Advances in Intelligent Systems and Computing 645

Elijah Blessing Rajsingh
Jey Veerasamy
Amir H. Alavi
J. Dinesh Peter *Editors*

Advances in Big Data and Cloud Computing

Springer

Advances in Intelligent Systems and Computing

Volume 645

Series editor

Janusz Kacprzyk, Polish Academy of Sciences, Warsaw, Poland
e-mail: kacprzyk@ibspan.waw.pl

The series "Advances in Intelligent Systems and Computing" contains publications on theory, applications, and design methods of Intelligent Systems and Intelligent Computing. Virtually all disciplines such as engineering, natural sciences, computer and information science, ICT, economics, business, e-commerce, environment, healthcare, life science are covered. The list of topics spans all the areas of modern intelligent systems and computing.

The publications within "Advances in Intelligent Systems and Computing" are primarily textbooks and proceedings of important conferences, symposia and congresses. They cover significant recent developments in the field, both of a foundational and applicable character. An important characteristic feature of the series is the short publication time and world-wide distribution. This permits a rapid and broad dissemination of research results.

Elijah Blessing Rajsingh
Jey Veerasamy · Amir H. Alavi
J. Dinesh Peter
Editors

Advances in Big Data and Cloud Computing

Springer

Editors
Elijah Blessing Rajsingh
Department of Computer Sciences Technology
Karunya University
Coimbatore, Tamil Nadu
India

Amir H. Alavi
Department of Civil and Environmental Engineering
University of Missouri
Columbia, MO
USA

Jey Veerasamy
Department of Computer Science, Erik Jonsson School of Engineering and Computer Science
University of Texas at Dallas
Richardson, TX
USA

J. Dinesh Peter
Department of Computer Sciences Technology
Karunya University
Coimbatore, Tamil Nadu
India

ISSN 2194-5357 ISSN 2194-5365 (electronic)
Advances in Intelligent Systems and Computing
ISBN 978-981-10-7199-7 ISBN 978-981-10-7200-0 (eBook)
https://doi.org/10.1007/978-981-10-7200-0

Library of Congress Control Number: 2017957703

Printed on acid-free paper

This Springer imprint is published by the registered company Springer Nature Singapore Pte Ltd.
part of Springer Nature
The registered company address is: 152 Beach Road, #21-01/04 Gateway East, Singapore 189721, Singapore

Contents

About the Editors

Elijah Blessing Rajsingh is currently the Registrar of Karunya University, Coimbatore, India. He received his Ph.D. from Anna University, India in 2005. His research areas include network security, mobile computing, wireless and ad hoc networks, medical image processing, parallel and distributed computing, grid computing, and pervasive computing. He has published a number of articles in reputed journals. He is a member of IEEE, CSI, and ISTE and has served as an advisory board member for various international conferences.

Jey Veerasamy is Director of the Center for Computer Science Education and Outreach and a member of the teaching faculty in the Department of Computer Science, University of Texas at Dallas, USA. Prior to joining the UT Dallas in August 2010, he worked in the US wireless telecom software industry (Nortel and Samsung) for 16 years, while also teaching online courses for several colleges. Having now returned to academia to focus on teaching, he travels to India regularly to offer technical lectures and workshops and shares his US experience with Indian students.

Amir H. Alavi received his Ph.D. degree in Civil Infrastructure Systems from Michigan State University (MSU). He also holds a M.S. and B.S. in Civil and Geotechnical Engineering from Iran University of Science & Technology (IUST). He is currently a senior research fellow in a joint project between the University of Missouri (MU), MSU, in Cooperation with the City Digital at U+ILABS in Chicago on Development of Smart Infrastructure. He is on the editorial board of several journals and is serving as ad-hoc reviewer for many indexed journals. He is among the Google Scholar 300 most cited authors within civil engineering domain (citation > 4100 times; h-index = 34). More, he is selected as the advisory board of Universal Scientific Education and Research Network (USERN), which belongs to all top 1% scientists and the Nobel laureates in the world.

J. Dinesh Peter is currently working in the Department of Computer Sciences Technology at Karunya University, Coimbatore. He received his Ph.D. in Computer Science from the National Institute of Technology Calicut. His research focus areas

include Big Data, image processing, and computer vision. He has authored several publications for reputed international journals. Further, he is a member of IEEE and CSI and has served as session chair and delivered plenary speeches for various international conferences and workshops.

An Ontology-Based Approach for Automatic Cloud Service Monitoring and Management

Kirit J. Modi, Debabrata Paul Chowdhury and Sanjay Garg

Abstract Cloud computing provides an efficient, on-demand, and scalable environment for the benefit of end users by offering cloud services as per service level agreement (SLA) on which both user and cloud service providers are mutually agreed. As the number of cloud users is increasing day by day, sometimes cloud service providers unable to offer service as per SLA, which results in SLA violation. To detect SLA violation and to fulfill the user requirements from the service provider, cloud services should be monitored. Cloud service monitoring plays a critical role for both the customers and service providers as monitoring status helps service provider to improve their services; at the same time, it also helps the customers to know whether they are receiving the promised QoS or not as per the SLA. Most existing cloud service monitoring frameworks are developed toward service provider side. This raises the question of correctness and fairness of monitoring mechanism; on the other hand, if monitoring is applied at user side, then it would become overhead to the clients. To manage such issues, an ontology-based Automatic Cloud Services Monitoring and Management (ACSMM) approach is proposed, where cloud service monitoring and management would be performed at the cloud broker, which is an intermediate entity between the user and service provider. In this approach, when SLA violation is detected, it sends an alert to both clients and service providers and generates the status report. Based on this status report, broker automatically reschedules the tasks to reduce further SLA violation.

Keywords Cloud service monitoring · Service Level Agreement
Cloud service · Ontology · Rescheduling

K. J. Modi (✉) · D. P. Chowdhury
U V Patel College of Engineering, Ganpat University, Gujarat, India
e-mail: kiritmodi@gmail.com

D. P. Chowdhury
e-mail: debabrata130891@gmail.com

S. Garg
Nirma University, Gujarat, India
e-mail: gargsv@gmail.com

E. B. Rajsingh et al. (eds.), *Advances in Big Data and Cloud Computing*,
Advances in Intelligent Systems and Computing 645,
https://doi.org/10.1007/978-981-10-7200-0_1

1 Introduction

In the era of Internet of Things (IoT) and cloud computing, IT resources, such as server, software, bandwidth, and network have been delivered by the service provider to customer as a service through Web known as cloud services. When hundreds of thousands of servers are connected together; then it produces massive, shared capacity for computing that can be provided through software, storage, and infrastructure. Nowadays, people make use of the services through a particular application, such as Gmail, Dropbox, or Facebook. In cloud computing, the service is acquired on an as-needed basis. When cloud service provider offers services to the customers, it is equally important to measure the quality of the service offered by the service provider. Cloud services are offered to users based on the legal agreement made between service provider and user known as Service Level Agreement (SLA). Due to economic benefits of cloud computing, all small and large organizations are moving toward the cloud-based solution [1]. Thus, SLA management is one of the biggest issues in the cloud computing environment. To detect the SLA violation and what Quality of Services (QoS) is offered by the service provider, monitoring of the cloud services needs to be performed. Cloud service monitoring plays a critical role for both the user and service providers in the sense that the monitoring status helps service provider to improve their services at the same time helps the customer to know whether they are receiving the promised QoS or not as per the SLA. There are several commercial and open-source cloud service monitoring tools in the usage, but all of them are service-provider-specific so they create the question of unfairness because monitoring is performed by the sevice provide side. This motivates us to design and develop a fair cloud service monitoring and management system.

Contribution: In this paper, an ontology-based Automatic Cloud Services Monitoring and Management (ACSMM) approach is proposed, in which cloud service monitoring and management is applied at cloud broker level using SLA and ontology. The term automatic defines the ability to monitor and manage the cloud services without any kind of human interference during the process. We develop a SLA ontology model for the semantic description of the QoS parameters. Our approach automatically monitors the cloud services and sends alerts, when SLA is violated and automatically takes reactive actions to reduce the further SLA violation.

2 Preliminary Concepts

In this section, we introduce the preliminary concepts related to cloud service monitoring, QoS model and ontology to understand the present problem.

2.1 Cloud Service Life Cycle

Before seeing the cloud service monitoring, it is necessary to understand the concept of cloud service life cycle [2]. The existing software development models, such as, waterfall model or spiral model, are not suitable for the cloud environment because these existing models require more human which makes it time-consuming for both customers and service providers. As we know, the main characterisitcs of the cloud computing are scalability, elasticity, on-demand service; thus, the conventional software development models are not suitable for the cloud environment. As a result, cloud service life cycle [2] concept is introduced, which consists of following five phases.

- **Requirements**: In the service requirements phase, the consumer specifies the technical or functional requirements and non-functional requirements of the services, that they want to consume. As a result, they issue Request for Service (RFS).
- **Service Discovery**: In the service discovery phase, the RFS generated in the previous phase is used to find the service providers that meet the technical or functional and non-functional requirements of the service.
- **Service Negotiation**: In the service negotiation phase, discussion between the service provider and customer regarding the service delivered is carried-out. Based on the discussion, the key outcome of this phase is known as service level agreement (SLA).
- **Service Composition**: Sometimes some complex requirements of customers cannot be fulfilled by single service provider. These types of requirements can be fulfilled by two or more than two service providers. Thus, two or more than two service providers are combined together to meet the complex requirements and provide a single composite service to the customers.
- **Service Consumption and Monitoring**: In this phase, the services are delivered to the consumers based on the SLA. After the services are provided to the consumer, it is necessary to regularly monitor the status of delivered service to check whether delivered services meet the functional and non-functional goals of the customers as specified in the SLA.

2.2 SLA

SLA [3] is a legal agreement between service provider and customer, in which services provided by the service provider are formally defined. It also specifies the action that could be taken in case of violation. In SLA, the key objectives are known as Service Level Objective (SLO). There is always confusion between the SLA and SLO. SLA is whole agreement, which includes time, location, and cost; whereas SLO contains only key objective or Key Performance Indicators (KPI),

which can be measured. The examples of the SLO are throughput, availability, response time, etc. To describe the service level agreement, Web Service Level Agreement [WSLA] [4] is used, which is based on the XML language.

2.3 Ontology

An ontology [2] is a data model that represents knowledge as a set of concepts within a domain and their relationship between these concepts. The two standards that govern the construction of the ontology are Resource Description Framework (RDF) and Web Ontology Language (OWL). In addition to these standards, ontology is made up of two main components: classes and relationships. The aim of the ontology is to understand the domain knowledge at the same time use and share that knowledge for various applications. Ontology helps to automate the various phases of cloud service life cycle. Ontology is the key component of the Semantic Web. The usage of ontologies allows meaning oriented information processing and interoperability support.

2.4 QoS Model

QoS parameters, such as availability, throughput, and response time, are considered as part of SLA in this work, which are defined as below.

- Availability: Availability [3] represents the idea of anywhere and anytime access to services. Availability is calculated by the formula presented as follows:

$$\text{Availibility} = \frac{(\text{Commited hour} - \text{Outage hour}) * 100}{\text{Commited hour}} \tag{1}$$

- Throughput: Throughput [3] represents the performance of tasks performed by a computing service over a particular time period. Throughput is calculated by the formula presented as follows:

$$\text{Throughput} = \frac{\text{Number of task executed}}{\text{Execution time of all tasks} + \text{Total Delay of the all tasks}} \tag{2}$$

- Response time: It is the time taken by a request until the arrival of the response at the requesting interface. The response time [3] of a task can be calculated as follows:

$$\text{Response time} = \text{Finish Time of task} - \text{Submission Time of task} \quad (3)$$

The above-defined preliminary concepts are applied by us to design and develop the cloud service monitoring and management system. To understand the importance of cloud service monitoring, we have presented the related work carried out by various researchers in the following section.

3 Related Work

In this section, we present the work related to cloud service monitoring published by various researchers by highlighting their key contributions as follows.

Joshi et al. [2] described a process to automate each phase of the cloud service life cycle using ontology. Authors implemented the cloud storage prototype, in which they automate cloud service discovery and cloud service consumption phase, but they haven't implemented the automation in negotiation and monitoring phases. This inspires us to propose a framework that automates the monitoring phase of the cloud service life cycle. In [5], authors discussed the requirements of SLA in cloud computing in detail and also discussed the different SLA metrics in cloud computing. The existing cloud service monitoring is based on some benchmark tests, which are not so much accurate to find the performance of the cloud services. Rehman and Zia [6] proposed a cloud service monitoring framework based on user feedback. Though, this solution is reliable and accurate but it has no solution in case of SLA violation. Khandelwal et al. [7] designed lightweight, scalable cloud service monitoring framework that provides correct and up to date measurements of performance parameter of the application. In this framework, it only measures the performance parameters and does not verify it with SLA. Sahai et al. [8] proposed an automated SLA monitoring engine for the Web services. The limitation of this approach that it only monitors the SLA defined in author-specific SLA definition language. Frey et al. [3] described the SLA life cycle, where authors discussed the general and specific KPIs, which help customers in the negotiation phase during the creation of SLA. This work helps us to understand the key QoS parameters that are defined in SLA. Mohamed et al. [9] proposed a mechanism for SLA violation detection without specifying the reactive action part when SLA is violated. Vaitheki and Urmela [10] presented an algorithm of rescheduling of resources for the SLA violation reduction, which helps us to automate the reactive action when SLA violation is detected by rescheduling of task to the lightly loaded virtual machines. Singh et al. [11] implemented an automatic resource management technique called

STAR based on SLA to provide better customer satisfaction, and they also compared STAR architecture with relevant resource management technique.

From the above literature study, we have observed that the cloud services monitoring is an important task to prevent the SLA violation; as a result, the performance of the cloud services could be improved. We have seen that the existing work on cloud service monitoring and management is service provider specific in most cases, where monitoring is performed by the service provider which raises the question of fairness of SLA violation. This motivates us to propose the automatic cloud service monitoring and management framework, in which monitoring is done by an intermediate entity which is popularly known as cloud broker. The next section present the proposed framework with necessary details.

4 Automatic Cloud Service Monitoring and Management

In this section, we present a framework for automatic cloud service monitoring and management and an approach developed using this framework.

4.1 Automatic Cloud Service Monitoring and Management Framework

Figure 1 shows a framework for automatic cloud service monitoring and management using SLA and ontology. This framework consists of eight main components as follows:

1. *Customers*: It may be user or computer that uses the cloud services through Web portal. The cloud services provided by the service provider may be situated anywhere in the world. Customer specifies their requirements in SLA which is used to perform the monitoring.
2. *User Interface*: User interface may be a Web portal though which customers interact with the cloud broker.
3. *Service Providers*: Service providers are those, which deliver services to the customers through Web portal. The examples of different cloud providers are Google, Amazon, Microsoft Azure, Rackspace, iWeb, CloudSigma, yahoo, salesforce, IBM, etc.
4. *Cloud Broker*: It is an intermediate entity, which interacts between the service providers and customers. It monitors cloud service to check whether SLA is violated or not. If SLA is violated, then broker performs rescheduling of the task to reduce further SLA violation.
5. *Monitoring*: This entity calculates the QoS parameters availability, throughput, and response time using Eq. (1), Eq. (2), Eq. (3), respectively and compares these parameters with the SLA. If SLA is violated, it sends alerts to both service providers and customers.

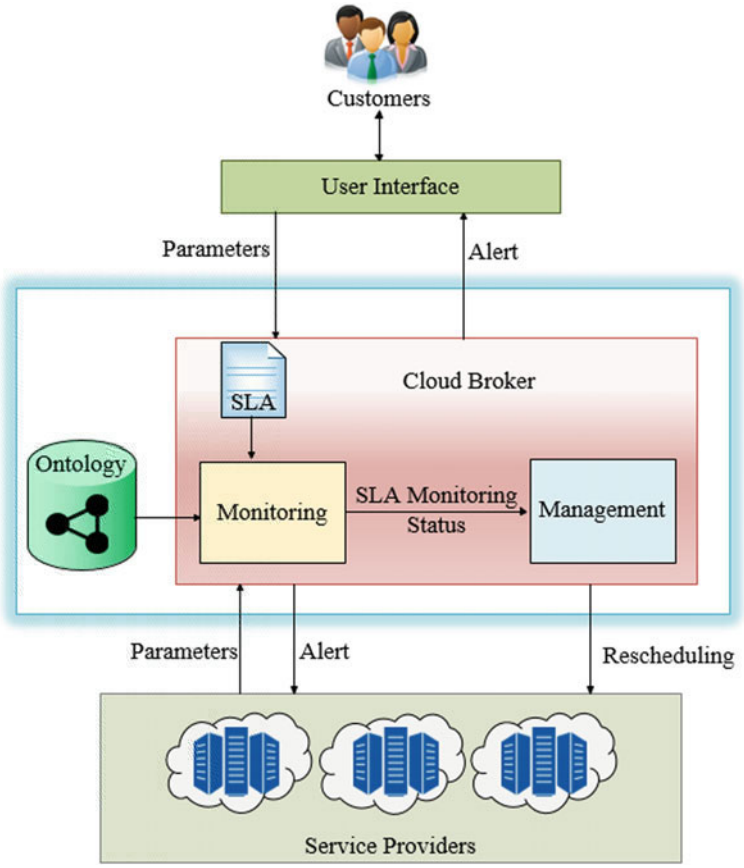

Fig. 1 Framework for automatic cloud service monitoring and management (ACSMM)

6. *Management*: This entity helps service providers to manage their resources to reduce SLA violation. When SLA is violated, this entity performs rescheduling of the task to reduce further SLA violation.
7. *SLA*: SLA is legal agreement between service provider and customers, in which services provided by the service provider is formally defined. It also specifies the action that could be taken in case of violation. This agreement is used by the monitoring entity for detection of SLA violation.
8. *Ontology*: It is a knowledge-base used by monitoring entity for SLA parameter matching in semantic manner.

In the negotiation phase of cloud service life cycle, during the creation of SLA, QoS parameters would be defined. Monitoring entity in the broker uses the SLA ontology and SLA for monitoring the cloud services. The customers and service

provider send parameters to the monitoring entity, which calculates the QoS parameters and compares with the threshold values specified in the SLA. If SLA is violated, monitoring entity sends alerts to the both customer and service provider. The monitoring entity sends the monitoring status to the managment entity as an input. For a particular task, if the SLA is violated, then it reschedules the task to the other VM which is lightly loaded; thus, it reduces the further SLA violation.

4.2 Ontology Model

Various SLA specification templates are proposed through which customers specify their requirements. The biggest problem with these templates is that they specify the same QoS parameters with different names. This problem can be resolved using the semantic knowledge of the SLA parameters by developing SLA ontology model.

It is important to note that the cloud platform is not providing any standards to specify the SLA parameters. To overcome the issue of heterogeneity of different SLA templates, we have developed a SLA ontology as shown in Fig. 2. This ontology stores the semantic knowledge of the SLA parameters to implement the mapping process of SLA parameters [12]. This mapping helps the monitoring entity to identify the QoS parameters defined in the SLA. Based on this information, the monitoring process is performed automatically to achieve efficiency in the presented work.

Figure 3 shows the SLA ontology, which contains the semantic information about the SLA parameters. From this information, it is clear that memory usage, memory utilization, memory consumption, storage requirement, memory requirement, and storage consumption are semantically equivalent to the storage functional requirement. Similarly, CPU, core, and processing element are semantically equivalent to the processor functional requirement.

For the non-functional requirement, we can infer that the required availability should be semantically same as the availability of the QoS parameter. Similarly, the required throughput should be semantically same as the throughput of the QoS parameter and so on.

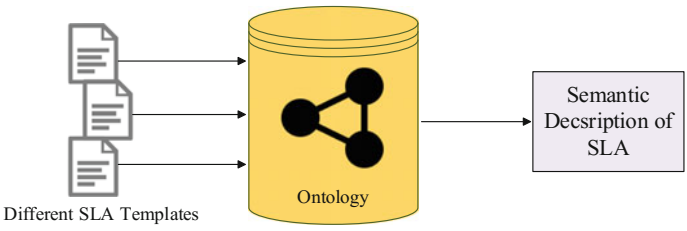

Fig. 2 Ontology model for SLA

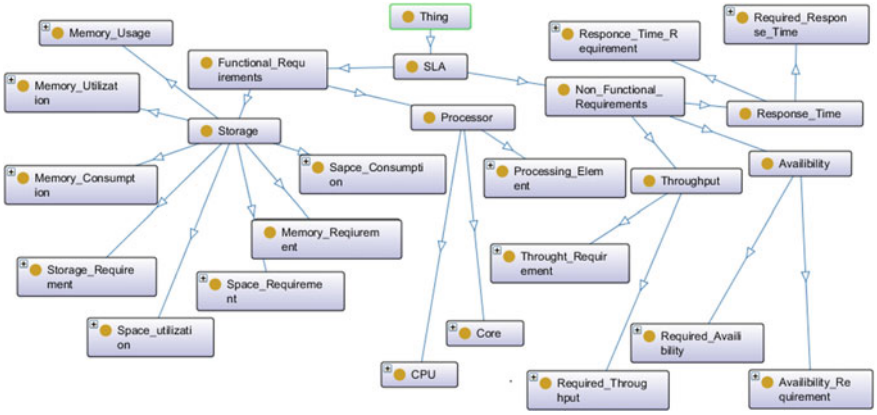

Fig. 3 SLA ontology

4.3 Automatic Cloud Service Monitoring and Management Approach

In this section, we present two algorithms: one is for automatic monitoring of cloud services and another is for rescheduling to manage the cloud services. In Algorithm 1, we monitor the QoS parameters of the cloud services. In case of SLA violation, an alert would be sent, and SLA violation report is delivered to the management module. The notations used in the Algorithm 1 and Algorithm 2 are defined as follows:

- *Cav* (*Calculated Availability*): It is the value of the QoS parameter availability for a customer calculated by monitoring entity. This value is compared with the threshold value of the availability to check the SLA violation.
- *Crt* (*Calculated Response Time*): It is the value of the QoS parameter for response time a customer calculated by monitoring entity. This value is compared with the threshold value of the response time to check the SLA violation.
- *Ctp* (*Calculated Throughput*): It is the value of the QoS parameter for throughput a customer calculated by monitoring entity. This value is compared with the threshold value of the throughput to check the SLA violation.
- *Monitoring_Status:* It monitors the status report of SLA violation for a particular time interval. This status report is used by the management entity for the rescheduling purpose.
- *Time_Interval*: It is the time interval after which monitoring is done. It is very important to determine the appropriate monitoring interval. If we choose very large monitoring interval then we would not be able to delect all SLA violation

and if we choose very small monitoring interval then it would adverly affect the system performance. Thus, we have to choose optimum monitoring interval depending upon the consumption of cloud resources.

- *semSLA_av* (*Threshold value for Availability*): It is semantically enabled threshold value of QoS parameter availability for a particular customer as specified in the SLA.
- *semSLA_rt* (*Threshold value for Response*): It is semantically enabled threshold value of QoS parameter response time for a particular customer as specified in the SLA.
- *semSLA_tp* (*Threshold value for Throughput*): It is semantically enabled threshold value of QoS parameter throughput for a particular customer as specified in the SLA.
- *sub_time* (*Subscription Time*): It specifies the time left from the due date of the end of the subscription. In our algorithm, if the subscription time is less than 10 days, then alter would we send to both customer and service provider.
- *vm* (*Virtual Machine*): In cloud, a task is executed when it is assigned to a particular virtual machine.

```
Algorithm 1: Algorithm for SLA based Monitoring
Input: semSLA_av, semSLA_rt, semSLA_tp, sub_time
Output: Monitoring_Status
1. while (MonitoringTime = = true)
2. if (Cav >= semSLA_av OR Crt<= semSLA_rt OR Ctp>= semSLA_tp OR
   sub_time<10) then
3.              sends alert to the customer
4.              sends alert to ServiceProvider
5. Monitoring_Status ← SLA_Violation_Result
6. endif
7. wait(Time_Interval)
8. endwhile
9. returnMonitoring_Status
10.   exit
```

In the above algorithm, the semantic enabled threshold value of availability, response time, throughput, subscription time is taken as input. These threshold values are defined in SLA; based on these values, monitoring is done. The output of the algorithm is monitoring status report which is based on the SLA violation result. The SLA violation result is calculated which is based on the percentage of SLA violation which can be calculated using Eq. (4).

$$\mathrm{SLA}_{\mathrm{Violation}}\% = \frac{\text{Number of violated QoS Paremeter}}{\text{total no QoS Parameters in SLA for a Customer}} \times 100 \quad (4)$$

First in algorithm while loop is taken, and loop is continuing until the monitoring time value become false. Then, we check the QoS parameter value with the threshold values specified in the SLA (step 2). If SLA is violated, then alert is sent

to both the client and service provider. The SLA violation result is stored in monitoring status report (step 3, 4, 5). The monitoring process is performed at regular time interval (step 7). It is very important to determine the appropriate monitoring interval. If we choose very long time interval, we will not able to take appropriate action in case of SLA violation; whereas if we choose very small monitoring interval, it will adversely affect the system performance. Thus, we have to choose optimum monitoring interval depending upon the consumption of cloud resources. At last, the algorithm returns the monitoring status report, which is used as input to the Algorithm 2. The complexity of the algorithm 1 is O(n), where n is number of time loop is executed.

```
Algorithm 2: Algorithm for rescheduling of task
Input: Monitoring_Status
Output: Task is assigned to vm having less load
1. for each task i in Monitoring_Status
2. if (SLA is violated for task i)
3. for each host
4. for each vm
5. Find vm having least load
6. Assign i to vm
7. endfor
8. endfor
9. endif
10.   endfor
11.   exit
```

Whenever SLA is violated, the rescheduling of task is performed as per the Algorithm 2; thus, it reduces the further SLA violation. In the Algorithm 2, the monitoring status report, which is the output of the Algorithm 1 would be the input of this algorithm. In the Algorithm 2, first we check the task whose SLA is violated (step 2). If SLA is violated algorithm find the virtual machine having least load (step 3, 4, 5). Then the selected virtual machine is assigned to that task (step 6); thus, this will reduce the further SLA violation. The complexity of the Algorithm 2 is O (n * m * k) where n is number of task, m is number of host, and k is number of host. In our approach, the monitoring is performed by Algorithm 1 after that the management of the SLA violation is performed by Algorithm 2, which is based on the monitoring status report of the task.

5 Experimental Setup and Results

In this section, we discuss the experimental setup and the results derived through experiments by applying our proposed work as follows:

5.1 Experimental Setup

We implement our framework using CloudSim 3.02 [13], which is a Java-based simulation tool for cloud environment. There are many features supported by CloudSim, i.e., network topologies, dynamic insertions of simulation entities, message passing applications, user-defined policies for resource allocation, etc. The various experiments are carried out on the machine that have 4 GB RAM, hard disk 500 GB, CPU 1.90 GHz, and Intel(R) Core(TM) i3-4030U processor. The machine is equipped with the 64 bits Windows 10 pro operating system. The tools used for the implementation are Eclipse Juno, jdk 1.8.0_77 Apache Jena, Protégé 5.0. The SLA of each customer is specified using WSLA language.

5.1.1 CloudSim Configuration

In our experiment, the parameters of data center, host, virtual machine, client, and cloudlet are defined. The value of these parameters is also given here:

- *Data center*: In our experiment, we have created two data centers and the both have same configuration as given in Table 1.
- *Host*: We have created total five hosts in our experimental setup. The first two hosts are in data center 1 and remaining three hosts are in data center 2. The different configuration of host is shown in Table 2
- *Virtual Machine*: We have created six virtual machines in our experimental work. The different configuration of the virtual machines is shown in Table 3. We assign Vm1 and Vm2 to host1 and host2, respectively. We assign the two virtual machines Vm3 and Vm4 to host3. We also assign two virtual machines Vm5 and Vm6 to the host4. We define another parameter million instructions (MIPS) for virtual machines. The virtual machine having higher MIPS will have better performance for the execution.

Table 1 Data center configuration

Arch	OS	Vmm	Time zone	Cost	Cost per memory	Cost per Storage	Cost per Bw
× 86	Linux	Xen	10.0	3.0	0.05	0.001	0.0

Table 2 Host configuration

Host Id	Datacentre Id	No. of Pes	RAM (GB)	Bandwidth (Gbit/s)	Storage (TB)
1	1	4	1	1	1
2	1	4	1	1	1
3	2	2	2	1	1
4	2	4	4	1	1

Table 3 VM configuration

Vm Id	MIPS	RAM (GB)	Bandwidth (Gbit/s)	No. of Pes	Vmm
1	2000	1	1	4	Xen
2	2000	1	1	4	Xen
3	1000	1	1	1	Xen
4	1000	1	1	1	Xen
5	1000	2	1	2	Xen
6	1000	2	1	2	Xen

Table 4 Cloudlet configuration

Cloudlet Id	No. of Pes	Required availability (%)	Required throughput	Required response time (s)
1	2	95	20	250
2	2	90	10	200
3	2	100	30	300
4	2	95	25	350
5	2	98	30	500
6	2	100	25	200
8	2	98	30	350

- *Cloudlet:* We have created eight tasks as cloudlets and for that the required QoS values are specified in Table 4. These QoS values are defined in the SLA.

In our experiment, we have calculated the values of the QoS parameters of the cloudlet and then checked them with the QoS values as specified in the SLA. If SLA violation is detected, then rescheduling is performed to reduce the further SLA violation. We compare the results SLA violation percentage with management module.

5.2 Experimental Results

In this section, we present the experimental results of our framework and compare the SLA violation results of with using rescheduling and without rescheduling. Figure 4 shows the SLA violation results of cloudlets without using rescheduling; thus, it means the management entity of our framework which is not included while taking the SLA violations results. Figure 5 shows the SLA violation results of cloudlets with rescheduling. We monitor the SLA parameters of the all eight cloudlets after every 400 seconds monitoring interval.

After first 400 seconds, if we compare both graphs, we get same results; as rescheduling is done, only after SLA violation is detected. After 800 seconds still

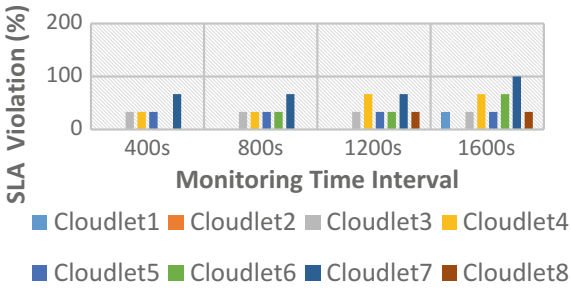

Fig. 4 SLA violation without rescheduling

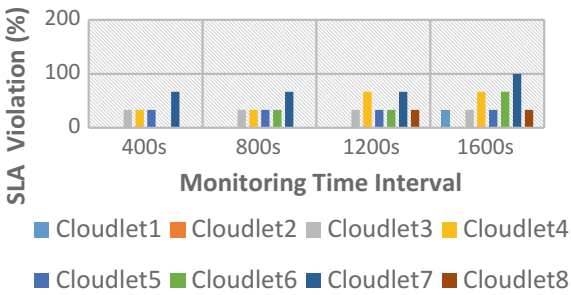

Fig. 5 SLA violation with rescheduling

results of the both graph are same, because there is no virtual machine so load is very less. After 1200 seconds, we clearly found that the SLA violations are reduced for cloudlet 3 and cloudlet 7. After 1600 seconds, we found that the SLA violation results are reduced for the cloudlet 3, cloudlet 4, cloudlet 5, cloudlet 6, and cloudlet 8. For the other cloudlets, the results remain same. By comparing both the graphs, we can easily conclude that the SLA violation results for the cloudlets are reduced when rescheduling is applied.

5.3 Comparative Study with Exiting Approaches

There are several cloud service monitoring frameworks proposed, but only few of them provides mechanism to reduce the SLA violation in automatic manner. In [14], authors presented the approach to automate the QoS management, but they did not specified the detail of proposed work regarding automated QoS management. The Detecting SLA violation Infrastructure (DeSVi) architecture [15] is one of the automatic SLA violation detection architecture, which provides timely guidance depending on the consumption of resources, but this architecture is service provider oriented. A detailed survey on different cloud service monitoring tools is described in [16, 17]. As per the survey, we observed that most of tools are service provider oriented and this may raise the question of unfair monitoring of cloud services in case of SLA violation.

6 Conclusion and Future Work

In this paper, we proposed a framework and approach for automatic monitoring and management of the cloud services to monitor the quality of offered services using SLA ontology. When SLA violation is detected, our approach sends the alert to both service provider and user. To reduce further SLA violation, our approach automatically finds the virtual machine having light load and allocate that virtual machine to the task. We have demonstrated the experimental results derived through rescheduling and compared these with traditional (without rescheduling) approach. The results show that the automatic service monitoring and rescheduling enhance the performance of the cloud services. From the proposed work, we can specify that it is a win-win situation for both customers and service providers because monitoring is applied at broker level, so it will provide a fair SLA violation results to the users at the same time automatic rescheduling helps service providers to manage the SLA. In future work, we focus to predict the SLA violation based on the current resource conditions in cloud by applying machine learning technique. In that case, we would require the previous knowledge of SLA violation condition to predict the SLA violation. Thus, we will take action based on the prediction which will reduce the SLA violation in the cloud environment at significant level.

References

1. Yashpalsinh, J., Modi, K.: Cloud computing-concepts, architecture and challenges. computing, electronics and electrical technologies (ICCEET), In: International Conference on. IEEE (2012)
2. Joshi, K., Yesha, Y., Finin, T.: Automating cloud services life cycle through semantic technologies. Serv Comput. IEEE Trans. **7**(1), 109–122 (2014)
3. Frey, S., Reich, C., Lüthje, C.: Key performance indicators for cloud computing SLAs. In: The Fifth International Conference on Emerging Network Intelligence, Emerging (2013)
4. Ludwig, H., Keller, A., Dan, A., King, R., Franck, R.: Web Service Level Agreement (WSLA) Language Specification. IBM Corporation, pp. 815–824 (2003)
5. Aljoumah, E., Al-Mousawi, F., Ahmad, I., Al-Shammri, M., Al-Jady, Z.: SLA in Cloud Computing Architectures: A Comprehensive Study. Int. J. Grid Distributed Comput. **8**(5), 7–32 (2015)
6. Zia, et al.: A framework for user feedback based cloud service monitoring. Complex, Intelligent and Software Intensive Systems (CISIS). In: 2012 Sixth International Conference on. IEEE (2012)
7. Khandelwal, H., Kompella, R., Ramasubramanian, R.: Cloud monitoring framework. Purdue University
8. Sahai, A., Machiraju, V., Sayal, M., Jin, L., Casati, F.: Automated SLA monitoring for web services, pp. 28–41. Management Technologies for E-Commerce and E-Business Applications. Springer, Berlin Heidelberg (2002)
9. Mohamed, S., Yousif, A., Bakri, M.: SLA Violation detection mechanism for cloud computing. Int. J. Comput. Appl. **133**(6), 8–11 (2016)
10. Vaitheki, K., Urmela, S.: A SLA violation reduction technique in Cloud by Resource Rescheduling Algorithm (RRA). Int. J. Comput. Appl. Eng. Technol. 217–224 (2014)

11. Singh, S., Chana, I., Buyya, R.: STAR: SLA-aware autonomic management of cloud resources. IEEE Transactions on Cloud Computing (2017)
12. Redl, C., Breskovic, I., Brandic, I., Dustdar, S.: Automatic SLA matching and provider selection in grid and cloud computing markets. In: Proceedings of the 2012 ACM/IEEE 13th International Conference on Grid Computing. IEEE Computer Society (2012)
13. Calheiros, R., Ranjan, R., Beloglazov, A., Rose, C., Buyya, R.: CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithm. Soft. Pract. Exp. **41**(1), 23–50 (2011)
14. Alhamazani, K., Ranjan, R., Rabbhi, F., Wamg, L., Mitra, K.: Cloud monitoring for optimizing the QoS of hosted applications. In: IEEE 4th International Conference on IEEE (2012)
15. Emeakaroha, V., Netto, M., Cleheiros, R., Brandic, I., Buyya, R., Rose, C.: Towards autonomic detection of SLA violations in Cloud infrastructures. Fut. Gen. Comput. Syst. **28**(7), 1017–1029 (2002)
16. Aceto, G., Botta, A., Donato, W., Pescape, A.: Cloud monitoring: a survey. Comput. Netw. 57.9, **57**(9), 2093–2115 (2013)
17. Alhamazani K., Ranjan, R., Mitra, K., Rabhi, S., Khan, S., Guabtni, A.: An overview of the commercial cloud monitoring tools: research dimensions, design issues, and state-of-the-art. Computing, **97**(4), 357–377 (2015)

Incorporating Collaborative Tagging in Social Recommender Systems

K. Vani and T. Swathiha

Abstract Recommender systems play a major role in recent times to help online users in finding the relevant items. Traditional recommender systems have been analysed immensely, but they ignore information like social friendships, tags which when incorporated in recommendation can improve its accuracy. With the advent of social networking sites, study of social recommender systems has become active. Most of the users ask their friends for recommendation. But not all friends have similar taste as that of the user, and different group of friends contribute to different recommendation tasks. So this paper proposes an approach to identify different group of friends by grouping users based on items and retains the personal interest of experienced by incorporating individual-based regularization in basic matrix factorization. Information like ratings, tags and friendship are used in predicting the missing values of user-item matrix efficiently. Empirical analysis on the dataset proves that the proposed approach is better than the existing methods.

Keywords Social recommendation · Tags · Personal interest
Interpersonal influence · Matrix factorization

1 Introduction

Recommender systems are one of the most important Web applications that provide many services and suggest some services automatically as per user's interest. Recommender systems have gained its popularity due to information overload prevailing in the Internet. It helps users to identify items which are interested to them. There are various techniques in personalized recommenders like content based and collaborative filtering technique [1]. The traditional recommender systems suffer from various limitations like cold start problem, data sparsity, scalability. To overcome these problems, social recommendation is introduced.

K. Vani (✉) · T. Swathiha
Department of CSE, PSG College of Technology, Coimbatore, India
e-mail: kvanisaras@gmail.com

E. B. Rajsingh et al. (eds.), *Advances in Big Data and Cloud Computing*,
Advances in Intelligent Systems and Computing 645,
https://doi.org/10.1007/978-981-10-7200-0_2

Such recommender systems incorporate information like friends, ratings, and tags which improve the accuracy of recommendation. Similarly, users' trust relationship can also be incorporated to improve accuracy of traditional recommender systems. Such recommender systems consider only trustable users and recommend the items rated by them. But it suffers from several limitations. Trust relationships need not be mutual like social relationships, and building such trust network is difficult since very few Websites provide trust assigning facility. Due to the rapid growth of social networking sites, users like to interact more with their friends rather than the trusted users. So these problems led to focus more on social recommender systems. In social recommender systems, not all friends have similar taste as that of the user and different group of friends can contribute to different recommendation tasks. So this paper proposes an approach to group users based on similar items and incorporates efficient regularization [2] to handle friends with different taste. Preference of any item by a user can be known with their ratings as well as from their tagging behaviour [3]. Ratings tell how much a user liked an item, whereas tags indicate why they liked the item [4]. So tagging information is also incorporated with friendship and rating information to make recommendations. Experiments have been conducted on dataset to evaluate the performance of this approach. The paper is organized as follows. Section 2 provides an overview of different approaches of recommender systems. Section 3 describes the social recommendation framework and Sect. 4 presents the experimental analysis and results.

2 Related Work

This section reviews important approaches of recommender systems including social recommender systems and tag-based recommender systems.

2.1 Tag-Based Recommender Systems

Collaborative filtering technique fails when there are diverse domains because people with similar taste in one domain may not agree well in other domain. So to improve recommendation quality, content information about items has been used as additional knowledge which led to the advent of collaborative tagging systems. Tagging information helps in easy retrieval of items [5–7]. It also helps in classifying their favourite items using the keywords. TF-based recommender system is one of the content-based filtering approaches which exploit tagging information. It models tag-based user profiles and item profiles. The simplest method to assign weight to a particular tag is by counting the number of times that tag has been used by the user, or the number of times the item has been annotated by that tag. Items which are highly tagged by the user will be recommended. Probabilistic latent semantic analysis (PLSA) and FolkRank algorithm are used to form tag-based

recommender systems. PLSA [8] is used to find the semantic structure of a data using co-occurrence matrix in a probabilistic framework. FolkRank algorithm [9] is used to perform topic-specific ranking in folksonomies.

2.2 Social Recommender Systems

Nowadays, the usage of social networking sites has been increased tremendously. Social influences play a key role when people are making decisions of adopting products. Social recommender systems [10] aim to reduce information overload problem by presenting the most relevant content. MF in social networks is proposed in [11]. It assumes that neighbours in the social network may have similar interest as that of a user. The objective function is

$$\frac{1}{2}\sum_{(i,u)observed}\left(R_{u,i}-\hat{R}_{u,i}\right)^2+\frac{\beta}{2}\sum_{all\,u}\left(\left(Q_u-\Sigma_v S^*_{u,v}Q_v\right)\left(Q_u-\Sigma_v S^*_{u,v}Q_v\right)^T\right)+\frac{\lambda}{2}\left(\|P\|_F^2+\|Q\|_F^2\right) \tag{2.1}$$

The second term in the objective function minimizes the distance between user profile Q_u and average of his friends' profiles Q_v. Matrix S denotes the user–user trust values. This equation can be optimized using gradient descent approach. CircleCon model [12] is found to be better than matrix factorization methods [13]. It is an extension of MF in social networks with inferred circle of friends. MF in social networks considers all friends, but preference of friends will be different in different categories. This idea is incorporated in CircleCon model. It infers a circle of friends from entire social network regarding a specific category of items.

Most of the approaches consider only the social network information. Social context factors like individual preference and interpersonal influence are ignored. But these factors which affect user's decision are incorporated in ContextMF model [14]. In ContextMF recommendation, a user sends or recommends a set of items to another user. Percentage of items adopted by user U from user V is denoted by interpersonal influence. Matrix W denotes the interpersonal similarity of users. All these factors contribute in predicting the missing values in user-item rating matrix. Individual preference is obtained by the user's history. The objective function of ContextMF model is

$$\frac{1}{2}\sum_{(u,i)}\left(R_{u,i}-\hat{R}_{u,i}\right)^2+\frac{\beta}{2}\sum_{u}\left(\left(Q_u-\sum_v S^*_{u,v}Q_v\right)\left(Q_u-\sum_v S^*_{u,v}Q_v\right)^T\right)+\frac{\gamma}{2}\sum_{u,v}\left(W^*_{u,v}-Q_uQ_v^T\right)+\frac{\lambda}{2}\left(\|P\|_F^2+\|Q\|_F^2\right) \tag{2.2}$$