Hindawi Publishing Corporation
Computational and Mathematical Methods in Medicine
Volume 2015, Article ID 549026, 8 pages
http://dx.doi.org/10.1155/2015/549026

Research Article

G2LC: Resources Autoscaling for Real Time Bioinformatics Applications in IaaS

Rongdong Hu,[1] Guangming Liu,[1,2] Jingfei Jiang,[1] and Lixin Wang[1]

[1] *School of Computer, National University of Defense Technology, Changsha 410073, China*
[2] *National Supercomputer Center, Tianjin 300457, China*

Correspondence should be addressed to Rongdong Hu; rongdonghu@nudt.edu.cn

Received 17 May 2015; Revised 22 June 2015; Accepted 23 June 2015

Academic Editor: Tao Huang

Cloud computing has started to change the way how bioinformatics research is being carried out. Researchers who have taken advantage of this technology can process larger amounts of data and speed up scientific discovery. The variability in data volume results in variable computing requirements. Therefore, bioinformatics researchers are pursuing more reliable and efficient methods for conducting sequencing analyses. This paper proposes an automated resource provisioning method, G2LC, for bioinformatics applications in IaaS. It enables application to output the results in a real time manner. Its main purpose is to guarantee applications performance, while improving resource utilization. Real sequence searching data of BLAST is used to evaluate the effectiveness of G2LC. Experimental results show that G2LC guarantees the application performance, while resource is saved up to 20.14%.

1. Introduction

With significant advances in high-throughput sequencing technologies and consequently the exponential expansion of biological data, bioinformatics encounters difficulties in analysis of vast amounts of data. The need for storing and processing large-scale genome data, easy access to analyses tools, and efficient data sharing and retrieval has presented significant challenges. At present, cloud computing is a promising solution to address these challenges. Cloud computing offers near-infinite amount of resources capacity at a competitive rate and allows users to obtain resources on demand with pay-as-you-go pricing model. The elasticity and enormous capacity of cloud make it possible for bioinformatics applications to return results in a real time way. It will help speed up the bioinformatics research process and relieve the storage pressure of massive data.

IaaS (Infrastructure as a Service), as one important form of cloud computing, mainly leverages the virtualization technology to create multiple VMs (Virtual Machines) on a physical host and can support rapid deployment of large-scale applications [1]. Cloud providers can reduce power consumption by consolidating various applications into a fewer number of physical hosts and switching idle hosts to low-power modes. However, virtualization also creates a new problem. The application performance relies on effective management of VM capacity. One essential requirement of cloud computing is providing reliable QoS defined in terms of SLA (Service Level Agreements). SLA violation will bring economic penalties to cloud providers. Therefore, they always strive to ensure the agreed performance of individual VM. It is nontrivial because of the complexity of applications, various resources usage patterns, shared underlying hardware infrastructure, and the performance correlation and interference among applications.

The focus of this work is on the performance of bioinformatics applications in IaaS. This work tries to take advantage of cloud elasticity to deal with changes in application loads. A resource autoscaling method, G2LC, is proposed to provide suitable processing power for bioinformatics application, to keep up with the changes of sequence length. The method is based on statistical learning load forecasting algorithm. Application performance is guaranteed by adjusting the forecasted results, while minimizing resource usage. Real traces data of BLAST, one of the most widely used bioinformatics programs for sequence searching, is used to evaluate the effectiveness of G2LC. Experimental results show that G2LC can save more than 20% of the resources, while guaranteeing application performance.

The rest of this paper is organized as follows. Section 2 describes the background. Section 3 proposes the detailed design and implementation of G2LC and Section 4 presents the experimental evaluation. Section 5 examines the related work and Section 6 makes the conclusions.

2. Background

According to [2], Real Time Systems (RTS) are those whose correctness depends not only on the logical results but also on the time in which such results are produced. In this type of applications, completion or response is always constrained by time. Failing in accomplishing this requirement could result in serious implications. Depending on the flexibility of such constraints, real time applications are generally classified into hard, firm, and soft. Hard real time applications are those where the nonfulfillment of the time constraints leads to system failure. Firm real time applications have hard constraints, but they allow certain level of tolerance. In the case of soft real time applications, the nonfulfillment of deadlines degrades the performance of system but does not destroy it by failure or crash. In this study, bioinformatics applications will be considered as soft real time applications. Regardless of the sequence length change, the result is required to return as quickly as possible.

Cloud computing is inherently real time and more specifically soft real time. Most existing cloud applications have stringent timing and performance requirements, such as voice and object recognition, image and video retrieval, financial systems, log processing, advertisement placement, and personalized recommendations. These applications are becoming increasingly latency sensitive and operating under demanding workloads that require fast response, for which some violations of the timing constraints are acceptable. For example, if an email system responds slowly, users may switch to another service provider. While the failure of the system to respond quickly may lead to customer churn and reduction on service provider's profits, it does not cause any catastrophic consequences.

On the other hand, cloud computing is also very suitable for RTS. In order to accomplish the time constraints, RTS normally demands large amount of computing resources. Cloud computing can offer this scalability. The virtualization and the resulting decoupling of infrastructure and application offered by cloud make it possible to rapidly scale the infrastructure to meet the resource requirements of the real time applications. Popular social applications, such as Facebook and Twitter, make further enhancement to enable real time communication. Major search engines jump in real time war by providing real time search results. Force.com provides real time integration with external cloud services such as Amazon Web Services, Facebook, Google App Engine, and Twitter.

However, the advent of other critical cloud computing targets such as the improvement of cost efficiency is creating a challenging atmosphere to real time applications. Cloud providers require not only accomplishing performance and time constraints of the applications, but also improving the resources utilization of data centers. The objective is to increase their profits while QoS is guaranteed. This balance is fundamental for the real time cloud.

3. Method: G2LC

3.1. VM Vertical Scaling. In IaaS, applications share the underlying hardware by running in isolated VMs. Each VM, during its initialization, is configured with a certain amount of resources (such as CPU, memory, and disk). A key factor for improving utilization efficiency is resource provisioning. The objective of VM vertical scaling is to ensure that VM capacity is matched with the workload, while overprovisioning wastes costly resources and underprovisioning degrades application performance. Existing virtualization technologies can adjust the capacity of a live VM locally based on time division multiplexing to maximize the resource utilization, also referred to as VM resizing.

Implementation of any policy is accompanied by operating costs. Chen et al. [3] set up a simulated environment and perform a preliminary experiment to illustrate the VM vertical scaling effect on three types of applications (CPU-, memory-, and network I/O-intensive). Experimental results show that the application performance degradation during the VM vertical scaling is smaller than that during VM migration, and the time of performance degradation is also shorter. The VM vertical scaling avoids the unnecessary VM migration by reallocating the spare resources to the heavy-loaded VM in very short time. By comparison, although VM migration can solve the performance problem, it spends much more time to do VM transmission, which generates significant interferences to the other colocated applications. Particularly, the network-intensive application receives serious interference when doing the VM migration, which is because much of the network I/O is preempted by the migration.

This work will adopt VM vertical scaling to adjust the VM processing capacity according to load changes of application. In IaaS virtualization platform, there are many mature tools (such as Xen (http://xenproject.org/), KVM (http://www.linux-kvm.org/page/Main_Page), and VMware (http://www.vmware.com/)) available for system manager to perform the monitoring and vertical scaling operations. For instance, we can use the Xen *xm* command to collect the CPU utilization of VM and use the Xen credit scheduler to set the CPU capacity limit of VM for vertical scaling.

3.2. Load Forecasting. One essential requirement of a real time cloud is providing reliable QoS defined in terms of SLA. The sequences processed by bioinformatics applications often have very different lengths causing dynamic resources usage pattern. The consolidation of VMs can lead to performance degradation when an application encounters an increasing demand resulting in an unexpected rise of resources usage. This may lead to SLA violation—increasing response time. Overprovisioning may help to ensure SLA, but it leads to inefficiency when the load decreases. The optimal strategy is to timely adjust resources provisioning according to the actual demands of the application. One precondition of this approach is to find out the future load.

In this work, we will adopt KSwSVR, proposed in our previous work, as our load prediction method [4]. It is based on statistical learning technology which is suitable for the complex and dynamic characteristics of the cloud computing environment. KSwSVR integrates an improved SVR (Support Vector Regression) algorithm and Kalman smoothing technology and does not require access to the internal details of application. The improved SVR gives more weight to more important data than standard SVR, using the historical information more reasonably. Kalman Smoother is employed to eliminate the noise of resources usage data coming from measurement error. In comparison with AR (Autoregression), BPNN (Back Propagation Neural Networks), and standard SVR, KSwSVR always has the minimum prediction error facing every type of resources and different predicted steps.

3.3. G2LC.

3.3. G2LC. We propose *G2LC* to improve the load forecasting-based resource autoscaling method with two adjustment mechanisms—*global gain* for predicted value and *local compensation* for error.

(i) SLA Definition. Resource utilization is commonly used by data center operators as a proxy for application performance because of the monotonic relationship between them and the fact that utilization is easily measurable at the OS level. In-depth researches on this relationship were also conducted, such as in [5, 6]. As it lies outside the sphere of our work, without loss of generality, we also adopt resource utilization to indicate QoS. In this study, SLA model of application is defined as follows:

$$\frac{1}{i-1}\sum_{j=1}^{i-1} F\left(x_j^{\text{alloc}} > x_j^{\text{use}}\right) = \text{cslaV}_i \le \text{slaV}$$

$$F(y) = \begin{cases} 0, & \text{if } y \text{ is true} \\ 1, & \text{if } y \text{ is false}. \end{cases} \tag{1}$$

x_j^{alloc} and x_j^{use} separately represent actual resources allocation value and real resources usage of application in time interval j. cslaV_i is the average SLA violation rate before interval i. It is the indication of the average QoS for VM and is restrained by slaV which is confirmed after the negotiation between cloud service providers and customers. slaV represents the user's tolerance for performance degradation. It is usually smaller than 5% for real time applications. As long as VM resource utilization is below 100%, that is, $x_j^{\text{alloc}} > x_j^{\text{use}}$, we judge that the resource is enough and there is no SLA violation.

Typically, users rent a VM with a certain capacity from IaaS providers. The resource configuration of the VM is unchanged at run time. This is currently a common practice, but it cannot effectively deal with the changing load. We take it as a comparison object in this work, and the resource saving rate is calculated as follows:

$$\text{resource saving rate} = \frac{\sum_{i=1}^{T}\left(x_{\text{Fixed},i} - x_{\text{G2LC},i}^{\text{alloc}}\right)}{\sum_{i=1}^{T} x_{\text{Fixed},i}}. \tag{2}$$

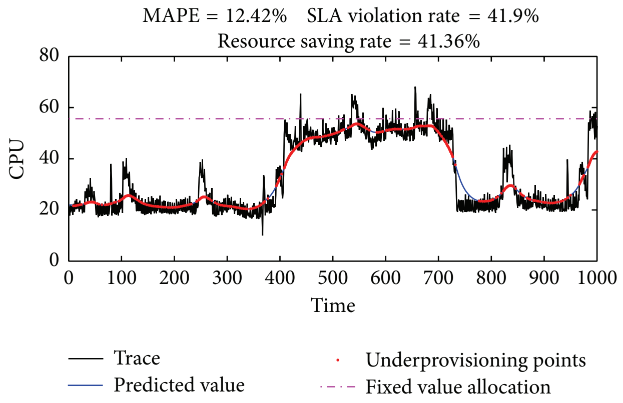

Figure 1: Resource autoscaling directly using predicted value.

(ii) Global Gain for Predicted Value. Even if the load prediction algorithm has high prediction accuracy, can we directly use the predicted value as the ultimate resource supply? We randomly select a piece of data from Google trace (https://code.google.com/p/googleclusterdata/) to analyze this issue. The data is linearly converted before test. Test results are shown in Figure 1.

If the resource allocated to VM is a fixed value, the CPU capacity must be more than 55.65 (horizontal dotted line) to keep the SLA violation rate below 5%. It can be seen from the figure that the loads continue to fluctuate, and the prediction accuracy of KSwSVR is acceptable. The MAPE (Mean Absolute Percentage Error) is only 12.42%. Compared with the fixed value allocation, the method based on the predicted value saves 41.36% of the resource usage in total. But the result of using the predicted value directly for the resource allocation is up to 41.9% SLA violation (red dots, underprovisioning points) rate, which is clearly unacceptable.

The root cause of this problem is the prediction error that any prediction algorithm cannot avoid. When the prediction target is variable, the error will be more notable. Therefore, we need to adjust the predicted value when making the VM scaling scheme.

Researchers from North Carolina State University, NetApp, and Google have studied the relationship between application performance and resource pressure (ratio of the total resource demand to the total resource allocation) [7]. They tested a web server and a database server. The result is shown in Figure 2. When the resource utilization of server exceeds 80%, application performance will seriously decline. If the target is to ensure the SLA violation rate less than 5%, the resource utilization of web servers and database servers must be kept, respectively, at 78% and 77% or less. The main reason is that current level of technology cannot effectively deal with load fluctuations. We need to provide a certain

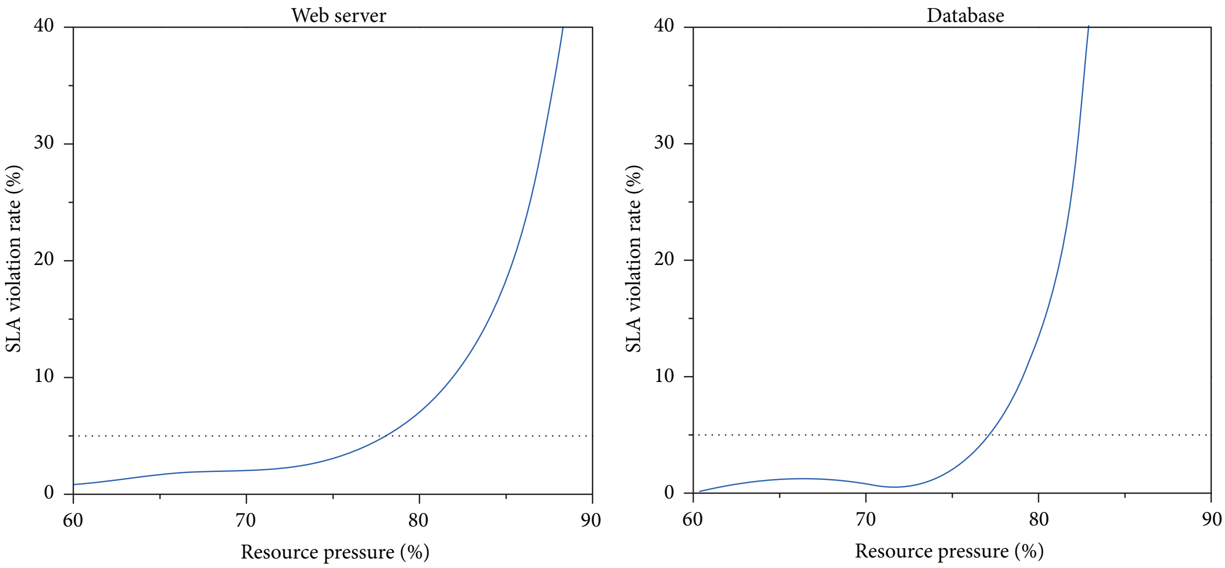

Figure 2: Relationship between application performance and resource pressure.

amount of redundant resources to maintain the application performance.

Inspired by this, we intend to adjust the forecast result $\widehat{x_t}$ by adding a gain coefficient $C_g > 1$, using more resources to meet the needs of real time applications:

$$x_t^{\text{alloc}} = C_g \widehat{x_t}. \quad (3)$$

We need to address the overprovisioning and underprovisioning problems in determining the amount of this part of redundant resources. For this purpose, we use an incremental traversal method to test the impact of the value of C_g on application performance and resource usage. The result is shown in Figure 3.

In the experiment, the value range of C_g is set to [1, 2]. In other words, the resource allocation amount increases from predicted value to its double. As C_g increases, more resources are added to application, and SLA violation rate decreases rapidly (*SLA violation rate*); that is, application performance is quickly enhanced. When $C_g = 1.3$, SLA violation rate drops below 5%. When $C_g > 1.6$, there is no SLA violation event. On the other hand, when C_g increases, the resource consumed by application also increases, and the cost advantage, relative to fixed value allocation, decreases linearly (*resource saving rate*). When C_g increases to 1.7, *resource saving rate* is reduced to zero; that is, the total resources usage of prediction-based dynamic allocation is quite equal to the one of fixed value allocation. If we continue to increase C_g, *resource saving rate* will become negative, and dynamic resource scaling will waste more resources.

If we only consider the resource utilization, the smaller the value of C_g, the better the management effect. However, to meet the application performance requirements, C_g should be set to 1.3 for the load in the experiment. That is, the resource utilization should be maintained at about 77% ≈ 1/1.3. It is consistent with the conclusion of [7] cited before. We also randomly selected a number of other load data from Google to test. The results showed that 1.3 is an ideal gain coefficient.

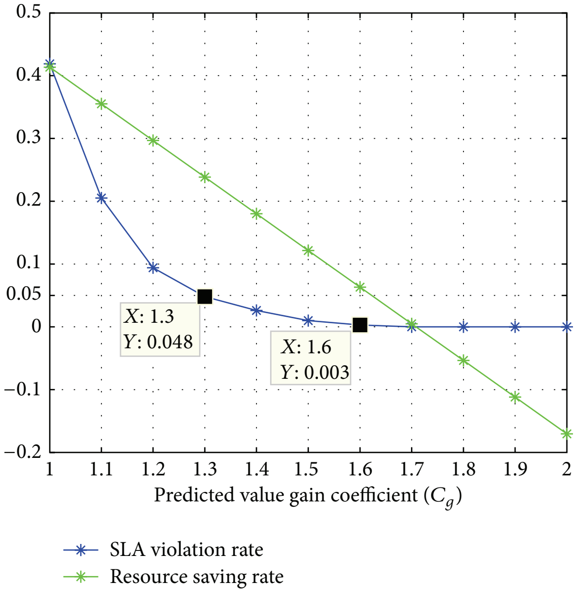

Figure 3: Impact of C_g on application performance and resource usage.

(iii) Local Compensation for Error. If the gain coefficient C_g is set to 1.3, the result of resource autoscaling based on load forecasting is shown in Figure 4.

As can be seen from the figure, after adding the gain coefficient C_g, the overall performance of the application is guaranteed well. Most of the time, because of the introduction of the gain coefficient, there is a significant gap between the resource provisioning curve and trace curve. This gap represents the wasted resource. SLA violation event mainly concentrated near the peak load, as it is difficult for prediction

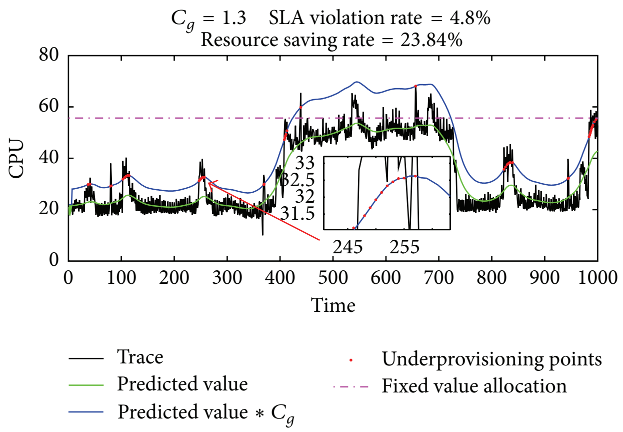

Figure 4: Resource autoscaling with gain coefficient C_g.

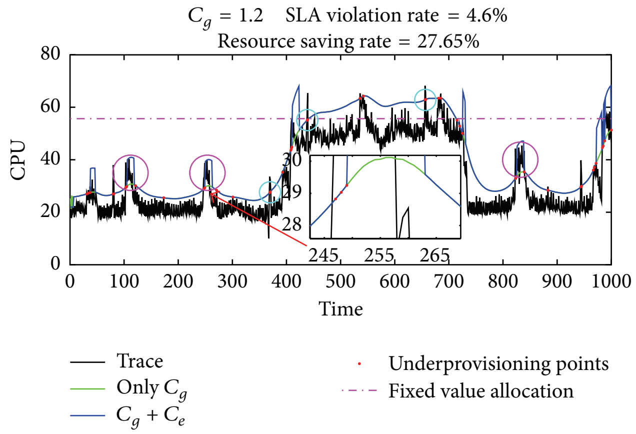

Figure 5: Resource autoscaling with C_g and C_e.

algorithm to deal with the temporary change of object. It is a problem that all current prediction algorithms cannot solve well. Therefore, predicted value needs postprocessing before being used. If we want to further improve the resource utilization, and also to ensure meeting application performance requirements, we need to amend the locality where the SLA violation events happen.

To this end, we further improve the resource scaling method and introduce a local error compensation mechanism to deal with the concentrated underprovisioning. The purpose is to reduce the SLA violation events as much as possible, providing space for further improving the resource utilization (by reducing gain coefficient C_g).

It should be noted that, in practice, IaaS service providers can perceive the VM underproviding based on VM resource utilization, but they cannot learn the specific deficiency. Therefore, we cannot directly use the difference between trace data and predicted value as the error compensation (which is actually the optimal solution).

We continue to introduce a local error compensation coefficient C_e based on (3):

$$\begin{aligned} x_t^{\text{alloc}} &= C_e C_g \widehat{x_t} \\ C_e &= \alpha^{\text{card}(\nu) - T_{\text{rd}}} \\ V &= \left\{ i \mid x_{t-i}^{\text{use}} = x_{t-i}^{\text{alloc}},\ i \in \{1, 2, \ldots, w_e\} \right\}, \end{aligned} \tag{4}$$

where w_e is the windows width; that is, error compensation mechanism will take into account the resource usage of the past w_e periods to develop the resource scaling scheme of the next period. Because the VM resource usage cannot exceed the amount of its total resource, $x_{t-i}^{\text{use}} = x_{t-i}^{\text{alloc}}$ means lack of resource. V is a set of SLA violation events occurring in last w_e periods. card(V) denotes the elements number of set V. T_{rd} is a threshold value. Once the number of SLA violation events within the window exceeds the threshold, the error compensation mechanism will be triggered. $\alpha = 1.1$ is a constant; that is, error compensation amount increases at a rate of 10% each time. Figure 5 shows the dynamic resource autoscaling process under the combinational effect of global gain coefficient C_g and local error compensation coefficient C_e.

On the whole, compared with Figure 4, SLA violation rate reduces from 4.8% to 4.6%. Not only is the application performance improved slightly, but also more resource is saved. *Resource saving rate* (compared with fixed value allocation) increases from 23.84% (only C_g) to 27.65% ($C_g + C_e$). That is, the introduction of reasonable local error compensation mechanism creates space for reducing the gain coefficient C_g, while improving application performance and resource utilization.

Particularly, the most intuitive change generated by the reduction of C_g from 1.3 to 1.2 is the shrink of the gap between the resource provisioning curve and trace curve. In other words, less resource is wasted. Another significant change is that the resource provisioning curve is no longer as smooth as before. The introduction of local error compensation coefficient C_e and window w_e makes the resource management system respond rapidly to the load spikes (as shown in magenta circles). In addition, we introduce T_{rd} as a resource compensation mechanism trigger condition threshold, mainly to avoid the unnecessary compensation operation triggered by glitches (transient load peaks, as shown in cyan circles). It helps to improve resource utilization and enhances the system stability.

Another significant change is that although the number of SLA violation events (marked as red dots) does not reduce much, their distribution has changed a lot. In Figure 4, the SLA violation events concentrated near the peak load. In Figure 5, the red dots become decentralized. For end-user applications, they may encounter sporadic request response delay but will not suffer long time "fake system halt." This will help to improve the user experience.

So far, we have adjusted the predicted values at two levels: global gain and local compensation. The control process of resources autoscaling is shown in Figure 6. Monitor collects VM resources utilization data x_i^{use} and sends them to the predictor. Predictor predicts the resource consumption $\widehat{x_t}$ in the next control cycle. Finally, the resource scaling scheme x_t^{alloc} is figured out after the adjusting of global gain C_g and local compensation C_e.

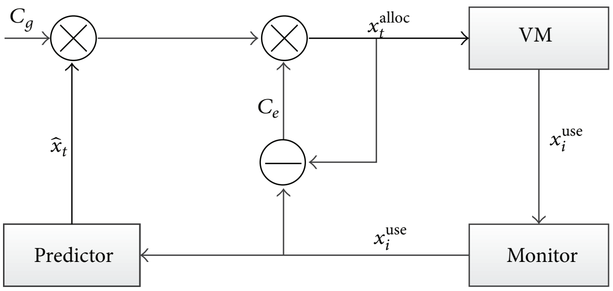

Figure 6: Control process of G2LC.

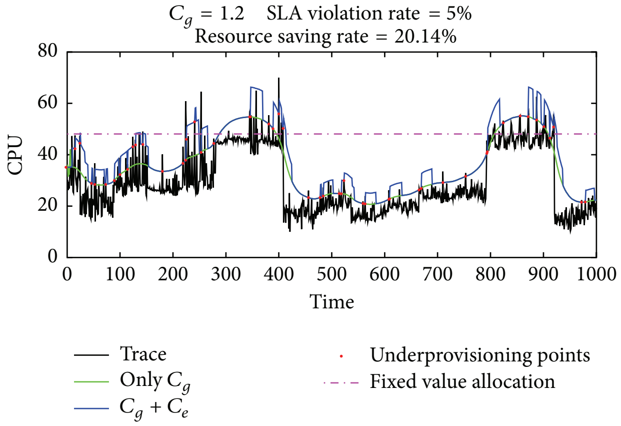

Figure 7: G2LC on BLAST trace.

4. Experiment with BLAST

Focusing on CPU utilization is a good way to understand the application performance, as it is typically proportional to the end-user productivity. Thus, CPU utilization can support greater transparency between cloud service providers and customers. Measuring and reporting CPU utilization is also a simple, affordable, and adequate way of gauging data center efficiency. Most importantly, many of the existing bioinformatics applications are compute-intensive applications. Hence, in this work, we focus on the CPU utilization of application.

It should be noted that the experiments in this paper mainly focus on CPU. So, the experimental conclusions surely apply to CPU-intensive applications. However, G2LC is also applicable to other types of applications (such as the memory-/disk-/network-intensive ones), because the existing virtualization technology can dynamically split these types of resources in a fine-grained way and the forecasting algorithm also applies to these resource objects.

4.1. Experiment Setup. BLAST (Basic Local Alignment Search Tool) [8] is one of the most widely used bioinformatics programs for sequence searching. It addresses a fundamental problem in bioinformatics research. BLAST is an algorithm for comparing primary biological sequence information, such as the amino-acid sequences of different proteins or the nucleotides of DNA sequences. A BLAST search enables a researcher to compare a query sequence with a library or database of sequences and identify library sequences that resemble the query sequence above a certain threshold. The heuristic algorithm it uses is much faster than other approaches, such as calculating an optimal alignment. This emphasis on speed is vital for making the algorithm practical on the huge genome databases currently available.

The effectiveness of G2LC is evaluated by using open real-world BLAST workload traces (http://ammatsun.acis.ufl.edu/amwiki/index.php/Prediction) rather than historical data generated by ourselves for the purpose of giving comparable and reproducible results.

The owners of the traces have comparatively assessed the suitability of several machine learning techniques for predicting spatiotemporal utilization of resources by BLAST [9]. They also extended Predicting Query Runtime (PQR) to the regression problem. BLAST was executed against the nonredundant (NR) protein sequence database from NCBI (National Center of Biotechnology Information). Given an input sequence, BLAST searches a database for similar sequences and calculates the best alignment of the matched sequences. Single nucleotide sequences of varying lengths served as input in the search process.

Different from their work focusing on run time prediction, this study is to guarantee the external performance of bioinformatics applications—returning the results in real time for different size of sequences (load). The traces provide the real search time of each sequence in nonvirtualization environments. If the search time of a sequence is longer, we believe it will need more computing resources in real time environment. Therefore, in the experiment, the search time attribute of sequence in traces is used as the load input. We expect to achieve a real time output by dynamic resource scaling.

4.2. Experimental Results. Experimental results are shown in Figure 7. With the change in the length of the sequence, the processing power of VM must be kept up with this change if we require BLAST to output the search result in a real time model. With the same parameters setting as before, G2LC not only guarantees the overall performance of BLAST, in this context, but also tries to minimize the gap between the resource provisioning curve and trace curve. The overall SLA violation rate is maintained at 5%. Compared with fixed value allocation with the same QoS, G2LC saved up to 20.14% of the resources.

To further analyze the effect of G2LC, we extract a small portion of the data to be described in detail, as shown in Figure 8. The global gain coefficient C_g makes the resource provisioning curve generally above the trace curve, guaranteeing the average performance of BLAST around the acceptable range. The introduction of local error compensation coefficient C_e makes the G2LC respond rapidly to the peak load growth (as shown in cyan circle). On the basis of C_g, C_e further reduces the probability of SLA violation event. In addition, the trigger threshold of resource compensation mechanism, T_{rd}, avoids the unnecessary compensation operation at spikes (as shown in magenta circles). It helps to save resource and enhances the system stability. But in some

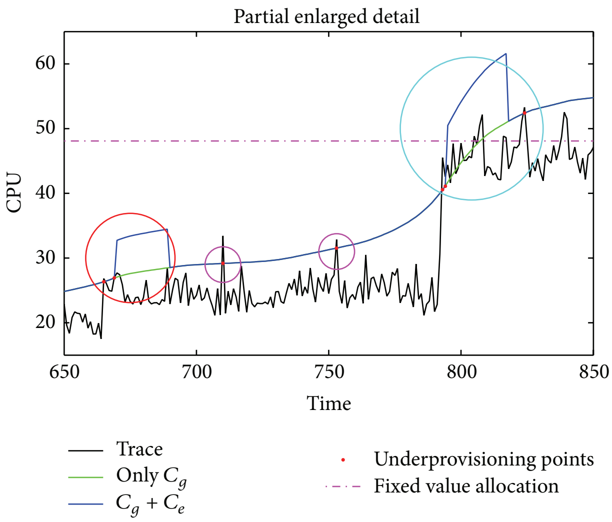

FIGURE 8: G2LC effect in detail.

cases, T_{rd} and window w_e will lead to a slight negative impact. As shown in red circle, once the number of SLA violation events in the window exceeds the threshold T_{rd}, the local compensation mechanism is triggered, regardless of whether the subsequent load increases. If subsequent load did not continue to increase, this will lead to a waste of resources. But its impact on the global effect is very slight, because the duration of resource waste cannot exceed the window width w_e.

5. Related Work

In this section, we briefly review some recent approaches on building and running bioinformatics applications on cloud platform.

As the field of bioinformatics expands, some researches have utilized cloud computing to deliver large computing capacity and on-demand scalability. Crossbow [10] is a cloud enabled tool that combines the aligner Bowtie and the SNP caller SOAPsnp and uses Hadoop for parallel computing. Rainbow [11] is a cloud-based software package that can assist in the automation of large-scale whole-genome sequencing (WGS) data analyses. It copies input datasets to Amazon S3 and utilizes Amazon's computing capabilities to run WGS data analyses pipelines. CloudMap [12] is a pipeline that greatly simplifies the analysis of mutant genome sequences from raw FASTQ reads to mapping plots and short lists of candidate mutations. CloudBurst [13] is a parallel read-mapping algorithm used for next-generation sequence data of the human genome and other reference genomes. It is implemented on a Hadoop-based cluster and aims to optimize the parallel execution. RSD-Cloud [14] runs a comparative genomics algorithm on Amazon EC2 for ortholog calculations across a wide selection of fully sequenced genomes. These projects focus on the solutions of specific problems by developing a tool or method.

Cloud BioLinux [15] is one of the early attempts to simplify the deployment and execution of bioinformatics applications on the cloud. It is a VM configured for high-performance bioinformatics using cloud platforms. At the beginning, over 135 bioinformatics tools have been deployed and configured on the VM. Li et al. [16] presented Hadoop-based applications employed in bioinformatics, covering next-generation sequencing and other biological domains. They described how to obtain an increase in performance by utilizing Hadoop on a cloud computing service and explored different alignment tools and applications that perform sequence alignment. Widera and Krasnogor [17] used Google App Engine computing platform as the computing resource. They introduced the method of building the computer generated protein models used in the protein structure prediction. The proposed Protein Models Comparator is their solution to the problem of large-scale model comparison and can be scaled for different data sizes. Hung and Hua [18] combined two different heterogeneous architectures, software architecture-Hadoop framework and hardware architecture-GPU, to develop a high performance cloud computing service, called Cloud-BLASTP, for protein sequence alignment. Cloud-BLASTP takes advantage of high performance, availability, reliability, and scalability. Cloud-BLASTP guarantees that all submitted jobs are properly completed, even when running job on an individual node or mapper experience failure.

Liu et al. [19] introduced a novel utility accrual scheduling algorithm for real time cloud computing services. The real time tasks are scheduled nonpreemptively with the objective to maximize the total utility. Two different time utility functions were proposed to model the real time applications for cloud computing that need not only to reward the early completions but also to penalize the abortions or deadline misses of real time tasks. Kim et al. [20] investigated power-aware provisioning of VMs for real time services. They modeled a real time service as a real time VM request and provisioned VMs using DVFS scheme. Several schemes were proposed to reduce power consumption by hard real time services and power-aware profitable provisioning of soft real time services.

In comparison to all these studies, G2LC is a general solution for bioinformatics applications to improve resource utilization in IaaS. It does not require access to the internal details of application and executes autoscaling scheme only based on the analysis of application resource utilization data. The purpose is to reduce service costs, while ensuring QoS at the same time.

6. Conclusions

With the rapid growth of next-generation sequencing technologies, more and more data have been discovered and published. To analyze such huge data, the computational performance becomes an important issue. The main focus of this work is on the performance of bioinformatics applications in IaaS. We try to take advantage of cloud elasticity to deal with changes in application loads, making it able to return a result in real time way. A resource autoscaling method, G2LC, is proposed to provide the right amount of resources, to keep up with the changes of sequence length. A statistical learning-based algorithm is adopted for load forecasting. While minimizing resource usage, application performance is guaranteed by adjusting the forecasted results with global gain and local error compensation. Real BLAST trace data is used to evaluate the effectiveness of G2LC. Experimental results show that G2LC can save more than 20% of the resources, while guaranteeing application performance.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgment

This work is supported by the National Natural Science Foundation of China (NSFC) under Grant no. 61303070.

References

[1] Z. Zhang, Z. Li, K. Wu et al., "VMThunder: fast provisioning of large-scale virtual machine clusters," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 12, pp. 3328–3338, 2014.

[2] J. A. Stankovic, "Misconceptions about real-time computing: a serious problem for next-generation systems," *Computer*, vol. 21, no. 10, pp. 10–19, 1988.

[3] W. Chen, X. Qiao, J. Wei, and T. Huang, "A two-level virtual machine self-reconfiguration mechanism for the cloud computing platforms," in *Proceedings of the 9th IEEE International Conference on Ubiquitous Intelligence & Computing (UIC '12) & 9th IEEE International Conference on Autonomic & Trusted Computing (ATC '12)*, pp. 563–570, IEEE, Fukuoka, Japan, September 2012.

[4] R. Hu, J. Jiang, G. Liu, and L. Wang, "Efficient resources provisioning based on load forecasting in cloud," *The Scientific World Journal*, vol. 2014, Article ID 321231, 12 pages, 2014.

[5] S. Kundu, R. Rangaswami, A. Gulati, M. Zhao, and K. Dutta, "Modeling virtualized applications using machine learning techniques," *ACM SIGPLAN Notices*, vol. 47, no. 7, pp. 3–14, 2012.

[6] H. Mi, H. Wang, Y. Zhou, M. R.-T. Lyu, and H. Cai, "Toward fine-grained, unsupervised, scalable performance diagnosis for production cloud computing systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 6, pp. 1245–1255, 2013.

[7] H. Nguyen, Z. Shen, and X. Gu, "Agile: elastic distributed resource scaling for infrastructure-as-a-service," in *Proceedings of the USENIX International Conference on Automated Computing (ICAC '13)*, San Jose, Calif, USA, June 2013.

[8] 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.

[9] A. Matsunaga and J. A. B. Fortes, "On the use of machine learning to predict the time and resources consumed by applications," in *Proceedings of the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, pp. 495–504, IEEE Computer Society, Melbourne, Australia, May 2010.

[10] 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.

[11] S. Zhao, K. Prenger, L. Smith et al., "Rainbow: a tool for large-scale whole-genome sequencing data analysis using cloud computing," *BMC Genomics*, vol. 14, no. 1, article 425, 2013.

[12] G. Minevich, D. S. Park, D. Blankenberg, R. J. Poole, and O. Hobert, "CloudMap: a cloud-based pipeline for analysis of mutant genome sequences," *Genetics*, vol. 192, no. 4, pp. 1249–1269, 2012.

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

[14] 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, no. 1, article 259, 2010.

[15] 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, no. 1, article 42, 2012.

[16] X. Li, W. Jiang, Y. Jiang, and Q. Zou, "Hadoop applications in bioinformatics," in *Proceedings of the 7th IEEE Open Cirrus Summit (OCS '12)*, pp. 48–52, IEEE, Beijing, China, June 2012.

[17] P. Widera and N. Krasnogor, "Protein models comparator: scalable bioinformatics computing on the Google App Engine platform," http://arxiv.org/abs/1102.4293.

[18] C.-L. Hung and G.-J. Hua, "Local alignment tool based on Hadoop framework and GPU architecture," *BioMed Research International*, vol. 2014, Article ID 541490, 7 pages, 2014.

[19] S. Liu, G. Quan, and S. Ren, "On-line scheduling of real-time services for cloud computing," in *Proceedings of the 6th World Congress on Services (SERVICES '10)*, IEEE, Miami, Fla, USA, July 2010.

[20] K. H. Kim, A. Beloglazov, and R. Buyya, "Power-aware provisioning of virtual machines for real-time Cloud services," *Concurrency Computation Practice and Experience*, vol. 23, no. 13, pp. 1491–1505, 2011.