An Attribute-based Availability Model for Large Scale IaaS Clouds with CARMA

An Attribute-based Availability Model for Large Scale IaaS Clouds with CARMA

Hongwu Lv, Jane Hillston, Paul Piho, and Huiqiang Wang

Abstract—High availability is one of the core properties of Infrastructure as a Service (IaaS) and ensures that users have anytime access to on-demand cloud services. However, significant variations of workflow and the presence of super-tasks, mean that heterogeneous workload can severely impact the availability of IaaS clouds. Although previous work has investigated global queues, VM deployment, and failure of PMs, two aspects are yet to be fully explored: one is the impact of task size and the other is the differing features across PMs such as the variable execution rate and capacity. To address these challenges we propose an attribute-based availability model of large scale IaaS developed in the formal modeling language CARMA. The size of tasks in our model can be a fixed integer value or follow the normal, uniform or log-normal distribution. Additionally, our model also provides an easy approach to investigating how to arrange the slack and normal resources in order to achieve availability levels. The two goals of our work are providing an analysis of the availability of IaaS and showing that the use of CARMA allows us to easily model complex phenomena that were not readily captured by other existing approaches.

Index Terms—availability, cloud computing, formal model, CARMA.

1 Introduction

THROUGH delivering IT services as computing utilities, cloud computing brings the promise of cost reduction for business computing [1]. Infrastructure as a Service (IaaS) is a classical form of cloud system, and supplies low level computing resources on which end users are able to deploy and run arbitrary Operating Systems (OS) and applications without managing the underlying infrastructure. IaaS cloud providers aim to guarantee a high level of availability to the consumer, captured by a service level agreement (SLA). Here, the availability of the IaaS is taken to mean the proportion of requests for one or more virtual machines (VMs) that can be satisfied. According to the investigation in reference [2], currently most IaaS cloud providers offer SLAs in terms of guaranteed availability. Note that unavailability will not only occur due to system failure, but also due to contention or mismatch between the resource configuration and the arriving requests, as we will see. Thus, even for a system with high machine availability (e.g. 99.9% system availability means 42 minutes of downtime per month) the IaaS availability may not be satisfactory. From the provider's perspective, any availability violation may cause the loss of revenue and damage business reputations [3]. Hence, maintaining the availability of IaaS is of significant interest.

For large IaaS clouds that contain millions of heterogeneous computing nodes, analysis based on models is generally focused on identifying key factors without being tied to specific details or particular applications [4]. At present, many formal methods have been used to study the performance and availability of IaaS, for example, Markov chains [5] [6] , Petri nets [7] [8] [9], fault trees [10] [4] and so on. However, there are two challenges not captured by previous IaaS availability models.

(i) The most important one is the effect of *Super-tasks* [11] on availability. Super-tasks are groups of tasks that communicate frequently or share a series of resources. Such tasks must usually be deployed in the same Physical Machine (PM) to increase efficiency and reduce failure. This phenomenon has some apparent impacts on the availability of IaaS. One issue is that existing cloud scheduling strategies are likely to be less effective when a group of tasks have to be deployed together. Meanwhile, perhaps fatally, burst tasks with ultra large task size may not find PMs to accommodate them, which greatly decreases the availability of cloud systems. Several critical factors have been studied in this field such as queuing [12], failure of PMs [13], VM migration [6], VM deployment [5] [14] and so on. But the primary focus of those papers is the performance of pool scheduling or management policies rather than availability. In this paper, we use task *size* to denote the number of VM required by a job, and we will investigate the impact of task size on availability. Recently, [15] [9] considered the number of vCPUs requested by each customer job. However, these authors compute the instantaneous probability of a job departure by solving Markov chains [15] [9] [11]. This approach is untenable if the execution rate, capacity, or any other features differ between PMs or evolve over time, especially when the size of tasks may be sequences of fixed numbers or some complex distribution.

(ii) The second challenge is that existing models are not flexible enough to describe differing task arrival patterns and distinctive features across PMs. Since there are various workloads with different parameters such as start time, end time, duration of tasks and so on, a large number of workload sub-models have to be established to approximate real workloads. Moreover, PMs in real cloud data centers may be from different manufacturers, or have been upgraded or

- *H. Lv and H. Wang were with the Department of Computer Science and Technology, Harbin Engineering University, Harbin, China, 150001. E-mail: lvhongwu@hrbeu.edu.cn*
- *Jane Hillston and Paul Piho are with the University of Edinburgh.*

Manuscript received March 28, 2018; revised .

replaced. They may have distinct features including different capacity, different configuration, different Mean Time To Repair (MTTR) and so on [9]. Thus the behaviors of PMs are heterogeneous due to their distinct features which may have an impact on availability analysis.

In this paper, we propose an attribute-based availability model of IaaS to enhance the ability to describe large scale cloud systems. In the model, each formal component such as the PM or a task specifies its private attributes. The processes formed as a composition of components may have different activities according to their attributes. The attribute-based model is also flexible enough to analyze the impacts of super-tasks. The model is written in CARMA, a powerful modeling formalism developed by the University of Edinburgh and the University of Florence. The main innovations are outlined as follows:

1) To our knowledge, this is the first investigation of the impact of task size on availability of an IaaS. We analyze the impact of task size following a normal distribution, a uniform distribution, a log-normal distribution or given by a fixed integer value.
2) Addressing the challenge of determining the completion time of a super-task. In the prior research [9] [15] [5] [12], the task completion time is obtained by solving a Markov process, but the PMs are treated as identical i.e. the same kind of machine, by default. In contrast, in real clouds, the execution rate, or capacity, or other features differ between PMs, and the current approach will be very inefficient or even infeasible. In our work, responding to a broadcast, a PM can autonomously decide whether it can accept a new job with its remaining capacity. Moreover, using the location attribute, PMs can execute concurrently and autonomously to determine job completion times.
3) Using attribute-based components, the heterogeneity of both task arrival patterns and PMs in IaaS can be modeled to reflect unique characteristics. Taking into account all features from capacity to execution mode rather than just the number of vCPUs [9] or the number of requested physical cores in VMs [15]. This is powerful enough to capture periodic tasks with different cycle lengths, burst tasks with different duration times, and PMs with an inherited ageing phenomenon. More importantly, the use of attributes allows complex behaviors of PMs to be captured in a single parameterized submodel.
4) Using *measure* functions, another characteristic of the CARMA specification language, we also provide an easy approach to investigating how to arrange slack and normal resources to achieve different availability levels when there are super-tasks with different task sizes.

To conclude, the two goals of our work are to analyse the availability of IaaS with respect to task size and to show that the use of CARMA allows us to model complex phenomena that were not readily captured by existing approaches.

The remainder of the paper is organized as follows. We briefly introduce the background in Section 2. Section 3 presents our availability model in detail. In Section 4, we present some simulation experiments to validate and analyze our model. Our findings are summarized in Section 5, where we also outline directions for future work.

2 Background

2.1 IaaS and Availability

IaaS availability model Availability is one of the core properties of cloud systems. To analyze and assess the availability of IaaS clouds, many approaches have been proposed. According to [4] and [16], formal analysis models are more focused on identifying key factors without being tied to specific details, and so are more suitable for application in large cloud systems compared to simulation tools (e.g. Cloudsim and NS2). Moreover, the availability of IaaS is affected by all factors that mean that resource requests cannot be granted, including network connection interruption, VM failure, hardware failure and mismatch between the requested resource and what can currently be deployed. In this paper, as in many others, e.g. [11] [17], we only consider the deployment aspects of availability; i.e. we assume that the infrastructure is failure free and consider only the provisioning of available resources to meet incoming job requests.

Furthermore, for large IaaS clouds with vast parameter spaces, a monolithic model may suffer from poor scalability [18] [11]. Thus hierarchical models are often adopted to construct a series of interacting sub-models. In 2009, Kim *et al.* [10] established a two-level hierarchical availability model to incorporate the effect of hardware failures, software failures and VM live migration. In this hierarchical model, fault trees are used for the upper level and homogeneous continuous time Markov chains (CTMC) are used to represent submodels at the lower level. But the differences between PMs are not considered. In [19], a hierarchical model based on Stochastic Reward Networks is constructed to analyze performance, availability, and power consumption; in [17] [18] the authors studied the impact of failures on IaaS availability, when failure is typically dealt with through migration of VMs; Chilwan *et al.* [20] investigated the effect of dynamic load in a cloud cluster on the service availability using analytical models; Ghosh *et al.* [7] researched the impact of I/O on availability. In those models, the PMs are classified into three types: hot, warm (hot standby), cold (cold standby), since different types of machines may need different times to deploy a VM with noticeable effects on availability. Furthermore, other models investigate concepts related to the availability of IaaS, such as capacity planning [21], VM migration [22] [23], sensitivity analysis [3], performance [24]. In those works, the effects of VM scheduling and PM management are considered in ways similar to previous availability models.

Recently, many cloud providers have started to offer OS container-based services (e.g., Docker [25], OpenVZ [26]) to customers. Compared to traditional VMs, a key feature of a container is that it is lightweight, providing a promising computing paradigm for cloud computing in the future. But security isolation is still a major concern [27] [4], and it will be hard to completely replace VM-based clouds before this issue is completely resolved. In [4], two scenarios are modeled to assess the availability of clouds, firstly when container instances are directly executed on the PM OS, and secondly when containers are grouped and run on VMs. Several papers [4], [28] and [29], have studied the availability of container-based clouds through model analysis, but

they do not consider the phenomenon of super-tasks, which is the focus of this paper.

The impacts of super-tasks In reality, users can request one or more VMs at a time [12]. Since cloud providers must ensure every users' availability according to SLO (Service Level Objectives) no matter how many tasks arrive in the next seconds, the phenomenon of super-tasks brings a significant challenge to the availability of IaaS. As a typical feature of workload variations, super-tasks within clouds were first discussed by Khazaei *et al.* in 2011 [14]. Thereafter, a large number of critical factors were studied in this field such as queuing [12], failure of PMs [13], VM migration [6], VM deployment [5] [14] and so on. Recently Chang *et al.* [9] proposed VM size to consider the number of vCPUs requested by each customer job. Asadi *et al.* [30] investigated the impact of resource heterogeneity on the power and performance in four different scenarios where the servers can run 2, 4, and 8 VMs according to their capacities. But the impact of task size has not been studied thoroughly, especially when the behavior and capacity of PMs are highly heterogeneous. In this paper, we attempt to examine the impact of task size following different distributions.

In most prior models, all PMs are treated as the same kind of machine and organized in pools. Yet there are millions of off-the-shelf PMs in real cloud data centers. PMs in the same pool may have distinct features including different capacity, different execution rate, different MTTR and so on [9], because they may be from different manufacturers, or have been upgraded or replaced. Thus the behaviors of PMs are heterogeneous due to their distinct features. However, in most of the relevant models the job completion time is computed by solving a Markov chain in which all differences in features of PMs are ignored even though they may have impacts on availability and performance analysis.

Task arrival patterns Similarly to task size, task arrival patterns also have a significant impact on availability. In 2012, Reiss *et al.* [31] gave a detailed analysis of workload heterogeneity and variability through the analysis of Google trace data. According to Reiss' results, the variation of workload is very complicated and hard to describe using a single distribution. In order to simulate and examine heterogeneous workloads, many workload models have been proposed encompassing a variety of task arrival patterns. For example, in [32], the workloads are divided into three kinds of task arrival pattern including constant tasks, periodic tasks and burst tasks. This model is used to describe complex workloads and to analyze the impact of task arrival patterns. Furthermore, some other arrival patterns such as on & off, predictable bursting and unpredictable bursting are identified. However, since there are various workloads with different parameters such as start time, end time, duration of tasks and so on, researchers of these prior models had to establish a large number of workload sub-models to approximate a real workload. Therefore, there is a need to build a new and flexible model to describe the different features of super-task arrival.

Provisioning of slack resources For an ideal cloud system there are as many PMs as needed. But in reality, large scale clouds such as Google Cloud, Amazon AWS and Ali Cloud always set a threshold for their users, and out of range resources may be accessed probabilistically but not absolutely. Thus the provider needs to provide slack resources to have a balance between availability and revenue. In [33], Carvalho *et al.* proposed a prediction method to reclaim slack resources to increase revenue, which derived from a substantial dataset from Google Cloud. However, it has not been reported how to support the optimization of slack resources in the context of super-tasks.

To conclude, the differences between previous work and this paper are compared in Table 1. It can be seen that this is the first investigation of the impact of task size following different distributions on availability of an IaaS.

2.2 CARMA

CARMA is a new stochastic process algebra for the representation of systems developed in the Collective Adaptive Systems (CAS) paradigm [35]. The language offers a rich set of communication primitives, and exploits *attributes*, captured in a *store* associated with each component, to enable attribute-based communication. For example, for many CAS systems the location is likely to be one of the attributes. Thus it is straightforward to model systems in which, for example, there is limited scope of communication, or interaction is restricted to co-located components, or where there is spatial heterogeneity in the behavior of agents.

A CARMA system consists of a *collective* operating in an *environment*. The collective is a multiset of components that models the behavior of a system; it is used to describe a group of interacting *agents* that cooperate to achieve a given set of tasks. The environment models all those aspects which are intrinsic to the context where the agents are operating, i.e. the environment mediates agent interactions. This is one of the key features of CARMA. The environment is not a centralized controller but rather something more pervasive and diffusive — the physical context of the real system — which is abstracted as the environment, exercising influence and imposing constraints on the different agents in the system. Specifically the environment is responsible for setting the rates at which actions are performed, and probabilities of receiving a given message. For example, in a model of a cloud system, the environment will determine the rate at which entities (PM, task, etc) implement their jobs, which may also depend on the current time. The role of the environment is also related to the spatially distributed nature of CAS — we expect that the location *where* an agent is will have an effect on *what* an agent can do.

A CARMA component captures an *agent* or entity in the system. It consists of a process, that describes the agent's behavior, and of a store, that models its *knowledge*. A store is a function which maps *attribute names* to *basic values*.

Processes located within a CARMA component interact with other components via the defined communication primitives. Specifically, CARMA supports both unicast and broadcast communication, and permits locally synchronous, but globally asynchronous communication. Distinct predicates (boolean expressions over attributes) associated with senders and potential receivers are used to filter possible interactions. Thus, a component can receive a message only when its store satisfies the target predicate. Similarly, a receiver also uses a *predicate* to identify accepted sources. An interaction will occur only when the sender satisfies the

TABLE 1
Comparison with previous analysis models.

Reference	Topics (VM/PM/Container)	Considering super-tasks / different task arrival patterns	Considering PM (VM) heterogeneous while computing job completion time
[10]	availability (hardware, software, VM)	no	–
[19] [18] [20] [34] [17]	availability (PM & VM)	no	no
[6]	VM migration (PM & VM)	super-task	no
[14] [5]	PM &VM deployment (PM & VM)	super-task	no
[12]	queuing (PM & VM)	super-task	no
[11]	availability (PM & VM)	super-task	no
[13]	failure of PM (PM & VM)	super-task	no
[15]	heterogeneous VM (VM)	only vcpu cores	no
[9]	heterogeneous workload (PM & VM)	only VM size	no
[30]	performance trade-off (VM)	only VM size per PM	no
[2] [19] [24] [8]	performance (PM & VM)	no	no
[21]	capacity planning (PM & VM)	no	no
[22] [23]	VM migration (VM)	no	no
[4]	availability (container & VM)	heterogeneous VM	–
[28] [29]	availability (container)	no	no
[3]	sensitivity analysis (PM & VM)	no	–
[32]	heterogeneous workload (VM)	Constant/burst/periodic task	no
our work	availability (PM & VM)	both	yes

'–' means this item is not involved.

predicate used by the receiver, and the receiver satisfies the predicate used by the sender. The execution of communicating actions takes time, which is assumed to be an exponentially distributed random variable whose parameter is determined by the environment.

More formally, we let SYS be the set of CARMA *systems* S defined by the following syntax:

$$S ::= N \textbf{ in } \mathcal{E}$$

where N is a collective and $\mathcal{E}$ is an environment. We let COL be the set of collectives N which are generated by the following grammar:

$$N ::= C \mid N \parallel N$$

A collective N is either a *component* C or the parallel composition of collectives $N_1 \parallel N_2$. The precise syntax of components is:

$$C ::= \mathbf{0} \mid (P, \gamma)$$

and we let COMP be the set of components C generated by this grammar. A component C can be either the *inactive component*, denoted by $\mathbf{0}$, or a term of the form (P, γ), where P is a *process* and γ is a *store*. A store is a function which maps *attribute names* to *basic values*. We let:

- ATTR be the set of *attribute names* a, a', a_1,..., b, b', b_1,...;
- VAL be the set of *basic values* v, v', v_1,...;
- Γ be the set of *stores* $\gamma, \gamma_1, \gamma', \ldots$, i.e. functions from ATTR to VAL.

The behavior of a component is specified via a process P. We let PROC be the set of CARMA processes P, Q,... defined by the following grammar:

$$P, Q ::= act.P \mid [\pi]P \mid P + Q \mid P \mid Q \mid \textbf{nil} \mid \textbf{kill} \mid A(A \triangleq P)$$

$$act ::= \alpha^\star[\pi_s]\langle\overrightarrow{e}\rangle\sigma \mid \alpha[\pi_r]\langle\overrightarrow{e}\rangle\sigma \mid \alpha^\star[\pi_s](\overrightarrow{x})\sigma \mid \alpha[\pi_r](\overrightarrow{x})\sigma$$

$$e ::= a \mid \mathsf{my}.a \mid x \mid v \mid \mathsf{now} \mid \cdots$$

$$\pi_s, \pi_r, \pi ::= \top \mid \bot \mid e_1 \bowtie e_2 \mid \neg\pi \mid \pi \wedge \pi \mid \cdots$$

The process specifications are fairly standard, with prefix (first action), choice and parallel composition all with their usual meanings. A *predicate* π is used to indicate that the process is only active when the predicate is true. The distinguished process **kill** removes the enclosing component from the collective. In the action descriptions, the following notation is used:

- α is an *action type* in the set ACTTYPE;
- π_s and π_r are *predicates* that define filters on the acceptable communication partners;
- x is a *variable* in the set of variables VAR;
- e is an expression in the set of expressions EXP[1];
- $\overrightarrow{\cdot}$ indicates a sequence of elements;
- σ is an *update*, i.e. a function from Γ to $Dist(\Gamma)$ in the set of *updates* Σ; where $Dist(\Gamma)$ is the set of probability distributions over Γ.

Formally, an environment consists of two elements: a *global store* γ_g, that models the overall state of the system, and an *evolution rule* ρ, which is a function that, depending on the global store and on the current state of the collective

1. The precise syntax of expressions e has been omitted for brevity. We only assume that expressions are built using the appropriate combinations of *values*, *attributes* (sometime prefixed with my), variables and the special term now. The latter is used to refer to the current time.

(i.e., on the configurations of each component in the collective), returns a tuple of functions $\varepsilon = \langle \mu_p, \mu_w, \mu_r, \mu_u \rangle$:

- $\mu_p : \Gamma \times \Gamma \times \text{ACT} \to [0,1]$, $\mu_p(\gamma_s, \gamma_r, \alpha)$ expresses the probability that a component with store γ_r can receive a broadcast message from a component with store γ_s when α is executed;
- $\mu_w : \Gamma \times \Gamma \times \text{ACT} \to [0,1]$, $\mu_w(\gamma_s, \gamma_r, \alpha)$ yields the weight that will be used to compute the probability that a component with store γ_r can receive a unicast message from a component with store γ_s when α is executed;
- $\mu_r : \Gamma \times \text{ACT} \to \mathbb{R}_{\geq 0}$, $\mu_r(\gamma_s, \alpha)$ computes the execution rate of action α executed at a component with store γ_s;
- $\mu_u : \Gamma \times \text{ACT} \to \Sigma \times \text{COL}$, $\mu_u(\gamma_s, \alpha)$ determines the updates on the environment (global store and collective) induced by the execution of action α at a component with store γ_s.

To extract observations from a model, a CARMA specification also contains a set of *measures*. Each measure is defined as:

$$\texttt{measure}\ m\_name[var_1 = \theta_1, \ldots, var_n = \theta_n] = expr;$$

where θ_1, ..., θ_n refer to the range of variables. Expression $expr$ can be used to count or to compute statistics about attribute values of components operating in the system. These expressions are used to compute the minimum/maximum/average value of expression $expr$ evaluated in the store of all the components satisfying boolean expression $guard$, respectively.

The formal semantics of CARMA gives rise to an Inhomogeneous-time Continuous Time Markov Chain (ICTMC) and the details of the translation method can be found in [35]. The state space of the system is represented as a finite, discrete set of states and the times of state transitions are governed by the rates given by the model description. The state space generated by CARMA models is usually too large to be analytically tractable and thus the models are analyzed by simulating individual time trajectories.

The specification and analysis of CARMA models is supported by an Eclipse plug-in [36] and a model simulator. The plug-in implements an appropriate high-level language, named the CARMA *Specification Language*, that simplifies the creation of CARMA models by providing rich syntactic constructs inspired by main stream programming languages.

2.3 An example to illustrate the advantages of CARMA

In order to facilitate understanding of the advantages of CARMA in the modeling process, a simple example of PM ageing is given in this subsection. PM ageing is a common phenomenon in IaaS and other long-running systems. For the sake of simplicity, we only assume that the ageing of the PM is related to the number of times it is executed. The direct consequence of ageing will be that the PM execution rate will slow down. The CARMA fragment describing the ageing phenomenon is shown in Fig. 1.

There is a collection of PMs, and each of them is mapped to a local store $\gamma_s, s \in \{1, 2, \cdots, n\}$. In each store, the number of times each PM performs a task is different due to its inherent capacity and the order of the tasks are received. As soon as the action $execute$ occurs, a value will be updated

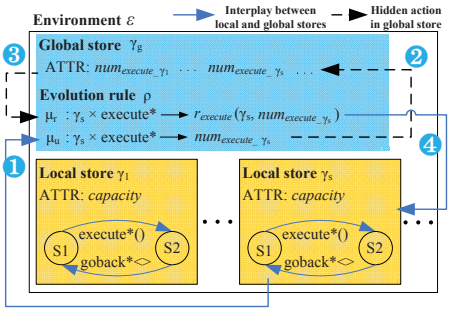

Fig. 1. A case study to illustrate the advantages of CARMA.

in the local store γ_s, as well as in the global store γ_g under the control of evolution rule function μ_u, resulting in the change of $num_{execute\_\gamma_s}$. Meanwhile, according to another evolution rule function μ_r, the rate of the execute action depends on the value of $num_{execute\_\gamma_s}$. Thus, under the supervision of μ_r, the execution rate of $execute$ in the local component will become lower. This feedback will make the execution rate more diverse across the PMs and it would be difficult to express with a single distribution function.

This case would be very complex to describe with other formal languages such as Petri nets; we would need to build each PM as a distinct submodel since the capacity and execution rate differ. However, it is easy to model this case with CARMA due to its inherited component structure and locally defined attributes.

3 AN ATTRIBUTE-BASED AVAILABILITY MODEL OF IAAS CLOUDS

In this paper, we build a new attribute-based availability model for large scale IaaS systems with CARMA, which allows us to model in an easy way complex phenomena that are not readily captured by existing approaches.

Based on previous research, the process of service provisioning for IaaS can be divided into 5 stages as in Fig. 2.

In contrast to the novel results that have been achieved in global queuing [12], VM deployment [5] and IO contention at runtime [7], the focus of this paper is on two aspects which have previously received little attention: super-task arrival (Section 3.1) and PM provisioning (Section 3.2). In our model, there are three kinds of tasks as in [32]. Each arriving task will be placed in the global queue. If the global queue is full, the job will be rejected and the availability is reduced at that time. Otherwise, every task in the queue is going to be selected. For most IaaS, the PMs are organized in PM pools classified as *hot*, *warm* and *cold*. But for different clouds, the scheduling policies may be quite different, for example, random choice, first fit, best fit, worst fit and so on. In this paper, the default strategy is best fit. Moreover, as in [32] [37], the slack resources will be reserved by providers to cope with bursty arrivals. When there are not sufficient reserved resources, a job can be uploaded to federation clouds to access opportunistic resources which are not guaranteed. Subsequently a job is allocated to a certain PM and deployed in one or more VMs, depending on its size. Finally, the VMs will be reclaimed when the job finishes.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2019.2943339, IEEE Transactions on Parallel and Distributed Systems

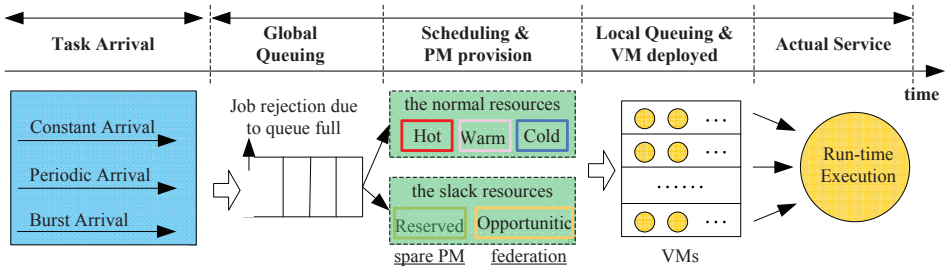

Fig. 2. The process of task arrival and service provisioning.

Correspondingly, the process of service provisioning is given in Fig. 3. It is modeled by six kinds of components $\text{COMP} = \{ConstantTasks,\ BurstTasks,\ PeriodicTasks,\ Scheduler,\ Slack,\ PM\}$. Every component is composed of several processes which can perform different actions according to their private attributes. In the figure, if there is more than one concurrent process in a component, the process names are given after the component name. Furthermore, the global queue is described by a list, named $task\_list$, in the $Scheduler$ component, and the local queue in a PM is divided into two lists: $capacity\_PM$ in $Scheduler$ and $stack$ in PMs. The former is convenient for centralized scheduling and the latter enables distributed implementation. To support distributed implementation, PMs have distinct identities based on their location which is defined as $\overrightarrow{loc} = \langle type, pos\rangle$, where $type$ denotes *Hot*, *Warm*, *Cold* and *Reserved*, and pos is the position in the pool. The PMs operate in parallel and their activities are determined by locations and attributes, which are critical for the analysis of availability when PMs have different features.

Note that the scheduling module can adopt a hierarchical approach when there are too many tasks to handle. Due to space constraints for the article, this will not be discussed. In the following subsections, we will introduce each sub-model represented in CARMA and as Markov Chains.

3.1 Modeling super-task arrivals

Prediction of cloud workloads has been considered in a series of papers [38] [39] [40]. The results show that workload variation follows certain rules and the task arrival pattern has some basic types. In this paper, we will follow Bruneo's classification and describe all other types by a combination of the three basic task arrival patterns: constant tasks, periodic tasks and burst tasks [32]. As stated in Section 1, it is difficult for existing models to describe a number of different workloads with different start times, durations and execution rates. Thus an attribute-based method is adopted in this subsection.

3.1.1 Constant arrival

Constant arrival is the simplest task type which executes with almost the same rate throughout the model execution. In this paper the start time, stopping time and task size are thought to be attributes of the component representing the arrival stream, $\text{ATTR} = \{\ t_{start},\ t_{end},\ size\ \}$. Thus we can create different constant arrival streams through the use of attributes. If both $t_{start} > 0$ and $t_{end} < endpoint$ are true, the task will be treated as a special case of burst task. The corresponding Markov chains for constant arrivals are shown in Fig. 4, where the middle one represents a non-zero start time and the bottom one captures an early finish, and *Initial* and *Stop* are two temporary states. All the cooperation (shared) actions in the model are highlighted in red.

Generally, a constant arrival can be modeled by the parameterized process:

$$\texttt{C} \quad = \quad [t_{start} \leqslant \mathsf{now} \leqslant t_{end}].c\_arrival[\top]\langle size\rangle.\texttt{C};$$

where the term now represents the current time. Then by assigning different values to those attributes, different constant arrivals are created using only one type of component.

3.1.2 Burst arrival

Burst arrival refers to a task that starts abruptly and finishes after a short interval, with a high throughput. All burst arrivals can be depicted by similar attributes, $\text{ATTR} = \{\ t_{start},\ \omega,\ size\ \}$, where ω denotes the duration of the burst task. The corresponding Markov chains are given in Fig. 5.

The burst arrival can be modeled by two processes in CARMA as follows.

$$\begin{aligned}\texttt{Timer} \quad &= \quad [\mathsf{now} < t_{start}].timer^{\star}[\bot]\langle\rangle.\texttt{Timer} \\ &+ \quad [\mathsf{now} \geqslant t_{start}].bell^{\star}[\bot]\langle\rangle.\texttt{B};\end{aligned}$$

$$\begin{aligned}\texttt{B} \quad &= \quad [\omega < N_{duration}].b\_arrival[\top]\langle size\rangle\sigma\,\{\omega'\}\,.\texttt{B} \\ &+ \quad [\omega \geqslant N_{duration}].no\_arrival^{\star}[\bot]\langle\rangle.\mathbf{nil};\end{aligned}$$

where $N_{duration}$ indicates the threshold of ω, and ω' is the updated value of ω.

3.1.3 Periodic arrival

A classic periodic arrival repeats its size in regular intervals or periods. This self-similar phenomenon has been observed in many clouds. The job arrival rate depends on the number of users, weekly cycle, seasonal factors and so on [41]. In terms of [32], the job arrival process has been modeled as a Markov Modulated Poisson Process (MMPP). MMPP is easy to describe by Markov Chains which are used as the underlying solution tool both in reference [32] and in our work, so we follow that approach in this paper. In further study, we will improve our model and tool to support simulating a periodic workload with Non-Homogeneous Poisson Processes which will be more accurate than the current MMPP assumption.

The left part of Fig. 6 is a simple example of periodic arrival, $\text{ATTR} = \{L, size, P_1, \ldots, P_i\_to\_P_{i+1},$

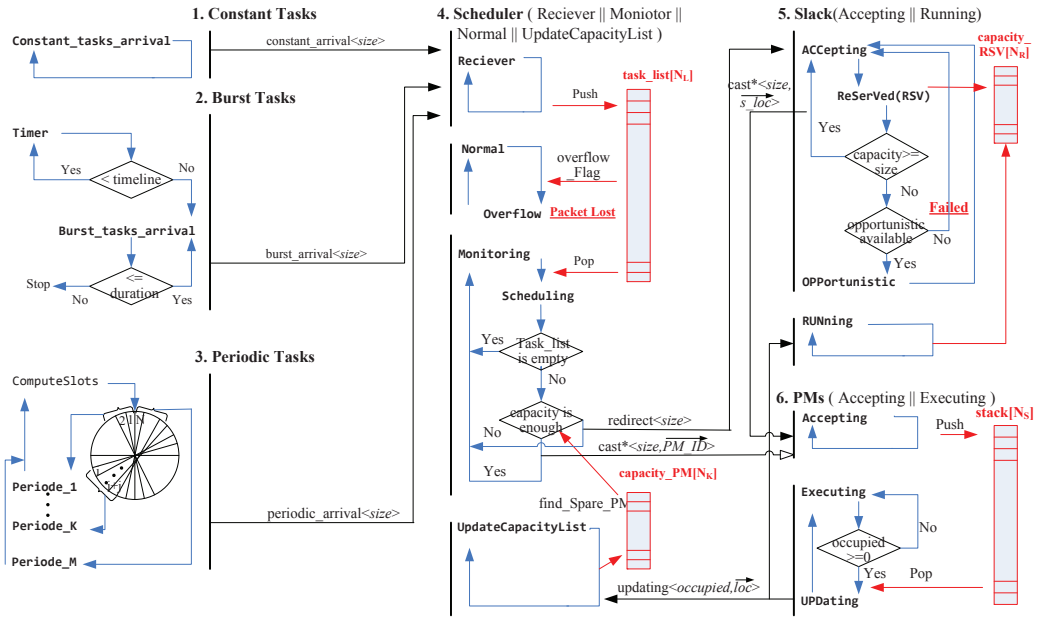

Fig. 3. The process of communication between components.

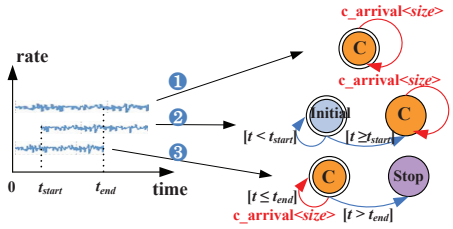

Fig. 4. The Markov chains of constant arrival.

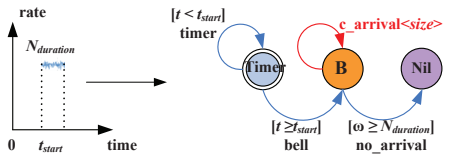

Fig. 5. The Markov chains of burst arrival.

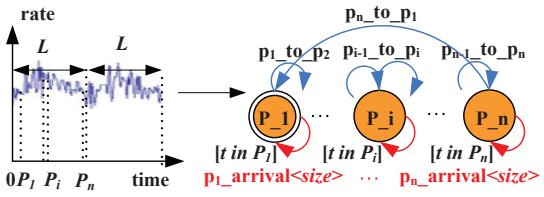

Fig. 6. The Markov chains of periodic arrival.

$\ldots, P_n, P_n\_to\_P_1\}$, $1 \leqslant i \leqslant n$, where the length of circle L and the division of periods is determined by the component's attributes. The corresponding Markov chain, MMPP(λ_{p_1}, $\lambda_{p_1\_to\_p_2}$,$\ldots$,λ_{p_n},$\lambda_{p_n\_to\_p_1}$), is given in Fig. 6. Denoting $T = (\text{now} \mod L)$, the periodic arrival can be modeled by processes with CARMA as follows.

$$
\begin{aligned}
\texttt{P\_1} &= [T \in P_1].p_1\_arrival[\top]\langle size\rangle.\texttt{P\_1} \\
&+ [T \in P_1\_to\_P_2].p_1\_to\_p_2^{\star}[\bot]\langle\rangle.\texttt{P\_2}; \\
&\ldots \\
\texttt{P\_i} &= [T \in P_i].p_i\_arrival[\top]\langle size\rangle.\texttt{P\_i} \\
&+ [T \in P_i\_to\_P_{i+1}].p_i\_to\_p_{i+1}^{\star}[\bot]\langle\rangle.\texttt{P\_i+1}; \\
&\ldots \\
\texttt{P\_n} &= [T \in P_n].p_n\_arrival[\top]\langle size\rangle.\texttt{P\_n} \\
&+ [T \in P_n\_to\_P_1].p_n\_to\_p_1^{\star}[\bot]\langle\rangle.\texttt{P\_1};
\end{aligned}
$$

3.1.4 Task size

The most important attribute of a super-task, its task size, can be a vector. For example, if a task needs x units CPU, y units memory and z units bandwidth, $\overrightarrow{size} = \langle x, y, z\rangle$. For brevity in this paper, we only consider it as a scalar quantity, the number of VMs needed, while it is technically straightforward to extend it to be a vector. Moreno *et al.* [38] have found the workload in task clusters to follow a log-normal distribution that in a small interval can be approximated by a normal distribution, but the distribution of task size has not been reported. In this paper, the normal distribution is chosen as the default setting of task size, and the results will be compared with that of a fixed integer value, a log-normal distribution and a uniform distribution in experiments. Notably, it is easy to change this assumption for any other distribution because of the use of attributes.

3.2 Modeling service provisioning

In this subsection, we focus on how to find a suitable PM to satisfy the VM deployment requirements when there are super-tasks. Consequently, PM provisioning, that plays a crucial role for service provisioning, will be modeled as well as slack resource provisioning and PM execution. VM deployment and queuing are not studied in detail in our model since they have been studied extensively in previous works [5] [11] [8].

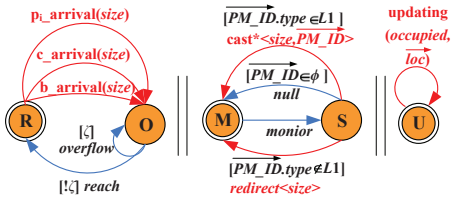

Fig. 7. The Markov chains of pool management schemes.

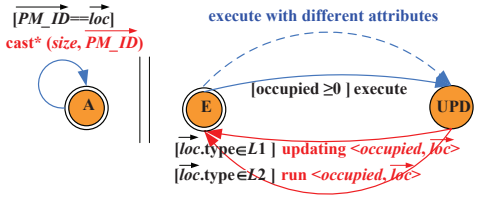

Fig. 8. The Markov chain corresponding to a PM.

3.2.1 Modeling PM scheduling

The component *Scheduler* is used to represent the PM provisioning schemes consisting of three parallel parts as shown in Fig. 7. The left Markov chain captures the function of the global queue. The state R receives the messages from different tasks, pushes them into a queue $task\_list$ and determines whether to set an overflow flag ζ. The phenomenon of overflow is indicated by state O, and the transitions between O and R are controlled by ζ.

As shown in the middle part, PMs are scheduled to provide available PMs. Firstly, M is the state of monitoring which explores spare PMs in the capacity list hosted in the data center. Meanwhile the state S captures a scheduling operation in the component. In the scheduling process, a hot PM is always the preferred option to reduce cost; then warm and cold PMs are chosen in turn incurring longer preparation time when no hot PMs are available. Next, a message with both the task size and the selected PM's identification is sent to all PMs in a broadcast manner. However, if there is no available PM after retrieving the list $capacity\_PM$, a message of task size is redirected to the component $Slack$ in order to search for slack resources. The role of process U is to receive messages from each PM and update the capacity list. Note that broadcast is a highly abstract operation that ignores the specific details of the scheduling process in different cloud computing systems. It is nevertheless useful in our model.

Letting the attribute set $\text{ATTR} = \{task\_list, capacity\_PM, size, \zeta\}$ and $L1 = \{Hot, Warm, Cold\}$ be the set of normal resources, the *Scheduler* can be described by CARMA processes as follows.

$$
\begin{aligned}
\mathtt{R} &= c\_arrival[\top](size)\sigma_1.\mathtt{O};\\
&+ b\_arrival[\top](size)\sigma_1.\mathtt{O};\\
&+ p_1\_arrival[\top](size)\sigma_1.\mathtt{O}; \ + \ \cdots\\
&+ p_i\_arrival[\top](size)\sigma_1.\mathtt{O}; \ + \ \cdots\\
&+ p_n\_arrival[\top](size)\sigma_1.\mathtt{O};\\
\mathtt{O} &= [\zeta].overflow^{\star}[\bot]\langle\rangle.\mathtt{O} + [\neg\zeta].reach^{\star}[\bot]\langle\rangle.\mathtt{R};\\
\mathtt{M} &= monitor^{\star}[\bot]\langle\rangle\sigma\{size := pop\,(task\_list)\,;\\
&\quad \overrightarrow{PM\_ID} := findPM\,(capacity\_PM, size)\,;\}.\mathtt{S};\\
\mathtt{S} &= [\overrightarrow{PM\_ID} \in \emptyset].null^{\star}[\bot]\langle\rangle.\mathtt{M}\\
&\quad \%\texttt{The queue is empty}\\
&+ [\overrightarrow{PM\_ID}.type \in L1].cast^{\star}[\top]\langle size, \overrightarrow{PM\_ID}\rangle\sigma\{\\
&\quad update\,(capacity\_PM)\,;\}.\mathtt{M}\\
&+ [\overrightarrow{PM\_ID}.type \notin L1].redirect[\top]\langle size\rangle.\mathtt{M};\\
\mathtt{U} &= updating[\top](occupied, \overrightarrow{loc})\sigma\{\\
&\quad releasePM\,(capacity\_PM, occupied, \overrightarrow{loc})\,;\}.\mathtt{U};
\end{aligned}
$$

$$Scheduler \triangleq R \parallel M \parallel S \parallel U;$$

where the function $findPM()$ is used to find a PM that has remaining capacity bigger than the size; $update()$ is a function that updates the $capacity\_PM$ when $releasePM\ ()$ releases the space occupied by the PM that has completed the task; σ_1 is responsible for pushing a new job and updating the overflow flag and includes two operations: $push\,(task\_list)$ and $\zeta\,(task\_list)$.

3.2.2 Modeling attribute-based PMs

In previous work, [15] [9] [11], the expected departure time of a job is found by solving a Markov chain. In our model instead, an attribute-based PM autonomously decides whether it can accept a new job with its remaining capacity based on information received via the broadcast mechanism. The location attribute provides a foundation for heterogeneity of the PMs with different behaviors. Hence, the completion time of a super-task is truly determined by the PMs rather than calculating a probability. This is especially important when numerous different PM types are involved, because the Markov chains would be too complicated to solve efficiently.

For PMs, the features such as capability, execution rate, or even PM ageing can be easily described by their attributes, $\text{ATTR} = \{stack, size, \overrightarrow{loc}, occupied\}$. To model different capacity, it is only necessary to modify the capacity list $capacity\_PM$ owned by component *Scheduler* and the corresponding $stack$ in the PM. As seen in Fig. 8, the messages received in state A will be filtered firstly by the PM's ID which is typically its location. An accepted task will be pushed on the stack $stack$ and picked up to be deployed in the state E. While transitioning from E to UPD (the state of updating), the PM may implement some different actions determined by its private attributes. Finally, in order to update the state of PMs, the number of VMs occupied in each PM and the PM's ID are sent by the action *update* or *run* due to its type.

Letting $L1$ be the set of normal resources as previously and $L2 = \{Reserved\}$ be the set of slack resources, a PM component can then be described by CARMA processes as follows.

$$
\begin{aligned}
\mathtt{A} &= cast^{\star}[\overrightarrow{PM\_ID} == \overrightarrow{loc}](size, \overrightarrow{PM\_ID})\sigma\,\{\\
&\quad push\,(stack)\,;\ \mathsf{my}.occupied := occupied + size;\\
&\quad timeout\_timer := \mathsf{now};\,\}.\mathtt{A}; \quad \%\texttt{for measure}\\
\mathtt{E} &= [occupied \geq 0].execute^{\star}[\bot]\langle\rangle\sigma\,\{\\
&\quad t\_size := pop\,(stack)\,;\\
&\quad \mathsf{my}.occupied := (timeout\_timer - \mathsf{now} > T_0)?\\
&\quad (occupied - t\_size) : 0;\,\}.\mathtt{UPD} \ \%\texttt{for measure}\\
\mathtt{UPD} &= [\overrightarrow{loc}.type \in L1].updating[\top]\langle\overrightarrow{loc}, occupied\rangle.\mathtt{E}\\
&+ [\overrightarrow{loc}.type \in L2].run[\top]\langle\overrightarrow{loc}, occupied\rangle.\mathtt{E};
\end{aligned}
$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2019.2943339, IEEE Transactions on Parallel and Distributed Systems

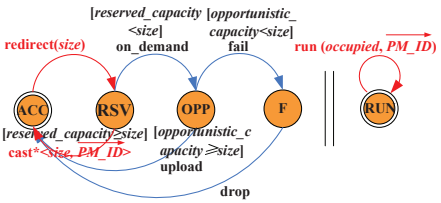

Fig. 9. The Markov chains of reserved source.

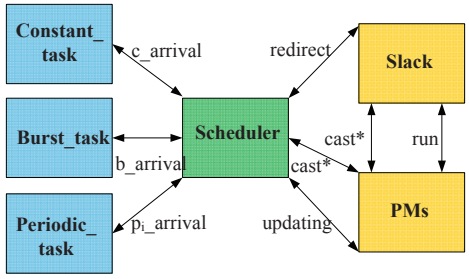

Fig. 10. The cooperation graph of CARMA sub-models.

$$\texttt{PMs} \triangleq A \parallel E;$$

where *occupied* denotes the number of VMs deployed at the current time, and T_0 is a threshold to avoid a false value of *occupied* caused by deadlock when no tasks arrive, such as after a burst task.

Furthermore, since the execution mode of processes can be defined in terms of attributes, the PMs are able to change the implementation branches or execution speed dynamically in accordance with the results of the feedback from the environment $\mathcal{E}$. Thus it is straightforward to depict PMs with more complex features such as PM ageing (as discussed in Section 2.3).

For the sake of simplicity, we adopt a strategy of incorporating a start-up cost when computing the execution rate of warm and cold PMs, i.e. the time to prepare the start-up state and the real execution time are treated as a single time interval $T_{implement}$. The execution rate is the reciprocal of $T_{implement}$ and satisfies $r_{cold} < r_{warm} < r_{hot}$.

3.2.3 Modeling of slack resources

For clouds, some slack resources are usually reserved to cope with burst arrivals, and these can also be resold to other providers opportunistically to increase the availability of the federation. This process is modeled by the Markov chain in Fig. 9.

The state RUN in the right part manages the reserved resources in the list $capacity\_RSV$. The meaning of the other processes is : 1) ACC is responsible for receiving messages from the $Scheduler$. 2) RSV expresses the process of exploring a spare PM in the slack resources. 3) OPP refers to the process of finding an opportunistic PM in the federation cloud. 4) F is a failure state reached when there is no available PM.

The sources of failures in large cloud systems are diverse, including hardware failures, PM failures, OS errors, link interruptions, etc. These details are difficult to fully consider, and an abstract model is generally used. For example, in [6] it is considered that a VM is unavailable and needs to migrate whenever a failure occurs. In this paper failure is similarly treated in an abstract way capturing all cases where computing resources are not accessible.

The attribute set of component $Slack$ is $\text{ATTR} = \{capacity\_RSV, \overrightarrow{s\_loc}, size\}$, and the processes in $Slack$ can be described in CARMA as follows.

$$\begin{aligned}
\texttt{ACC} &= redirect[\top](size)\sigma\{ \\
&\quad \overrightarrow{s_loc} := findPM(capacity_RSV, size);\}.\texttt{RSV}; \\
\texttt{RSV} &= [\overrightarrow{s_loc}.pos \notin \emptyset]cast^{\star}[\top]\langle\overrightarrow{s_loc}, size\rangle.\sigma\{ \\
&\quad refresh(capacity_RSV, \overrightarrow{s_loc}, size);\}.\texttt{ACC} \\
&+ [\overrightarrow{s_loc}.pos \in \emptyset].on_demand^{\star}[\bot]\langle\rangle.\texttt{OPP}; \\
\texttt{OPP} &= upload^{\star}[\bot]\langle\rangle.\texttt{ACC} + fail^{\star}[\bot]\langle\rangle.\texttt{F}; \\
\texttt{F} &= drop^{\star}[\bot]\langle\rangle.\texttt{ACC}; \\
\texttt{RUN} &= run[\top](occupied, \overrightarrow{PM_ID})\sigma\{ \\
&\quad release(capacity_RSV, occupied, \overrightarrow{PM_ID});\}.\texttt{RUN};
\end{aligned}$$

$$\texttt{Slack} \triangleq ACC \parallel RUN;$$

where the function $refresh()$ refreshes the state of $capacity\_RSV$ after deploying a task and $release()$ recycles the space occupied by the PM that has completed the task.

3.3 The cooperation model

In the above discussions, we have established 5 sub-models, and the relationship between them can be described in a cooperation model as in Fig. 10.

The differences between individual tasks and PMs in a real IaaS cloud mean that the underlying Markov chain may be very complex. But with a CARMA model of cooperating components we are able to define the model in a clear and simple manner using the inherited attributes.

4 Simulation and Analysis

In this section, we will analyze the availability of an IaaS system using the model built in the last section. The set of metrics used is briefly discussed in Section 4.1 and in Section 4.2 we introduce the parameters used in the following experiments. The impacts of the task size and PM attributes are analyzed in Section 4.3. Finally, the assignment policies for slack and normal resources are investigated in order to achieve a higher level of availability.

4.1 Metrics

Rather than the probability of at least one available VM, we define three groups of metrics to assess the impact of task size on availability. The first is the utilization level of resources measuring how efficiently resources are used. The second is the probability of using slack resources. The last group focuses on the states representing scenarios of unavailability, in order to evaluate the availability of the whole cloud.

4.1.1 The utilization level of VMs

For super-tasks, not all VMs are fully utilized since the remaining capacity may not be large enough for new arrivals. Generally, if more VMs can be deployed on each PM, the probability of not finding an available VM will be less. Therefore, the average number of VMs deployed in a PM is a major factor in assessing the level of utilization.

Assuming $N_i(t)$ is the number of VM deployed in the ith PM at time point t and $N_{i,max}$ is the capacity of PM_i, the average utilization rate of VMs for PM_i can be defined by $N_i(t)/N_{i,max}$. Thus for a PM pool, i.e. *hot*, *warm*, *cold* and *reserved*, the average utilization rate for VMs is

$$\eta_{pool}(t) = \frac{\sum_{i=1}^{N_{pool}} N_i(t)}{\sum_{i=1}^{N_{pool}} N_{i,max}}, \tag{1}$$

where N_{pool} is the number of PMs in a certain pool, and $pool \in L1 \cup L2$.

Similar to Eq(1), the utilization rate for VMs in all normal PMs, $\eta_{normal}(t)$, refers to the average utilization level of normal VMs in the whole system. A higher value of $\eta_{normal}(t)$ means a lower probability of using the slack resources.

$$\eta_{normal}(t) = \frac{\sum_p \sum_{i=1}^{N_p} N_i(t)}{\sum_p \sum_{i=1}^{N_p} N_{i,max}},\ p \in L1, \tag{2}$$

Based on the above discussions, the maximum average utilization rate of a VM for a cloud can be computed by

$$max_\eta = Max_{t \in T_1} \{\eta_{normal}(t)\}, \tag{3}$$

where T_1 is the duration of the experiment, which must be long enough for $\eta_{normal}(t)$ to reach a stable value.

4.1.2 The proportion of slack allocation

When there are no available VMs in the hot/warm/cold pools, the slack resources are called. Hence the ratio of calling the slack can also be used to assess the availability level of an IaaS. The metric P_S is defined as the proportion of tasks redirected to the reserved resources, and P_O is the proportion of tasks redirected to the opportunistic resources. Due to the phenomenon of overflow, some components are not scheduled at all, which will not be included in our formula. Let N_{total} be the total number of tasks, $N_{overflow}$ be the number of task rejected, N_{RSV} and N_{OPP} are the number of tasks redirected to reserved and opportunistic resources respectively. Then P_S and P_O are expressed as

$$P_S = \frac{N_{RSV}}{N_{total} - N_{overflow}}, \tag{4}$$

$$P_O = \frac{N_{OPP}}{N_{total} - N_{overflow}}. \tag{5}$$

4.1.3 Availability

Availability is a key metric for evaluating a cloud's ability to meet customer needs. In our model, the proportion of tasks accepted and deployed successfully in VMs is used to express availability. Assuming N_{fail} is the number of tasks which fail in the *Slack* component, then by similar arguments to eq(4), the rate of failure R_{fail} is

$$R_{fail} = \frac{N_{fail}}{N_{total} - N_{overflow}}. \tag{6}$$

Finally, the rate of unavailability is defined as

$$R_{unavail} = \frac{N_{overflow} + N_{fail}}{N_{total}}. \tag{7}$$

4.2 Experimental environment setting

For brevity, a simple case is chosen as the default setting in the following experiments. Firstly, the number of PMs in each kind of pool is set to 5 to keep the plots a reasonable size and show the growth trend of the utilization level of VMs. In fact, the number of hot, warm and cold PMs can be easily expanded to 1000 or more in our experiments.

Next, the action rates are estimated using the classical method proposed by Huang *et al.* [42]. Given the average sojourn time of an action α is T, then the rate of α is $1/E(T)$, where $E(T)$ is mathematical expectation of T. Since the unit of time in this paper is a neutral one — "time units", the unit of action rate is "tasks per time unit". For the sake of simplicity, we will omit these units in the following discussion. The rate of execution $r_{execution}$ of hot, warm and cold PM is set to 4, 2 and 1 respectively. The periodic arrival here has three periods, with lengths 7, 13 and 10, but this can also be expanded by changing only some parameters. The default periodic task arrival rates in these 3 periods are respectively 10, 5 and 1, and $r_{P1\_to\_P2}$, $r_{P2\_to\_P3}$ and $r_{P3\_to\_P1}$ are 7, 3 and 7. In addition, the probability of finding opportunistic resources or failing is 95% and 5%, and the rate of failure is 5.

Thirdly, in the following experiments, task size following three kinds of distribution will be considered, including a uniform distribution, a truncated normal distribution and a log-normal distribution. The maximum capacity of PM in this paper is 10 by default. If the task size follows a uniform distribution $U(0, max\_task\_size)$, the value of max_task_size can also be set to 10. Moreover, if X is a random variable following the normal distribution $N(0, \sigma)$, the task size is set to be an integer $Y = \lfloor |X| \rfloor + 1$, and we denote the new truncated normal distribution as $TN(0, \sigma)$. For comparison, when the task size follows a truncated normal distribution and log-normal distribution, most of the task sizes should be less than max_task_size. Due to the properties of the normal distribution, more than 99.999% stay within $[\mu - 4.26\sigma, \mu + 4.26\sigma]$, thus this meets our needs when $\sigma = 2$. Correspondingly, we can choose the log-normal distribution $lognormal(0, 1)$ with a variable Z and the task size is set to be an integer $\lfloor Z \rfloor + 1$. Then the probability distributions for the task size following the normal and log-normal distributions are compared in Fig. 11. The task size following the log-normal distribution possesses a heavy tail, which means that there are more tasks with a size greater than 10. Additionally, the task size can also be set to a fixed integer value, as explained in Section 3.1.

Other critical parameters have the values as given in Table 2. All other action rates in the model are set to 100 by default. Moreover, the default strategy of pool management is best fit and we do not consider the ageing mechanism of PMs initially. By default, in simulation experiments, the simulation time is 30, the number of replications is 100 and the number of samplings is 50. For all the CARMA models experimental settings and codes can be found at the website https://codeocean.com/capsule/7497086/tree.

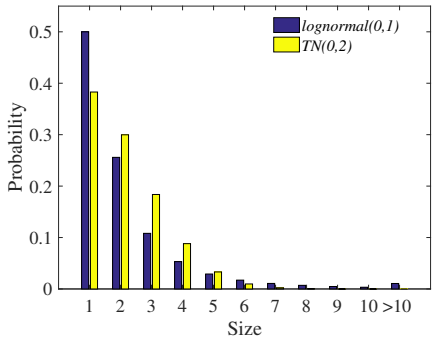

Fig. 11. The probability distribution of each task size while following the truncated normal and log-normal distribution.

TABLE 2
The default model parameter values.

Symbol	Description	Default value
L_g	Length of global queue	100
r_{burst}	Rate of burst task arrival	30
$r_{constant}$	Rate of constant task arrival	5
$r_{monitor}$	Rate of monitoring in Scheduler	500
r_{cast}	Rate of unicasting in Scheduler	500
$r_{updating}$	Rate of updating capacity list in Scheduler	500
$r_{overflow}$	Rate of *overflow*	500
r_{reach}	Rate of *reach*	500
r_{upload}	Rate of execution of opportunistic resources	10
r_{fail}	Rate of failure to deploy in federation clouds	5
t_{start}	Start time of burst arrivals	5
$N_{duration}$	Number of jobs in a burst arrival	200
t_{begin}	Start time of a constant arrival	0
t_{end}	End time of a constant arrival	$+\infty$

Moreover, we highlight that, based on the formal model, we can analyze different scenarios by changing the parameters.

In the following sections, the CARMA tool[2] developed by the University of Edinburgh and the University of Florence is used to model and analyze the case study.

4.3 Analysis of the impact of attributes values

In this subsection, different task arrival patterns are firstly validated with respect to various important attributes. Next, the most critical attribute, the task size is analyzed. Finally, we discuss the impacts of PM's attributes on availability.

4.3.1 Analyzing the impacts of tasks arrival patterns

Taking the case in Section 4.2 as an example, we will study the impact of task arrival patterns on the number of virtual machines deployed. In the simplest scenario where there is only a burst task and its size is fixed at 1, the numbers of hot VMs deployed is shown in Fig. 12 (a). To aid explanation, we will denote PMs in the hot pool as Hot_i, $i = 1, 2, \ldots, 5$. It is found that the PMs are chosen from Hot_1 to Hot_5 in turn with a best fit strategy when a task arrives, and the utilization level decreases from Hot_1 to Hot_5 as expected. When $N_{duration}$ is changed to 100, the results are as given in Fig. 12 (b), and naturally fewer VMs are needed when fewer tasks arrive. Similarly, the results for constant arrivals are shown in Fig. 12 (c), and those for periodic arrivals with different periods in Fig. 12 (d). Finally, mixing the three kinds of tasks, we can get a more complex workload. Assuming there is a burst arrival, a periodic arrival and a constant arrival set as in Table 2, the results when all task sizes are fixed to 1 or following a normal distribution, are separately shown in Figs. 12 (e) and (f). For the latter, the utilization rate stays at a lower level.

The results in Fig. 12 give confidence that the model is behaving as anticipated, whilst the attributed-based task arrival submodels decrease the complexity of modeling different workloads.

4.3.2 Impacts of task size

In addition to the task arrival patterns, the task size is another important feature of workflow variation. In the following experiments, we investigate the impact of the distribution of task size, comparing a uniform distribution, a truncated normal distribution, a log-normal distribution and that of fixed integer values. The results are shown in Fig. 13.

Fig. 13 reveals that the utilization rate of VMs differs greatly when the task size differs. Moreover when the task size is more than 6, a large number of VMs will be unused since the residual space is not enough for a new task. Similarly, when the task size follows a uniform distribution (Fig. 13 (j)), a truncated normal distribution (Fig. 13 (k)) or a log-normal distribution (Fig. 13 (l)), not all of VMs can be placed in hot PMs because there is not sufficient capacity to deploy a new arrival. Furthermore, compared to the results of uniform distribution, there are more VMs placed in Fig. 13 (k) owing to some smaller size tasks arriving, and similar results can been observed in Fig. 13 (l).

Based on the above discussion, the maximum average utilization rate max_η with different task size is compared in Fig. 14 (a). It shows that if the task size is a factor of 10, the maximum capacity, max_η is more likely to reach a higher level. Furthermore, to investigate the influences from the distribution function and maximum capacity, we change the parameters of the log-normal distribution σ and capacity, and the results are separately shown in Fig. 14 (b) and (c). From Fig. 14 (b), P_O seems to increase with the growth of σ, and it is more obvious for log-normal distribution that almost 40% of tasks require the opportunistic resources when task size follows $lognormal(0, 10)$. As stated in Section 4.1.2, the main reason is that the size becomes larger when σ grows. For $lognormal(0, 10)$, more than 40% of size values fall in $(10, +\infty)$, resulting in a lot of capacity being wasted. It is the same for the case of a truncated normal distribution. In Fig. 14 (c), the task size follows $lognormal(0, 1)$. While the maximum capacity of PM increases from 10 to 50, the value of max_η and the maximum number of VMs running is compared. It is apparent that max_η stays at a high level in all scenarios and increases with the growth of the maximum capacity. It implies that more resources will be fully used when PMs have a larger capacity and less residual resource

2. http://quanticol.github.io/CARMA/

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2019.2943339, IEEE Transactions on Parallel and Distributed Systems

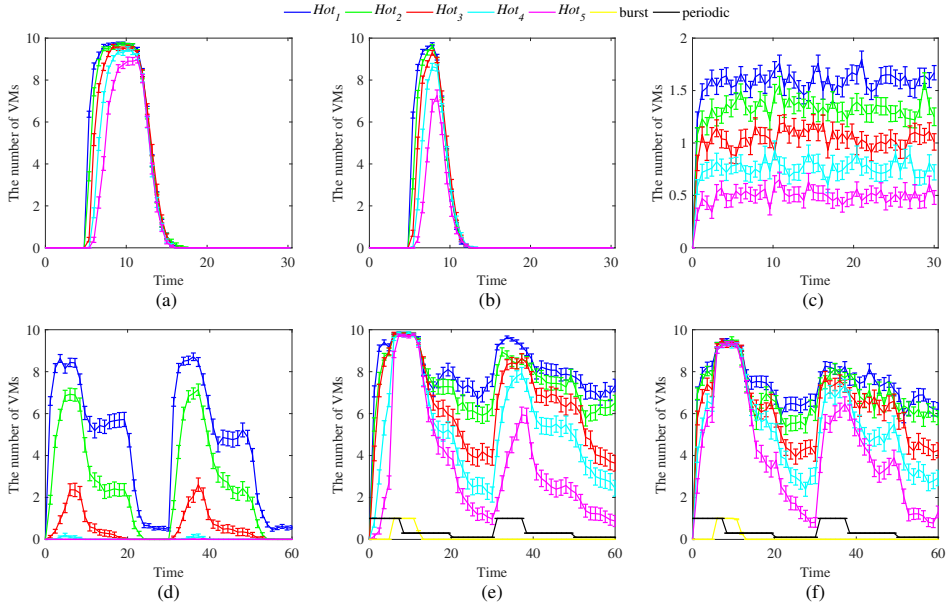

Fig. 12. The average utilization level of VMs for a certain PM confronted with different kinds of tasks, including (a) a burst arrival with $\omega = 100$, (b) a burst arrival with $\omega = 200$, (c) a constant arrival, (d) a periodic arrival with a non-uniform division of cycle, (e) a mixture task and (f) a mixture task whose size follows truncated normal distribution $TN\,(0, 2)$.

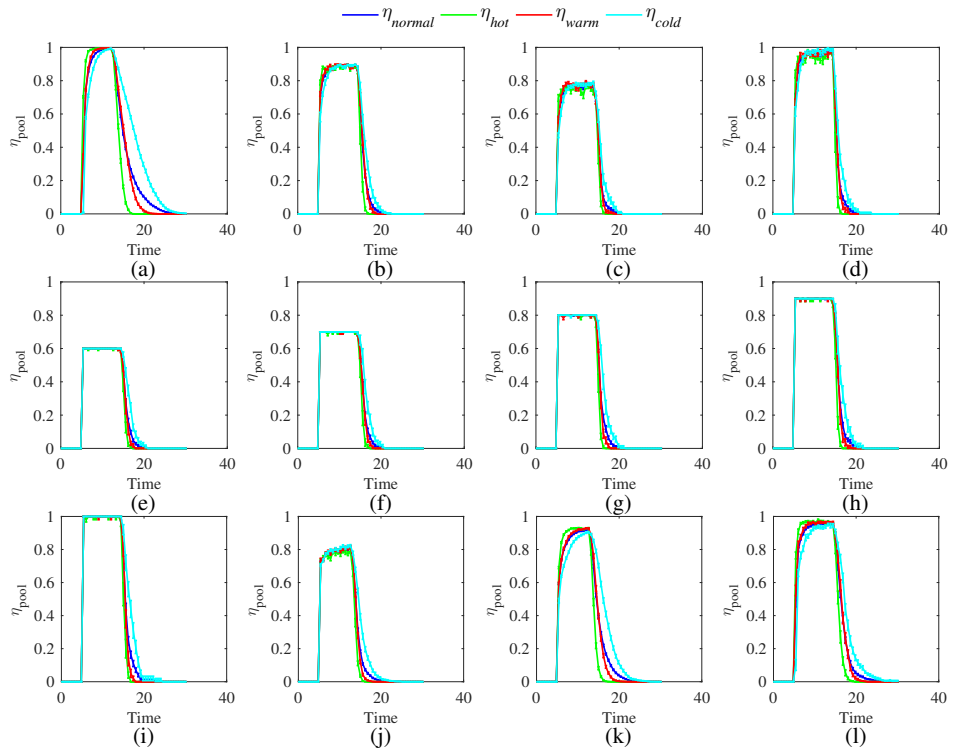

Fig. 13. The impact of different task size while (a) $size = 2$, (b) $size = 3$, (c) $size = 4$, (d) $size = 5$, (e) $size = 6$, (f) $size = 7$, (g) $size = 8$, (h) $size = 9$, (i) $size = 10$, (j) $size$ following the uniform distribution $u(1, 10)$, (k) the truncated normal distribution $TN\,(0, 2)$ and (l) the log-normal distribution $lognormal\,(0, 1)$.

will be wasted. From the above discussions, a higher utilization level of PMs is achievable only if the task size is a factor of the maximum capacity.

4.3.3 Analyzing the impacts of PM attributes

Attributes play a crucial role in capturing the differences between PMs, decreasing the complexity of building tailored models for each kind of PM. In this subsection, three simple experiments are made investigating the impacts of different

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2019.2943339, IEEE Transactions on Parallel and Distributed Systems

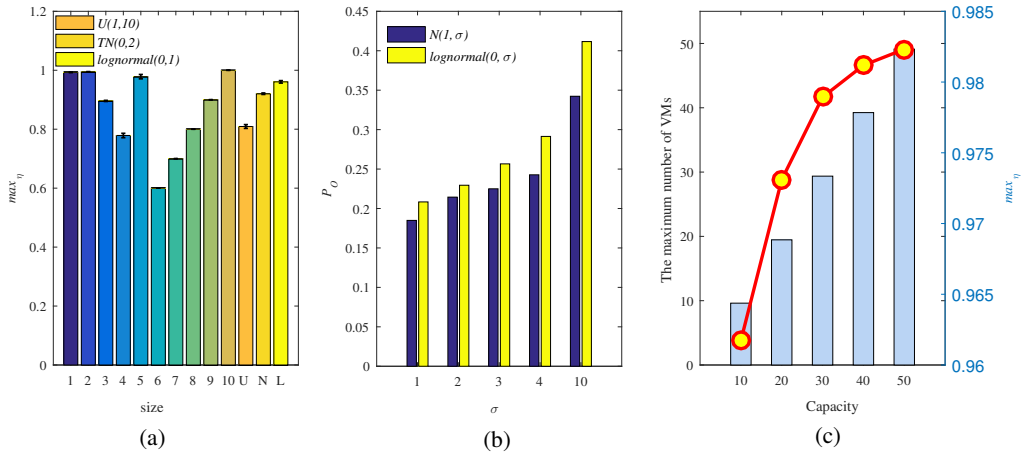

Fig. 14. The impact of distribution function and maximum capacity (a) when the task size is set to different values, (b) when the task size follows different distributions, (c) when changing the maximum capacity.

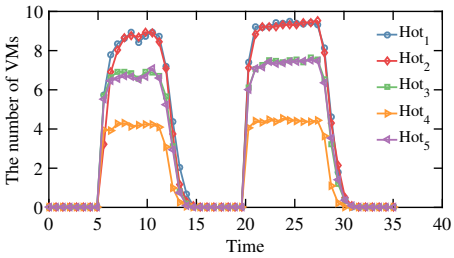

Fig. 15. The impacts of different capacity.

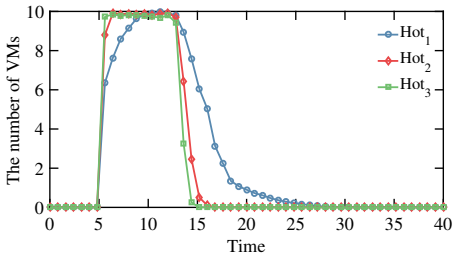

Fig. 16. The utilization of VMs while PMs have different execution rates.

capacity, execution rate and PM ageing.

Firstly, assuming there are five hot PMs and the vector of their capacity is (10, 10, 8, 5, 8), and a light workload with a burst task arrives at timepoint 5 and a heavy workload with 5 burst tasks arrives at timepoint 20, the impact of the PM's capability is shown in Fig. 15. For simplicity, all the task sizes are fixed to 1. From the figure, it can be seen that the last 3 PMs are chosen first on the time axis for their smaller capacity because of a best fit strategy. Moreover, the maximum number of VMs used in each PMs is limited by their capacity, and finally they reached the upper bound capacity due to a heavy load.

Secondly, the rate of execution in a PM is also an essential factor in the utilization level of resources. Due to the use of attributes, every PM can have a distinct execution rate, for example, the execution rates of Hot_1, Hot_2, Hot_3 are separately 1.0, 5.0, 10. In Fig. 16, the utilization level of each VM is given when there is a heavy workload of burst task with fixed size 1. As expected the PMs with lower execution rate, such as 1.0, will need more time to complete their work. Because the rate of execution $r_{execute}$ is the reciprocal of average sojourn time λ, when $r_{execute}$ decreases, the execution time increases. Moreover, for the first two PMs, the peaks of the utilization level also appears much lower than the others. The reason is that when some tasks finish, the execution results cannot be sent in time to get a new task owing to the slower execution speed.

Lastly, we consider a more complex situation incorporating the phenomenon of PM ageing as described in Section 2.3. For the simplicity of this discussion, we assume that the relationship between $r_{execute}$ and the execution time t_e follows a simple function $r_{execute} = 4 * \exp(-0.001 * t_e)$, and assume that only the second PM is suffering ageing in a scenario with 10 constant arrivals. The results are shown in Fig. 17. As the rate of execution declines in the second PM, the number of tasks handled per unit time becomes smaller and more time is required to finish the task. Observe that the simple exponential relation can be easily replaced by a different function without affecting our modeling and analysis methods. This and the previous examples demonstrate how the use of attributes allow complex behaviors of PMs to be captured in a single parameterized submodel.

4.4 Analysis of the resource configuration scheme

In general, some extra resources will be reserved for use when there are not enough normal PMs spare to deal with a burst arrival or heavy loads. These are significant for an IaaS in order to ensure the SLA with the users can be satisfied. An optimal arrangement scheme of normal resources and slack resources is helpful to save expenses for the same level of availability.

First, let us consider the case that the number of each kind of normal PMs (Hot/Warm/Cold) is equal, which we denote as $N_{H/W/C}$. Then we analyze the impact of the resource configuration scheme while both the number of the slack PMs N_{slack} and $N_{H/W/C}$ increase from 1 to 10.

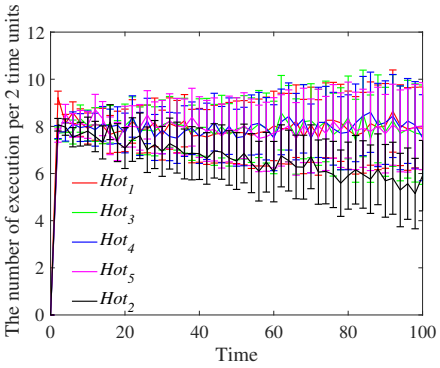

Fig. 17. The number of VMs used in each PM as the second hot PM ages.

When there are 10 constant task arrivals, the results are given in Fig. 18. The color bar is on a logarithmic scale, and the dark blue indicates a value of 10^{-6}. In Fig. 18 (a)∼(c), P_S, P_O and R_{fail} decrease with the growth of N_{slack} and $N_{H/W/C}$. Correspondingly, the ratio of unavailability is shown in Fig. 18 (d). From this figure, we can see that to have expected availability of 99.999%, one scheme needs 9 Hot/Warm/Cold PMs and 10 reserved PMs, and the other scheme needs 10 Hot/Warm/Cold PMs and 9 reserved PMs. Therefore, the former is optimal and less costly. For a complex scenario, there may be more schemes and we need to compare them via a utility function based on the cost of each type of PMs.

When the task size is not fixed but normally distributed, P_O and $R_{unavail}$ are compared in Fig. 18 (e) and (f). To achieve an availability of 99.999%, the number of PMs N_{slack} and $N_{H/W/C}$ are both 13. In other words, providers have to provide more PMs to deal with super-tasks. It is easy to find solutions that meet the given level of availability in the picture. For instance, if we choose $R_{unavail} = 0.001$, the alternative schemes are marked by red stars in Fig. 18(f). Then the most economical solution can be readily found at a given availability value.

Similarly, we can also change the ratio of hot, warm and cold PM to find an optimal arrangement scheme to gain a higher availability with a fixed number of PMs.

5 Conclusion

An attribute-based availability model for large scale IaaS has been built and described in this paper. Using attributes, both the workload components and the PMs can be modeled with different features such as start time, periodic cycle length, PM capability and even PM execution mode. This not only simplifies the process of modeling but also makes the formal model closer to actual clouds. Furthermore, due to the heterogeneity in the behavior of PMs, the PMs can determine job completion times individually. Based on that, the impact of task size was thoroughly investigated in our model, considering sizes not only following truncated normal distributions, uniform distributions, log-normal distributions, but also any fixed integer value. Experiments showed that more computing resources are needed to ensure availability when there are super-tasks, and that the task size following a uniform distribution is more severe than the case of a truncated normal one. We can also arrange PM deployment schemes to increase availability according the results of availability analysis. Additionally, we showed that CARMA offers an easier way to model IaaS clouds' complex phenomena than existing approaches.

In further work, we will study the impact of different scheduling strategies on availability when IaaS have super-tasks and the impact of different configuration schemes of PMs on revenue for providers.

Acknowledgments

The authors thank the reviewers for their valuable comments. The research of Hongwu Lv and Huiqiang Wang is supported in part by the National Natural Science Foundation of China (Nos. 61402127), the China Scholarship Council Fund (201706685020), the National Science and Technology Major Project of China (No. 2016ZX03001023-005) and Tianjin Key Laboratory of Advanced Networking (TANK), Tianjin University. The research of Paul Piho is supported by EPSRC grant EP/L01503X/1 (CDT in Pervasive Parallelism).

References

[1] B. Varghese and R. Buyya, "Next generation cloud computing: New trends and research directions," *Future Generation Computer Systems*, vol. 79, pp. 849–861, 2018.

[2] R. Ghosh, F. Longo, and K. S. Trivedi, "Performance analysis of large cloud," in *Large scale and big data: Processing and management*, S. Sakr and M. Gaber, Eds. Boca Roton: Auerbach Publications, 2014, ch. 18, pp. 557–578.

[3] B. Liu, X. Chang, Z. Han, K. Trivedi, and R. J. Rodríguez, "Model-based sensitivity analysis of iaas cloud availability," *Future Generation Computer Systems*, vol. 83, pp. 1–13, 2018.

[4] S. Sebastio, R. Ghosh, and T. Mukherjee, "An availability analysis approach for deployment configurations of containers," *IEEE Transactions on Services Computing*, 2018.

[5] H. Khazaei, J. Mišić, and V. B. Mišić, "A fine-grained performance model of cloud computing centers," *IEEE Transactions on parallel and distributed systems*, vol. 24, no. 11, pp. 2138–2147, 2013.

[6] H. Khazaei, J. Misic, and V. B. Misic, "Performance of an IaaS cloud with live migration of virtual machines," in *Global Communications Conference (GLOBECOM), 2013 IEEE*. IEEE, 2013, pp. 2289–2293.

[7] R. Ghosh, F. Longo, V. K. Naik, and K. S. Trivedi, "Modeling and performance analysis of large scale IaaS clouds," *Future Generation Computer Systems*, vol. 29, no. 5, pp. 1216–1234, 2013.

[8] F. Longo, R. Ghosh, V. K. Naik, A. J. Rindos, and K. S. Trivedi, "An approach for resiliency quantification of large scale systems," *ACM SIGMETRICS Performance Evaluation Review*, vol. 44, no. 4, pp. 37–48, 2017.

[9] X. Chang, R. Xia, J. K. Muppala, K. S. Trivedi, and J. Liu, "Effective modeling approach for IaaS data center performance analysis under heterogeneous workload," *IEEE Transactions on Cloud Computing*, vol. 6, no. 4, pp. 991–1003, 2018.

[10] D. S. Kim, F. Machida, and K. S. Trivedi, "Availability modeling and analysis of a virtualized system," in *2009 15th IEEE Pacific Rim International Symposium on Dependable Computing*. IEEE, 2009, pp. 365–371.

[11] H. Khazaei, J. Mišić, V. B. Mišić, and N. B. Mohammadi, "Availability analysis of cloud computing centers," in *Global Communications Conference (GLOBECOM), 2012 IEEE*. IEEE, 2012, pp. 1957–1962.

[12] H. Khazaei, J. Misic, and V. B. Misic, "Performance of cloud centers with high degree of virtualization under batch task arrivals," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 12, pp. 2429–2438, 2013.

[13] H. Khazaei, J. Mišić, V. B. Mišić, and S. Rashwand, "Analysis of a pool management scheme for cloud computing centers," *IEEE Transactions on parallel and distributed systems*, vol. 24, no. 5, pp. 849–861, 2013.

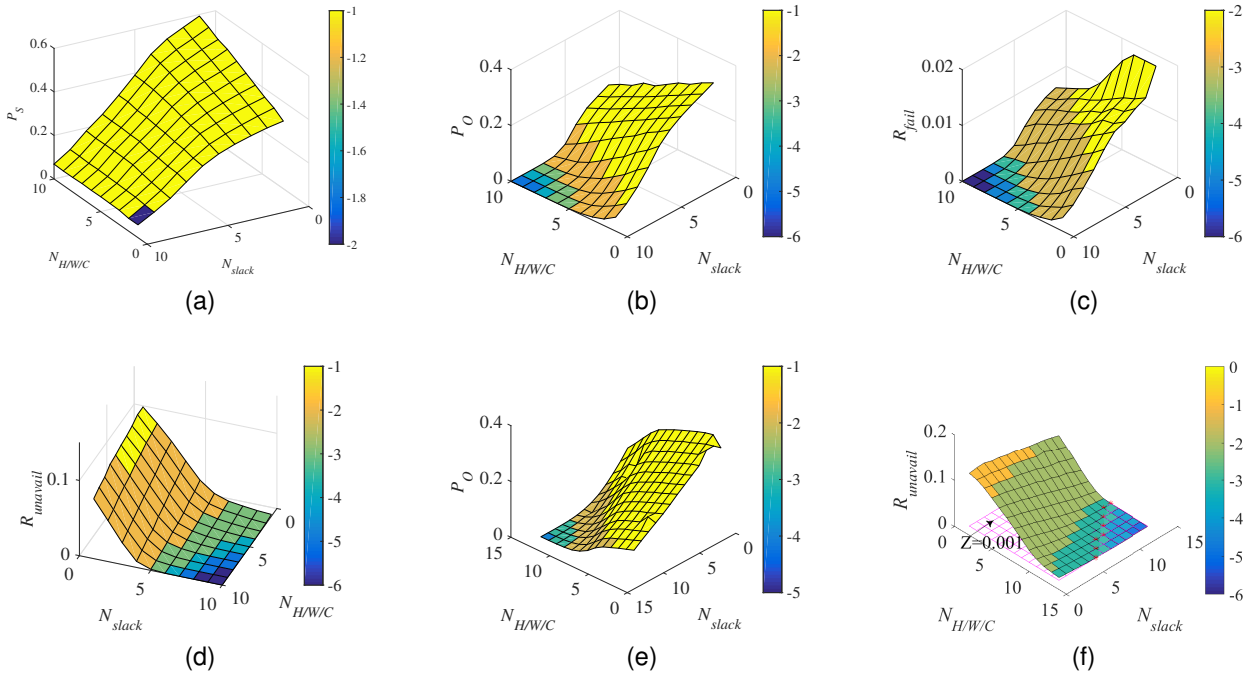

Fig. 18. When the task size is 1, the impact of different resource configuration schemes includes (a) the ratio of triggering slack resources, (b) the ratio of triggering opportunistic resources, (c) the proportion of failure, (d) the probability of unavailability. When the task size follows a truncated normal distribution $TN(0,2)$, (e) the ratio of triggering opportunistic resources and (f) the probability of unavailability.

[14] H. Khazaei, J. Misic, and V. B. Misic, "Performance analysis of cloud centers under burst arrivals and total rejection policy," in *Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE*. IEEE, 2011, pp. 1–6.

[15] B. Wang, X. Chang, and J. Liu, "Modeling heterogeneous virtual machines on IaaS data centers," *IEEE Communications Letters*, vol. 19, no. 4, pp. 537–540, 2015.

[16] D. Cotroneo, A. K. Iannillo, R. Natella, R. Pietrantuono, and S. Russo, "The software aging and rejuvenation repository: Http://openscience. us/repo/software-aging," in *2015 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE, 2015, pp. 108–113.

[17] R. Ghosh, F. Longo, F. Frattini, S. Russo, and K. S. Trivedi, "Scalable analytics for IaaS cloud availability," *IEEE Transactions on Cloud Computing*, vol. 2, no. 1, pp. 57–70, 2014.

[18] F. Longo, R. Ghosh, V. K. Naik, and K. S. Trivedi, "A scalable availability model for infrastructure as a service cloud," in *2011 IEEE/IFIP 41st International Conference on Dependable Systems & Networks (DSN)*. IEEE, 2011, pp. 335–346.

[19] E. Ataie, R. Entezari-Maleki, L. Rashidi, K. S. Trivedi, D. Ardagna, and A. Movaghar, "Hierarchical stochastic models for performance, availability, and power consumption analysis of iaas clouds," *IEEE Transactions on Cloud Computing*, 2017.

[20] A. Chilwan, A. Undheim, and P. E. Heegaard, "Effects of dynamic cloud cluster load on differentiated service availability," in *2012 21st International Conference on Computer Communications and Networks (ICCCN)*. IEEE, 2012, pp. 1–6.

[21] R. Ghosh, F. Longo, R. Xia, V. K. Naik, and K. S. Trivedi, "Stochastic model driven capacity planning for an infrastructure as a service cloud," *IEEE Transactions on Services Computing*, vol. 7, no. 4, pp. 667–680, 2013.

[22] R. Thakkar, R. Trivedi, and M. Bhavsar, "Experimenting with energy efficient vm migration in iaas cloud: Moving towards green cloud," in *International Conference on Future Internet Technologies and Trends*. Springer, 2017, pp. 56–65.

[23] M. Torquato, I. Umesh, and P. Maciel, "Models for availability and power consumption evaluation of a private cloud with vmm rejuvenation enabled by vm live migration," *The Journal of Supercomputing*, vol. 74, no. 9, pp. 4817–4841, 2018.

[24] R. Ghosh, V. K. Naik, and K. S. Trivedi, "Power-performance trade-offs in iaas cloud: A scalable analytic approach," in *2011 IEEE/IFIP 41st International Conference on Dependable Systems and Networks Workshops (DSN-W)*. IEEE, 2011, pp. 152–157.

[25] "Docker," https://www.docker.com/, accessed July 26, 2019.

[26] "Openvz," https://openvz.org/, accessed July 26, 2019.

[27] H. Jin, Z. Li, D. Zou, and B. Yuan, "Dseom: A framework for dynamic security evaluation and optimization of mtd in container-based cloud," *IEEE Transactions on Dependable and Secure Computing*, 2019.

[28] H. Khazaei, C. Barna, N. Beigi-Mohammadi, and M. Litoiu, "Efficiency analysis of provisioning microservices," in *2016 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE, 2016, pp. 261–268.

[29] S. Sebastio, R. Ghosh, A. Gupta, and T. Mukherjee, "Contav: a tool to assess availability of container-based systems," in *2018 IEEE 11th Conference on Service-Oriented Computing and Applications (SOCA)*. IEEE, 2018, pp. 25–32.

[30] A. N. Asadi, M. A. Azgomi, and R. Entezari-Maleki, "Evaluation of the impacts of failures and resource heterogeneity on the power consumption and performance of iaas clouds," *The Journal of Supercomputing*, vol. 75, no. 5, pp. 2837–2861, 2019.

[31] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, "Heterogeneity and dynamicity of clouds at scale: Google trace analysis," in *Proceedings of the Third ACM Symposium on Cloud Computing*. ACM, 2012, p. 7.

[32] D. Bruneo, "A stochastic model to investigate data center performance and QoS in IaaS cloud computing systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 3, pp. 560–569, 2014.

[33] M. Carvalho, W. Cirne, F. Brasileiro, and J. Wilkes, "Long-term SLOs for reclaimed cloud computing resources," in *Proceedings of the ACM Symposium on Cloud Computing*. ACM, 2014, pp. 1–13.

[34] R. Matos, J. Dantas, J. Araujo, K. S. Trivedi, and P. Maciel, "Redundant eucalyptus private clouds: Availability modeling and sensitivity analysis," *Journal of Grid Computing*, vol. 15, no. 1, pp. 1–22, 2017.

[35] M. Loreti and J. Hillston, "Modelling and analysis of collective adaptive systems with CARMA and its tools," in *Formal Methods for the Quantitative Evaluation of Collective Adaptive Systems*. Springer, 2016, pp. 83–119.

[36] QUANTICOL, "Carma," https://blog.inf.ed.ac.uk/quanticol/carma/, accessed Jan 18, 2018.

[37] M. Carvalho, D. A. Menascé, and F. Brasileiro, "Capacity planning for IaaS cloud providers offering multiple service classes," *Future Generation Computer Systems*, vol. 77, pp. 97–111, 2017.
[38] I. S. Moreno, P. Garraghan, P. Townend, and J. Xu, "Analysis, modeling and simulation of workload patterns in a large-scale utility cloud," *IEEE Transactions on Cloud Computing*, vol. 2, no. 2, pp. 208–221, 2014.
[39] A. A. Eldin, A. Rezaie, A. Mehta, S. Razroev, S. S. de Luna, O. Seleznjev, J. Tordsson, and E. Elmroth, "How will your workload look like in 6 years? analyzing wikimedia's workload," in *Cloud Engineering (IC2E), 2014 IEEE International Conference on*. IEEE, 2014, pp. 349–354.
[40] A. Khan, X. Yan, S. Tao, and N. Anerousis, "Workload characterization and prediction in the cloud: A multiple time series approach," in *Network Operations and Management Symposium (NOMS), 2012 IEEE*. IEEE, 2012, pp. 1287–1294.
[41] A. Verma, G. Dasgupta, T. K. Nayak, P. De, and R. Kothari, "Server workload analysis for power minimization using consolidation," in *Proceedings of the 2009 conference on USENIX Annual technical conference*. USENIX Association, 2009, pp. 28–28.
[42] Y. Huang, C. Kintala, N. Kolettis, and N. D. Fulton, "Software rejuvenation: Analysis, module and applications," in *The Twenty-Fifth International Symposium on Fault-Tolerant Computing. Pasadena, CA, USA*. IEEE, 1995, pp. 381–390.

Hongwu Lv received the B.A. degree in Information and Computing Science from Harbin Engineering University, China, in 2006, the PhD degree in Computer Applied Technology from Harbin Engineering University, China, in 2011. He is currently an assistant professor at College of Computer Science and Technology, Harbin Engineering University. His research interests include the study of formal modeling and performance evaluation with particular regards to the availability of cloud computing and to modeling edge clouds.

Jane Hillston gained a BA in Mathematics from the University of York, UK in 1985 and an MS in Mathematics from Lehigh University, USA in 1987. After a short spell in industry, she studied for a PhD in Computer Science at the University of Edinburgh, which was awarded in 1994. She was appointed Professor of Quantitative Modelling in the School of Informatics at the University of Edinburgh in 2006.

Her research is concerned with formal approaches to modelling dynamic behaviour, particularly the use of stochastic process algebras for performance modelling and stochastic verification. She has published over 100 journal and conference papers and held several UK and European grants.

Paul Piho received a MMath in Mathematics degree from the University of Durham, UK, in 2015. He is currently working towards a PhD degree in the School of Informatics at the University of Edinburgh, UK. His research interests are formal languages and scalable analysis methods for modelling and control of stochastic collective dynamics.

Huiqiang Wang received the BA in computer systems and engineering from the Harbin Institute of Technology, China in 1982, an MS in in computer applications from Harbin Shipbuilding Engineering Institute, China in 1985, a PhD degree in Computer Applied Technology from Harbin Engineering University, China, in 2005. He is currently an professor at College of Computer Science and Technology, Harbin Engineering University.

Huiqiang Wang's research interests is the security of distributed systems, particular regards to cognitive network, cloud computing and indoor positioning. He has published over 100 journal and conference papers.