Efficient Dynamic Task Scheduling in Virtualized Data Centers with Fuzzy Prediction

Xiangzhen Kong[1], Chuang Lin[1], Yixin Jiang[1], Wei Yan[1], and Xiaowen Chu[2]

[1] Department of Computer Science and Technology, Tsinghua University
Beijing, 100084, P. R. China
Email:xiangzhen1985@gmail.com,{clin,yxjiang}@csnet1.cs.tsinghua.edu.cn,
yanw08@mails.tsinghua.edu.cn

[2] Department of Computer Science, Hong Kong Baptist University,
Hong Kong, P.R. China
Email: chxw@comp.hkbu.edu.hk

***Abstract*—**System virtualization provides low-cost, flexible and powerful executing environment for virtualized data centers, which plays an important role in the infrastructure of Cloud computing. However, the virtualization also brings some challenges, particularly to the resource management and task scheduling. This paper proposes an efficient dynamic task scheduling scheme for virtualized data centers. Considering the availability and responsiveness performance, the general model of the task scheduling for virtual data centers is built and formulated as a two-objective optimization. A graceful fuzzy prediction method is given to model the uncertain workload and the vague availability of virtualized server nodes, by using the type-I and type-II fuzzy logic systems. An on-line dynamic task scheduling algorithm named *SALAF* is proposed and evaluated. Experimental results show that our algorithm can improve the total availability of the virtualized data center while providing good responsiveness performance.

***Keywords*—**Virtualized data center; Task scheduling; Fuzzy logic; Availability; Load-balance

1. INTRODUCTION

The trend toward server-side computing and the exploding popularity of Internet services has made data centers become an integral part of the Internet fabric rapidly. Data centers become increasingly popular in large enterprises, banks, telecom, portal sites, etc [1-3]. As data centers are inevitably growing more complex and larger, it brings many challenges to the deployment, resource management and service dependability, etc [4]. Virtualization is viewed as an efficient way against these challenges. Server virtualization opens up the possibility of achieving higher server

consolidation and more agile dynamic resource provisioning than is possible in traditional platforms [5]. A data center built using server virtualization technology with virtual machines (VMs) as the basic processing elements is called a virtualized (or virtual) data center (VDC) [6-8]. Due to the advantages in deployment, management, dependability and cost, VDCs become the next infrastructure trend with the popularity of Cloud computing and IaaS (Infrastructure as a Service) [9, 10], such as Amazon EC2 [11] and VMware vCloud [12].

However, the characteristics of virtualization also bring new challenges to VDCs, particularly to the task scheduling and resource management. Server consolidation [13, 14] makes many VMs run in a physical server. VMs are loosely coupled with the underlying hardware and share the hardware resources of the physical server such as CPU, memory and network. The loosely-coupled and highly-shared features of virtualization make it difficult to accurately measure the running parameters and resources usage information of each VM, which cause some complexity in resource management and task scheduling in VDCs. In addition, the mechanism of live migration [15] makes it possible for VM appliance to move between different physical servers in a VDC. It further exacerbates the dynamicity and nondeterminacy of VDCs. Such characteristics bring challenges for the traditional task scheduling schemes to work well in VDCs.

We focus on the task scheduling problem of VDCs in this paper. Some performance metrics, such as high throughput, low response delay and short makespan, are the conventional optimization goals for task scheduling. The traditional scheduling algorithms usually assume that all server nodes are always available for processing. In practice, this assumption is often not plausible in some scenarios where certain breakdowns, requirements for maintenance, or other constraints that make the server nodes unavailable for processing exist [16]. For example, in VDCs, a node is sometimes unavailable during the processes of backup, update maintenance or live migration. However many service applications require data center platform with high availability, particularly for some critical services such as military, healthcare applications [16]. So availability is also a critical metric that a scheduling policy in VDCs should take into considerations. But in practical applications, it is unpractical for users to accurately specify their availability requirements in the SLAs (Service Layer Agreements) for their submitted tasks. A more friendly way is to let users designate a fuzzy level of availability

requirements, such as high, medium or low level. Then, how to deal with the vagueness of availability requirements in the task scheduling is also a challenge. Moreover, to improve the availability needs more extra processing overhead that may impact on the scheduling performance. So achieving high performance and availability simultaneously is also a concern, as they are usually conflicting with each other [16].

Furthermore, the existing researches on the scheduling in VDCs mostly focus on the infrastructure layer, such as resource provisioning and VMs placement [7, 8, 17-19]. They are dedicated to improve the performance of the data center infrastructure, but the service layer requirements specified by SLAs are ignored. Different task classes for different applications (FTP, streaming media, etc.) often have distinct requirements in the SLAs such as response time and availability, which are closely related to the Quality of Service (QoS) from the users' view. To improve the user experience, it motivates us to study the task scheduling problem from the service layer by considering different requirements of performance and availability in the SLAs.

In this paper, we propose an effective task scheduling scheme for VDCs, which makes a good trade-off between availability and performance. The multiclass tasks model is introduced in our scheme, and different classes of tasks are characterized by their distinct arrival rates, service time, and availability requirements. The contributions are concluded as the following aspects: 1) We build a general model for the task scheduling in VDCs and formulate it as a two-objective optimization problem; 2) We give a graceful fuzzy prediction method to model the uncertain workload and the vague availability of virtualized server nodes, using the type-I and type-II fuzzy logic systems; 3) We design and evaluate a dynamic task scheduling algorithm which could efficiently improve the total availability of the VDC while maintaining good responsiveness performance.

The rest of this paper is organized as follows. Section 2 introduces the background. The general formal model of the task scheduling system in VDCs is introduced in section 3. In section 4, availability and load-balance predictors are constructed using type-I and interval type-II fuzzy logic systems, respectively. Section 5 presents the new dynamic scheduling algorithm. Section 6 gives the performance evaluation of the algorithm, followed with the conclusions in Section 7.

2. BACKGROUND

2.1 Task Scheduling in Virtualized Data Centers

Task scheduling is to assign tasks to different executive units while satisfying some constraints. In VDCs, a VM with the corresponding VMM and HW works as the basic executive unit called virtual executive unit (*VEU*), which is the provider of services specified in the SLAs. Task scheduling techniques can be either static or dynamic. Static scheduling schemes assume a fixed tasks set and a priori knowledge of the characteristics of the workload with respect to the systems. It is usually impractical in real systems, particularly for the VDCs. In VDCs, server consolidation [13, 14] enables that several VMs run on the same hardware machine and share the underlying hardware resources through the VMM, which helps increasing the utilization of server resources. Moreover, virtualization provides a good isolation between different VMs, and VM appliance can dynamically migrate to another machine through live migration [15]. These characteristics and mechanisms bring dynamicity and nondeterminancy to task scheduling in VDCs, and it is hard for the static scheduling schemes to get detailed prior knowledge. So we attempt to design an effective dynamic task scheduling scheme to assign tasks to different executive entities (VMs) for improving satisfaction degree of the availability requirements in SLAs while maintaining good responsiveness.

2.2 Type-I and Type-II Fuzzy Logic Systems

We turn to the fuzzy logic system (FLS) to deal with the challenges caused by the dynamicity of virtualization and the vagueness of availability requirements in the scheduling strategy of VDCs. A FLS is particularly good at handling uncertainty, vagueness and imprecision. FLSs are widely used in many areas, and could also efficiently deal with the uncertainty in task scheduling [20]. The concept of type-II fuzzy sets was introduced by Zadeh as an extension of the concept of an ordinary fuzzy set, i.e. the type-I fuzzy set [21]. The membership grades of type-II fuzzy sets are themselves type-I fuzzy sets, which include a primary membership and a secondary membership. Type-II fuzzy logic systems are very useful in the scenario where it is difficult to determine an exact membership function for a fuzzy set; hence, they are useful for incorporating uncertainties. In [22], three cases under which type-II FLS is more suitable for use than type-I are summarized. General type-II FLSs

are computationally intensive, so a special case with the secondary MFs be interval sets called Interval Type-II FLSs (IT2-FLSs) were studied. Liang et al. [23] proposed an efficient and simplified method to compute the input and antecedent operations for IT2-FLSs. Hereafter, the IT2-FLS becomes a powerful and popular tool that is widely used in recent years, and we first introduce it to the task scheduling in VDCs. For the high dynamicity of virtual machines, the workload data sampled from each server node are usually noisy too. It causes difficulties to make a good decision in the process of task scheduling. To deal with the noise in the training data, the IT2-FLS is applied in the task scheduling policy of the VDCs.

3. Model Descriptions and Formulations

3.1 The Framework of Task Scheduling in Virtualized Data Centers

We use the *MSQMS-LQ* (Multi-classes Single Queue to Multiple Servers with Local Queues) model for the task scheduling in a VDC (see Fig. 1). There is a shared waiting queue before entering the scheduler. Each virtual machine acts as the server with a local queue for arrived tasks. Since different class of tasks may have different characteristics and availability requirements, multiclass task model [16, 24] is applied in our framework. Assume that there are K classes of non-preemptive tasks, where K stands for the number of different levels of task availability requirement specified by SLAs. In this paper, we consider K=5, which stands for 5-levels of availability requirements: *very high*, *high, medium*, *low* and *very low*. There are m physical servers in the data center, and the *Virtual Server*$_i$ consists of a HW, a VMM and n_i VMs. Then, there are totally $\sum_{i=1}^{m} n_i = N$ virtual machines.

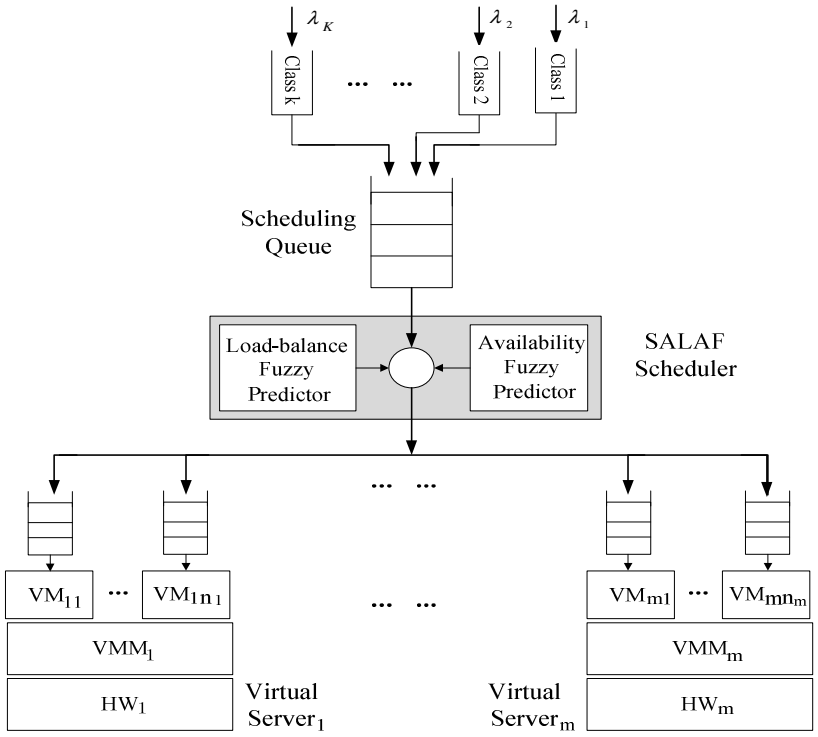

Fig. 1. The framework of the task scheduling in a virtualized data center.

Table I. Notations and definitions.

Notation	Definition
m	Total number of the virtual servers
N	Total number of the virtual machines.
K	Total number of the task classes
n_i	Total number of VMs hosted in the *i*th virtual server
f_{HW_i} , f_{VMM_i} , $f_{VM_{ik}}$	Failure rates of the HW_i, VMM_i, VM_{ik}
r_{HW_i} , r_{VMM_i} , $r_{VM_{ik}}$	Repair rates of the HW_i, VMM_i, VM_{ik}
A_{HW_i} , A_{VMM_i} , $A_{VM_{ik}}$	Availability of the HW_i, VMM_i, VM_{ik}
$\mu_{LA}(\cdot)$	Membership function of low availability nodes set
$\mu_{MA}(\cdot)$	Membership function of medium availability nodes set
$\mu_{HA}(\cdot)$	Membership function of high availability nodes set
x_{HW_i} , x_{VMM_i} , $x_{VM_{ik}}$	The availability inputs of membership function for HW_i, VMM_i, VM_{ik}
λ_i	Average arrival rate of the *i*th class tasks
μ_{ij}	Service rate when the *i*th class tasks are dispatched on VEU_j
p_{ij}	Probability of that the *i*th class tasks are dispatched to VEU_j
Λ_j	Aggregate task arrival rate of the VEU_j
ρ_j	The server utilization of VEU_j
$\overline{S_j}$	The expectation value of service time on VEU_j
σ_{S_j}	Standard deviation of service time on VEU_j
T_j	The service time of VEU_j
α_i	Availability requirement level of the *i*th class tasks specified in the SLA.
β_j	Availability level provided by the VEU_j

The model of the scheduling system, depicted in Fig. 1, is composed of a scheduling queue, a task scheduler using the algorithm of *SALAF* (*Scheduling Algorithm based on Load-balance and Availability Fuzzy prediction*), and local task queue for each VM. The goal of *SALAF* is to make a good task allocation decision for each arrived task to satisfy its availability requirement and maintain a good performance in response time. The core of the framework is the *SALAF* task scheduler, which consists of two important components: *Availability Fuzzy Predictor* (*AFP*) and *Load-balance Fuzzy Predictor* (*LFP*). The *AFP* computes the availability of each virtual execution unit (*VEU*), by combing the availability of HW, VMM and VM. It predicts that which *VEU* could satisfy the availability requirement of each arrival task. The *LFP* predicts the load-balance status through autoregression model *AR*(16) and interval type-II FLS. The details will be introduced in the section 4.

3.2 Model Formulations

At first, we build a general formal model for the task scheduling in a virtualized data center. We summarize the notations and their definitions used throughout this paper in Table I. There are *K* classes of tasks, and each class has an availability level specified by SLAs. Values of availability levels are in the range {*very low*, *low*, *medium*, *high*, *very high*}. There are *N* virtual execution units, and $VEU_j = \{HW_i, VMM_i, VM_{ik}\}$, where $1 \le i \le m$, $1 \le k \le n_i$ and $1 \le j \le N$. We assume that the arrival of the tasks of the *i*th ($1 \le i \le K$) class conforms to a Poisson process with rate λ_i. The interval arrival time of Poisson arrival tasks conforms to the exponential distribution. According to the additive property of exponential distribution, the aggregate tasks arrival of all classes is also a Poisson process with rate $\lambda = \sum_{i=1}^{K} \lambda_i$.We model the sub-system of VEU_j which includes a VM and a local queue by an *M/G/1* queue model. We assume that p_{ij} is the probability that the tasks of the *i*th class are dispatched to VEU_j, where $1 \le j \le N$.Then the aggregate task arrival rate for VEU_j is expressed as $\Lambda_j = \sum_{i=1}^{K} \lambda_i p_{ij}$. Let μ_{ij} be the service rate of tasks in class *i* allocated into VEU_j. Then the server utilization of VEU_j is $\rho_j = \sum_{i=1}^{K} \left(p_{ij} \lambda_i / \mu_{ij} \right)$, where *K* is the total number of the task classes.

The expectation value and standard deviation of service time for the tasks of *K* classes dispatched on VEU_j are

$$\overline{S_j} = \frac{1}{\Lambda_j}\sum_{i=1}^{K}\left(\frac{p_{ij}\lambda_i}{\mu_{ij}}\right) \text{and}\ \sigma_{S_j} = \sqrt{\frac{1}{k-1}\sum_{i=1}^{K}\left(\frac{1}{\mu_{ij}} - \overline{S_j}\right)^2} \tag{1}$$

Hence, according to the *Pollaczek-Khinchin mean-value formula* [25], we obtain the service time of VEU_j as

$$T_j = \overline{S_j} + \frac{\rho_j \overline{S_j}\left(1+\left(\sigma_{S_j}/\overline{S_j}\right)^2\right)}{2(1-\rho_j)} \tag{2}$$

Therefore, the expected response time of all the *K* classes tasks can be expressed as

$$T = \sum_{i=1}^{K}\left(\frac{\lambda_i}{\lambda}\cdot\sum_{j=1}^{N}\left(p_{ij}T_j\right)\right) \tag{3}$$

Let α_i denote the availability requirement level of the *i*th class tasks, which could be specified in the SLA. Let β_j denote the availability level provided by the VEU_j, so $\alpha_i, \beta_j \in$ {*very low*, *low*, *medium*, *high*, *very high*}. We assign the numbers of 0-4 to the availability level *very low* to *very high*. If $\alpha_i \le \beta_j$, the availability requirement of the task can be satisfied by the VEU_j. Hence, the probability that the *i*th class task is distributed to the VEU_j and the availability requirement can be satisfied is $p_{ij}\cdot Prob\{\alpha_i \le \beta_j\}$. Therefore, the total satisfaction probability of all the *K* classes tasks' availability requirement is

$$A = \sum_{i=1}^{K}\left(\frac{\lambda_i}{\lambda}\cdot\sum_{j=1}^{N}\left(p_{ij}\cdot Prob\{\alpha_i \le \beta_j\}\right)\right) \tag{4}$$

We formulate the scheduling problem as a trade-off between availability and the average response time, and obtain the formalized optimization problem as:

$$Max\ A = \sum_{i=1}^{K}\left(\frac{\lambda_i}{\lambda}\cdot\sum_{j=1}^{N}\left(p_{ij}\cdot Prob\{\alpha_i \le \beta_j\}\right)\right)$$

$$Min\ T = \sum_{i=1}^{K}\left(\frac{\lambda_i}{\lambda}\cdot\sum_{j=1}^{N}\left(p_{ij}T_j\right)\right)$$

$$s.t.\quad \begin{cases} \sum_{j=1}^{N} p_{ij} = 1 \\ \forall 1 \le i \le K, 1 \le j \le N \end{cases}$$

The key is to determine the assignment probability p_{ij} considering the dynamicity of the virtual machines and the availability levels specified in the SLAs. In the following sections, we will develop a dynamic scheduling algorithm to find a trade-off solution which maximizes the availability satisfaction probability *A* as well as minimizes the response time *T*. Prior to that there are still two critical problems. One is how to get the availability level provided by VEU_j, i.e. β_j. On the other hand, we pursue the minimum average response time, and it is usually achieved with good load-balance which means allocating more tasks to the *VEU*s with relatively lighter load. So the load status of each *VEU* and the load-balance of the VDC should be determined in the process of dynamic scheduling. To deal with the dynamicity of *VEU*s' load status and the vagueness of the tasks' availability requirement level, we turn to fuzzy logic systems.

4. AVAILABILITY AND LOAD-BALANCE FUZZY PREDICTION

4.1 Availability Fuzzy Prediction

This subsection presents the construction of the *Availability Fuzzy Predictor* (*AFP*) shown in Fig. 1. Availability is defined as the ratio of the time period when a server node is functional during a given interval. For simplicity, the concept of *Intrinsic Availability* [26] is used commonly in engineering, which is defined as

$$IA = MTBF/(MTBF + MTTR) \tag{5}$$

where $MTBF$ is the Mean Time Between Failures and $MTTR$ is the Mean Time To Repair.

In the scheduling model of virtualized data centers, the availability requirement level of each class tasks α_i is specified in the SLA, and we need to get the availability level β_j provided by each VEU_j. Each *VEU* consists of a VM and the corresponding VMM and HW. We turn to fuzzy logic system,

and the common type-I FLS is used. The key is to determine the input and output fuzzy variables, and construct the fuzzy inference rules.

1) Define the input and output fuzzy variables

A *VEU* is the set of one HW, one VMM and one of the VMs that run on the VMM. To get the availability of the *VEU*, the inputs of the fuzzy logic system include the intrinsic availability of HW, VMM and the corresponding VM. In reliability engineering, the failure rate and repair rate of a component can be obtained by statistical methods. It is normally assumed that the time to fail and the time to repair conform to exponential distribution [26]. Therefore, according to Eq. (5), we can get the intrinsic availability of HW_i by the following Proposition 1.The availability of VMM_i and VM_{ik} can be gotten similarly.

[PROPOSITION 1]: Given the failure rate of the hardware machine HW_i is f_{HW_i}, and the repair rate is r_{HW_i}, then the intrinsic availability can be expressed by

$$A_{HW_i} = r_{HW_i} /(r_{HW_i} + f_{HW_i}) \tag{6}$$

PROOF: Under the assumption of exponential distribution, the expectation value of the time between failures is $MTBF = 1/ f_{HW_i}$ and the expectation value of the time to repair is $MTTR = 1/ r_{HW_i}$. According to Eq. (5), we have $A_{HW_i} = r_{HW_i} /(r_{HW_i} + f_{HW_i})$. □

In engineering, the availability measure of a component is usually not so accurate. It is often expressed by the count of '9' after the decimal point of the availability value. For example, saying the availability of a component is three '9' stands for the ratio of the time period when the component is functional during a given interval is about 0.999. It means that the downtime is no more than 8.76 hours in a year. For hardware and software, there are different *de facto* standards about the availability level. Generally, if a hardware component has the availability of three '9' to four '9', it is considered as an acceptable availability. In regard to the software, the standard may lower to about two '9', because of the vulnerability of software. To make the different levels of availability inputs uniformly-spaced, we make logarithmic transformation to the inputs while keeping the monotonicity. Taking the HW as an example, the input variable form is

$$x_{HW_i} = -log\left(1 - A_{HW_i}\right) \tag{7}$$

Therefore, we can define the membership functions (MFs) of the hardware component (HW_i) and the software components (VMM_i and VM_{ik}) in the form of piecewise function, shown in Fig. 2. Taking the HW_i as an example, the MFs of the fuzzy sets of low availability (μ_{LA}), medium availability (μ_{MA}) and high availability (μ_{HA}) are separately defined as follows:

$$\mu_{LA}\left(A_{HW_i}\right) = \begin{cases} 1 & A_{HW_i} \le 0.99 \\ log(1 - A_{HW_i}) + 3 & 0.99 < A_{HW_i} \le 0.999 \\ 0 & Other \end{cases} \tag{8}$$

$$\mu_{MA}\left(A_{HW_i}\right) = \begin{cases} -log(1 - A_{HW_i}) - 2 & 0.99 < A_{HW_i} \le 0.999 \\ log(1 - A_{HW_i}) + 4 & 0.999 < A_{HW_i} \le 0.9999 \\ 0 & Other \end{cases} \tag{9}$$

$$\mu_{HA}\left(A_{HW_i}\right) = \begin{cases} -log(1 - A_{HW_i}) - 3 & 0.999 < A_{HW_i} \le 0.9999 \\ 1 & A_{HW_i} > 0.9999 \\ 0 & Other \end{cases} \tag{10}$$

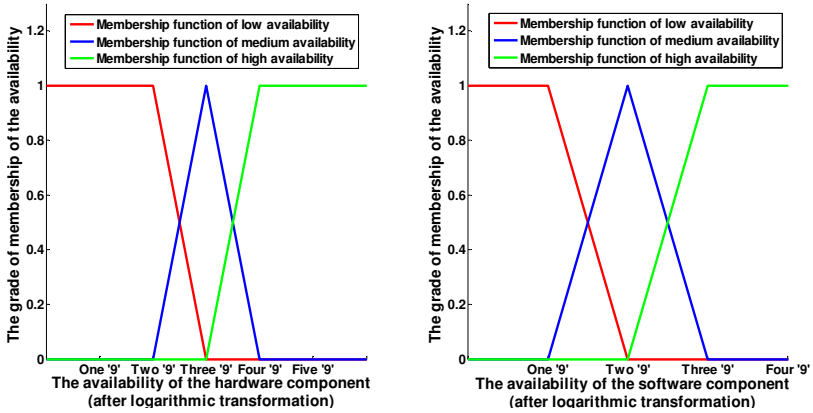

Fig. 2. Membership functions of low, medium and high availability fuzzy sets for hardware and software.

2) Construct the fuzzy inference rules

Since the availability of one *VEU* is related to the corresponding HW, VMM and VM, we construct the fuzzy inference rules each of which have three antecedents and one consequent: **1) Antecedent 1:** *Availability of the* HW_i ; **2) Antecedent 2:** *Availability of the* VMM_i ; **3) Antecedent 3:** *Availability of the* VM_{ik} ; **4) Consequent:** *Availability level of the virtual execution unit* VEU_j . Where, $1 \le i \le m$, $1 \le k \le n_i$ and $1 \le j \le N$.

According to the membership function (8)-(10), each of antecedents has three values. So there are total 27 items of rules. For example, *if* $A(HW_i)$=*low and* $A(VMM_i)$=*low and* $A(VM_{ik})$=*low, then* $A(VEU_j)$=*very low*. Due to the limited space for this paper, we omit the complete rules table.

4.2 Load-balance Fuzzy Prediction

In this subsection, we use the fuzzy logic system to get the candidate *VEU* sets that satisfy the load-balancing status in VDC, which implements the *Load-balance Fuzzy Predictor* (*LFP*) shown in Fig. 1. The responsiveness performance is closely related to the processing capacity and workload of each *VEU*, while a *VEU* consists of a VM, a VMM and the sharing HW of the corresponding physical server. The VMM contributes little to the task workload of *VEU*. We only use the workload of VM and physical server to distinguish the workload status of different *VEU*s in VDC, and select the candidate *VEU* set considering the overall load-balancing status of the VDC. Because of the loosely-coupled relation between HW and VMs and the dynamicity of VMs, the workload data sampled from each server node are usually noisy. So we turn to the interval type-II fuzzy logic systems (IT2-FLSs). Four steps are applied to get the candidate *VEU* sets for each arrived task.

1) Get the load indicators by auto-regression model

The CPU process queue length and memory utilization are usually used as indicators to reflect the workload status of a server node [27]. In our scheduling scheme, we sample these two load indicators of each node periodically. Due to transmission delay before a task arriving at a *VEU*, the load status when the scheduling decision is made is always different from the load status when the task is actually executed in the *VEU*. The high dynamicity of virtual machines amplifies this difference. So we first need to predict the workload when a task arrives at one *VEU* in the scheduling.

Dinda and O'Hallaron [28] studied the prediction power of several models and show that simple practical models such as auto-regression (*AR*) are sufficient for load prediction. *AR*(*n*) is a common linear forecast model where *n* represents the number of steps which are used for forecasting a new one. In this paper, we use the *AR*(16) model to get the load indicators of a *VEU* node in the VDC, including CPU process queue length and memory utilization.

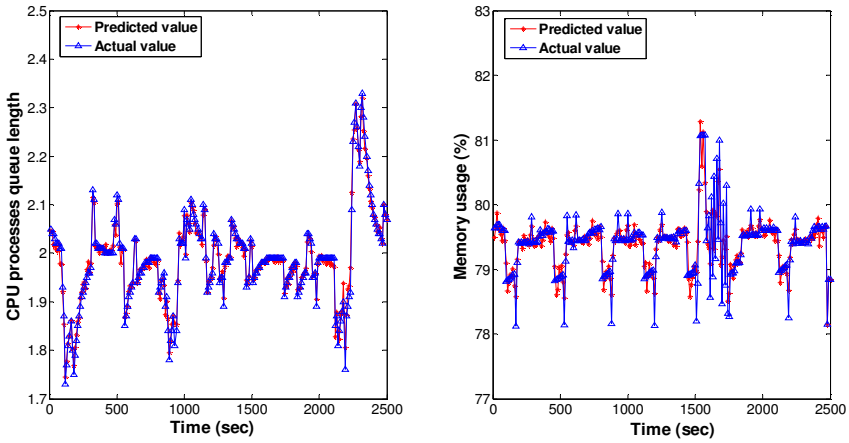

Fig. 3. The workload prediction results of *AR*(16) vs. the actual load of the ChinaGrid nodes.

In order to acquire the more reasonable model parameters which approach the realistic systems, we develop a sensor program and deploy it to eight server nodes in a practical data center of ChinaGrid [29]. Every second, the sensors record the workload data including CPU process queue length and memory utilization. The sensors totally collect the records for 5 hours. We use the multi-steps load prediction technology [27]. Fig. 3 shows the testing result of prediction of the *AR*(16) model. The mean absolute measurement errors of the prediction results of CPU process queue length and memory utilization are 3.17% and 2.85%, which are with an acceptable error range (less than 5%).

2) Infer the load status of each server node

We use the two load indicators, the CPU process queue length and memory utilization, to infer the overall load status of a server node, and then we will determine the load-balancing status of the whole virtualized data centers and select the candidate *VEU*s set. Interval type-II FLS (IT-2 FLS) is used, and we only need to determine the form of the primary membership function (MF) (because the secondary MF is interval sets in IT-2 FLS). We analyze the data traces of the server nodes in the practical data center of ChinaGrid to get the information of parameters distribution.

The statistical distribution curves of the CPU process queue length and memory utilization are respectively shown in Fig. 4. We can approximately fit them with Gaussian distribution curves. Then we partition the data into segments, and compute the mean value and standard deviation for each segment. Table Ⅱ also shows that the variation of standard deviation is much bigger than the variation of mean value. So the primary membership can be modelled by a type-II Gaussian MF with uncertain variance [23] (see Fig. 5).

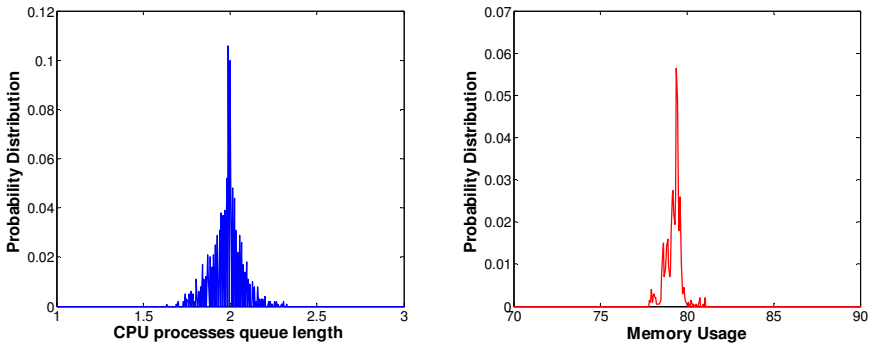

Fig. 4. The statistical distribution of CPU queue length and memory usage of the server nodes in ChinaGrid.

Table Ⅱ. The statistical analysis of CPU and memory load of the server node in ChinaGrid.

Data set	CPU process queue length		Memory utilization	
	Mean	Std	Mean (%)	Std
Segment 1	1.9815	0.083972	79.0952	0.30745
Segment 2	2.0451	0.073998	79.1565	0.59859
Segment 3	1.9736	0.121144	79.0798	0.37352
Segment 4	2.0048	0.085025	79.1543	0.35758
Segment 5	1.9408	0.077651	79.2661	0.33718
Segment 6	2.0062	0.067939	79.2671	0.34818
Segment 7	1.9549	0.085049	79.3451	0.35803
Segment 8	1.9912	0.048164	79.433	0.58207
Segment 9	2.0205	0.112657	79.349	0.37335
Segment 10	1.9538	0.057133	79.449	0.33759
Entire Data Set	1.9872	0.0809	79.2595	0.38997
Normalized Std	0.01650	0.276175	0.001695	0.26069

We construct the rules set of the FLS. Antecedent and consequent membership functions are chosen based on the statistical analysis of the data traces. The load level is divided into: *Light*, *Medium_to_Light*, *Medium*, *Medium_to_Heavy*, *Close_to_Heavy*, and *Heavy*. We construct the fuzzy inference rules set which has 36 items. For example, under the condition that CPU process queue length is more than *Close_to_Heavy*, CPU process queue length has a more important influence over memory utilization. However, when CPU process queue length is equal to or less than *Medium*, memory utilization plays a more important role to system load status. We omit the complete rule table here for the space limitation. Then we get six candidate sets of the *VEU*s with the load levels from *Light* to *Heavy*.

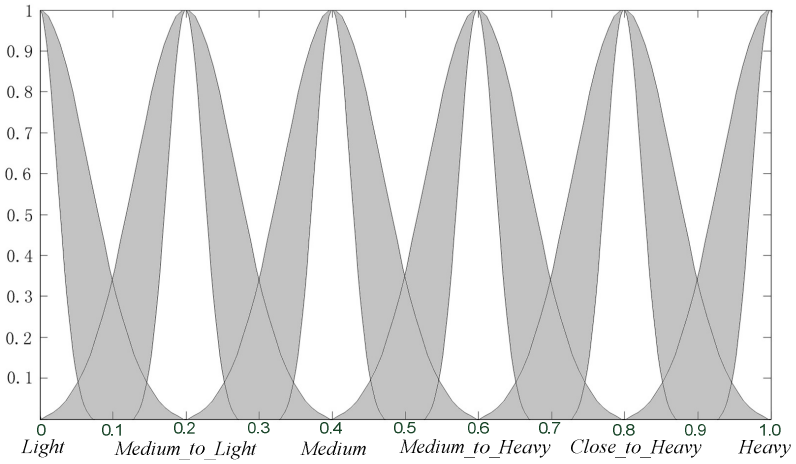

Fig. 5. Membership functions of the IT-2 FLS for predicting the load status of each server node.

3) Infer the load-balancing status of virtualized data center

The concept of load-balance is inherently vague. In this subsection, based on the load status prediction of each server node using IT-2 FLS in step 2, we use a simple type-I FLS to infer the overall load-balancing status of the whole virtualized data center. The inputs of the FLS are the percentage of the server nodes which have load status from *Light* to *Heavy* gotten in step 2, and the membership functions of the load-balancing status are shown in Fig. 6. There are total 55 items in the fuzzy inference rules table. For the space limitation, we don't list it here.

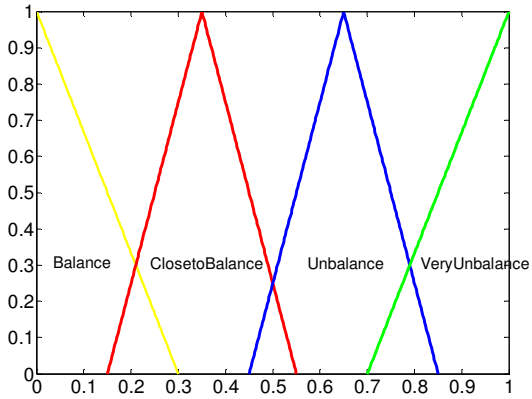

Fig. 6. Membership functions of the load-balancing status of the VDC.

4) Pick out the candidate sets according to load-balancing status

We pick out the candidate set from the six sets of *VEU*s gotten in step 2, according to the load-balancing status of the VDC in step 3. An intuitive idea is to dispatch the new coming task to the nodes with relatively lightweight load. In most cases, the more unbalanced the data center is, the larger the node set with lightweight load is. So when the system is in the *VeryUnbalance* status, the

node set with lightweight load is large enough in most cases. When the data center is relatively balanced, the nodes set with lightweight load is small and we can expand the candidate nodes set appropriately. According to this idea, we construct the rules for the FLS to select candidate nodes set as shown in Table Ⅲ.

Table Ⅲ. Candidate sets for different load-balance status.

Balance status	candidate machine set
veryUnbalance	*Light*
unBalance	*Light, Medium_to_Light*
ClosetoBalance	*Light, Medium_to_Light, Medium*
Balance	*Light, Medium_to_Light, Medium, Medium_to_Heavy*

We have shown how to determine the candidate *VEU* nodes set for each arrived task of the virtualized data center, considering the availability requirements and workload status. Then, which *VEU* node is selected from the candidate set as the scheduling aim depends on the certain scheduling algorithm. In next section, we will propose a novel dynamic task scheduling algorithm for VDCs.

5. THE PROPOSED SCHEDULING ALGORITHM: SALAF

5.1 The dynamic task scheduling algorithm

After building the availability and the load-balance fuzzy predictors, we use them in the general scheduling framework of virtualized data center, as shown in Fig. 1. We propose an on-line dynamic task scheduling algorithm called *SALAF* (*Scheduling Algorithm based on Load-balance and Availability Fuzzy prediction*). The details of the algorithm are depicted in Fig. 7.

Every arrived task is placed in the scheduling queue after it is submitted to the VDC. The scheduler gets tasks from the FIFO (First-In First-Out) queue. For each task at the head of the waiting queue, steps 4-10 select the candidate *VEU* set $\boldsymbol{A}$, which can satisfy the availability requirements of the task specified in the SLA. The FLS *AFP* (Availability Fuzzy Predictor) is applied to get availability level of each *VEU*. Steps 11-15 determine the candidate *VEU* set $\boldsymbol{L}$, which can meet the load-balance requirement. The FLS *LFP* (Load-balance Fuzzy Predictor) is used to get candidate *VEU* set that satisfy the load-balance requirement. After that, step 17 finds the intersection set $\boldsymbol{C}$ which can both satisfy the two requirements of availability and load-balance. If the set $\boldsymbol{C}$ is not null, steps 18-24 select the *VEU* which has the minimum excepted response time from $\boldsymbol{C}$ as the

pre-allocated aim *VEU*. When the set $\boldsymbol{C}$ is null, the scheduler gives priority to the satisfaction of the availability requirement. If $\boldsymbol{A}$ is not null, steps 26-31 select a *VEU* which has the minimum excepted response time in the set $\boldsymbol{A}$ as the pre-allocated aim. Otherwise, scheduler selects a *VEU* which has the highest availability level to satisfy the availability requirement as much as possible by steps 32-34. At last, the current task is allocated to the selected aim *VEU* in step 35.

1. Let $\boldsymbol{\Omega}$ denote the set of all *VEU*s;
2. Initialize the availability demand level for each class of tasks when they are submitted;
3. **For** each task at the head of the queue **Do**
4. Create two empty candidate sets of *VEU*s $\boldsymbol{A}$ and $\boldsymbol{L}$;
5. **For** each $VEU_j = \{HW_i, VMM_i, VM_{ik}\}$ in $\boldsymbol{\Omega}$ **Do**
6. Calculate the current availability of each component A_{HW_i}, A_{VMM_i} and $A_{VM_{ik}}$;
7. Apply the *FLS* $AFP\left(A_{HW_i}, A_{VMM_i}, A_{VM_{ik}}\right)$ (see subsection 4.1) to get the availability level of the VEU_j;
8. **If** the VEU_j can satisfy the task availability requirements **Then**
9. Add VEU_j to set $\boldsymbol{A}$;
10. **End if**
11. Apply *AR*(16) to predict the CPU process queue length(CPU_i) and memory utilization (MEM_i);
12. Apply $LFP(CPU_i, MEM_i)$ to infer the load-balancing status of VDC if the task is assigned to VEU_j;
13. **If** the VEU_j can satisfy the load-balancing demand **Then**
14. Add VEU_j to set $\boldsymbol{L}$;
15. **End if**
16. **End for**
17. Let $\boldsymbol{C} = \boldsymbol{A} \cap \boldsymbol{L}$, $T_{\min} = INFINITE$;
18. **If** $\boldsymbol{C} \neq \varnothing$ **Then**
19. **For** each $VEU_j \in \boldsymbol{C}$ **Do**
20. Let $p_{ij} = 1$; calculate expected response time *T* (*see Eq. 3*);
21. **IF** $T < T_{\min}$ **Then**
22. $T_{\min} = T; \quad aim = j$;
23. **End if**
24. **End for**
25. **Else if** $\boldsymbol{A} \neq \varnothing$ **Then**
26. **For** each $VEU_j \in \boldsymbol{A}$ **Do**
27. Let $p_{ij} = 1$; calculate expected response time *T;*
28. **IF** $T < T_{\min}$ **Then**
29. $T_{\min} = T; \quad aim = j$;
30. **End if**
31. **End for**
32. **Else**
33. Select one of the *VUE*s which have the highest availability level as the *aim* ;
34. **End if**
35. Allocate current $task_i$ to the VEU_{aim}
36. **End for**

Fig. 7. The scheduling algorithm *SALAF*.

5.2 Algorithm analysis

SALAF is an on-line dynamic task scheduling algorithm. We analyze its worst-case time complexity to get the computational overhead for each arriving task in the VDC.

[THEOREM 1]: The algorithm *SALAF* has approximate linear worst-case time complexity with the total number of the virtual machines in the VDC.

PROOF: The most time-consuming operations are the steps to find the candidate *VEU* set from step 5 to 16. There are *N* rounds for the *N VEU*s. In each round, the fuzzy predictors *AFP* and *LFP* are used. As shown in section 4, the time complexity of the fuzzy prediction process is $\boldsymbol{O}(\tau \cdot K)$ for *K* classes of tasks, where τ is a constant and it is in proportion to the number of rules in the FLSs. So the worst-case time complexity of the steps 5-16 is $\boldsymbol{O}(\tau \cdot N \cdot K)$. Steps 17-24, steps 25-31 and steps 32-34 are to select a *VEU* in some subset of the candidate *VEU* set. The worst-case time complexity of them is all $\boldsymbol{O}(N)$. Hence the total worst-case time complexity of *SALAF* is $\boldsymbol{O}(\tau \cdot N \cdot K + N)$. Since τ is a constant and the number of task classes *K* is a non-big integer in practical VDCs, *SALAF* can be approximately considered having linear worst-case time complexity algorithms which is positively related to the numbers of *VEU*s *N* in the VDC. And the number of *VEU*s equal to the number of virtual machines, so *SALAF* has approximate linear worst-case time complexity with the total number of the virtual machines in the VDC. □

6. SIMULATIONS

In this section, we evaluate the *SALAF* using simulation experiments. It is assumed that the task arrival conforms to Poisson process, and the task execution times are uniformly distributed. The parameters related to the availability including the failure rate and repair rate of HW, VMM and VM are chosen to represent the characteristics of software and hardware for real-world systems [26]. The parameter settings are shown in Table Ⅳ.

Table Ⅳ. The parameter settings of experiment environment.

Parameters	Values (Fixed)→(Varied)
Number of virtual servers (i.e., number of HW and VMM)	(10)→(4,6,8,10,12,14)
Total Number of *VEU*s	(30)→(10,20,30,40,50,60)
Average task arrival rates	(1.0)→(0.4,0.6,0.8,1.0,1.2,1.4)
Average failure rates of hardware	(0.00001:0.0001:0.001)

Average failure rates of VMM and VM	(0.001:0.01:0.1)
Average repair rate of hardware	0.1
Average repair rate of VMM and VM	1
Average process rates of *VEU*	(0.1:0.05:0.01)
Average transmission delay	10

We select two scheduling algorithms in common use to compare with the *SALAF*. They are *Min-min* and *Sufferage* [16, 30]. These two algorithms are selected because they are two representative dynamic scheduling algorithms and have been widely used in real systems. They are described in brief as follows:

(1) *Min-min*: For each submitted task, the *VEU* providing the earliest completion time is tagged. Among all of the mapped tasks, the one that has the minimal earliest completion time is chosen and then allocated to the tagged *VEU*.

(2) *Sufferage*: A *VEU* is assigned to a task that would "suffer" most in terms of completion time if that *VEU* is not allocated to the task.

There are two major evaluating indicators: Average Response Time (*ART*) of each task, and the Availability Satisfaction Percentage (*ASP*) that is the percentage of the tasks whose availability requirements can be satisfied by the assigned *VEU*s. We study the performance of the three scheduling algorithms under different task arrival rates and different structure parameters of the data center. The parameter settings of the experimental environment are shown in table Ⅳ.

Fig. 8 (a) and (b) show the changes of *ASP* and *ART* respectively when the tasks arrival rate varies from 0.4 to 1.4. We can see that the *ASP* of the *SALAF* is considerably higher than *Min-min* and *Sufferage*, while the *ART* of the *SALAF* is close to those of *Min-min* and *Sufferage*. It is because that *Min-min* and *Sufferage* take the best time performance as the only goal and ignore the availability requirements of tasks. It shows that the *SALAF* scheme can improve the system availability while maintaining good responsiveness.

Fig. 9 (a) and (b) show the changes of *ASP* and *ART* respectively when the number of hardware machines varies from 4 to 14. It also shows the advantages of the *SALAF* in *ASP* while keeping a good *ARP*. Besides, Fig. 9 (a) and (b) also show that both *ASP* and *ART* are enhanced rapidly, with increasing the number of hardware machines. It matches the common sense: the more the hardware machines, the less the average number of virtual machines on each machine, so each machine has

lighter workload, and it is easier to guarantee better availability and responsiveness performance. However, using more machines means more construction cost of VDC and worse utilization of hardware resource, so it needs a trade-off for building a VDC.

The VMs are the main executive units in a VDC. There are usually several VMs running on each hardware server, called *server consolidation*. We study the impact of the average number of VMs on each server (called *consolidation ratio*) on the availability and performance of task scheduling schemes. Fig. 10 (a) and (b) show the changes of *ASP* and *ART* respectively when the average number of VMs on each virtual server varies from 1 to 6. As more virtual machines are consolidated into each server, the availability and responsiveness are improved. Comparing with the way to increase the number of machines, server consolidation is a much cost-saving way. On the other hand, the consolidation ratio is not the bigger the better. Fig. 10 (a) and (b) show that when the consolidation ratio exceeds a threshold value, the improvement of availability and response time is not obvious. Considering the cost of consolidation, it seems that there exists an optimal consolidation ratio in a VDC that may be related to the hardware resource and the workload, which is another issue to be considered in our future works.

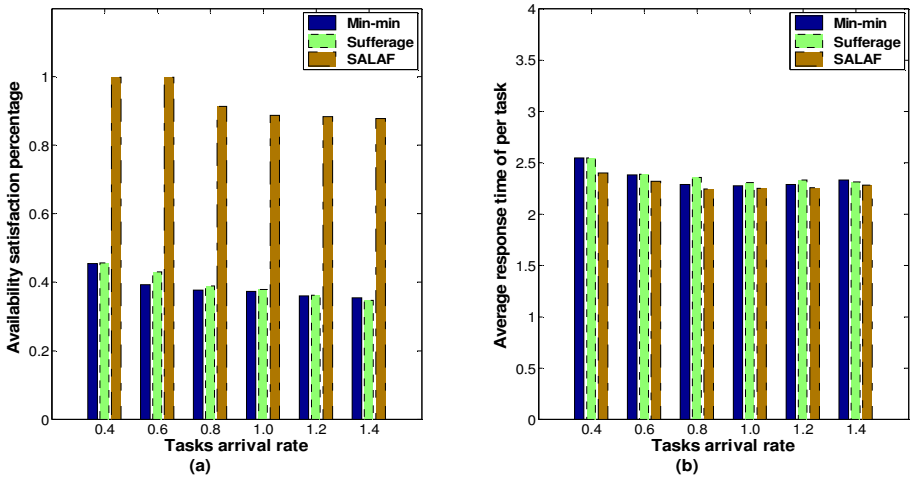

Fig. 8. ASP (a) and ART (b) vary with the tasks arrival rate.

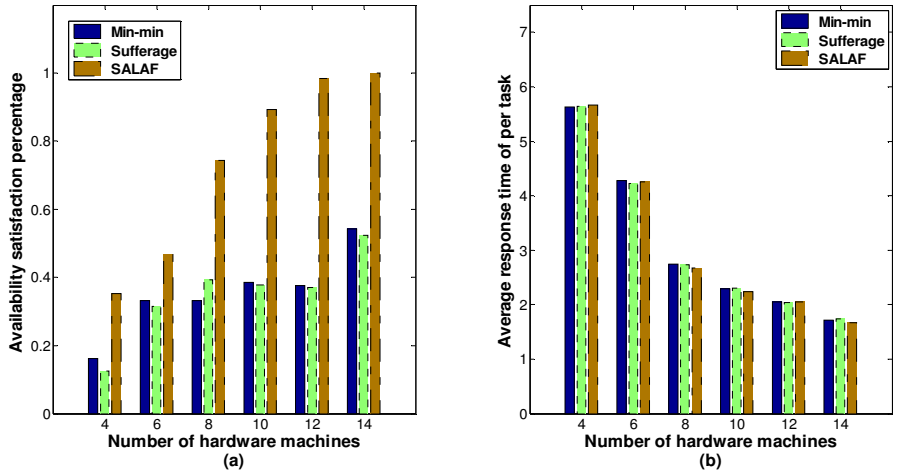

Fig. 9. ASP (a) and ART (b) vary with the number of hardware machines.

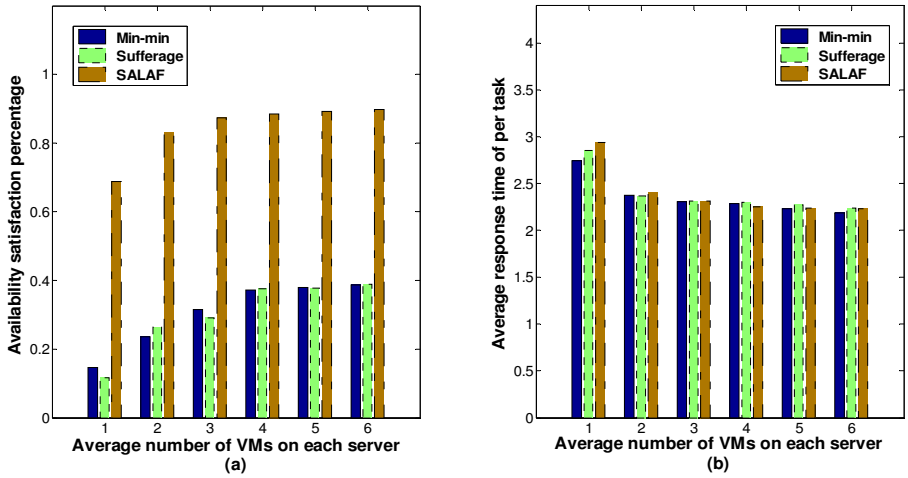

Fig. 10. ASP (a) and ART (b) vary with the average number of VMs on each server.

7. CONCLUSIONS

In this paper, we have studied the task scheduling problem in virtual data centers, considering the performance and availability requirements of SLAs. The general model of the task scheduling in VDC is built by *MSQMS-LQ*, and the problem is formulated as an optimization problem with two objectives: average response time and availability satisfaction percentage. Then we give a graceful fuzzy prediction method to model the uncertain workload and the vague availability of virtualized server nodes, by using the type-I and type-II fuzzy logic systems. Based on the fuzzy prediction systems, an on-line dynamic task scheduling algorithm named *SALAF* is proposed. The worst-case time complexity of *SALAF* is analyzed. The experimental results show that the proposed algorithm

could efficiently improve the total availability of VDCs while maintaining good responsiveness performance.

ACKNOWLEDGMENTS

We are much obliged to Prof. Zuhtu Hakan Akpolat (Firat University, TURKEY) for the interval type-II FLS Matlab toolbox.

This work is financially supported by the National Grand Fundamental Research 973 Program of China (No. 2010CB328105, No. 2009CB320504), and the National Natural Science Foundation of China (No.60932003, No. 90718040).

REFERENCES

[1] Joseph D, Tavakoli A, Stoica I. "A Policy-aware Switching Layer for Data Centers". *ACM SIGCOMM Computer Communication Review*, Vol. 38, Issue 4, pp. 51-62, 2008.

[2] Arregoces M, and Portolani M. "Data Center Fundamentals". *Cisco Press*, 2003.

[3] Snevely R. "Enterprise Data Center Design and Methodology". *Sun Microsystems, Inc. Prentice Hall*. 2002.

[4] Snyder J. "Data Center Growth Defies Moore's Law". *InfoWorld*, 2007. http://www.pcworld.com/article/id,130921/article.html.

[5] S. Govindan et al. "Xen and Co.: Communication-Aware CPU Management in Consolidated Xen-Based Hosting Platforms". *IEEE Transactions on Computers*, vol 58 , issue 8, 2009

[6] Graupner S, Kotov V, Trinks H. "Resource-Sharing and Service Deployment in Virtual Data Centers". *In Proceedings of the 22nd International Conference on Distributed Computing Systems*, pp. 666 – 674, 2002

[7] Xu J, Zhao M, Fortes J, Carpenter R, Yousif M. "On the Use of Fuzzy Modeling in Virtualized Data Center Management". *In Proceedings of the 4th International Conference on Autonomic Computing (ICAC '07)*, 2007.

[8] Zhu X et al. "1000 Islands: An Integrated Approach to Resource Management for Virtualized Data Centers". *Cluster Computing*, Vol 12, No 1, pp. 1573-7543 (Online), 2009.

[9] Luis M. Vaquero, Luis Rodero-Merino, Juan Caceres, Maik Lindner. "A break in the clouds: towards a cloud definition". *ACM SIGCOMM Computer Communication Review*, Vol. 39, Issue 1, pp. 50-55 Jan 2009.

[10] Xiangzhen Kong, Jiwei Huang, Chuang Lin, Peter Ungsunan. "Performance, Fault-tolerance and Scalability Analysis of Virtual Infrastructure Management System". *IEEE International Symposium on Parallel and Distributed Processing with Applications (ISPA'09)*, Auguest, 2009.

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

[12] VMware, "VMware vCloud". http://www.vmware.com/technology/cloud-computing.html

[13] Marty M and Hill M. "Virtual Hierarchies to Support Server Consolidation". *in Proceedings of the 34th annual international symposium on Computer architecture*, pp. 46-56, 2007.

[14] VMware Server Consolidation, 2009. http://www.vmware.com/solutions/consolidation

[15] Clark C, Fraser K, Hand S, Hansen J, Jul E, Limpach C, Pratt I, and Warfield A. "Live migration of virtual machines". *In Proceedings of the Second Symposium on Networked Systems Design and Implementation (NSDI'05)*, May 2005.

[16] Qin X, Xie T. "An Availability-Aware Task Scheduling Strategy for Heterogeneous Systems". *IEEE Transactions on Computers*, Vol. 57, No. 2, Feb, pp. 188-199, 2008.

[17] Wood T, Shenoy P, Venkataramani A, and Yousif M. "Black-box and Gray-box Strategies for Virtual Machine Migration". *In Proceedings of the Fourth Symposium on Networked Systems Design and Implementation (NSDI'07)*, Cambridge, MA, April 2007.

[18] Ying Song, Hui Wang Yaqiong Li,Binquan Feng,Yuzhong Sun. "Multi-Tiered On-Demand Resource Scheduling for VM-Based Data Center". *In Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid'09)*, 2009, pp. 148-155.

[19] Xiaoqiao Meng,Vasileios Pappas, Li Zhang,Thomas J. Watson. "Improving the Scalability of Data Center Networks with Traffic-aware Virtual Machine Placement". *The 29th Conference on Computer Communications (Infocom'10)*, March, 2010. To appear, [online] http://www.cs.ucla.edu/~xqmeng/paper/TVMPP.pdf

[20] Dubois D, Fargier H, Fortemps P. “Fuzzy scheduling: Modelling flexible constraints vs. coping with incomplete knowledge”. *European Journal of Operational Research*, Vol.147, Issue 2, pp. 231–252, 2003.

[21] Zadeh L. “The concept of a linguistic variable and its application to approximate reasoning—I”. *Inform. Sci*., Vol. 8, pp. 199–249, 1975.

[22] Karnik N, Mendel J, and Liang Q. “Type-2 Fuzzy Logic Systems”. *IEEE Transactions on Fuzzy System*, Vol. 7, No, 6, pp. 643-658, Dec. 1999.

[23] Liang Q and Mendel J. “Interval type-2 fuzzy logic systems: theory and design”. *IEEE Transactions on Fuzzy System*, Vol. 8, No. 5, pp. 535–550, Oct. 2000.

[24] Sethuraman J and Squillante M. “Optimal Stochastic Scheduling in Multiclass Parallel Queues”. *In Proc. ACM SIGMETRICS*, 1999.

[25] Kleinrock L. “Queueing Systems: Volume I: Theory”, *John Wiley & Sons*, New York, 1975, pp. 187.

[26] Birolini A. “Reliability Engineering Theory and Practice”, Springer-Verlag Berlin Heidelberg, 2007.

[27] Wu Y, Yuan Y, Yang G, Zheng W. “Load prediction using hybrid model for computational grid”. *In Proc. 8th IEEE/ACM International Conference on Grid Computing (GRID)*, pp. 235-242, 2007.

[28] Dinda P and O'Hallaron D, “Host load prediction using linear models”. *Cluster Computing* ,3, 4 (2000).

[29] The ChinaGrid website. [Online]. Available: http://www.chinagrid.edu.cn.

[30] Song S, Hwang K, and Kwok Y. “Risk-Resilient Heuristics and Genetic Algorithms for Security-Assured Grid Job Scheduling”. *IEEE Transactions on Computers*, Vol. 55, No. 6, pp. 703-719, June 2006.