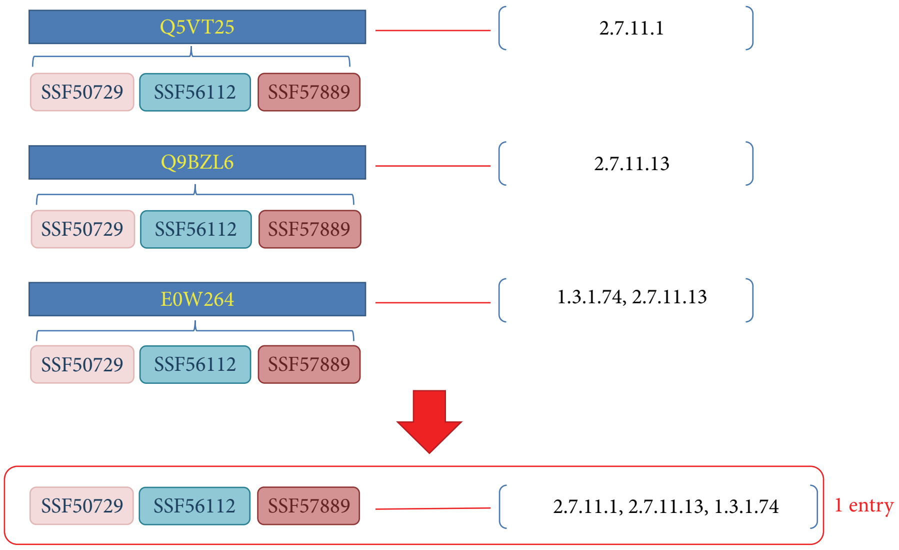


Figure 1: Illustration of an entry.

proteins composed of 1,218 SUPERFAMILY domains and 3,306 related EC numbers.

Relying on the rationale that structural domains are related to protein functions, we integrated enzymes sharing the same domain architecture as a single entry. For example, Q5VT25, Q9BZL6, and E0W264 share the {SSF50729, SSF56112, SSF57889} architecture with EC numbers 2.7.11.1, 2.7.11.13, and 1.3.1.74 and are considered a single entry (Figure 1). There are 5,203 entries collected in this study, and each entry consists of one type of domain architecture associated with several enzyme reactions.

Our proposed method accounts for the frequency of each potential type of domain architecture from a set, and a rank is assigned according to several criteria. After determining the domain architecture that has the greatest score, we obtain the corresponding enzyme reactions.

2.2. Methods. Because domains are fundamental structural units that can fold into a compact block, we considered the appearance of a domain in an enzyme and omitted the repetition of domains. As a result, the number of domain architectures is nearly 5 times the number of types of SUPERFAMILY domains but does not grow exponentially. This shows a tendency for one domain to accompany others to form one type of domain architecture for a protein. The ERP method is used to predict enzyme reactions from components of domain architectures. In the model-building process, there are 2 main phases: "domain architecture enumeration" and "enzyme reaction ranking."

Before building the prediction model, we divided 5,203 entries into 2 sets, the training set and the testing set. The training set is used to establish the prediction model and the testing set is adopted for verification. The details of the model simulation are described in the 5-fold cross-validation section.

2.2.1. The Enzyme Reaction Prediction Method. The first phase of model building is based on the rationale that one domain has a tendency to accompany others to form one type of domain architecture. We enumerated all possible subsets from domain architectures in the training set and estimated each subset according to 4 measurements: comprising existence, succinctness, consistency, and simplicity. The domain architecture candidate with the highest priority was thus obtained. In the second phase, we ranked enzyme reactions in a list according to their intensity values associated with one specific type of domain architecture.

Domain Architecture Enumeration. After inspecting the set of domain architectures, we learned that the number of types of adjacent domains was considerably lower than the numbers encountered when enumerating every possible combination. Thousands of architectures are enumerated exhaustively among all the possible subsets associated with the domain architectures of proteins. A possibility of tandem domains appearing with the expression of enzyme-catalyzed reactions also exists. Thus, we propose the following 4 measurements to sequentially estimate each domain's architecture during the domain architecture enumeration phase. For example, if the domain architecture {SSF50729, SSF56112, SSF57889} could not be found, 6 subarchitectures comprising {SSF50729, SSF56112}, {SSF50729, SSF57889}, {SSF56112, SSF57889}, {SSF50729}, {SSF56112}, and {SSF57889} are considered.

(1) Existence of the Protein Consisting of a Given Domain Subset. In the process of enumerating all possible subsets of domain architectures, many putative subsets may be produced. If one subset matches one type of domain architecture of an enzyme, it is reasonable that this domain subset contributes directly to its catalyzed reactions and is awarded higher priority than other subsets are.

(2) Succinctness Measurement of the Domain Architecture of Enzymes. One reaction can be catalyzed by various enzymes that can comprise a variety of domain architectures. Among them, each subset of one type of domain architecture could also include another type of an enzyme. The $\text{Succinctness}_{\text{domain\_arch}}$ equation (1) is designed to identify the most relevant architecture. Given an enumerated domain subset called domain_arch, we collected a set of entries, $\text{Entries}_{\text{domain\_arch}}$, that have domain architectures containing domains in domain_arch. The number of reactions associated with the entries which have domain architecture that exactly match domain_arch is denoted as $|\text{ECs}_{\text{exact}}|$. The number of reactions associated with the entries that have architectures containing domains in domain_arch is denoted as $|\text{ECs}_{\text{included}}|$. The $\text{Succinctness}_{\text{domain\_arch}}$ measurement is calculated as the ratio of $|\text{ECs}_{\text{exact}}|$ to $|\text{ECs}_{\text{included}}|$. The type of domain subset with a greater $\text{Succinctness}_{\text{domain\_arch}}$ value is assigned higher priority among a set of architecture candidates for the query domain architecture. For example, a query architecture domain_arch consisting of domains SSF56112 and SSF57889 is involved in 10 entries involving 5 types of enzyme reactions, comprising 2.7.10.2, 2.7.11.1, 2.7.11.13, 2.7.1.107, and 1.3.1.74 ($|\text{ECs}_{\text{included}}| = 5$) in Figure 2. The exactly matched architecture {SSF56112, SSF57889} is associated with 3 reactions, 2.7.10.2, 2.7.11.1, and 2.7.11.13, such that the $\text{Succinctness}_{\{\text{SSF56112, SSF57889}\}}$ is estimated as 0.6. We assign priority to the candidate with the greatest succinctness value because the corresponding chemical reactions proceed without requiring auxiliary domains as follows:

$$\text{Succinctness}_{\text{domain\_arch}} = \frac{|\text{ECs}_{\text{exact}}|}{|\text{ECs}_{\text{included}}|}. \quad (1)$$

(3) Multiplicity of Enzyme Reactions from One Type of Domain Architecture. An enzyme can catalyze different reactions; alternatively, different enzymes may share the same domain architecture. Considering a domain subset domain_arch, we collected all entries that have domain architectures containing domains of domain_arch. Among these entries, the number of involved reactions is defined similarly to the definition of $|\text{ECs}_{\text{included}}|$ in the previous paragraph, but we denoted it as k for simplicity. To clearly observe the expression of one specific reaction among various architectures, we separated an entry with multiple reactions into several entries with a single reaction, and the number of entries with a single reaction is counted as N (Figure 3). Furthermore, we also mark the number of entries associated with each reaction EC_i as n_i ($i = 1, \ldots, k$), such that $N = \sum_{i=1}^{k} n_i$. The mean value $\overline{n} = N/k$ is calculated as the average number of entries, and the difference $(n_i - \overline{n})$ is estimated for each reaction EC_i. Because k and $\text{Entries}_{\text{domain\_arch}}$ are variables dependent on the set of domains in domain_arch, we provided $\text{Consistency}_{\text{domain\_arch}}$ (2), which summarizes the different terms and is normalized by N and weighted with (n_i/N) for each reaction for comparison with other architecture candidates.

If the expression of each reaction is equal, then n_i approaches the mean value, such that the consistency value becomes smaller. As the consistency value approaches zero, it unambiguously indicates a strong relationship between enzyme-catalyzed reactions and the corresponding domain architecture:

$$\begin{aligned}\text{Consistency}_{\text{domain\_arch}} &= \frac{n_1}{N}\left(\frac{n_1-\overline{n}}{N}\right) + \frac{n_2}{N}\left(\frac{n_2-\overline{n}}{N}\right) \\ &\quad + \cdots + \frac{n_k}{N}\left(\frac{n_k-\overline{n}}{N}\right) \\ &= \sum_{i=1}^{k} \frac{n_i}{N}\left(\frac{n_i-\overline{n}}{N}\right),\end{aligned} \quad (2)$$

$$\begin{aligned} n_1 &= n_{2.7.10.2} = 2 \\ n_2 &= n_{2.7.11.1} = 7 \\ n_3 &= n_{2.7.11.13} = 6 \\ n_4 &= n_{2.7.1.107} = 1 \\ n_5 &= n_{1.3.1.74} = 1 \end{aligned} \Longrightarrow \begin{aligned} N &= \sum_{i=1}^{k} n_i = 17 \quad (\text{where } k = 5) \\ \overline{n} &= \frac{N}{k} = \frac{17}{5}, \end{aligned} \quad (3)$$

$$\begin{aligned} &\text{Consistency}_{\{\text{SSF56112,SSF57889}\}} \\ &= \frac{n_1}{N}\left(\frac{n_1-\overline{n}}{N}\right) + \frac{n_2}{N}\left(\frac{n_2-\overline{n}}{N}\right) + \frac{n_3}{N}\left(\frac{n_3-\overline{n}}{N}\right) \\ &\quad + \frac{n_4}{N}\left(\frac{n_4-\overline{n}}{N}\right) + \frac{n_5}{N}\left(\frac{n_5-\overline{n}}{N}\right) \\ &= \sum_{i=1}^{5} \frac{n_i}{N}\left(\frac{n_i-\overline{n}}{N}\right) \\ &= \frac{2}{17}\left(\frac{2-(17/5)}{17}\right) + \frac{7}{17}\left(\frac{7-(17/5)}{17}\right) \\ &\quad + \frac{6}{17}\left(\frac{6-(17/5)}{17}\right) + \frac{1}{17}\left(\frac{1-(17/5)}{17}\right) \\ &\quad + \frac{1}{17}\left(\frac{1-(17/5)}{17}\right) \\ &= 0.114878892733564. \end{aligned} \quad (4)$$

(4) Simplicity of Domain Architecture. In the case that no protein matching the query architecture domain_arch is found, the fewest number of domains in an architecture candidate is preferred.

The aforementioned 4 measurements for 6 subsets of the domain architecture {SSF50729, SSF561112, SSF57889} are listed in Table 1. Because each subset has the same domain architecture as another protein, the subset {SSF56112, SSF57889} with the highest succinctness value of 0.6 has the highest priority.

Enzyme Reaction Ranking. After determining the domain architecture for a nonannotated enzyme, various related enzyme reactions can be retrieved from the universe data set (Figure 4). The $\text{Intensity}_{\text{EC}_i}$ (5) is calculated based on the ratio

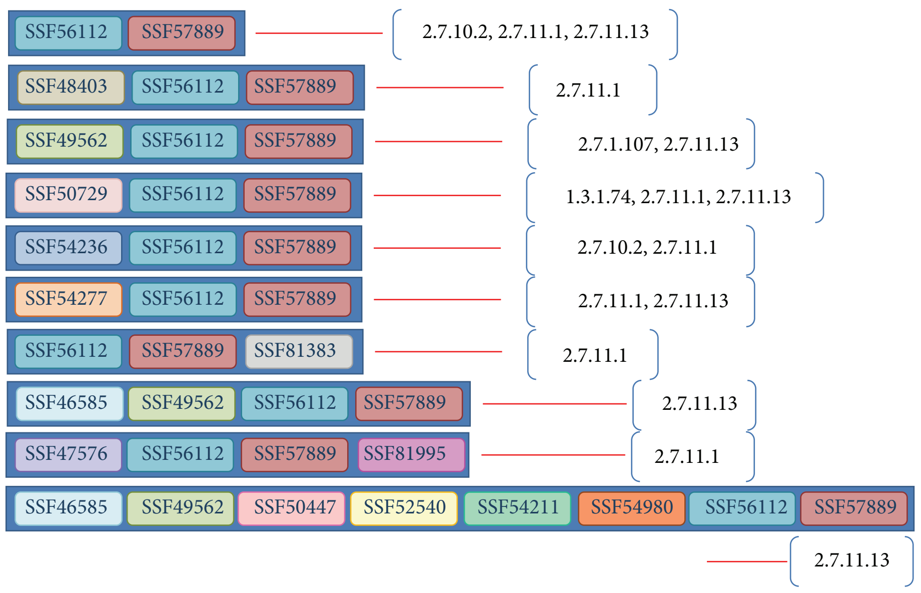


Figure 2: Entries containing domains SSF56112 and SSF57889.

of the mean value to the number of entries associated with EC_i to evaluate the strength of the relationship between the reaction EC_i and the determined domain architecture. An $\mathrm{Intensity}_{\mathrm{EC}_i}$ value of less than 1 indicates that the expression of EC_i is greater than the mean value; thus, low values are preferred. The intensity values corresponding to reactions of the domain architecture {SSF56112, SSF57889} are calculated as follows:

$$\mathrm{Intensity}_{\mathrm{EC}_i} = \frac{\bar{n}}{n_i} = \frac{N}{n_i \times k}, \tag{5}$$

$$\begin{aligned}
\mathrm{Intensity}_{2.7.10.2} &= \frac{\bar{n}}{n_1} = \frac{\bar{n}}{n_{2.7.10.2}} = \frac{17/5}{2} = 1.7,\\
\mathrm{Intensity}_{2.7.11.1} &= \frac{\bar{n}}{n_2} = \frac{\bar{n}}{n_{2.7.11.1}} = \frac{17/5}{7}\\
&= 0.4857142857142857,\\
\mathrm{Intensity}_{2.7.11.13} &= \frac{\bar{n}}{n_3} = \frac{\bar{n}}{n_{2.7.11.13}} = \frac{17/5}{6}\\
&= 0.5555555555555556,\\
\mathrm{Intensity}_{2.7.1.107} &= \frac{\bar{n}}{n_4} = \frac{\bar{n}}{n_{2.7.1.107}} = \frac{17/5}{1} = 3.4,\\
\mathrm{Intensity}_{1.3.1.74} &= \frac{\bar{n}}{n_5} = \frac{\bar{n}}{n_{1.3.1.74}} = \frac{17/5}{1} = 3.4.
\end{aligned} \tag{6}$$

2.2.2. The Association Rule Method. In the field of data mining, the association rule (AR) method is an established method for detecting the relationship between items, particularly for a large database, T. Given a large transaction set, if 2 sets, X and Y, are involved in a rule, $X \rightarrow Y$, 2 constraints must be met: (1) the union of item sets X and Y must appear frequently in T, and (2) the relationship between item sets X and Y is close. A frequent set satisfies the condition that the number of transactions containing that set is higher than the support threshold. If set Y accompanies set X in various transactions, a close relationship exists between sets X and Y. The confidence value can be estimated as the ratio of the number of transactions containing both item sets (X and Y) to that containing item set X alone. If the confidence value of the item sets in a rule is higher than the given confidence threshold, it is placed into the rule set.

2.2.3. Fivefold Cross-Validation. In a classification model, the parameters of the model are optimized to fit the training set as much as possible during the fitting process. An overfitting problem results when another independent validation data set (from the same population) is used to test the model and does not fit as well as the training set did. Cross-validation is a technique used to infer the goodness of fit of a model to a validation set. We used 5-fold cross-validation, in which the sample is randomly divided into 5 subsets: one subset is retained as the testing set, and the other subsets are assigned to the training set. The numbers of entries in each fold for the 4-numerical-block EC number set are shown in Table 2. One round of 5-fold cross-validation involves taking one part as the testing set and the remainder as the training sets, resulting in 20 total rounds of testing.

3. Results and Discussion

A chemical reaction may be catalyzed by more than one enzyme, and an enzyme may catalyze more than one reaction. By considering the relationship between enzymes and chemical reactions as a mapping problem, we create a many-to-many mapping problem. Although there are various methods available that approach this type of problem from different

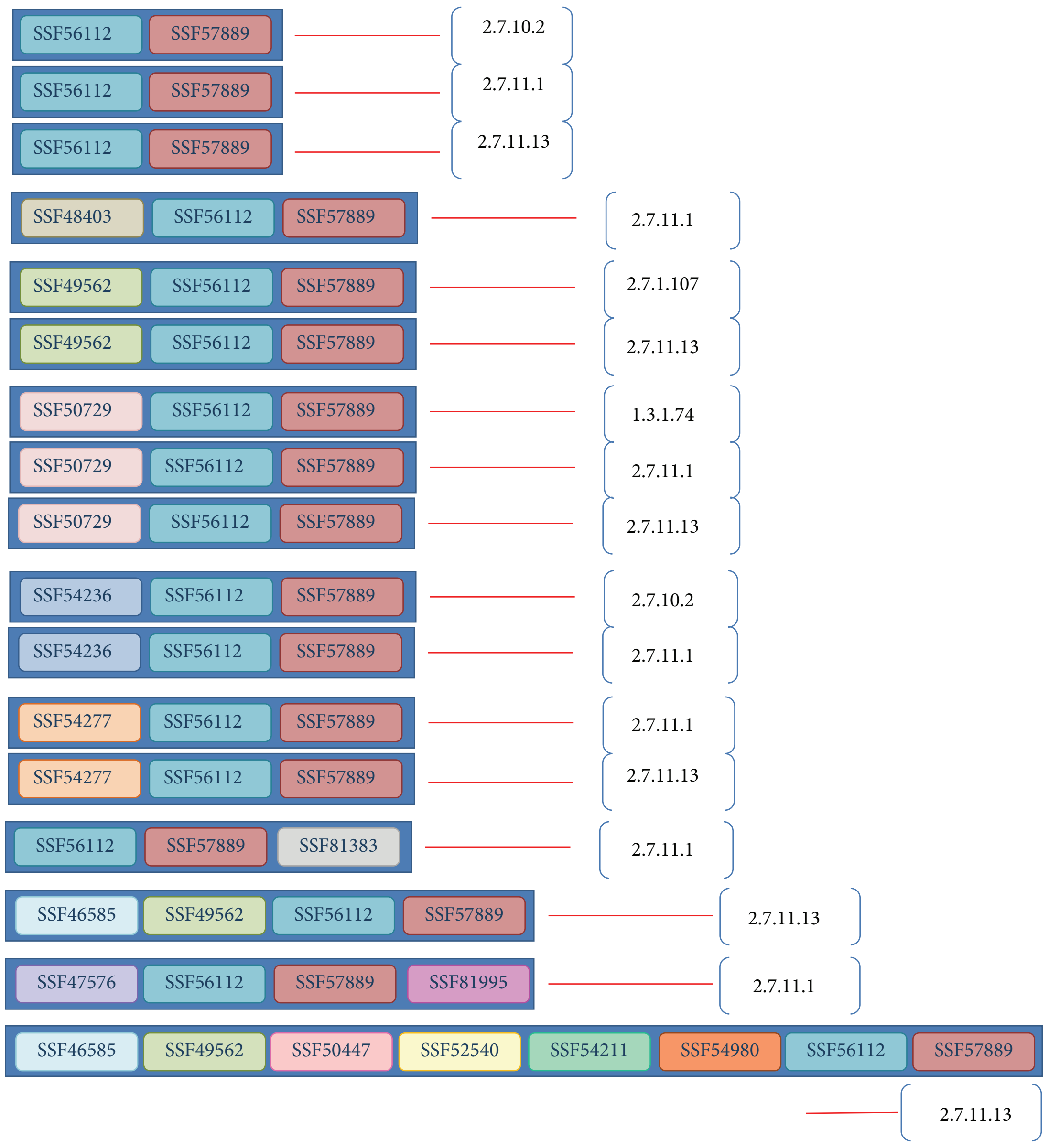


FIGURE 3: Separating entries into certain types of an architecture with one EC number.

TABLE 1: Four measurement values for the six subsets of the domain architecture {SSF50729, SSF56112, SSF57889}.

Domain architecture	Existence	Succinctness	Consistency	Number (domains)
{SSF50729, SSF56112}	1	0.45	0.0688271604938272	2
{SSF50729, SSF57889}	1	0.5	0.058641975308642	2
{SSF56112, SSF57889}	**1**	**0.6**	**0.114878892733564**	**2**
{SSF50729}	1	0.191489361702128	0.0559722260571086	1
{SSF56112}	1	0.595744680851064	0.0765587606003442	1
{SSF57889}	1	0.4	0.11141975308642	1

TABLE 2: The number of entries in each fold.

Data set	Fold_1	Fold_2	Fold_3	Fold_4	Fold_5	Total
4-numerical-block EC number set	1,041	1,041	1,041	1,040	1,040	5,203

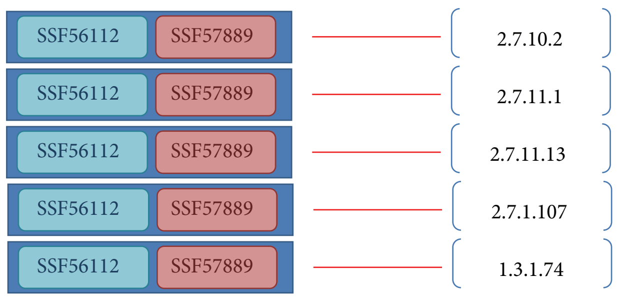


Figure 4: EC numbers connected with the architecture {SSF56112, SSF47889} by the ERP method.

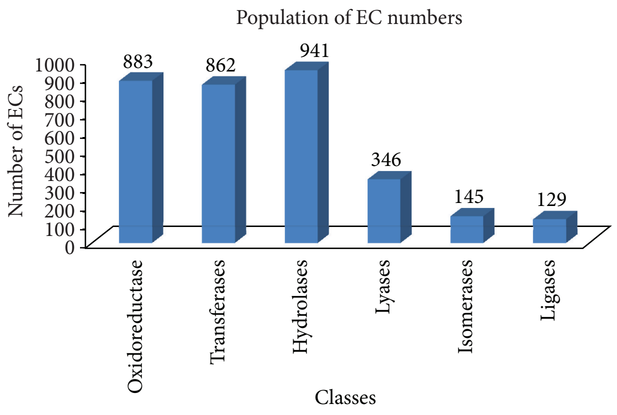


Figure 5: Population of EC numbers in the universe data set according to the six NC-IUBMB classes.

viewpoints, we present this intuitive method, which is based on the frequency of domain architecture and, in an enzyme, the associated catalyzed reactions.

To examine the feasibility of our method, we compiled data from the UniProtKB and SUPERFAMILY domains of the InterPro database. A total of 1,664,839 proteins are associated with 1,218 SUPERFAMILY domains and 3,306 4-numerical-block EC numbers. The population of the 6 NC-IUBMB classes is shown in Figure 5. If one type of domain architecture was only associated with one enzyme reaction, then we collect these entries as a single-EC set. Entries associated with more than one enzyme-catalyzed reaction were assigned to a multiple-EC set. There were single-EC entries and multiple-EC entries in both the training set and the testing set. The ratio of the number of single-EC entries to the number of the multiple-EC entries in the testing set was approximately 6 : 4. Detailed information is shown in the "Testing set" column in Table 3.

To avoid the bias caused by the selection of the training data set, we used 20 runs of 5-fold cross-validation. From 5,203 entries, approximately 4,160 entries were used for model building, and the remaining 1,040 entries were used for verification. According to the complexity of classification problems, it is difficult to predict multiple reactions of entries from domain architectures. We separated 1,040 entries into 2 sets: 624 entries for the single-EC set and 416 entries for the multiple-EC set; hence, there are 2 main rows in Table 3. If an entry's domain architecture could be determined by a model, it indicated that the entry could be predicted by the model and it would be counted in the "Match" column. The "Hit" column records the number of entries that were predicted correctly.

For comparison with our ERP method, we used the established Apriori algorithm [27] to mine for ARs implemented in a data mining package, Data-Mining-AssociationRules-0.10, of the Comprehensive Perl Archive Network (CPAN) [28]. The support and confidence threshold values used according to Chiu's settings [13] were 3 and 0.6, respectively. Table 3 shows that entries that were predicted using the AR method were considerably fewer than those predicted using the ERP method. To compare the 2 methods fairly, the same testing sets were used in the "AR" and "ERP1" rows, and entry sets that could be matched using the AR method were used as the testing set in row "ERP2." When more entries were predicted using the ERP method (the "ERP1" column), it resulted in a lower prediction rate than when using the AR method (the "AR" column) in Table 4. However, the ERP method is slightly more effective when considering entries that could be predicted using the AR method (the "ERP2" column).

The accuracy is provided by the ratio of the number of entries predicted correctly to the number of entry-matching rules of each method in Table 4. After 20 runs of 5-fold cross-validation, the mean accuracy values for 100 simulations were estimated. In a single-EC case, both the AR and ERP results reached 90%. However, estimation was less accurate for multiple-EC reactions. It is worth mentioning that both the ERP1 and ERP2 results are higher than the AR method in the multiple-EC set.

In the model-building phase, we implemented the AR method in a server equipped with 12 CPUs (4 cores, 3 packages) and 128 GB of memory, and the server used for the ERP method was equipped with 2 CPUs (2 cores, 1 package) and 8 GB of memory. The average model-building time was over 1 hour for the AR method and 15 minutes for the ERP method. The reasons may be that the AR method needed to produce frequent item sets and many redundant rules was generated. Furthermore, estimates of the prediction time for a batch of query domain architectures are shown in Figure 6. The vertical axis indicates the execution time in seconds, and the horizontal axis marks the number of entries in a batch query.

A substantial demand exists for enzymes for industrial and medical applications in the global market; thus, enzyme function annotation is receiving considerable attention because it offers reductions in the cost of chemical processes. In this study, we proposed the ERP tool for annotating enzyme reactions based on the query domain architecture (Figure 7). After providing the domain architecture of a protein, the tool is used to determine whether available enzyme reactions exist; if not, an absence message is displayed. If enzyme reactions are available, the ERP tool is used to locate one type of the same domain architecture such that the corresponding enzyme reactions could be obtained with confidence. If the same architecture is not found, the next most promising subset is chosen from the given domain architecture, and its corresponding enzyme reactions are

Table 3: The average number of entries for 100 simulations.

Data set	Method	Hit	Match	Testing set
Single EC	AR	25.36 ± 4.49	27.92 ± 4.43	624.60 ± 15.15
	ERP1	298.95 ± 13.12	592.53 ± 14.93	624.60 ± 15.15
	ERP2	25.82 ± 4.47	27.92 ± 4.43	27.92 ± 4.43
Multiple ECs	AR	3.76 ± 1.81	44.44 ± 5.02	416.00 ± 15.24
	ERP1	137.35 ± 11.72	378.61 ± 14.87	416.00 ± 15.24
	ERP2	18.47 ± 3.37	44.44 ± 5.02	44.44 ± 5.02

Table 4: Accuracy of the AR and ERP models.

	AR	ERP1	ERP2
Single EC	90.72% ± 5.71%	50.45% ± 1.85%	92.39% ± 5.22%
Multiple ECs	8.36% ± 3.80%	36.28% ± 2.74%	41.66% ± 6.62%

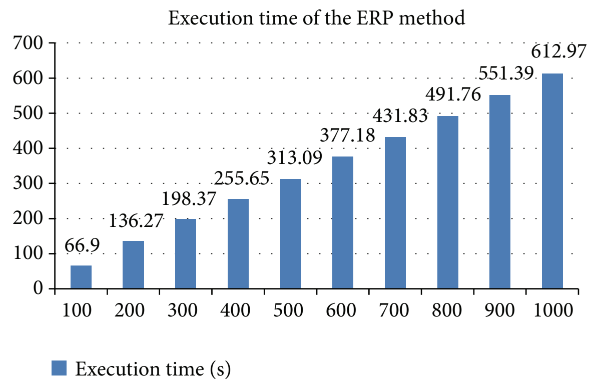


Figure 6: The average execution time of the ERP method for 100 simulations.

provided. If a similar domain architecture or a domain subset exists, proteins consisting of this architecture are displayed.

To implement the deduction of enzyme reactions from the domain architectures of enzymes, we designed a tool by using the Perl script language as follows. The set of domains in a protein must be listed before applying the ERP method. In the "Domain set" dialog, the domain set may be comma-or space-delimited. When the domain set is ready, pressing the "Predict" button starts processing according to the flowchart in Figure 7.

Two main situations in which analysis of the entered domain set could fail are described as follows.

(1) If the ERP tool cannot deduce the corresponding enzyme reactions from the ERP integrated universe set, a failure message, such as the domain architecture failure notice {SSF54211, SSF54236} shown in Figure 8 is displayed in the results dialog, indicating that enzyme reactions associated with the query domain architecture could not be deduced from the universe data set.

(2) In deducible cases, the existence of enzymes sharing the same architecture is considered. If the corresponding protein exists, succinctness and consistency values expressing the strength of the domain architecture are listed. If no enzyme sharing the same architecture is located, subsets of the domain architecture are evaluated, and the domain subset with the highest priority is selected.

In the event that the same architecture protein (Figure 9) is found, a confirmation message is displayed and the domain architecture (Figure 9, {SSF51110, SSF55486}) is identified. The succinctness value of 1 indicates that an enzyme with this type of domain architecture is capable of catalyzing the reaction denoted as 3.4.24.21 without any auxiliary domains. The consistency value of 0 indicates that a strong relationship between the domain architecture {SSF51110, SSF55486} and enzyme reaction 3.4.24.21 exists and that an association with other enzyme reactions does not exist. Because only one associated enzyme reaction exists, the strength measurement $\text{Intensity}_{3.4.24.21}$ is calculated as 1. The protein consisting of the architecture {SSF51110, SSF55486} is shown in Figure 9 as accession number F4KTN6 and UniProt ID F4KTN6_9SPHI.

In the absence of a protein consisting of the same architecture (Figure 10), the subsets of domain architecture {SSF54211, SSF54814} are enumerated as {SSF54211} and {SSF54814}. After evaluating the 4 measurements used for enumerating the domain architecture, the candidate with the highest priority {SSF54814} is obtained. Similarly, an enzyme with this architecture is capable of catalyzing the reaction 2.7.7.8 independently and with succinctness value of 1. The relationship between the domain set {SSF54814} and enzyme reaction 2.7.7.8 is strong according to the consistency value of 0. Only one reaction, 2.7.7.8, is related to {SSF54814}; thus, $\text{Intensity}_{2.7.7.8}$ is calculated as 1. The protein with the accession number D9PMT6 and UniProt ID D9PMT6_9ZZZZ consisting of this type of domain architecture {SSF54814} is listed in Figure 10.

4. Conclusion

In this study, we investigated the intimate relationship between domain architecture and enzyme-catalyzed reactions by applying various criteria to the compiled universe data set of domains and EC numbers. The advent of

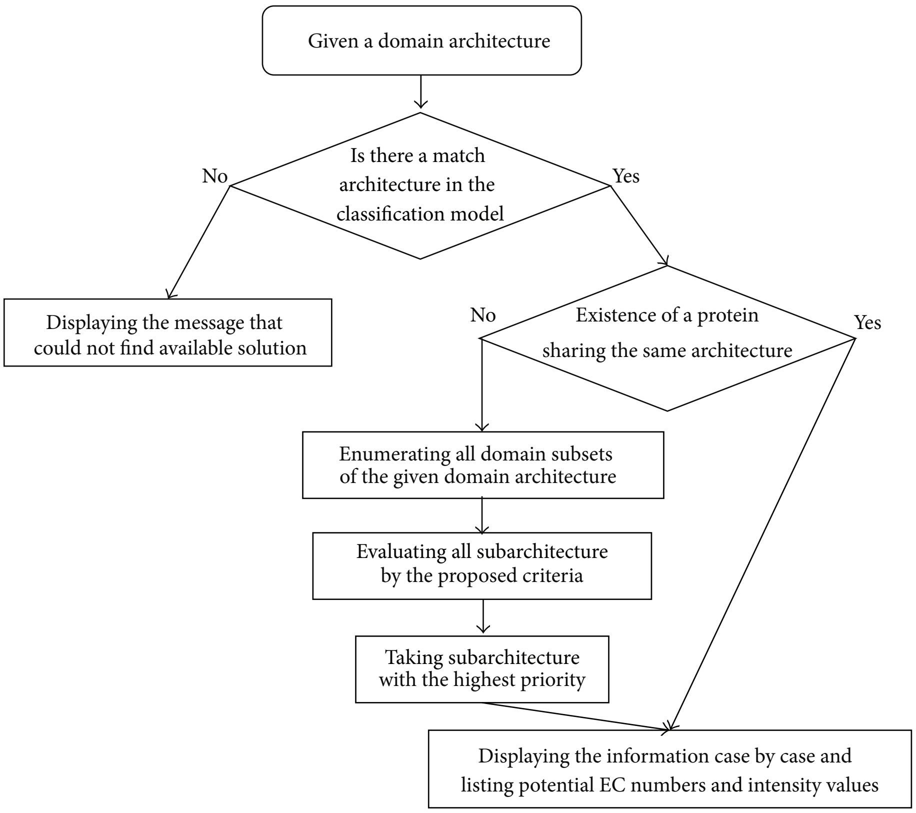
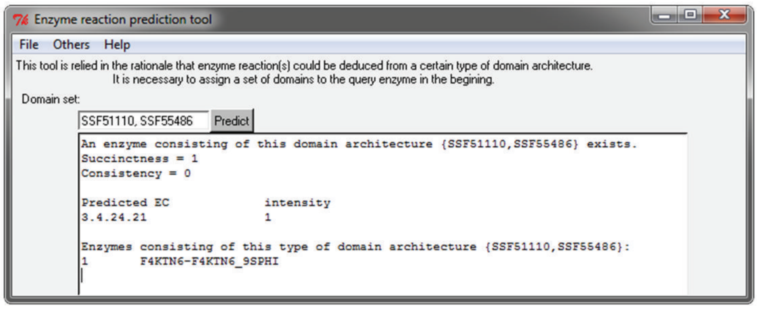

Given a domain architecture
Is there a match architecture in the classification model
No
Yes
Displaying the message that could not find available solution
Existence of a protein sharing the same architecture
No
Yes
Enumerating all domain subsets of the given domain architecture
Evaluating all subarchitecture by the proposed criteria
Taking subarchitecture with the highest priority
Displaying the information case by case and listing potential EC numbers and intensity values

FIGURE 7: Workflow of querying a domain architecture in the ERP model.



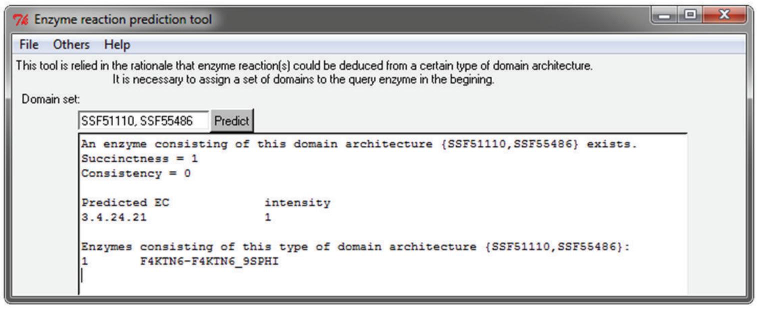
This tool is relied in the rationale that enzyme reaction(s) could be deduced from a certain type of domain architecture.
It is necessary to assign a set of domains to the query enzyme in the begining.
Domain set:
SSF51110, SSF55486

```
An enzyme consisting of this domain architecture {SSF51110,SSF55486} exists.
Succinctness = 1
Consistency = 0

Predicted EC              intensity
3.4.24.21                 1

Enzymes consisting of this type of domain architecture {SSF51110,SSF55486}:
1        F4KTN6-F4KTN6_9SPHI
```

FIGURE 8: Message of the failure case for the domain architecture {SSF54211, SSF54236}.



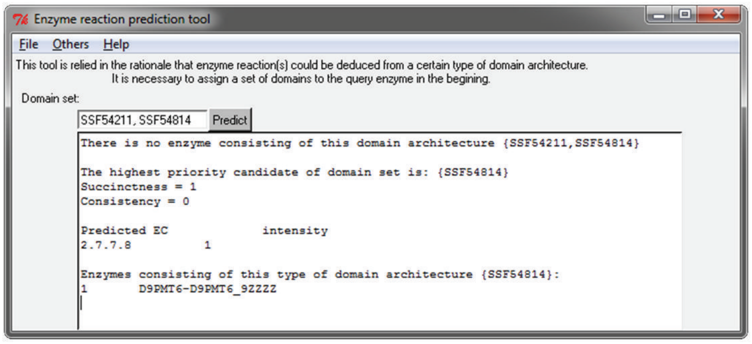
This tool is relied in the rationale that enzyme reaction(s) could be deduced from a certain type of domain architecture.
It is necessary to assign a set of domains to the query enzyme in the begining.
Domain set:
SSF51110, SSF55486

```
An enzyme consisting of this domain architecture {SSF51110,SSF55486} exists.
Succinctness = 1
Consistency = 0

Predicted EC              intensity
3.4.24.21                 1

Enzymes consisting of this type of domain architecture {SSF51110,SSF55486}:
1        F4KTN6-F4KTN6_9SPHI
```

FIGURE 9: The case of existence of the same architecture protein for the domain architecture {SSF51110, SSF55486}.



This tool is relied in the rationale that enzyme reaction(s) could be deduced from a certain type of domain architecture.
It is necessary to assign a set of domains to the query enzyme in the begining.

Domain set: SSF54211, SSF54814

```
There is no enzyme consisting of this domain architecture {SSF54211,SSF54814}

The highest priority candidate of domain set is: {SSF54814}
Succinctness = 1
Consistency = 0

Predicted EC            intensity
2.7.7.8         1

Enzymes consisting of this type of domain architecture {SSF54814}:
1       D9PMT6-D9PMT6_9ZZZZ
```

Figure 10: The report displayed in the case of an absence of any protein for the domain architecture {SSF54211, SSF54814}.

high-throughput techniques has produced numerous gene sequences, and annotating each enzyme reaction based on experimental results is difficult. However, we can consider domains as segments of sequences that fold into compact structural units; thus, we can model protein sequences and structures as these folded domains. We can identify and retrieve domains by integrating established sequence alignment tools with the proposed ERP tool.

Acknowledgments

This work was supported in part by the National Science Council of Taiwan (under Grants NSC100-2221-E-126-010-MY3 and NSC100-2221-E-182-057-MY3) and by Chang Gung Medical Hospital (under Grant CMRPD2A0012). The authors would like to thank their laboratory colleagues, Professor Chi-Neu Tsai, and anonymous reviewers for useful comments and suggestions.

References

[1] R. Garrett and C. M. Grisham, *Biochemistry*, Thomson Brooks/Cole, Belmont, California, USA, 3rd edition, 2005.

[2] "Enzymes in industrial applications: global markets," BBC Research BIO030F, 2011.

[3] International Union of Biochemistry and Molecular Biology, Nomenclature Committee, and E. C. Webb, *Enzyme Nomenclature 1992: Recommendations of the Nomenclature Committee of the International Union of Biochemistry and Molecular Biology on the Nomenclature and Classification of Enzymes*, International Union of Biochemistry and Molecular Biology by Academic Press, San Diego, Calif, USA, 1992.

[4] C. Claudel-Renard, C. Chevalet, T. Faraut, and D. Kahn, "Enzyme-specific profiles for genome annotation: PRIAM," *Nucleic Acids Research*, vol. 31, no. 22, pp. 6633–6639, 2003.

[5] C. Yu, N. Zavaljevski, V. Desai, and J. Reifman, "Genome-wide enzyme annotation with precision control: catalytic families (CatFam) databases," *Proteins*, vol. 74, no. 2, pp. 449–460, 2009.

[6] S. Quester and D. Schomburg, "EnzymeDetector: an integrated enzyme function prediction tool and database," *BMC Bioinformatics*, vol. 12, article 376, 2011.

[7] M. Scheer, A. Grote, A. Chang et al., "BRENDA, the enzyme information system in 2011," *Nucleic Acids Research*, vol. 39, no. 1, pp. D670–D676, 2011.

[8] W. Tian, A. K. Arakaki, and J. Skolnick, "EFICAz: a comprehensive approach for accurate genome-scale enzyme function inference," *Nucleic Acids Research*, vol. 32, no. 21, pp. 6226–6239, 2004.

[9] M. Kotera, Y. Okuno, M. Hattori, S. Goto, and M. Kanehisa, "Computational assignment of the EC numbers for genomic-scale analysis of enzymatic reactions," *Journal of the American Chemical Society*, vol. 126, no. 50, pp. 16487–16498, 2004.

[10] Q. Zhang and J. Aires-De-Sousa, "Structure-based classification of chemical reactions without assignment of reaction centers," *Journal of Chemical Information and Modeling*, vol. 45, no. 6, pp. 1775–1783, 2005.

[11] D. A. R. S. Latino, Q. Zhang, and J. Aires-de-Sousa, "Genome-scale classification of metabolic reactions and assignment of EC numbers with self-organizing maps," *Bioinformatics*, vol. 24, no. 19, pp. 2236–2244, 2008.

[12] Y. Yamanishi, M. Hattori, M. Kotera, S. Goto, and M. Kanehisa, "E-zyme: predicting potential EC numbers from the chemical transformation pattern of substrate-product pairs," *Bioinformatics*, vol. 25, no. 12, pp. i179–i186, 2009.

[13] S. Chiu, C. Chen, G. Yuan, and T. Lin, "Association algorithm to mine the rules that govern enzyme definition and to classify protein sequences," *BMC Bioinformatics*, vol. 7, article 304, 2006.

[14] E. Kretschmann, W. Fleischmann, and R. Apweiler, "Automatic rule generation for protein annotation with the C4.5 data mining algorithm applied on SWISS-PROT," *Bioinformatics*, vol. 17, no. 10, pp. 920–926, 2001.

[15] K. Chou and Y. Cai, "Using functional domain composition and support vector machines for prediction of protein subcellular location," *The Journal of Biological Chemistry*, vol. 277, no. 48, pp. 45765–45769, 2002.

[16] Y. Fujiwara and M. Asogawa, "Protein function prediction using hidden Markov models and neural networks," *NEC Research and Development*, vol. 43, no. 4, pp. 238–241, 2002.

[17] S. Pasek, A. Bergeron, J. Risler, A. Louis, E. Ollivier, and M. Raffinot, "Identification of genomic features using microsyntenies of domains: domain teams," *Genome Research*, vol. 15, no. 6, pp. 867–874, 2005.

[18] K. Forslund and E. L. L. Sonnhammer, "Predicting protein function from domain content," *Bioinformatics*, vol. 24, no. 15, pp. 1681–1687, 2008.

[19] T. Koestler, A. von Haeseler, and I. Ebersberger, "FACT: Functional annotation transfer between proteins with similar feature architectures," *BMC Bioinformatics*, vol. 11, article 417, 2010.

[20] D. R. Bandura, V. I. Baranov, O. I. Ornatsky et al., “Mass cytometry: technique for real time single cell multitarget immunoassay based on inductively coupled plasma time-of-flight mass spectrometry,” *Analytical Chemistry*, vol. 81, no. 16, pp. 6813–6822, 2009.

[21] E. E. Schadt, M. D. Linderman, J. Sorenson, L. Lee, and G. P. Nolan, “Computational solutions to large-scale data management and analysis,” *Nature Reviews Genetics*, vol. 11, no. 9, pp. 647–657, 2010.

[22] J. Ekanayake, T. Gunarathne, and J. Qiu, “Cloud technologies for bioinformatics applications,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 6, pp. 998–1011, 2011.

[23] L. Holm and C. Sander, “Parser for protein folding units,” *Proteins*, vol. 19, no. 3, pp. 256–268, 1994.

[24] M. Magrane and U. Consortium, “UniProt Knowledgebase: a hub of integrated protein data,” *Databass*, vol. 2011, article bar009, 2011.

[25] R. Apweiler, T. K. Attwood, A. Bairoch et al., “The InterPro database, an integrated documentation resource for protein families, domains and functional sites,” *Nucleic Acids Research*, vol. 29, no. 1, pp. 37–40, 2001.

[26] J. Gough, K. Karplus, R. Hughey, and C. Chothia, “Assignment of homology to genome sequences using a library of hidden Markov models that represent all proteins of known structure,” *Journal of Molecular Biology*, vol. 313, no. 4, pp. 903–919, 2001.

[27] R. Agrawal and R. Srikant, “Fast algorithms for mining association rules in large databases,” in *Proceedings of the 20th International Conference on Very Large Data Bases*, pp. 487–499, 1994.

[28] “The Association Rules Package of CPAN,” http://search.cpan.org/dist/Data-Mining-AssociationRules/lib/Data/Mining/AssociationRules.pm.

Hindawi Publishing Corporation
BioMed Research International
Volume 2013, Article ID 398968, 6 pages
http://dx.doi.org/10.1155/2013/398968

Research Article

Cloud Prediction of Protein Structure and Function with PredictProtein for Debian

László Kaján,[1] Guy Yachdav,[1,2,3] Esmeralda Vicedo,[1] Martin Steinegger,[1] Milot Mirdita,[1] Christof Angermüller,[1] Ariane Böhm,[1] Simon Domke,[1] Julia Ertl,[1] Christian Mertes,[1] Eva Reisinger,[1] Cedric Staniewski,[1] and Burkhard Rost[1,2,3,4,5]

[1] *TUM, Department of Informatics, Bioinformatics & Computational Biology-I12, Boltzmannstraß 3, 85748 Garching, Germany*
[2] *Columbia University, Department of Biochemistry and Molecular Biophysics and New York Consortium on Membrane Protein Structure (NYCOMPS), 701 West 168th Street, New York, NY 10032, USA*
[3] *Biosof LLC, 10th Floor, 138 West 25th Street, New York, NY 10001, USA*
[4] *WZW-Weihenstephan, Alte Akademie 8, Freising, Germany*
[5] *Institute for Advanced Study (TUM-IAS), Lichtenbergstraß 2a, 85748 Garching, Germany*

Correspondence should be addressed to László Kaján; lkajan@rostlab.org

Received 4 January 2013; Accepted 5 July 2013

Academic Editor: Ching-Hsien (Robert) Hsu

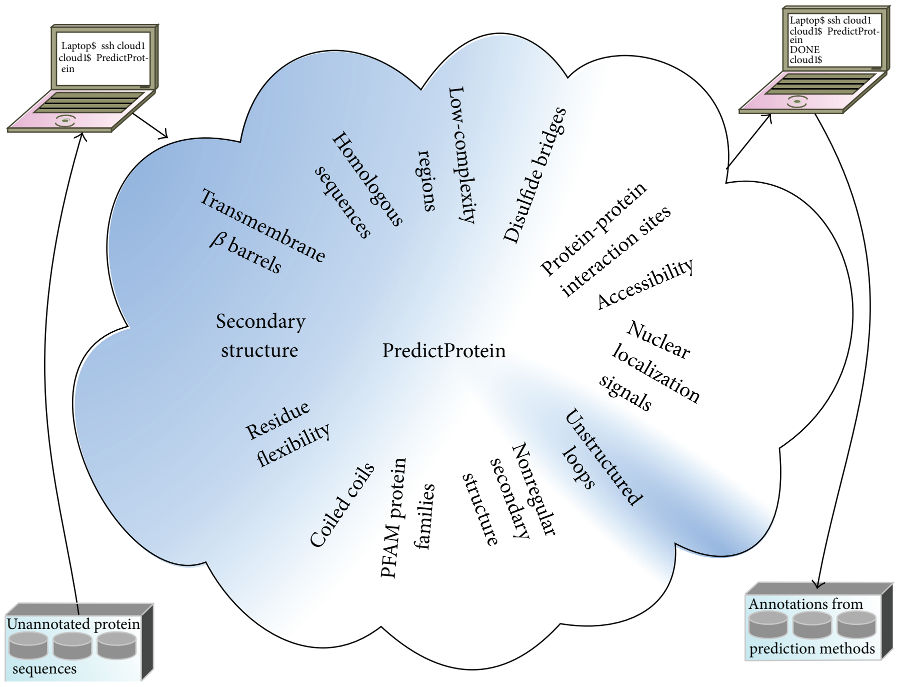
We report the release of PredictProtein for the Debian operating system and derivatives, such as Ubuntu, Bio-Linux, and Cloud BioLinux. The PredictProtein suite is available as a standard set of open source Debian packages. The release covers the most popular prediction methods from the Rost Lab, including methods for the prediction of secondary structure and solvent accessibility (profphd), nuclear localization signals (predictnls), and intrinsically disordered regions (norsnet). We also present two case studies that successfully utilize PredictProtein packages for high performance computing in the cloud: the first analyzes protein disorder for whole organisms, and the second analyzes the effect of all possible single sequence variants in protein coding regions of the human genome.

1. Background

Bioinformatics is embracing cloud computing. Recent months have seen the publication of cloud sequence analysis platforms, CloVR [1] and Galaxy Cloud [2], and the cloud version of Bio-Linux [3], Cloud BioLinux [4]. Cost analysis depicts cloud computing as an attractive and sustainable solution for computational biology and bioinformatics [5–8]. The rate of data generation of "next generation" sequencing (NGS) drives the efforts to turn to cloud computing as a solution to handling peak-time loads, without the need to maintain large clusters [9]. Cloud-enabled bioinformatics tools are now available in the context of high throughput sequencing and genomics [10].

The Rost Lab provides protein structure and function prediction tools for cloud computing in the PredictProtein suite [11]. PredictProtein began as an Internet server for sequence analysis and the prediction of aspects of protein structure and function in 1992 [12]. Queried with a protein sequence, PredictProtein returns secondary structure and accessibility predictions, predictions of unstructured loops, nuclear localization signals, protein-protein interaction sites, disulfide bonds, regions lacking regular secondary structure, protein family hits, low-complexity regions, bacterial transmembrane beta barrels, coiled-coil regions, protein residue flexibility, and homologous sequences (Figure 1).

Cloud computing is commonly realized on machine instances that run on virtual hardware providing "infrastructure as a service" (IaaS) [13, 14]. This type of cloud computing instantiates compute nodes from machine images. Machine images usually contain an operating system with software tools. For example, one could request the instantiation of 10 worker nodes of PredictProtein on Debian operating system at the Amazon EC2 IaaS offering.



```
Laptop$  ssh cloud1
cloud1$  PredictProt-
ein
```

```
Laptop$ ssh cloud1
cloud1$  PredictProt-
ein
DONE
cloud1$
```

Figure 1: Protein annotation by PredictProtein. PredictProtein annotates input sequences with the features shown.

The PredictProtein cloud solution builds upon the open source operating system Debian [15] and provides its functionality as a set of free [16] software packages. Bio-Linux is an operating system for bioinformatics and computational biology. The latest Bio-Linux release 7 provides more than 500 bioinformatics programs on an Ubuntu Linux base [17]. Ubuntu is a "derivative" operating system [18] based on Debian, with its own additions. Cloud BioLinux is a comprehensive cloud solution that is derived from Bio-Linux and Ubuntu. Debian derivatives can easily share packages between each other. For example, Debian packages are automatically incorporated in Ubuntu [19] and are also usable in Cloud BioLinux (the procedure is described in [4]).

2. Implementation

The PredictProtein suite is implemented as a set of free packages released at http://debian.org/. Software packaging conformed with the Policy Manual [20], and following the recommendations of the Developer's Reference [21].

3. Results and Discussion

High-throughput experiments generate vast amounts of data at an ever-increasing rate; the pace of creating reliable annotations needed to use that data increases much slower. One of the major challenges for computational tools is to narrow the resulting increase in the protein annotation gap [22]. Of the over 35 m (million) sequences in the UniProt Knowledgebase 2013_05 [23], only about 500 k (500 thousand) have explicit experimental annotations in Swiss-Prot [24]. Computational prediction methods, such as those included in PredictProtein, can annotate important features for the remainder and enable us to draw scientific insights. Unfortunately, the task is often intractable for any single desktop computer within reasonable time. Fortunately, cloud computing is now at hand. On-demand servers in the cloud promise to fit computing power to most tasks economically, and without a fair portion of the usual worries of system management: hardware purchasing, recruiting a system manager, high availability issues, and so forth ([13] and the references therein). One problem remains: how to get the often adhoc analysis toolset from the desktop environment into the cloud? Directly addressing this problem, here we report the first Debian package release of the protein feature prediction toolset "PredictProtein," developed at the Rost Lab.

The publication of scientific results has, overall, changed surprisingly little since the Internet exists [25]. Research code is regularly distributed as a "zip" file of the development directory. Often, the only "documentation" distributed along with the code is the published paper accompanied by some "README" file. Software distributed this way often fails outside the laboratory without expert attention. In order to address this issue in the PredictProtein suite, we decided to apply the community and time-tested packaging and release requirements of Debian to PredictProtein components. We have traced all dependencies, eliminated convenience copies, carefully documented each of our prediction methods, and made them go through the thorough review process every Debian package receives. This converted PredictProtein from an adhoc implementation to a reusable software component (Figure 2).

Our packages facilitate the generation of purpose-built machine images for cloud computing. As an example, we distribute a slim PredictProtein machine image (PPMI) through

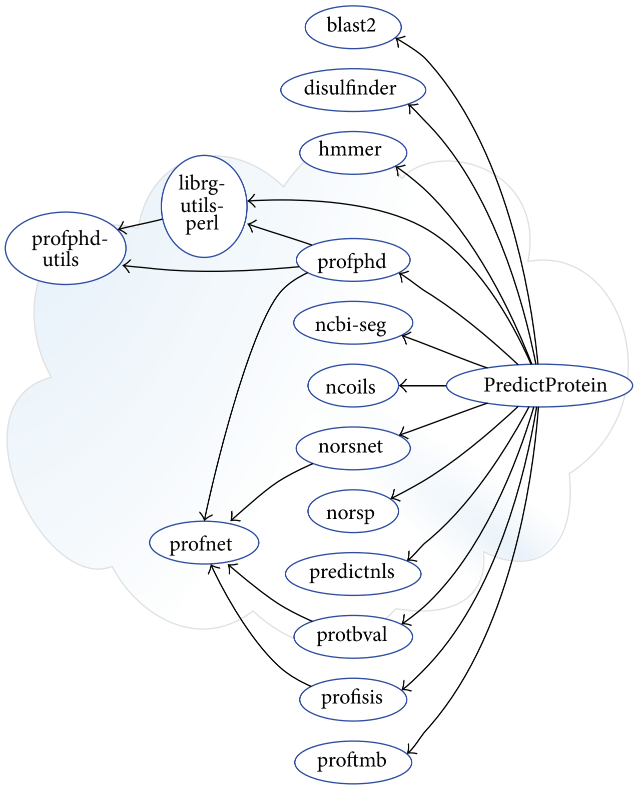


Figure 2: Package dependencies for PredictProtein. Arrows represent "depends on" relationships. Only significant dependencies are shown for clarity. Convenience copies of "profnet" for "profphd," "norsnet," "profbval," and "profisis" have been merged to a single "profnet" package. Similar merging was done for all code convenience copies.

the PredictProtein website [26]. This image contains a minimal installation of Debian with the command line version of PredictProtein. Databases are provided as a separate disk image. The PPMI is bootable on server instances in cloud infrastructure services, or on locally installed virtualization software. The latter allows for a cross-platform solution to use PredictProtein. Apart from virtualization, "chroot" environments present an option to run the software on Linux distributions where Debian packages are not readily usable. After booting the machine image, a friendly message at the login prompt offers usage tips and directions to documentation. A "Getting Started with PredictProtein" guide is available online [27]. The PPMI and the data image are updated regularly and are freely available at http://predictprotein.org/. For a comprehensive bioinformatics and computational biology computing environment, we recommend using PredictProtein with Bio-Linux [3] or Cloud BioLinux [4], where PredictProtein is either preinstalled or is easily installable from package repositories. We plan to release the web-based graphical interface of PredictProtein for these platforms in the near future.

The PredictProtein suite has attracted respectable popularity both online and offline. PredictProtein has been operating continuously since 1992, that is, the dawn of the Internet. Today, over 100,000 online users are registered; over 500 users access the PredictProtein web page every day and 12,000 unique users apply the service every month. Our Media Wiki page presenting an overview of the Rost Lab software packages has been accessed nearly 60,000 times since its launch 36 months ago. Adoption of the PredictProtein packages by the community has also been remarkable. Over 200 packages of the PredictProtein suite are installed from the Debian repository alone, while these and other installations have performed over 57 million protein feature predictions over the past year, not counting our own usage. Out of this, ~30 million were secondary structure and accessibility predictions from the "profphd" method [28].

4. PredictProtein Packages

The following protein feature prediction methods—components of PredictProtein—are available (feature—"package name"): secondary structure, accessibility, and transmembrane helices—"profphd" [29–31]; unstructured loops—"norsnet" [32]; nuclear localization signals—"predictnls" [33]; protein-protein interaction sites—"profisis" [34]; disulfide bridges—"disulfinder" [35]; nonregular secondary structure—"norsp" [36]; PFAM hits—"hmmer" [37, 38]; local complexity—"ncbi-seg" [39]; bacterial transmembrane beta barrels—"proftmb" [40]; coiled-coils—"ncoils" [41]; protein residue flexibility—"profbval" [42]; sequence homologies—"blast2" [43]; protein feature prediction suite—"predictprotein" [11].

These tools are available under a free license through Debian and are automatically incorporated into other Linux distributions such as Ubuntu. An overview of the packages offered for bioinformatics and cloud computing, complete with literature references, is available at Debian Med [44]. PredictProtein is listed in the Biology task.

5. Case Study 1: Protein Disorder in Completely Sequenced Organisms

The goal of this study is to collect evidence for three hypotheses on protein disorder: (1) it is more useful to picture disorder as a distinct phenomenon than as an extreme example of protein flexibility; (2) there are many very different flavors of protein disorder, but it is advantageous to recognize just two main types, namely, *well structured* and *disordered*; (3) nature uses protein disorder as a tool to adapt to different environments [45]. We predicted protein disorder both on an in-house compute grid and on a compute grid manually setup in the OpenNebula [46] cloud service provided by the CSC Finland [47]. Data and tool (the PPMI) images for grid nodes in the cloud were downloaded from http://predictprotein.org/. The PPMI image was extended with a grid client, and a separate machine instance was used as grid master. PredictProtein for the local grid was installed from the main Debian repository. Required databases (28 GB) were included on a data disk image for cloud machine instances. Input to PredictProtein jobs consisted of protein sequences (in total less than 1 GB). Grid job submissions to the local and the cloud grid were manually adjusted according to available resources. Over 9 million disorder predictions were made over the course of the past few years.

6. Case Study 2: Comprehensive In Silico Mutagenesis of Human Proteome

This project aims at providing information about the functional effect of every possible point mutation in all human proteins, that is, for the replacement of $19 * N$ amino acids for a protein with N residues. Overall, this generated 300 million human sequence variants (point mutants). The method SNAP [48] predicted the effect of each variant, that is, each "nonsynonymous single nucleotide polymorphisms" (nsSNPs) upon protein function. These predictions are useful for several reasons. First, the study of all possible mutations in human will provide the background against which we can assess the effect of mutations that are actually observed between people. This is crucial for both the advance toward personalized medicine and health and the understanding of human diversity and variation. Second, our computation provides quick "look-up" answers available for all the important variants that are observed and implied in important phenotypes. The only way to cover those lookups is by precomputing all the possible changes. SNAP can take advantage of PredictProtein results for faster processing. With the PredictProtein packages presented here, a solution was built in the form of a public Amazon Machine Image (AMI, ami-3f5f8156) that allows running PredictProtein on the Amazon Elastic Compute Cloud (EC2). We extended an Ubuntu-based StarCluster [49] AMI with PredictProtein and its required databases (28 GB). Because every protein can be computed independently, we formed a grid job out of each protein and used the Grid Engine (GE) to distribute work on the machine instances. We used StarCluster to automate grid setup on the EC2. Because a lot of CPU power was needed, the "Cluster Compute Eight Extra Large Instance" was chosen. This instance type is especially crafted for big data with a lot of CPU power. One instance has 60.5 GB memory, 88 EC2 Compute Units (2x Intel Xeon E5-2670, eight-core-architecture "Sandy Bridge"), and 3370 GB instance storage. The sequence variants were analyzed based on the human reference proteome from the National Center for Biotechnology Information (build 37.3, proteins, 21MB). We processed 29,036 sequences with 16,618,608 residues. This amounted to predicting the functional effect of 315,753,552 individual amino acid changes.

7. Conclusion

The open source release of the PredictProtein protein structure and function prediction suite from the Rost Lab is now available for Debian and derivative operating systems, such as Ubuntu, Bio-Linux, and Cloud BioLinux. The software, due to its standard packaging, is readily deployable in the cloud. Successfully addressing the challenges of cloud computing brings PredictProtein—developed over almost two decades—into the present and the future. In accordance with the Rost Lab open policy [50], and supported by anonymous statistics, PredictProtein is now shared with a wide range of users. We encourage the bioinformatics community to take advantage of our open source software, itself a result of the collaboration of the wider open source software community.

Conflict of Interests

The authors declare that they have no conflict of interests.

Authors' Contributions

L. Kaján and G. Yachdav (equal contributors) have redesigned "predictprotein," performed initial software packaging, and wrote the paper; E. Vicedo, M. Steinegger and M. Mirdita performed case studies, and reviewed the paper; C. Angermüller, A. Böhm, S. Domke, J. Ertl, C. Mertes, E. Reisinger, and C. Staniewski finalized the packaging for Debian; B. Rost provided initial implementation of the "predictprotein" core module and reviewed the paper.

Acknowledgments

This work was supported by the Alexander von Humboldt Foundation, the IAS at the TU in Munich and the National Institutes of Health (R01-LM07329 to G. Yachdav). M. Steinegger and M. Mirdita received funding from Amazon and Cycle Computing for case study 2. The authors thank all those who funded our research. The authors thank researchers who deposited data into publicly available datasets and programmers who provided their work under a free license: our work stands upon their shoulders and would not have been possible without them. The authors wish to thank the Debian project in general and Steffen Möller and Andreas Tille in particular for their tireless support.

References

[1] S. V. Angiuoli, M. Matalka, A. Gussman et al., "CloVR: a virtual machine for automated and portable sequence analysis from the desktop using cloud computing," *BMC Bioinformatics*, vol. 12, p. 356, 2011.

[2] E. Afgan, D. Baker, N. Coraor et al., "Harnessing cloud computing with Galaxy Cloud," *Nature Biotechnology*, vol. 29, no. 11, pp. 972–974, 2011.

[3] D. Field, B. Tiwari, T. Booth et al., "Open software for biologists: from famine to feast," *Nature Biotechnology*, vol. 24, no. 7, pp. 801–803, 2006.

[4] K. Krampis, T. Booth, B. Chapman et al., "Cloud BioLinux: pre-configured and on-demand bioinformatics computing for the genomics community," *BMC Bioinformatics*, vol. 13, p. 42, 2012.

[5] J. T. Dudley, Y. Pouliot, R. Chen, A. A. Morgan, and A. J. Butte, "Translational bioinformatics in the cloud: an affordable alternative," *Genome Medicine*, vol. 2, no. 8, p. 51, 2010.

[6] S. V. Angiuoli, J. R. White, M. Matalka, O. White, and W. F. Fricke, "Resources and costs for microbial sequence analysis evaluated using virtual machines and cloud computing," *PLoS ONE*, vol. 6, no. 10, Article ID e26624, 2011.

[7] P. Kudtarkar, T. F. DeLuca, V. A. Fusaro, P. J. Tonellato, and D. P. Wall, "Cost-effective cloud computing: a case study using the comparative genomics tool, roundup," *Evolutionary Bioinformatics*, vol. 2010, no. 6, pp. 197–203, 2010.

[8] M. Steinegger, *HPC Full in Silico Mutagenesis, in Department of Bioinformatics*, Technical University of Munich, Munich, Germany, 2012.

[9] L. D. Stein, "The case for cloud computing in genome informatics," *Genome Biology*, vol. 11, no. 5, p. 207, 2010.

[10] E. Afgan, D. Baker, N. Coraor, B. Chapman, A. Nekrutenko, and J. Taylor, "Galaxy CloudMan: delivering cloud compute clusters," *BMC Bioinformatics*, vol. 11, supplement 12, p. S4, 2010.

[11] B. Rost and J. Liu, "The PredictProtein server," *Nucleic Acids Research*, vol. 31, no. 13, pp. 3300–3304, 2003.

[12] B. Rost, G. Yachdav, and J. Liu, "The PredictProtein server," *Nucleic Acids Research*, vol. 32, pp. W321–W326, 2004.

[13] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, "A break in the clouds: towards a cloud definition," *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 1, pp. 50–55, 2008.

[14] V. A. Fusaro, P. Patil, E. Gafni, D. P. Wall, and P. J. Tonellato, "Biomedical cloud computing with amazon web services," *PLoS Computational Biology*, vol. 7, no. 8, Article ID e1002147, 2011.

[15] J. J. Amor, G. Robles, J. M. González-Barahona, and I. Herraiz, "From pigs to stripes: a travel through debian," in *Proceedings of the Debian Annual Developers Meeting (DebConf '05)*, Citeseer, 2005.

[16] "The Debian Free Software Guidelines (DFSG)," http://www.debian.org/social_contract#guidelines.

[17] B. T. Dawn Field, T. Booth, S. Houten, D. Swan, N. Bertrand, and M. Thurston, "Bio-Linux 7," http://nebc.nerc.ac.uk/tools/bio-linux/bio-linux-7-info, 2012.

[18] "Debian Derivatives," http://wiki.debian.org/Derivatives.

[19] "NEW packages through Debian," https://wiki.ubuntu.com/UbuntuDevelopment/NewPackages#NEW_packages_through_Debian.

[20] "Debian Policy Manual," http://www.debian.org/doc/debian-policy/, 2012.

[21] "Debian Developer's Reference," http://www.debian.org/doc/manuals/developers-reference/, 2012.

[22] Y. Bromberg, G. Yachdav, Y. Ofran, R. Schneider, and B. Rost, "New in protein structure and function annotation: hotspots, single nucleotide polymorphisms and the "Deep Web"," *Current Opinion in Drug Discovery and Development*, vol. 12, no. 3, pp. 408–419, 2009.

[23] M. Magrane and U. Consortium, "UniProt Knowledgebase: a hub of integrated protein data," *Database*, vol. 2011, p. bar009, 2011.

[24] A. Bairoch, B. Boeckmann, S. Ferro, and E. Gasteiger, "Swiss-Prot: juggling between evolution and stability," *Briefings in Bioinformatics*, vol. 5, no. 1, pp. 39–55, 2004.

[25] R. Gentleman, "Reproducible research: a bioinformatics case study," *Statistical Applications in Genetics and Molecular Biology*, vol. 4, no. 1, p. 1034, 2005.

[26] "PredictProtein Website," http://predictprotein.org/.

[27] L. Kajan, "Getting Started with PredictProtein," http://wiki.debian.org/DebianMed/PredictProtein, 2013.

[28] B. Rost and C. Sander, "Improved prediction of protein secondary structure by use of sequence profiles and neural networks," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 90, no. 16, pp. 7558–7562, 1993.

[29] B. Rost and C. Sander, "Combining evolutionary information and neural networks to predict protein secondary structure," *Proteins*, vol. 19, no. 1, pp. 55–72, 1994.

[30] B. Rost and C. Sander, "Conservation and prediction of solvent accessibility in protein families," *Proteins*, vol. 20, no. 3, pp. 216–226, 1994.

[31] B. Rost, R. Casadio, P. Fariselli, and C. Sander, "Transmembrane helices predicted at 95% accuracy," *Protein Science*, vol. 4, no. 3, pp. 521–533, 1995.

[32] A. Schlessinger, J. Liu, and B. Rost, "Natively unstructured loops differ from other loops," *PLoS Computational Biology*, vol. 3, no. 7, p. e140, 2007.

[33] M. Cokol, R. Nair, and B. Rost, "Finding nuclear localization signals," *EMBO Reports*, vol. 1, no. 5, pp. 411–415, 2000.

[34] Y. Ofran and B. Rost, "ISIS: interaction sites identified from sequence," *Bioinformatics*, vol. 23, no. 2, pp. e13–e16, 2007.

[35] A. Ceroni, A. Passerini, A. Vullo, and P. Frasconi, "Disulfind: a disulfide bonding state and cysteine connectivity prediction server," *Nucleic Acids Research*, vol. 34, pp. W177–W181, 2006.

[36] J. Liu and B. Rost, "NORSp: predictions of long regions without regular secondary structure," *Nucleic Acids Research*, vol. 31, no. 13, pp. 3833–3835, 2003.

[37] R. D. Finn, J. Mistry, J. Tate et al., "The Pfam protein families database," *Nucleic Acids Research*, vol. 38, supplement 1, pp. D211–D222, 2010.

[38] S. R. Eddy, "Accelerated profile HMM searches," *PLoS Computational Biology*, vol. 7, no. 10, Article ID e1002195, 2011.

[39] J. C. Wootton and S. Federhen, "Statistics of local complexity in amino acid sequences and sequence databases," *Computers and Chemistry*, vol. 17, no. 2, pp. 149–163, 1993.

[40] H. Bigelow and B. Rost, "PROFtmb: a web server for predicting bacterial transmembrane beta barrel proteins," *Nucleic Acids Research*, vol. 34, pp. W186–W188, 2006.

[41] A. Lupas, "[30] Prediction and analysis of coiled-coil structures," *Methods in Enzymology*, vol. 266, pp. 513–524, 1996.

[42] A. Schlessinger, G. Yachdav, and B. Rost, "PROFbval: predict flexible and rigid residues in proteins," *Bioinformatics*, vol. 22, no. 7, pp. 891–893, 2006.

[43] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool," *Journal of Molecular Biology*, vol. 215, no. 3, pp. 403–410, 1990.

[44] S. Möller, H. N. Krabbenhöft, A. Tille et al., "Community-driven computational biology with Debian Linux," *BMC Bioinformatics*, vol. 11, supplement 12, p. S5, 2010.

[45] A. Schlessinger, C. Schaefer, E. Vicedo, M. Schmidberger, M. Punta, and B. Rost, "Protein disorder–a breakthrough invention of evolution?" *Current Opinion in Structural Biology*, vol. 21, no. 3, pp. 412–418, 2011.

[46] R. Moreno-Vozmediano, R. Montero, and I. Llorente, "IaaS cloud architecture: from virtualized data centers to federated cloud infrastructures," *IEEE Computer Society*, vol. 45, no. 12, pp. 65–72, 2012.

[47] T. H. Nyrönen, M. A. Babar, C. E. Cuesta, and J. E. Savolainen, "Delivering ICT infrastructure for biomedical research," in *Proceedings of the WICSA/ECSA 2012 Companion Volume*, pp. 37–44, ACM, Helsinki, Finland, 2012.

[48] Y. Bromberg and B. Rost, "SNAP: predict effect of non-synonymous polymorphisms on function," *Nucleic Acids Research*, vol. 35, no. 11, pp. 3823–3835, 2007.

[49] "StarCluster," http://star.mit.edu/cluster/index.html.

[50] R. H. Lathrop and B. Rost, "ISCB public policy statement on open access to scientific and technical research literature," *Bioinformatics*, vol. 27, no. 3, pp. 291–294, 2011.

Hindawi Publishing Corporation
BioMed Research International
Volume 2013, Article ID 170356, 7 pages
http://dx.doi.org/10.1155/2013/170356

Research Article

Cloud Computing for Protein-Ligand Binding Site Comparison

Che-Lun Hung[1] **and Guan-Jie Hua**[2]

[1] *Department of Computer Science and Communication Engineering, Providence University, Taiwan Boulevard, Shalu District, Taichung 43301, Taiwan*
[2] *Department of Computer Science and Information Engineering, Providence University, Taiwan Boulevard, Shalu District, Taichung 43301, Taiwan*

Correspondence should be addressed to Che-Lun Hung; clhung@pu.edu.tw

Received 18 December 2012; Accepted 28 March 2013

Academic Editor: Ching-Hsien Hsu

The proteome-wide analysis of protein-ligand binding sites and their interactions with ligands is important in structure-based drug design and in understanding ligand cross reactivity and toxicity. The well-known and commonly used software, SMAP, has been designed for 3D ligand binding site comparison and similarity searching of a structural proteome. SMAP can also predict drug side effects and reassign existing drugs to new indications. However, the computing scale of SMAP is limited. We have developed a high availability, high performance system that expands the comparison scale of SMAP. This cloud computing service, called Cloud-PLBS, combines the SMAP and Hadoop frameworks and is deployed on a virtual cloud computing platform. To handle the vast amount of experimental data on protein-ligand binding site pairs, Cloud-PLBS exploits the MapReduce paradigm as a management and parallelizing tool. Cloud-PLBS provides a web portal and scalability through which biologists can address a wide range of computer-intensive questions in biology and drug discovery.

1. Introduction

By virtue of its 3D structure, a protein performs thousands of life-critical functions at the molecular level. Detection and characterization of protein structural ligand binding sites and their interactions with binding partners are pivotal to a wide range of structure-function correlation problems—predicting functions for structural genomics targets, identifying and validating drug targets, prioritizing and optimizing drug leads, and correlating molecular functions to physiological processes in drug design [1].

Xie et al. [2–4] proposed an efficient and robust algorithm called SMAP, which quantitatively characterizes the geometric properties of proteins. Ligand binding sites predicted by SMAP have been experimentally validated [4–7]. SMAP has also been applied to drug design problems, such as constructing drug-target interaction networks [4], designing polypharmacology drugs [5], assigning old drugs to new indications [6], and predicting the side effects of drugs [8, 9]. The web service tool SMAP-WS [1] implements SMAP via Opal [10]. Although the parallel implementation of SMAP improves the speed of database searching, it cannot operate at the scale and availability demanded by current Internet technology.

Recently, an Internet service concept known as cloud computing has become popular for providing various services to users. The cloud computing environment is a distributed system with extremely scalable IT-related capabilities, providing multiple external customers with numerous services. Cloud computing also enables the copying of vast datasets to many users with high fault tolerance. Another popular open-source software framework designed for data-intensive distribution is Hadoop [11]. This framework processes petabytes of data intercepting thousands of nodes. Hadoop provides the MapReduce programming model, by which parallel computing of large data sets can be implemented in the cloud computing environment. MapReduce enables distributed computing of the mappers and reducers. Each mapper performs an independent map operation which is parallelized with the tasks of other mappers. Similarly, a set of reducers can perform a set of reduce operations. All

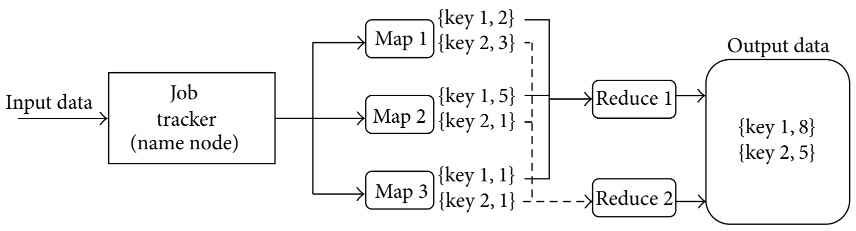


Figure 1: Map/reduce framework of Hadoop.

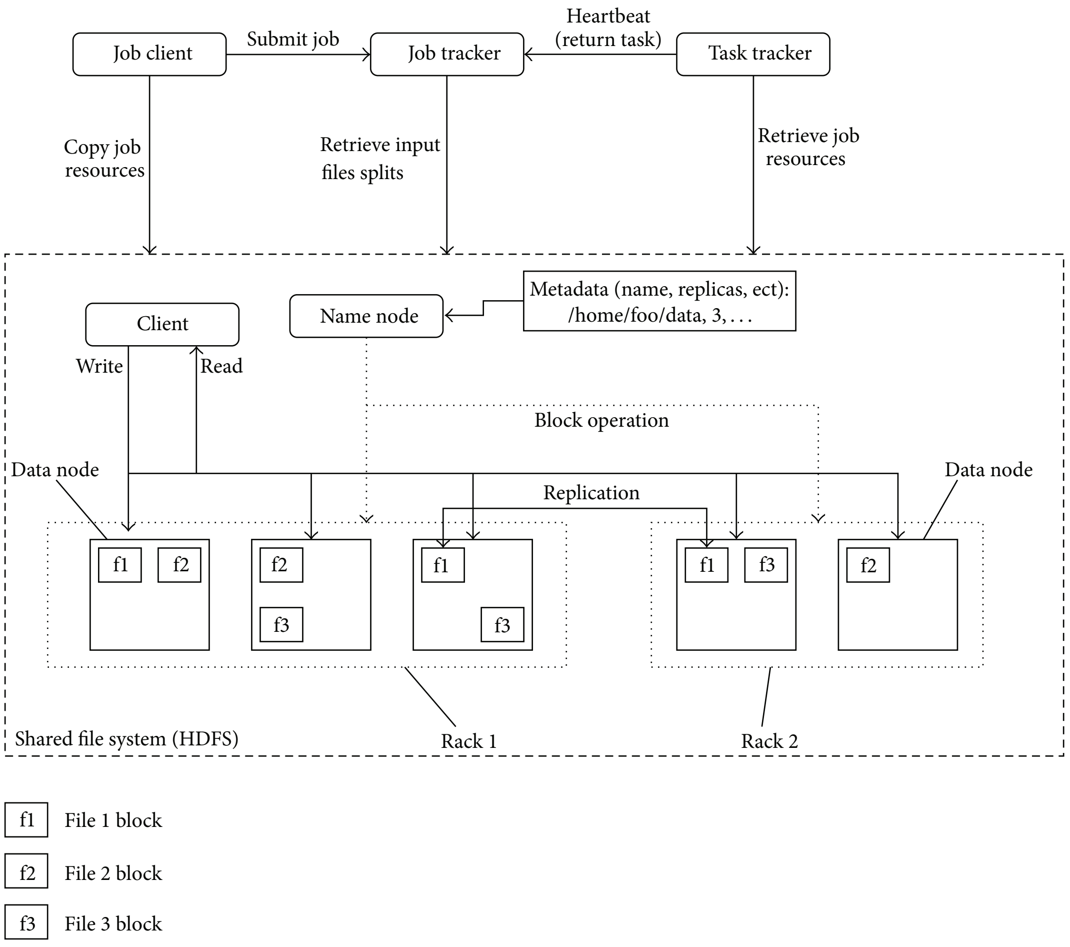


Figure 2: The architecture of Hadoop cluster.

outputs of the map operations possessing the same key are presented to the same reducer at the same time. Two additional important benefits of Hadoop are scalability and fault tolerance. Hadoop can guide jobs toward successful completion even when individual nodes or network components experience high failure rates. Meanwhile, a machine can be readily attached as a mapper and reducer in the Hadoop cluster. The Hadoop platform, therefore, is regarded as a superior solution to real-world data distribution problems. To date, Hadoop has been applied in a range of bioinformatics domains [12–16].

Cloud computing platforms are usually based on virtualization technology. Computing resources are combined or divided into one or more operating environments using methodologies such as hardware and software partitioning or aggregation, partial or complete machine simulation, and emulation and time sharing. A virtual machine (VM) is a machine simulation created by virtualization technology, which resides in a physical machine and shares its physical resources. The web service Amazon Elastic Compute Cloud (Amazon EC2) [17] uses virtualization technology to generate resizable computing capacity in the cloud. The service provides a true virtual computing environment, allowing users to launch VMs with a variety of operating systems. Users can construct their own elastic cluster systems by attaching or removing VMs.

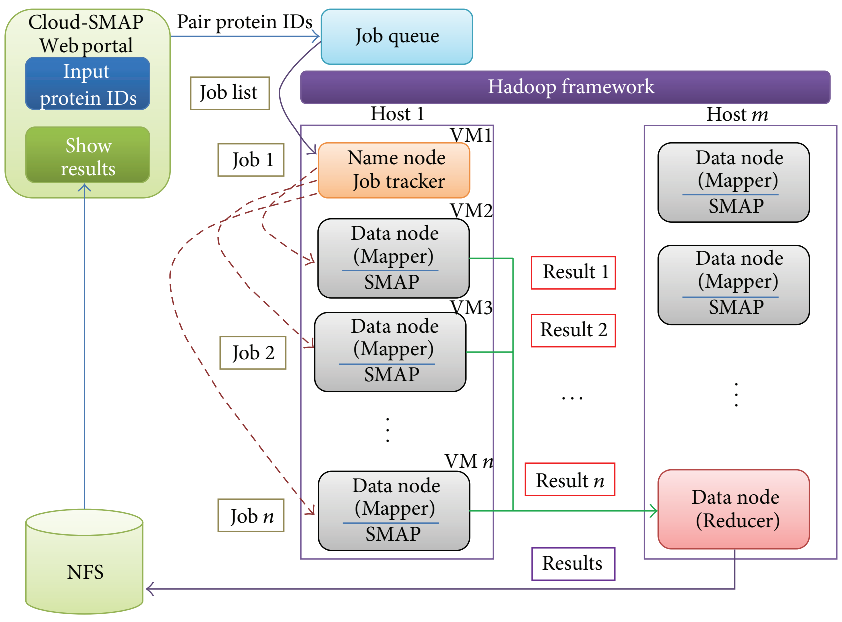


FIGURE 3: The cloud platform of Cloud-PLBS.

In this paper, we combine three technologies, Hadoop framework, virtualization, and SMAP, to develop a cloud computing service for structural ligand binding site comparison. Each mapper or reducer in the cloud platform is a VM. The platform uses MapReduce to simultaneously process numerous comparison jobs. Similarly, the number of VMs can be adjusted to the size of the comparison job (large and small jobs demand more and fewer VMs, resp.). Hadoop enables our cloud platform to recover the comparison job from a crashed VM or physical machine by reassigning the job to a healthy VM or a physical machine. The cloud platform can achieve high performance, scalability, and availability. The experimental results demonstrate that applying the Hadoop framework on a virtualization platform enhances the computational efficiency of the proposed service. The cloud service is available at http://bioinfo.cs.pu.edu.tw/cloud-PLBS/index.html.

2. Method

Cloud-PLBS is a robust, elastic cloud computing service for protein-ligand binding site comparison. It guarantees rapid return of comparison results. Cloud-PLBS embraces three technologies, virtualization, Hadoop, and SMAP, used to build the cloud computing infrastructure, perform parallel computation, and compare ligand binding sites, respectively.

2.1. Structural Proteome-Wide Ligand Binding Site Comparison. SMAP is an efficient and robust algorithm that performs pair-wise comparison of two potential ligand binding sites. The user enters two protein structure IDs, and SMAP downloads the relevant protein structures from the RCSB Protein Data Bank (PDB) [18]. Protein structure binding sites are compared in four stages.

Step 1. The protein structures are represented by C-α atoms for structural variation tolerance.

Step 2. Amino acid residues are characterized by surface orientation and a geometric potential.

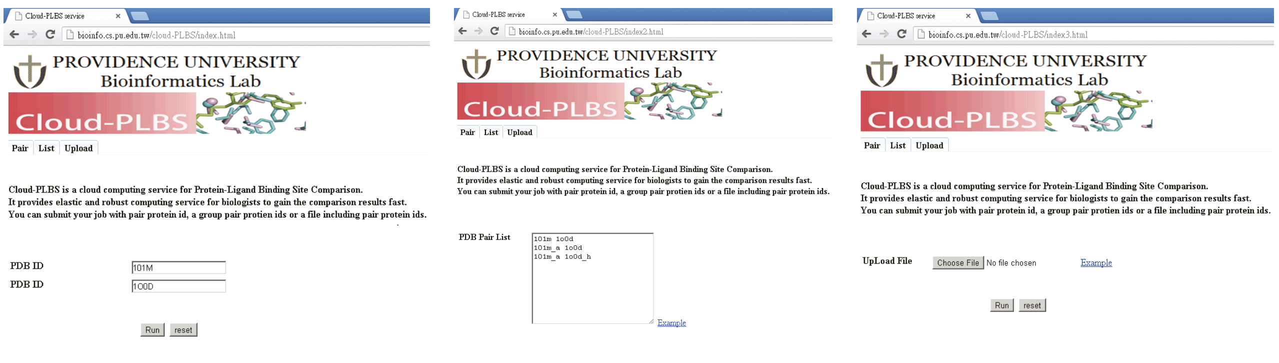
Step 3. Protein structures are compared using a sequence order-independent profile-profile alignment (SOIPPA) algorithm.

Step 4. Similarity between two binding sites is determined through the combination of geometrical fit, residue conservation and physiochemical similarity.

In Cloud-PLBS, each paired protein structure comparison is regarded as an SMAP job. Each SMAP job compares two ligand binding sites by the four stages listed above.

2.2. Cloud-PLBS by Combining Hadoop and Virtualization. As mentioned above, Cloud-PLBS comprises Hadoop, virtualization, and SMAP. Hadoop coordinates computing nodes to parallelize distributed data. Parallel computing applications are developed via the map/reduce parallel programming model. The standard map/reduce mechanism has been applied in many successful cloud computing service providers, such as Yahoo, Amazon EC2, IBM, and Google. The map/reduce framework of Hadoop is illustrated in Figure 1. Input data are divided into smaller chunks corresponding to the number of mappers. The mapper stage output is formatted as $\langle$key, value$\rangle$ pairs. Output from all mappers is classified by key before being distributed to the reducer. The reducer then combines the keyed values. Its output is also formatted as $\langle$key, value$\rangle$ pairs, where each key is unique.

The Hadoop cluster includes a single master and multiple slave nodes. The master node comprises a job tracker, task tracker, name node and data-node. A slave node, or



(a)

bioinfo.cs.pu.edu.tw/cloud-PLBS/index.html

Cloud-PLBS is a cloud computing service for Protein-Ligand Binding Site Comparison.
It provides elastic and robust computing service for biologists to gain the comparison results fast.
You can submit your job with pair protein id, a group pair protien ids or a file including pair protein ids.

PDB ID 101M
PDB ID 100D

(b)

bioinfo.cs.pu.edu.tw/cloud-PLBS/index2.html

Cloud-PLBS is a cloud computing service for Protein-Ligand Binding Site Comparison.
It provides elastic and robust computing service for biologists to gain the comparison results fast.
You can submit your job with pair protein id, a group pair protien ids or a file including pair protein ids.

PDB Pair List

```
101m 1o0d
101m_a 1o0d
101m_a 1o0d_h
```

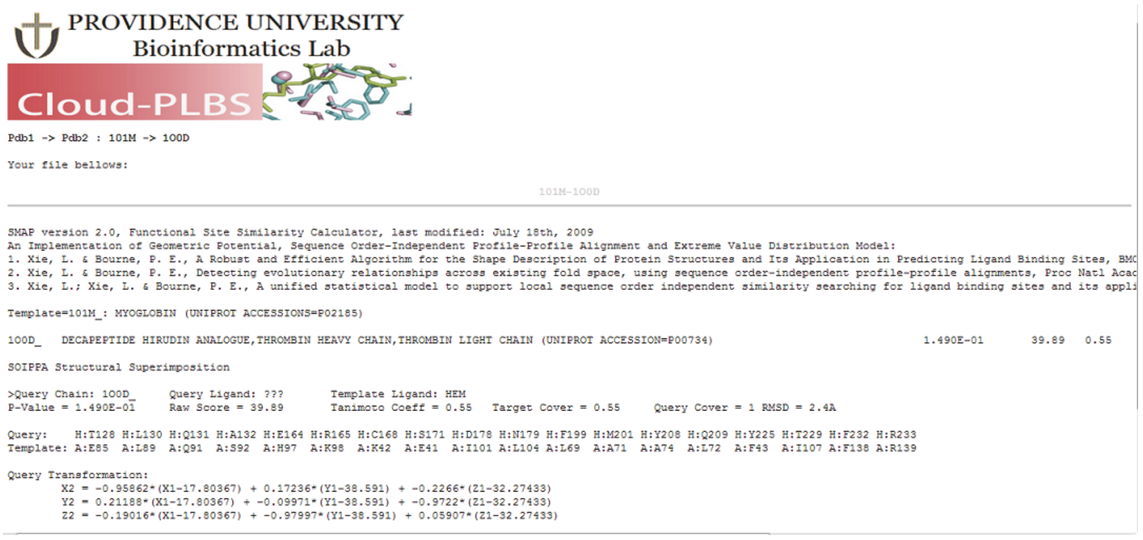
(c)

bioinfo.cs.pu.edu.tw/cloud-PLBS/index3.html

Cloud-PLBS is a cloud computing service for Protein-Ligand Binding Site Comparison.
It provides elastic and robust computing service for biologists to gain the comparison results fast.
You can submit your job with pair protein id, a group pair protien ids or a file including pair protein ids.

UpLoad File

FIGURE 4: Web portal of Cloud-PLBS for entering protein IDs. (a) Two protein IDs. (b) List of paired protein IDs. (c) Upload file.



```
Pdb1 -> Pdb2 : 101M -> 100D

Your file bellows:

                                   101M-100D

SMAP version 2.0, Functional Site Similarity Calculator, last modified: July 18th, 2009
An Implementation of Geometric Potential, Sequence Order-Independent Profile-Profile Alignment and Extreme Value Distribution Model:
1. Xie, L. & Bourne, P. E., A Robust and Efficient Algorithm for the Shape Description of Protein Structures and Its Application in Predicting Ligand Binding Sites, BMC
2. Xie, L. & Bourne, P. E., Detecting evolutionary relationships across existing fold space, using sequence order-independent profile-profile alignments, Proc Natl Acad
3. Xie, L.; Xie, L. & Bourne, P. E., A unified statistical model to support local sequence order independent similarity searching for ligand binding sites and its appli

Template=101M_: MYOGLOBIN (UNIPROT ACCESSIONS=P02185)

100D_   DECAPEPTIDE HIRUDIN ANALOGUE,THROMBIN HEAVY CHAIN,THROMBIN LIGHT CHAIN (UNIPROT ACCESSION=P00734)                    1.490E-01      39.89   0.55

SOIPPA Structural Superimposition

>Query Chain: 100D_     Query Ligand: ???       Template Ligand: HEM
P-Value = 1.490E-01     Raw Score = 39.89       Tanimoto Coeff = 0.55   Target Cover = 0.55     Query Cover = 1 RMSD = 2.4A

Query:    H:T128 H:L130 H:Q131 H:A132 H:E164 H:R165 H:C168 H:S171 H:D178 H:N179 H:F199 H:M201 H:Y208 H:Q209 H:Y225 H:T229 H:F232 H:R233
Template: A:E85  A:L89  A:Q91  A:S92  A:H97  A:K98  A:K42  A:E41  A:I101 A:L104 A:L69  A:A71  A:A74  A:L72  A:F43  A:I107 A:F138 A:R139

Query Transformation:
        X2 = -0.95862*(X1-17.80367) + 0.17236*(Y1-38.591) + -0.2266*(Z1-32.27433)
        Y2 = 0.21188*(X1-17.80367) + -0.09971*(Y1-38.591) + -0.9722*(Z1-32.27433)
        Z2 = -0.19016*(X1-17.80367) + -0.97997*(Y1-38.591) + 0.05907*(Z1-32.27433)
```

FIGURE 5: The result produced by Cloud-PLBS. The protein IDs are 101 M and 100D.

computing node, consists of a data node and task tracker. The job tracker distributes map/reduce tasks to computing nodes within the cluster, ideally those already containing the data, or at least within the same rack. A task tracker node accepts map, reduce and shuffle operations from the job tracker. The architecture of the Hadoop cluster is shown in Figure 2.

Hadoop Distributed File System (HDFS) is the primary file system used by the Hadoop framework. Each input file is split into data blocks that are distributed to data nodes. Hadoop also creates multiple replicas of data blocks and distributes them to data nodes throughout a cluster, ensuring reliable, extremely rapid computations. The name node serves as both a directory namespace manager and a node metadata manager for the HDFS. The HDFS architecture operates on a single name-node.

Resource capacity permitting virtualization technology can host several virtual machines within a physical machine. The proposed cloud service platform combines Hadoop and virtualization technology, such that all nodes of the Hadoop cluster reside in VMs. The cloud computing architecture of Cloud-PLBS is illustrated in Figure 3. As shown in that figure, master node (name node) and slave node (data node) constitute the master VM and slave VM, respectively. Submitted SMAP jobs are recorded in a job queue. The master node periodically obtains SMAP jobs from the job queue and assigns them to slave nodes; a slave node (or mapper) performs the task. Once all of the SMAP jobs are complete, the reducer collects the comparison results from all mappers and stores them in the Network File System (NFS) storage. A single comparison result is stored in a single file in NFS. This architecture imbues Cloud-PLBS with three desirable characteristics: high performance, scalability, and availability.

2.2.1. High Performance. In Cloud-PLBS, the SMAP jobs are performed in parallel by the map/reduce framework. The number of SMAP jobs that can be performed simultaneously is the number of data nodes. If the number of SMAP jobs exceeds the number of data nodes, the number node assigns the remaining jobs as soon as a data node becomes available.

2.2.2. Availability. In the event of system failure, Cloud-PLBS continues performing SMAP jobs via the Hadoop

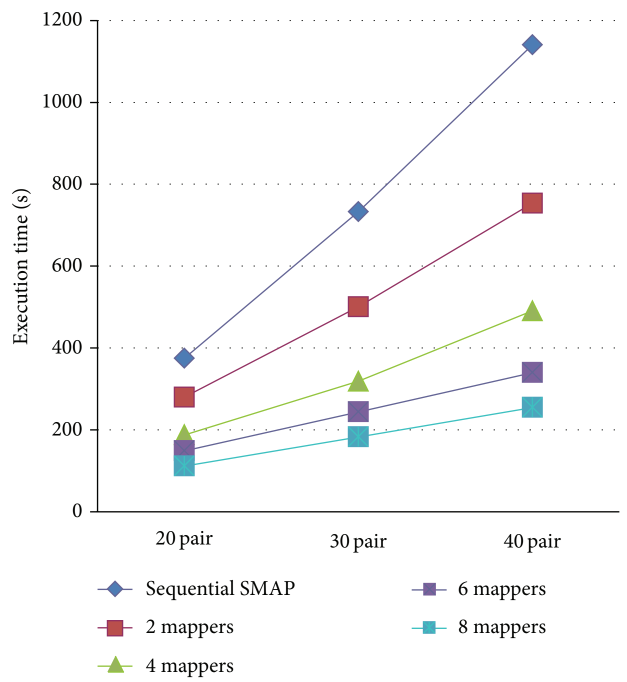


Figure 6: Performance of sequential SMAP program and Cloud-PLBS using 2, 4, 6, and 8 mappers.

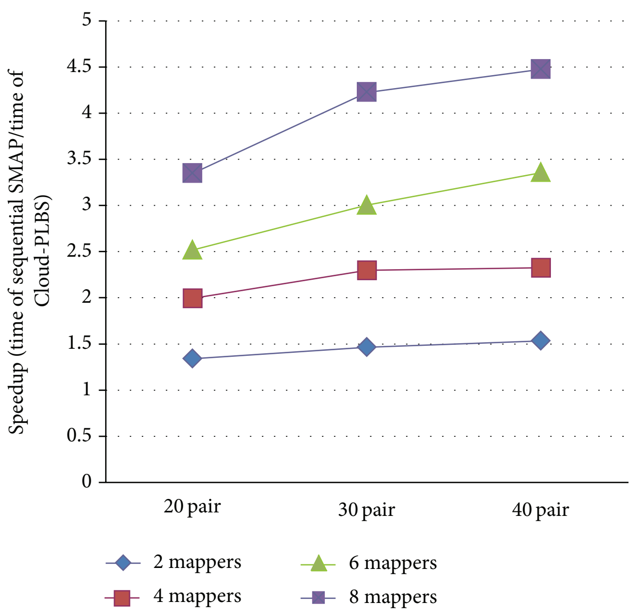


Figure 7: Execution speed of sequential SMAP program and Cloud-PLBS using 2, 4, and 6 mappers.

fault tolerance mechanism. When a data node (mapper) fails during SMAP computation, name node reassigns its job to another slave node (mapper). Therefore, in Cloud-PLBS, all of the submitted SMAP jobs are executed in the event of data node failure. A hardware failure on the physical server will terminate all virtual machines running on it. In this more catastrophic event, SMAP jobs can be reassigned to several new virtual machines created on available hosts. As a result of this operation, Cloud-SMAP has high availability.

2.2.3. Scalability. If excessively many SMAP jobs are submitted, Cloud-PLBS can create new slave VMs as data nodes to accept more jobs, leading to enhanced performance. New VMs are easily created in the Cloud-PLBS architecture. At the same time, redundant VMs can be destroyed to preserve physical resources.

3. Cloud-PLBS Platform

Cloud-PLBS is a software (SaaS) as a service service operating under the Hadoop framework and virtualization technology. The cloud computing platform is composed of an NFS server and four IBM blade servers in the Providence University Cloud Computation Laboratory. Each server is equipped with two Quad-Core Intel Xeon 2.26 GHz CPUs, 24 G RAMs, and 296 G disks. Each server can accommodate 8 virtual machines; each virtual machine is set to one core CPU, 2 G RAM, and 30 G disk running under the Ubuntu operating system version 10.4 with Hadoop version 0.2 MapReduce framework. Each virtual machine is responsible for a map operation and a reduce operation. Therefore, up to eight map/reduce operations may be undertaken.

Figure 4 shows the web portal of Cloud-PLBS. Data may be entered in three ways: by entering two protein IDs (Figure 4(a)), by listing several pairs of protein IDs (Figure 4(b)), or by uploading containing paired protein IDs (Figure 4(c)). All of these pair protein IDs are recorded in a job queue upon submission. The name node (mater node) extracts the paired protein IDs from the queue, and assigns individual SMAP jobs to data nodes (slave nodes). Figure 5 shows the results of comparisons produced by Cloud-PLBS.

4. Performance Evaluation

To assess the performance of the proposed cloud service, we compared the execution time between stand-alone SMAP and Cloud-PLBS. The performance of both programs depends upon the number of SMAP jobs (the number of paired protein IDs) and the number of computing nodes (the number of VMs). Therefore, the performance between the programs is tested with respect to these two factors. The results are shown in Figure 6. As shown in the figure, the execution time of 20 protein pairs (jobs) can be reduced from 375 seconds (consumed by the sequential SMAP program) to 280 seconds, 188 seconds, 149 seconds, and 112 seconds by executing Cloud-PLBS with 2, 4, 6, and 8 mappers, respectively. Given 20, 30 and 40 protein pairs, Cloud-PLBS with 2, 4, and 6 and 8 mappers saves roughly 30%, 54%, 66%, and 74% execution time (relative to sequential SMAP) in average, respectively (see Table 1). Figure 7 demonstrates the enhanced speed achieved by Cloud-PLBS using different numbers of mappers. Clearly, the execution time is effectively reduced when more than two mappers are involved. In general, more mappers (VMs) achieve a faster processing speed.

Table 1: Execution time and proportional reduction (relative to sequential SMAP) of Cloud-PLBS using different numbers of mappers.

Method	20 pairs		30 pairs		40 pairs	
	Execution time (sec)	Reduction rate	Execution time (sec)	Reduction rate	Execution time (sec)	Reduction rate
Sequential SMAP	375		733		1141	
Cloud-PLBS for 2 mappers	280	24.44%	501	31.66%	754	33.92%
Cloud-PLBS for 4 mappers	188	48.87%	319	56.49%	491	56.97%
Cloud-PLBS for 6 mappers	149	60.27%	244	66.72%	340	70.21%
Cloud-PLBS for 8 mappers	112	70.13%	183	75.03%	255	77.65%

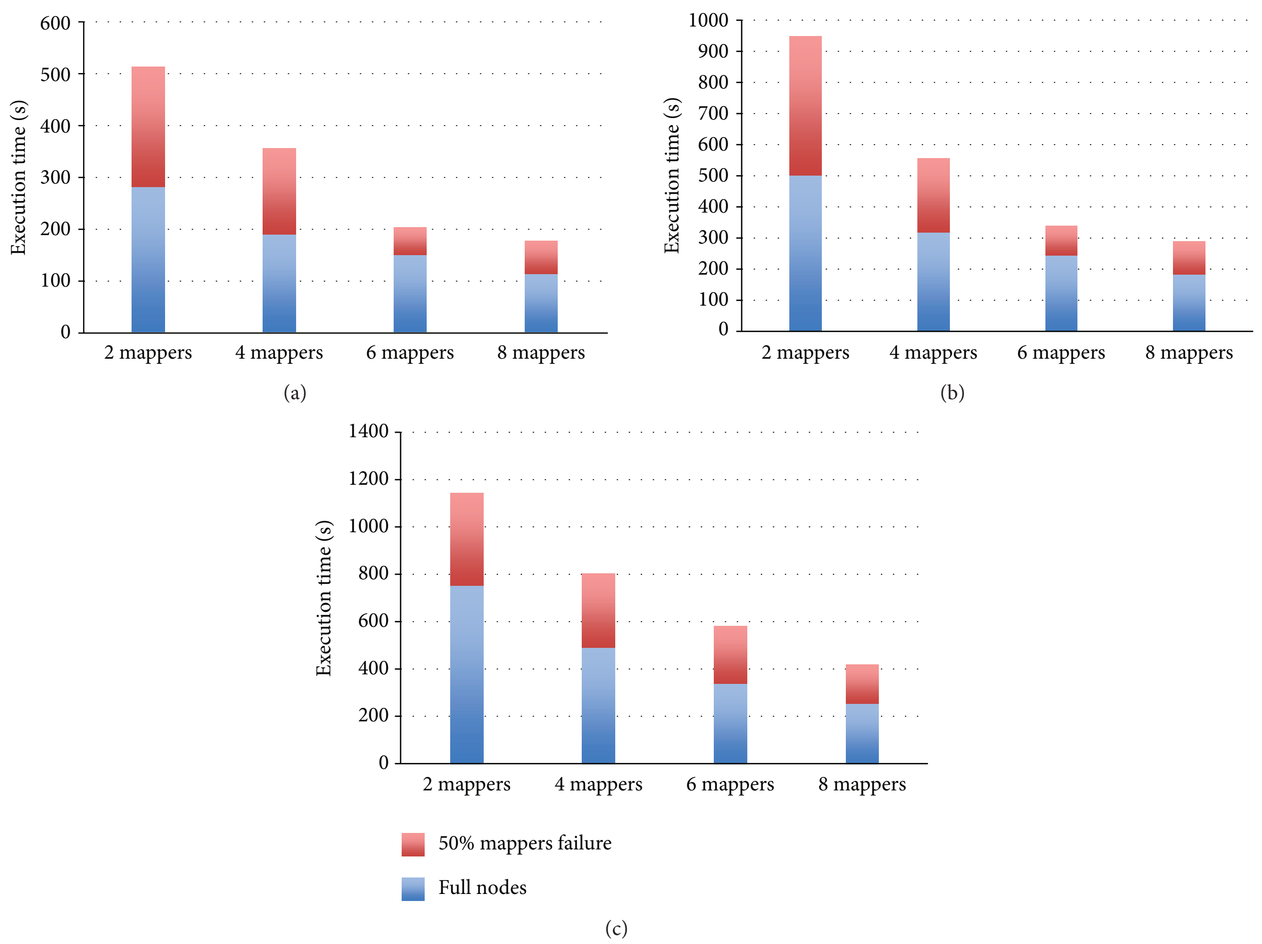


Figure 8: Execution time of a half of node failure of Cloud-PLBS. (a) 20 pair (b) 30 pair (c) 40 pair.

To evaluate the reliability and availability of the proposed cloud service, we performed a simulation to observe the performance when mappers fail. In this simulation, half of the mappers failed in the duration of executing SMAP. According to the features of Hadoop, the computing process at the failed node is able to continue at another node that has the replica of data of the failed node. In this simulation, the heartbeat time is set to one minute, and the number of replica is set to three as default. Therefore, all of jobs can be completed even when some of the nodes fail. Figures 8(a), 8(b), and 8(c) demonstrate the performance between the different number of nodes meeting corresponding half of nodes fail for processing 20 pair, 30 pair, and 40 pair data set, respectively. The execution time with no failure is shown as the blue bar, and the execution time with failure in a half of nodes is the sum of blue bar and red bar which is extra time when failure occurrence. From the experiment results, it shows that the jobs can be completed less than the double successful execution time in the proposed service. Although half of the nodes fail, the execution time of redundancy is related to the number of nodes too. There are extra 165 seconds for 8 mappers, 263 seconds for 6 mappers, 313 seconds for 4 mappers, and 391 seconds for 2 mappers when half of the nodes fail Occurs, respectively. Thereby, our cloud service is node failure-free.

5. Conclusion

The detection and characterization of protein ligand binding sites and their interactions with binding partners are an essential component of modern drug design. The software tool SMAP was designed to achieve these goals. Although SMAP outperforms most existing ligand binding site comparison tools, it cannot achieve the high scalability and availability demanded by huge database searching.

In this paper, we exploit the new internet service concept known as cloud computing. The proposed cloud computing service is called Cloud-PLBS (where PLBS denotes protein-ligand binding site). The platform integrates the Hadoop framework, virtualization technology, and SMAP tool to guarantee high performance, availability, and scalability. Cloud-PLBS ensures that all submitted jobs are properly completed, even on a large cloud platform where individual nodes or network components are prone to failure. We experimentally verified that the platform is computationally more efficient than standard SMAP. Therefore, it presents as a desirable tool for analyzing protein structure and function under reasonable time constraints.

Conflict of Interests

This paper has a potential conflict of interests with the SMAP software developed by J. Ren, L. Xie, W. Li, and P. Bourne and published in Nucleic Acids Research Web Server Issue (2010). Therefore, the authors declare the conflict of interests with SMAP software and Cloud-PLBS here.

Acknowledgment

This research was partially supported by the National Science Council under the Grants NSC-99-2632-E-126-001-MY3 and NSC-100-2221-E-126-007-MY3.

References

[1] J. Ren, L. Xie, W. W. Li, and P. E. Bourne, "SMAP-WS: a parallel web service for structural proteome-wide ligand-binding site comparison," *Nucleic Acids Research*, vol. 38, supplement 2, pp. W441–W444, 2010.

[2] L. Xie and P. E. Bourne, "A robust and efficient algorithm for the shape description of protein structures and its application in predicting ligand binding sites," *BMC Bioinformatics*, vol. 8, supplement 4, article S9, 2007.

[3] L. Xie and P. E. Bourne, "Detecting evolutionary relationships across existing fold space, using sequence order-independent profile-profile alignments," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 105, no. 14, pp. 5441–5446, 2008.

[4] L. Xie, L. Xie, and P. E. Bourne, "A unified statistical model to support local sequence order independent similarity searching for ligand-binding sites and its application to genome-based drug discovery," *Bioinformatics*, vol. 25, no. 12, pp. i305–i312, 2009.

[5] J. D. Durrant, R. E. Amaro, L. Xie et al., "A multidimensional strategy to detect polypharmacological targets in the absence of structural and sequence homology," *PLOS Computational Biology*, vol. 6, Article ID e1000648, 2010.

[6] S. L. Kinnings, N. Liu, N. Buchmeier, P. J. Tonge, L. Xie, and P. E. Bourne, "Drug discovery using chemical systems biology: repositioning the safe medicine Comtan to treat multi-drug and extensively drug resistant tuberculosis," *PLoS Computational Biology*, vol. 5, no. 7, Article ID e1000423, 2009.

[7] J. R. Miller, S. Dunham, I. Mochalkin et al., "A class of selective antibacterials derived from a protein kinase inhibitor pharmacophore," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 106, no. 6, pp. 1737–1742, 2009.

[8] L. Xie, J. Li, L. Xie, and P. E. Bourne, "Drug discovery using chemical systems biology: identification of the protein-ligand binding network to explain the side effects of CETP inhibitors," *PLoS Computational Biology*, vol. 5, no. 5, Article ID e1000387, 2009.

[9] L. Xie, J. Wang, and P. E. Bourne, "In silico elucidation of the molecular mechanism defining the adverse effect of selective estrogen receptor modulators," *PLoS Computational Biology*, vol. 3, no. 11, article e217, 2007.

[10] S. Krishnan, L. Clementi, J. Ren, P. Papadopoulos, and W. Li, "Design and evaluation of Opal2: a toolkit for scientific software as a service," in *Proceedings of the 5th 2009 World Congress on Services (SERVICES '09)*, pp. 709–716, IEEE Conference, Los Angeles, Calif, USA, September 2009.

[11] Hadoop—Apache Software Foundation project home page, http://hadoop.apache.org/.

[12] R. C. Taylor, "An overview of the Hadoop/MapReduce/HBase framework and its current applications in bioinformatics," *BMC Bioinformatics*, vol. 11, supplement 12, article S1, 2010.

[13] M. C. Schatz, "CloudBurst: highly sensitive read mapping with MapReduce," *Bioinformatics*, vol. 25, no. 11, pp. 1363–1369, 2009.

[14] B. Langmead, M. C. Schatz, J. Lin, M. Pop, and S. L. Salzberg, "Searching for SNPs with cloud computing," *Genome Biology*, vol. 10, no. 11, article R134, 2009.

[15] C. L. Hung and Y. L. Lin, "Implementation of a parallel protein structure alignment service on cloud," *International Journal of Genomics*, vol. 2013, Article ID 439681, 8 pages, 2013.

[16] C. L. Hung and C. Y. Lin, "Open reading frame phylogenetic analysis on the cloud," *International Journal of Genomics*, vol. 2013, Article ID 614923, 9 pages, 2013.

[17] Amazon Elastic Compute Cloud (Amazon EC2), http://aws.amazon.com/ec2/.

[18] H. M. Berman, J. Westbrook, Z. Feng et al., "The protein data bank," *Nucleic Acids Research*, vol. 28, no. 1, pp. 235–242, 2000.

Hindawi Publishing Corporation
BioMed Research International
Volume 2013, Article ID 909717, 8 pages
http://dx.doi.org/10.1155/2013/909717

Research Article

A High Performance Cloud-Based Protein-Ligand Docking Prediction Algorithm

Jui-Le Chen,[1,2] Chun-Wei Tsai,[3] Ming-Chao Chiang,[4] and Chu-Sing Yang[1]

[1] *Department of Electrical Engineer, National Cheng Kung University, Institute of Computer and Communication Engineering, Tainan 70101, Taiwan*
[2] *Department of Digital Multimedia Design, Tajen University, Pingtung 90741, Taiwan*
[3] *Department of Applied Informatics and Multimedia, Chia Nan University of Pharmacy & Science, Tainan 71710, Taiwan*
[4] *Department of Computer Science and Engineering, National Sun Yat-sen University, Kaohsiung 80424, Taiwan*

Correspondence should be addressed to Chun-Wei Tsai; cwtsai0807@gmail.com

Received 7 December 2012; Accepted 12 April 2013

Academic Editor: Ching-Hsien Hsu

The potential of predicting druggability for a particular disease by integrating biological and computer science technologies has witnessed success in recent years. Although the computer science technologies can be used to reduce the costs of the pharmaceutical research, the computation time of the structure-based protein-ligand docking prediction is still unsatisfied until now. Hence, in this paper, a novel docking prediction algorithm, named fast cloud-based protein-ligand docking prediction algorithm (FCPLDPA), is presented to accelerate the docking prediction algorithm. The proposed algorithm works by leveraging two high-performance operators: (1) the *novel* migration (information exchange) operator is designed specially for cloud-based environments to reduce the computation time; (2) the *efficient* operator is aimed at filtering out the worst search directions. Our simulation results illustrate that the proposed method outperforms the other docking algorithms compared in this paper in terms of both the computation time and the quality of the end result.

1. Introduction

The ultimate goal of most people is looking for every possible solution that provides a more comfortable life; therefore, most researchers have done their best to advance the interest of human from different positions, domains, concerns, and backgrounds. One important work for the lighthearted life is finding a new drug for particular disease. Needless to say, drug design can always help human health because it can be used in preventing and curing diseases. The structure-based drug design [1] usually can be used to predict the interactions between small drug molecules and protein receptor complexes, and now, it is one of the well-known computer-aided drug design methods. With advance of computer technologies, the prediction method based on theoretical computing method and molecular modeling to establish the three-dimensional structure for designing a new drug molecule can be used to speed up finding the good possible candidate solutions. As observed by Volkamer et al. [2], even though we invest more than one thousand billion US dollars for drug development, the prediction accuracy and the development time are still unsatisfied. In other words, the prediction accuracy of the docking prediction is no more than 70% while the drug discovery process still takes a tremendous amount of computation time just to find the possible drugs.

To measure the simulation results, the Van der Waals (VDW), atomic radius, charge, torsional angles, intermolecular hydrogen bonds, and hydrophobicity of the contact force are usually used to bind the energy between receptor and ligand. The empirical energy function [3], such as the score function, is usually used to evaluate the results of ligand molecular docking conformation which is suitable or not for binding area of receptor. Each candidate solution of the protein-ligand docking prediction (PLDP) problem contains the three-dimensional coordinates of the ligand center point, the four orientation parameters, and some additional special atoms, such as coal, nitrogen, and hydrogen whose free torsion degrees are used as the parameters. The set of candidate solutions X can be expressed as the total energy of the protein-ligand interaction and the sum of the internal energy for both ligand and protein which is given as follows:

$$\min E_{\text{total}}(X) = E_v + E_h + E_e + E_i + E_d, \tag{1}$$

where E_v, E_h, and E_e denote, respectively, the interaction forces of intermolecular, namely, Van der Waals forces, hydrogen bond, and electronic potential energy; E_i is the internal attraction of ligand and protein molecules; E_d is the desolvation of binding area meaning the performance for hydrophobic.

Because the search space of possible conformations is extremely large, how to reduce the computation time has become a very important research issue, especially that all these problems are usually either NP-hard or NP-complete problem [4]. Hence, a high-performance search method is required to speed up the overall performance of the search process. This explains why many search methods for reducing the computation time have been presented to solve the docking problem [5]. The heuristic algorithms, such as simulated annealing (SA) [6] and genetic algorithm (GA) [3, 7], provide a fast method to search for approximate solutions which are faster than the brute force search algorithms and traditional search algorithms. As such, it is one of the efficient ways for solving the docking problem [8].

To enhance the performance of heuristic algorithms for the docking problem, this paper presents a novel protein-ligand docking prediction algorithm to speed up the process of drug design and development on a *cloud computing* environment, by using a novel migration method while at the same time attempting to improve the accuracy rate (success rate) of prediction by using an efficient operator to filter out the worst search direction.

The rest of the paper is organized as follows. In Section 2, a brief introduction to the parallel computing for the protein-ligand docking prediction problem is given. After that, the concept and design of the proposed algorithm are detailed in Section 3. Section 4 begins with a brief description of the materials presented in this paper and then compares the simulation results of the proposed algorithm with those of other protein-ligand docking prediction algorithms. The conclusion is drawn in Section 5.

2. Related Work

In addition to using metaheuristics to improve the performance of the docking prediction algorithm as we mentioned in Section 1, another way is to enhance the computation power of hardware, such as parallel computing. However, since the communication and synchronization costs of the search algorithm for PLDP on a cloud computing environment are much higher than those on a grid or cluster computing environments, to enhance the performance of the protein-ligand docking prediction process, we have to take into consideration these factors in the design and development of protein-ligand docking prediction algorithm (PLDPA).

Among others, three major parallel computation models are usually used in the evolutionary computation and other metaheuristics [9–11] for enhancing the search performance, to not only cut down the computation time but also improve the quality of the end result. These parallel computation models are master-slave model [9], fine-grained model (cellular model) [9], and coarse-grained model (island model) [11]. For instance, the master-slave model for genetic algorithm will divide the population into several subpopulations and then assign them to different processors to accelerate the computation speed. For the fine-grained model, it also divides the population into several subpopulations each of which are assigned to different processors or machines. But each subpopulation can only exchange information with other subpopulations to which they are directly connected. The coarse-grained model (also called the island model) uses the concept of island and migration to exchange the information between subpopulations. Unlike traditional computing approaches, the parallel computation models take into account both the architecture of the search algorithm and the computation resources together. The major concerns now become how the chromosomes communicate and exchange information between the subpopulations to affect their search performance [12–15].

For the protein-ligand docking prediction (PLDP), Wang et al. [16] use the master-slave model for Lamarckian genetic algorithm (LGA) (one kind of hybrid genetic algorithm for which the genetic algorithm (GA) plays the role of global search while the local search algorithm plays the role of fine-tuning the search results found by GA) to speed up the computation time of the docking prediction process. Various successful works have been presented in recent years. For instance, Kannan and Ganji [17] used GPU to speed up the search process of PLDP. Sampling methods [18] have been employed to provide not only better initial seeds but also the possibility of finding better results. Some researchers [19, 20] attempted to redesign or modify the scoring function for PLDP because the scoring function takes a large percentage of the computation time [17]. These researches focus on either reducing the computation time, increasing the accuracy of prediction, or both.

As a promising research area in recent years (after the grid [21] and cluster [22, 23] system), cloud computing provides a better way to enhance the performance of PLDP, such as a tremendous amount of the compute and storage resources, which leverages the strengths of grid computing and cluster computing. How to apply the PLDP algorithm to this new infrastructure have nowadays become a critical research issue. Figure 1 gives the details of master-slave model and island model. For these distributed computing models, the main concern is how to divide the computations of a search algorithm and then dispatch them to the computer nodes to improve the search performance.

3. The Proposed Method

3.1. Concept. Since most heuristic algorithms do not guarantee that they can find the optimal solution, one of the most

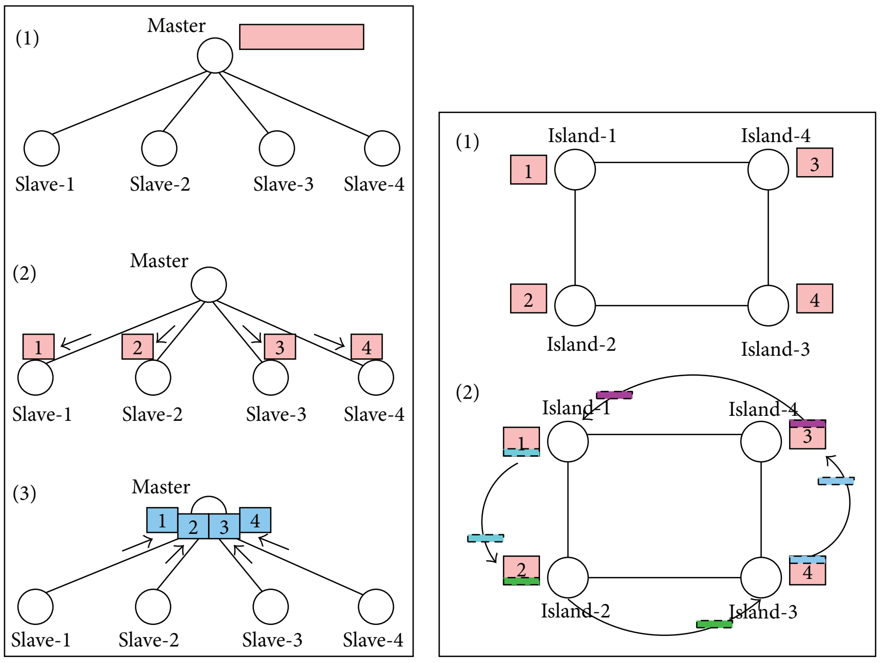


FIGURE 1: A simple example for illustrating the parallel computation models. (a) Master-slave model; (b) island model.

important problems for the heuristic algorithms to deal with is to balance the computation cost and the quality of the solution. For the cluster computing environment, most computer resources (nodes) are centralized in the same place; therefore, the communication and synchronization costs are not as high. For grid and cloud computing environments, they are, however, an important issue because most computing nodes are not centralized in the same place. Also, from the perspective of algorithm design, because the total computation time of each slave (or island) is different, no matter which of the parallel computation models is used, it is almost unavoidable to waste time waiting for the other slaves to finish their tasks.

The proposed algorithm integrates two efficient operators. The first one is a novel migration operator to mitigate the costs incurred on a cloud-based environment whereas the second operator is the pattern reduction operator [29, 30] to filter out the worst search directions. Just like other researches on protein-ligand docking prediction and [29], the main focus of this research is not only on the development of a faster search process but also on getting better solutions for the binding locations of the protein-ligand docking prediction.

3.2. The FCPLDPA. As shown in Algorithm 1, the proposed algorithm (FCPLDPA) is applied to the Lamarckian genetic algorithm (LGA) to solve the docking problem for the rigid protein and flexible drug molecules. The FCPLDPA will first construct the initial solution S and then divide it into m subpopulations (the number of m is predefined by the user, which usually matches the number of virtual machines (computing nodes) that can be used for solving the docking prediction problem) $S = \{S_1, S_2, \dots, S_m\}$. Next, each subpopulation will be dispatched to a virtual machine (island). Like the island model, each subpopulation will now undergo the evolution process independently from each other except that some of the chromosomes are *immigrations* or *emigrations* of the island. Unlike the classical island model, the migrations of the proposed algorithm are not restricted to be at the same iteration number (era) because such a restriction may delay the migration process, by waiting for the other islands to converge. Algorithm 2 gives the details of the evolution process for the islands and the migration procedure, which include the evolution process of the simple genetic algorithm—selection, crossover, and mutation operations. However, in addition to applying the pattern reduction operator [29] (as shown in line 5 of Algorithm 2)

```
(1) Create an initial population S.
(2) Divide the population into m subpopulations and
    dispatch them to m islands.
(3) Do
(4)     For each island
(5)         **Perform** the evolutionary process (EP).
(6)     End
(7) While the stop criterion is satisfied, then stop and output
    the best result.
```

ALGORITHM 1: Outline of FCPLDPA for the protein-ligand docking prediction problem.

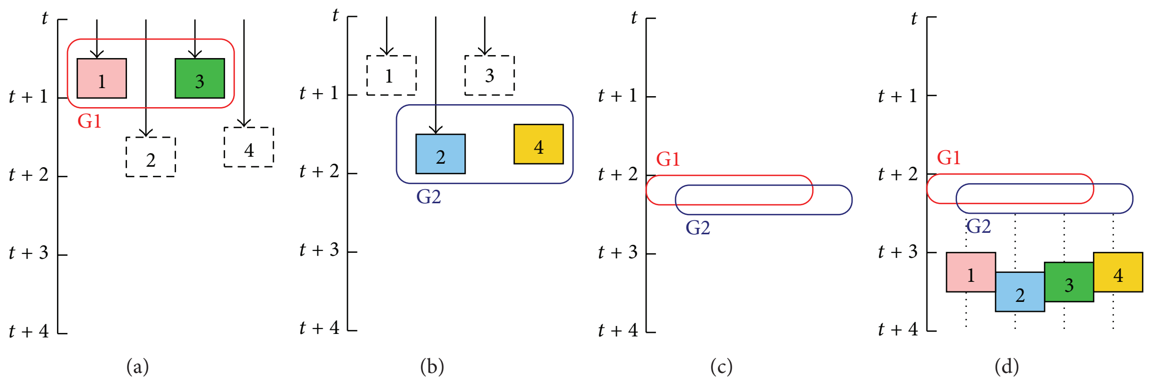


FIGURE 2: A simple example illustrating how the proposed algorithm works. (a) island-1 and island-2 complete their work at time $t + 1$; (b) island-2 and island-4 complete their work at about time $t + 2$; (c) after the exchange of information between all the islands, FCPLDPA will synchronize the information; (d) and then FCPLDPA will continue to let all the islands exchange information with approximate island.

```
(1) $T_i \leftarrow 0$ and $\mathcal{F}_i \leftarrow$ false.
(2) Calculate the fitness value of each chromosome.
(3) Select the approximate chromosomes to be the parents
    of the next generation.
(4) Perform the crossover and mutation operators to create
    the children.
(5) Apply the pattern reduction (PR) operator to eliminate
    computations that are essentially redundant.
(6) $T_i \leftarrow T_i + 1$
(7) If $(T_i \% M_T = 0)$ Synchronize the migration.
(8)     If $\psi(T_i, \mathcal{F}_i, \mathcal{M}_i) = 1$, perform the emigration.
(9)     If $\psi(T_i, \mathcal{F}_i, \mathcal{M}_i) = 2$, perform the immigration.
(10)    Go to step 2 otherwise.
(11) End If
```

ALGORITHM 2: Outline of the evolutionary process (EP).

to the proposed algorithm, the timing for migrating the chromosomes to the other islands (virtual machines) is the main concern of this paper, as shown in lines 7 and 10 of Algorithm 2. The migration mechanism is as given below:

$$\psi(T_i, \mathcal{F}_i, \mathcal{M}_i) = \begin{cases} 1 & \text{if } \mathcal{T}_i = \mathcal{T}_H,\ \mathcal{F}_i = \text{false} \\ 2 & \text{if } \mathcal{T}_i = \mathcal{T}_H,\ \mathcal{F}_i = \text{false},\ \mathcal{M}_i = \text{true}, \\ 0 & \text{otherwise}, \end{cases} \quad (2)$$

where M_T denotes the timing (i.e., migration interval) to synchronize all the islands (virtual machines), T_i is the number of iterations that has been performed on the evolution process of the ith virtual machine (VM), $\mathcal{T}_H$ is the threshold to determine the timing to migrate the chromosomes to the other islands, $\mathcal{F}_i$ is the flag to show whether any migration process has been done or not since the previous migration process was performed, and $\mathcal{M}_i$ is the policy of master to determine the timing to exchange information between islands. More precisely, because the migration process of the islands occurs for particular islands (as shown in Figure 1, this means that some of the islands will complete their tasks at about the same time). The synchronization process is needed to be performed after some of the migration processes (or once every M_T iterations) so that the proposed algorithm is able to transmit the information from one island to the others. Note that M_T is defaulted to 25, and T_i is defaulted to 5.

In the *case of emigration*, that is, the first case in (2), it illustrates that the migration process will be performed when the number of iterations at the ith island is equal to T_H and F_i is false. In this case, the proposed algorithm will select a chromosome (elite solution) to migrate to the other islands, just like the island model of GA. However, in the *case of immigration*, that is, the second case in (2), any chromosome that wants to enter the ith island must also satisfy the condition $T_i = T_H$ (i.e., after T_H iterations). The master needs to choose the islands the evolution processes of which are completed at about the same time to exchange the information. It will become time oriented; thus, most of the migration processes need not to wait for the evolution processes of the other islands to finish.

A simple example is given in Figure 2 to explain the main idea of the migration strategy of the proposed algorithm. Figure 2(a) shows that not all the islands will complete their work at the same time if the time for migration is set up to, say, once every five iterations. The problem of all the islands not being able to finish the same work at the same time is owing to two important factors. One is due to the communication cost (including the synchronization and other transmission costs) while the other is due to the randomness of the convergence speed of most metaheuristics. The proposed algorithm attempts to deal with this problem, by letting islands exchange the information once they completed their work at about the same time. A very simple method is to let island-b exchange information with island-a if the completion time of island-b is closest to that of island-a and they have not exchanged information to the other islands in this information exchange round (i.e., over the past five iterations). Similar to Figure 2(a), Figure 2(b) also lets island-2 and island-4 exchange information (chromosome migration) at time $t + 2$ because their completion times are closest to each other.

Figure 2(c) shows that when all the islands have exchanged their information after M_T iterations that is, migrate the chromosomes to other islands, the proposed algorithm will then perform a synchronization procedure. That is, it will randomly pick the chromosomes from these groups (island-1 and island-3 as the first group while island-2 and island-4 as the second group) and migrate them to islands of the other group so that the information can be circulated to all the islands. After that, as shown in Figure 2(d), FCPLDPA will continuously let all the islands evolve their subpopulations again and migrate their chromosomes to the other islands later. In summary, if the communication costs are high and the convergence speeds of all the islands are not the same, the proposed algorithm can be used to avoid wasting time to wait for the other islands to complete their work. The detailed analysis will be given in later sections.

4. Simulation Results

In this paper, the performance of the proposed algorithm is evaluated by using it to solve the protein-ligand docking problem. All the empirical analyses are conducted on 16 VMs, each of which consists of one CPU (2.5 GHz), 4 GB of memory, a 100 MBps NIC card, and GNU C++ v4.12, and runs CentOS_64 v6.2. Also, the test platform for docking is AutoDock 4.2 [31].

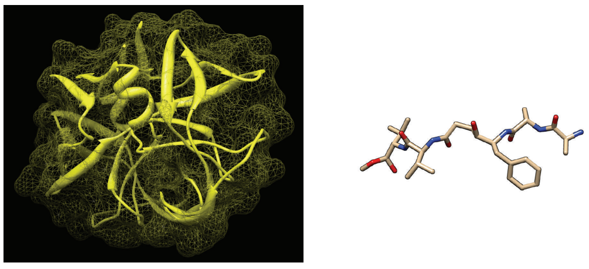
Several state-of-the-art algorithms, namely, differential evolution [32], particle swarm optimization [33], Lamarckian genetic algorithm with island model (parallel GA; PGA) [34, 35], FCPLDPA without PR, and FCPLDPA with PR, are applied to the AutoDock environment to search for the possible protein-ligand binding sites. The energy function which calculates the energy value between the protein and ligand molecule is used by these search algorithms to determine which conformation is the candidate with the most appropriate binding points. The details of composition formula for free energy expressions could be referred to the study of [3]. For all these algorithms, the population size N is fixed at 256, the subpopulation size for the parallel model is fixed at N/I_n where I_n denotes the number of islands, the crossover rate is set equal to 0.9, the mutation rate is set equal to 0.08, and the number of generations ℓ is set equal to 10,000.

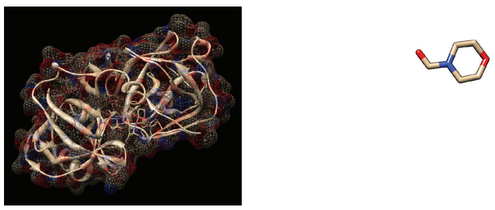
4.1. Materials. The molecular docking problem can be regarded as the key matching problem where the lock and key are the receptor and ligand, respectively, and the goal of all the simulations is to provide an efficient way to find the approximate positions of the key and lock from a large search space or the candidate set of drugs. Although until now, the accuracy of most candidate sets of approximate positions (solutions) found by a search algorithm is not as precise as it is supposed to be; it does provide an efficient way to find out a good drug molecule which is not validated yet by a laboratory. In this paper, an effective tool for drug design based on structure-based protein molecule, AutoDock [6], is used to evaluate the proposed algorithm and the other state-of-the-art docking prediction algorithms, such as GA and parallel GA (PGA). In addition, four different kinds of data sets, as shown in Figure 3, are used to evaluate the performance of the proposed algorithm and the state-of-the-art docking prediction algorithms compared in this paper. The data sets are taken from the RCSB Protein Data Bank database (http://www.pdb.org/).

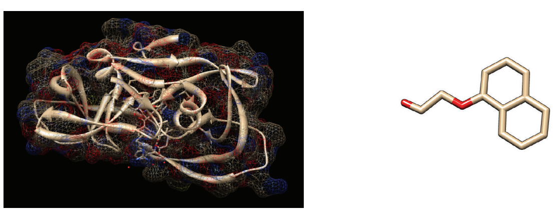
4.2. Results. Our observation shows that most computation costs of LGA come from the local search process. But this characteristic will be changed for the parallel GA on a cloud computing environment. As shown in Figure 4, two interesting phenomena can be easily observed. First, the communication costs will increase as the number of islands (virtual machines) increases. These results show that the communication costs and the convergence speed of different islands all may affect the performance of the system. Second, the local search process may change the percentage of the computation time for all the operators of protein-ligand docking prediction. Figures 4(a) and 4(b) show that the local search of PGA takes much more computation time than the function evaluation, especially when we invest much more resources to the local search process for the four different datasets which differs from the observation described in [17] because Kannan and Ganji believe that the function evaluation takes most of the computation time of the whole convergence process. However, our observation is that the function evaluation will not affect the computation time of the whole search process. These results help us emphasize that the local search, the communication cost, and the different convergence speed of all the islands are the other important factors to be taken into account for the computation time of protein-ligand docking prediction, especially when we are using the cloud computing environment to solve the protein-ligand docking prediction problem.

As shown in Table 1, seven different datasets are used to evaluate the performance of the proposed algorithm and the other docking prediction algorithms. For each algorithm, the table gives the percentage of time taken by the initialization, selection, crossover, mutation, function evaluation, local search, and send and receive operators. The results show that the proposed algorithm outperforms the PGA in most cases, either without PR or with PR in terms of the success rate and the average time.

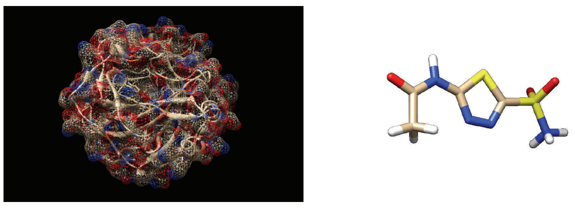
Comparison of the proposed algorithm with PGA shows that if the send and receive costs can be decreased, the overall computation time can also be decreased. According to our observation, FCPLDPA without PR can provide a better success rate than PGA and DE because the proposed algorithm postpones the transmission of information from the island to all the other islands; therefore, the search diversity between islands can be maintained. Another strategy for the proposed algorithm is to combine it with PR, called FCPLDPA with PR (FCPLDPA + PR). The simulation results in Table 1 also show that FCPLDPA + PR can provide better results than DE, PSO, PGA, and FCPLDPA alone in terms of the success rate with a little more investment of the computation time.

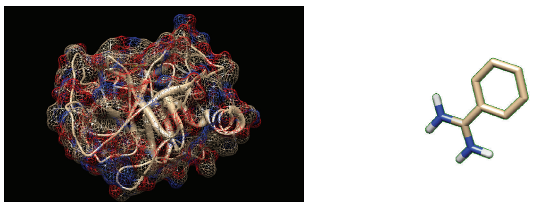

(a) 1AAQ [24]


(b) 1EPO [25]


(c) 1HIV [26]

(d) 4PHV [27]


(e) 1AZM [26]


(f) 3PTB [27]

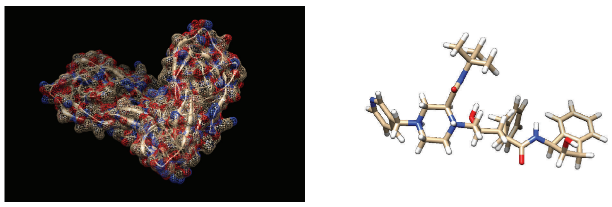
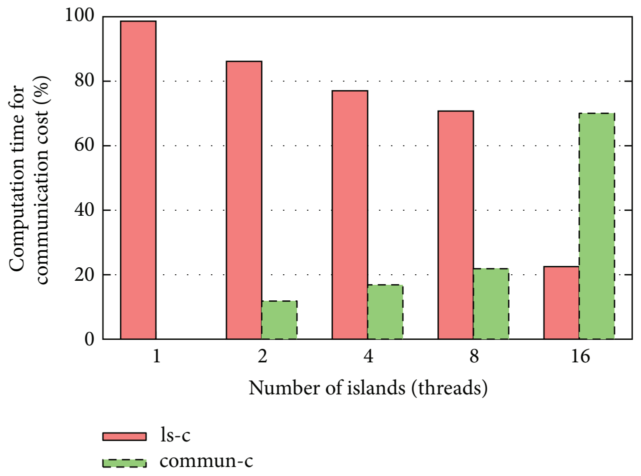
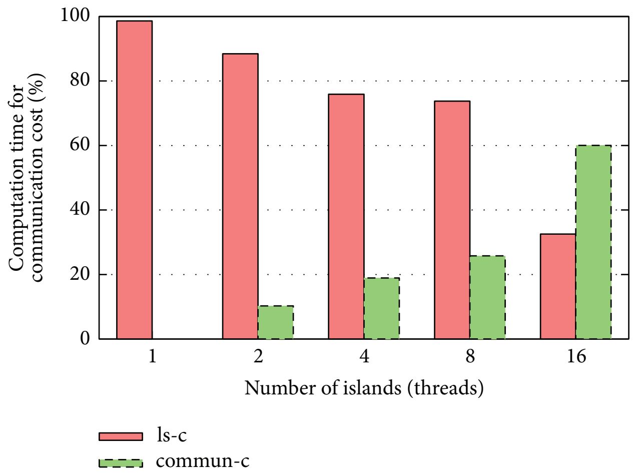

(g) 1HSH [27]

Figure 3: Structure diagrams [28] for both protein and ligand molecules.



(a)



(b)

Figure 4: Percentage of the time due to communication (commun-c) and local search (ls-c) while the number of islands represents LGA. (a) 1AAQ; (b) 1EPO.

Table 1: Comparison of the proposed algorithm with the other docking prediction algorithms.

Islands	1	2	4	8	16	32
Chromosomes	256	128	64	32	16	8
DE (island model)						
Initialization	0.11%	0.13%	0.34%	0.53%	1.34%	2.63%
Mutation	2.39%	2.44%	2.36%	2.37%	2.76%	2.72%
Evaluation	0.61%	0.71%	0.68%	0.66%	0.77%	0.64%
Local search	96.88%	93.92%	90.13%	82.86%	70.59%	48.69%
Send and receive	0.00%	3.51%	7.17%	14.24%	23.31%	43.96%
Success rate	22.86%	27.14%	30.71%	33.57%	40.71%	39.29%
Average time	1,726.52	995.23	506.34	251.93	154.33	104.07
PSO (island model)						
Initialization	0.07%	0.14%	0.26%	0.78%	1.56%	3.12%
Evaluation	2.37%	2.75%	2.74%	2.11%	3.28%	2.94%
Local search	97.55%	95.44%	94.61%	90.96%	82.4%	52.69%
Send and receive	0.00%	1.68%	2.39%	6.15%	13.02%	26.14%
Success rate	17.86%	21.43%	25.00%	31.43%	35.71%	32.14%
Average time	284.41	176.39	87.39	42.94	28.15	18.78
PGA (island model)						
Initialization	0.20%	0.29%	0.68%	0.94%	0.90%	0.90%
Selection	0.20%	0.34%	0.68%	0.94%	0.90%	0.90%
Crossover	0.01%	0.01%	0.01%	0.01%	0.01%	0.01%
Mutation	0.01%	0.01%	0.01%	3.86%	0.10%	0.01%
Evaluation	0.98%	1.69%	3.41%	4.72%	4.44%	4.50%
Local search	98.63%	88.95%	76.72%	62.57%	26.22%	22.20%
Send and receive	0.00%	8.98%	18.29%	28.50%	67.28%	71.35%
Success rate	15.00%	19.29%	21.43%	28.54%	34.29%	27.14%
Average time	572.74	330.71	168.07	84.33	50.14	33.59
FCPLDPA without PR						
Initialization	—	0.34%	0.68%	0.94%	0.90%	0.90%
Selection	—	0.34%	0.68%	0.94%	0.90%	0.90%
Crossover	—	0.01%	0.01%	0.01%	0.01%	0.01%
Mutation	—	0.01%	0.01%	1.29%	0.01%	0.01%
Evaluation	—	1.78%	3.74%	5.26%	5.19%	5.89%
Local search	—	93.35%	84.13%	69.61%	30.69%	26.67%
Send and receive	—	4.43%	10.41%	19.42%	61.69%	65.61%
Success rate	—	25.00%	25.71%	30.71%	37.14%	32.14%
Average time	—	313.49	153.78	75.90	42.69	30.54
FCPLDPA with PR						
Initialization	—	0.34%	0.68%	0.86%	0.90%	0.90%
Selection	—	0.31%	0.68%	0.83%	0.90%	0.90%
Crossover	—	0.01%	0.01%	0.01%	0.01%	0.01%
Mutation	—	0.01%	0.01%	1.43%	0.01%	0.01%
Evaluation	—	1.78%	3.75%	5.02%	4.06%	5.96%
Local search	—	93.20%	83.59%	70.60%	32.66%	28.50%
Send and receive	—	4.48%	10.24%	20.88%	61.17%	64.09%
Success rate	—	27.86%	32.14%	40.00%	47.14%	45.71%
Average time	—	320.91	156.19	77.56	44.07	33.49

5. Conclusion

In this paper, a novel docking prediction algorithm named fast cloud-based protein-ligand docking prediction algorithm (FCPLDPA) is presented to enhance the performance of metaheuristics (i.e., GA-based algorithm) for pharmaceutical research. The simulation results show that the proposed algorithm can not only significantly reduce the computation cost of GA in solving the protein-ligand docking prediction problem by using cloud computing technologies but also improve the quality of the end result by using the pattern reduction method. They also show the possibility of using cloud computing technologies and the dilemmas we need to face when applying the drug prediction approaches to the cloud computing environment. More precisely, many approaches can be used to reduce the computation costs of metaheuristics, such as investing more computing resources to finish the job faster or using better search strategy (i.e., sampling or dimension reduction methods). However, according to our observation, the improvement was not proportional to the investment because the communication costs and the different convergence speeds of virtual machines (islands) all affect the performance of the docking system on cloud. The main purpose of this research is to eliminate the *waiting between different virtual machines* of cloud-based docking prediction algorithm. The simulation results are consistent with our assumptions and observations that the communication costs and the different convergence speeds between islands may strongly impact the performance of the purposed algorithm. Two efficient operators are employed in this paper: (1) the novel migration operator is aimed to avoid wasting of the computation power and waiting for the other virtual machines on a cloud computing environment, and (2) the pattern reduction operator is aimed to enhance the search performance. The main contributions of this research can be summarized as follows: (1) we discovered that the communication costs and the different convergence speeds between virtual machines (islands) will eventually affect the performance of the search algorithm on cloud; and (2) we presented a high-performance cloud-based protein-ligand docking prediction algorithm to deal with this problem to guide the search algorithm to find the approximate candidate solution quickly. In the future, we will focus on finding a more efficient prediction method to improve the accuracy of the solution of FCPLDPA while reducing the computation time of the whole process.

Acknowledgment

This work was supported in part by the National Science Council of Taiwan, ROC, under Contracts NSC101-2219-E-006-004, NSC101-2221-E-041-012, NSC101-2221-E-041-013, and NSC99-2221-E-110-052.

References

[1] J. Shen, X. Xu, F. Cheng et al., "Virtual screening on natural products for discovering active compounds and target information," *Current Medicinal Chemistry*, vol. 10, no. 21, pp. 2327–2342, 2003.

[2] A. Volkamer, D. Kuhn, T. Grombacher, F. Rippmann, and M. Rarey, "Combining global and local measures for structure-based druggability predictions," *Journal of Chemical Information and Modeling*, vol. 52, no. 2, pp. 360–372, 2012.

[3] G. M. Morris, D. S. Goodsell, R. S. Halliday et al., "Automated docking using a Lamarckian genetic algorithm and an empirical binding free energy function," *Journal of Computational Chemistry*, vol. 19, no. 14, pp. 1639–1662, 1998.

[4] R. Ladner, "On the structure of polynomial time reducibility," *Journal of the ACM*, vol. 22, no. 1, pp. 155–171, 1975.

[5] N. Moitessier, P. Englebienne, D. Lee, J. Lawandi, and C. R. Corbeil, "Towards the development of universal, fast and highly accurate docking/scoring methods: a long way to go," *British Journal of Pharmacology*, vol. 153, no. 1, pp. S7–S26, 2008.

[6] D. S. Goodsell and A. J. Olson, "Automated docking of substrates to proteins by simulated annealing," *Proteins*, vol. 8, no. 3, pp. 195–202, 1990.

[7] G. Jones, P. Willett, R. C. Glen, A. R. Leach, and R. Taylor, "Development and validation of a genetic algorithm for flexible docking," *Journal of Molecular Biology*, vol. 267, no. 3, pp. 727–748, 1997.

[8] R. Thomsen, "Flexible ligand docking using evolutionary algorithms: investigating the effects of variation operators and local search hybrids," *BioSystems*, vol. 72, no. 1-2, pp. 57–73, 2003.

[9] E. Cantu-Paz, "A survey of parallel genetic algorithms," *Calculateurs Paralleles, Reseaux et Systems Repartis*, vol. 10, no. 2, pp. 141–171, 1998.

[10] H. Muhlenbein, "Parallel genetic algorithms, population genetics and combinatorial optimization," in *Proceedings of the International Conference on Genetic Algorithms*, pp. 416–421, San Francisco, Calif, USA, 1989.

[11] Y. Maeda, M. Ishita, and Q. Li, "Fuzzy adaptive search method for parallel genetic algorithm with island combination process," *International of Journal of Approximate Reasoning*, vol. 41, pp. 59–73, 2005.

[12] E. Cantú-Paz and D. E. Goldberg, "Efficient parallel genetic algorithms: theory and practice," *Computer Methods in Applied Mechanics and Engineering*, vol. 186, no. 2–4, pp. 221–238, 2000.

[13] E. Cantu-Paz, *Efficient and Accurate Parallel Genetic Algorithms*, Kluwer Academic, Norwell, Mass, USA, 2000.

[14] K. Chellapilla, "Combining mutation operators in evolutionary programming," *IEEE Transactions on Evolutionary Computation*, vol. 2, no. 3, pp. 91–96, 1998.

[15] W. Y. Lin, T. P. Hong, and S. M. Liu, "On adapting migration parameters for multi-population genetic algorithms," in *Proceedings of IEEE International Conference on Systems, Man and Cybernetics (SMC '04)*, pp. 5731–5735, October 2004.

[16] L. Wang, Z. J. Weng, Y. Liang, Y. Wang, Z. Zhang, and R. H. Di, "Design and implementation of parallel lamarckian genetic algorithm for automated docking of molecules," in *Proceedings of the 10th IEEE International Conference on High Performance Computing and Communications (HPCC '08)*, pp. 689–694, September 2008.

[17] S. Kannan and R. Ganji, "Porting autodock to CUDA," in *Proceedings of IEEE Congress on Evolutionary Computation (CEC '10)*, pp. 1–8, July 2010.

[18] P. Yao, A. Dhanik, N. Marz et al., "Efficient algorithms to explore conformation spaces of flexible protein loops," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 5, no. 4, pp. 534–545, 2008.

[19] K. Tøndel, E. Anderssen, and F. Drabløs, "Protein Alpha Shape (PAS) Dock: a new gaussian-based score function suitable for docking in homology modelled protein structures," *Journal of Computer-Aided Molecular Design*, vol. 20, no. 3, pp. 131–144, 2006.

[20] B. Sadjad and Z. Zsoldos, "Toward a robust search method for the protein-drug docking problem," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 8, no. 4, pp. 1120–1133, 2011.

[21] I. Foster, *The Grid: A New Infrastructure for 21st Century Science*, John Wiley & Sons, 2003.

[22] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th utility," *Future Generation Computer Systems*, vol. 25, no. 6, pp. 599–616, 2009.

[23] G. F. Pfister, *In Search of Clusters*, Prentice-Hall, Upper Saddle River, NJ, USA, 2nd edition, 1998.

[24] G. B. Dreyer, D. M. Lambert, T. D. Meek, T. J. Carr, and T. A. Tomaszek, "Hydroxyethylene isostere inhibitors of human immunodeficiency virus-1 protease: structure-activity analysis using enzyme kinetics, X-ray crystallography, and infected T-cell assays," *Biochemistry*, vol. 31, no. 29, pp. 6646–6659, 1992.

[25] B. Veerapandian, J. B. Cooper, A. Sali, T. L. Blundell, and R. L. Rosati, "Direct observation by X-ray analysis of the tetrahedral intermediate of aspartic proteinases," *Protein Science*, vol. 1, no. 3, pp. 322–328, 1992.

[26] N. Thanki, J. K. Rao, S. I. Foundling, W. J. Howe, and J. B. Moon, "Crystal structure of a complex of HIV-1 protease with a dihydroxyethylene-containing inhibitor: comparisons with molecular modeling," *Protein Science*, vol. 1, no. 8, pp. 1061–1072, 1992.

[27] R. Bone, J. P. Vacca, P. S. Anderson, and M. K. Holloway, "X-ray crystal structure of the HIV protease complex with L-700, 417, an inhibitor with pseudo C2 symmetry," *Journal of the American Chemical Society*, vol. 113, no. 24, pp. 9382–9384, 1991.

[28] E. F. Pettersen, T. D. Goddard, C. C. Huang, and G. S. Couch, "UCSF chimera—a visualization system for exploratory research and analysis," *Journal of Computational Chemistry*, vol. 25, no. 13, pp. 1605–1612, 2004.

[29] C. W. Tsai, J. L. Chen, and C. S. Yang, "An improved LGA for proteinligand docking prediction," in *Proceedings of IEEE Congress on Evolutionary Computation*, pp. 1–6, 2012.

[30] C. W. Tsai, C. S. Yang, and M. C. Chiang, "A time efficient pattern reduction algorithm for k-means based clustering," in *Proceedings of IEEE International Conference on Systems, Man, and Cybernetics (SMC '07)*, pp. 504–509, October 2007.

[31] G. M. Morris, R. Huey, W. Lindstrom et al., "Software news and updates AutoDock4 and AutoDockTools4: automated docking with selective receptor flexibility," *Journal of Computational Chemistry*, vol. 30, no. 16, pp. 2785–2791, 2009.

[32] R. Thomsen and M. H. Christensen, "MolDock: a new technique for high-accuracy molecular docking," *Journal of Medicinal Chemistry*, vol. 49, no. 11, pp. 3315–3321, 2006.

[33] H. M. Chen, B. F. Liu, H. L. Huang, S. F. Hwang, and S. Y. Ho, "SODOCK: swarm optimization for highly flexible protein-ligand docking," *Journal of Computational Chemistry*, vol. 28, no. 2, pp. 612–623, 2007.

[34] H. Muhlenbein, "Parallel genetic algorithms, population genetics and combinatorial optimization," in *Proceedings of the Workshop on Parallel Processing: Logic, Organization, and Technology (WOPPLOT '89)*, pp. 398–406, 1991.

[35] D. Whitley, S. Rana, and R. Heckendorn, "The island model genetic algorithm: on separability, population size and convergence," *Journal of Computing and Information Technology*, vol. 7, pp. 33–48, 1999.

Hindawi Publishing Corporation
BioMed Research International
Volume 2013, Article ID 791051, 16 pages
http://dx.doi.org/10.1155/2013/791051

Research Article

Streaming Support for Data Intensive Cloud-Based Sequence Analysis

Shadi A. Issa,[1] Romeo Kienzler,[2] Mohamed El-Kalioby,[1] Peter J. Tonellato,[3] Dennis Wall,[3] Rémy Bruggmann,[4] and Mohamed Abouelhoda[1,5]

[1] *Center for Informatics Sciences, Nile University, Giza, Egypt*
[2] *IBM Innovation Center, Zurich, Switzerland*
[3] *Center for Biomedical Informatics, Harvard Medical School, Boston, MA, USA*
[4] *Department of Biology, University of Bern, Bern, Switzerland*
[5] *Systems and Biomedical Engineering Department, Faculty of Engineering, Cairo University, Giza, Egypt*

Correspondence should be addressed to Rémy Bruggmann; remy.bruggmann@biology.unibe.ch and Mohamed Abouelhoda; mabouelhoda@yahoo.com

Received 10 September 2012; Revised 26 December 2012; Accepted 17 February 2013

Academic Editor: Ming Ouyang

Cloud computing provides a promising solution to the genomics data deluge problem resulting from the advent of next-generation sequencing (NGS) technology. Based on the concepts of "resources-on-demand" and "pay-as-you-go", scientists with no or limited infrastructure can have access to scalable and cost-effective computational resources. However, the large size of NGS data causes a significant data transfer latency from the client's site to the cloud, which presents a bottleneck for using cloud computing services. In this paper, we provide a streaming-based scheme to overcome this problem, where the NGS data is processed while being transferred to the cloud. Our scheme targets the wide class of NGS data analysis tasks, where the NGS sequences can be processed independently from one another. We also provide the *elastream* package that supports the use of this scheme with individual analysis programs or with workflow systems. Experiments presented in this paper show that our solution mitigates the effect of data transfer latency and saves both time and cost of computation.

1. Introduction

Over the past few years, cloud computing has emerged as a new form of providing scalable computing resources on demand. Customers using cloud services have access to remote computational resources that can be scaled up and down and they are charged according to the time of utilization. The cloud model is appealing for many scientific applications, where large computational resources are required on an ad hoc basis for analyzing large datasets produced or collected after some experimental work. Currently, there are a number of academic as well as commercial cloud computing providers worldwide; these include Amazon Web Services (AWS) [1] (which pioneered the provision of such services), Microsoft Azure [2], IBM Smart Cloud Enterprise [3], Rackspace [4], Magellan [5], and DIAG [6], to name a few.

The bioinformatics academic community has already recognized the advantages of cloud computing since its early days and considered it as a promising solution to overcome the ever increasing genomic data volume [7–12], especially for the scientists with limited computational power. Cloud-based software tools have been developed by the academic community for the analysis of biological sequences. These include, among others, Crossbow [13], RSD-Cloud [14], Myrna [15], and CloudBurst [16]. The life science industry has moved in the same direction and started to support cloud computing as well. Interestingly, recent NGS instruments can stream the sequenced reads to the cloud infrastructure during the sequencing process (https://basespace.illumina.com/). This has the advantage that all the new sequence data become available in the cloud upon completion of the wet-lab work.

This exciting advancement in providing cloud-based bioinformatics services is however limited by the latency of copying the user's data to the computing machines in the cloud. To take one example, uploading the African human genome dataset (130 GB) takes around 37 hours with upload rate of one MB/s, while processing this dataset (as we will show in the experiment) using Bowtie [17] takes about 32 hours. That is, the data transfer time can exceed or at least be a considerable fraction of the processing time. This directly increases the overall experiment time and accordingly increases the associated costs. Current solutions to overcome this problem are mostly commercial and they only focus on the reduction of the data transfer time, by using faster data transfer protocols and compression techniques (c.f., [18], http://www.filecatalyst.com/ and http://asperasoft.com/). These solutions are however limited by the user's bandwidth and the nature of the data that is compressed and transferred. In this paper, we show that it is possible to further reduce the overall experiment time by incorporating an online data processing (streaming) scheme to process the data while it is transferred. This solution fits the wide class of NGS problems, in which the NGS sequences can be processed independently from one another; the problems of mapping NGS sequences to a reference genome or searching them in a given set of databases are examples of these problems. In the aforementioned example of the African human genome, the overall processing time using our scheme will converge to the data transfer time, as the data transfer and computation proceed in parallel. As we will show in the paper, this scheme has the extra advantage of reducing the overall cost of the experiment due to the use of fewer compute nodes.

In this paper, we present the incremental (online) data processing package *elastream* (*elastic-stream*) that has the following set of features:

(i) automatic creation and management of a computer cluster in the cloud (including MapReduce clusters), equipped with necessary NGS analysis tools,

(ii) automatic submission of jobs to the cluster and monitoring them,

(iii) incremental (online) data processing for individual tools as well as for workflow engines installed on the cloud machines, even if the tools and engines do not directly read/write to standard Unix pipes,

(iv) adaptive load balancing where the number of cluster nodes can be increased or decreased in run time in response to changes in the computation load.

To further facilitate the use of *elastream* for individual applications, we provide a client software that can be used from the user's local machine to activate the *elastream* cloud cluster and submit analysis jobs to it. Furthermore, we also provide add-on's in the form of workflows to enhance the popular workflow systems Taverna [19, 20] and Galaxy [21]. These add-on's facilitate the use of cloud computing power with the data streaming option. These add-on's are useful for the developers and users of the Taverna and Galaxy systems to scale up their resources and enhance the performance of their workflows.

This paper is organized as follows: Section 2 includes related work and a summary of the Amazon cloud computing products. In Section 3, we introduce our *elastream* package, which supports establishment and use of a cloud computing cluster. In this section, we explain the design principles as well as the implementation details of *elastream*. Section 4 introduces our on-line processing scheme for individual tools as well as for workflow systems. Section 5 introduces the features of *elastream* distribution and its add-ons. In Section 6, we present a demonstration of our scheme based on *elastream* in the Galaxy workflow system. We also evaluate the performance of the streaming solution. Finally, Section 7 includes the conclusions and future work.

2. Background and Related Work

2.1. Cloud Computing and Amazon Web Services

2.1.1. Cloud Computing Services. Cloud computing provides access to remote computing resources (processors, memory, storage, bandwidth, software, etc.), where such resources are encapsulated as services that can be metered and charged for on a pay-per-usage basis. From a service-oriented point of view, cloud computing services can be categorized as Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), or Software-as-a-Service (SaaS).

The SaaS model provides an abstraction of traditional Internet applications. A piece of software is deployed at the cloud provider's site and is accessed as a remote service. Using this model, analysis tools are deployed as accessible remote services, allowing users to access and execute these tools. Crossbow [13], (http://bowtie-bio.sourceforge.net/crossbow/ui.html), which is hosted at Amazon, is an example of these tools. In this case, users do not need to worry about the low-level issues related to resource allocation and execution of the tool; these are handled by the SaaS provider.

The PaaS model provides an abstraction of a complete development platform deployed as a service. The platform typically comprises a restricted software development environment with an associated software stack. This enables application builders to develop new programs, usually for specified classes of applications. A scientific workflow system, for example, Galaxy [21], deployed in the cloud is an example of a PaaS. In this case, again, it is the PaaS provider who handles all resource allocation decisions and low level details of workflow execution.

Within the IaaS model, a cloud service provider, such as Amazon, hosts large pools of computing machines and offers access to them via a set of APIs. Users can configure the machines as they wish (operating system, software, etc.) and then use them for executing their applications. The machines provided are commonly virtual machines (VMs) that the provider manages on behalf of the consumer. For the provider, the use of the VM abstraction supports scalability by allowing a physical machine to be shared by multiple VMs and also allowing them to bill users only for the time the VMs are running. For the user, any number of virtual machines can be allocated and configured. Moreover, SaaS and PaaS applications can be freely installed and used. In this case, the user has to handle decisions about VM configuration and software installation as well as about allocation and de-allocation of the VMs based on performance and budget requirements.

2.1.2. Amazon Web Services. Because the current version of *elastream* is based on the Amazon cloud infrastructure, we review the basic technical and financial features of this infrastructure. Our use of the Amazon platform is motivated by the fact that it is the largest and most popular one so far. Though, we would like to stress that the methods and approaches presented in this paper are applicable to any cloud computing platforms and are not specific to Amazon; it is planned that future versions of *elastream* will support more platforms.

Amazon Web Services (AWS) of Amazon offers infrastructure as a service (IaaS) in terms of computational power (CPUs and RAM), storage, and connectivity. AWS offers a variety of machine instance types that range in computing power and cost. Table 1 summarizes the features of some instance types including the strongest ones. With each of these types, mounted disks (called *ephemeral* disks) are also provided. Machine instances are created from Amazon Machine Images (AMIs), which are templates containing software configurations (e.g., operating system, application server, and applications). AWS includes a directory of AMIs prepared by the AWS team and by the community. The deposited AMIs in this directory have different operating systems and are equipped with different applications.

Because the ephemeral disks are volatile and vanish with the termination of the virtual machine, AWS offers two types of persistent storage: EBS and S3. The former is defined in terms of volumes, where one or more EBS volumes can be attached (mounted) to a running virtual machine instance, similar to a USB thumb drive (volume size ranges between 1 GB and 1 TB). The latter is like a data center providing data hosting, accessed through certain methods (basically POST and GET methods).

The AWS business model is "pay-as-you-go," where the user is charged only when own machines are running. The user is also charged for reserved storage on Amazon and for data transfer out of the AWS site and from/to persistent storage solutions. Table 1 summarizes the storage options and their prices in AWS (last price update November 2012). For more information about the AWS pricing schemes, we refer the readers to the documentation available in the AWS website [1].

2.2. Related Work

2.2.1. Cloud-Based Solutions for Sequence Analysis. Currently, there are some cloud-based programs for the analysis of next-generation sequencing data. These include, among others, Crossbow [13], RSD-Cloud [14], Myrna [15], and CloudBurst [16]. In addition, there are some libraries and packages that support the creation and management of computer clusters in the cloud. To the best of our knowledge, these include so far StarCluster [22], Vappio [23], and CloudMan [24]. StarCluster [22] has been developed as a general cluster management solution for AWS and it is not specific to bioinformatics applications or any bioinformatics use cases. CloudMan [24] has been developed as part of the Galaxy project to basically provide a version of the Galaxy workflow system [21, 25] in the cloud. Vappio [23], unlike CloudMan, is a standalone library for supporting the creation of a computer cluster in the cloud. It enables submission of remote jobs to the cloud instances. These solutions assume that the data should be available in the cloud before any processing takes place. Our work in this paper can be used to enhance these solutions with incremental processing features.

2.2.2. Online Data Processing. Online data processing (also referred to as stream or incremental data processing) has been addressed since the early days of distributed computing, especially in the area of distributed database systems. Specifically, pipelined query evaluation models have been introduced to hide data transfer latencies within the processing of the queries [26]. The same approach can be readily used in other applications and over a computer cluster empowered by a job-scheduler (e.g., PBS) to manage job submissions. Online processing with the MapReduce framework is relatively new and has just appeared in [27, 28]. The involved approach in these papers is based on modifying the MapReduce implementation and providing a stream-based data processing system underneath. The problem of these solutions is that they have not yet been supported by the Amazon MapReduce product. In this paper, we will overcome this limitation by following a different approach based on the elasticity property of the cloud model. But once it is supported by Amazon, we will enhance our package with this feature to further serve the bioinformatics community.

In preliminary work [29, 30], we evaluated the incremental data processing approach for certain bioinformatics tools like SHRiMP [31] and Bowtie [17] based on an industrial streaming engine (IBM InfoShphere Streams). In this paper, we extend this work in several directions: first, we introduce a scheme that supports stream processing in a generic way with no dependencies on any streaming engine. Second, our scheme is applicable not only to specific software tools but also to workflow systems like Galaxy [21] and Taverna [19, 20]. Finally, our work supports incremental processing over the Elastic MapReduce framework of AWS.

3. *Elastream*: Design and Implementation

3.1. Block Diagram. Elastream is a software package composed of a set of modules for constructing a computer cluster in the cloud and executing analysis jobs on it. Figure 1 shows the block diagram of *elastream*. As shown in Figure 1, the package is composed of three basic modules: *cloud cluster creation module*, *cloud cluster runtime module*, and *job module*.

Table 1: Amazon services: virtual machines, storage, data transfer, and disk access. This information is for the Amazon US site. Prices for other sites are available on the AWS website.

Resource type	AWS service	Service unit	CPUs (#(GHz))	Memory (GB)	Cost ($/Hr)
Computation	EC2	m1.large	2 (2)	7.5	0.32
		m1.xlarge	4 (2)	15	0.64
		c1.xlarge	8 (2.5)	7	0.66
		m2.4xlarge	8 (3.25)	68.4	1.80
		cc1.4xlarge	8 (4.19)	23	1.30
Resource type	**AWS service**	**Service unit**	**Size**	**Tiers**	**Cost ($/GB/month)**
Storage	S3	Bucket	Unlimited	1st 1 TB	0.14
	S3	Bucket	Unlimited	Next 450 TB	0.1
	S3	Bucket	Unlimited	Next 4000 TB	0.08
	EBS	Volume	Up to 1 TB		0.10
Resource type	**AWS service**	**Service unit**	**Type**	**Size**	**Cost ($/GB/month)**
Data transfer	S3	I/O	Data IN/within AWS	Any	0.00
	S3	I/O	Data OUT	1st 1 GB	0.00
	S3	I/O	Data OUT	Next 10 TB	0.12
	S3	I/O	Data OUT	Next 100 TB	0.07
	S3	I/O	Data OUT	Next 150+ TB	0.05
Disk access	S3	API	GET, PUT, POST	1 K request	0.01
	S3	API	COPY, LIST	1 K request	0.01
	EBS	I/O	R/W	1 M request	0.1

3.1.1. Cloud Cluster Creation Module. This module includes functions for creation of the cluster in the cloud. These functions can be categorized into three submodules.

The first submodule includes functions for setting up the master node of the cluster. The master node is created from the *elastream* virtual machine image, which we have already prepared and deposited in AWS as AMI. This virtual image includes the Linux operating system and all necessary software libraries and packages. It also includes the whole *elastream* package. (Detailed description of the image is given in Section 5.) This sub-module is based on invoking certain APIs provided by AWS. The activation of the cluster node includes some built in bootstrapping scripts that conduct necessary configuration steps (e.g., SSH key settings) and installation of some important libraries and packages.

The second submodule includes functions for creating the worker nodes and associating them to each other and to the master node. The worker nodes are created from the same *elastream* virtual machine image used for creating the master node. This sub-module also includes installation and configuration of the job scheduler (PBS Torque is the default job scheduler) over the created worker nodes.

The third sub-module includes functions for creating the EBS volumes and attaching them to the cluster nodes. It also includes functions for connecting the nodes to the S3 storage to save the result data. There are also functions to establish shared storage among the cluster nodes using NFS or through S3 using S3fs so that the input data becomes available to every job running at any node.

3.1.2. Cloud Cluster Run-Time Module. This module includes functions responsible for checking the cluster status and terminating the cluster. It also includes functions for adding more nodes and attaching more EBS volumes and S3 buckets to the cluster nodes.

3.1.3. Job Module. This module includes functions responsible for submitting jobs to the cluster from a remote machine. It also includes functions for checking job status and redirecting the results to certain directories or to the persistent S3 storage.

3.2. Use Case Scenario. Figure 2 shows the basic use case scenario, in which the functions of *elastream* are used to create a computer cluster in the cloud from remote user's machine and to submit analysis jobs to it. As mentioned before, the cluster is created from the specific *elastream* virtual machine image we have already prepared and deposited in AWS. The first step in this use case scenario is that the user installs the client program of *elastream* from its website. This client program invokes the cluster creation module using the user's credentials so that the created computer cluster is associated with the user's account in AWS. The creation procedure includes the following steps.

First, the function for creating the master node from the *elastream* machine image is invoked. Once the master node is created, a job request is sent to it to execute a program in the master node that creates other worker nodes. This job request includes invocation of the node creation function to create the specified number of worker nodes. Technically, the node creation function is the same as the one used for creating the master node. The only difference is that it already has the credential information, which is reused automatically. We would like to stress that the image of any created machine

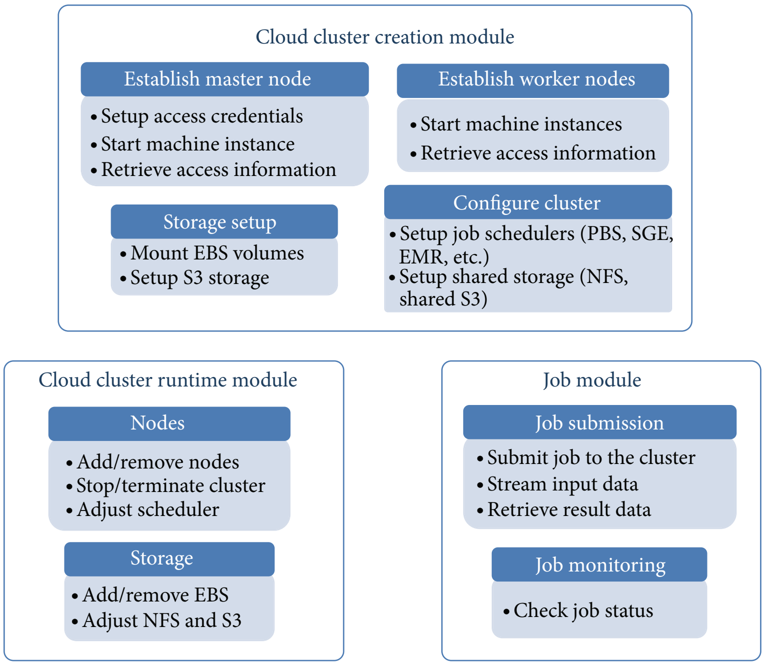


FIGURE 1: *Elastream* block diagram. *Elastream* is composed of three modules. Each module includes submodules conducting certain tasks.

includes the *elastream* package with all its functions that can be directly used once the machine is activated. After the creation of all worker nodes, another job request is sent to the master node to configure the cluster and the job scheduler. This job invokes a certain script in the cloud cluster to accomplish this task. Once the configuration tasks have been successfully completed, the cluster is ready to run any analysis job.

Running an analysis job can be achieved by using the client program (1) to execute a command line of the analysis tool, (2) to specify the input, and (3) to specify the destination directory of the output. Note that the command line itself can specify that the analysis task runs through the installed job scheduler. Note also that the job should invoke a program already installed in the *elastream* machine. (Note that all *elastream* programs are accessible once the cluster starts.) It is important to mention that this mode of operation does not prevent the user from accessing and utilizing the cluster, using for example the SSH program. The user manual and source code of the *elastream* functions are available on the package website.

3.3. Major Implementation Details

3.3.1. Elastream Virtual Machine Image. To facilitate the use of *elastream*, we prepared a virtual machine image deposited at Amazon public pages. (The package website includes details about this image and its ID in AWS.) The *elastream* image is based on Ubuntu Linux and it is equipped with a number of software packages, including Amazon Command Line Tools (the APIs of Amazon), PBS Torque as a job scheduler, NFS as a shared file system, s3fs [32] to handle the S3 as a shared file system, Python/Perl interpreters, MPICH2, and C/C++ and JRE. The image comprises a large library of sequence analysis software tools, summarized in the next section. It also contains the ready-to-use *elastream* modules presented above. Furthermore, it includes a server module and a client module to facilitate communication between the nodes as described below.

3.3.2. Client-Server Software Pattern. To facilitate the communication between the local machine and the master node at one side and between the master node and other worker nodes on the other side, we used a client-server software pattern. We developed a server module and preinstalled it in the machine image. This server module starts automatically whenever the respective machine is activated from its image; this includes the master node as well as any worker node. (We use operating system features to enable creation and automatic startup of the server; see the manual for more details.) The server module listens to certain ports, identifies the incoming messages from the client, maps them to one of the functions in the modules discussed above, and executes them. The client connects to the server through the specified port and invokes one of the server functions. Note that the *elastream* machine image includes a copy of the client program so that its functionalities are used inside the cloud to create extra nodes and to submit specific jobs to all or certain worker nodes. The server and client are written in Python, and both of them use APIs of AWS to handle all cloud-related functions. They also use (shell) scripts we developed to configure the cluster and the associated job scheduler. For remote job submissions (from the client program to the cloud cluster), we have developed and used an asynchronous

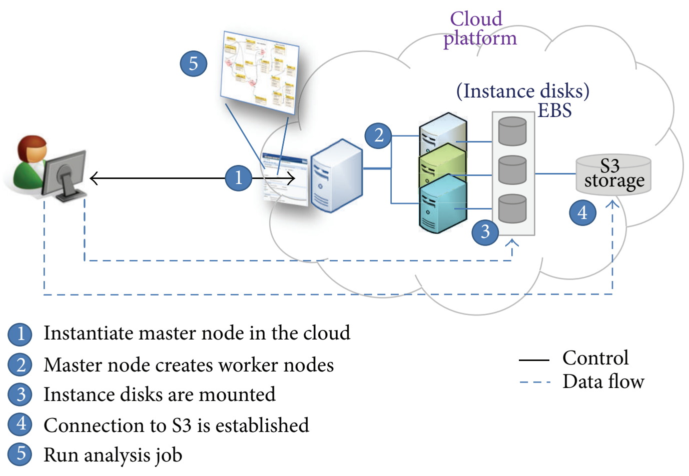

Cloud platform
(Instance disks) EBS
S3 storage
1 Instantiate master node in the cloud
2 Master node creates worker nodes
3 Instance disks are mounted
4 Connection to S3 is established
5 Run analysis job
Control
Data flow

Figure 2: Use case scenario based on the functions of *elastream*. The user uses the client program from own local machine to establish and use a computer cluster in the cloud.

protocol. This protocol is similar to the RESTful protocol and it is implemented in Python.

3.3.3. Establishment of MapReduce Clusters. MapReduce [33] is a programming model and an execution framework that facilitates the processing of large amounts of data on a computer cluster. Amazon offers a product for MapReduce called *Elastic MapReduce* (*EMR*), based on the open source Hadoop implementation of MapReduce.

Compared to job schedulers, the MapReduce model is more complex, as it requires that the analysis task is formulated in terms of a *Map* and a *Reduce* functions. The former function processes the input items in parallel and emits the results as well as some *key-value* pairs. The *Reduce* function uses these pairs to postprocess the output of the *Map* function in parallel. Considerable programming experience is usually needed so as to fit the structure of computation at hand in terms of Map-Reduce functions. Moreover, not all problems can be formulated in the MapReduce model. Nevertheless, the advantage of using the EMR product lies in its lower machine price compared to traditional nodes of the same type (e.g., one c1.xlarge instance costs $0.66 when used in traditional cluster and costs just $0.12 when used in EMR). These reduced costs make it appealing to use the EMR for NGS data processing. Furthermore, various bioinformatics programs are already based on the MapReduce framework and are demonstrated to work using the EMR product. Examples of these tools include Crossbow [13], RSD-Cloud [14], Myrna [15], and CloudBurst [34].

The client program of *elastream* can create an EMR cluster using the APIs of AWS from the user's local machine. The creation steps are similar to that of the traditional cluster but there are some differences due to the MapReduce model and Hadoop implementation. The EMR cluster cannot be created from a user's own image, such as the *elastream* image we prepared. It can only be built from specific EMR images previously created by the Amazon team. The EMR image contains the basic Hadoop code and basic programming languages (Java and Python), but it does not include any analysis software. Therefore, the required analysis programs should be installed using a bootstrap script specified in the creation function. Note that the bootstrap script is executed before the Hadoop system starts, and it can be generally used for any necessary (initialization) tasks related to the required analysis. (The *elastream* manual includes an example of this bootstrap script).

Analysis jobs can be directly submitted, once the cluster is created and Hadoop system starts. The analysis job is specified using a distinct *elastream* command, and it should include the path to the input data as well as the *Map* and *Reduce* functions implemented either in Java or Python. The *elastream* composes a Hadoop job using these items and executes it on the EMR cluster.

3.3.4. Stream Processing Support. To support online data processing, the job submission/execution method of the *elastream* has to be extended with an additional layer. The following section includes the underlying scheme and the implementation details of this layer. In that section, we will discuss this scheme with traditional and EMR clusters. We will also discuss how it can be used within workflow systems.

4. Online Sequence Processing

In this part, we describe our method to support on-line sequence processing for both individual analysis tools and workflow systems. Our method does not depend on any streaming engine and does not require that the involved tools or systems are able to read and write to the standard Unix pipes.

4.1. Supporting Individual Tools. To support incremental data processing in the cloud, we developed the software design pattern shown in Figure 3. This pattern works only

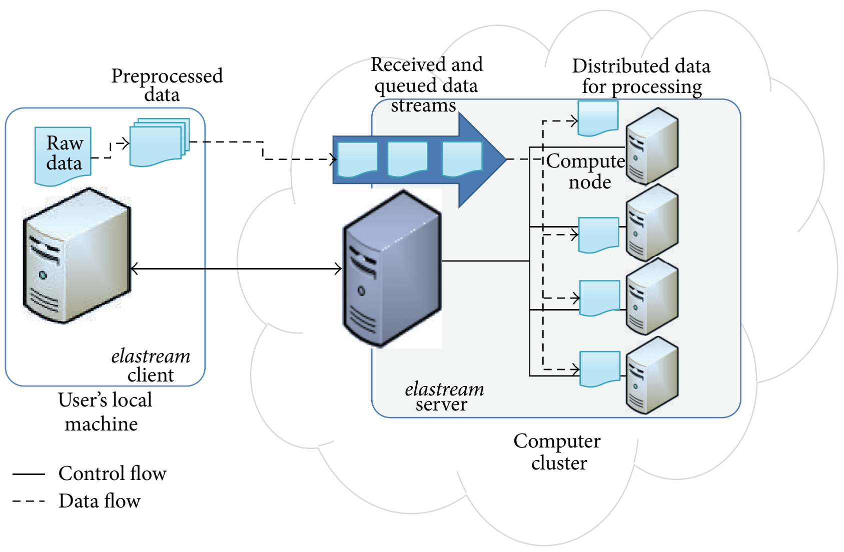


Figure 3: Software design pattern to support incremental data processing in the cloud. The client streams the data into the cloud cluster. The server monitors the received data buckets and manages the launch of analysis jobs on cluster nodes. After completion, the output data is either transferred back to the client machine or transferred to the user's S3 account.

for data intensive tasks in which input sequences can be processed independently from one another. Examples of such problems include, among others, mapping NGS sequences to a reference genome and searching sequences in a given set of databases. In this pattern, a local machine streams the data to a cloud cluster and an analysis program already available in that cluster will begin with the processing as soon as the data arrive. To achieve this, there is a server (we call it *streaming server*) installed in the cloud machine and a client program (we call it *streaming client*) at the local machine. The streaming client communicates with the streaming server to start a job in the cloud. The job submission in this pattern includes (1) sending the command line specifying the respective program call and (2) the transfer of the data from the local machine to the cloud machine. Note that the data transfer issue is in sharp what distinguishes this method of job submission from the previously described offline (nonstreaming based) one, which requires that the input data is completely uploaded to the cloud before starting the analysis. While the data is being transferred, the streaming server monitors the incoming data stream, parses it into sequences, and accumulates the sequences in buckets. When a bucket is complete, the analysis tool is invoked to process the bucket at hand. If the data transfer rate is so high that many buckets are ready at one time point, then more jobs are launched in parallel to process the buckets. After completion, the output data can be downloaded to the user's local machine or exported to an S3 account.

The client module of this design pattern can do more than establishing a connection to the server and transferring data to it. Actually, it can preprocess the data before sending it to further speed up data transfer and reduce the server side work. This pre-processing includes partitioning the data into chunks and compressing them. In this case, the server expects to receive chunks and it just forward them to the next processing steps.

This design pattern is implemented in *elastream* by extending the functionality of both the server and the client programs. The *elastream* server is extended by two extra threads per job. The first thread is for receiving the data stream, and the second is for monitoring the incoming data and constructing the buckets. The latter thread is also responsible for submitting the jobs to process the completed buckets in a pipelined fashion; that is, a just completed bucket can be directly processed even if the previous buckets are still being processed. The client program can be extended by dividing the data into buckets and sending them in sequence.

4.2. Streaming for MapReduce-Based Applications. The *Elastic MapReduce* (EMR) product of AWS does not support incremental data processing and assumes that all the data is available in the cloud in advance. To overcome this limitation, we use the same scheme in which the input data is divided into buckets and these are processed independently. We also make use of the elasticity property of the cloud to expand the cluster when needed. The details are as follows.

Elastream provides a programmatic means for creating and using the *Elastic MapReduce* (EMR) product of AWS. Therefore, an initial EMR cluster is first constructed when an analysis job is submitted. The *streaming server* monitors the received data and accumulates them into buckets. Once a bucket is complete, a Hadoop job is submitted to process this bucket over the EMR. This solution looks fine, but its scalability is in fact limited due to the following reason. EMR is offline in the sense that new buckets cannot join the parallel processing of the currently running job even if

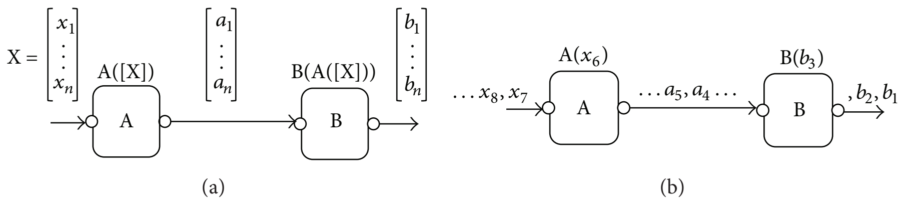


FIGURE 4: Streaming/pipelining in workflow systems: (a) no pipelining, where A starts computation only when all its input items $[x_1 \cdots x_n]$ are available, and so does B which processes the output items of A. (b) Pipelining, where A starts computation when any data item is available, and so does B. Note that in this mode A and B run concurrently on different data items.

there are enough resources, and the new buckets have to be queued to be processed one after another. To overcome this limitation, we exploit the elasticity property of the cloud and automatically create another EMR cluster to process the pending buckets. This elastic creation of EMR clusters enables processing of the buckets with minimal queuing time.

4.3. Supporting Workflow Systems. Bioinformatics workflows include the use of multiple software tools and data resources in a staged fashion, with the output of one tool being passed as input to the next. Workflow systems have been introduced to facilitate the design and execution of sophisticated workflows. Examples of the systems that support sequence analysis applications include, among others, Taverna [19, 20], Kepler [35], Triana [36, 37], Galaxy [21], Conveyor [34] Pegasus [38], and Pegasys [39]. In these systems, the workflows are represented in the form of a directed graph, where nodes represent tasks to be executed and edges represent either data flow or execution dependencies between different tasks. The workflow system maps the edges and nodes in the graph to real data and software components. The workflow engine (also called execution or enactment engine) executes the software components either locally on the user's machine or remotely at distributed locations. The engine takes care of data transfer between the nodes and can also exploit the use of high-performance computing architectures so that independent tasks run in parallel. This allows the application scientists to focus on the logic of their applications without worrying about the technical details of invoking the software components or using distributed computing resources.

There are two classes of workflow engines: one that supports stream processing (also referred to as *pipelining* in workflow literature) and others that do not. For example, the engines of Kepler [35] and Taverna [19, 20] belong to the first class, while the engines of Galaxy and Conveyor [34] belong to the second one. The idea of pipelining in workflow engines is illustrated in Figure 4. In engines that support pipelining, the output of task A is a list of items $[a_1, a_2, \ldots, a_n]$ and task B can start processing whenever an item a_i is produced. That is, tasks A and B can run concurrently in such workflow systems. In engines not supporting pipelining, task A must finish computation over all list items $[a_1, \ldots, a_n]$, before B starts processing.

Changing a workflow system to support pipelining requires some modification of the workflow engine itself, which is a difficult task. Still, this modification is not sufficient per se to achieve efficient online processing during data transfer. This is because some tools within the workflow may not support pipelining at all, which would lead to blocking at some stage of the workflow. To overcome these issues and to support on-line data processing, even for those workflow engines that do not support pipelining, we suggest the following strategy.

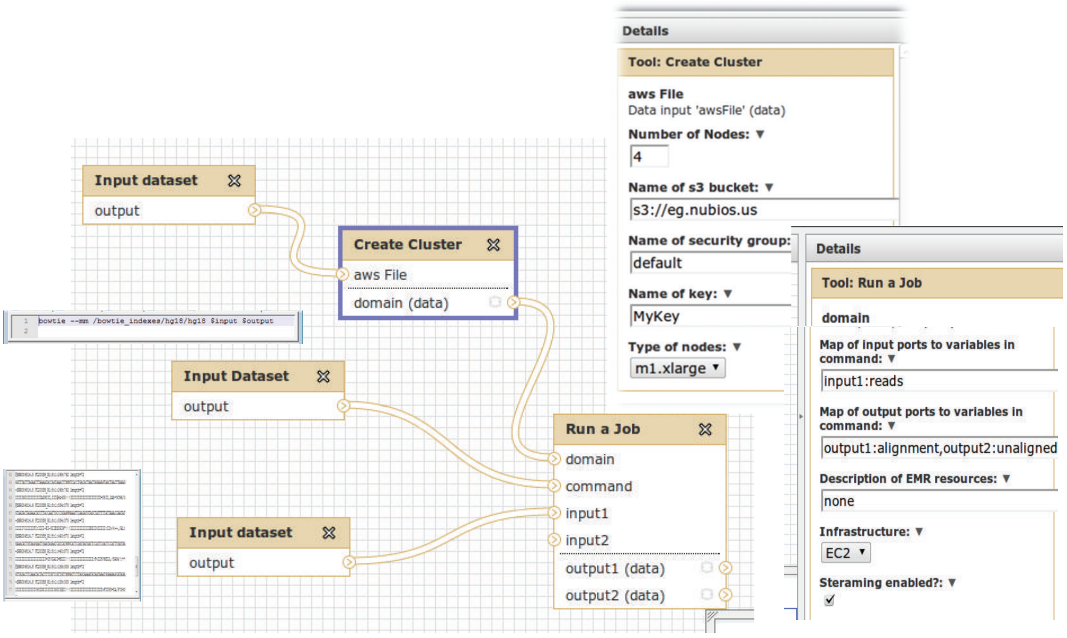
We handle the whole workflow system as a usual program that can be invoked from the command line to execute a given workflow over certain input data. In the streaming mode, the streaming server monitors the incoming data items to establish sequence buckets. Once a bucket is ready, the workflow engine is invoked to process this bucket. If multiple buckets are available, multiple instances of the workflow engine are invoked in parallel to process these buckets. This solution permits stream processing even if the workflow engine does not directly support this mode of execution and even if the tools within the workflow do not support pipelining.

5. *Elastream* Distribution

5.1. Basic Components. The *elastream* distribution includes the package executables, the source code, and the client program to be used from a local machine to invoke package functionality in the cloud. The distribution also includes the following additional features that further facilitate its use for bioinformatics applications.

(i) We prepared a virtual machine image (AMI) deposited at the AWS machine image directory. The *elastream* package website includes details about this image and its identifier in AWS. This virtual machine image can be used to create a computer cluster from the AWS interface or from our client program installed in the local machine. The *elastream* image includes a set of preinstalled tools that can be directly accessed upon the creation of the cluster. In the current version of *elastream*, there are about 200 tools, coming from BioLinux, EMBOSS [40], SAMtools [41], fastx [42], NCBI BLAST Toolkit [43–45], and other individual sequence analysis programs. Addition of extra tools and updating this image is explained in the *elastream* manual.

(ii) To save costs and to facilitate usage of database-dependent programs, we prepared snapshots of different biological databases and indexes, including the



```
bowtie --mm /bowtie_indexes/hg18/hg18 $input $output
```

Tool: Create Cluster

aws File
Data input 'awsFile' (data)

Number of Nodes: 4

Name of s3 bucket: s3://eg.nubios.us

Name of security group: default

Name of key: MyKey

Type of nodes: m1.xlarge

Tool: Run a Job

domain

Map of input ports to variables in command: input1:reads

Map of output ports to variables in command: output1:alignment,output2:unaligned

Description of EMR resources: none

Infrastructure: EC2

Steraming enabled?: ✓

Figure 5: The use of *elastream* in Galaxy in the form of a workflow that establishes the cloud machines and submit jobs. The key nodes of this workflow are the Create Cluster and Run a job. The parameters associated with these nodes are set in the right pane of the web-based interface. We show a part of the file including a command line and a part of another file including NGS sequences; these files are referenced to in the nodes "Input dataset" connected to the "Run a Job."

NCBI nucleotide and protein databases in the form of raw and formatted sequences; the raw human genome sequence, and precomputed indexes of it for Bowtie [17]. These snapshots are made available to the user free of charge through a simple interface, to create EBS volumes and mount them to the cluster.

5.2. Elastream in Workflow Systems. We have developed add-on's (as subworkflows) to support the popular workflow systems Taverna and Galaxy with cloud computing power enhanced with the data streaming option. These add-on's are available as part of *elastream* distribution.

For Galaxy, we have created a workflow that enables streaming based on *elastream*. This workflow, which is shown in Figure 5, is composed of the following. There are two major nodes: "Create Cluster" and "Run a Job." The former is responsible for creating a cluster and the latter is responsible for submitting a job. These two nodes use the functionalities of the *elastream* client program, which is installed within the Galaxy tool set. The *elastream* website includes a link to a Galaxy system with this workflow already built in. The website includes also information for Galaxy administrators on how to integrate this workflow in their systems. We also like to attract the attention that these workflow nodes can be generically used in other user created workflows.

The number and type of machines for the "Create Cluster" node are specified in an adjacent editing area on the right side of the GUI. The two nodes titled "Input Dataset" connected to "Create Cluster" node specify the credential files (certificate and private key files) required to associated the created cluster with the user's account. (Note that input node names cannot be changed in Galaxy).

The "Run a Job" node is responsible for (1) transferring the data from the local user's machine to the cloud machines, (2) invoking a tool (or a system) already installed in the cloud machine, and (3) transferring the result data to the local machines or the S3 cloud storage. The parameters for the "Run a Job" node are also specified in an adjacent editing area on the right-hand side of the GUI. The parameters mainly associate the input files to variable names in the command line and specify if streaming is enabled or not. The input node connected to the command port of the node "Run a Job" specifies the command line of the analysis program in the cloud. The input node connected to the input1 port specifies the input data. The input to this workflow is a set of sequences to be processed in the cloud. The "Run a Job" node includes an optional process to split the input data into chunks before sending them to the cloud.

For Taverna, the workflow that enables streaming is composed of two Taverna sub-workflows for management of the cloud cluster and submission of jobs in streaming mode. This workflow can be generically used as sub-workflows in other user created workflows running on Taverna. Figure 6 shows these two Taverna workflows. The first workflow (called *CreateCluster*) establishes a computer cluster in the cloud. The blue nodes (rectangles) in this workflow define the parameters for creating the cluster. The node titled nodes specifies the number and type of nodes, the node path contains the working directory in the cloud machine, the

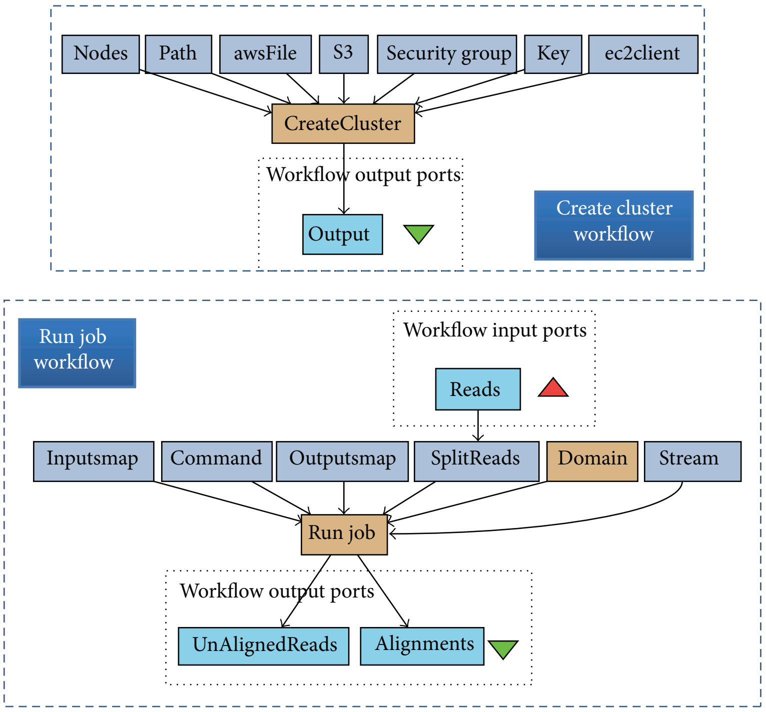


FIGURE 6: The use of *elastream* with Taverna in the form of a workflow that establishes the cloud machines and submit jobs. The key nodes of this workflow are the Create Cluster and run Job nodes. The former is responsible for creating computer cluster in the cloud. The latter workflow submits jobs to the remote cloud machines.

node awsFile includes the Amazon access credentials, the node S3 states the name of S3 bucket, the nodes securityGroup and key specify the security groups, and the node ec2client defines the path to the *elastream* client tools on the machine where Taverna runs. (Note that the parameters of a node in Taverna are specified through other direct predecessor nodes and not through edit boxes in GUI as in Galaxy.)

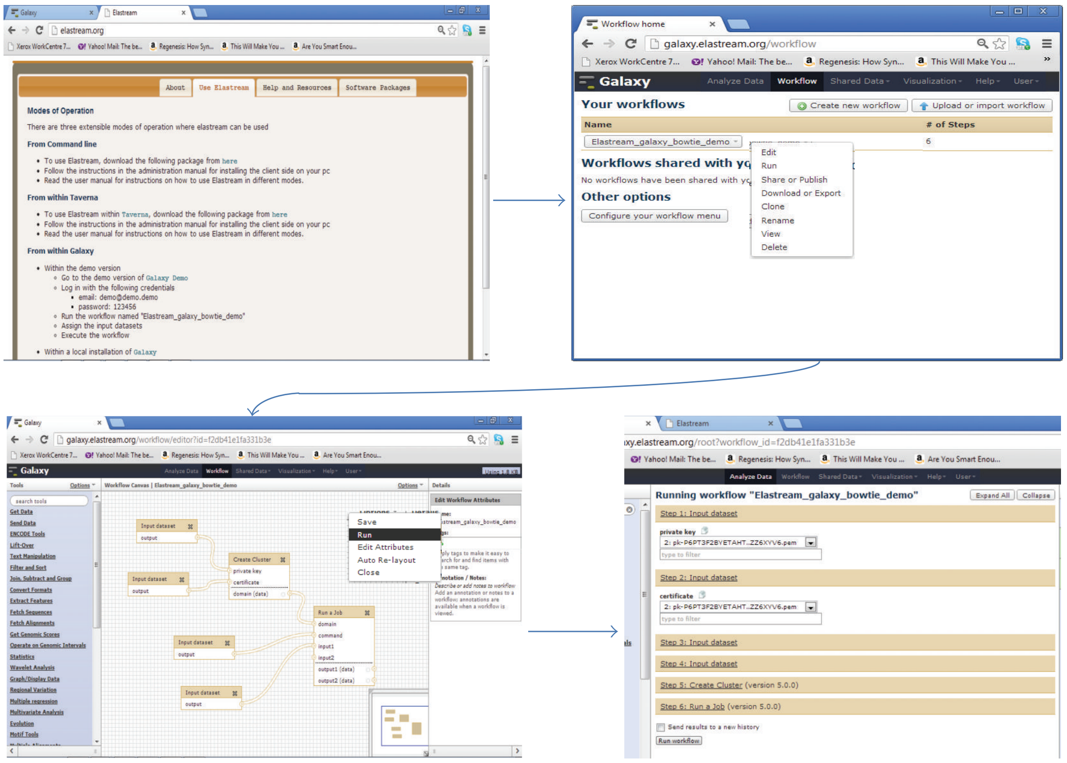
The second workflow (called *runJob*) is responsible for (1) transferring the data from the local user's machine to the cloud machines, (2) invoking a tool (or a system) already installed in the cloud machine, and (3) sending the result data to the local machines or cloud S3. The input to this workflow is a set of sequences to be processed in the cloud. The blue nodes specify the parameters of this workflow: The node titled stream specifies the streaming option, the node command includes the command line to be invoked, and the nodes Inputsmap and OutputsMap map the input and output ports to the command (see the manual for more details). If output ports are not specified, then the result data will not be transferred back to the local machine and remain in the cloud. This workflow includes an optional process to split the input data into chunks before sending them to the cloud.

6. Demonstrations and Experiments

In this section, we demonstrate the use of our streaming scheme *elastream* and evaluate its performance. In the following subsection, we demonstrate the use of the Galaxy workflow that supports streaming as discussed in Section 5.2. In the subsequent sub-section, we evaluate the performance of our solution using traditional and EMR computer clusters.

6.1. Streaming Cloud-Based Workflows of Galaxy. Figure 7 shows the sequence of steps for using the Galaxy workflows, which we have provided to support the use of cloud computing enhanced with the streaming option within Galaxy. From the *elastream* website, the user can access the Galaxy workflow system, which is running on our local infrastructure. Our workflows that support cloud usage with streaming are accessed by selecting the elastream_demo from the workflows drop-down menu. The workflow editing page shows the workflow nodes, where the user can specify the number and type of the cluster machines. If anyone decides to execute this workflow, one will be forwarded to the execution and input page to enter the paths to the credential file, security file, command line file, and input NGS data. For this demo, we have already provided example files for these input types. The example command line includes invocation of the Bowtie program [17] with a set of NGS sequences as input. We stress that this workflow can be used in other Galaxy workflows or with tools other than Bowtie.

6.2. Evaluating Performance. We compare two use case scenarios: in the first, the data is processed after it is transferred completely to the cloud (i.e., without on-line processing). In the second, the data is processed while it is transferred (i.e., with on-line processing).



Modes of Operation

There are three extensible modes of operation where elastream can be used

From Command line

- To use Elastream, download the following package from here
- Follow the instructions in the administration manual for installing the client side on your pc
- Read the user manual for instructions on how to use Elastream in different modes.

From within Taverna

- To use Elastream within Taverna, download the following package from here
- Follow the instructions in the administration manual for installing the client side on your pc
- Read the user manual for instructions on how to use Elastream in different modes.

From within Galaxy

- Within the demo version
  - Go to the demo version of Galaxy Demo
  - Log in with the following credentials
    - email: demo@demo.demo
    - password: 123456
  - Run the workflow named "Elastream_galaxy_bowtie_demo"
  - Assign the input datasets
  - Execute the workflow
- Within a local installation of Galaxy

Running workflow "Elastream_galaxy_bowtie_demo"

Step 1: Input dataset

private key: 2: pk-P6PT3F2BYETAHT..ZZ6XYV6.pem

Step 2: Input dataset

certificate: 2: pk-P6PT3F2BYETAHT..ZZ6XYV6.pem

Step 3: Input dataset

Step 4: Input dataset

Step 5: Create Cluster (version 5.0.0)

Step 6: Run a Job (version 5.0.0)

Send results to a new history

Figure 7: The Galaxy workflow system supported with cloud computing functionalities and streaming option. The upper left screen shot is the *elastream* page, from which the user can access the Galaxy workflow system installed on the *elastream* infrastructure. The upper right screen shot is the Galaxy page, where the user selects the *elastream* workflow. The lower left is the *elastream* workflow, and the lower right is the input/execution page, where the user specifies the paths of the input files including command line, credentials, and input data.

Table 2: Description of the used datasets. All the datasets are NGS sequences (reads). The third column includes the number of sequences in millions, and the third column is the data size in GB. There are four sets of the African human genomes of different sizes. The final dataset (130 G) is the original complete one. The three previous ones (1 G, 10 G, 40 G) are subsets of it.

Description	Source	No_Seq	Size
E. coli genome	[13]	≈9 million	≈1.4 GB
1 G African human genome	http://trace.ddbj.nig.ac.jp/dra/index_e.shtml (study SRP000239)	≈7 million	≈1 GB
10 G African human genome	http://trace.ddbj.nig.ac.jp/dra/index_e.shtml (study SRP000239)	≈72 million	≈10 GB
40 G African human genome	http://trace.ddbj.nig.ac.jp/dra/index_e.shtml (study SRP000239)	≈437 million	≈40
130 G African human genome	http://trace.ddbj.nig.ac.jp/dra/index_e.shtml (study SRP000239)	≈1419 million	≈130 GB

The data intensive task we used in our experiments is the NGS read mapping, where millions of reads have to be aligned to a reference genome. On the traditional cluster, we used the popular program Bowtie [17] as an example tool that performs this task. On the MapReduce cluster, we used the popular Crossbow [13] program. Crossbow is a MapReduce based version of Bowtie implemented using Hadoop and it is enhanced with more functions for SNP detection using soapSNP.

The parameters of these experiments are the data size, the upload speed, and the size of the computer cluster. We used datasets of different sizes, as described in Table 2. The upload speed has been controlled using the program Trickle [46]. To reduce the effects of location, time-of-the-day, and congestion on data transfer, we ran the whole experiments within the Amazon cloud environment. That is, we created one machine to represent the user's machine. This machine is equipped with the Trickle program and the *elastream* client program. The bandwidth we observed within the Amazon site is approximately 20 MB/s to 30 MB/s, which is large enough for our experiments.

Table 3 shows the runtimes for executing Bowtie in streaming and non-streaming modes using computer clusters of variable size. Each cluster node is of the type c1.xlarge and is composed of 8 cores. Each row in the table shows the upload speed, the upload time, and the computation time for

Table 3: Running times in minutes for mapping NGS reads to a reference genome using Bowtie based on the use of traditional computer cluster. The column titled upload_speed specifies the upload speed. The column titled "upload" includes the time in minutes for uploading the data to the cloud with the respective upload speed. The column titled "compTime" includes the computation time in minutes of the whole dataset after being uploaded to the cloud. The column titled totalTimeS includes the experiment time in streaming mode and the column titled totalTimeT includes the time in nonstreaming mode, where all the data is first transferred and then processed. The numbers in brackets in this column are the respective monetary cost.

Upload_speed	Read size	Nodes	Upload	CompTime	TotalTimeT	TotalTimeS
			E. coli reads			
250 KB/s	1.4 G	1	100	3	103 ($1.32)	102 ($1.32)
250 KB/s	1.4 G	2	100	3	103 ($1.98)	102 ($2.64)
250 KB/s	1.4 G	4	100	3	103 ($3.3)	102 ($5.28)
			1 GB human reads			
250 KB/s	1 G	1	71	17	88 ($1.32)	72 ($1.32)
250 KB/s	1 G	2	71	11	82 ($1.98)	72 ($2.64)
250 KB/s	1 G	4	71	7	73 ($3.3)	72 ($5.28)
			10 GB human reads			
250 KB/s	10 G	1	800 (13.3 h)	220	1021 ($11.88)	832 ($9.24)
250 KB/s	10 G	2	800 (13.3 h)	130	930 ($12.54)	818 ($9.24)
250 KB/s	10 G	4	800 (13.3 h)	60	860 ($11.88)	818 ($9.24)
			E. coli reads			
1 MB/s	1.4 G	1	25	3	28 ($0.66)	27 ($0.66)
1 MB/s	1.4 G	2	25	3	28 ($1.32)	27 ($1.32)
1 MB/s	1.4 G	4	25	3	28 ($2.64)	27 ($2.64)
			1 GB human reads			
1 MB/s	1 G	1	18	17	35 ($0.66)	21 ($0.66)
1 MB/s	1 G	2	18	11	29 ($1.32)	21 ($1.32)
1 MB/s	1 G	4	18	7	25 ($2.64)	21 ($2.64)
			10 GB human reads			
1 MB/s	10 G	1	200	220	421 ($5.28)	231 ($2.64)
1 MB/s	10 G	2	200	130	330 ($5.94)	215 ($5.28)
1 MB/s	10 G	4	200	60	261 ($5.28)	215 ($10.56)
			40 GB human reads			
1 MB/s	40 G	1	690	590	1280 ($14.52)	1100 ($12.54)
1 MB/s	40 G	2	690	325	1015 ($15.18)	695 ($15.84)
1 MB/s	40 G	4	690	180	870 ($15.84)	695 ($31.68)
			130 GB human reads			
1 MB/s	130 G	1	2220	1720	3940 ($43.56)	3600 ($39.6)
1 MB/s	130 G	2	2220	940	3160 ($45.54)	2400 ($52.8)
1 MB/s	130 G	4	2220	520	2740 ($48.18)	2400 ($105.6)
1 MB/s	130 G	8	2220	284	2504 ($50.82)	2400 ($211.2)
			E. coli reads			
10 MB/s	1.4 G	1	2.5	3	5.5 ($0.66)	5 ($0.66)
10 MB/s	1.4 G	2	2.5	3	5.5 ($1.32)	5 ($1.32)
10 MB/s	1.4 G	4	2.5	3	5.5 ($2.64)	5 ($2.64)
			10 GB human reads			
10 MB/s	10 G	1	18	220	238 ($2.64)	180 ($1.98)
10 MB/s	10 G	2	18	130	148 ($3.96)	85 ($2.64)
10 MB/s	10 G	4	18	60	78 ($3.3)	50 ($2.64)
			40 GB human reads			
10 MB/s	40 G	1	70	590	660 ($7.26)	686 ($7.92)
10 MB/s	40 G	2	70	310	380 ($8.58)	350 ($7.92)
10 MB/s	40 G	4	70	170	240 ($8.58)	180 ($7.92)
10 MB/s	40 G	8	70	95	165 ($11.22)	100 ($10.56)
10 MB/s	40 G	16	70	53	123 ($11.88)	73 ($21.12)

Table 3: Continued.

Upload_speed	Read size	Nodes	Upload	CompTime	TotalTimeT	TotalTimeS
130 GB human reads						
10 MB/s	130 G	1	224	1720	1944 ($21.78)	2050 ($23.1)
10 MB/s	130 G	2	224	950	1174 ($23.76)	1100 ($25.08)
10 MB/s	130 G	4	224	520	744 ($26.4)	580 ($26.4)
10 MB/s	130 G	8	224	284	508 ($33.66)	320 ($31.68)
10 MB/s	130 G	16	224	160	384 ($34.32)	235 ($42.24)

certain cluster size. It also includes the total experiment time without streaming (which is the summation of upload and computation times) and total time with streaming.

From the results in Table 3, we observe the following.

(i) The use of more machines leads to further reduction of the runtime. For example, it takes 950 minutes to analyze the 130 G human dataset using a cluster of two nodes, and it takes 160 minutes if the cluster size is increased to 16 nodes.

(ii) The streaming mode reduces the overall experiment time, because there is an overlap between data transfer and computation. For example, it takes 508 minutes to upload the 130 G human datasets and to analyze it using a cluster of 8 nodes without streaming (with upload speed of 10 MB/s). With streaming, it takes 320 minutes, which saves about 188 minutes (i.e., ≈37%). With 16 nodes, it takes 384 minutes without streaming and 235 with streaming (i.e., 38%). One can easily note that the overall experiment time with streaming converges to the overall data transfer time of 224 minutes.

(iii) Comparing the different datasets, we note that the advantage of using on-line processing is more apparent with larger data sizes. For small datasets, like the *E. coli*, where the computation time is neglected, the overhead associated with processing the buckets outweighs the advantage of streaming.

(iv) With slower transfer rate, there is no advantage in using more machines, because there is no much data to be processed in parallel. The *E. coli* and 1 M human genome cases with transfer rates of 250 KB/s and 1 MB/s represent this situation.

(v) The advantage with respect to the cost of the experiment can be observed when we fix the computation time and compare the experiment cost. For the 130 GB human dataset, the cost of finishing the analysis in 320 minutes using streaming over a cluster with 8 nodes is $31.68. To finish the experiment in the same time, a larger cluster of more than 16 nodes is needed with a cost larger than $34.32.

Table 4 shows the runtimes in minutes for mapping NGS reads to a reference genome using Crossbow based on the Elastic MapReduce (EMR) product of Amazon. In Table 4, there are more than one EMR clusters in each experiment; this is because we establish a new cluster when more buckets become available. The column titled cluster includes the number of these clusters. Each cluster is composed of 4 nodes of the type `c1.xlarge`.

In this experiment, the results are analogous to that obtained with the traditional cluster running Bowtie. Here, we also observe that the streaming mode is also superior to the nonstreaming mode with larger datasets. The use of streaming option is not advantageous for small datasets, because Crossbow has some overhead time to preprocess each bucket received, and this overhead outweighs the gain of streaming.

7. Conclusions

In this paper, we have introduced *elastream* as a framework for supporting incremental data processing in the cloud. This framework, which is based on the client-server model, is composed of (1) a module for creating and management of cloud computing infrastructure, (2) a module for submission of jobs to the cloud machines with incremental processing feature, (3) a prepared virtual machine image equipped with a large library of bioinformatics tools and databases and all necessary tools, and (4) add-ons in the form of workflows for the popular workflow systems Taverna and Galaxy.

Elastream targets the class of tasks where the input data is composed of large number of sequences that can be processed in parallel. Examples of such tasks include NGS read mapping and blast based queries. Our experiments have shown that the streaming option is useful when the dataset is large enough and the amount of computation at the server size is considerable. With streaming option, one can use fewer machines to finish computation, which leads to reduction of the cost. To sum up, our *elastream* facilitates the use of cloud computing resources for these tasks and its streaming option is of significant advantage to mitigate the effect of the associated data transfer latency.

Currently *elastream* is limited to AWS and to Linux environment. In future versions, we will extend it to include other cloud providers and the Windows operating system. Streaming for MapReduce requires that the tool has short preprocessing time, as this forms an overhead that limits the use of on-line processing option. In this version of *elastream*, we did not use streaming-based MapReduce solution, because they are not yet supported by Amazon. Once they are supported, we will integrate them as part of our package distribution.

All resources related to *elastream* are available at http://www.nubios.nileu.edu.eg/tools/elastream and http://www.elastream.org/.

Table 4: Running times in minutes for mapping NGS reads to a reference genome using Crossbow. The column titled speed specifies the upload speed. The column titled cluster includes the number of created EMR clusters. The column titled compTime incudes the computation time of the whole data after uploading it. The column titled totalTimeS includes the experiment time in streaming mode, and the column titled totalTimeT includes the time in nonstreaming mode, where all the data is first transferred and then processed. The number in brackets in this column is the respective monetary cost.

Upload_speed	Read_size	Clusters	Upload_time	CompTime	TotalTimeT	TotalTimeS
			E. coli reads			
250 KB/s	1.4 G	1	100	9	109 ($0.72)	110 ($0.96)
250 KB/s	1.4 G	2	100	7	107 ($1.2)	110 ($1.92)
250 KB/s	1.4 G	4	100	7	107 ($2.26)	110 ($3.84)
			1 GB human reads			
250 KB/s	1 G	1	71	6	77 ($0.72)	84 ($0.96)
250 KB/s	1 G	2	71	5	76 ($1.2)	84 ($1.92)
250 KB/s	1 G	4	71	5	76 ($1.2)	84 ($3.84)
			10 GB human reads			
250 KB/s	10 G	1	800	60	860 ($2.16)	820 ($6.82)
250 KB/s	10 G	2	800	31	831 ($2.64)	820 ($13.44)
250 KB/s	10 G	4	800	18	818 ($3.6)	820 ($26.88)
			E. coli reads			
1 MB/s	1.4 G	1	25	9	34 ($0.6)	65 ($0.96)
1 MB/s	1.4 G	2	25	7	32 ($1.08)	65 ($1.92)
1 MB/s	1.4 G	4	25	7	32 ($2.04)	65 ($3.84)
			1 GB human reads			
1 MB/s	1 G	1	18	6	24 ($0.6)	31 ($0.48)
1 MB/s	1 G	2	18	5	23 ($1.09)	32 ($0.96)
1 MB/s	1 G	4	18	5	23 ($2.04)	31 ($0.1.92)
			10 GB human reads			
1 MB/s	10 G	1	200	60	260 ($0.96)	220 ($1.92)
1 MB/s	10 G	2	200	31	231 ($1.54)	220 ($3.84)
1 MB/s	10 G	4	200	18	218 ($2.40)	220 ($7.68)
			40 GB human reads			
1 MB/s	40 G	1	690	580	1270 ($6.24)	630 ($5.28)
1 MB/s	40 G	2	690	300	990 ($6.24)	630 ($10.56)
1 MB/s	40 G	4	690	150	840 ($7.2)	630 ($21.12)
1 MB/s	40 G	8	690	80	770 ($9.12)	630 ($42.24)
			130 GB human reads			
1 MB/s	130 G	1	2220	1890	4110 ($19.8)	2360 ($19.2)
1 MB/s	130 G	2	2220	945	3165 ($19.8)	2320 ($37.44)
1 MB/s	130 G	4	2220	476	2696 ($19.8)	2320 ($74.88)
1 MB/s	130 G	8	2220	184	2404 ($19.8)	2320 ($149.76)
1 MB/s	130 G	16	2220	126	2346 ($27.48)	2320 ($299.52)
			10 GB human reads			
10 MB/s	10 G	1	20	60	80 ($1.08)	160 ($1.44)
10 MB/s	10 G	2	20	31	51 ($1.08)	95 ($1.92)
10 MB/s	10 G	4	20	18	38 ($2.04)	65 ($3.84)
			40 GB human reads			
10 MB/s	40 G	1	70	580	650 ($5.04)	610 ($5.28)
10 MB/s	40 G	2	70	300	370 ($5.04)	310 ($5.76)
10 MB/s	40 G	4	70	150	220 ($6.00)	170 ($5.76)
10 MB/s	40 G	8	70	80	150 ($7.92)	110 ($7.68)
			130 GB human reads			
10 MB/s	130 G	1	224	1890	2114 ($15.84)	1960 ($15.84)
10 MB/s	130 G	2	224	945	1169 ($15.84)	1000 ($16.32)
10 MB/s	130 G	4	224	476	700 ($15.84)	520 ($17.28)
10 MB/s	130 G	8	224	184	470 ($15.84)	300 ($19.2)
10 MB/s	130 G	16	224	126	350 ($23.52)	300 ($38.4)

Authors Contribution

All authors contributed equally to this work and all authors have read and approved the manuscript.

Acknowledgments

This work was made possible by NPRP grant 4-1454-1-233 from the Qatar National Research Fund (a member of Qatar Foundation). The statements made herein are solely the responsibility of the authors. The authors thank Amazon for supporting the experimental work on AWS.

References

[1] AWS (Amazon Web Services), http://aws.amazon.com/.

[2] W. Azure, http://www.microsoft.com/windowsazure/.

[3] IBM Smart Cloud Enterprise, http://www.ibm.com/cloud-computing/.

[4] Rackspace, http://www.rackspace.com/.

[5] "Magellan—a cloud for Science," http://magellan.alcf.anl.gov/.

[6] DIAG-Data Intensive Academic Grid, http://diagcomputing.org/.

[7] E. Pennisi, "Will computers crash genomics?" *Science*, vol. 331, no. 6018, pp. 666–668, 2011.

[8] M. C. Schatz, B. Langmead, and S. L. Salzberg, "Cloud computing and the DNA data race," *Nature Biotechnology*, vol. 28, no. 7, pp. 691–693, 2010.

[9] A. Bateman and M. Wood, "Cloud computing," *Bioinformatics*, vol. 25, no. 12, p. 1475, 2009.

[10] J. T. Dudley and A. J. Butte, "In silico research in the era of cloud computing," *Nature Biotechnology*, vol. 28, no. 11, pp. 1181–1185, 2010.

[11] L. D. Stein, "The case for cloud computing in genome informatics," *Genome Biology*, vol. 11, no. 5, article 207, 2010.

[12] V. Fusaro, P. Patil, E. Gafni, D. Wall, and P. Tonellato, "Biomedical cloud computing with Amazon web services," *PLOS Computational Biology*, vol. 7, no. 8, Article ID e1002147, 2011.

[13] B. Langmead, M. C. Schatz, J. Lin, M. Pop, and S. L. Salzberg, "Searching for SNPs with cloud computing," *Genome Biology*, vol. 10, no. 11, article R134, 2009.

[14] D. Wall, P. Kudtarkar, V. Fusaro, R. Pivovarov, P. Patil, and P. Tonellato, "Cloud computing for comparative genomics," *BMC Bioinformatics*, vol. 11, article 259, 2010.

[15] B. Langmead, K. D. Hansen, and J. T. Leek, "Cloud-scale RNA-sequencing differential expression analysis with Myrna," *Genome Biology*, vol. 11, no. 8, article R83, 2010.

[16] M. C. Schatz, "CloudBurst: highly sensitive read mapping with MapReduce," *Bioinformatics*, vol. 25, no. 11, pp. 1363–1369, 2009.

[17] B. Langmead, C. Trapnell, M. Pop, and S. L. Salzberg, "Ultrafast and memory-efficient alignment of short DNA sequences to the human genome," *Genome Biology*, vol. 10, no. 3, article R25, 2009.

[18] C. Rapier and B. Bennett, "High speed bulk data transfer using the SSH protocol," in *Proceedings of the 15th ACM Mardi Gras Conference: From Lightweight Mash-Ups to Lambda grids: Understanding the Spectrum of Distributed Computing Requirements, Applications, Tools, Infrastructures, Interoperability, and the Incremental Adoption of Key Capabilities (MG '08)*, vol. 11, pp. 1–11, ACM.

[19] T. Oinn, M. Addis, J. Ferris et al., "Taverna: a tool for the composition and enactment of bioinformatics workflows," *Bioinformatics*, vol. 20, no. 17, pp. 3045–3054, 2004.

[20] D. Hull, K. Wolstencroft, R. Stevens et al., "Taverna: a tool for building and running workflows of services," *Nucleic Acids Research*, vol. 34, pp. W729–W732, 2006.

[21] B. Giardine, C. Riemer, R. C. Hardison et al., "Galaxy: a platform for interactive large-scale genome analysis," *Genome Research*, vol. 15, no. 10, pp. 1451–1455, 2005.

[22] StarCluster, http://web.mit.edu/stardev/cluster/.

[23] Vappio, http://vappio.sf.net/.

[24] E. Afgan, D. Baker, N. Coraor, B. Chapman, A. Nekrutenko, and J. Taylor, "Galaxy CloudMan: delivering cloud compute clusters," *BMC Bioinformatics*, vol. 11, supplement 12, article S4, 2010.

[25] J. Goecks, A. Nekrutenko, J. Taylor, and T. G. Team, "Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences," *Genome Biology*, no. 8, article R86, 2010.

[26] G. Graefe, "Query evaluation techniques for large databases," *ACM Computing Surveys*, vol. 25, no. 2, pp. 73–170, 1993.

[27] D. Logothetis, C. Trezzo, K. Webb, and K. Yocum, "In-situ MapReduce for log processing," in *Proceedings of the USENIX Conference on USENIX Annual Technical Conference (USENIX-ATC'11)*, pp. 9–9, USENIX Association, 2011.

[28] N. Backman, K. Pattabiraman, and U. Cetintemel, "C-MR: a continuous MapReduce processing model for low-latency stream processing on multi-core architectures," 2010.

[29] R. Kienzler, R. Bruggmann, A. Ranganathan, and N. Tatbul, "Large-scale DNA sequence analysis in the cloud: a stream-based approach," in *Proceedings of the Euro-Par VHPC Workshop*, 2011.

[30] R. Kienzler, R. Bruggmann, A. Ranganathan, and N. Tatbul, "Stream as you go: the case for incremental data access and processing in the cloud," in *Proceedings of the ICDE DMC Workshop*, 2012.

[31] S. M. Rumble, P. Lacroute, A. V. Dalca, M. Fiume, A. Sidow, and M. Brudno, "SHRiMP: accurate mapping of short color-space reads," *PLoS Computational Biology*, vol. 5, no. 5, Article ID e1000386, 2009.

[32] s3fs, "FUSE-based le system backed by Amazon S3," http://code.google.com/p/s3fs/.

[33] J. Dean and S. Ghemawat, "MapReduce: simplied data processing on large clusters," in *Proceedings of the 6th Conference on Symposium on Opearting Systems Design and Implementation (OSDI '04)*, vol. 6, pp. 10–10, USENIX Association, 2004.

[34] B. Linke, R. Giegerich, and A. Goesmann, "Conveyor: a workflow engine for bioinformatic analyses," *Bioinformatics*, vol. 27, no. 7, Article ID btr040, pp. 903–911, 2011.

[35] B. Ludäscher, I. Altintas, C. Berkley et al., "Scientific workflow management and the Kepler system," *Concurrency Computation Practice and Experience*, vol. 18, no. 10, pp. 1039–1065, 2006.

[36] I. Taylor, M. Shields, I. Wang, and A. Harrison, "Visual grid workflow in Triana," *Journal of Grid Computing*, vol. 3, no. 3-4, pp. 153–169, 2005.

[37] I. Taylor, M. Shields, I. Wang, and A. Harrison, "The Triana work ow environment: architecture and applications," in *Workflows for e-Science*, pp. 320–339, Springer, New York, NY, USA, 2007.

[38] E. Deelman, G. Singh, M. H. Su et al., "Pegasus: a framework for mapping complex scientific workflows onto distributed

systems," *Scientific Programming*, vol. 13, no. 3, pp. 219–237, 2005.

[39] S. P. Shah, D. Y. M. He, J. N. Sawkins et al., "Pegasys: software for executing and integrating analyses of biological sequences," *BMC Bioinformatics*, vol. 5, article 40, 2004.

[40] P. Rice, L. Longden, and A. Bleasby, "EMBOSS: the european molecular biology open software suite," *Trends in Genetics*, vol. 16, no. 6, pp. 276–277, 2000.

[41] H. Li, B. Handsaker, A. Wysoker et al., "The Sequence Alignment/Map format and SAMtools," *Bioinformatics*, vol. 25, no. 16, pp. 2078–2079, 2009.

[42] FASTX-Toolkit, http://hannonlab.cshl.edu/fastx_toolkit/.

[43] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool," *Journal of Molecular Biology*, vol. 215, no. 3, pp. 403–410, 1990.

[44] S. F. Altschul, T. L. Madden, A. A. Schäffer et al., "Gapped BLAST and PSI-BLAST: a new generation of protein database search programs," *Nucleic Acids Research*, vol. 25, no. 17, pp. 3389–3402, 1997.

[45] Z. Zhang, S. Schwartz, L. Wagner, and W. Miller, "A greedy algorithm for aligning DNA sequences," *Journal of Computational Biology*, vol. 7, no. 1-2, pp. 203–214, 2000.

[46] M. Eriksen, "Trickle: a userland bandwidth shaper for Unix-like systems," in *Proceedings of the Annual Conference on USENIX Annual Technical Conference (ATEC '05)*, pp. 43–43, 2005.

Hindawi Publishing Corporation
BioMed Research International
Volume 2013, Article ID 939460, 11 pages
http://dx.doi.org/10.1155/2013/939460

Research Article

Exploiting GPUs in Virtual Machine for BioCloud

Heeseung Jo,[1] Jinkyu Jeong,[2] Myoungho Lee,[3] and Dong Hoon Choi[4]

[1] *Department of Information Technology, Chonbuk National University, 567 Baekje-daero, Deokjin-gu, Jeonju-si, Jeollabuk-do 561-756, Republic of Korea*
[2] *Department of Computer Science, Korea Advanced Institute of Science and Technology, 291 Daehak-ro, Yuseong-gu, Daejeon 305-701, Republic of Korea*
[3] *Department of Computer Science and Engineering, Myongji University, 116 Myoungji-ro, Cheoin-gu, Yongin, Gyeonggi-do 449-728, Republic of Korea*
[4] *Korea Institute of Science and Technology Information (KISTI), 245 Daehak-ro, Yuseong-gu, Daejeon 305-806, Republic of Korea*

Correspondence should be addressed to Dong Hoon Choi; choid@kisti.re.kr

Received 7 December 2012; Revised 20 March 2013; Accepted 1 April 2013

Academic Editor: Chun-Yuan Lin

Recently, biological applications start to be reimplemented into the applications which exploit many cores of GPUs for better computation performance. Therefore, by providing virtualized GPUs to VMs in cloud computing environment, many biological applications will willingly move into cloud environment to enhance their computation performance and utilize infinite cloud computing resource while reducing expenses for computations. In this paper, we propose a BioCloud system architecture that enables VMs to use GPUs in cloud environment. Because much of the previous research has focused on the sharing mechanism of GPUs among VMs, they cannot achieve enough performance for biological applications of which computation throughput is more crucial rather than sharing. The proposed system exploits the pass-through mode of PCI express (PCI-E) channel. By making each VM be able to access underlying GPUs directly, applications can show almost the same performance as when those are in native environment. In addition, our scheme multiplexes GPUs by using hot plug-in/out device features of PCI-E channel. By adding or removing GPUs in each VM in on-demand manner, VMs in the same physical host can time-share their GPUs. We implemented the proposed system using the Xen VMM and NVIDIA GPUs and showed that our prototype is highly effective for biological GPU applications in cloud environment.

1. Introduction

Virtualization technology has been widely adopted into computing systems to increase hardware resource utilization and reduce total cost of ownership (TCO). Virtualization technology enables multiple computing environments to be consolidated in a single physical machine. This consolidation brings efficient use of hardware resources and flexible resource provisioning to each computing environment [1]. In cloud computing, virtualization is key enabling technology because flexible resource provisioning is essential for unpredictable user demands.

Although virtualization adds an additional software layer over bare metal hardware, the overhead incurred by the additional layer has been reduced by various efforts. Hardware vendors have extended their architectures to support virtualization. In system software area, paravirtualization techniques, such as the Xen [2], reduce the performance gap between native environment and virtualized environment by slightly modifying operating systems in virtual machines (VMs). Due to the narrowed performance gap, the virtualization technology expands its coverage from cloud computing to high performance computing [3–5]. Recently, biological applications, which require high performance computing environment, are moving into cloud computing environment due to the narrowed performance gap and the advantage of flexible resource provisioning [6–8]. It is better to use elastic resources in cloud than to build one's own computing cluster in terms of TCO.

Meanwhile, graphic processing units (GPUs) are recently exploited in high performance computing because a GPU provides one or two orders of magnitude faster computation than a CPU does. The computing capability of GPU has been rapidly improved for a decade. To cover vastly increased demands of 2D and 3D data processing, GPU embeds hundreds of computing cores in a single chipset. In addition, memory bandwidth in GPU is also widened to follow the increased computing power. For example, NVIDIA Tesla C2090, a state-of-the-art GPU device, is equipped with 512 computing cores, 6 GB of dedicated memory and 177 GB/s internal memory bus [9]. It provides 1331 GFLOPS for single precision floating point operation and 665 GFLOPS for double precision floating point operation.

With the advance of GPU hardware, general purpose GPU computation on graphic processing units (GPGPU) has been emerging to exploit high performance computing capability of GPU not only for 3D graphic manipulation but also for general-purpose computation. The GPU programming framework such as the CUDA [10] and the OpenCL [11] provide application programming interfaces (APIs) of underlying GPU hardware, GPU programming model, and memory model. Due to the open APIs, many general-purpose applications can exploit fast GPUs for their computation-intensive applications [12].

Data copies between host memory and internal memory of GPU could be overhead of GPU computing. In GPU programming models, it is essential to store data on GPU-accessible memory for GPU to manipulate them. Accordingly, the data copies become a critical performance bottleneck of GPU computing. The data copy overhead, however, has been alleviated by making host memory GPU-accessible. In the application processing unit (APU) of AMD, CPU and GPU are integrated in a single chipset, and the two computing units share the same memory controller so that GPU can access the host memory. In addition, various studies have been conducted to lessen the data copy overhead. Despite of the data copy overhead, the performance improvement by GPU computing is significant in many research areas [10, 13–15].

With the trend of GPU computing in high performance computing and biological applications, virtualization needs to be incorporated in the GPU based high performance computing platforms. By providing virtualized GPUs to VMs in cloud computing environment, many biological applications will willingly move into cloud environment to reduce their expenses for computations. However, few studies have been done to use GPU computing in virtual machine environment, and most of them have limitations in terms of performance penalties by GPU virtualization overhead. Therefore, it is also important to minimize the overhead caused by GPU virtualization.

In this paper, we propose a BioCloud system architecture that enables VMs to use GPUs in cloud environment. From the high performance computing power of GPUs, biological applications hosted in cloud can also show high performance while minimizing TCO of their computing infrastructure. The proposed system exploits the pass-through mode of PCI express (PCI-E) channel. By making each VM to be able to access underlying GPUs directly, applications can show almost the same performance as when those are in native environment. In addition, our scheme multiplexes GPUs by using hot plug-in/out device features of PCI-E channel. By adding or removing GPUs in each VM in on-demand manner, VMs in the same physical host can time-share its GPUs.

The rest of the paper is organized as follows. Section 2 introduces brief background and describes related work. Section 3 presents the design and operation of our GPU virtualization and sharing mechanisms. Section 4 demonstrates evaluation results and usability, and Section 5 discusses the superiority of our proposed system for biological applications. Finally, Section 6 concludes our work.

2. Background and Related Work

2.1. VMM and GPU Virtualization. Virtualization provides an illusion, a VM, to its hosted operating system. By multiplexing underlying hardware resources, a physical host can consolidate multiple VMs simultaneously. The core of virtualization technology is virtual machine monitor (VMM) that is in charge of multiplexing hardware resources such as CPU, memory, and I/O devices to multiple VMs. The common role of VMMs is to provide virtualized resources albeit their implementations diverse from emulation of virtual devices to hardware-assisted virtualization.

Hardware-level assists to virtualization take overhead out of a VMM. Previously, a VMM either emulates the behavior of virtualized devices or incorporates with operation systems in VMs. The overhead of these approaches affects the system performance as compared to native environment. To reduce this overhead, hardware vendors extended their CPUs to support virtualization. The Intel VT [16] and the AMD-V [17] unburden the VMM overhead of CPU and memory virtualization. In addition, the Intel VT-d [18] and the AMD-Vi [19] support device virtualization so that operating systems in VMs can directly access I/O devices without security concerns. Despite of these technology advances, GPU has limitations for virtualization by itself. Since a GPU is in charge of manipulating graphic data, a large amount of data should be transferred from a VM to a GPU device. Since the amount of data is too large, emulation, one of methods to virtualize a device, is inefficient for GPUs.

In GPU virtualization, there have been some related previous research such as the gVirtuS [20], the GViM [21], and the vCUDA [22]. These ultimately aim at providing the flexible sharing of a GPU among VMs taking performance degradation lying down. Therefore, they have the similar architecture that host operating system or VMM manages the overall operations and privileges of GPU. In the case of the gVirtuS based on KVM, one of VMMs working on top of host operating system provides a mechanism to access GPU based on the communication between virtual device drivers on host operating system and VM for each. The GViM and the vCUDA are based on the Xen, a VMM on bare-metal, and similarly use virtual device drivers between VMs and VM0 which is in charge of I/O. Although these GPU virtualization architectures, which are based on virtual device drivers, can enhance the sharing of GPUs among VMs, there are two critical limitations as below.

(i) Reimplementation and low flexibility of GPU APIs: for sharing of GPU among VMs, the management of GPU is concentrated on host operating system or VMM, and the communication between virtual device drivers is highly dependent on the implementation of them. Actually, they have to reimplement all GPU APIs, and this limits flexibility and portability. For example, if a version of GPU APIs is updated or modified, the virtual device drivers should be reengineered according to the changes. Even more, VMs in a physical host should use the same GPU APIs and version as the implemented virtual device drivers.

(ii) High performance overhead: the architectures on related work adopt the fine-grained time-sharing technique among VMs to share GPU, but these largely depredate the overall performance of GPU by increasing the communication traffic between virtual device drivers and system bus. Moreover, the response times of GPU are not uniform due to the scheduling of VMs. According to the papers, they show 10–40% performance degradation.

In order to overcome these limitations, our scheme uses the direct pass-through approach to use GPU in virtualized environment [23]. By using the direct pass-through approach, operating systems in VMs can exclusively and directly access underlying GPU devices. Accordingly, the performance penalty when VMM involves in arbitrating GPUs can be eliminated.

2.2. GPU and Biological Application. The term GPU was mentioned by NVIDIA Corporation first, when it announced a new graphic controller named GeForce in 1999. In the early 1990s, graphic controllers in general PCs were in charge of simply translating computation results of CPU to visual characters on monitors. After the mid of 1990s, the role of graphic controller started to change into manipulating rich multimedia contents. Multimedia contents usually require lightning effects and texture mapping in order to make the contents more realistic. These computations burden CPU, thus an additional coprocessor, like GPU, is needed to lessen the computation load for multimedia processing in CPU.

Although the architecture of GPU is similar to that of CPU, GPU has enhanced processing parallelism as shown in Figure 1. In the figure, CPU has a few high performance arithmetic and logic units (ALUs) and the large region of chipset is assigned to internal caches. The reason for this composition is to improve performance in task-level parallelism. On the other hands, GPU consists of many small ALUs optimized for graphic data processing in a single chipset. Since the characteristic of computations on GPU is highly data parallel, multiple data can be calculated by the same arithmetic operations. Accordingly, GPU outperforms the CPU in terms of parallel processing on the same computation with multiple data [24].

Since computations supported by GPU are specialized to graphical data manipulation, general-purpose computations cannot be easily ported into GPU. To address this obstacle, various general-purpose GPU frameworks have been built.

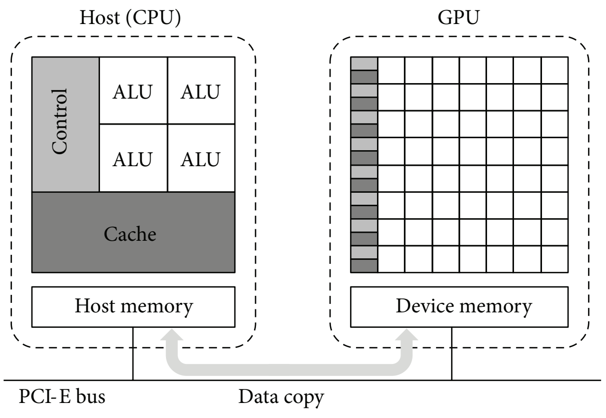


Figure 1: The architecture of a GPU equipped machine on PCI-E channel.

The CUDA of NVIDIA and the OpenCL of Khronos Group are most representative frameworks in the area of general-purpose high performance computing. These frameworks provide an extended C language so that non-graphic-friendly programmers can intuitively translate their computation logics into GPU-friendly ones.

From the consensus between GPU performance improvement and easy programming APIs, biological applications prevalently start to use GPUs for their computations. Especially, computations requiring a large amount of data, such as next generation sequencing and protein simulation, are proper targets to exploit GPUs. Manavski and Valle suggested a GPU implementation of Smith-Waterman sequence alignment [6], and Vouzis and Sahinidis transformed the BLAST tool to a GPU based application, named the GPU-BLAST [7]. The barraCUDA [25] and the G-aligner [26] are also kinds of short sequence alignment tool with GPU acceleration.

3. GPU Virtualization

3.1. Overview. Our approach for virtualizing and sharing of GPUs is based on the GPU direct access scheme in our previous work [23]. The previous work exploits the PCI-E direct pass-through mechanism of GPUs in order to reduce the virtualization overhead and to increase the GPU API flexibility. Since the direct pass-through approach can minimize the interference incurred by VMM, VMs can achieve bare-metal performance. Moreover, since each VM can use their own GPU APIs, it can be freed from the reengineering and modification of GPU APIs.

The direct pass-through approach in virtualized system should be supported by input/output memory management unit (IOMMU) hardware feature. Similar to a traditional memory management unit (MMU) which translates CPU-visible virtual addresses to physical addresses, IOMMU takes care of mapping device-visible I/O addresses to physical addresses and also provides memory protection from misbehaving devices.

Figure 2 shows the system architecture diagram of the proposed scheme, the GPU virtualization using direct pass-through. The system we suppose has a privileged control VM,

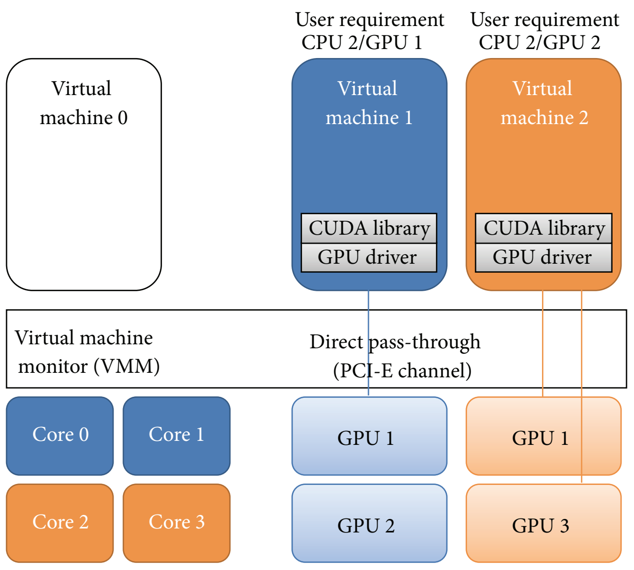


FIGURE 2: The system architecture of direct pass-through GPU virtualization.

generally called VM0, to interface with VMM and control other user VMs. Note that VM0 denotes the privileged control VM, and guest VMs denote other VMs of users. In the system, each GPU is attached to PCI-E channels and VMM passes the control of PCI-E channel and GPU to VMs. Each VM has its own GPU API and GPU device driver, and it can access and control GPU without the intervention of VMM.

However, the direct pass-through mechanism has a limitation that the sharing of GPU among VMs is not possible. For example, once a GPU is allocated to a VM at booting time of the VM, it cannot be deallocated until the VM halts. To address this problem, we extend our previous work to have the coarse-grained sharing mechanism based on the hot plug functionality of PCI-E channel. The sharing mechanism enables VM0 to allocate GPUs to VMs and deallocate GPUs from VMs while the VMs are running. To enhance the utilization of GPUs, we add two features: (1) allocating GPUs when a VM actually requires GPUs to compute its data and (2) deallocating GPUs immediately after the computation.

The hot plug-in/out mechanism that is adopted in this research is a mechanism to install or remove PCI devices on online. To utilize the hot plug mechanism in virtualized systems, VMM needs the functionality of IOMMU. The system we suppose has GPU installed on a PCI-E channel, and a GPU can be allocated or deallocated by using the PCI-E channel hot plug-in/out. Due to this mechanism, the users of VMs can utilize virtualized GPUs in the same way with native GPUs.

Figure 3 shows the overall coarse-grained GPU sharing mechanism. First, (1) each guest VM requests GPU allocation to the VM0 when it needs GPU computation. Then, (2) the VM0 checks the GPU pool which has all GPUs installed on the host machine to find an available GPU. If there is an available GPU in the GPU pool, the VM0 plugs in the GPU into the requested VM. (3) The guest VM processes its job, and (4) the GPU is revoked from the guest VM after the end of computation using the hot plug-out message.

In this mechanism, largely, we have two main considerations. One is when to allocate and deallocate GPUs. Although the users of guest VMs can directly request the allocation and deallocation of GPUs, it might decrease usability. It is inconvenient to allocate GPUs manually before invoking a GPU application. We need to provide more convenient interfaces. The other is how to prevent excessive occupation of GPUs by VMs. It is crucial to increase the overall utilization of GPUs in a system. Therefore, we need a compulsory revoking mechanism when a GPU of a guest VM is not utilized for computation. Then, the reclaimed GPU can be used for other guest VMs. For this mechanism, we designed and implemented the GPU-Admin module in the VM0 and the *GPU-Manager* module in the guest VMs, respectively, as shown in Figure 4. The detailed operations of them are described in the following subsections.

3.2. *GPU-Admin.*

The GPU-Admin is a daemon process in the VM0 and performs allocation and deallocation of GPUs according to the request of the GPU-Managers. It also takes in charge of compulsory revocation of unused GPUs. For this responsibility, it periodically checks the status of GPUs allocated for guest VMs and deallocates them if GPUs are not actually used for computation.

In the initial stage of the GPU-Admin, it identifies the number of GPUs installed on a virtualized system and records the PCI-E channel information and the slot ID of each GPU. Using this information, the GPU-Admin registers all GPUs into the GPU pool for later management. All of the registered GPUs are initialized and stay in available state, and one of them is allocated via the GPU-Admin when a guest VM requests a GPU. At this moment, the GPU-Admin stores the virtual machine identification (VMID) and IP address of the guest VM to be referred for usage checking and compulsory revoking operations. After the initialization of GPU pool, the GPU-Admin creates two worker threads: the ManagerListener which handles the requests of the GPU-Manager and the PoolChecker to prevent unnecessary GPU occupation of VMs.

3.2.1. *ManagerListener.*

The ManagerListener worker is a part of GPU-Admin to accept and handle the requests from guest VMs. The message types of GPU-Managers in guest VMs are five, and the corresponding reactions are as follows.

(1) GPU allocation request by a user: it is the case that the user of a guest VM explicitly requests the allocation of GPU for computation. Responding to this request, the ManagerListener picks an available GPU from the GPU pool and allocates it into the requested guest VM via hot plug-in mechanism.

(2) GPU deallocation request by a user: on contrary to (1), this message is to explicitly deallocate an allocated GPU of guest VM by a user. The ManagerListener deallocates the GPU from the guest VM and registers it again into the GPU pool after reinitialization of the GPU.

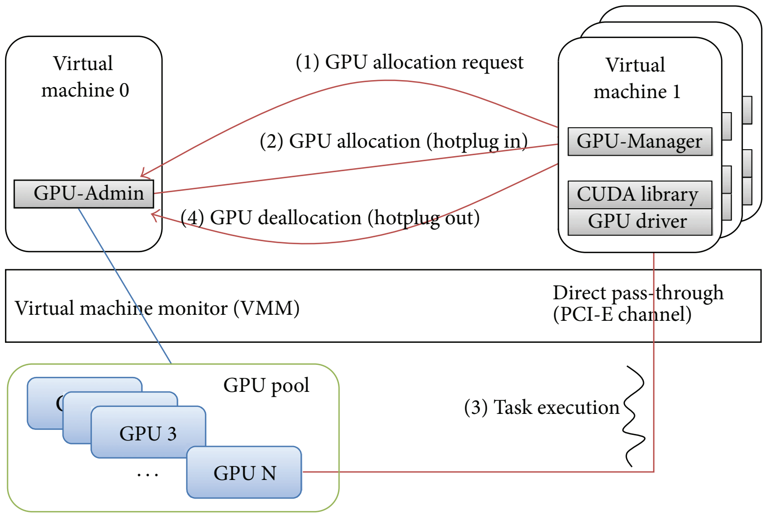

Virtual machine 0
GPU-Admin
(1) GPU allocation request
(2) GPU allocation (hotplug in)
(4) GPU deallocation (hotplug out)
Virtual machine 1
GPU-Manager
CUDA library
GPU driver
Virtual machine monitor (VMM)
Direct pass-through (PCI-E channel)
GPU pool
GPU 3
GPU N
(3) Task execution

Figure 3: The overall coarse-grained GPU sharing mechanism sequences.

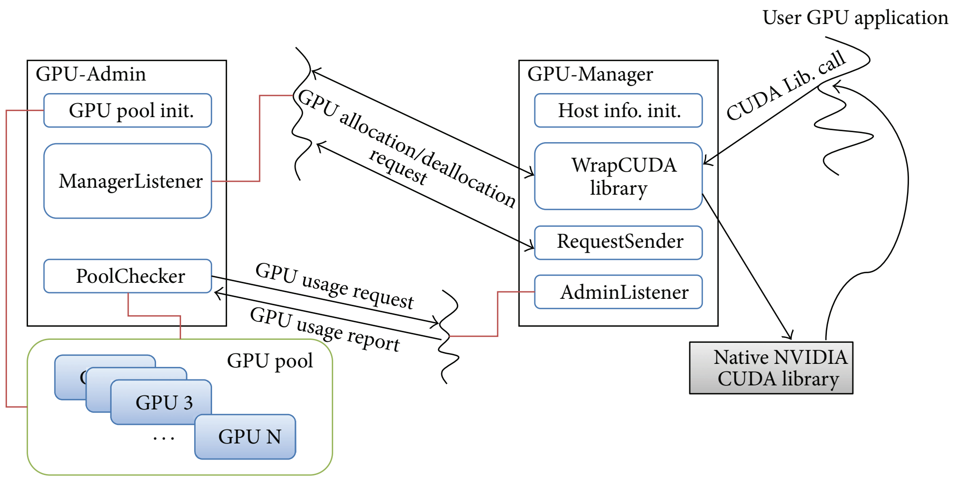


Figure 4: The detailed modules and operations of the GPU-Admin and the GPU-Manger.

(3) GPU allocation request by the WrapCUDA library: when a user of a VM invokes a GPU application without an allocated GPU, the WrapCUDA library detects the situation and automatically requests GPU allocation to the GPU-Admin. This request message is delivered implicitly and transparently without recognition of users. The handling of this message is the same with the case (1).

(4) GPU deallocation request by the WrapCUDA library: before finishing GPU application, if it calls several specific CUDA library call, the WrapCUDA library requests the deallocation of GPUs implicitly. This mechanism is also performed without the intervention of users. Responding to this message, the ManagerListener deallocates the GPU similarly to the case (2).

(5) Disconnection from WrapCUDA library: from the case (3) to (4), the ManagerListener and the WrapCUDA library keeps their connection. If the connection is disconnected for any reasons such as halting or finishing the GPU application, the ManagerListener recognizes it and then deallocates the GPU of the guest VM.

In our prototype, we use TCP/IP network communication method between the GPU-Admin and the GPU-Manager to utilize the virtue of its well-defined and concrete interfaces. Since the communication messages are not incurred frequently, its overhead is negligible. The hot plug-in/out operations of GPUs by the ManagerListener are performed via the management interface of the Xen.

In the case that there is no available GPU in the GPU pool, the allocation request is inserted into the waiting queue of the ManagerListener, and the response is blocked until a GPU become available by being deallocated from other guest VMs. If more than one allocation request messages are waiting in the waiting queue, the ManagerListener prioritizes them according to their waiting time.

3.2.2. PoolChecker. The major role of the PoolChecker is to find the allocated GPUs to guest VMs and decide that they are currently used for computation or not. We add this mechanism to prevent the situation that a guest VM excessively occupies GPUs while it does not actually utilize GPUs for computation. Due to this mechanism, the system can achieve higher overall utilization of GPUs by inhabiting unnecessary occupation of GPUs.

The PoolChecker works periodically. When triggered, it checks the GPU pool to search GPUs already allocated to guest VMs and sends a message requesting usage report of GPU to each GPU-Manager of guest VMs. On receipt of this message, each GPU-Manager investigates the usage of its own GPU and replies to the PoolChecker. The PoolChecker tracks the utilization of each GPU and deallocates it forcibly, if it has not been used for computation for a while. This GPU utilization check mechanism of the PoolChecker is a last resort to prevent useless occupation of GPUs by guest VMs. In most cases, where the users of guest VMs are not evil, the unused GPUs are immediately reclaimed by the deallocation message or the disconnection message of the WrapCUDA library.

Our prototype sets the trigger period of the PoolChecker as five seconds and the reclaim time limit as ten seconds. These configured values are decided intuitively because we assume that the users of VMs might not need GPUs currently, if the GPUs are not used for computation more than ten seconds. The time-out value of the PoolChecker is easily reconfigurable by editing the configuration file of the PoolChecker.

3.3. GPU-Manager. The GPU-Manager is a module working in a guest VM to provide interfaces allocating or deallocating GPUs and report the utilization of its GPU responding to the request of the PoolChecker in the GPU-Admin. The interfaces to request GPU allocation and deallocation can be divided in two. One is a transparent and implicit interface, which is requested by the WrapCUDA library when a user starts and finishes a GPU application. The other is an explicit interface, which is performed by a user action of guest VM.

In the initialization stage of the GPU-Manager, it identifies its hostname, local IP address, and GPU-Admin IP address to communicate with the GPU-Admin for allocation and deallocation of GPUs. The GPU-Manager consists of three parts: the AdminListener thread which handles the request of the PoolChecker, the WrapCUDA library hooking the API calls of the native CUDA library to support the implicit allocation or deallocation, and the RequestSender to provide the explicit user interface.

3.3.1. AdminListener. AdminListener thread works similarly to the ManagerListener of GPU-Admin. It opens a specified port and waits the GPU usage report request. After allocation of GPUs, the PoolChecker sends periodical requests to check the utilization of GPUs, and the AdminListener responds to this request. To get the utilization of its GPUs, the AdminListener searches the usage of GPU driver module in the proc filesystem, a virtual filesystem presenting process and system information. At least, if more than one GPU application utilizes GPU hardware, the driver module works and the usage of the driver module increases. Therefore, sending this information, the AdminListener reports the usage of GPUs to the PoolChecker. Then, the GPU-Admin can deallocate the GPUs of guest VMs based on this criteria, if it decides that the GPUs are not used for computation.

3.3.2. WrapCUDA Library. The WrapCUDA library of GPU-Manager is to allocate and deallocate GPUs automatically. To increase the usability of GPUs, the WrapCUDA library enables GPU allocation and deallocation without the intervention of users, when a GPU application starts and finishes.

The WrapCUDA library is implemented in a shared library and is preloaded using LD_PRELOAD environment variable after the startup of guest VMs. It is in charge of dynamic allocation of GPU when a user starts a GPU application even if one's VM does not have a GPU. A GPU application should use a specified programming framework such as the CUDA to interface GPU, and several APIs of them are generally called to probe and initialize its GPU in the initial stage of the application. For example, the cudaGetDeviceCount() function to get the number of GPUs installed on system and the cudaMalloc() to allocate heap memory on GPU are representatives. The WrapCUDA library hooks these kinds of several native CUDA library calls. If a GPU application calls the wrapped CUDA library functions to access GPU, they are redirected to the WrapCUDA library which implements the same function due to LD_PRELOAD environment value. Then, the WrapCUDA library checks whether its guest VM has a GPU and requests GPU allocation if the guest VM has no GPU. After a GPU is allocated, the function of WrapCUDA library executes the native function of the CUDA library.

Similarly, a GPU application calls several CUDA APIs to release the resource of GPU at finishing time, and the WrapCUDA library catches these calls before transferring to the native CUDA library for deallocation of GPU from its VM. Although a GPU application might not call these resource release APIs by implementation, the WrapCUDA library can deallocate the GPU of VM immediately, since the connection between the WrapCUDA library and the GPU-Admin is disconnected after the halt of a GPU application. Additionally, as mentioned in Section 3, we also implement the PoolChecker mechanism, which checks the status of GPU and deallocates it after several seconds to prevent the occupation of GPUs unnecessarily.

In order to interpose the API calls of GPU applications, the WrapCUDA library has to define API functions to catch and embed the implementations of them. These might not be a burden to implement the WrapCUDA library, since the number of APIs to start GPU application is limited to

Table 1: The functions that are wrapped by the WrapCUDA library.

GPU allocation call	GPU deallocation call
cudaGetDeviceCount ( )	cudaThreadExit ( )
cudaGetDevice ( )	
cudaMalloc ( )	
cudaDeviceReset ( )	
cudaChooseDevice ( )	
cudaDeviceSynchronize ( )	

Table 2: The specifications of evaluation system.

Device	Specification
CPU	Intel(R) Xeon(R) E5620 (2.40 GHz)
Chipset	Intel(R) 5520
Memory	DDR3 1333 MHz (24 G)
PCI slot	PCI Express Gen2, 4 EA
GPU	NVIDIA Quadro FX 3800, 4 EA

several ones and the patterns to develop a GPU application are regularized. Actually, our prototype implements the API functions listed in Table 1, and the top three functions can cover all the 84 GPU application examples and several biological GPU applications used in the evaluation section. Even more, for the implementation of a function call in the WrapCUDA library, we used seventeen C language code lines as shown in Algorithm 1. This engineering overhead is negligible for an experienced programmer and can be applied to other GPU programming frameworks besides the CUDA programming framework.

3.3.3. RequestSender. To run GPU applications without the WrapCUDA library mechanism, the users of VMs can explicitly request the allocation and deallocation of GPUs. The RequestSender module provides the explicit interfaces. In this case, the GPU explicitly allocated via the RequestSender is specially handled as an exception for the PoolChecker. Therefore, it is not deallocated automatically until the user requests deallocation of the GPU via the RequestSender interface. Since this mechanism enables a user of VM to monopolize GPUs in a system, it should be allowed to trusted users when they need continuous and reliable GPU computing environment.

4. Evaluation

4.1. Overview. We used the Xen VMM for implementing and evaluating the prototype of the proposed BioCloud architecture. For GPU computation, we used the NVIDIA GPUs and the CUDA programming frameworks. The host machine specification is summarized in Table 2. Briefly, the machine is equipped with Intel Xeon E5620 CPU, four NVIDIA Quadro FX 3800 GPUs, and 24 GB of main memory. Each GPU is installed on PCI-E (2nd generation) channel in the host machine.

The main goals of the prototype evaluation are (1) the GPU virtualization overhead of our scheme, when GPU applications run in virtualized environment as compared to those in native environment, (2) the latencies of GPU hot plug-in/out in VMs, and (3) the benefits by our scheme.

4.2. Virtualization Overhead. In this section, we compare the proposed virtualization scheme with others which are mentioned in the related work and measure the biological application execution time when each application runs in native environment and in virtualized environment using our scheme.

Figure 5 shows the execution time comparison of BlackScholes application among the GViM, the vCUDA, and our scheme. Unfortunately, we cannot replay all the three schemes (gVirtuS, GViM, vCUDA) because their hardware and software configuration are too outdated. Instead, we borrowed the evaluation results compared native environment based on their papers. Note that the gVirtuS scheme is omitted because it does not evaluate BlackScholes application. Although their evaluation environments are all different from each other, we can focus on the gap between native and VM and identify that the execution time overhead of our scheme is less than 0.5%, while those of others are 25~73%. Because the other two schemes are based on the virtual device driver mechanism and focused on the sharing of a GPU among VMs, their overheads are not negligible.

Figure 6 is the evaluation results of several biological applications. In this evaluation, we ran the barraCUDA [25], CUDASW++ [27], MUMmerGPU [28], and CUDA-MEME [29]. The x-axis denotes the workloads and biological applications, and the y-axis is normalized execution time which includes disk I/O, CPU computation, and GPU computation. The overhead of our scheme is 3% on average, and the maximum overhead case is the MUMmerGPU with the SSUIS workload by 10%. From our analysis, the most part of overhead is caused by the disk I/O of virtualized system to read workload data files, not by the GPU virtualization. Except the disk I/O time, the GPU computation time is the same as that in the native environment. Comparing the result of Figure 5 which does not include the disk I/O overhead, we can confirm that the direct pass-through GPU virtualization mechanism shows the least overhead and highly useful for long running biological applications.

4.3. Sharing Effect

4.3.1. GPU Allocation and Deallocation Overhead. Our scheme uses GPU hot plug-in/out to provide sharing of GPUs among VMs. In this evaluation, we measure additional time overhead to hot plug-in/out of GPUs to a VM. Table 3 shows the time for allocating and deallocating a GPU in a VM. Each value is the average of ten attempts. As shown in the table, the allocation time and the deallocation time are 1.3 seconds, identically. Considering general biological applications take from tens of minutes to hundreds of minutes, this allocation and deallocation time penalty is negligible compared to the total execution time of general biological applications.

4.3.2. Effect of Real Workload. The main role of our scheme is to make a GPU directly accessible and to coordinate

```
cudaError_t cudaGetDeviceCount(int *count) {
    static cudaError_t (*ofuncp)(int *count);
    char *error;
    cudaError_t rs;

    checkgpuon(); // Checks having GPU. Request a GPU, if not.

    if (!ofuncp) {
        ofuncp = dlsym(RTLD_NEXT, "cudaGetDeviceCount");
        if ((error = dlerror()) != NULL) {
            fputs(error, stderr);
            exit(1);
        }
    }
    rs = ofuncp(count);
    return rs;
}
```

Algorithm 1: An example implementation of the WrapCUDA library function.

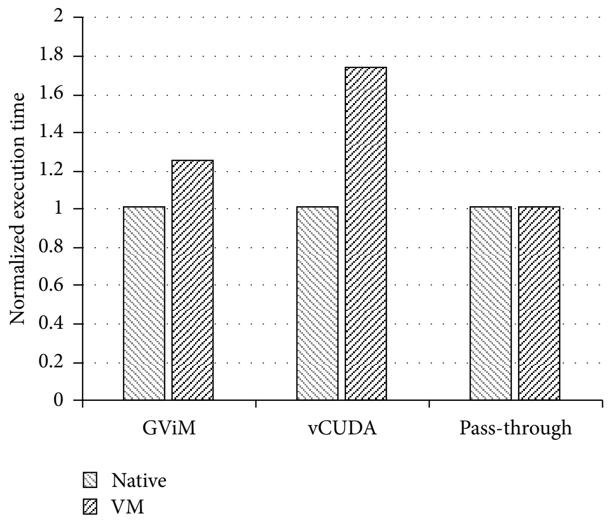


Figure 5: The performance comparison with other schemes.

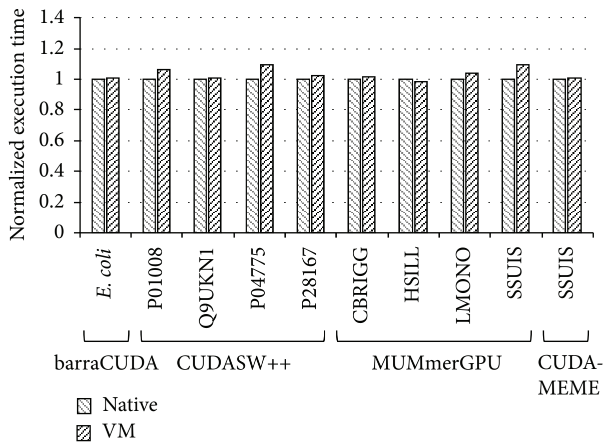


Figure 6: The performance evaluation using bioapplications.

the ownership of a GPU among VMs for time sharing. In order to reveal the effect of GPU sharing in our scheme, we measured the application execution time when multiple VMs share a finite number of GPUs. In this evaluation, we used the barraCUDA application that performs next generation sequencing. Each of the barraCUDA applications performs five-million read mappings of *E. coli* genome data. One execution of the application takes 18 seconds in average. One workload in a VM runs the barraCUDA application five times with sleep time between consecutive runs. The sleep time models the behavior of a user that interprets the result of a previous run and adjusts parameters for the next run. We varied the sleep time to 9, 18, and 36 seconds, and each are 50%, 100%, and 200% of the application execution time, respectively. During this sleep time, an allocated GPU to a VM is returned to the GPU-Admin and could be reallocated to another VM for sharing. Accordingly, we can expect performance improvement by reducing the idle time of GPUs.

Figure 7 shows the evaluation results with four VMs and varying the number of GPUs from one to four. The sleep time of each evaluation is denoted in the parentheses of each legend. For example, *Execution(S9)* denotes that the sleep time is 9 seconds. *Execution* denotes the real execution of four VMs while *Theoretical* denotes the theoretical execution time in four VMs when a GPU(s) is exclusively used by the VM. Hence, the time is the same as the execution time when four workloads run in a single VM sequentially with the given number of GPUs. Albeit the theoretical execution time is not realizable without our mechanism, the time is used for comparison purpose. The execution time in our scheme is normalized to the theoretical time with the same configuration of sleep time and the number of GPUs, for each.

In case of *Execution(S9)*, the sleep time after each completion of application is 9 seconds. When four VMs share one GPU, our scheme shows reduced total execution time by 20% as compared to the theoretical time. When the

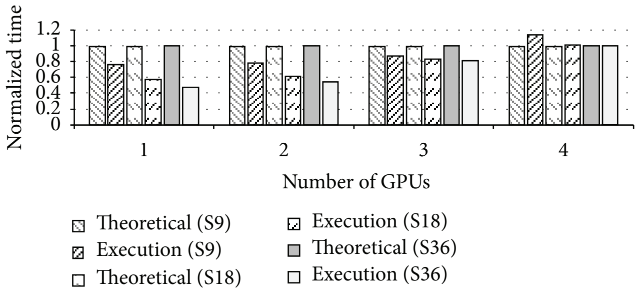


FIGURE 7: The GPU sharing effect using the barraCUDA.

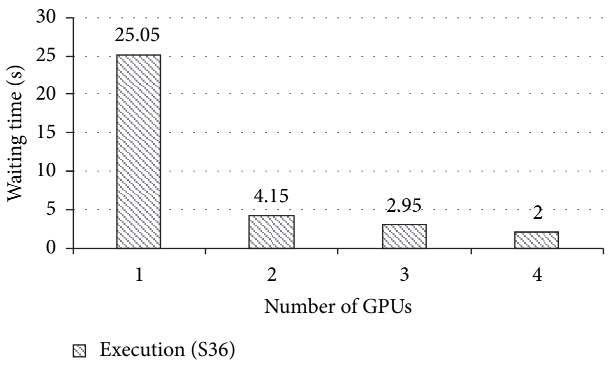


FIGURE 8: The average waiting time of VMs.

sleep time increases, our scheme reduces the total execution time because the increased sleep time naturally increases the probability to share the GPU during the sleep time. In case of *Execution(S36)* with one GPU, our scheme shows reduced total execution time by 53%. As the number of GPUs increases, the performance benefit of our scheme reduces. When the number of VMs is the same as the number of GPUs, our scheme shows no performance benefit. But, we believe that the number of VMs requiring GPUs will be more than the number of GPUs installed in a physical host, since multiple applications and VMs will be consolidated in a single physical machine in the cloud computing environment.

4.3.3. Waiting Time. Finally, we measure the waiting time, the time to wait for allocation of a GPU, in the same evaluation. When there is no available GPU, a VM should be idle until a GPU becomes available. Accordingly, this additional waiting time can worse the overall execution time of each workload. Figure 8 shows the average GPU waiting time with varying the number of GPUs from one to four. When the number of GPU is one, the average waiting time is 25.05 seconds. Although this waiting time could increase the execution time of a single workload, the overall performance is still improved as shown in Figure 7. In addition, when the number of GPU increases more than one, the average waiting time is significantly reduced to 4.15 seconds. Except the hot plug-in time, 1.3 seconds as shown in Table 3, a VM should wait 2.85 seconds on average when the number of GPU is two. This result indicates that if the number of GPUs is more than one, GPUs can be efficiently shared among VMs with negligible performance degradation.

TABLE 3: The time to hot plug-in/out of PCI-E channel.

Operation	Time (second)
GPU allocation (hot plug-in)	1.3 ± 0.1
GPU deallocation (hot plug-out)	1.3 ± 0.1

The modeled scenario in the two previous subsections is synthetic and might be different case by case. Note that the benefit gap and the waiting time in these evaluations will diverse according to the running time and the sleep time of GPU applications. But, at least, we can confirm that our mechanism to virtualize and share GPUs works fine and is reasonably effective for biological GPU applications in BioCloud.

5. Discussion

In this section, we discuss how our proposed architecture is suitable for biological applications in terms of memory and time of workloads in bioinformatics.

One of major characteristics of workloads in bioinformatics is massively data-intensive computing. For example, in a protein sequence alignment workload, a large volume of gnome references are indexed by large-size hashing index (tens of GBs). The workload can perform better when the reference indices are (mostly) in GPU memory and most of them are filled in GPU's internal cache memory [26]. When multiple workloads share a single GPU device at fine-grained level (e.g., API-level multiplexing [20–22]), the GPU memory should be shared between multiple workloads. Accordingly, the memory size for each workload is inevitably reduced. Otherwise, when a GPU context switch occurs, the data in a GPU's memory should be replaced with the data for the next workload [30]. This data replacement leads to unnecessary and slow data copies between host memory and GPU memory (and memory copies between host memory and guest VM memory [22]). Since the bioinformatics workloads are highly data intensive, these penalties by memory sharing can result in performance degradation.

Our scheme, however, does not cause those penalties. In our GPU virtualization architecture, a workload can fully exploit the memory in a GPU device while the workload is running. A GPU context switch only occurs after a currently scheduled workload finishes. Accordingly, our scheme can show better performance by eliminating those memory penalties.

The other characteristic to note is that bioinformatics workload usually takes a long time from several minutes to hundreds of minutes depending on the workloads [25, 28]. Therefore, when the API-level multiplexing schemes are used, fine-grained sharing of GPUs may result in thousands of context switches on a GPU in a second. This frequent context switch causes frequent flushes of warmed-up data in GPU internal caches and even in GPU memory [30, 31]. Although the batching GPU APIs [22] are aimed at reducing the frequent context switches, it cannot eliminate whole context switches in a long-time period (tens of seconds) so that flushing warmed-up data in GPU cache and GPU memory is inevitable.

Our GPU-virtualization architecture can avoid the frequent context switches in a GPU device thereby showing better performance. Since a GPU is multiplexed at coarse-grained level (at workload), each scheduled workload can fully exploit the cache and memory in a GPU without unnecessary flushing of data until the workload finishes. When the context switch occurs, the previous workload never reuses the warmed-up data in those memory spaces because the workload is finished. As evidence, our architecture shows minimal performance degradation as compared to the other schemes.

6. Conclusions

For higher utilization of systems, the machine virtualization and cloud computing trend will prevail more and more. Besides, the high performance computing based on GPU also has proved its outstanding capabilities for general-purpose computation in many research areas. Especially, the biological applications are outstanding, since their computation can derive a large amount of benefit from many cores of GPUs.

For biological GPU applications, we propose a cloud system to exploit GPUs in VM while multiplexing them among VMs and achieving almost the same performance as that with native use of GPUs. Considering the characteristics of bioinformatics workloads which have long execution time, our GPU virtualization mechanism is focused on high GPU computation throughput, rather than sharing GPUs among VMs. Although our prototype is based on the Xen VMM and NVIDIA GPUs, it can be easily ported into other implementations. In the evaluation section, we showed the effectiveness of our mechanism using a modeled scenario. Although the performance benefit and the waiting time can diverse in case by case, we believe that our scheme is highly effective for biological computation using GPUs in cloud environment.

Acknowledgments

This work was partially supported by Defense Acquisition Program Administration and Agency for Defense Development.

References

[1] D. T. Meyer, G. Aggarwal, B. Cully et al., "Parallax: virtual disks for virtual machines," in *Proceedings of the 3rd ACM European Conference on Computer Systems (EuroSys '08)*, pp. 41–54, April 2008.

[2] P. Barham, B. Dragovic, K. Fraser et al., "Xen and the art of virtualization," in *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP '03)*, pp. 164–177, New York, NY, USA, October 2003.

[3] S. K. Garg, C. S. Yeo, A. Anandasivam, and R. Buyya, "Environment-conscious scheduling of HPC applications on distributed Cloud-oriented data centers," *Journal of Parallel and Distributed Computing*, vol. 71, no. 6, pp. 732–749, 2011.

[4] J.-H. Um, S. B. Park, J. H. Seo, and D. H. Choi, "Design of bio-cloud service for genomic analysis based on virtual infrastructure," in *Proceedings of the 6th International Conference on Computer Sciences and Convergence Information Technology (ICCIT '11)*, pp. 304–307, Seogwipo, South Korea, 2011.

[5] X. Qiu, J. Ekanayake, S. Beason et al., "Cloud technologies for bioinformatics applications," in *Proceedings of the 2nd ACM Workshop on Many-Task Computing on Grids and Supercomputers (MTAGS '09)*, New York, NY, USA, November 2009.

[6] S. A. Manavski and G. Valle, "CUDA compatible GPU cards as efficient hardware accelerators for Smith-Waterman sequence alignment," *BMC Bioinformatics*, vol. 9, supplement 2, article S10, 2008.

[7] P. D. Vouzis and N. V. Sahinidis, "GPU-BLAST: using graphics processors to accelerate protein sequence alignment," *Bioinformatics*, vol. 27, no. 2, Article ID btq644, pp. 182–188, 2011.

[8] Z. Meng, J. Li, Y. Zhou, Q. Liu, Y. Liu, and W. Cao, "CloudBLAST: an efficient mapreduce program for bioinforrnatics applications," *BMEI*, vol. 4, pp. 2085–2089, 2011.

[9] NVIDIA, http://www.nvidia.com/docs/IO/105880/DS_Tesla-M2090_LR.pdf.

[10] S. A. Manavski, "CUDA compatible GPU as an efficient hardware accelerator for AES cryptography," in *Proceedings of the IEEE International Conference on Signal Processing and Communications (ICSPC '07)*, pp. 65–68, Dubai, UAE, November 2007.

[11] Khronos OpenCL Working Group, *The OpenCL 1. 0 Specification*, Khronos Group, 2008.

[12] GPGPU, http://gpgpu.org/.

[13] D. L. Cook and A. D. Keroymytis, *Cryptographics: Exploiting Graphics Cards for Security*, Advancements in Information Security series, Springer, New York, NY, USA, 2006.

[14] D. L. Cook, J. Ioannidis, A. D. Keromytis, and J. Luck, "CryptoGraphics: secret key cryptography using graphics cards," in *Proceedings of the Cryptographer's Track RSA Conference (CT-RSA '05)*, pp. 334–350, San Francisco, CA, USA, February 2005.

[15] N. K. Govindaraju, S. Larsen, J. Gray, and D. Manocha, "A memory model for scientific algorithms on graphics processors," in *Proceedings of the ACM/IEEE Conference on Supercomputing*, Tampa, Fla, USA, November 2006.

[16] R. Uhlig, G. Neiger, D. Rodgers et al., "Intel virtualization technology," *Computer*, vol. 38, no. 5, pp. 48–56, 2005.

[17] "AMD virtualization: introducing AMD virtualization," 2006, http://www.amd.com/virtualization/.

[18] R. Hiremane, "Intel virtualization technology for directed I/O (Intel VT-d)," *Technology@Intel Magazine*, vol. 4, no. 10, 2007.

[19] "AMD I/O virtualization technology (IOMMU) specification," 2007.

[20] G. Giunta, R. Montella, G. Agrillo, and G. Coviello, "A GPGPU transparent virtualization component for high performance computing clouds," *Euro-Par 2010 Parallel Processing*, vol. 6271, pp. 379–391, 2010.

[21] V. Gupta, A. Gavrilovska, K. Schwan et al., "GViM: GPU-accelerated virtual machines," in *Proceedings of the 3rd ACM Workshop on System-level Virtualization for High Performance Computing (HPCVirt '09)*, pp. 17–24, March 2009.

[22] L. Shi, H. Chen, and J. Sun, "VCUDA: GPU accelerated high performance computing in virtual machines," in *Proceedings of the 23rd IEEE International Parallel and Distributed Processing Symposium (IPDPS '09)*, pp. 1–11, Rome, Italy, May 2009.

[23] H. Jo, M. Lee, and D. H. Choi, “GPU virtualization using PCI direct pass-through,” *Applied Mechanics and Materials*, vol. 2, no. 2, pp. 113–118, 2013.

[24] V. W. Lee, C. Kim, J. Chhugani et al., “Debunking the 100X GPU vresus CPU Myth: an evaluation of throughput computing on CPU and GPU,” in *Proceedings of the 37th International Symposium on Computer Architecture (ISCA ’10)*, pp. 451–460, New York, NY, USA, June 2010.

[25] P. Klus, S. Lam, D. Lyberg et al., “BarraCUDA-a fast short read sequence aligner using graphics processing units,” *BMC Research Notes*, vol. 5, article 27, 2012.

[26] M. Lu, Y. Tan, G. Bai, and Q. Luo, “High-performance short sequence alignment with GPU acceleration,” *Special Issue on Data Intensive Escience of Distributed and Parallel Databases*, vol. 30, pp. 385–399, 2012.

[27] Y. Liu, D. L. Maskell, and B. Schmidt, “CUDASW++: optimizing Smith-Waterman sequence database searches for CUDA-enabled graphics processing units,” *BMC Research Notes*, vol. 2, article 73, 2009.

[28] M. C. Schatz, C. Trapnell, A. L. Delcher, and A. Varshney, “High-throughput sequence alignment using graphics processing units,” *BMC Bioinformatics*, vol. 8, article 474, 2007.

[29] Y. Liu, B. Schmidt, W. Liu, and D. L. Maskell, “CUDA-MEME: accelerating motif discovery in biological sequences using CUDA-enabled graphics processing units,” *Pattern Recognition Letters*, vol. 31, no. 14, pp. 2170–2177, 2010.

[30] S. Kato, M. McThrow, C. Maltzhan, and S. Brandt, “Gdev: first-class GPU resource management in the operating system,” in *Proceedings of the USENIX annual technical conference*, 2012.

[31] S. Kato, K. Lakshmanan, R. Rajkumar, and Y. Ishikawa, “TimeGraph: GPU scheduling for real-time multi-tasking environments,” in *Proceedings of the USENIX Annual Technical Conference*, 2011.

Hindawi Publishing Corporation
BioMed Research International
Volume 2013, Article ID 469363, 12 pages
http://dx.doi.org/10.1155/2013/469363

Methodology Report

wFReDoW: A Cloud-Based Web Environment to Handle Molecular Docking Simulations of a Fully Flexible Receptor Model

Renata De Paris,[1] Fábio A. Frantz,[2] Osmar Norberto de Souza,[1] and Duncan D. A. Ruiz[2]

[1] *Laboratório de Bioinformática, Modelagem e Simulação de Biossistemas (LABIO), Faculdade de Informática (FACIN), Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS), Avenida Ipiranga 6681, Prédio 32, Sala 608, 90619-900 Porto Alegre, RS, Brazil*
[2] *Grupo de Pesquisa em Inteligência de Negócio (GPIN), Faculdade de Informática (FACIN), Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS), Avenida Ipiranga 6681, Prédio 32, Sala 628, 90619-900 Porto Alegre, RS, Brazil*

Correspondence should be addressed to Osmar Norberto de Souza; osmar.norberto@pucrs.br and Duncan D. A. Ruiz; duncan.ruiz@pucrs.br

Received 6 December 2012; Revised 28 February 2013; Accepted 6 March 2013

Academic Editor: Ming Ouyang

Molecular docking simulations of fully flexible protein receptor (FFR) models are coming of age. In our studies, an FFR model is represented by a series of different conformations derived from a molecular dynamic simulation trajectory of the receptor. For each conformation in the FFR model, a docking simulation is executed and analyzed. An important challenge is to perform virtual screening of millions of ligands using an FFR model in a sequential mode since it can become computationally very demanding. In this paper, we propose a cloud-based web environment, called web Flexible Receptor Docking Workflow (wFReDoW), which reduces the CPU time in the molecular docking simulations of FFR models to small molecules. It is based on the new workflow data pattern called self-adaptive multiple instances (P-SaMIs) and on a middleware built on Amazon EC2 instances. P-SaMI reduces the number of molecular docking simulations while the middleware speeds up the docking experiments using a High Performance Computing (HPC) environment on the cloud. The experimental results show a reduction in the total elapsed time of docking experiments and the quality of the new reduced receptor models produced by discarding the nonpromising conformations from an FFR model ruled by the P-SaMI data pattern.

1. Introduction

Large-scale scientific experiments have an ever-increasing demand for high performance computing (HPC) resources. This typical scenario is found in bioinformatics, which needs to perform computer modeling and simulations on data varying from DNA sequence to protein structure to protein-ligand interactions [1]. The data flood, generated by these bioinformatics experiments, implies that technological breakthroughs are paramount to process an interactive sequence of tasks, software, or services in a timely fashion.

Rational drug design (RDD) [2] constitutes one of the earliest medical applications of bioinformatics [1]. RDD aims to transform biologically active compounds into suitable drugs [3]. *In silico* molecular docking simulation is one of the main steps of RDD. It is used to deal with compound discovery, typically by computationally virtual screening a large database of organic molecules for putative ligands that fit into a binding site [4] of the target molecule or receptor (usually a protein). The best ligand orientation and conformation inside the binding pocket is computed in terms of the free energy of bind (FEB) by software, for instance the AutoDock4.2 [5].

In order to mimic the natural, *in vitro* and *in vivo*, behavior of ligands and receptors, their plasticity or flexibility should be treated in an explicit manner [6]: our receptor is a protein that is an inherently flexible system. However, the majority of molecular docking methods treat the ligands as flexible and the receptors as rigid bodies [7]. In this study we model the explicit flexibility of a receptor by using an ensemble of conformations or snapshots derived from its molecular dynamics (MD) simulations [8] (reviewed by [9]). The resulting model receptor is called a fully-flexible receptor (FFR) model. Thus, for each conformation in the FFR model, a docking simulation is executed and analyzed [7].

Organizing and handling the execution and analysis of molecular docking simulations of FFR models and flexible ligands are not trivial tasks. The dimension of the FFR model can become a limiting factor because instead of performing docking simulations in a single, rigid receptor conformation, we must carry out this task for all conformations that make up the FFR model [6]. These conformations can vary in number from thousands to millions. Therefore, the high computing costs involved in using FFR models to perform practical virtual screening of thousands or millions of ligands may make it unfeasible. For this reason, we have been developing methods to simplify or reduce the FFR model dimensionality [6, 9, 10]. We named this simpler representation of an FFR model a reduced fully flexible receptor (RFFR) model. An RFFR model is achieved by eliminating redundancy in the FFR model through clustering its set of conformations, thus generating subgroups, which should contain the most promising conformations [6].

To address these key issues, we propose a cloud-based web environment, called web Flexible Receptor Docking Workflow (wFReDoW), to fast handle the molecular docking simulations of FFR models. To the best of our knowledge, it is the first docking web environment that reduces both the dimensionality of FFR models and the overall docking execution time using an HPC environment on the cloud. The wFReDoW architecture contains two main layers: Server Controller and (flexible receptor middleware) FReMI. Server Controller is a web server that prepares docking input files and reduces the size of the FFR model by means of the self-adaptive multiple instances (P-SaMIs) data pattern [9]. FReMI handles molecular docking simulations of FFR models integrated with an HPC environment on Amazon EC2 resources [11].

There are a number of approaches that predict ligand-receptor interactions on HPC environments using Auto-Dock4.2 [5]. Most of them use the number of ligands to distribute the tasks among the processors. For instance, DOVIS 2.0 [12] uses a dedicated HPC Linux cluster to execute virtual screening where ligands are uniformly distributed on each CPU. VSDocker 2.0 [13] and Mola [14] are other examples of such systems. Whilst VSDocker 2.0 works on multiprocessor computing clusters and multiprocessor workstations operated by a Windows HPC Server, Mola uses AutoDock4.2 and AutoDock Vina to execute the virtual screening of small molecules on nondedicated compute clusters. Autodock4.lga.MPI [15] and mpAD4 [16] use another approach to enhance the performance. As well as the docking parallel execution, Autodock4.lga.MPI and mpAD4 reduce the quantity of network I/O traffic during the loading of grid maps at the beginning of each docking simulation. Another approach is the AutoDockCloud [17]. This is a high-throughput screening of parallel docking tasks that uses the open source Hadoop framework implementing the MapReduce paradigm for distributed computing on a cloud platform using AutoDock4.2 [5]. Although every one of these environments reduces the overall elapsed time of the molecular docking simulations, they only perform docking experiments with rigid receptors. Conversely, wFReDoW applies new computational techniques [6, 10, 11, 18] to reduce the CPU time in the molecular docking simulations of FFR models using public databases of small molecules, such as ZINC [19].

In this work we present the wFReDoW architecture and its execution. From the wFReDoW executions we expect to find better ways to reduce the total elapsed time in the molecular docking simulations of FFR models. We assess the gains in performance and the quality of the results produced by wFReDoW using a small FFR model clustered by data mining techniques, a ligand from ZINC database [19], different P-SaMI parameters [10], and an HPC environment built on Amazon EC2 [18]. Thus, from the best results obtained, we expect that future molecular docking experiments, with different ligands and new FFR models, will use only the conformations that are significantly more promising [6] in a minimum length of time.

2. Methods

2.1. The Docking Experiments with an FFR Model. To perform molecular docking simulations we need a receptor model, a ligand, and docking software. We used as receptor the enzyme 2-*trans*-enoyl-ACP (CoA) reductase (EC 1.3.1.9) known as InhA from *Mycobacterium tuberculosis* [20]. The FFR model of InhA was obtained from a 3,100 ps (1 picosecond = 10^{-12} second) MD simulation described in [21], thus making an FFR model with 3,100 conformations or snapshots. In this study, for each snapshot in the FFR model, a docking simulation is executed and analyzed. Figure 1 illustrates the receptor flexibility.

The ligand triclosan (TCL400 from PDB ID: 1P45A) [20] was docked to the FFR model. We chose TCL from the referred crystal structure because it is one of the simplest inhibitors cocrystallized with the InhA enzyme. Figure 2 illustrates the reference position of the TCL400 ligand into its binding site (PDB ID: 1P45A) and the position of the TCL ligand after an FFR InhA-TCL molecular docking simulation.

For docking simulations, we used the AutoDock Tools (ADT) and AutoDock4.2 software packages [5]. Input coordinate files for ligand and the FFR model of InhA were prepared with ADT as follows. (1) Receptor preparation. A PDBQT file for each snapshot from the FFR model was generated employing Kollman partial atomic charges for each atom type. (2) Flexible ligand preparation. The TCL ligand was initially positioned in the region close to its protein binding pocket and allowed two rotatable bonds. (3) Reference ligand preparation. This is the ideal position and orientation of the ligand that is expected from docking simulations. A TCL reference ligand was also prepared using the coordinates of the experimental structure (PDB ID: 1P45A). It is called the reference ligand position. (4) Grid preparation. For each snapshot a grid parameter file (GPF) was produced with box dimensions of 100 Å × 60 Å × 60 Å.

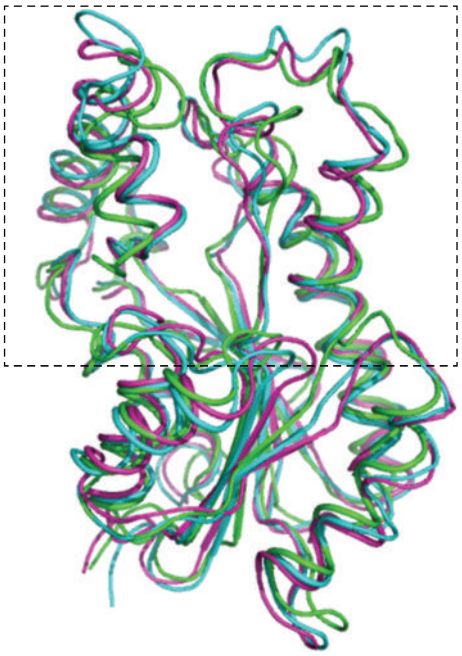


Figure 1: Flexibility of the InhA enzyme receptor from *Mycobacterium tuberculosis* [PDB ID: 1P45A]. Superposition of different InhA conformations, represented as ribbons, along an MD simulation. The initial conformation of the simulation is the experimental crystal structure and is colored in green. Two other conformations or snapshots were taken from the MD simulation at 1,000 ps (blue) and 3,000 ps (magenta). The outlined rectangle highlights the most flexible regions of this receptor.

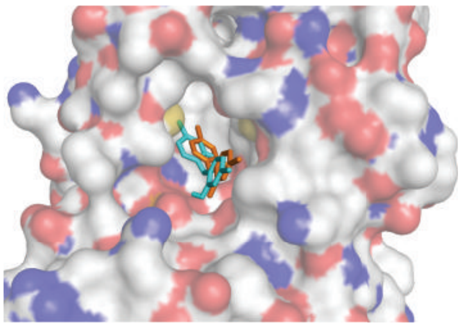


Figure 2: Molecular docking simulation. Molecular surface representation of the binding pocket of the InhA enzyme receptor in the crystal structure [PDB ID: 1P45A] colored by atom type (carbon and hydrogen: light grey; nitrogen: blue; oxygen: red; sulphur: yellow). The TCL ligand (TCL400 from PDB ID: 1P45A) is represented by stick models. The crystallographic reference for the TCL ligand is colored orange. The TCL ligand generated by molecular docking simulation is colored cyan.

The other parameters maintained the default values. (5) Docking parameters. Twenty-five Lamarckian genetic algorithm (LGA) independent runs were executed for each docking simulation. The LGA search method and parameters were: a population size of 150 individuals, a maximum of 250,000 energy evaluations and 27,000 generations. The other docking parameters were kept at default values.

2.2. Reducing the Fully Flexible Receptor Model. The snapshots of the FFR model used in this study are derived from an MD simulation trajectory of the receptor. Even though this approach is considered the best to mimic the natural behavior of ligands and receptors [9], its dimension or size may become a limiting factor. Moreover, the high computing cost involved could also make the practical virtual screening of such receptor models unfeasible. For these reasons, new methods have been developed to assist in the simplification or reduction of an FFR model to an RFFR model. The primary rationale of this approach is to eliminate redundancy in the FFR model through clustering of its constituent conformations [6]. This is followed by the generation of subgroups with the most promising conformations via the P-SaMI data pattern [10].

2.2.1. Clusters of Snapshots from an FFR Model. The clusters of snapshots used in this study were generated using clustering algorithms with different similarity functions developed by [6, 7]. Basically, in this approach, our FFR model was used to find patterns that define clusters of snapshots with similar features. In this sense, if a snapshot is associated with a docking with significantly negative FEB, for a unique ligand, it is possible that this snapshot will interact favorably with structurally similar ligands [6]. As a consequence, the clusters of snapshots, which were related to different classes of FEB values, are postprocessed using the P-SaMI data pattern to select the receptor conformations and, thus, to reduce the complexity of the FFR model.

2.2.2. P-SaMI Data Pattern for Scientific Workflow. P-SaMI is the acronym for pattern-self-adaptive multiple instances—a data pattern for scientific workflows developed by [10]. The purpose of this approach is to define a data pattern which is able to dynamically perform the selection of the most promising conformations from clusters of similar snapshots. As shown in Figure 3, the P-SaMI first step is to capture a clustering of snapshots from [6]. Next, P-SaMI divides each cluster into subgroups of snapshots to progressively execute *autogrid4* and *autodock4* for each conformation that makes up the FFR model using an HPC environment. The results (*docking results*) are the best FEB value for each docked snapshot. From these results, P-SaMI uses previous FEB results (*evaluation criteria*) to determine the status and priority of the subgroups of snapshots. Status denotes whether a subgroup of snapshots is active (A), finalized (F), discarded (D), or with changed priority (P). Priority indicates how promising the snapshots are belonging to that subgroup, on a scale of 1 to 3 (1 being the most promising). Thus, if the docking results of a subgroup present an acceptable value of FEB then that subgroup is credited with a high priority. Conversely, the subgroup has its priority reduced or its status changed to "D" and is discarded, unless all the snapshots of that subgroup have already been processed (status "F").

The reason for using P-SaMI in this work is to make full use of its data pattern to eliminate the exhaustive execution of docking simulations of an FFR model without affecting its quality [6, 10] from clusters of snapshots produced by [6, 7] as input files. In this sense, we make use of a web server

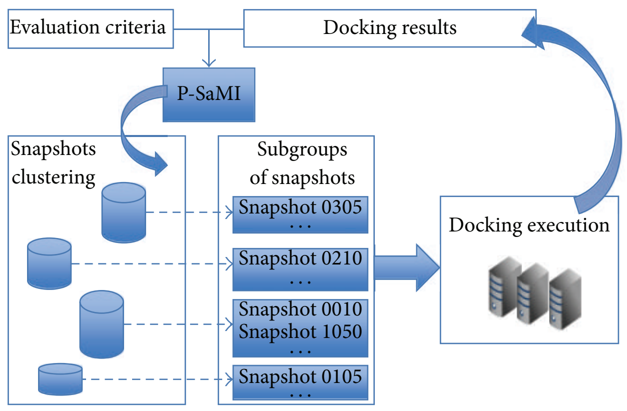


Figure 3: Model of P-SaMI data pattern execution. Clustered snapshots are divided into subgroups using the P-SaMI data pattern. Molecular docking simulations are executed on these subgroups. P-SaMI analyses the docking results, based on some evaluation criteria, to select promising conformations from subgroups of snapshots.

environment, herein called server controller, to perform the P-SaMI data pattern and a middleware (FReMI) to handle promising snapshots and send them to an HPC environment on the cloud to execute the molecular docking simulations.

2.3. HPC on Amazon EC2 Instances. Cloud computing is a new promising trend for delivering information technology services as computing utilities [22]. Commercial cloud services can play an attractive role in scientific discovery because they provide computer power on demand over the internet, instead of several commodity computers connected by a fast network. Our virtual HPC environment on Amazon EC2 was built using the GCC 4.6.2 and MPICH2 based on a master-slave paradigm [23]. It contains 5 High-CPU extra large (*c1.xlarge*) EC2 Amazon instances, each equipped with 8 cores with 2.5 EC2 computer units, 7 GB of RAM, and 1,690 GB of local instance storage. A rating of one EC2 computer units is a unit of CPU capacity which corresponds to 1.0–1.2 GHZ 2007 Opteron or 2007 Xeon processor.

Figure 4 shows the cluster pool created on Amazon EC2's instances where the same files directory is shared by network file system (NFS) among the instances to store all input and output files used during run time of FReMI. In this pool, all data are stored on the Elastic Block Store (EBS) of the master machine and all the instances have permission to read and write in this shared directory, even if a slave instance terminates. However, if the master instance terminates, all data are lost because the master instance EBS volume terminates at the same time. Thus, the S3cmd source code (S3cmd is an open source project available under GNU Public License v2 and free for commercial and private use. It is a command line tool for uploading, retrieving, and managing data in Amazon's S3. S3cmd is available at http://s3tools.org/s3cmd) and package is used to replicate the most important information from Amazon EC2 to Amazon S3 bucket (bucket is the space to store data on Amazon S3. Each bucket is identified with a unique bucket name).

3. Results

The results are aimed at showing the wFReDoW architecture and validating its execution using clusters of snapshots of a specific FFR model against a single ligand. From these results we try to evidence that the proposed cloud-based web environment can be more effective than other methods used to automate molecular docking simulations with flexible receptors, such as [24]. In this sense we divided our results into three parts. Firstly, we present the wFReDoW conceptual architecture to get a better understanding about its operation. Next, a set of experiments is examined to discover the best FReMI performance on Amazon EC2 Cloud. Finally, the new RFFR models are presented by means of the wFReDoW execution.

3.1. wFReDoW Conceptual Architecture. This section presents the wFReDoW conceptual architecture (Figure 5) which was developed to speed up the molecular docking simulations for clusters of the FFR model's conformations. wFReDoW contains two main layers: Server Controller and FReMI. Server Controller is a web workflow based on P-SaMI data pattern that prepares Autodock input files and selects promising snapshots through docked snapshots. FReMI is a middleware based on the many-task computing (MTC) [25] paradigm that handles high-throughput docking simulations using an HPC environment built on Amazon EC2 instances. In our study, MTC is used to address the problem of executing multiple parallel tasks in multiple processors. Figure 5 details the wFReDoW conceptual architecture with its layers and interactions. The wFReDoW components are distributed in three layers: Client, Server Controller and FReMI.

3.1.1. Client Layer. The Client layer is a web interface used by the scientist to configure the environment. It initializes the wFReDoW execution and analyzes information about the molecular docking simulations. Client is made up of three main components: (i) *Setup* component sets up the whole environment before starting the execution; (ii) *Execute* starts the wFReDoW execution and; (iii) *Analyze* shows the provenance of each docking experiment. The communication between Client and Server Controller is done by means of Ajax (http://api.jquery.com/category/ajax/).

3.1.2. Server Controller. Server Controller is a web workflow environment that aids in the reduction of the execution time of molecular docking simulations of FFR models by means of P-SaMI data pattern. It was built using the web framework FLASK 0.8 (http://flask.pocoo.org/) and the Python 2.6.6 libraries. The Server Controller central role is to select promising subgroups of snapshots from an FFR model based on the P-SaMI data pattern [10]. It contains three components: *Configuration*, *Molecular Docking*, and *P-SaMI*. The *Configuration* component only stores data sent from *Setup* (Client layer).

The *Molecular Docking* component manages the P-SaMI input files and performs the predocking steps required for AutoDock4.2 [5]. Firstly, the *Prepare Files* activity reads the

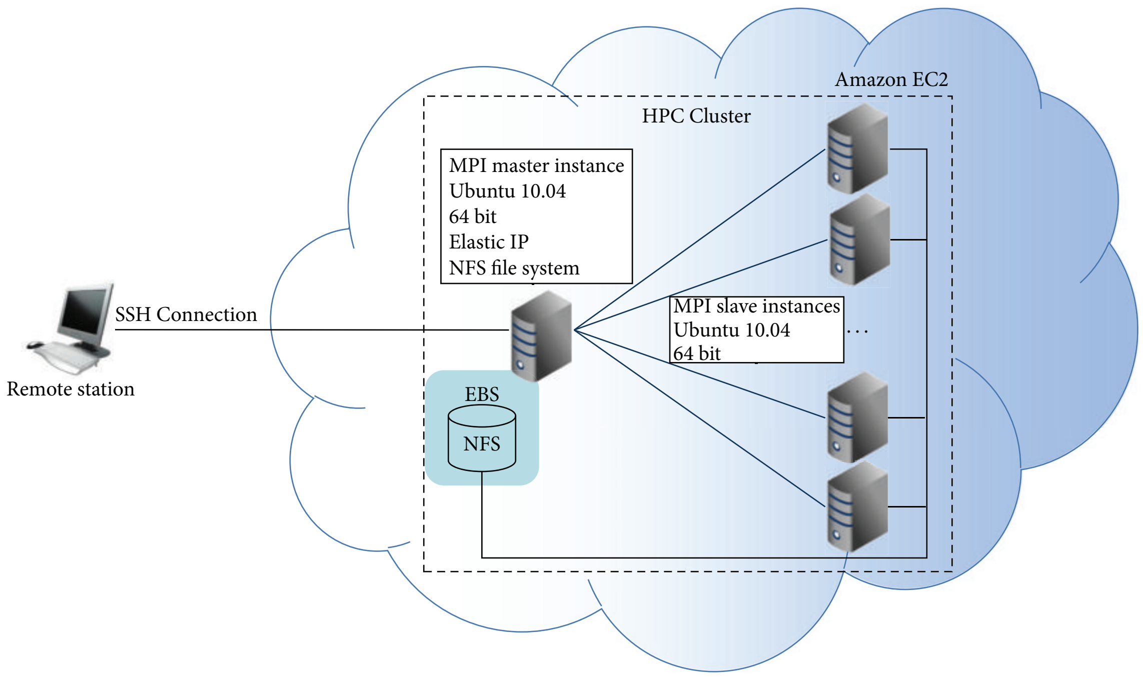


Figure 4: MPI cluster environment created to execute FReMI on Amazon Ec2. The remote station represents the machine outside Amazon EC2 used to connect the MPI master instance by an SSH connection. The MPI master instance is the machine that manages the MPI slaves during the FReMI execution. It also holds the FReMI source code and the I/O files stored on the Amazon Elastic Block Store (EBS). All instances may access EBS through NFS.

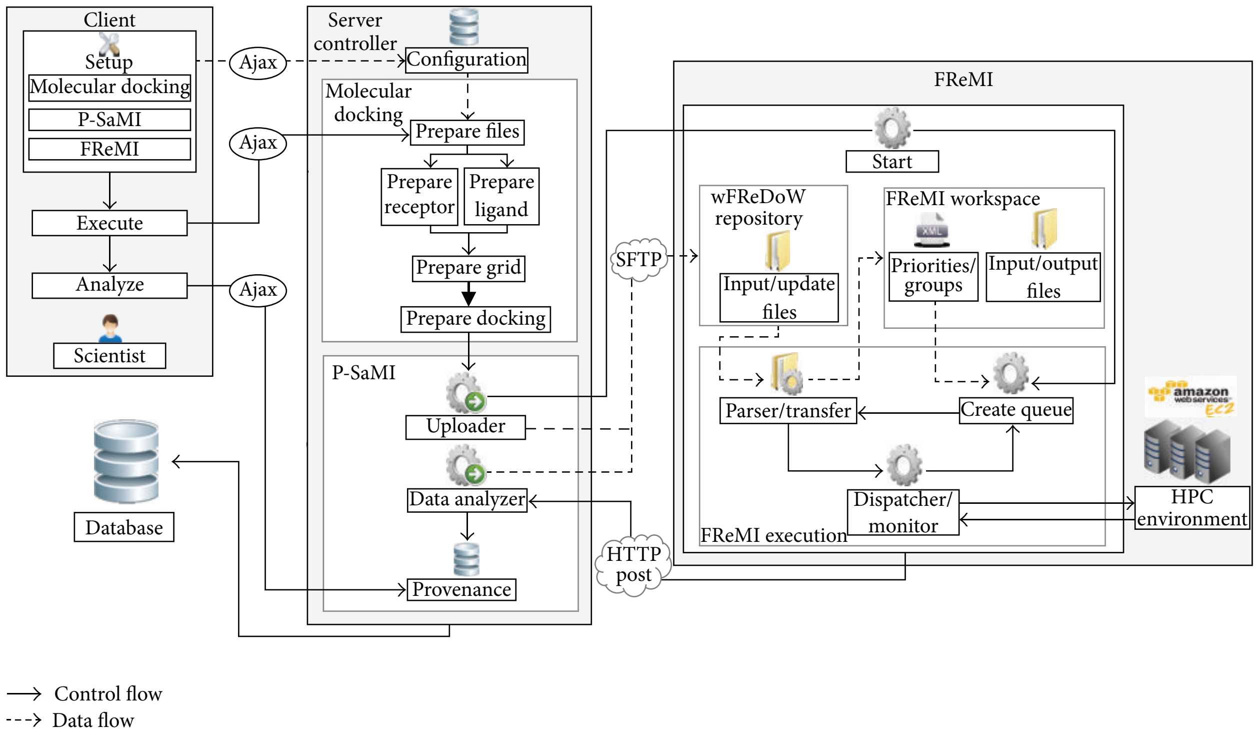


Figure 5: wFReDoW conceptual architecture and its interactions. The two left boxes show the tasks to be executed by a user on the web server, which sends and receives messages to and from FReMI on Amazon EC2. The HPC environment represents the MPI cluster on Amazon EC2.

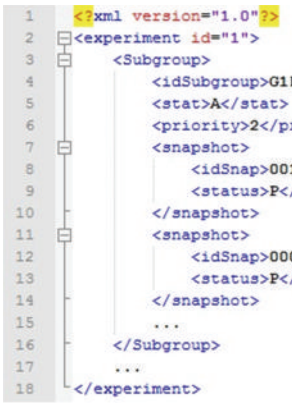
clustering of snapshots generated by [6] and stores them in the *Database*. Next, the *Prepare Receptor* and *Prepare Ligand* activities generate the PDBQT files used as input files to *autogrid4* and *autodock4*. Finally, the *Prepare Grid* and *Prepare Docking* activities create the input files according to the *autogrid4* and *autodock4* parameters, respectively.

After all files have been prepared by the *Molecular Docking* component, the *P-SaMI* component is invoked. This identifies the most promising conformations using the P-SaMI data pattern [10] from different clusters of snapshots of an FFR model identified by [6]. The *P-SaMI* component contains three activities: *Uploader*, *Data Analyzer*, and *Provenance*.

Uploader starts the FReMI execution and generates subgroups from snapshot clustering [6]. These subgroups are stored in an XML file structure, called wFReDoW control file (Figure 6). The wFReDoW control file is sent to the *Parser/Transfer* component (within FReMI) before starting the wFReDoW execution. It contains three root tags described as: *experiment*, *subgroup*, and *snapshot*. The experiment identification (id) is a unique number created for each new docking experiment with an FFR model and one ligand. The *subgroup* tag specifies the information of the subgroups. The *stat* and *priority* tags indicate how promising the snapshots belonging to that subgroup are, according to the rules of the P-SaMI data pattern. The *snapshot* tag contains information about the snapshots and is used by FReMI to control the docked snapshots.

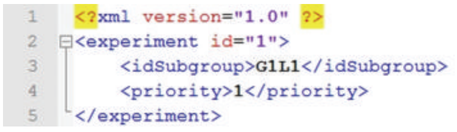
The *Data Analyzer* activity examines the docking results, which are sent from FReMI by HTTP Post, based on P-SaMI data pattern. The result of these analyses is a parameter set that is stored in the wFReDoW update files (Figure 7). Thus, to keep FReMI updated with the P-SaMI results, *Data Analyzer* sends wFReDoW update files to FReMI by SFTP protocol every time P-SaMI modifies the priority and/or status of a subgroup of snapshots.

The *Database* component is based on FReDD database [26], built with PostgreSQL 4.2 (http://www.postgresql.org/docs/9.0/interactive/), and is used to provide provenance about data generated by Server Controller. The *Provenance* activity stores the Server Controller data in the Database component. Hence, the scientist is able to follow wFReDoW execution whenever he/she needs.



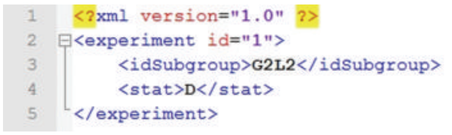
```
<?xml version="1.0"?>
<experiment id="1">
    <Subgroup>
        <idSubgroup>G1L1</idSubgroup>
        <stat>A</stat>
        <priority>2</priority>
        <snapshot>
            <idSnap>001055</idSnap>
            <status>P</status>
        </snapshot>
        <snapshot>
            <idSnap>000100</idSnap>
            <status>P</status>
        </snapshot>
        ...
    </Subgroup>
    ...
</experiment>
```

Figure 6: Fragment of the wFReDoW control file. The file places the subgroups of snapshots generated by data mining techniques and its parameters according to P-SaMI.



```
<?xml version="1.0" ?>
<experiment id="1">
    <idSubgroup>G1L1</idSubgroup>
    <priority>1</priority>
</experiment>
```

(a)



```
<?xml version="1.0" ?>
<experiment id="1">
    <idSubgroup>G2L2</idSubgroup>
    <stat>D</stat>
</experiment>
```

(b)

Figure 7: Examples of wFReDoW update files. (a) An XML file where the priority from G1L1 subgroup changed to 1. (b) An XML file where the status from G2L2 subgroup changed to D.

3.1.3. FReMI: Flexible Receptor Middleware.

FReMI is a middleware on the Amazon Cloud [18] that handles many tasks to execute, in parallel, the molecular docking simulations of subgroups of conformations of FFR models. It also provides the interoperability between the Server Controller layer and the virtual HPC environment built using the Amazon EC2 instances. FReMI contains five different components: *Start*, *wFReDoW Repository*, *FReMI workspace*, *FReMI execution*, and *HPC environment*. *Start* begins the execution of FReMI and *HPC Environment* denotes the virtual cluster on EC2 instances. The remaining components are described below.

The *wFReDoW Repository* contains the *Input/Update Files* repository. This repository stores all files sent by Server Controller layer using the SFTP network protocol. It consists of predocking files, a wFReDoW control file (Figure 6), and different wFReDoW update files (Figure 7).

The *FReMI Workspace* component represents the directory structure used to store the huge volume of data manipulated to execute the molecular docking simulations. The input files placed in the *wFReDoW Repository* are transferred, during FReMI's execution time, to its workspace by the *Parser/Transfer* activity within the *FReMI Execution* set of activities.

The *FReMI Execution* component—the engine of FReMI—contains every procedure invoked to run the middleware. Its source code was written in the C programming language and its libraries. Figure 8 shows the data flow control followed by the *FReMI Execution* component. Basically, the *FReMI Execution* identifies the active snapshots (status A), inserts them in queues of balanced tasks that are created based on subgroup priorities emerging from the P-SaMI data pattern, and submits these queues into the HPC environment. These actions are performed through three activities: *Create Queue*, *Parser/Transfer*, and *Dispatcher/Monitor*.

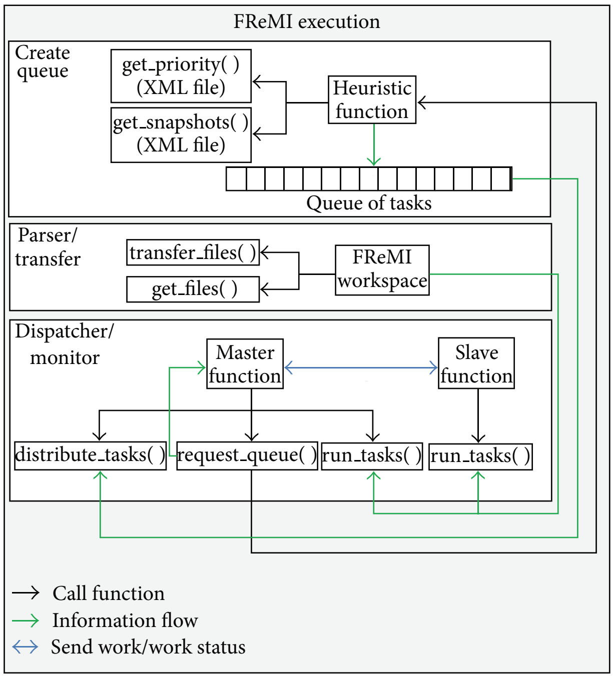


Figure 8: Scheme of the FReMI execution implementation. The *Create Queue*, *Parser/Transfer*, and *Dispatcher Monitor* components include the main functions executed by FReMI. The *Dispatcher/Monitor* component deals with the master-slave paradigm on the EC2 instances.

The *Create Queue* activity produces a number of queues of balanced tasks during FReMI run time based on the information from wFReDoW control file (Figure 6). According to the priorities, this activity uses a heuristic function to determine how many processors from HPC environment will be allocated for each subgroup of snapshots. Furthermore, it uses the status to identify whether a snapshot should be processed or not. For this purpose, the *Create Queue* activity starts calculating the maximum number of snapshots that each queue can support. Thus, the amount of nodes or machines allocated (N) and the amount of parallel tasks (T) executed per node are used to obtain the queue length (Q), with the following equation:

$$Q = N \times T. \tag{1}$$

Afterward, the amount of snapshots per subgroup is calculated in order to achieve the balanced distribution of tasks in every queue created. A balanced queue contains one or more snapshots of an active group. From the subgroup priorities, it is possible to determine the percentage of snapshots to be included in the queues. Thus, subgroups with higher priority are queued before those with lower priority. Equation (2) is used to calculate the amount of snapshots for a balanced queue:

$$S_g = Q \times \left(\frac{P_g}{\sum (P_g)} \right). \tag{2}$$

S_g is the amount of snapshots of the subgroup g that are placed in the queue. Q is the queue length from (1). P_g is the priority of the subgroup g, and $\sum(P_g)$ is the sum of the priorities of all subgroups. From (2) one queue of balanced tasks (B_q) is created with the following equation:

$$B_q = \sum (S_g). \tag{3}$$

The *Parser/Transfer* activity handles and organizes the files sent by the Server Controller layer to its workspace on FReMI. It has three functions: to transfer all files received from Server Controller to the FReMI workspace by means of the *transfer file* function (see Figure 8); to perform a parse on predocking files in order to recognize the FReMI's files directory structure; and to update the parameters of the subgroups of snapshots, when necessary, using the *get files* function. The purpose of this last activity is to maintain FReMI updated with the Server Controller layer.

The functions from the *Dispatcher/Monitor* activity, as shown in Figure 8, are invoked to distribute tasks among the processors/cores from the virtual computer cluster on EC2 Amazon [18] based on the master-slave paradigm [23]. *Slave Function* only runs the tasks while *Master Function*, aside from running tasks, also performs two other functions: *distribute tasks*, which is activated when a node/machine asks for more work; and *request queue*, which is activated when the queue of tasks is empty. Furthermore, to take advantage of the multiprocessing of each virtual machine, we use the hybrid parallel programming model [27]. This model sends bags of tasks among the nodes by means of MPI and it shares out the tasks inside every node by OpenMP parallelization.

3.2. FReMI-Only Execution on Amazon EC2 MPI Cluster. The purpose of executing this set of experiments is to obtain the best MPI/OpenMP performance in the HPC environment on Cloud, which reduces the total elapsed time in the molecular dockings experiments, in order to become the reference to the wFReDoW experiments. For this reason, we have processed the TCL ligand (TCL400 from PDB ID: 1P45A) with two rotatable bonds against all 3,100 snapshots that make up the FFR model using FReMI-only execution. The HPC environment was executed on a scale of 1 to 8 EC2 instances. The number of tasks executed per instance was 32 (from (1): $T = 32$), and the size of the queues of balanced tasks ranged according to the number of instances used. The performance of each FReMI-only experiment versus the number of cores used is shown in Figure 9.

The performance gain obtained using the virtual MPI/OpenMP cluster on Amazon EC2 is substantial when compared to the serial version. We observed that the serial version, which was performed using only one core from an EC2 instance, took around 4 days to execute all 3,100 snapshots from the FFR model, and its parallel execution decreased this time by over 92% for the scales of cores examined. Even though the overall time of the parallel executions was reduced considerably, we also evaluated the speedup and efficiency in the virtual HPC environment to take further advantage of every core scaled during the wFReDoW execution.

The FReMI-only execution is unable to take advantage of more than 48 cores because its efficiency ranges only from 22% to 29% (see Figure 9). Conversely, the cores were

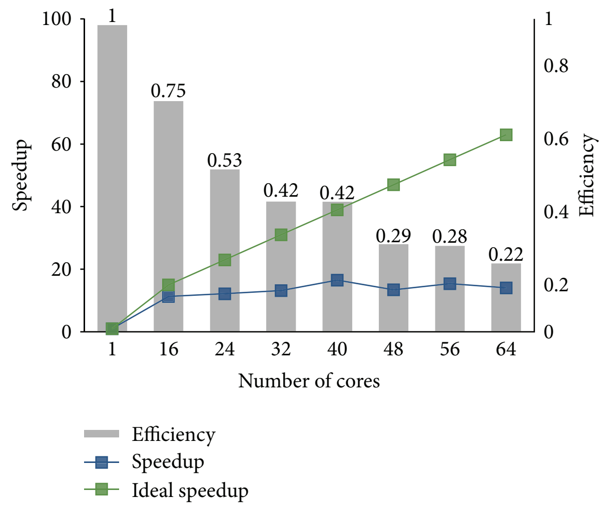


Figure 9: FReMI-only execution performance on Amazon EC2 determined for 3,100 docking tasks running on a scale of 1 to 8 EC2 instances.

well used during the execution when we used less than 40. As can be seen, the best FReMI-only execution efficiency (i.e., 42%) was achieved using 32 and 40 cores from virtual HPC environment. However, the overall execution time spent between them was 7 hours and 28 minutes for 32 cores against 5 hours and 47 minutes for 40 cores. As a consequence of these assessments, the best FReMI-only configuration found in this set of experiments was 5 c1.xlarge EC2 Amazon instances with 8 cores each. It is worth mentioning that this configuration is able to reduce the total docking experiment time (i.e., 5 hours and 47 minutes) about 94% from its reference serial execution time, which took 90 hours and 47 minutes.

3.3. wFReDoW Execution on Amazon EC2 MPI Cluster. The main goal of this set of experiments is to show the performance gains in the molecular docking simulations of an FFR model and the new flexible models produced using wFReDoW. The wFReDoW experiments were conducted using 3,100 snapshots from an FFR InhA model, which are clustered by similarity functions [6], and TCL ligand (TCL400 from PDB ID: 1P45A) with two rotatable bonds. We used only an FFR model and a single ligand to evaluate wFReDoW because our goal was to analyze the performance gain in the docking experiments of FFR models by investigating the best way to coordinate, in one unique environment, all the computational techniques, such as data mining [6], data patterns for scientific workflow [10], cloud computing [18], parallel program, web server and the FReMI middleware. This variety of technological approaches contains their particular features and limits that should be dealt with in order to obtain an efficient wFReDoW implementation, avoiding fault communications, overhead, and idleness issues. Thus, from the best results, we expect that future wFReDoW executions may allow practical use of totally fully flexible receptor models playing in virtual screening of thousands or millions of compounds, which are in virtual chemical structures libraries [3], such as ZINC database [19].

According to the P-SaMI data pattern, the analyses start after a percentage of snapshots has been docked. In these experiments we seek to know how many snapshots are discarded and what the quality is of the RFFR models which are produced for each clustering when the P-SaMI data pattern starts to evaluate after 30%, 40%, 50%, 70%, and 100% of the docked snapshots. When 100% of snapshots are docked P-SaMI does not analyze the docking results. Thus, we perform fifty different kinds of docking experiments—one P-SaMI configuration for each clustering of snapshots. In this sense, Server Controller prepared three different wFReDoW control files—one for each clustering of snapshots generated by [6]—and four different P-SaMI configurations followed the above mentioned percentage.

Figure 10 summarizes the total execution time and the number of snapshots docked and discarded for each wFReDoW experiment. In this Figure, each graph represents the wFReDoW results obtained by running a P-SaMI configuration for each clustering of snapshots, which are represented by 01, 02, and 03 clustering. Every clustering contains 3,100 snapshots from the FFR model, which are grouped from 4 to 6 clusters depending on the similarity function used by [6]. The total time execution for each experiment (one clustering for one P-SaMI configuration) is calculated from the moment the preparation of the wFReDoW control file (in the Server Controller) begins, until the last docking result comes in the Server Controller.

4. Discussion

In this paper we presented the roles of wFReDoW—a cloud-based web environment to faster execute molecular docking simulations of FFR models—and, through its execution, we showed the RFFR models produced. As can be observed in Figure 10, wFReDoW, as well as creating new RFFR models, also speeds up the docking experiments for all cases due to the reduction of docking experiments provided by the P-SaMI data pattern and the simultaneous docking execution performed by the virtual HPC environment. Although we use a small FFR model and only a single ligand, it is clear to see that wFReDoW is a promising tool to start performing molecular docking simulations for new FFR models even using large libraries of chemical structures for the practice of virtual screening.

4.1. wFReDoW Performance. According to [10], the earlier the analysis starts (in this case 30%), the larger the quantity of unpromising snapshots that can be recognized and discarded is. Figure 10 evidences this statement. The wFReDoW results show that when P-SaMI data pattern starts the analyses of the FFR model with 30% of docked snapshots, the number of unpromising snapshots discarded is higher. Additionally, as this percentage increases, the number of unpromising docked snapshots increases as well. Consequently, if the number of docked snapshots decreases, the overall execution time also

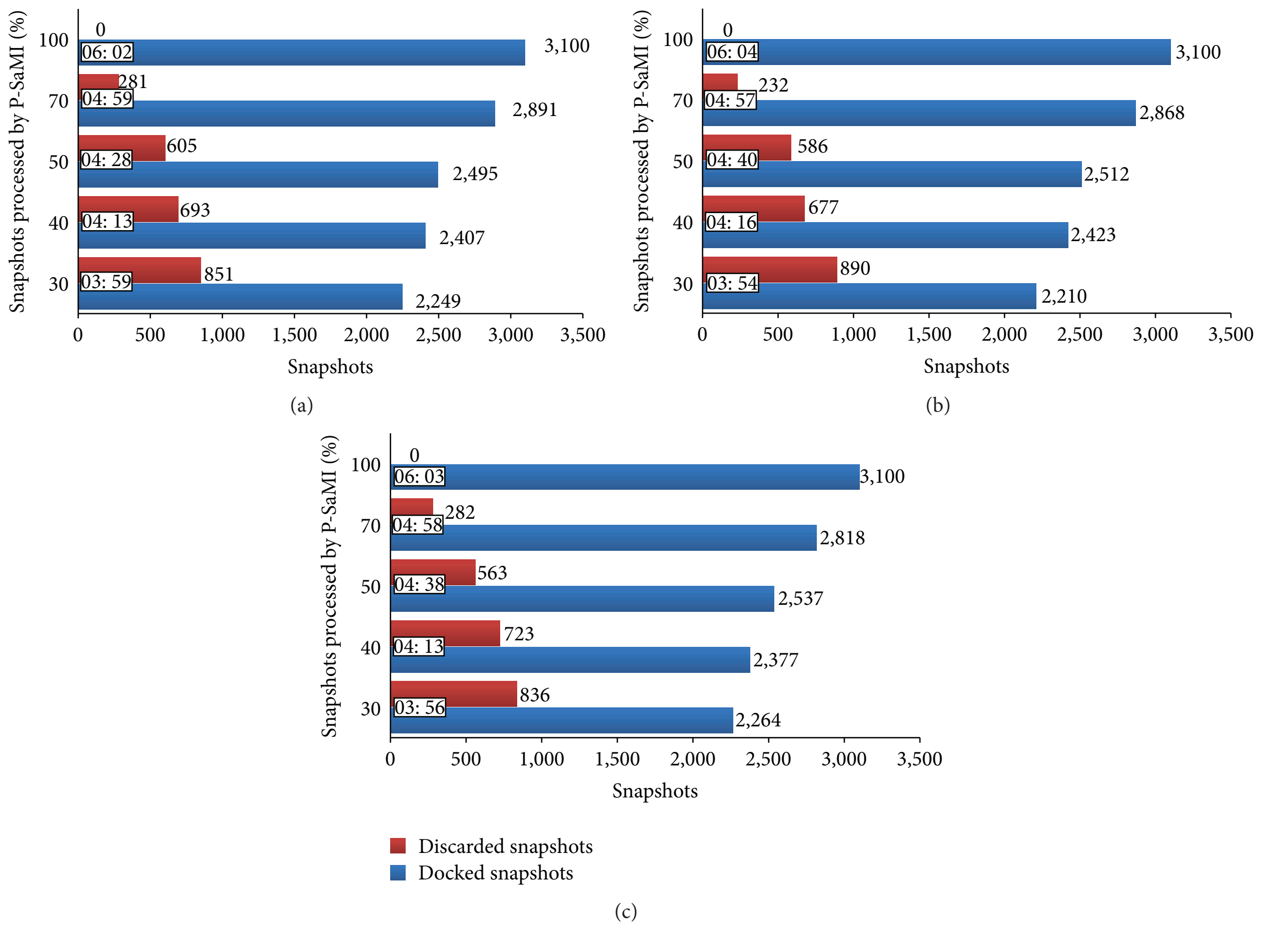


FIGURE 10: Results of the wFReDoW experiments using a different P-SaMI configuration for each clustering of snapshots. Docked snapshots (blue bar) are examples of RFFR models. (a) wFReDoW results for clustering 01. (b) wFReDoW results for clustering 02. (c) wFReDoW results for clustering 03.

decreases. Thus, considering the best run time of wFReDoW, that is, 3 hours and 54 minutes (Figure 10), the gain achieved by the use of P-SaMI showed a fall of 30% from the FReMI-only overall execution (5 hours and 47 minutes).

Another consideration for wFReDoW performance is that FReMI middleware also runs in local cluster infrastructure. However, the efficiency is not the same. We also executed FReMI only using a sample of snapshots from the FFR InhA model on the Atlantica cluster with the intention to compare the performance gains obtained between the virtual and the local cluster infrastructures (Atlantica cluster consists of 10 nodes connected by a fast network system. Each node contains two CPUs Intel Xeon Quad-Core E5520 2.27 GHZ with Hyper-Threading, and 16 GB of RAM, aggregating 16 cores per node. The cluster is connected by a two-gigabit Ethernet network, one for communication between nodes and another for management. Atlantica cluster supplies high performance computational resources for the academic community.) We made several investigations for different nodes and core scales, even for different numbers of tasks executed per node. At the end we found that, in most cases, Amazon EC2 outperforms the Atlantica cluster. For instance, using the same number of cores from Amazon EC2, that is, 5 nodes with 8 cores each, for a sample of 126 snapshots from the FFR model and 16 tasks executed per instance (from (1): $T = 16$), the total execution time was 14.94 minutes for the Atlantica cluster and 8.78 minutes for Amazon EC2. Possibly, this performance difference is because we used the Atlantica cluster in a nonexclusive mode, sharing the cluster's facilities. From this evidence and our previous studies, we concluded that the EC2 configuration bestows itself as a very attractive HPC solution to execute molecular docking simulations of a larger set of snapshots and for different ligands.

4.2. The Quality of the RFFR Models Produced. We showed that the approach used in this study enhances the performance of the molecular docking simulations of FFR models in most cases. However, to make sure that the P-SaMI data pattern selected the best snapshots from the cluster of snapshots used, we verified the quality of the RFFR models built by wFReDoW. Regarding this, we took only the first run of the 25 runs performed by AutoDock 4.2, which contains the best FEB of each docking, to evaluate the produced models. The best docking result of each snapshot was organized according to the percentage of snapshots with

Table 1: Analysis of the wFReDoW results obtained by running a P-SaMI configuration for each clustering of snapshots. Column 1 identifies the three different types of clustering. Column 2 specifies the percentage of docked snapshots after which P-SaMI analysis of the model quality starts. Columns 3, 5, and 7 display the total number of selected snapshots that are in the best 10%, best 20%, and best 30%, respectively. Columns 4, 6, and 8 present the accuracy percentage for the best 10%, 20%, and 30%, respectively.

Clustering	P-SaMI	Best 10%	Accuracy %	Best 20%	Accuracy %	Best 30%	Accuracy %
01	30%	305	98.39	598	96.45	879	94.52
01	40%	305	98.39	600	96.77	887	95.38
01	50%	306	98.71	603	97.25	894	96.13
01	70%	308	99.35	608	98.06	910	97.85
02	30%	302	97.42	593	95.65	871	93.66
02	40%	302	97.42	599	96.61	888	95.48
02	50%	303	97.74	599	96.61	891	95.81
02	70%	308	99.35	612	98.71	913	98.17
03	30%	300	96.77	596	96.13	885	95.16
03	40%	301	97.10	599	96.61	891	95.81
03	50%	303	97.74	604	97.42	898	96.56
03	70%	303	97.74	610	98.39	909	97.74
—	100%	310	100.00	620	100.00	930	100.00

the best FEB values in an ascending order (set of best FEB). Then, we investigated if the selected snapshots belonged to the percentage of this set. As a result we obtained the data described in Table 1 with the number of docked snapshots for each set of best FEB and its respective accuracy.

Based on the data illustrated in Table 1 we can observe that wFReDoW worked well for all P-SaMI analyses. This is evidenced from the computed accuracy in the produced RFFR models, which contain more than 94% of its snapshots within the set of best FEB values. In the clustering 02, for instance, when P-SaMI started the analysis in 70%, wFReDoW worked best, selecting 308 of the 310 best ones, 612 of the 620 best ones, and 913 of the 930 best ones. Whilst, when P-SaMI started the analysis in 30% in the same clustering, wFReDoW selected 302 of the 10% best ones, 593 of the 20% best ones, and 871 of the 30% best ones. Even though wFReDoW selected fewer snapshots in the latter P-SaMI configuration, it represents 97.42%, 95.65% and 93.66% of the 10%, 20%, and 30% best FEB, respectively. The difference between the best and worst wFReDoW selections is slight. However, the difference between them of 1 hour in the total wFReDoW execution time (3 hours and 54 minutes for P-SaMI analysis from 30% against 4 hours and 57 minutes for P-SaMI analysis from 70%) could be a good motivation to start the P-SaMI analyses when only 30% of the snapshots have been docked. Consequently, it also is a promising opportunity for reducing the overall execution time and preserving the quality of the models produced.

It is worth mentioning that wFReDoW is only capable of building an RFFR model, without losing the quality of its original model, if the clustering methods used as input data contain high affinity among the produced clusters of snapshots from [6]. This means that wFReDoW, with its features, is always able to improve the performance. However, for improving the quality of the RFFR models produced, the used clustering also needs to be of a high quality.

4.3. Amazon Cloud. The most significant advantage of shared resources is the guaranteed access time of the resources wherever you are and whenever you need. There is no competition or restrictions for access to the machines. However, it is necessary to pay for as many computing nodes as needed, which are charged at an hourly rate. The rate is calculated for what resources are being used and when; for example, if you do not need computing time, you do not need to pay.

5. Conclusions

The main contribution of our article is wFReDoW, a cloud-based web environment to faster handle molecular docking simulations of FFR models using more than one computational approach cooperatively. wFReDoW includes the P-SaMI data pattern to select promising snapshots and the FReMI middleware that uses an HPC environment on the Amazon EC2 instances to reduce the total elapsed time of docking experiments. The results showed that the best FReMI-only performance decreased the overall execution time by about 94% with its respective serial execution. Furthermore, wFReDoW reduced the total execution time a further 10–30% from FReMI-only best execution without affecting the quality of the produced RFFR models.

There are several possible ways to further improve the efficiency of wFReDoW. One of the biggest limitations for wFReDoW's performance is that the Server Controller layer runs in a web server located outside of Amazon EC2. Even though we posted all docking input files inside wFReDoW repository (inside FReMI layer) in advance, there are still a large number of files that are transferred during the wFReDoW execution. In this experiment, the time taken to transfer these files was irrelevant since our FFR model holds only 3,100 snapshots. However, when using FFR models with hundreds to thousands of snapshots, the time will be increased significantly. A way to enhance the overall performance is by the use of an EC2 instance to host the Server Controller layer. This would greatly reduce the time taken to transfer the files from Server Controller to FReMI. Furthermore, the Server Controller layer could also send only the docking input files from promising snapshots during the wFReDoW execution, contributing to the reduction in the amount of files transferred and in the overall elapsed time.

wFReDoW was tested with a single ligand and an FFR model containing only 3,100 conformations of InhA generated by an MD simulation. MD simulations are now running on tens to hundreds of nanoseconds for the same model. This could produce FFR models with more than 200,000 snapshots! wFReDoW should be tested with such models. Additionally, it would be interesting to make use of other ligands by means of investigation of public databases of small molecules, such as ZINC [19].

Conflict of Interests

The authors declare no conflict of interest.

Acknowledgments

The authors thank the reviewers for their comments and suggestions. This work was supported in part by grants (305984/2012-8 and 559917/2010-4) from the Brazilian National Research and Development Council (CNPq) to Osmar Norberto de Souza and from EU Project CILMI to Duncan D. A. Ruiz. Osmar Norberto de Souza is a CNPq Research Fellow. Renata De Paris was supported by a CNPq M.S. scholarship. FAF was supported by HP-PROFACC M.S. scholarship.

References

[1] N. M. Luscombe, D. Greenbaum, and M. Gerstein, "What is bioinformatics? A proposed definition and overview of the field," *Methods of Information in Medicine*, vol. 40, no. 4, pp. 346–358, 2001.

[2] I. D. Kuntz, "Structure-based strategies for drug design and discovery," *Science*, vol. 257, no. 5073, pp. 1078–1082, 1992.

[3] I. M. Kapetanovic, "Computer-aided drug discovery and development (CADDD): in silico-chemico-biological approach," *Chemico-Biological Interactions*, vol. 171, no. 2, pp. 165–176, 2008.

[4] B. Q. Wei, L. H. Weaver, A. M. Ferrari, B. W. Matthews, and B. K. Shoichet, "Testing a flexible-receptor docking algorithm in a model binding site," *Journal of Molecular Biology*, vol. 337, no. 5, pp. 1161–1182, 2004.

[5] G. M. Morris, R. Huey, W. Lindstrom et al., "Software news and updates AutoDock4 and AutoDockTools4: automated docking with selective receptor flexibility," *Journal of Computational Chemistry*, vol. 30, no. 16, pp. 2785–2791, 2009.

[6] K. S. Machado, A. T. Winck, D. D. A. Ruiz, and O. Norberto de Souza, "Mining flexible-receptor docking experiments to select promising protein receptor snapshots," *BMC Genomics*, vol. 11, supplement 5, article S6, 2010.

[7] K. S. Machado, A. T. Wick, D. D. A. Ruiz, and O. Norberto de Souza, "Mining flexible-receptor molecular docking data," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 1, no. 6, pp. 532–541, 2011.

[8] J. H. Lin, A. L. Perryman, J. R. Schames, and J. A. McCammon, "The relaxed complex method: accommodating receptor flexibility for drug design with an improved scoring scheme," *Biopolymers*, vol. 68, no. 1, pp. 47–62, 2003.

[9] H. Alonso, A. A. Bliznyuk, and J. E. Gready, "Combining docking and molecular dynamic simulations in drug design," *Medicinal Research Reviews*, vol. 26, pp. 531–568, 2006.

[10] P. Hübler, *P-SaMI: self-adapting multiple instances—a data pattern to scientific workflows (in portuguese: P-SaMI: padrão de múltiplas instâncias autoadaptáveis—um padrão de dados para workflows científicos) [Ph.D. thesis]*, PPGCC-PUCRS, Porto Alegre, Brasil, 2010.

[11] R. De Paris, F. A. Frantz, O. Norberto de Souza, and D. D. A. Ruiz, "A conceptual many tasks computing architecture to execute molecular docking simulations of a fully-flexible receptor model," *Advances in Bioinformatics and Computational Biology*, vol. 6832, pp. 75–78, 2011.

[12] X. Jiang, K. Kumar, X. Hu, A. Wallqvist, and J. Reifman, "DOVIS 2.0: an efficient and easy to use parallel virtual screening tool based on AutoDock 4.0," *Chemistry Central Journal*, vol. 2, article 18, 2008.

[13] N. D. Prakhov, A. L. Chernorudskiy, and M. R. Gainullin, "VSDocker: a tool for parallel high-throughput virtual screening using AutoDock on Windows-based computer clusters," *Bioinformatics*, vol. 26, no. 10, pp. 1374–1375, 2010.

[14] R. M. V. Abreu, H. J. C. Froufe, M. J. R. P. Queiroz, and I. C. F. R. Ferreira, "MOLA: a bootable, self-configuring system for virtual screening using AutoDock4/Vina on computer clusters," *Journal of Cheminformatics*, vol. 2, no. 1, article 10, 2010.

[15] B. Collignon, R. Schulz, J. C. Smith, and J. Baudry, "Task-parallel message passing interface implementation of Autodock4 for docking of very large databases of compounds using high-performance super-computers," *Journal of Computational Chemistry*, vol. 32, no. 6, pp. 1202–1209, 2011.

[16] A. P. Norgan, P. K. Coffman, J. A. Kocher, K. J. Katzman, and C. P. Sosa, "Multilevel parallelization of AutoDock 4.2," *Journal of Cheminformatics*, vol. 3, pp. 1–7, 2011.

[17] S. R. Ellingson and J. Baudry, "High-throughput virtual molecular docking with AutoDockCloud," *Concurrency and Computation: Practice and Experience*, 2012.

[18] "Amazon elastic compute cloud," http://aws.amazon.com/ec2/.

[19] J. J. Irwin and B. K. Shoichet, "ZINC—a free database of commercially available compounds for virtual screening," *Journal of Chemical Information and Modeling*, vol. 45, no. 1, pp. 177–182, 2005.

[20] M. R. Kuo, H. R. Morbidoni, D. Alland et al., "Targeting tuberculosis and malaria through inhibition of enoyl reductase: compound activity and structural data," *Journal of Biological Chemistry*, vol. 278, no. 23, pp. 20851–20859, 2003.

[21] E. K. Schroeder, L. A. Basso, D. S. Santos, and O. N. De Souza, "Molecular dynamics simulation studies of the wild-type, I21V, and I16T mutants of isoniazid-resistant *Mycobacterium tuberculosis* enoyl reductase (InhA) in complex with NADH: toward the understanding of NADH-InhA different affinities," *Biophysical Journal*, vol. 89, no. 2, pp. 876–884, 2005.

[22] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th utility," *Future Generation Computer Systems*, vol. 25, no. 6, pp. 599–616, 2009.

[23] C. Banino, O. Beaumont, L. Carter, J. Ferrante, A. Legrand, and Y. Robert, "Scheduling strategies for master-slave tasking on

heterogeneous processor platforms," *IEEE Transactions on Parallel and Distributed Systems*, vol. 15, no. 4, pp. 319–330, 2004.

[24] K. S. Machado, E. K. Schroeder, D. D. Ruiz, E. M. L. Cohen, and O. Norberto de Souza, "FReDoWS: a method to automate molecular dockings simulations with explicit receptor flexibility and snapshots selection," *BMC Genomics*, vol. 12, pp. 2–13, 2011.

[25] I. Raicu, I. Foster, M. Wilde et al., "Middleware support for many-task computing," *Cluster Computing*, vol. 13, pp. 291–314, 2010.

[26] A. T. Winck, K. S. MacHado, O. Norberto de Souza, and D. D. Ruiz, "FReDD: supporting mining strategies through a flexible-receptor docking database," *Advances in Bioinformatics and Computational Biology*, vol. 5676, pp. 143–146, 2009.

[27] R. Rabenseifner, G. Hager, and G. Jost, "Hybrid MPI/OpenMP parallel programming on clusters of multi-core SMP nodes," in *Proceedings of the 17th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP '09)*, pp. 427–436, IEEE Press, Weimar, Germany, February 2009.

Hindawi Publishing Corporation
BioMed Research International
Volume 2013, Article ID 721738, 8 pages
http://dx.doi.org/10.1155/2013/721738

Research Article

GPU-Based Cloud Service for Smith-Waterman Algorithm Using Frequency Distance Filtration Scheme

Sheng-Ta Lee,[1] Chun-Yuan Lin,[1] and Che Lun Hung[2]

[1] *Department of Computer Science and Information Engineering, Chang Gung University, No. 259 Sanmin Road, Guishan, Taoyuan 33302, Taiwan*
[2] *Department of Computer Science & Communication Engineering, Providence University, No. 200 Section 7, Taiwan Boulevard, Shalu, Taichung 43301, Taiwan*

Correspondence should be addressed to Che Lun Hung; clhung@pu.edu.tw

Received 7 December 2012; Accepted 13 March 2013

Academic Editor: Ching-Hsien Hsu

As the conventional means of analyzing the similarity between a query sequence and database sequences, the Smith-Waterman algorithm is feasible for a database search owing to its high sensitivity. However, this algorithm is still quite time consuming. CUDA programming can improve computations efficiently by using the computational power of massive computing hardware as graphics processing units (GPUs). This work presents a novel Smith-Waterman algorithm with a frequency-based filtration method on GPUs rather than merely accelerating the comparisons yet expending computational resources to handle such unnecessary comparisons. A user friendly interface is also designed for potential cloud server applications with GPUs. Additionally, two data sets, H1N1 protein sequences (query sequence set) and human protein database (database set), are selected, followed by a comparison of CUDA-SW and CUDA-SW with the filtration method, referred to herein as CUDA-SWf. Experimental results indicate that reducing unnecessary sequence alignments can improve the computational time by up to 41%. Importantly, by using CUDA-SWf as a cloud service, this application can be accessed from any computing environment of a device with an Internet connection without time constraints.

1. Introduction

The Smith-Waterman (SW) algorithm searches for a sequence database to identify the similarities between a query sequence and subject sequences [1, 2]. However, this algorithm is prohibitively high in terms of time and space complexity; the exponential growth of sequence databases also poses computational challenges [3]. Owing to the computational challenges of the Smith-Waterman algorithm, some faster heuristic solutions (e.g., FASTA [4] and BLAST [5, 6]) have been devised to reduce the time complexity yet degrading the sensitivity of alignment results.

The feasibility of using massive computational devices to enhance the performance of many bioinformatics programs has received considerable attention in recent years, especially many-core devices such as FPGAs [7–9], Cell/BEs [10–12], and GPUs [13]. The recent emergence of GPUs has led to the creation of hundreds of cores, with their computational power having exceeded one TFLOPS and NVIDIA released the CUDA programming environment [14], which allows programmers to use a common programming language (e.g., C/C++) to develop GPU-related applications to enhance the computing performance. Additionally, the feasibility of using GPUs to accelerate the SW database search problem has been widely studied, in which the pioneering work is proposed by Liu et al. [15] to develop SW algorithm using OpenGL for general-purpose GPUs (GPGPU). Following the development of the CUDA programming model, SW-CUDA [16] as the CUDA-based SW solution on GPUs could run on multiple G80 GPUs. However, SW-CUDA distributed the SW algorithm among multicore CPUs and GPUs, making it a highly dependent CPU, owing to their inability to utilize the entire computational power of GPUs. Thereafter, CUDASW++ 1.0 [17], as designed for multiple G200 GPUs, deployed all of the SW computations on GPUs to fully utilize the powerful GPUs. In contrast to previous works, CUDASW++ 2.0 [18] contributes to SW database search problem and optimizes the SIMT abstraction in order to outperform CUDASW++ 1.0. The previous research significantly improves the performance of SW algorithm; in addition, CUDASW++ 2.0 significantly reduces the search time in protein database searches.

However, when using a sequence to query a protein database, biologists do not require all results between the query sequence and all database sequences; however, the similarities are more than at a certain level. Therefore, many computations can be omitted when performing protein database searches if the minimal difference of all alignment combinations can be known in advance, allowing us to omit the extremely different combinations and retain the possible combinations in order to perform the SW alignment. Related research in recent years has heavily focused on establishing the multicore of a multicomputer system. Having received considerable attention in bioinformatics research, cloud computing integrates a large amount of computational power and storage resources, as well as provides different services through a network, such as infrastructure as a service (IaaS), platform as a service (PaaS), and software as a service (SaaS). In these cloud services, users can access desired services without location constraints. Therefore, a cloud service focuses on acquiring services via a remote connection through a network, such as the Amazon EC2 service which is an IaaS and provides various virtual machines with operating systems for users. Other service such as the Google App Engine is a PaaS cloud computing platform for developing and hosting web applications in Google-managed data centers. Other services using the SaaS platform are those such as G-mail or Dropbox services. This cloud computing platform can be viewed as an extended SaaS concept, which refers to customized software, made available via the Internet. Thus, no real computing environments in a local client do not need to be set up since these software applications do not need to ask each end user to manually download, install, configure, run, or use the software applications on their own computing environments. By using cloud services, users can even use a mobile device to complete their tasks, which could only be completed on a PC previously.

This work implements an efficient CUDA-SW program for a SW database search on GPUs. A real-time filtration method based on the frequency distance [19], referred to hereinafter as CUDA-SWf, is also designed to reduce unnecessary computations efficiently. Before the database search, a frequency vector is constructed for the query sequence and the database sequences. Frequency distances are then counted on GPUs for all combinations between query and database sequences. Frequency distance refers to the minimum difference between the query and database sequence, allowing us to record frequency distance in order to determine which combinations should be used to perform a SW alignment and then output the alignment results. Additionally, a friendly user interface (UI) is designed for the potential cloud server with GPUs. Cloud service is combined with GPU computing, in which the SaaS concept through a network is used and a UI is provided to access the service. In our test data sets, the CUDA-SWf can reduce up to 41% of the computational time by comparing with CUDA-SW. Moreover, CUDA-SWf is about 76x faster than its CPU version.

The rest of this paper is organized as follows. Section 2 briefly describes the preliminary concepts for SW algorithm, CUDA programming model, and related works for SW algorithm on GPUs. Section 3 then introduces the method of CUDA-SW algorithm and the implementations of the frequency filtration method. Next, Section 4 summarizes the experimental results. Conclusions are finally drawn in Section 5, along with recommendations for future research.

2. Related Works

2.1. SW Algorithm. The SW algorithm is designed to identify the optimal local alignment between two sequences by estimating the similarity score of an alignment matrix. The computation is based on a scoring matrix such as BLOSUM62 [20] or PAM250 [21] and on a gap-penalty function. Given two sequences S_1 and S_2 whose lengths are l_1 and l_2, respectively, the SW algorithm calculates the similarity score $H(i, j)$ of two sequences ending at positions i and j of S_1 and S_2. Next, $H(i, j)$ is computed, as shown in (1), for $1 \le i \le l_1$, $1 \le j \le l_2$:

$$\begin{aligned} E(i,j) &= \max\{E(i,j-1) - G_e,\ H(i,j-1) - G_i - G_e\},\\ F(i,j) &= \max\{F(i-1,j) - G_e,\ H(i-1,j) - G_i - G_e\},\\ H(i,j) &= \max\{0, E(i,j), F(i,j), H(i-1,j-1)\\ &\qquad + \mathrm{sc}(S_1[i], S_2[j])\}, \end{aligned} \tag{1}$$

where sc denotes the character substitution scoring matrix, G_i represents the gap opening penalty, and G_e refers to the gap extension penalty. A scoring matrix sc gives the substitution rates of amino acids in proteins, as derived from alignments of protein sequences.

The recurrences are initialized as $H(i, 0) = H(0, j) = E(i, 0) = F(0, j) = 0$ for $0 \le i \le l_1$ and $0 \le j \le l_2$. The maximum local alignment score refers to the maximum score in H function. Estimating each cell in H function depends on its left, upper, and upper-left neighbors, as shown by the three arrows in Figure 1. Additionally, this data dependency implies that all cells on the same minor diagonal in the alignment matrix are independent of each other and can be calculated in parallel. Thus, the alignment can be estimated in a minor-diagonal order from the top-left corner to the bottom-right corner in the alignment matrix, where calculating the minor diagonal i only requires the results of minor diagonals $i - 1$ and $i - 2$.

2.2. CUDA Programming Model (CUDA 3.2). Compute unified device architecture (CUDA) is an extension of C/C++, in which users can write scalable multithreaded programs for GPU computing field. The CUDA program is implemented in two parts: host and device. The host is executed by CPU, and the device is executed by GPU. The function executed

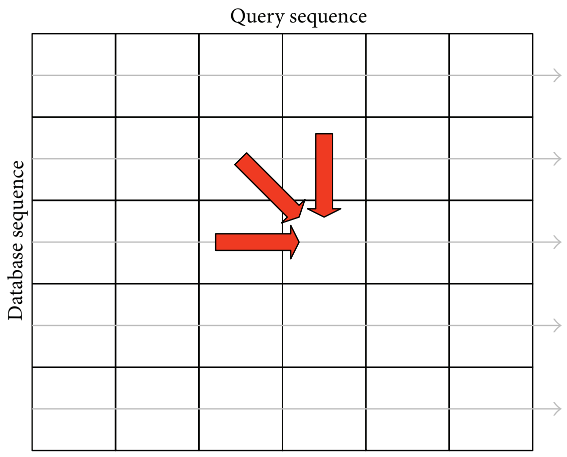


FIGURE 1: Smith-Waterman method.

on the device is called a *kernel*. The kernel function can be invoked as a set of concurrently executing threads, and it is executed by *threads*. These threads are in a hierarchical organization which can be combined into *thread blocks* and *grids*. A grid is a set of independent thread blocks, and a thread block contains many threads. The grid size is the number of thread blocks per grid, and the block size is the number of threads per thread block. Threads in a thread block can communicate and synchronize with each other. Threads within a thread block can communicate through a per-block *shared memory*, whereas threads in different thread blocks fail to communicate or synchronize directly. Besides shared memory, four memory types are per-thread private *local memory*, *global memory* for data shared by all threads, *texture memory*, and *constant memory*. Of these memory types, the fastest memories are the *registers* and shared memories. The global memory, local memory, texture memory, and constant memory are located on the GPU's memory. Besides shared memory accessed by single thread block and registers only accessed by a single thread, the other memory can be used by all of the threads. The caches of texture memory and constant memory are limited to 8 KB per *streaming multiprocessor*. The optimum access strategy for constant memory is all threads reading the same memory address. The texture cache is designed for threads to read between the proximity of the address in order to achieve an improved reading efficiency. The basic processing unit in NVIDIA's GPU architecture is called the *streaming processor*. Many streaming processors perform the computation on GPU. Several streaming processors can be integrated into a streaming multiprocessor. While the program runs the kernel function, the GPU device schedules thread blocks for execution on the streaming multiprocessor. The SIMT scheme refers to threads running on the streaming multiprocessor in a small group of 32, called a *warp*. For instance, NVIDIA GeForce GTX 260, each streaming multiprocessor with 16,384 32-bit registers, has 16 KB of shared memory. The registers and shared memory used in a thread block affect the number of thread blocks assigned to the streaming multiprocessor. Streaming multiprocessor can be assigned up to 8 thread blocks. More details and other version of CUDA can be found in the CUDA programming guides.

2.3. SW Algorithm on GPUs. The several platforms that the SW algorithm has been implemented on include FPGAs, Cell/Bes, and GPUs [7–18]. A query sequence compared with all database sequences is more practical than with a single sequence [22–26] (pairwise comparison). Many works have implemented the SW algorithm on GPUs. Liu et al. [13] first attempted to implement the SW algorithm on a GPU by using OpenGL. The SW algorithm has subsequently been implemented on NVIDIA graphics cards by using CUDA [14, 16]. As for database searches, many efficient methods implement the SW algorithm either by a thread called intertask parallelization or by a thread block called intratask parallelization [27]. By using intertask parallelization [27], this work calculates the similarity score of each pair of input sequences by a single thread. Additionally, a related work developed a method to perform large sequence alignment, not only a similarity score, but also alignment results, with limitations on hardware [28]. Those works improved the performance of the SW database search by using GPUs to reduce the time spent. However, increasing the efficiency of a database search is of priority concern. Performing a protein database search involves finding the most similar protein sequence in a specific database; biologists frequently perform this task. However, many low-quality results are available when performing all database comparisons, indicating the low similarity between query sequence and database sequences. The ability to identify those sequences and distinguish them from deep comparisons will significantly decrease the computational time. Additionally, the ability to qualify a filtration algorithm under this circumstance allows us to reduce computational resources and time. The most similar sequence can be obtained by filtering out the dissimilarity of characters, followed by a series of computations. When sequences are filtered, the level of filtering depends on the length of the query sequence. Longer database sequences are generally preserved to prevent containment of the query sequence. Hence, a longer query sequence implies a more efficient filtering algorithm implemented in this work.

3. CUDA-SW and CUDA-SWf Methods

There are two methods, CUDA-SW and CUDA-SWf, designed and implemented in this work. By integrating the frequency-based filtration method [19], CUDA-SWf performs better by reducing the comparisons than the CUDA-SW. The CUDA-SWf algorithm can be divided into three parts.

Part 1: Inputs Processing (Host, CPU). The inputs of CUDA-SWf are a query sequence and a specific protein database with a large amount of sequences. Before filtration on the device (GPU) is performed, these inputs must be processed in the following steps.

(1) For a query sequence, CUDA-SWf records the query string and the query length, referred to hereinafter as "Q_s" and "Q_l," respectively, followed by an analysis of the string

character structure to construct a frequency vector (FV) for a query sequence named "Q_v." The Q_v is an integer array with 26 indices that record the frequency of each alphabet occurring in a string. Finally, Q_s is stored in a character array, Q_l is stored as an integer, and Q_v is stored in an integer array.

(2) For a protein database, CUDA-SWf scans the entire database and then records the sequence string and sequence length for each database sequence, which is stored in the host memory. All database strings are stored in three one-dimensional arrays, referred to hereinafter as "D_s," "D_l," and "D_e," respectively. Notably, D_s stores all characters of each database sequence; D_l stores the length of each database sequence in D_s; D_e stores the start position of each database sequence in D_s. The sequence length must be shorter than 2,000 characters; owing to that when executing the SW algorithm, some data must be stored in the local memory; in addition, local memory size for each thread is limited. In this step, CUDA-SWf does not construct the frequency vector for each database sequence; owing to that the database contains a large amount of sequences and the cost is high for constructing the frequency vector for each database sequence on the host (sequentially). CUDA-SWf constructs a frequency vector for each database sequence on the device (GPU) when executing the filtration method (run time filtration method).

Part 2: Implementation of the Frequency Filtration Method (Device, GPU). Inputs on the host should first be transferred from the host to the device. Because the query data are used and not updated, the query string, Q_s, query length, Q_l, and query frequency vector, Q_v, are stored in the constant memory. The size of database sequence data (D_s, D_l, and D_e) is too large and stored in the global memory.

When implementing the filtration method, assume that two similar sequences found by SW algorithm may have a certain number of the same characters. As restated, counting the different characters can help to filter out the dissimilar sequences by the enormous difference among character structures. Counting the different characters for each database sequence and query sequence is relatively easy; CUDA-SWf allows a thread to analyze the difference between the query and a database sequence. To analyze the differences between query and database sequences, each thread must construct an FV for a database sequence named "D_v." Similar to Q_v, the D_v value of each database sequence is also an array with 26 indices to store the appeared frequency of each alphabet. Next, counting the sum of the differences between the number of each alphabet in the D_v and Q_v allows us to calculate the differences in their character structure, which is called *frequency distance* (FD). Frequency distance refers to the minimum differences between two sequences. The details of FV and FD can be found in the literature [19].

Finally, a variable "mismatch percentage (MP)" is available to determine whether to perform SW comparisons. Notably, MP refers to the allowed maximum differences ratio between a query and a database sequence; a small value implies a strict filter due to the small FD allowed; otherwise, it implies loose with large FD. When the FD value between a query sequence and a database sequence is greater than MP, it refers to a situation in which the maximum similarity ratio of these two sequences is not satisfied, and this database sequence can be filtered out. When the FD value between a query sequence and a database sequence is lower than MP, it refers to a situation in which the maximum similarity ratio of these two sequences may be satisfied, and this database sequence should make a SW comparison with the query sequence. An attempt is made to prevent database sequences from having too long length, which would make the sequences filtered out due to the large value of FD. When calculating FD, if D_l is longer than Q_l, CUDA-SWf will consider that this database sequence must be compared with the query sequence by a SW algorithm. In doing so, a situation can be avoided in which the query sequence is a local (partial) sequence of the database sequences.

Part 3: SW Comparison (Device and Host). Following selection of the frequency filtration method, CUDA-SWf performs the SW comparison for each selected database sequence with the query sequence. CUDA-SWf uses a thread to make a SW comparison that is called intertask parallelization. To improve the load balance and memory access pattern, CUDA-SWf moves the selected database sequences to the host memory before making SW comparisons for sorting and rearranging the memory pattern for selected database sequences for two subjects: (i) improved load balance for each thread in the same thread block and (ii) coalesced global memory access [17]. In the CUDA programming model, a thread block occupies the resource of a streaming multiprocessor (SM) until all threads in the same thread block complete their computations. To improve the load balance for interftask parallelism, CUDA-SWf must ensure that all threads in the same thread block are assigned a similar length of sequences to achieve a better load balance by sorting the database sequences to assemble the sequences of a similar length, as shown in Figure 2. In order to simply the work in CUDA-SWf, the sorting is performed on CPU. After sorting the database sequences, CUDA-SWf converts the memory configuration from the row major to the column major, as shown in Figure 3 in order to coalesced global memory access. Therefore, all threads in a thread block can access sequences in a continuous memory space. During implementation of the SW algorithm, the alignment sequences must be stored in the global memory and then moved to the local memory of a multiprocessor. The Fermi architecture has per-SM L1 cache and unified L2 cache to service the load/store to global memory; to maximize the performance of cache memory, all threads in the same warp should access the alignment data in global memory to maximize the efficiency of cache memory.

To output the alignment result by the trace back path, the original SW comparison must calculate and store the values in a $M \times N$ matrix (M denotes the query length and N represents the selected database sequence length), explaining why its space complexity is $O(N^2)$, assuming that M is equal to N. In this work, CUDA-SWf only reports the similarity score, not alignment result, and does not need to record the trace back path, explaining why its runtime space complexity to each thread can decrease $O(2N)$ and

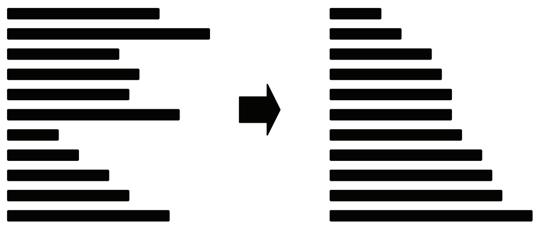


Figure 2: Sorting of the selected sequence to assemble the sequences of a similar length for an improved load balance.

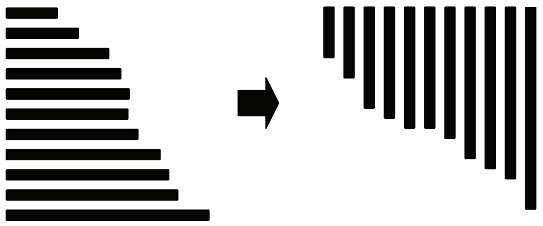


Figure 3: Memory patterns of sequences in the global memory.

suitable for using the intertask parallelization. Because each thread service requires a selected sequence comparison to perform the query sequence, the shared memory cannot load all alignment data. CUDA-SWf thus stores the alignment data of each thread in the local memory. In the Fermi architecture, it is still efficient to store data in the local memory due to L1/L2 cache. Notably, performance of the local memory is not far away from that of the shared memory and is even better than that of the shared memory when the bank conflict occurs in shared memory. The SW comparison of each thread can be divided to three steps: (i) create alignment data: when the comparison is initiated, each thread must create two integer arrays A and B, in which size denotes the length of a selected sequence and stored in the local memory. Owing to that the size limitation of local memory is 16 KB per-thread and the maximum length of database sequence is 2,000. (ii) Row by row comparison: CUDA-SWf can only output the alignment similarity score. Array A is first assigned the value of 0 and, then, each row cell can be calculated simultaneously and the calculated score is stored in array B. Next, the values in array B are moved to array A. Finally, the next row is calculated until all comparisons are finished. (iii) Store the maximum score and final output: when each row comparison is completed, CUDA-SWf confirms the maximum score and records it; finally, CUDA-SWf stores the maximum score in the global memory and, then, moves it to the host memory and finally outputs the database sequences that are similar to the query sequence. The flowchart of CUDA-SWf is shown in Figure 4. The CUDA-SW method is similar to CUDA-SWf without the frequency filtration method.

4. Results

CUDA-SWf was implemented on NVIDIA Tesla C2050 (G400 GPU) with 14 streaming multiprocessors, consisting of 448 CUDA cores and 2.5 GB RAM. The host (CPU) is Intel

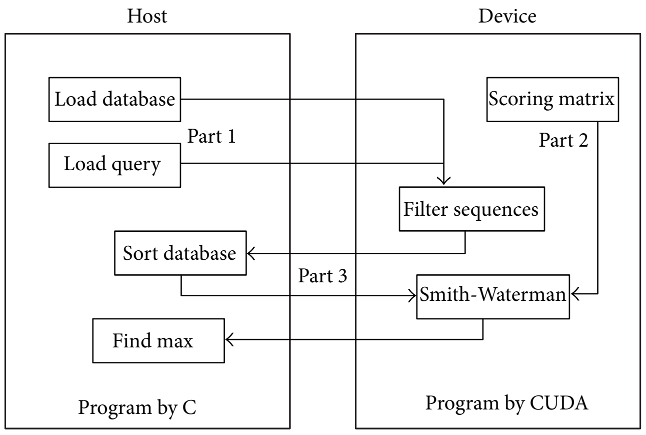


Figure 4: Flowchart of CUDA-SWf.

Xeon E5506 2.13 GHz with 12 GB RAM running on Linux operation system. The protein sequence database was human protein database downloaded from NCBI (http://www.ncbi.nlm.nih.gov/); the query sequences were selected from the H1N1 virus database from the Influenza Virus Resource from NCBI (http://www.ncbi.nlm.nih.gov/genomes/FLU/FLU.html). The testing data sets include the following: (1) 32,799 protein sequences of human with an average length of 555 as the database, and (2) H1N1 virus protein sequences that were randomly selected from the NCBI H1N1 virus database, and the length brackets are 100, 200, 300, 400, 500, 600, and 700 as query sequences. After deleting the protein sequence with length larger than 2,000, there are 32,133 human sequences used in the following tests. The gap open penalty was set to 10.0; the gap extension penalty was set to 2.0; the scoring matrix was BLOSUM62. Next, the MP was set to 10%, 30%, 50%, and 100%, implying the number of different characters between query sequence and database sequences. When the MP is set to 100%, it means that no filtration method is used in CUDA-SWf. The number of threads in a thread block is set to 128; the number of thread blocks depends on the number of sequences that must be compared with query sequences.

Table 1 shows the overall computation time of CPU version of SW algorithm, CUDA-SW, and CUDA-SWf for human protein database and H1N1 virus sequences under various query sequence lengths with MP of 10%. The overall computation time of CUDA-SWf is the sum of computation time in each part. Table 1 indicates that the proposed frequency filtration method can reduce up to 46% of the computation time by filtering out the database sequences in which the minimum different ratio exceeds 10%. Besides, there are two observations in Table 1. First, the computation time increases when the query sequence (H1N1 virus) length increases. The time complexity of SW algorithm is proportional to the query sequence length. Second, the improved ratio increases when the query sequence length increases. The reason is that the number of filtered database sequences is few when the query sequence length is short. When the query sequence length is short, most of database sequences have larger length than it, and they should make SW comparisons in Part 3 of CUDA-SWf.

Table 1: Overall computation time of CPU version of SW algorithm, CUDA-SW, and CUDA-SWf with MP (10%).

H1N1 virus query sequence length (bp)	CPU version of SW (second)	CUDA-SW (second)	CUDA-SWf (second)	Improved ratio (CUDA-SWf versus CUDA-SW)
100	49.91	6.79	6.68	1.62%
200	97.84	7.04	6.25	11.22%
300	145.6	7.27	5.71	21.46%
400	193.62	7.52	5.02	33.24%
500	243.56	7.77	4.71	39.38%
600	293.43	8.02	4.52	43.64%
700	343.31	8.29	4.48	45.96%

Table 2 shows the overall computation time of CUDA-SW for human protein database and H1N1 virus sequences under various MPs with the query length of 700. Table 2 indicates that the number of selected database sequences decreases when the MP decreases. When MP is 100%, there are 32,133 human protein sequences selected to make following SW comparisons; when MP is 10%, only 21.8% of 32,133 human protein sequences can be selected. Therefore, the computation time of CUDA-SWf is reduced from 8.27 to 4.4 (near to 47% improved ratio). When doing the filtration method, extra computation time is needed for CUDA-SWf to construct FV and calculate FD for each database sequence and sorting database sequences on the host. From Table 2, the best score can be found by CUDA-SWf under various MPs. It implies that the frequency filtration method in CUDA-SW is suitable for database search problem. Besides, in Table 2, the worst score found by CUDA-SWf when MP is 10% is closer to that when MP is 100%. This phenomenon indicates that a selected database sequence with low FD may have large difference to a query sequence. Therefore, the FD can be used to filter out the dissimilar sequences; however, it cannot be used to determine the similarity score.

Figure 5 shows the speedup ratio of CUDA-SW and CUDA-SWf by comparing with CPU version of SW algorithm for Human protein database and H1N1 virus sequences under various query sequence lengths with MP of 10%. From Figure 5, the speedup ratios of CUDA-SW range from 7x to 41x; the speedup ratios of CUDA-SWf range from 7x to 76x. The improvement is significant when the query sequence length is larger than 400 due to large number of database sequence filtered out.

For the user interface, this work constructs a workbench for CUDA-SWf with QT Creator 2.4.1 (http://qt.nokia.com//products) on Ubuntu 10.04.1, as shown in Figure 6. As a cross-platform application framework, QT is used to design the same UI for different operating systems then through a network, which transfer the input data to a cloud server. Figure 6 reveals 7 steps to run the CUDA-SWf method.

Step 1 (select the scoring matrix). Notably, the scoring matrix is needed when doing the SW comparison. Five matrices are provided in this work: Blosum50, Blosum62, Blosum80, PAM100, and PAM250.

Table 2: Overall computation time of CUDA-SWf with query sequence length (700).

MP	Number of selected database sequences	Differences (worst score)	Differences (best score)	CUDA-SWf (second)
100%	32,133	3,542	1,169	8.27
50%	17,913	3,542	1,169	5.77
30%	8,578	3,536	1,169	4.63
10%	7,007	3,525	1,169	4.4

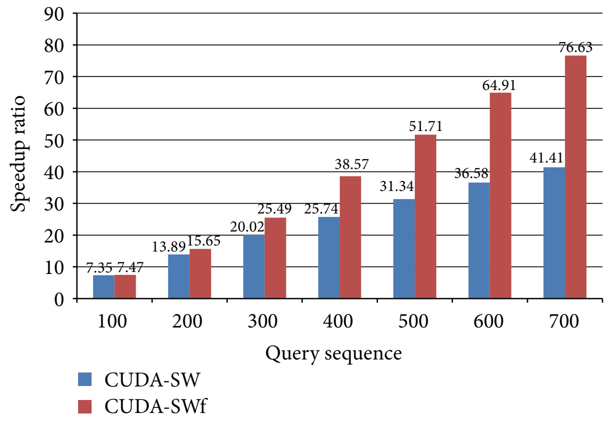
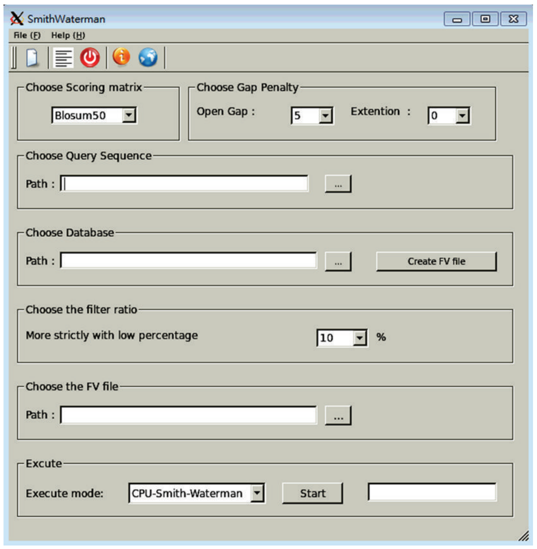


Figure 5: Speedup ratio of CUDA-SW, and CUDA-SWf with MP (10%).

Step 2 (select the gap penalty). Users can select the desired penalty. The open gap penalty range is 5~20, and the gap extension penalty range is 0~10.

Step 3 (select query sequence). Users select a sequence as a query sequence. If a new query file is available, a new file can be created using File(F)->New(N).

Step 4 (select the database). A database can be selected or created by the button "Create FV file." Users can download the database from NCBI. Also, a new database can be created using the button, in order to implement the frequency filtration method. This button creates two files: the first one



Blosum50
Open Gap : 5
Extention : 0
10 %
Execute mode: CPU-Smith-Waterman

FIGURE 6: Workbench of CUDA-SWf.

is the new database sorted by length; and the second one is the FV file by the new database file.

Step 5 (select the filter ratio (MP)). Filter ratio can allow users to determine how strict the CUDA-Swf is used with the filtration method. Users can choose from 10% ~100%. 10% refers to the sequences with those with more than 90% similarity to be computed.

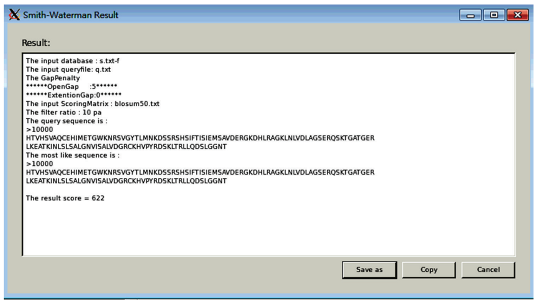
Step 6 (select the FV file). Users select the FV file to be created at Step 4, which helps to execute the frequency filtration method.

Step 7 (execute CUDA-SWf). Two modes can be selected, CPU or GPU. GPU version requires CUDA. Following their execution, the result window is shown (Figure 7). The empty text line displays a message with some errors.

The workbench for CUDA-SWf is freely available to download at http://163.25.101.18/~ppcb/main/research/CUDASWF.html.

5. Conclusions

This work designs and implements a novel CUDA-SWf method to solve the Smith-Waterman database search problem with a frequency-based filtration method and CUDA. The proposed method focuses on the intratask parallelization to calculate the frequency distance and perform Smith-Waterman comparisons on a single GPU. Experimental results demonstrate that the proposed CUDA-SWf method achieves up to 76x speedup ratio under a single GPU for the computation time. Moreover, CUDA-SWf can improve the computational time by up to 41% than CUDA-SW without the frequency filtration method. These results demonstrate that CUDA-SWf can accelerate the Smith-Waterman algorithm on GPUs, and the novel idea is still worth to be



```
The input database : s.txt-f
The input queryfile: q.txt
The GapPenalty
******OpenGap    :5******
******ExtentionGap:0******
The input ScoringMatrix : blosum50.txt
The filter ratio : 10 pa
The query sequence is :
>10000
HTVHSVAQCEHIMETGWKNRSVGYTLMNKDSSRSHSIFTISIEMSAVDERGKDHLRAGKLNLVDLAGSERQSKTGATGER
LKEATKINLSLSALGNVISALVDGRCKHVPYRDSKLTRLLQDSLGGNT
The most like sequence is :
>10000
HTVHSVAQCEHIMETGWKNRSVGYTLMNKDSSRSHSIFTISIEMSAVDERGKDHLRAGKLNLVDLAGSERQSKTGATGER
LKEATKINLSLSALGNVISALVDGRCKHVPYRDSKLTRLLQDSLGGNT

The result score = 622
```

FIGURE 7: Result window of CUDA-SWf.

designed and proposed in order to enhance the performance of CUDA applications.

Authors' Contribution

Sheng-Ta Lee and Chun-Yuan Lin contributed equally to this work.

Acknowledgments

The authors would like to thank the National Science Council of the Republic of China, Taiwan, for partially supporting this research under Contract Nos. NSC100-2221-E-126-007-MY3 and NSC100-2221-E-182-057-MY3. This work was also supported by Chang Gung Memorial Hospital (CMRPD2A0012 and CMRPD2A0013). The authors wish to thank the facilities and financial support from Research Center for Emerging Viral Infections (RECVI) and High Speed Intelligent Communication (HSIC) Research Center in Chang Gung University Taiwan. Ted Knoy is appreciated for his editorial assistance.

References

[1] T. F. Smith and M. S. Waterman, "Identification of common molecular subsequences," *Journal of Molecular Biology*, vol. 147, no. 1, pp. 195–197, 1981.

[2] O. Gotoh, "An improved algorithm for matching biological sequences," *Journal of Molecular Biology*, vol. 162, no. 3, pp. 705–708, 1982.

[3] D. W. Mount, *Sequence and Genome Analysis*, Cold Spring Harbor Laboratory Press, 2004.

[4] W. R. Pearson and D. J. Lipman, "Improved tools for biological sequence comparison," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 85, no. 8, pp. 2444–2448, 1988.

[5] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool," *Journal of Molecular Biology*, vol. 215, no. 3, pp. 403–410, 1990.

[6] S. F. Altschul, T. L. Madden, A. A. Schäffer et al., "Gapped BLAST and PSI-BLAST: a new generation of protein database search programs," *Nucleic Acids Research*, vol. 25, no. 17, pp. 3389–3402, 1997.

[7] T. Oliver, B. Schmidt, D. Nathan, R. Clemens, and D. L. Maskell, "Using reconfigurable hardware to accelerate multiple sequence

alignment with ClustalW," *Bioinformatics*, vol. 21, no. 16, pp. 3431–3432, 2005.

[8] T. Oliver, B. Schmidt, and D. L. Maskell, "Reconfigurable architectures for bio-sequence database scanning on FPGAs," *IEEE Transactions on Circuits and Systems II*, vol. 52, pp. 851–855, 2005.

[9] I. T. S. Li, W. Shum, and K. Truong, "160-fold acceleration of the Smith-Waterman algorithm using a field programmable gate array (FPGA)," *BMC Bioinformatics*, vol. 8, article 185, 2007.

[10] A. Szalkowski, C. Ledergerber, P. Krahenbuhl, and C. Dessimoz, "SWPS3—fast multi-threaded vectorized Smith-Waterman for IBM Cell/B.E. and x86/SSE2," *BMC Research Notes*, vol. 1, p. 107, 2008.

[11] M. S. Farrar, "Optimizing Smith-Waterman for the Cell Broadband Engine," http://farrar.michael.googlepages.com/SW-Cell-BE.pdf.

[12] A. Wirawan, C. K. Kwoh, N. T. Hieu, and B. Schmidt, "CBESW: sequence alignment on the playstation 3," *BMC Bioinformatics*, vol. 9, article 377, 2008.

[13] Y. Liu, W. Huang, J. Johnson, and S. H. Vaidya, "GPU accelerated smith–waterman," in *Proceedings of the Computational Science (ICCS '06)*, vol. 3994 of *Lecture Notes in Computer Science*, Part 4, pp. 188–195.

[14] NVIDIA GPU Computing Documentations, http://docs.nvidia.com/cuda/index.html.

[15] W. Liu, B. Schmidt, G. Voss, and W. Müller-Wittig, "Streaming algorithms for biological sequence alignment on GPUs," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 9, pp. 1270–1281, 2007.

[16] S. A. Manavski and G. Valle, "CUDA compatible GPU cards as efficient hardware accelerators for Smith-Waterman sequence alignment," *BMC Bioinformatics*, vol. 9, supplement 2, article S10, 2008.

[17] Y. Liu, D. L. Maskell, and B. Schmidt, "CUDASW++: optimizing Smith-Waterman sequence database searches for CUDA-enabled graphics processing units," *BMC Research Notes*, vol. 2, article 73, 2009.

[18] Y. Liu, B. Schmidt, and D. L. Maskell, "CUDASW++2.0: enhanced Smith-Waterman protein database search on CUDA-enabled GPUs based on SIMT and virtualized SIMD abstractions," *BMC Research Notes*, vol. 3, article 93, 2010.

[19] T. Kahveci, V. Ljosa, and A. K. Singh, "Speeding up whole-genome alignment by indexing frequency vectors," *Bioinformatics*, vol. 20, no. 13, pp. 2122–2134, 2004.

[20] S. Henikoff and J. G. Henikoff, "Amino acid substitution matrices from protein blocks," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 89, no. 22, pp. 10915–10919, 1992.

[21] M. O. Dayhoff, R. M. Schwartz, B. C. Orcutt et al., "A model of evolutionary change in proteins," in *Atlas of Protein Sequence and Structure*, M. O. Dayhoff, Ed., National Biomedical Research Foundation, 1978.

[22] A. Wozniak, "Using video-oriented instructions to speed up sequence comparison," *Computer Applications in the Biosciences*, vol. 13, no. 2, pp. 145–150, 1997.

[23] T. Rognes and E. Seeberg, "Six-fold speed-up of Smith-Waterman sequence database searches using parallel processing on common microprocessors," *Bioinformatics*, vol. 16, no. 8, pp. 699–706, 2000.

[24] M. Farrar, "Striped Smith-Waterman speeds database searches six times over other SIMD implementations," *Bioinformatics*, vol. 23, no. 2, pp. 156–161, 2007.

[25] B. Alpern, L. Carter, and K. S. Gatlin, "Microparallelism and high-performance protein matching," in *Proceedings of the ACM/IEEE Supercomputing Conference*, pp. 536–551, December 1995.

[26] W. R. Rudnicki, A. Jankowski, A. Modzelewski, A. Piotrowski, and A. Zadrozny, "The new SIMD implementation of the smith-waterman algorithm on cell microprocessor," *Fundamenta Informaticae*, vol. 96, no. 1-2, pp. 181–194, 2009.

[27] Y. Liu, B. Schmidt, and D. L. Maskell, "MSA-CUDA: multiple sequence alignment on graphics processing units with CUDA," in *Proceedings of the 20th IEEE International Conference on Application-Specific Systems, Architectures and Processors (ASAP '09)*, pp. 121–128, July 2009.

[28] A. Khajeh-Saeed, S. Poole, and J. B. Perot, "Acceleration of the Smith-Waterman algorithm using single and multiple graphics processors," *Journal of Computational Physics*, vol. 229, no. 11, pp. 4247–4258, 2010.

Hindawi Publishing Corporation
BioMed Research International
Volume 2013, Article ID 658925, 8 pages
http://dx.doi.org/10.1155/2013/658925

Review Article

Translational Biomedical Informatics in the Cloud: Present and Future

Jiajia Chen,[1,2] Fuliang Qian,[1] Wenying Yan,[1] and Bairong Shen[1]

[1] *Center for Systems Biology, Soochow University, Suzhou 215006, China*
[2] *School of Chemistry and Biological Engineering, Suzhou University of Science and Technology, Suzhou 215011, China*

Correspondence should be addressed to Bairong Shen; bairong.shen@suda.edu.cn

Received 8 December 2012; Accepted 17 February 2013

Academic Editor: Ming Ouyang

Next generation sequencing and other high-throughput experimental techniques of recent decades have driven the exponential growth in publicly available molecular and clinical data. This information explosion has prepared the ground for the development of translational bioinformatics. The scale and dimensionality of data, however, pose obvious challenges in data mining, storage, and integration. In this paper we demonstrated the utility and promise of cloud computing for tackling the big data problems. We also outline our vision that cloud computing could be an enabling tool to facilitate translational bioinformatics research.

1. Introduction

The rate of accumulation of biomolecular data is increasing astonishingly. This information explosion is being driven by the development of low-cost, high-throughput experimental technologies in genomics, proteomics, and molecular imaging, amongst others. Success in the life sciences will depend on our ability to rationally interpret these large-scale, high-dimensional data sets into clinically understandable and useful information, which in turn requires us to adopt advances in informatics. Translational informatics, given the available data resources, is now evolving as a promising methodology that can drive the translation of laboratory data at the bench to health gains at the bedside. This “translation” involves correlating genotype with phenotype, which often requires dealing with information at all structural levels ranging from molecules and cells to tissues and organs, individuals to populations.

2. Translational Bioinformatics: Imperative to Collaborate

According to the scale of investigation, the fields of translational informatics can be roughly classified into four subdisciplines [1]: (1) bioinformatics (molecules and cells); (2) imaging informatics (tissues and organs); (3) clinical informatics (individuals); and (4) public health informatics (populations). Each of the subfields is directed at a particular level of research scale. Table 1 outlines the spectrum of translational bioinformatics activities. The four subfields of translational bioinformatics are compared along several dimensions, including (1) areas of research purpose; (2) data types; (3) informatics tools to support practice.

Bioinformatics traditionally concerns applying computational approaches to the analysis of massive data from genomics, proteomics, metabolomics, and the other “-omic” subfields. Such research might help better comprehend the intricate biological details at molecular and cellular levels. Imaging informatics is focused on what happens at the level of tissues and organs. The essential informatics techniques to extract and manage the biological knowledge from images are summarized in Table 1. At the individual level, clinical bioinformatics is oriented to provide the technical infrastructure to understand clinical risk factors and pathophysiological mechanisms. As for public health informatics, the stratified population of patients is at the center of interest. Such research relies on informatics solutions to study shared risk factors for disease on a population level.

At each of these levels, large amounts of experimental data are being generated. To fully understand a disease phenomenon, however, it is important to gather data at various levels and analyze them in an integrated fashion. While the four

Table 1: Spectrum of translational bioinformatics activities.

Subfields	Research purpose	Data types	Informatics tools
Bioinformatics	Sequencing Structure analysis Expression analysis Phylogenetic analysis Structure modeling	Sequence information Microarray Mass spectrum SNP Haplotypes	Pattern recognition [45–47] Data mining [48–50] Machine learning [47, 51, 52] Visualization [53–55] Automatic annotation [49, 55–58]
Imaging informatics	Image feature identification Image segmentation Image reconstruction Image annotation Image indexing Image visualization	DICOM JPEG TIFF PNG GIF BMP	Content-based image retrieval [59, 60] Natural language processing [61, 62]
Clinical informatics	Clinical decision support Clinical information access Electronic patient record system Disease reclassification	Clinical laboratory results Physical examination Symptoms and signs Patient history Prescriptions	Probabilistic decision-making [63] Expert reasoning system [64–66] Assessment and validation vocabularies [67, 68] Text-parsing tools [69]
Public health informatics	Tracking of infectious diseases Assessment clinical interventions Monitoring disease risk factors	PHCDM-based health information	Access control [70, 71] Information security technology [37, 72] Semantic and syntactic standards [73] Structured data-collection techniques [74, 75]

areas of research differ in their scientific foundations, they nevertheless share a core set of informatics methodologies, such as data acquisition systems, controlled vocabularies, knowledge representation, simulation and modeling, information retrieval, and signal and image processing, which provide a basis for their intersection.

3. Crisis Looms for Multidisciplinary Collaboration

The current push for personalized disease treatment is encouraging bioinformatics to seamlessly integrate data acquired from multiple levels of investigation, from molecular scale to organisms and tissues and further to individuals and populations. To achieve this goal, multidisciplinary collaboration between the fundamental aspects of translational informatics (e.g., bioinformatics, imaging informatics, clinical informatics, and public health informatics) has become essential.

However, the large scale and high dimensionality of data have posed obvious challenges in data mining, storage, and integration. Traditionally, basic research, clinical research, and public health are seen as different worlds based on distinct or incompatible principles. Data transfer, access control, and model building rank are among the most pressing challenges.

4. Cloud Computing to the Rescue

Recent studies and commentaries [2–6] have proposed cloud computing as a solution that addresses many of the limitations mentioned above. Cloud computing is a relatively recent invention. It refers to a flexible and scalable internet infrastructure where processing and storage capacity are dynamically provisioned. The basic idea of cloud computing is to divide a large task into subtasks, which can then be executed on a number of parallel processors. A key technology with the cloud is the virtual machine (VM) that can be prepackaged with all software needed for a particular analysis.

Large utility-computing services have been emerging in the commercial sector, for example, the Amazon Elastic Compute Cloud (EC2) (http://aws.amazon.com/ec2/) [7], and noncommercial public cloud computing platforms also exist to support research, such as the IBM/Google Cloud Computing University Initiative [8] launched by Google and IBM.

Cloud computing infrastructures offer a new way of working. It features a special parallel programming model (e.g., MapReduce [9] designed by Google) to efficiently scale computation to many thousands of commodity machines. These commodity machines form a cluster that can be rented over the internet. Applications in the cloud have also benefited from hadoop (http://hadoop.apache.org/) [10], an open-source implementation of MapReduce. Since it is easy to fine tune and highly portable, Hadoop, together with MapReduce, has been widely used for large-scale distributed data analysis in both academy and industry.

Cloud computing infrastructures offer a highly flexible and economical means of working. Cloud computing provides scalable, flexible access to larger computer processing power and storage and avoids the fixed cost of capital investments in local computing infrastructures, computing maintenance, and personnel. The end users are essentially renting capacity on their demand [11].

Cloud computing allows the sharing of data in real-time collaboration with other users. It addresses one of the challenges related to transferring and sharing data. Researchers

Table 2: Application of cloud computing in bioinformatics research.

Software	Website	Description	Reference
ArrayExpressHTS	http://www.ebi.ac.uk/Tools/rwiki/	RNA-Seq data processing and quality assessment	[76]
Bioscope	http://www.lifescopecloud.com/	Reference-based read mapping	[77]
Cloud-MAQ	http://sourceforge.net/projects/cloud-maq/	Read mapping and assembly	[78]
CloudAligner	http://sourceforge.net/projects/cloudaligner/	Read mapping	[79]
CloudBurst	http://cloudburst-bio.sourceforge.net/	Reference-based read mapping	[6]
CloudCoffee	http://www.tcoffee.org/homepage.html	Multiple sequence alignment	[20]
Contrail	http://contrail-bio.sourceforge.net/	De novo read assembly	[80]
Crossbow	http://bowtie-bio.sourceforge.net/crossbow/	Read mapping and SNP calling	[4]
FX	http://fx.gmi.ac.kr/	RNA-Seq data analysis for gene expression levels and genomic variant calling	[81]
GeneSifter	http://www.geospiza.com/Products/AnalysisEdition.shtml	Customer-oriented NGS data analysis services	[82]
Myrna	http://bowtie-bio.sf.net/myrna/	Differential expression analysis for RNA-Seq data	[83]
PeakRanger	http://www.modencode.org/software/ranger/	Peak caller for ChIP-Seq data	[84]
Roundup	http://rodeo.med.harvard.edu/tools/roundup/	Optimized computation for comparative genomics	[19]
SeqMapReduce	http://www.seqmapreduce.org/	Read mapping	[85]
YunBe	http://tinyurl.com/yunbedownload/	Gene set analysis for biomarker identification	[86]

can store their data in the cloud with high availability. For example, Amazon web services provide free access to many useful data sets, for example, the Ensembl [12] and 1000 Genomes data [13]. In addition, the users can have thousands of on-demand powerful computers ready to run their analysis. To this end, cloud computing has the potential to facilitate large-scale efforts in translational data integration and analysis.

5. Translational Bioinformatics Research in the Cloud

There is considerable enthusiasm in the bioinformatics community to deploy open-source applications in the cloud. Various services provided by cloud-computing vendors are described below.

5.1. Cloud-Based Application in Bioinformatics. Numerous of studies have reported the successful application cloud computing in bioinformatics research. Most of these cloud computing applications deal with high-throughput sequence data analysis. CloudBLAST [14] is the first cloud-based implementation to solve sequence analysis problems. Other projects have since been launched on the cloud. Some initiatives have utilized preconfigured software on cloud systems to support large-scale sequence processing. Some tools are available for sequence alignment, short read mapping, SNP identification, genome annotation, and RNA differential expression analysis, amongst others (Table 2). Efforts in comparative genomics [15–20] and proteomics [21] have also incorporated the cloud to expedite their data processing.

5.2. Cloud-Based Application in Imaging Informatics. The volumes of high-resolution and dynamic imaging data can be estimated to reach petabytes, which indicates that the image reconstruction and analysis is computationally demanding. Cloud-computing is an obvious potential contributor to this end. Image clouds would enable multinational sharing of imaging data, as well as advanced analysis of imaging data away from its place of origin.

Many studies have shown the utility of MapReduce for solving large-scale medical imaging problems in a cloud computing environment. For example, Meng et al. [22] developed an ultrafast and scalable image reconstruction technique for 4D cone-beam CT using MapReduce in a cloud computing environment. Avila-Garcia et al. [23] proposed a cloud computing-based framework for colorectal cancer imaging analysis and research for clinical use. Silva et al. [24] implemented a set of DICOM routers interconnected through a public cloud infrastructure to support medical image exchange among institutions.

Imaging clouds is also making unprecedentedly large-scale imaging research feasible. For example, Euro-Bioimaging [25], a pan-European research infrastructure project aims to deploy a distributed biological and biomedical imaging infrastructure in Europe in a coordinated and harmonized manner. It is expected to offer platforms for storing, remotely accessing, and post-processing imaging data on a large scale.

5.3. Cloud-Based Application in Clinical Informatics. A major challenge for clinical bioinformatics pertains to the accommodation of the range of heterogeneous data into a single, queryable database for clinical or research purposes. Electronic health record (EHR), an integrated clinical information storage systems, has recently emerged and stimulated increased research interest. EHR is a record in digital format that is capable of organizing clinical data by phenotypic categories. An ideal EHR provides complete personal health and medical summary by integrating personal medical information from different sources. The inclusion of genetic imaging and population-based information in EHR has the potential to provide patients with valuable risk assessment based on their genetic profile and family history and to carve a niche for personalized cancer management.

The potential benefits of cloud computing facilitating EHR sharing and EHR integration have been realized. With cloud computing, EHR service could store data into cloud servers. In this way the resources could be flexibly utilized and the operation cost can be reduced. It is envisioned that through the internet or portable media, cloud computing can reduce electronic health record startup expenses, such as hardware, software, networking, personnel, and licensing fees and therefore will promise an explosion in the storage of personal health information online [26–29].

Many previous studies proposed different cloud-based frameworks in an attempt to improve EHR. Among them, Chen et al. [30] proposed a new patient health record access control scheme under cloud computing environments which allows accurate access to patient health record with security and is suitable for enormous multiusers. Chen et al. [31] proposed an EHR sharing and integration system in healthcare clouds. Doukas et al. [32] presented the implementation of a mobile system that enables electronic healthcare data storage, update and retrieval using cloud computing. Rolim et al. [33] proposed a cloud-based solution to automate processes for patients' vital data collection via a network of sensors connected to legacy medical devices and to deliver the data to a medical center's cloud for storage, processing, and distribution. The system provides users with real-time data accessibility labor work to collect, input, and analyze the information. Rao et al. [34] introduced a pervasive cloud-based healthcare application called Dhatri, which leveraged the power of cloud computing and mobile communications technologies to enable physicians to access real-time patient health information from remote areas.

Besides academic researches described above, multiple commercial vendors are competing on this relatively new market. Many world-class commercial companies have heavily invested in the cloud, offering personal medical records services, such as Microsoft's HealthVault [35], which is currently the largest commercial personal health report platform.

5.4. Cloud-Based Application in Public Health Informatics. Public health informatics heavily relies on the data exchange between public health departments and clinical providers. However, public health's information technology systems lack the capabilities to accept the types of data proposed for exchange. Data silos across organizations and programs will present a set of challenges. With cloud services, however, public health applications, software systems, and services would be made available to health departments, therefore facilitating the exchange of specified types of data between different organizations. In addition, through remote hosting and shared computing resources, public health departments could overcome the problem of funding constraints and insufficient infrastructure for public health systems.

6. Concerns and Challenges for the Biomedical Cloud

Cloud computing offers new possibilities for biomedical research, as data can now be easily accessed and shared. Despite the potential gains achieved, there are also several important issues to be addressed before the cloud computing can become more popular. The most significant concerns pertain to information security and data transfer bottlenecks.

6.1. Information Security and Privacy. Lately, many healthcare organizations are looking to move data and applications to a cloud environment. While this offers flexibility and easy access to computational resources, it also introduces security and privacy concerns, which are particularly evident in fields such as clinical informatics and public health informatics. Highly specialized data, such as clinical data from human studies, have exceptional security needs. Hosting such data on publicly accessible servers may increase the risk of security breaches. There are additional privacy concerns relating to personal information. Therefore these data must be posted according to privacy and security rules, such as the Health Insurance Portability and Accountability Act (HIPAA) [36]. For a biomedical cloud to be viable, a secure protection scheme will be necessary to protect the sensitive information of the medical record. For example, sensitive data will have to be encrypted before entering the cloud. Also, only authorized users are allowed to place and acquire sensitive security metadata in the cloud. More advanced encryption measures as well as access control schemes need to be deployed under cloud computing environments.

So far, some research efforts have been made to build security and privacy architectures for biomedical cloud computing [37, 38]. Main cloud service providers (e.g., Amazon, Microsoft, and Google) have also made commitments to develop best practices to protect data security and privacy.

6.2. Data Transfer Bottlenecks. Another major obstacle to moving to the cloud is the time and cost of data transfer. Biomedical research institutions may need to frequently export or import large volumes of data (on the order of terabytes and soon to be petabytes) to and from the cloud. Given the size of the data set, one may find that there is a data transfer bottleneck. Networking bandwidth limitation causes delays in data transfer and incurs high bandwidth costs from service providers. Bandwidth costs might be low for smaller internet-based applications that are not data intensive. However, as applications continue to become more data intensive, these costs can quickly add up, making data transfer costs an important issue. For applications that require substantial data movement on a regular basis, cloud computing currently does not make economic sense.

7. Future Developments and Applications

As discussed above, the future of translational medical bioinformatics will depend on integration of diverse data types of patient characteristics. It is therefore crucial to develop an open, data-sharing environment. We suggest that future initiatives should include (1) development of standards to facilitate informational exchange, (2) integration of databases to allow cross-referencing of multilevel data.

7.1. The Need for Standardized Data Formats. Data exchange across the subfields of translational bioinformatics is often difficult because the data come from heterogeneous informatics platforms and are stored in different formats (e.g., numerical values, free text, and graphical and imaging material). The high dimensionality of potential data types mandates standards to represent data in a uniform manner. To work toward this goal, integrated medical/biological terminologies and ontologies have to be adopted, together with advanced semantic-based models and natural language processing (NLP) techniques to objectively describe medical and biomolecular findings.

Numerous attempts have been made in developing standards for data integration in specialized domains. For example, minimum information about microarray experiment (MIAME) [39] is a standard developed to represent and exchange microarray data. In the field of imaging informatics, existing standards include the foundational model of anatomy clinical community, and digital imaging and communications in medicine (DICOM) [40]. Health level 7 (HL7), clinical data standards interchange consortium (CDISC), systematized nomenclature of medicine (SNOMED) and the international statistical classification of diseases and related health problems (ICD-10) represent the standard for clinical community.

These community-specific standards alone, however, are not sufficient to enable intercommunity data sharing. In this regard, the development of integrated standards will be essential. While it is unlikely to develop a single standard to cover all domains, semantic mapping between terminologies seems more practical. Several pioneering medical informatics projects are underway to define such intercommunity standard. For example, the ACGT project [41] launched by the European Union developed a set of methodological approaches as well as tools and services for semantic integration of distributed multilevel databases.

7.2. The Need for Unified Databases. Currently different layers of biomedical data are stored within databases that are highly distributed, and often not interoperable. Even the databases that hold large data sets are often specialized and fragmented, obstructing the path to information sharing. We need database integration to allow cross-referencing of multilevel data for research or clinical purposes. Opportunities to develop integrated storage systems are increasing as a result of participatory initiatives. Funded by the US National Institutes of Health (NIH), many local platforms in the biomedical informatics space have been established to support data sharing, including informatics for integrating biology and the bedside (i2b2) [42], cancer biomedical informatics grid (caBIG) [43], and biomedical informatics research network (BIRN) [44].

NIH-funded i2b2 Center developed an open-source scalable informatics framework that integrates clinical research data from medical record and genomic data from basic science research. This platform helps better understand the genetic bases of complex diseases. To date, i2b2 has been deployed by over 70 sites internationally. caBIG aims to provide open source standards for data exchange and interoperability in cancer research. At the heart of the caBIG approach is a grid middleware infrastructure called caGrid. caGrid is a service-oriented platform that provides the tools for organizations to integrate data silos, securely share data, and compose analysis pipelines. caBIG enjoys widespread adoption throughout the cancer community. BIRN is an initiative funded by NIH to provide infrastructure, software tools, strategies, and advisory services for sharing biomedical research across disparate groups. These efforts contributed to the transfer and integration of distributed, heterogeneous and multilevel data across the major realms of translational bioinformatics.

8. Conclusion

Biomedical cloud, given the proper architecture, could integrate all the petabytes of available biomedical informatics data in one place and process them on a continuous basis. In this way, we would continuously observe the connections between genotypic profiles and phenotypic data. We can envision that the cloud-supported translational bioinformatics endeavors will promote faster breakthroughs in the diagnosis, prognosis, and treatment of human disease.

Conflict of Interests

The authors declare that there is no conflict of interests.

Authors' Contribution

J. Chen and F. Qian contributed equally to this work.

Acknowledgments

The authors gratefully acknowledge financial support from the National Natural Science Foundation of China Grants (91230117, 31170795, and 91029703), the Specialized Research Fund for the Doctoral Program of Higher Education of China (20113201110015), International S&T Cooperation Program of Suzhou (SH201120), and the National High Technology

Research and Development Program of China (863 program, Grant no. 2012AA02A601).

References

[1] K. A. Kuhn, A. Knoll, H. W. Mewes et al., "Informatics and medicine—from molecules to populations," *Methods of Information in Medicine*, vol. 47, no. 4, pp. 283–295, 2008.

[2] L. D. Stein, "The case for cloud computing in genome informatics," *Genome Biology*, vol. 11, no. 5, article 207, 2010.

[3] M. Baker, "Next-generation sequencing: adjusting to data overload," *Nature Methods*, vol. 7, no. 7, pp. 495–499, 2010.

[4] B. Langmead, M. C. Schatz, J. Lin, M. Pop, and S. L. Salzberg, "Searching for SNPs with cloud computing," *Genome Biology*, vol. 10, no. 11, article R134, 2009.

[5] M. C. Schatz, B. Langmead, and S. L. Salzberg, "Cloud computing and the DNA data race," *Nature Biotechnology*, vol. 28, no. 7, pp. 691–693, 2010.

[6] M. C. Schatz, "CloudBurst: highly sensitive read mapping with MapReduce," *Bioinformatics*, vol. 25, no. 11, pp. 1363–1369, 2009.

[7] *Amazon Elastic Compute Cloud (Amazon EC2)*, http://aws.amazon.com/ec2/.

[8] *IBM/Google Cloud Computing University Initiative*, http://www.ibm.com/ibm/ideasfromibm/us/google/index.shtml.

[9] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[10] *Hadoop*, http://hadoop.apache.org/.

[11] A. Rosenthal, P. Mork, M. H. Li, J. Stanford, D. Koester, and P. Reynolds, "Cloud computing: a new business paradigm for biomedical information sharing," *Journal of Biomedical Informatics*, vol. 43, no. 2, pp. 342–353, 2010.

[12] P. Flicek, M. R. Amode, D. Barrell et al., "Ensembl 2012," *Nucleic Acids Research*, vol. 40, Database issue, pp. D84–D90, 2012.

[13] *1000 Genomes Project*, http://www.1000genomes.org.

[14] A. Matsunaga, M. Tsugawa, and J. Fortes, "CloudBLAST: combining MapReduce and virtualization on distributed resources for bioinformatics applications," in *Proceedings of the 4th IEEE International Conference on eScience (eScience '08)*, pp. 222–229, December 2008.

[15] G. Sudha Sadasivam and G. Baktavatchalam, "A novel approach to multiple sequence alignment using hadoop data grids," *International Journal of Bioinformatics Research and Applications*, vol. 6, no. 5, pp. 472–483, 2010.

[16] D. P. Wall, P. Kudtarkar, V. A. Fusaro, R. Pivovarov, P. Patil, and P.J. Tonellato, "Cloud computing for comparative genomics," *BMC Bioinformatics*, vol. 11, article 259, 2010.

[17] I. Kim, J. Y. Jung, T. F. Deluca, T. H. Nelson, and D. P. Wall, "Cloud computing for comparative genomics with windows azure platform," *Evolutionary Bioinformatics Online*, vol. 8, pp. 527–534, 2012.

[18] G. Zhao, D. Bu, C. Liu et al., "CloudLCA: finding the lowest common ancestor in metagenome analysis using cloud computing," *Protein Cell*, vol. 3, no. 2, pp. 148–152, 2012.

[19] P. Kudtarkar, T. F. Deluca, V. A. Fusaro, P. J. Tonellato, and D. P. Wall, "Cost-effective cloud computing: a case study using the comparative genomics tool, roundup," *Evolutionary Bioinformatics Online*, vol. 6, pp. 197–203, 2011.

[20] P. di Tommaso, M. Orobitg, F. Guirado, F. Cores, T. Espinosa, and C. Notredame, "Cloud-Coffee: implementation of a parallel consistency-based multiple alignment algorithm in the T-coffee package and its benchmarking on the Amazon Elastic-Cloud," *Bioinformatics*, vol. 26, no. 15, pp. 1903–1904, 2010.

[21] B. D. Halligan, J. F. Geiger, A. K. Vallejos, A. S. Greene, and S. N. Twigger, "Low cost, scalable proteomics data analysis using Amazon's cloud computing services and open source search algorithms," *Journal of Proteome Research*, vol. 8, no. 6, pp. 3148–3153, 2009.

[22] B. Meng, G. Pratx, and L. Xing, "Ultrafast and scalable cone-beam CT reconstruction using MapReduce in a cloud computing environment," *Medical Physics*, vol. 38, no. 12, pp. 6603–6609, 2011.

[23] M. S. Avila-Garcia, A. E. Trefethen, M. Brady, F. Gleeson, and D. Goodman, "Lowering the barriers to cancer imaging," in *Proceedings of the 4th IEEE International Conference on eScience (eScience '08)*, pp. 63–70, Indianapolis, Ind, USA, December 2008.

[24] L.A. Silva, C. Costa, and J. L. Oliveira, "DICOM relay over the cloud," *International Journal of Computer Assisted Radiology and Surgery*, 2012.

[25] *Euro-Bioimaging*, http://www.eurobioimaging.eu/.

[26] E. J. Schweitzer, "Reconciliation of the cloud computing model with US federal electronic health record regulations," *Journal of the American Medical Informatics Association*, vol. 19, no. 2, pp. 161–165, 2011.

[27] G. Fernandez-Cardenosa, I. de la Torre-Diez, M. Lopez-Coronado, and J. J. Rodrigues, "Analysis of cloud-based solutions on EHRs systems in different scenarios," *Journal of Medical Systems*, vol. 36, no. 6, pp. 3777–3782, 2012.

[28] J. Haughton, "Look up: the right EHR may be in the cloud. Major advantages include interoperability and flexibility," *Health Management Technology*, vol. 32, no. 2, p. 52, 2011.

[29] J. Kabachinski, "What's the forecast for cloud computing in healthcare?" *Biomedical Instrumentation and Technology*, vol. 45, no. 2, pp. 146–150, 2011.

[30] T. S. Chen, C. H. Liu, T. L. Chen, C. S. Chen, J. G. Bau, and T. C. Lin, "Secure Dynamic access control scheme of PHR in cloud computing," *Journal of Medical Systems*, vol. 36, no. 6, pp. 4005–4020, 2012.

[31] Y. Y. Chen, J. C. Lu, and J. K. Jan, "A secure EHR system based on hybrid clouds," *Journal of Medical Systems*, vol. 36, no. 5, pp. 3375–3384, 2012.

[32] C. Doukas, T. Pliakas, and I. Maglogiannis, "Mobile healthcare information management utilizing Cloud Computing and Android OS," *Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, vol. 2010, pp. 1037–1040, 2010.

[33] C. O. Rolim, F. L. Koch, C. B. Westphall, J. Werner, A. Fracalossi, and G. S. Salvador, "A cloud computing solution for patient's data collection in health care institutions," in *Proceedings of the 2nd International Conference on eHealth, Telemedicine, and Social Medicine (eTELEMED '10)*, pp. 95–99, New York, NY, USA, February 2010.

[34] G. S. V. R. K. Rao, K. Sundararaman, and J. Parthasarathi, "Dhatri—a pervasive cloud initiative for primary healthcare services," in *Proceedings of the 14th International Conference on Intelligence in Next Generation Networks (ICIN '10)*, Berlin, Germany, October 2010.

[35] *MicrosoftHealthVault*, http://www.microsoft.com/en-us/healthvault/.

[36] E. Hansen, "HIPAA (Health Insurance Portability and Accountability Act) rules: federal and state enforcement," *Medical Interface*, vol. 10, no. 8, pp. 96–102, 1997.

[37] M. Freedman, K. Nissim, and B. Pinkas, "Efficient private matching and set intersection," in *Advances in Cryptology-EUROCRYPT 2004*, pp. 1–19, 2004.

[38] V. Danilatou and S. Ioannidis, "Security and privacy architectures for biomedical cloud computing," in *Proceedings of the 10th IEEE International Conference on Information Technology and Applications in Biomedicine (ITAB '10)*, November 2010.

[39] A. Brazma, P. Hingamp, J. Quackenbush et al., "Minimum information about a microarray experiment (MIAME)—toward standards for microarray data," *Nature Genetics*, vol. 29, no. 4, pp. 365–371, 2001.

[40] H. Blume, "DICOM (Digital Imaging and Communications in Medicine) state of the nation. Are you afraid of data compression?" *Administrative Radiology Journal*, vol. 15, no. 11, pp. 36–40, 1996.

[41] M. Tsiknakis, D. Kafetzopoulos, G. Potamias, A. Analyti, K. Marias, and A. Manganas, "Building a European biomedical grid on cancer: the ACGT Integrated Project," *Studies in Health Technology and Informatics*, vol. 120, pp. 247–258, 2006.

[42] I. S. Kohane, S. E. Churchill, and S. N. Murphy, "A translational engine at the national scale: informatics for integrating biology and the bedside," *Journal of the American Medical Informatics Association*, vol. 19, no. 2, pp. 181–185, 2011.

[43] K. K. Kakazu, L. W. Cheung, and W. Lynne, "The Cancer Biomedical Informatics Grid (caBIG): pioneering an expansive network of information and tools for collaborative cancer research," *Hawaii Medical Journal*, vol. 63, no. 9, pp. 273–275, 2004.

[44] J. R. MacFall, W. D. Taylor, D. E. Rex et al., "Lobar distribution of lesion volumes in late-life depression: the Biomedical Informatics Research Network (BIRN)," *Neuropsychopharmacology*, vol. 31, no. 7, pp. 1500–1507, 2006.

[45] P. Baldi and A. D. Long, "A Bayesian framework for the analysis of microarray expression data: regularized t-test and statistical inferences of gene changes," *Bioinformatics*, vol. 17, no. 6, pp. 509–519, 2001.

[46] W. Pan, "A comparative review of statistical methods for discovering differentially expressed genes in replicated microarray experiments," *Bioinformatics*, vol. 18, no. 4, pp. 546–554, 2002.

[47] P. Tamayo, D. Slonim, J. Mesirov et al., "Interpreting patterns of gene expression with self-organizing maps: methods and application to hematopoietic differentiation," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 96, no. 6, pp. 2907–2912, 1999.

[48] D. W. Huang, B. T. Sherman, and R. A. Lempicki, "Systematic and integrative analysis of large gene lists using DAVID bioinformatics resources," *Nature Protocols*, vol. 4, no. 1, pp. 44–57, 2009.

[49] S. Götz, J. M. García-Gómez, J. Terol et al., "High-throughput functional annotation and data mining with the Blast2GO suite," *Nucleic Acids Research*, vol. 36, no. 10, pp. 3420–3435, 2008.

[50] K. G. Becker, D. A. Hosack, G. Dennis et al., "PubMatrix: a tool for multiplex literature mining," *BMC Bioinformatics*, vol. 4, article 61, 2003.

[51] J. Quackenbush, "Computational analysis of microarray data," *Nature Reviews Genetics*, vol. 2, no. 6, pp. 418–427, 2001.

[52] F. M. Selaru, Y. Xu, J. Yin et al., "Artificial neural networks distinguish among subtypes of neoplastic colorectal lesions," *Gastroenterology*, vol. 122, no. 3, pp. 606–613, 2002.

[53] A. J. Saldanha, "Java Treeview—extensible visualization of microarray data," *Bioinformatics*, vol. 20, no. 17, pp. 3246–3248, 2004.

[54] K. Rutherford, J. Parkhill, J. Crook et al., "Artemis: sequence visualization and annotation," *Bioinformatics*, vol. 16, no. 10, pp. 944–945, 2000.

[55] A. A. Porollo, R. Adamczak, and J. Meller, "POLYVIEW: a flexible visualization tool for structural and functional annotations of proteins," *Bioinformatics*, vol. 20, no. 15, pp. 2460–2462, 2004.

[56] W. Fleischmann, S. Möller, A. Gateau, and R. Apweiler, "A novel method for automatic functional annotation of proteins," *Bioinformatics*, vol. 15, no. 3, pp. 228–233, 1999.

[57] L. B. Koski, M. W. Gray, B. F. Lang, and G. Burger, "AutoFACT: an automatic functional annotation and classification tool," *BMC Bioinformatics*, vol. 6, article 151, 2005.

[58] F. Meyer, D. Paarmann, M. D'Souza et al., "The metagenomics RAST server—a public resource for the automatic phylogenetic and functional analysis of metagenomes," *BMC Bioinformatics*, vol. 9, article 386, 2008.

[59] J. R. Smith and S. F. Chang, "VisualSEEk: a fully automated content-based image query system," in *Proceedings of the 4th ACM international conference on Multimedia*, pp. 87–98, November 1996.

[60] D. M. Squire, W. Müller, H. Müller, and T. Pun, "Content-based query of image databases: inspirations from text retrieval," *Pattern Recognition Letters*, vol. 21, no. 13-14, pp. 1193–1198, 2000.

[61] L. Ohno-Machado, V. Bafna, A. A. Boxwala et al., "iDASH: integrating data for analysis, anonymization, and sharing," *Journal of the American Medical Informatics Association*, vol. 19, no. 2, pp. 196–201, 2012.

[62] C. Friedman, "A broad-coverage natural language processing system," *Proceedings/AMIA. Annual Symposium. AMIA Symposium*, pp. 270–274, 2000.

[63] J. M. Beck, W. J. Ma, R. Kiani et al., "Probabilistic population codes for Bayesian decision making," *Neuron*, vol. 60, no. 6, pp. 1142–1152, 2008.

[64] E. S. O'Neill, N. M. Dluhy, and E. Chin, "Modelling novice clinical reasoning for a computerized decision support system," *Journal of Advanced Nursing*, vol. 49, no. 1, pp. 68–77, 2005.

[65] H. Lindgren, "Decision support system supporting clinical reasoning process—an evaluation study in dementia care," *Studies in Health Technology and Informatics*, vol. 136, pp. 315–320, 2008.

[66] J. Diaz Guzman and A. Gomez de la Camara, "The diagnostic process in Neurology: from clinical reasoning to assessment of diagnostic tests," *Neurologia*, vol. 18, supplement 2, pp. 1–2, 2003.

[67] A. Subramanian, B. Westra, S. Matney et al., "Integrating the nursing management minimum data set into the logical observation identifier names and codes system," *Proceedings/AMIA. Annual Symposium. AMIA Symposium*, p. 1148, 2008.

[68] A. W. Forrey, C. J. Mcdonald, G. Demoor et al., "Logical Observation Identifier Names and Codes (LOINC) database: a public use set of codes and names for electronic reporting of clinical laboratory test results," *Clinical Chemistry*, vol. 42, no. 1, pp. 81–90, 1996.

[69] C. Friedman, "Semantic text parsing for patient records," *Medical Informatics*, vol. 8, pp. 423–448, 2005.

[70] G. Eysenbach and A. R. Jadad, "Evidence-based patient choice and consumer health informatics in the Internet age," *Journal of Medical Internet Research*, vol. 3, no. 2, p. E19, 2001.

[71] K. D. Mandl, P. Szolovits, and I. S. Kohane, "Public standards and patients' control: how to keep electronic medical records accessible but private," *British Medical Journal*, vol. 322, no. 7281, pp. 283–287, 2001.

[72] S. A. Buckovich, H. E. Rippen, and M. J. Rozen, "Driving toward guiding principles: a goal for privacy, confidentiality, and security of health information," *Journal of the American Medical Informatics Association*, vol. 6, no. 2, pp. 122–133, 1999.

[73] D. M. Lopez and B. Blobel, "Enhanced semantic interoperability by profiling health informatics standards," *Methods of Information in Medicine*, vol. 48, no. 2, pp. 170–177, 2009.

[74] S. Baxter, A. Killoran, M. P. Kelly, and E. Goyder, "Synthesizing diverse evidence: the use of primary qualitative data analysis methods and logic models in public health reviews," *Public Health*, vol. 124, no. 2, pp. 99–106, 2010.

[75] G. Q. Hu, K. Q. Rao, and Z. Q. Sun, "Identification of a detailed function list for public health emergency management using three qualitative methods," *Chinese Medical Journal*, vol. 120, no. 21, pp. 1908–1913, 2007.

[76] A. Goncalves, A. Tikhonov, A. Brazma, and M. Kapushesky, "A pipeline for RNA-seq data processing and quality assessment," *Bioinformatics*, vol. 27, no. 6, pp. 867–869, 2011.

[77] M. H. Rahimi, *Bioscope: Actuated Sensor Network for Biological Science*, University of Southern California, Los Angeles, Calif, USA, 2005.

[78] A. K. Talukder, S. Gandham, H. A. Prahalad, and N. P. Bhattacharyya, "Cloud-MAQ: the cloud-enabled scalable whole genome reference Assembly application," in *Proceedings of the 7th IEEE and IFIP International Conference on Wireless and Optical Communications Networks (WOCN '10)*, Colombo, Sri Lanka, September 2010.

[79] T. Nguyen, W. Shi, and D. Ruden, "CloudAligner: a fast and full-featured MapReduce based tool for sequence mapping," *BMC Research Notes*, vol. 4, article 171, 2011.

[80] M. C. Schatz, A. L. Delcher, and S. L. Salzberg, "Assembly of large genomes using second-generation sequencing," *Genome Research*, vol. 20, no. 9, pp. 1165–1173, 2010.

[81] D. Hong, A. Rhie, S. S. Park et al., "FX: an RNA-Seq analysis tool on the cloud," *Bioinformatics*, vol. 28, no. 5, pp. 721–723, 2012.

[82] C. Sansom, "Up in a cloud?" *Nature Biotechnology*, vol. 28, no. 1, pp. 13–15, 2010.

[83] B. Langmead, K. D. Hansen, and J. T. Leek, "Cloud-scale RNA-sequencing differential expression analysis with Myrna," *Genome Biology*, vol. 11, no. 8, article R83, 2010.

[84] X. Feng, R. Grossman, and L. Stein, "PeakRanger: a cloud-enabled peak caller for ChIP-seq data," *BMC Bioinformatics*, vol. 12, article 139, 2011.

[85] Y. Li and S. Zhong, "SeqMapReduce: software and web service for accelerating sequence mapping," *Critical Assessment of Massive Data Anaysis (CAMDA)*, vol. 2009, 2009.

[86] L. Zhang, S. Gu, Y. Liu, B. Wang, and F. Azuaje, "Gene set analysis in the cloud," *Bioinformatics*, vol. 28, no. 2, pp. 294–295, 2012.