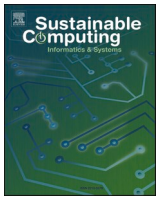
Sustainable Computing: Informatics and Systems 33 (2022) 100617

Contents lists available at ScienceDirect

Sustainable Computing: Informatics and Systems

journal homepage: www.elsevier.com/locate/suscom



# Modeling and evaluation of dispatching policies in IaaS cloud data centers using SANs

Ehsan Ataie [a,*], Reza Entezari-Maleki [b,c,d], Sayed Ehsan Etesami [e], Bernhard Egger [f], Leonel Sousa [d], Ali Movaghar [g]

[a] Department of Computer Engineering, University of Mazandaran, Babolsar, Iran
[b] School of Computer Engineering, Iran University of Science and Technology, Tehran, Iran
[c] School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Tehran, Iran
[d] INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, Lisbon, Portugal
[e] Electrical and Computer Engineering Department, University of Toronto, Toronto, Canada
[f] Department of Computer Science and Engineering, Seoul National University, Seoul, South Korea
[g] Department of Computer Engineering, Sharif University of Technology, Tehran, Iran

ARTICLE INFO

*Keywords:*
IaaS cloud
Dispatching policies
Performance evaluation
Analytical modeling
Stochastic activity network

ABSTRACT

Infrastructure-as-a-Service (IaaS) clouds not only have to meet business requirements but also need to consider other important metrics that influence the quality of service, such as performance, availability, and power consumption. In this paper, we present a Stochastic Activity Network (SAN) based analytical approach that simultaneously computes these metrics under a given load for IaaS cloud data centers. The model's high-level abstraction of IaaS clouds allows us to evaluate different resource management strategies and features, including cloud federation and dispatching policies. In a first step, we model a single rack including its autonomous local manager responsible for scheduling IaaS requests onto available PMs. In a second step, we present a unified model that represents an entire data center, including an energy-aware central manager, to take advantage of structure-awareness for managing cloud resources in an optimized way. We introduce and evaluate several dispatching policies that can be used by the cloud central manager to demonstrate the applicability of the unified SAN model. The model is validated against the well-known CloudSim framework. Extended simulations are also conducted to apply the proposed model to a real-world IaaS cloud.

## 1. Introduction

Over the past two decades, cloud computing has continuously risen in popularity thanks to its convenient service access and efficient resource usage. It has been adopted in many scenarios, including social networks, scientific computation and business applications [1,2]. In this context, providers of Infrastructure-as-a-Service (IaaS) clouds offer computing resources in the form of virtual machines (VMs), on which users can run their own programs [3,4]. Virtualization comes with many advantages, such as isolation of distinct co-located workloads, enabling consolidation of multiple workloads onto fewer servers for an improved resource utilization [5]. Furthermore, virtualization facilitates resource management and supports cloud providers in attaining important goals, such as minimizing power consumption [6,7].

Service level agreements (SLAs) define the obligations and expectations between providers and clients in terms of quality of service (QoS) metrics and penalties incurred when violations of these expectations occur [6,8]. A non-fulfillment of an SLA because of a small additional service time can cause a significant revenue loss to cloud service providers [9]. On the other hand, it has been shown that the average resource utilization of a data center is typically around 30% meaning that many resources of a data center remain idle most of the time [9]. Therefore, precise modeling and evaluation of performance, which help system providers and managers in assessing the impact of resource allocation policies on the operation of the data center, and its costs and benefits are of great importance in cloud environments.

Resources in cloud data centers are likely to become unavailable. Various reasons may cause a cloud resource to become unavailable, such

* Corresponding author.
*E-mail addresses:* ataie@umz.ac.ir (E. Ataie), entezari@iust.ac.ir (R. Entezari-Maleki), ehsan.etesami@mail.utoronto.ca (S.E. Etesami), bernhard@csap.snu.ac.kr (B. Egger), las@inesc-id.pt (L. Sousa), movaghar@sharif.edu (A. Movaghar).

https://doi.org/10.1016/j.suscom.2021.100617
Received 11 September 2021; Accepted 30 October 2021
Available online 25 November 2021

as power supply failure, software and hardware failure, and scheduled maintenance [10]. An hour of service unavailability costs the business sector over 108 thousand USD, and IT outages cause a revenue loss of more than USD 26.5 billion per year [11]. Failures of data center components cause additional energy consumption because the workload needs to be recomputed after recovery [12]. Data center outages also compromise the most important asset of cloud providers: the trust of users in the cloud [11]. Therefore, a cloud system's availability of services and resources is one of the most critical factors of client satisfaction and thus needs to be taken into account when modeling such systems.

In addition to performance and availability, the power consumption of cloud data centers is a topic that has recently received a lot of attention [13,14]. Assuming that currently known growth factors remain constant, the data centers' electricity consumption of 286 TWh in 2016 is expected to grow to around 320 TWh by 2030 [15]. As demand for computing and communication continues to grow, servers, networks, and data centers will consume more and more energy. Thus, concerns about power consumption of cloud infrastructures are growing. As an example, Amazon has announced the construction of a 150 MW wind farm to power its cloud-based Amazon Web Services (AWS) data centers in 2015 [16]. Today, Amazon operates several wind and solar farms around the world capable of producing more than 7.6 TWh of renewable energy annually, estimating to power all operations with 100% renewable energy by 2025 [17]. Therefore, the modeling and quantification of power consumption of cloud data center components, as well as the study of the impact of different resource management policies on power consumption, are important and can lead to improvements in energy efficiency and reduction in costs.

The main methods to evaluate the performance of a complex computer system, such as IaaS clouds, are based on measurement, simulation, and analytical modeling [18–23]. Measurement-based evaluation needs extensive experimentation with different workloads and system configurations. This may be impractical due to the large size of cloud systems as well as time and budget constraints. A promising approach for such large systems is simulation-based modeling because the low overhead of simulation allows us to perform a statistically significant number of experiments for a wide range of cloud configurations. Moreover, simulation models can be used for sensitivity analysis, forecasting, bottleneck analysis, what-if analysis, and data center design optimization. What-if questions can be easily answered using simulation, but it is time consuming to get a dependable answer. Applying analytical modeling to cloud computing can not only help providers to evaluate performance, dependability, and power related metrics under different situations but also prove valuable in terms of time and budget constraints.

In this paper, we present an analytical model to evaluate the power consumption, availability, and performance of cloud systems. A cloud system is composed of a number of racks, and each rack is composed of several physical machines (PMs). To evaluate such a system, we first model a single rack, which includes the autonomous local manager responsible for scheduling IaaS requests onto available PMs in the rack. In order to save power, it is assumed that idle PMs of a rack are powered down whenever there are no waiting requests in the local queue of the rack. Applying the model of a single rack, a unified second order model is presented in a subsequent step to evaluate an IaaS cloud data center that encompasses the central resource manager of the cloud. This central manager is responsible for dispatching requests to the racks or submitting them to federated clouds if the target racks are not able to host those requests. Afterwards, several dispatching policies that can be used by the cloud central manager are introduced, and evaluated based on different metrics.

The formalism used in this paper, to model an IaaS cloud data center, is the stochastic activity network (SAN), which is a stochastic generalization of petri nets (PNs) [24,25] defined for modeling and analyzing distributed systems [26,27]. Using SANs features, the representation of timeliness, parallelism, degradable performance, and fault tolerance is achieved [28]. Applying the SAN formalism, we can graphically model various components of a complex system, and then relate them using input/output gates, basic elements of the SAN. In addition to the unique capabilities of that formalism, its supporting tool Möbius [29] is also one of the incentives that motivates system analyzer to use it. Both the atomic and composite SAN models are supported by the Möbius tool. After graphically modeling a system, one can apply one of the existing analytical solvers of the Möbius to assess the system under analysis, or use its simulation engine to conduct experiments on a real system and simulate wide ranges of situations. In addition to modeling the cloud system using SANs, we extend the well-known cloud simulator *CloudSim* [30] to support global, local, and federation queues, plus management events. The modified CloudSim is then used to cross-validate the presented models.

The main contributions of this paper are summarized as follows.

(i) We present a new analytical model for individual racks of an IaaS cloud. The model encompasses a significant level of detail of the system, such as PMs' switching on and off, VM multiplexing, VM provisioning and serving, and failure/repair events of PMs.
(ii) Using a layered approach, we present an analytical model for an IaaS cloud data center. This model is based on the single rack model and provides a high-level abstraction of the system, making it possible to incorporate resource management strategies and features, including dispatching and federation policies.
(iii) Taking advantage of the presented models, various dispatching policies are introduced and assessed in terms of performance, availability, and power consumption to demonstrate the applicability of the presented models.

The remainder of this paper is organized as follows. Section 2 provides a review on related work on cloud performance and dependability analysis. Section 3 introduces the reference architecture and description of the system. In Section 4, we present the models to analyze the power consumption, availability, and performance of a single rack as well as an entire data center; several dispatching policies are introduced and modeled using the presented SANs. Section 5 introduces various metrics that are computed by a steady-state analysis of our SANs for a cloud data center. Numerical results obtained from the presented model for a typical cloud configuration and a cross-validation against the CloudSim framework are provided in Section 6. Moreover, Möbius is used to simulate a realistic configuration of cloud data centers in Section 6. Finally, the conclusion and future research directions are provided in Section 7.

## 2. Related work

State-space Markovian techniques have been used by several studies to model cloud systems. To reduce the modeling complexity of large-scale cloud data centers and services, some of the presented models have adopted an interacting stochastic sub-modeling approach and used the fixed-point iteration method to resolve dependencies between sub-models. Ghosh et al. [31] have presented an analytical approach for the end-to-end analysis of an IaaS cloud system's performability. Interacting stochastic sub-models were exploited to separately design performance, availability and performability sub-models. Pure performance analysis was provided assuming no failing resources. In the pure availability analysis, failures of PMs were considered. By assigning performance metrics as reward rates to the states of the availability model, the performability of the system was analyzed. Although the joint analysis of performance and availability has been addressed, power consumption of PMs was not considered in [31].

Ghosh et al. [32] have used the Markov reward approach to develop an analytical model for analyzing the trade-off between performance and power consumption in IaaS clouds. Similar to the models presented in [31], PMs were also grouped into three distinct pools. The VM

provisioning models, one for each pool, and the resource provisioning decision model are interacting sub-models based on continuous timed Markov chains (CTMC), designed to analytically solve the whole model using the fixed-point iteration technique. Unlike the approach proposed herein, failure and repair of PMs were not considered in [32]. The work presented in [31] was extended by introducing a monolithic model based on stochastic reward nets (SRNs) for IaaS clouds [33]. It discusses also how the interacting approach can decrease the model complexity both in execution time and storage space when compared to the monolithic analytic model. Failures and power consumption have not been taken into consideration in [33].

Liu et al. [34] have presented a monolithic analytical model based on SRNs for IaaS cloud data centers, where PMs were grouped into three distinct pools with different average repair times. To cope with the scalability problem, a decomposition technique was applied to approximate the monolithic model using interactive sub-models, where each pool was modeled as a separate SRN-based sub-model and the interactions among the sub-models were presented by an import graph. The cyclic dependencies among the sub-models were solved by exploiting the fixed-point iteration technique. Simulation was also exploited to investigate the sensitivity of large-scale data centers availability regarding repair policies and system input parameters. Mainly availability was well analyzed in [34], but performance and power consumption were not considered.

Ghosh et al. [35] have presented a stochastic approach to evaluate the availability of an IaaS cloud in which failures were reduced through migration of physical machines among the three different pools referred above. Results obtained from the numerical analysis of the approximate and monolithic models were compared in terms of the size of the underlying Markov chains, the downtime, the number of machines in each category, and the solution time. In contrast to the approach presented herein, the requests arrival and serving process were not considered in [35], being the work only focused on cloud availability. Khazaei et al. [36] have analyzed the steps for serving an IaaS request and the associated delays. CTMCs were applied to model different modules of the system. The term *supertask* was used to refer to a set of requests that one user submits simultaneously, each of which for a single VM. The effect of the input parameters of the models on the response time and rejection probability were also investigated. In other research of the same authors, the rejection probability and the total delay of supertasks in an IaaS cloud, where live migration of VMs is allowed, have been evaluated [37]. They have proposed to divide the presented model into two interacting CTMC-based models, each of which corresponding to a different serving step in a cloud-based system. The models were then jointly solved to achieve the final solution. In contrast to [36,37], our proposed models deal with availability and power consumption.

Ataie et al. [23] have proposed an SRN for a group of PMs, named a cluster, in an IaaS cloud data center. This SRN was then used to derive a monolithic model for the complete cloud data center. To overcome the limitations of the monolithic model in terms of scalability, two approximate SRN models have been proposed to evaluate the percentage of available PMs, the response time, and the power consumed by PMs. Results show that approximate models generate a state space smaller than the monolithic model, without sacrificing the accuracy of the output measures. Authors in [23] adopted a random policy to dispatch incoming requests among underlying clusters while smart dispatching policies are introduced herein. Moreover, power consumption of network equipments and cooling systems were not considered in [23]. Entezari-Maleki et al. [18] have applied the SAN formalism to simultaneously model the application of the two methods, which are grouping virtualized servers in different power consumption pools and dynamic voltage and frequency scaling (DVFS), in order to save power within an IaaS cloud ecosystem. After modeling the system, performance measures such as the throughput and the blocking probability, together with the power consumption metric, were defined and evaluated. Furthermore, a sensitivity analysis of the variation of input parameters was conducted. Nevertheless, the approach proposed in [18] does not account for the behavior or failures/repairs of servers. Moreover, dispatching requests is not a concern in that work. However, the concept of DVFS modeled in [18] can be complementary to the work presented in this paper.

Asadi et al. [38] have presented SRN-based analytical models to analyze the impact of process migration methods and resource allocation algorithms on the performance and power consumption of virtualized systems. Processes, also named resource requesters, were categorized into three categories, according to their required resources. The resource allocation algorithms first-fit, best-fit, worst-fit, and random were used to allocate a computer to a process. Then, the impact of applying two process migration methods, power-aware and performance-aware, on power and performance was investigated. In contrast to the approach presented herein, availability of resources was not considered in [38]. Moreover, the hierarchical resource management architecture considered herein is different from the one investigated in [38].

Other approaches have been proposed for analyzing the cloud system behavior through queuing networks. Khazaei et al. [39] have modeled a cloud data center using a queuing system with single task arrivals and a buffer of finite capacity for tasks. The performance of the model was evaluated by using both analytical and approximate Markov chain models, allowing to compute the complete probability distribution of the response time and the number of tasks. Furthermore, the blocking probability and the probability of immediate service were evaluated. Xia et al. [19] have proposed a queuing network-based analytical framework to model and evaluate cloud data centers that are enabled with dynamic voltage scaling (DVS) capability. Several energy- and performance-related metrics, including energy consumption rate and VM completion time were considered. The effectiveness of the proposed model was validated through a case study conducted on a sample data center. On the contrary, we choose SAN formalism for modeling which facilitates automated generation of Markov chains. Furthermore, availability was not taken into consideration neither in [39] nor in [19].

There exist other approaches that use hierarchical models to evaluate cloud environments according to different metrics. Qiu et al. [40] have proposed a hierarchical model for analyzing and evaluating three correlated measures of a large service in a cloud system, namely reliability, performance, and power consumption. The proposed modeling method applies Markov models, queuing theory, and the Bayesian approach. Moreover, a profit optimization model was developed based on the proposed correlation model, and an additional parameter was introduced for evaluating and balancing the trade-off between power consumption and performance. Sousa et al. [41] have proposed a modeling strategy based on a hierarchical approach for planning the cloud infrastructure. They have employed state-space models, such as stochastic petri nets (SPNs), and combinatorial models, such as reliability block diagrams, and combined them to represent the dependencies between different components of the infrastructure of a cloud system. The proposed modeling strategy can be used to select the proper cloud infrastructure based on dependability and cost requirements. Moreover, a stochastic model generator was implemented for providing the automatic generation of cost and dependability models. In contrast to our work, dispatching policies at rack level were not considered in [40,41]. In addition to modeling VMs in cloud environments, mathematical models have been proposed for the analysis of other distributed systems or alternative virtualization approaches and runtime environments. Entezari-Maleki et al. [42] have presented a SAN model for an entire grid environment, for which the availability of both the resource management system and grid resources was investigated. Furthermore, the impact of different task scheduling algorithms on the grid availability was considered. However, performance and power consumption were not analyzed in [42]. Sebastio et al. [43] have presented non-state-space and state-space analytical models for analyzing container's availability. Analytical and simulative solutions were

obtained for the developed models. An open-source software tool for evaluating the availability of a containerized system based on simulation was also developed.

A comparison of the related work discussed in this section is presented in Table 1. To summarize, only few approaches apply analytical models to simultaneously analyze power consumption, availability, and performance of cloud systems. In contrast to existing techniques, our approach enables data center designers to model and evaluate a layered structure of an IaaS cloud system containing many racks. It takes several aspects of real clouds into account that were not usually considered in previously presented approaches. In addition, the approach presented in this paper allows incorporating different resource management strategies, including dispatching and federation policies, which increases the practical interest of the approach.

## 3. System description

In order to investigate a more realistic model of cloud systems, we consider a layered architecture in which two management layers are used by an IaaS cloud provider in a hierarchical arrangement. A similar structure was considered in [40,44–46,20]. A high-level representation of the architecture modeled in this work is shown in Fig. 1. In this architecture, the cloud system has a main component, named *Cloud Central Manager* (CCM), responsible for monitoring and managing cloud resources. The CCM receives VM requests from the users. The requests are submitted to the *global queue* of the CCM, which typically processes requests in a first in first out (FIFO) order. According to the active resource allocation policy, the CCM chooses an autonomous group of PMs on top of which the requested VMs can be instantiated, and then dispatches the requests to the target group. In this work, a rack of servers comprises a group of PMs, however other groupings could be considered. Moreover, the runtime environment considered in the reference architecture is the VM, meaning that we do not investigate container-based systems or serverless computing. Different resource allocation policies, implemented in the CCM of the cloud provider, affect performance, availability, and power consumption of the whole data center [47].

After a request has been dispatched by the CCM, it is moved into the *local queue* of the *Local Manager* (LM) of the target rack. If the local queue of the target LM is full, instead of dropping the request, it is put into a *federation queue* to be sent to the federated clouds. Cloud federation allows various clouds that belong to the same or different organizations to dynamically join each other in order to achieve a common goal, e.g. optimization of resource utilization [8]. The LMs can be treated as VM allocators. They decide the PM on which to provision a request, and then they send the request to the selected PM. We assume, in this paper, that the incoming VMs are individual. Therefore, groups of VMs are not herein considered. Through VM multiplexing, a cloud system can provide more virtual resources than the actual number of physical ones. This technique, common in modern systems, is implemented by allocating multiple VMs to a single PM [48]. The virtual machine monitor (VMM) of each PM is responsible for allocating the required resources to each VM. We assume that each IaaS request can be served by a single VM. This is a common assumption in related work [8,31,32,36,47,49,50].

Idle PMs, hosting no provisioned VMs, can be switched to standby or off modes to save power. Usually, the power consumption of a standby PM is so low that it can be neglected; hence, we treat the *standby* mode as *powered-off* in the remainder of this paper. When the LM of a rack receives a request, the VM can be provisioned immediately if there is an available VM in the LM's *virtual resource pool*, which means that there exists at least one powered-on PM in the rack with sufficient system resources. Otherwise, if there is a powered-off PM, the machine is first switched on before deploying the requested VM. If all PMs are fully occupied, the request has to wait until some tasks finish and the PMs release the associated VMs. The component of the LM which is responsible for switching PMs to on and off modes is called *power manager*.

Physical machines of a data center are failure-prone. The failure rate of a powered-off PM is typically smaller than that of a powered-on PM, i.e. the mean time between failure (MTBF) of a powered-off PM is larger than of a powered-on one. While the failure of a powered-off PM has no impact on running VMs, the failure of a powered-on PM causes both the available and provisioned VMs on that PM to fail. A failed VM should be restarted after its PM is repaired. However, when a provisioned VM fails, its associated request should be restarted on another VM available for hosting that request. Hence, the request corresponding to a failed provisioned VM should be returned to the waiting queue of the LM to be processed later.

Besides computing servers, the data center architecture under study encompasses two other structural components of interest: *cooling systems* and *network switches*. We assume that the data center has several rack-based cooling systems. In a rack-based design, the cold air is delivered directly inside the rack to reduce the length of the airflow path. This structure allows cooling capacity to be adjusted to the actual needs of specific racks, thus increasing the energy efficiency [51]. In the architecture considered in this paper, there is a *Top of Rack (ToR) switch* dedicated to each rack, which physically connects all PMs inside that rack to the network [21,52–54]. Furthermore, it is assumed that an *aggregate switch* consolidates and aggregates traffic taken from the

**Table 1**
Summary of the related work discussed in Section 2. SS: state-space; MC: Markov chains; QN: queuing networks; RBD: reliability block diagram; FT: fault trees; SIM: simulation; MSR: measurement; PER: performance; AVL: availability; POW: power; REL: reliability; CST: cost.

| Related work | State-space or non-state-space | Context | Main formalism | Other formalisms | Validation technique | Simulation tools | Evaluation measures |
|---|---|---|---|---|---|---|---|
| Ghosh et al. [31] | SS | Cloud | MC | – | – | – | PER, AVL |
| Ghosh et al. [32] | SS | Cloud | MC | – | – | – | PER, POW |
| Ghosh et al. [33] | SS | Cloud | SRN | MC | – | – | PER |
| Liu et al. [34] | SS | Cloud | SRN | – | SIM | SPNP | AVL |
| Ghosh et al. [35] | SS | Cloud | SRN | MC | SIM | SPNP | AVL |
| Khazaei et al. [36] | SS | Cloud | MC | – | SIM | Artifex | PER |
| Khazaei et al. [37] | SS | Cloud | MC | – | – | – | PER |
| Ataie et al. [23] | SS | Cloud | SRN | – | SIM | CloudSim | PER, AVL, POW |
| Entezari-Maleki et al. [18] | SS | Cloud | SAN | – | SIM | CloudSim | PER, POW |
| Asadi et al. [38] | SS | Cloud | SRN | – | SIM | CloudSim | PER, POW |
| Khazaei et al. [39] | SS | Cloud | QN | – | SIM | Artifex | PER |
| Xia et al. [19] | SS | Cloud | QN | – | MSR/ SIM | SimEvents | PER, POW |
| Qiu et al. [40] | SS | Cloud | MC | QN, Bayesian | SIM | – | REL, PER, POW |
| Sousa et al. [41] | Both | Cloud | SPN | RBD | SIM | SMG4CIP | AVL, CST |
| Entezari-Maleki et al. [42] | SS | Grid | SAN | – | SIM | Möbius | AVL |
| Sebastio et al. [43] | Both | Cloud | SRN | FT | SIM | ContAV | AVL |

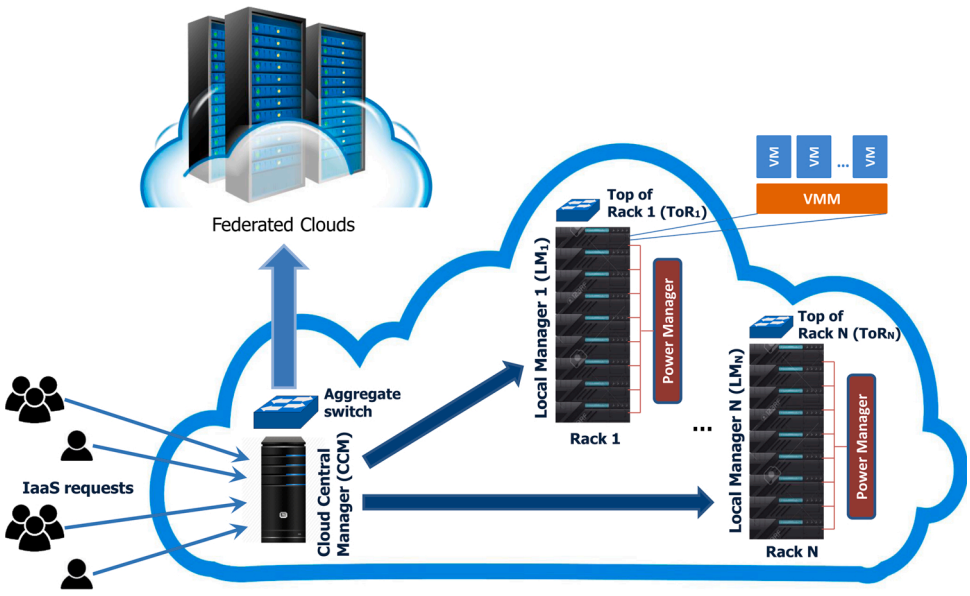



**Fig. 1.** The hierarchical resource management architecture for an IaaS cloud.

uplinks of the ToR switches into a high speed link [53,54]. In such an architecture, the total power consumption of a data center can be reduced by switching idle PMs, ToR switches, and cooling systems to low-power modes whenever possible.

## 4. Proposed models and policies

In order to evaluate the performance, availability, and power consumption of different dispatching policies that can be used by an IaaS cloud resource manager, according to the architecture introduced in Section 3, a SAN for a single rack is proposed in Section 4.1. This SAN model is the first step to derive a unified model in Section 4.2 that captures the behavior of the entire data center. Afterwards, various dispatching policies are introduced in Section 4.3 to be utilized in the CCM of the model proposed in Section 4.2, for dispatching VM requests to the existing racks in the data center. The research methodology adopted in this paper follows the design science research approach, in which building and evaluating a model are intertwined tasks.

### 4.1. SAN model of a rack

The SAN model proposed to analyze a rack, in a cloud data center, is depicted in Fig. 2. It assumes that all timed activities are exponentially distributed [8,31,32,35,36,47,55]. The timed activity $T_{rack}$ models the request arrival process to a rack. When $T_{rack}$ completes, one token is moved into place $P_q$ by output gate $OG_{rack}$. Place $P_q$ represents the local queue of the rack, and a token in this place represents a queued request. If the number of tokens in place $P_q$ reaches $Q_r$, the maximum size of the input queue of a rack, input gate $IG_{rack}$ prevents the activity $T_{rack}$ from completion. The completion rate of $T_{rack}$ is $\lambda_{rack}$.

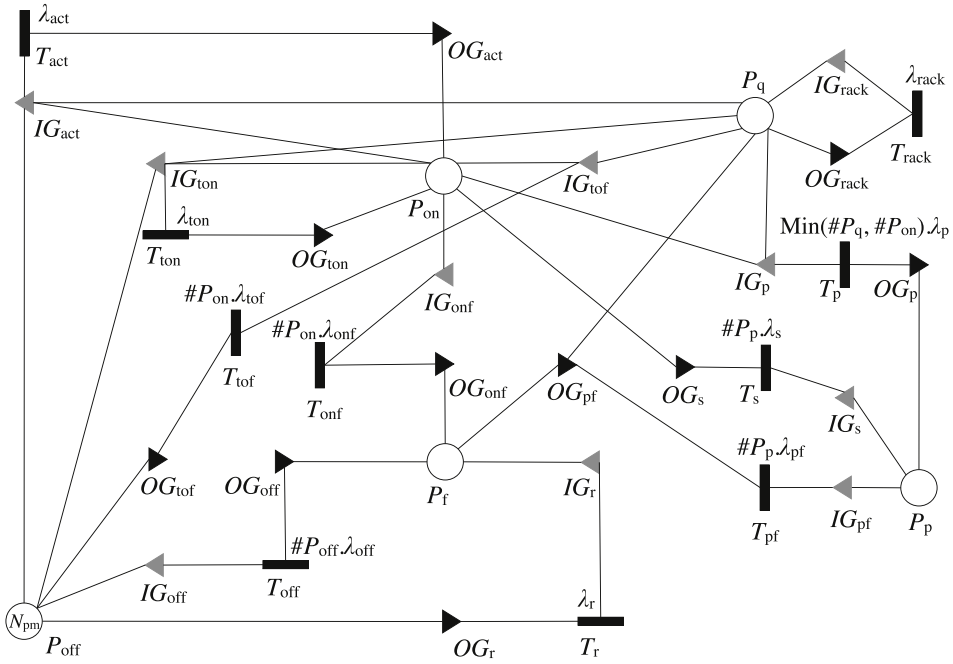



**Fig. 2.** The SAN proposed to model a rack.

Place $P_{on}$ models the pool of VMs available in the system, and tokens in this place denote the non-provisioned VMs that will be provisioned later and allocated to incoming VM requests. If there is a token in both places $P_q$ and $P_{on}$, timed activity $T_p$ is activated, and then completes. When this activity completes, a token is removed from each of the places $P_q$ and $P_{on}$ by input gate $IG_p$, and a new token is put into place $P_p$ by output gate $OG_p$. Multiple VM provisioning tasks can execute concurrently, hence, the actual completion rate of activity $T_p$ is the minimum number of tokens inside places $P_q$ and $P_{on}$ times $\lambda_p$, where $\lambda_p$ is the provisioning rate of a VM. If the number of tokens in place $P_q$ is larger than the number of tokens in place $P_{on}$, the existence of a token in place $P_{off}$ is checked by the input gate $IG_{ton}$; place $P_{off}$ represents powered-off PMs. In order to reduce the power consumed by the servers, as well as, by the corresponding cooling system and network switches, we attempt to switch PMs off whenever no VMs are requested. Thus, we assume that all $N_{pm}$ servers of the rack are initially in place $P_{off}$. If there is a token in $P_{off}$, the timed activity $T_{ton}$ completes. As soon as this activity completes, one token is removed from place $P_{off}$, and $L$ tokens are added to place $P_{on}$ through gates $IG_{ton}$ and $OG_{ton}$, respectively. Using this mechanism, a powered-off PM is switched on whenever there are more requests than available VMs. Consequently, $L$ new VMs are added to the virtual resource pool. The parameter $L$ defines the maximum number of VMs that can be concurrently provisioned on a PM. Otherwise, the request has to wait since there is neither an available VM nor a powered-off PM, which means that all PMs have been completely allocated to the requests or have failed. The completion rate of $T_{ton}$ is $\lambda_{ton}$.

In order to reduce the time required to provision a VM, a proactive mechanism for switching on powered-off PMs is embedded in the model through the activity $T_{act}$. If there exist no tokens in places $P_q$ and $P_{on}$, which means neither a request is waiting in the queue nor a VM is available, and there is a token in place $P_{off}$, indicating that there exists a powered-off PM, activity $T_{act}$ is activated. When this activity completes, a token is removed from $P_{off}$ by input gate $IG_{act}$, and $L$ tokens are placed into $P_{on}$ by output gate $OG_{act}$. Completion rate of activity $T_{act}$ is $\lambda_{act}$. If this rate is chosen carefully, some VMs would be available in $P_{on}$ just a short time before a request arrives to $P_q$, leading to a smaller mean waiting time for requests.

A token in place $P_p$ signals that a VM has already been provisioned for a VM request. Timed activity $T_s$ models the actual VM serving process. Activity $T_s$ can complete if there is a token in place $P_p$. Upon completion of this activity, a token is removed from place $P_p$ by input gate $IG_s$ and added to place $P_{on}$ by output gate $OG_s$. This token movement from place $P_p$ to place $P_{on}$ represents that a provisioned VM has finished its own job in serving a VM request and is put back into the pool of available VMs ready to be allocated to a new request. The completion rate of activity $T_s$ is $[\#\ P_p] \cdot \lambda_s$, where $[\#\ P_p]$ and $\lambda_s$ denote the number of tokens in place $P_p$ and the service rate of a VM request, respectively. When there are at least $L$ tokens in place $P_{on}$ and no token in place $P_q$, activity $T_{tof}$ is activated. When this activity completes, $L$ tokens are removed from place $P_{on}$, and one token is added to place $P_{off}$ through gates $IG_{tof}$ and $OG_{tof}$, respectively. Using this mechanism, idle powered-on PMs are switched off to save power. The completion rate of $T_{tof}$ activity is $\lfloor [\#\ P_{on}]/L \rfloor \cdot \lambda_{tof}$, where $[\#\ P_{on}]$ and $\lambda_{tof}$ denote the number of tokens in place $P_{on}$ and the rate of switching a single PM off, respectively.

Physical machines, and consequently VMs running on them, can fail. Timed activity $T_{pf}$ models the failure of provisioned VMs due to the failure of their related PMs. If $T_{pf}$ completes, $L$ tokens are removed from place $P_p$ by input gate $IG_{pf}$, and one and $L$ tokens are added to places $P_f$ and $P_q$, respectively, by output gate $OG_{pf}$. Using this mechanism, the requests associated to failed VMs return to the local queue to be provisioned later on other VMs. The completion rate of $T_{pf}$ is $\lfloor [\#\ P_p]/L \rfloor \cdot \lambda_{pf}$, where $1/\lambda_{pf}$ represents the MTBF of a PM containing provisioned VMs. Tokens in place $P_f$ represent failed PMs waiting for repair. Available VMs may also fail due to the failure of their hosting PMs. Activity $T_{onf}$ models this failure event. Upon completion of $T_{onf}$, $L$ tokens are removed from place $P_{on}$, and one token is added to place $P_f$ through gates $IG_{onf}$ and $OG_{onf}$. The completion rate of $T_{onf}$ is $\lfloor [\#\ P_{on}]/L \rfloor \cdot \lambda_{onf}$, where $1/\lambda_{onf}$ denotes the MTBF of a powered-on PM with available VMs. On the other hand, powered-off PMs can also fail. The failure event of powered-off PMs is modeled by timed activity $T_{off}$. When activity $T_{off}$ completes, a token is moved from place $P_{off}$ to place $P_f$ through gates $IG_{off}$ and $OG_{off}$. The completion rate of $T_{off}$ is $[\#\ P_{off}] \cdot \lambda_{off}$, where $[\#\ P_{off}]$ represents the number of tokens in place $P_{off}$, and $1/\lambda_{off}$ denotes the MTBF of a powered-off PM. Activity $T_r$ represents the repair process of failed PMs, and it is activated when there is at least one token in place $P_f$. When it completes, with rate $\lambda_r$, one token is removed from place $P_f$ by input gate $IG_r$, and another one is deposited into place $P_{off}$ by output gate $OG_r$, meaning that a repaired PM is accessible through powered-off state. Table 2 lists the predicates and functions for all input and output gates of the presented SAN model.

### 4.2. SAN model of an entire data center

The SAN proposed in Section 4.1 to model a single rack is applied to model an IaaS cloud data center containing several racks. As described in Section 3, the hierarchical structure considered herein contains two management levels: the CCM, as the data center-level management layer, and the LM as the rack-level management layer. The CCM receives the user requests for VMs and, based on the active policy, decides to which LM the request is dispatched. Afterwards, the target LM decides how to serve the request according to its own available resources. The proposed model, as depicted in Fig. 3, considers the existence of several racks, each operating according to the SAN model shown in Fig. 2.

Timed activity $T_{in}$ represents request arrivals to the data center. When $T_{in}$ completes, a token is added to place $P_{in}$ by output gate $OG_{in}$. Place $P_{in}$ models the global queue of the cloud data center, and tokens in this place show VM requests waiting in the queue to be dispatched. If the number of tokens inside place $P_{in}$ reaches $Q_d$, the maximum size of the global queue, input gate $IG_{in}$ prevents activity $T_{in}$ from completion. The completion rate of $T_{in}$ is $\lambda_{in}$. If a token exists in place $P_{in}$, activity $T_{sl}$ is activated. This activity models the dispatching of VM requests to the racks. When it completes, a token is removed from $P_{in}$ by input gate $IG_{sl}$, and a token is added to either $P_{q_i}$, $1 \le i \le N_r$, or $P_{out}$. The $N_r$ stands for the number of racks existing in the cloud data center, place $P_{q_i}$ represents the local queue of $rack_i$, and place $P_{out}$ serves as a holder for requests that are going to be submitted to federated clouds. The completion rate of $T_{sl}$ is $\lambda_{sl}$.

Details of the output function of gate $OG_{sl}$ depend on the specific

**Table 2**
Gate predicates/functions of the SAN model presented in Fig. 2.

| Gate | Predicate | Function |
|---|---|---|
| $IG_{rack}$ | $[\#\ P_q] < Q_r$ | |
| $OG_{rack}$ | | $[\#\ P_q]$ ++ |
| $IG_p$ | $([\#\ P_q] > 0)$ && $([\#\ P_{on}] > 0)$ | $[\#\ P_q]$ −− |
| | | $[\#\ P_{on}]$ −− |
| $OG_p$ | | $[\#\ P_p]$ ++ |
| $IG_{ton}$ | $([\#\ P_{off}] > 0)$ && $([\#\ P_q] - [\#\ P_{on}] > 0)$ | $[\#\ P_{off}]$ −− |
| $OG_{ton}$ | | $[\#\ P_{on}] \mathrel{+}= L$ |
| $IG_{act}$ | $([\#\ P_{off}] > 0)$ && $([\#\ P_q] == 0)$ && $([\#\ P_{on}] == 0)$ | $[\#\ P_{off}]$ −− |
| $OG_{act}$ | | $[\#\ P_{on}] \mathrel{+}= L$ |
| $IG_{tof}$ | $([\#\ P_{on}] >= L)$ && $([\#\ P_q] == 0)$ | $[\#\ P_{on}] \mathrel{-}= L$ |
| $OG_{tof}$ | | $[\#\ P_{off}]$ ++ |
| $IG_s$ | $[\#\ P_p] > 0$ | $[\#\ P_p]$ −− |
| $OG_s$ | | $[\#\ P_{on}]$ ++ |
| $IG_{pf}$ | $[\#\ P_p] >= L$ | $[\#\ P_p] \mathrel{-}= L$ |
| $OG_{pf}$ | | $[\#\ P_f]$ ++ |
| | | $[\#\ P_q] \mathrel{+}= L$ |
| $IG_{onf}$ | $[\#\ P_{on}] >= L$ | $[\#\ P_{on}] \mathrel{-}= L$ |
| $OG_{onf}$ | | $[\#\ P_f]$ ++ |
| $IG_{off}$ | $[\#\ P_{off}] > 0$ | $[\#\ P_{off}]$ −− |
| $OG_{off}$ | | $[\#\ P_f]$ ++ |
| $IG_r$ | $[\#\ P_f] > 0$ | $[\#\ P_f]$ −− |
| $OG_r$ | | $[\#\ P_{off}]$ ++ |

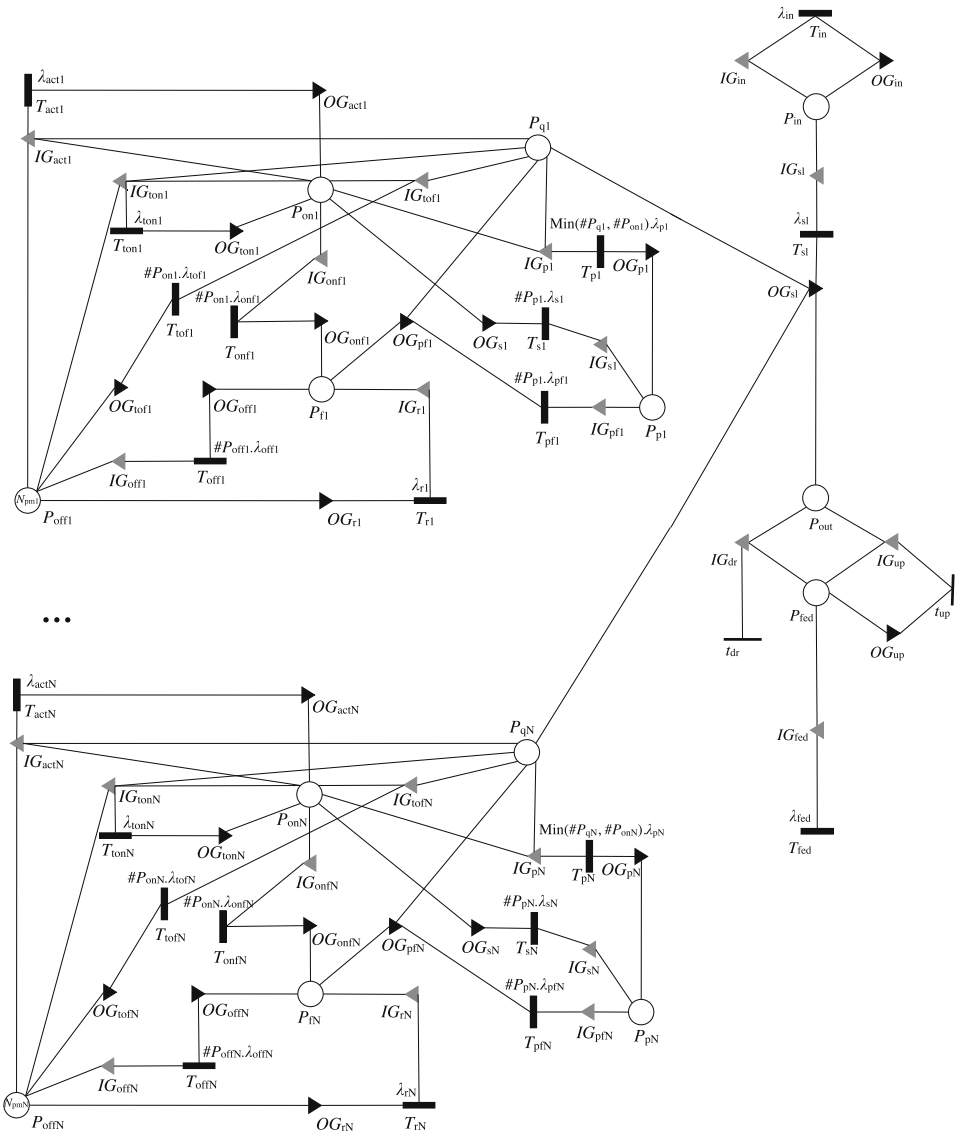



**Fig. 3.** The SAN proposed to model an entire data center.

policy applied by the CCM to dispatch VM requests among the racks. Hence, the output function differs from one policy to another. However, besides the specific strategy employed to dispatch the VM requests, it is assumed that one of the existing racks is chosen as the target rack to host the request. After selecting the target rack according to the predefined dispatching mechanism, the local queue of the rack is checked. If there exists enough space in the queue to hold the VM request, the request is submitted to that rack. Otherwise, the token representing the VM request is put into a temporary place, $P_{out}$, to be sent to federated clouds later. The details of the places, activities, and gates of the individual racks in Fig. 3 are the same as those presented in Section 4.1.

Place $P_{fed}$ models the federation queue in which requests that are going to be submitted to the federated clouds are waiting. When there is a token in place $P_{out}$, either $t_{up}$ or $t_{dr}$ is activated. If the number of tokens inside place $P_{fed}$ is equal to or greater than $Q_f$, the federation queue is full and cannot accept more requests. In this situation, instantaneous activity $t_{dr}$ is activated, and it can complete. This activity models the event of dropping a request when the federation queue is full. After completion of activity $t_{dr}$, a token is removed from place $P_{out}$ by input gate $IG_{dr}$. On the other hand, if the number of tokens inside place $P_{fed}$ is smaller than $Q_f$, then instantaneous activity $t_{up}$ is activated and can complete. Upon completion of this activity, one token is removed from place $P_{out}$ by input gate $IG_{up}$, and a new token is deposited into place $P_{fed}$ by output gate $OG_{up}$. Using this activity, a waiting request is moved from temporary place $P_{out}$ to the federation queue modeled by $P_{fed}$. Timed activity $T_{fed}$ represents the process of submitting requests to the federated clouds and it is activated if a token exists in $P_{fed}$. When this activity completes, a token is removed from place $P_{fed}$ by input gate $IG_{fed}$. The completion rate of $T_{fed}$ is $\lambda_{fed}$. The predicates and functions corresponding to the input and output gates of the SAN model of Fig. 3 are represented in Table 3.

**Table 3**
Gate predicates/functions of the SAN model presented in Fig. 3.

| Gate | Predicate | Function |
|---|---|---|
| $IG_{in}$ | $[\#\ P_{in}] < Q_d$ | |
| $OG_{in}$ | | $[\#\ P_{in}]$ ++ |
| $IG_{sl}$ | $[\#\ P_{in}] > 0$ | $[\#\ P_{in}]$ −− |
| $IG_{dr}$ | $([\#\ P_{out}] > 0)$ && $([\#\ P_{fed}] >= Q_f)$ | $[\#\ P_{out}]$ −− |
| $IG_{up}$ | $([\#\ P_{out}] > 0)$ && $([\#\ P_{fed}] < Q_f)$ | $[\#\ P_{out}]$ −− |
| $OG_{up}$ | | $[\#\ P_{fed}]$ ++ |
| $IG_{fed}$ | $[\#\ P_{fed}] > 0$ | $[\#\ P_{fed}]$ −− |

### 4.3. Dispatching policies

Various policies are proposed for dispatching VM requests by the CCM component of the architecture shown in Fig. 1. The following two definitions are introduced for describing the policies.

**Definition 1.** Computing Capacity of $Rack_i$

For a given rack, $rack_i$, the computing capacity is expressed by the total number of available VMs and potential virtual resources inside the rack minus the number of waiting requests in the local queue of $rack_i$. Potential virtual resources mean VMs that can become available if powered-off PMs are switched on. More precisely, the computing capacity of $rack_i$ is computed by Eq. (1):

$$\mathrm{RCC}_i = [\#P_{\mathrm{on}_i}] + L{\cdot}[\#P_{\mathrm{off}_i}] - [\#P_{q_i}] \tag{1}$$

where $[\#P_{\mathrm{on}_i}]$, $[\#P_{\mathrm{off}_i}]$, and $[\#P_{q_i}]$ are the numbers of tokens in places $P_{on}$, $P_{off}$, and $P_q$ of $rack_i$, respectively.

**Definition 2.** Queue Capacity of $Rack_i$

The queue capacity represents the free capacity in the local queue of $rack_i$ and is defined as the maximum size of the local queue minus the number of requests that are currently enqueued in the queue. The queue capacity is computed by Eq. (2):

$$\mathrm{RQC}_i = Q_{r_i} - [\#P_{q_i}] \tag{2}$$

where $Q_{r_i}$ and $[\#P_{q_i}]$ are the maximum queue size and the number of tokens in place $P_q$ of $rack_i$, respectively.

It is assumed that the CCM is structure-aware, able to calculate the computing capacity and the queue capacity of each rack by receiving feedback information from the racks. According to Definitions 1 and 2, two well-known strategies, *best-fit* and *worst-fit*, are used to setup dispatching policies. Five different policies are considered in the proposed SAN model for choosing the target rack.

- ***Policy 1***: CCM chooses the rack with the *minimum computing capacity*.
- ***Policy 2***: CCM chooses the rack with the *maximum computing capacity*.
- ***Policy 3***: CCM chooses the rack with the *minimum queue capacity*.
- ***Policy 4***: CCM chooses the rack with the *maximum queue capacity*.
- ***Policy 5***: CCM *randomly* chooses a target rack. The CCM is non-structure-aware, which means it is not capable of collecting feedback from the LMs. In addition, the management layers are assumed to be non-energy-aware. Idle PMs are never powered off, the ToR switches do not switch to sleep mode, cooling systems are always on, and the proactive mechanism is disabled.

While policies *1* and *3* select the best-fitting target rack, policies *2* and *4* follow the worst-fit strategy. All these policies are applied to the model presented in Fig. 3. The random dispatch, *policy 5*, is used as a baseline. For the random policy, unused parts of the SAN model from Fig. 3 are eliminated. This comprises activities corresponding to the proactive mechanism and the switching off of PMs. Moreover, activity $T_{sl}$ and its related gates, used to model the CCM component and its decision making algorithm, are replaced with $N_r$ distinct activities modeled by $T_{\mathrm{sl}_i}$ and their corresponding input/output gates, $\mathrm{IG}_{\mathrm{sl}_i}$ and $\mathrm{OG}_{\mathrm{sl}_i}$, $1 \le i \le N_r$. In this modified model, if a token exists in place $P_{in}$, all $T_{\mathrm{sl}_i}$ activities are activated and compete for completion. By completion of one of these activities, the input gate associated to the completed activity removes a token from $P_{in}$ and the corresponding output gate adds a token to either $P_{q_t}$, $1 \le t \le N_r$, the local queue of the target rack, or $P_{out}$, depending on whether the local queue of the target rack has free space or not. The completion rate of $T_{\mathrm{sl}_i}$ is $\lambda_{\mathrm{sl}_i}$.

## 5. Performance metrics

This section introduces figures of merit that are obtained from solving the SAN model introduced in Section 4.2. These metrics can be computed using Markov reward models (MRMs) [56]. The MRM approach assigns reward rates to each feasible marking of a SAN model before computing the expected rewards in the steady-state. It is also possible to associate impulse rewards to an activity of a SAN model, which then result in the expected throughput of that activity. In the following, some measures of interest are introduced.

The **federation probability ($\mathbf{Pr_{fed}}$)** represents the probability of a request to be sent to the federated clouds. For the SAN model presented in Fig. 3, the federation probability can be derived by Eq. (3):

$$\mathrm{Pr}_{\mathrm{fed}} = \frac{\mathrm{Thr}(T_{\mathrm{fed}})}{\mathrm{Thr}(T_{\mathrm{sl}})} \tag{3}$$

where $Thr(T_{fed})$ and $Thr(T_{sl})$ represent the expected throughputs of timed activities $T_{fed}$ and $T_{sl}$, respectively. The throughputs are computed by defining impulse rewards for those activities, returning the value 1 whenever they complete. In the simplified model for *policy 5*, the federation probability is computed by Eq. (4):

$$\mathrm{Pr}_{\mathrm{fed}} = \frac{\mathrm{Thr}(T_{\mathrm{fed}})}{\Sigma_{i=1}^{N_r}\mathrm{Thr}(T_{\mathrm{sl}_i})} \tag{4}$$

where $Thr(T_{fed})$ and $\mathrm{Thr}(T_{\mathrm{sl}_i})$ represent the steady-state throughputs of activities $T_{fed}$ and $T_{\mathrm{sl}_i}$, respectively, and $N_r$ is the total number of racks in the data center.

The **mean size of local queues (MQS)** represents the mean number of requests waiting in the local queues of all racks. It can be obtained by Eq. (5):

$$\mathrm{MQS} = \Sigma_{i=1}^{N_r} E[r_i^{\mathrm{ql}}] \tag{5}$$

where $r_i^{\mathrm{ql}}$ stands for the reward function that counts the number of tokens in place $P_q$ of $rack_i$.

The **mean waiting time (MWT)** metric defines the mean time that requests wait in the local queues of the racks. For the SAN model of Fig. 3, it is computed by Eq. (6), by applying the Little's law [57]:

$$\mathrm{MWT} = \Sigma_{i=1}^{N_r}\left(\frac{E[r_i^{\mathrm{ql}}]}{\mathrm{Thr}(T_{\mathrm{pf}_i}) + \lambda_{\mathrm{sl}}{\cdot}E[r_i^{\mathrm{sl}}]}{\cdot}\frac{E[r_i^{\mathrm{sl}}]}{\Sigma_{i=1}^{N_r}E[r_i^{\mathrm{sl}}]}\right) \tag{6}$$

where $r_i^{\mathrm{ql}}$ represents a reward function counting the number of tokens in place $P_q$ of $rack_i$, and $\mathrm{Thr}(T_{\mathrm{pf}_i})$ represents the steady-state throughput of activity $T_{pf}$ of $rack_i$. Moreover, $r_i^{\mathrm{sl}}$ is the reward function that computes the probability of $rack_i$ being selected when a request is dispatched. For the simplified model of *policy 5*, this metric is also computed by the Little's law as Eq. (7):

$$\mathrm{MWT} = \frac{\Sigma_{i=1}^{N_r}\left(\frac{E[r_i^{\mathrm{ql}}]}{\mathrm{Thr}(T_{\mathrm{pf}_i})+\mathrm{Thr}(T_{\mathrm{sl}_i})}\right)}{N_r} \tag{7}$$

where $\mathrm{Thr}(T_{\mathrm{sl}_i})$ represents the steady-state throughput of activity $T_{\mathrm{sl}_i}$.

The **unavailability of PMs (UnAva)** stands for the probability of a PM being in a failed state, which is calculated by Eq. (8):

$$\mathrm{UnAva} = \frac{\Sigma_{i=1}^{N_r}E[r_i^f]}{\Sigma_{i=1}^{N_r}N_{\mathrm{pm}_i}} \tag{8}$$

where $r_i^f$ represents the reward function counting the number of tokens in place $P_f$ of $rack_i$ and $N_{\mathrm{pm}_i}$ is the number of PMs inside $rack_i$.

An important metric is the **total power consumption ($\mathbf{Pow_{total}}$)**, the mean value of power consumed by all servers, ToR switches, cooling systems, and the aggregate switch. The total power consumption is

computed by Eq. (9):

$$\mathrm{Pow}_{\mathrm{total}} = \mathrm{Pow}_{\mathrm{pm}} + \mathrm{Pow}_{\mathrm{tor}} + \mathrm{Pow}_{\mathrm{cool}} + \mathrm{Pow}_{\mathrm{aggr}} \tag{9}$$

where $Pow_{pm}$, $Pow_{tor}$, and $Pow_{cool}$ are the measures described in the following, and $Pow_{aggr}$ is the mean power consumed by the aggregate switch, which is assumed to be a constant value denoted by $p_{aggr}$.

The **power consumption of PMs ($\mathbf{Pow_{pm}}$)** is the mean value of the power consumed by the servers of a data center. Since we assume that powered-off PMs consume a negligible amount of power, this metric is expressed by considering the power consumption of powered-on PMs in the data center as Eq. (10):

$$\begin{aligned}\mathrm{Pow}_{\mathrm{pm}} &= p_{\mathrm{idle}} \cdot (\Sigma_{i=1}^{N_r} N_{\mathrm{pm}_i} - \Sigma_{i=1}^{N_r} E[r_i^{\mathrm{of}}] - \Sigma_{i=1}^{N_r} E[r_i^{f}]) \\ &+ p_{\mathrm{pvm}} \cdot \Sigma_{i=1}^{N_r} E[r_i^{p}] + p_{\mathrm{avm}} \cdot \Sigma_{i=1}^{N_r} E[r_i^{\mathrm{on}}]\end{aligned} \tag{10}$$

where $N_{\mathrm{pm}_i}$ is the number of PMs inside $rack_i$. Reward functions $r_i^{\mathrm{of}}$, $r_i^{f}$, $r_i^{p}$ and $r_i^{\mathrm{on}}$ count the number of tokens in places $P_{off}$, $P_f$, $P_p$, and $P_{on}$ of $rack_i$, respectively. Furthermore, $p_{idle}$, $p_{pvm}$, and $p_{avm}$ represent the power consumptions of an idle PM hosting no VM, a provisioned VM, and an available VM, respectively.

The **power consumption of ToR switches ($\mathbf{Pow_{tor}}$)** stands for the mean value of the power consumed by the top of rack switches and is computed by Eq. (11):

$$\mathrm{Pow}_{\mathrm{tor}} = p_{\mathrm{tor}} \cdot \Sigma_{i=1}^{N_r} E[r_i^{\mathrm{qn}}] \tag{11}$$

where $p_{tor}$ is the power consumed by a top of rack switch when it is on, and $r_i^{\mathrm{qn}}$ is the probability that the queue of $rack_i$ is not empty. This is computed by assigning the reward function of Eq. (12) to the SAN model of Fig. 3:

$$r_i^{\mathrm{qn}} = \begin{cases} 1, & [\#P_{q_i}] > 0 \\ 0, & \text{otherwise} \end{cases} \tag{12}$$

Finally, the **power consumption of cooling systems ($\mathbf{Pow_{cool}}$)** evaluates the mean value of the power consumed by the cooling systems attached to the racks. It can be computed using Eq. (13):

$$\mathrm{Pow}_{\mathrm{cool}} = p_{\mathrm{cool}} \cdot \Sigma_{i=1}^{N_r} (1 - E[r_i^{\mathrm{re}}]) \tag{13}$$

where $p_{cool}$ is the power consumed by a cooling system when it is on, and $r_i^{\mathrm{re}}$ is the probability of $rack_i$ being in the idle state. By idle state, we mean that no PM is powered on inside the rack and no request is waiting in its local queue, implying that the cooling system can be switched to standby mode. Thus, $r_i^{\mathrm{re}}$ can be computed by assigning the reward function of Eq. (14) to the SAN model of Fig. 3:

$$r_i^{\mathrm{re}} = \begin{cases} 1, & (([\#P_{\mathrm{on}_i}] == 0) \text{ and } ([\#P_{p_i}] == 0) \text{ and } ([\#P_{q_i}] == 0)) \\ 0, & \text{otherwise} \end{cases} \tag{14}$$

## 6. Performance evaluation

In this section, the dispatching policies introduced in Section 4.3 are evaluated and compared based on the metrics described in Section 5. The results obtained from the model are also cross-validated against the results obtained with the CloudSim framework. In addition, the proposed model of Fig. 3 is simulated with the Möbius simulation engine for all presented dispatching policies.

### 6.1. Numerical results

Herein, the proposed SANs are applied to model a cloud data center containing three racks to illustrate how the proposed models work. The models are solved with the Möbius tool [29], for a wide range of input parameters. For the sake of brevity, we only report herein a small sub-set of the full set of numerical results obtained. Most of the values considered as input parameters of the proposed model are in line with the other proposals in the related state-of-the-art [8,31,32,35,36,23,47,55,58,59]. These values are presented in Table 4. It is assumed that the types of ToR switches, the aggregate switch, and the rack-based cooling systems are HP5920, HP6600, and HP Modular Cooling System G2, respectively [60]. The rate of the proactive enabling mechanism ($\lambda_{\mathrm{act}_i}$) is tuned based on the probability of each $rack_i$ chosen as a target rack; therefore, it differs between racks and it varies when the request arrival rate changes. Furthermore, for *policy 5*, each $\lambda_{\mathrm{sl}_i}$ is set to 1000 req/h to be almost equivalent to $\lambda_{sl}$ for the other policies.

The numerical solver used to solve the model in the steady-state is an iterative steady-state solver. Successive Over-Relaxation (SOR), which is a variant of the Gauss-Seidel method, is exploited as the iterative method of the numerical solver. SOR converges in fewer iterations and requires less memory than the other supported iterative method, Jacobi. 46 min and 2 GB are the time and memory required to solve the proposed model on a system with an Intel® Core™ i7 @ 2.00 GHz CPU and 4 GB of RAM.

Fig. 4 represents the steady-state probabilities of requests being federated to other clouds ($Pr_{fed}$). Since the worst-fit policies, i.e. *policy 2* and *policy 4*, dispatch the requests almost fairly, they are the least probable ones that may face fullness of local queues in all arrival ranges. On the other hand, the best-fit policies, *policy 1* and *policy 3*, reasonably experience more federation probability. The greatest federation probability results from *policy 5* when arrival rates of requests take larger values. While *policy 5* is not very probable to federate requests when arrival rates are small, its probability grows faster when arrival rates become large. In Fig. 5, the steady-state mean size of local queues (*MQS*) of racks are compared for different dispatching policies. Since idle PMs are not switched off in *policy 5*, they are available for provisioning more quickly in contrast to other policies. Thus, the mean queue sizes for *policy 5* is minimal among all the policies. Putting the *policy 5* aside, *policy 1* and *policy 3* result in the smallest and largest queue sizes, respectively, wherein *policy 2* and *policy 4* are in the middle of the range.

The average waiting time of requests in the local queues (*MWT*) of racks can be observed in Fig. 6. As previously explained, it is reasonable that the smallest waiting time is achieved by *policy 5*. Policies *1* and *2* show very similar behavior, and are much better than *policy 4* and *policy 3*. In fact, the relative behavior of the policies for the mean waiting time metric is similar to the one we have observed for the mean queue sizes in

**Table 4**
Configuration of the IaaS cloud data center under study.

| Parameter | Meaning | Value/range |
|---|---|---|
| $\lambda_{in}$ | Request arrival rate | [1, 20] req/h |
| $\lambda_{sl}$ | Dispatching rate of the CCM | 3000 req/h |
| $\lambda_{fed}$ | Federation rate | 50 req/h |
| $1/\lambda_{p_i}$ | Mean provisioning time of a VM | 2 min |
| $1/\lambda_{s_i}$ | Mean service time of a VM | 20 min |
| $1/\lambda_{\mathrm{ton}_i}$ | Mean switch-on time of a PM | 3 min |
| $1/\lambda_{\mathrm{tof}_i}$ | Mean switch-off time of a PM | 1 min |
| $1/\lambda_{\mathrm{off}_i}$ | MTBF of a powered-off PM | 1000 h |
| $1/\lambda_{\mathrm{onf}_i}$ | MTBF of a powered-on PM containing available VMs | 100 h |
| $1/\lambda_{\mathrm{pf}_i}$ | MTBF of a powered-on PM containing provisioned VMs | 66.7 h |
| $1/\lambda_{r_i}$ | Mean repair time of a PM | 3 h |
| $Q_d$ | Size of global queue | 1 |
| $Q_{r_i}$ | Size of local queue | 2 |
| $Q_f$ | Size of federation queue | 2 |
| $p_{avm}$ | Power consumption of an available VM | 1 W |
| $p_{pvm}$ | Power consumption of a provisioned VM | 15 W |
| $p_{idle}$ | Power consumption of an idle PM | 150 W |
| $p_{tor}$ | Power consumption of a ToR switch | 366 W |
| $p_{aggr}$ | Power consumption of the aggregate switch | 405 W |
| $p_{cool}$ | Power consumption of a cooling system | 950 W |
| $L$ | Multiplexing factor | 1 |
| $N_r$ | Number of racks | 3 |
| $N_{\mathrm{pm}_i}$ | Number of PMs inside each rack | 3 |

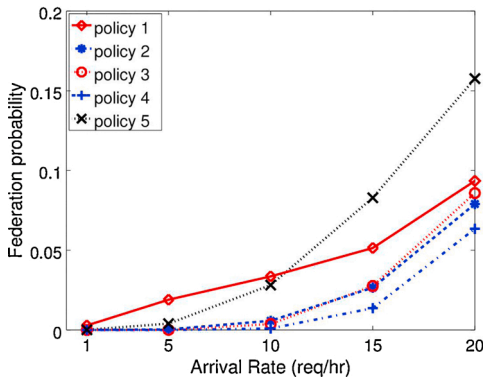



**Fig. 4.** The steady-state federation probability ($Pr_{fed}$) for different dispatching policies.

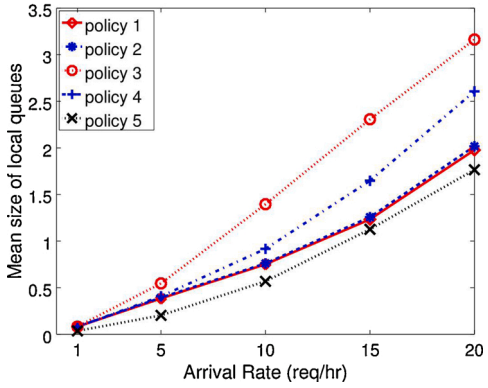



**Fig. 5.** The mean queue size (*MQS*) for different dispatching policies.

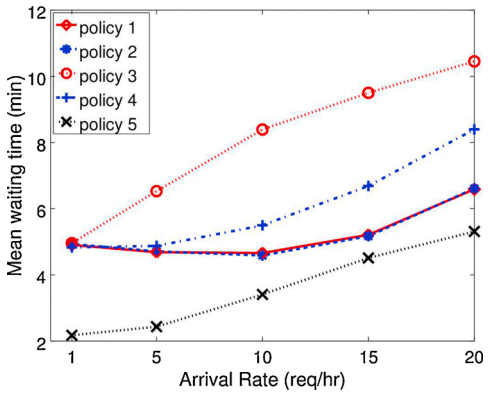



**Fig. 6.** The mean waiting time (*MWT*) for different dispatching policies.

Fig. 5. Fig. 7 shows the steady-state unavailability of PMs (*UnAva*) when different dispatching policies are applied. As the arrival rate increases, the probability of a PM being in the powered-on state as well as the probability of a VM to be in the provisioned state increase for all dispatching policies. As stated in Section 3, powered-on PMs are more probable to fail compared to the powered-off ones. Moreover, the failure rate of a powered-on PM increases as the number of VMs provisioned on top of the PM increases. This is the trend of all curves in Fig. 7. For *policy 5*, powered-on PMs cannot be switched off. Hence, this policy shows a higher percentage of failed PMs. We can observe that the other dispatching policies behave nearly in the same manner.

Fig. 8 shows the steady-state power consumption of the main components of our case study, when different dispatching policies are applied. In Fig. 8(a), the power consumption of PMs is represented. As

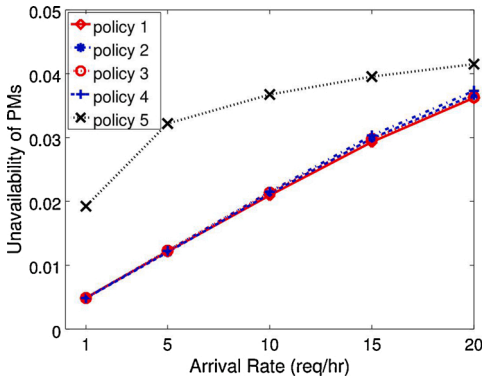



**Fig. 7.** The steady-state unavailability (*UnAva*) of PMs for different dispatching policies.

the arrival rate increases, the proportion of powered-off PMs to powered-on PMs decreases, and the power consumption increases, as shown in Fig. 8(a). The power consumption of PMs observed for *policy 5* is oversize since it never moves powered-on servers to the powered-off state. The other plots are relatively close to each other. As shown in Fig. 8(b), the power consumption of ToR switches for *policy 5* is a constant value, much larger than those for the other policies. This is the result of disabling sleep or standby modes of ToR switches in *policy 5*. In contrast, these switches consume much less power with other policies since they go to sleep mode whenever there is no request in their local queues. When the arrival rate gradually increases, ToR switches find less time to sleep, and therefore their power consumption grows. As it can be seen in Fig. 8(b), both *policy 1* and *policy 2* outperform *policy 3* and *policy 4*, since the mean probabilities of local queues being empty with *policy 1* and *policy 2* are larger than those for the other policies.

Fig. 8(c) represents the power consumed by the cooling systems. Again, the power consumption obtained for *policy 5* is constant and much larger than that for other policies, because cooling systems in *policy 5* are assumed to be always on, independent of the workload of the racks. Cooling systems consume less power in other policies because they sleep whenever their corresponding racks are in idle state, containing neither waiting requests nor powered-on PMs. When the arrival rate increases, racks are less probable to be idle, and consequently the power consumption of their cooling systems grows. In Fig. 8(d), the total power consumption of PMs, ToR switches, cooling systems, and the aggregate switch of the cloud data center are compared. The aggregate switch is assumed to be always on and ready to process requests submitted to the cloud data center with minimum delay. As expected, *policy 5* consumes always the most power. In the whole range of the request arrival rate, *policy 1* and *policy 3* show less power consumption than *policy 2* and *policy 4*, where best-fit policies noticeably outperform worst-fit ones. When the arrival rate increases, it is expected that the total power consumption of all policies gradually converge.

From the results reported herein, it can be concluded that all energy-aware policies produce a significant improvement on the power consumption metrics in contrast to the baseline policy, *policy 5*. They also improve the unavailability of PMs and the federation probability in comparison with *policy 5*. However, besides these improvements, user-oriented performance metrics, e.g. the mean waiting time and the size of local queues, degrade. Amongst the dispatching policies considered here, no one is the winner for all figures of merit: *policy 3* outperforms the others with respect to the power consumption of cooling systems, *policy 4* acts as the best one with respect to the federation probability, and *policy 1* outperforms the others with respect to the power consumption of ToR switches. Both *policy 1* and *policy 2* are the best in what concerns the mean waiting time and the mean local queue size, while *policy 1* and *policy 3* stand out regarding the total power consumption.

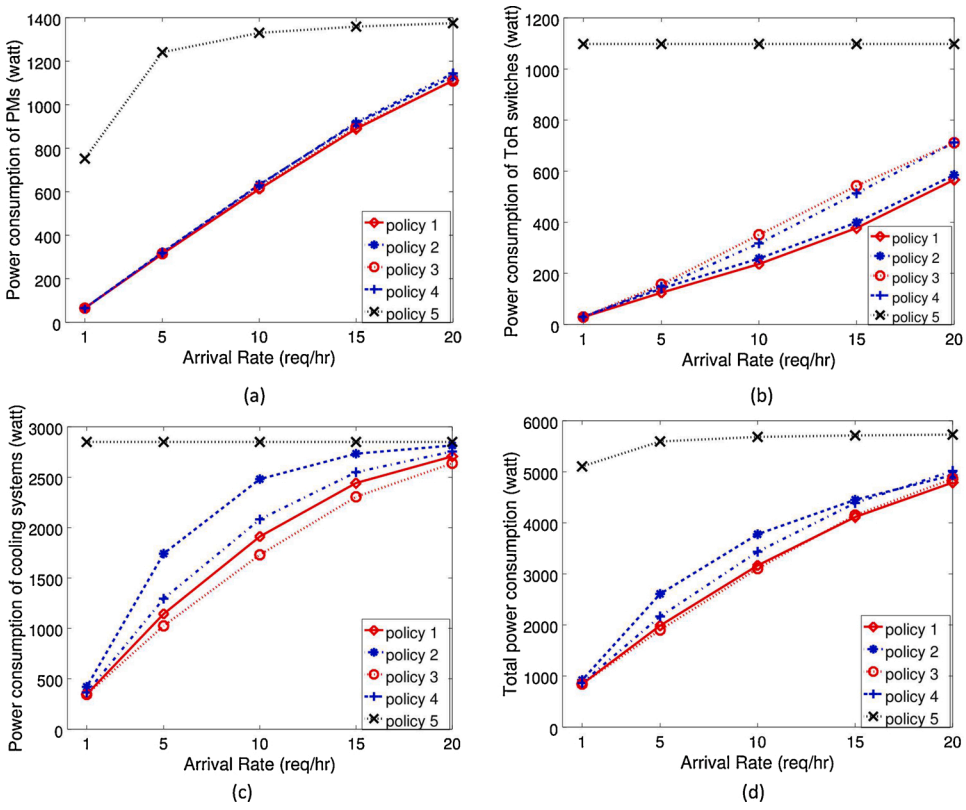



**Fig. 8.** The steady-state power consumptions of (a) PMs ($Pow_{pm}$), (b) ToR switches ($Pow_{tor}$), (c) cooling systems ($Pow_{cool}$), and (d) the entire data center ($Pow_{total}$) obtained with different dispatching policies.

### 6.2. Model validation

In order to demonstrate the accuracy of the proposed approach, results obtained by analytically solving the SAN model are compared against the results obtained from simulating the same system using the CloudSim framework [30]. The CloudSim framework is a Java-based API to support modeling and simulation of cloud infrastructures and services. This framework encompasses well-defined components in the context of real cloud systems (e.g. Datacenter, Host, VM, Cloudlet, etc.) and takes many details of such systems into account (e.g. VM scheduling, cloudlet scheduling, etc.). CloudSim is a comprehensive platform for modeling data centers, service brokers and resource management policies, which provides facilities for the simulation of federated cloud environments. Moreover, the features provided by the framework can be easily extended for modeling custom cloud computing systems. In order to use CloudSim for cross-validation, the framework has been extended and adapted to the requirements of the presented models. To this end, we extend CloudSim with respect to the functionalities already provided with the framework by defining: (i) global, local, and federation queues of the data center, (ii) switching on/off events of PMs, (iii) failure/repair events of PMs, (iv) provisioning events of VMs, and (v) request arrival events. To apply these modifications, some existing java classes in CloudSim package were extended, including *DatacenterBroker*, *Datacenter*, *Host*, *Cloudlet*, and *Predicate* classes.

In the simulation with the CloudSim, we adopt the method of independent replications wherein the number of replicas is 30 and the size of each replica is one million hours. Using the extended framework, a simulation environment is created. It consists of three cloud-based data centers managed by a broker. Each data center in the simulation environment acts as a rack in our proposed SAN model. On the other hand, the broker is responsible for choosing a suitable data center to handle an incoming request based on the introduced dispatching policies. Each data center includes three homogeneous compute hosts, each one has a CPU core with a processing capability of 1000 Million Instructions Per Second (MIPS), 2 GB of RAM, and 10 Gbps of network bandwidth. Each machine can allocate and host a VM for an incoming request at a given instant of time. Creating and provisioning times for each VM, like all other events used for simulating the sample data center by the CloudSim framework, follow an exponential distribution. The incoming requests are also submitted at an inter-arrival delay following an exponential distribution. An application corresponding to a VM request is characterized in terms of its computational requirements in MIPS, and its instruction length is assigned based on the required service time following an exponential distribution with the defined rate. The Storage Area Network is used to store and retrieve chunks of data which is common in cloud data centers. Since networking is not the focus of this study, a minimal data transfer of 300 KB overhead for each application is considered; therefore, the transferring delay is negligible despite the available bandwidth. All tests have been executed on an Intel® Core™ i5 @ 2.53 GHz system with 4 GB RAM on which each replica takes around 20 *minutes* to be executed.

For the sake of brevity, we only report the validation results for two measures, the mean waiting time (*MWT*) and the total power consumption ($Pow_{total}$). Other measures show almost the same behavior when the analytical results are compared to the simulation. Tables 5 and 6 compare the steady-state mean waiting times and total power consumptions obtained by the proposed SAN model and the CloudSim framework, respectively. The percent error (PE) indicates the difference between an average simulated value and a known or exact one, computed by the analytical model, as a percentage of the known value. Each cell of the simulation part of Tables 5 and 6 is the average of the corresponding values computed from independent replicas, for which the standard deviation (SD) is also computed. Simulation results are reported with a confidence level of 0.95. The maximum errors for the mean waiting time and power consumption are 7.1% and 1.5%, respectively. As it can be concluded from Tables 5 and 6, the results of

**Table 5**
Comparison of the mean waiting times obtained with the proposed analytic model and the CloudSim framework (min).

| Policy | Analytic model | | | | | CloudSim | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Request arrival rate ($\lambda_{in}$) | | | | | Request arrival rate ($\lambda_{in}$) | | | | | | | | | | | | | | |
| | 1 | 5 | 10 | 15 | 20 | 1 | PE | SD | 5 | PE | SD | 10 | PE | SD | 15 | PE | SD | 20 | PE | SD |
| ***policy 1*** | 4.90 | 4.69 | 4.66 | 5.21 | 6.59 | 5.10 | 0.6 | 4.1 | 4.98 | 1.2 | 6.2 | 4.99 | 1.0 | 7.1 | 5.54 | 1.3 | 6.3 | 6.84 | 0.9 | 3.8 |
| ***policy 2*** | 4.93 | 4.71 | 4.59 | 5.17 | 6.61 | 5.11 | 0.5 | 3.7 | 4.89 | 0.6 | 3.8 | 4.83 | 0.7 | 5.2 | 5.44 | 0.8 | 5.2 | 6.83 | 0.7 | 3.3 |
| ***policy 3*** | 4.96 | 6.53 | 8.39 | 9.50 | 10.45 | 5.16 | 0.6 | 4.0 | 6.84 | 0.9 | 4.7 | 8.72 | 1.1 | 3.9 | 9.86 | 1.4 | 3.8 | 10.83 | 1.3 | 3.6 |
| ***policy 4*** | 4.83 | 4.88 | 5.50 | 6.69 | 8.40 | 5.00 | 0.5 | 3.5 | 5.03 | 0.5 | 3.1 | 5.66 | 0.5 | 2.9 | 6.90 | 0.7 | 3.1 | 8.66 | 0.8 | 3.1 |
| ***policy 5*** | 2.18 | 2.44 | 3.41 | 4.51 | 5.31 | 2.21 | 0.1 | 1.4 | 2.53 | 0.3 | 3.7 | 3.53 | 0.4 | 3.5 | 4.62 | 0.4 | 2.4 | 5.40 | 0.3 | 1.7 |

both the model and the simulator are very close to each other, for all dispatching policies introduced in this paper, which validates the proposed models.

### 6.3. Extended simulations

Although analytical methods provide different means to precisely solve the proposed models, these methods require that the model meet a strict set of requirements, which often limits their application to a restricted subset of models. To be able to solve the proposed models for more realistic parameter values, we exploit the Möbius simulation engine. The Möbius tool provides a simulation engine which is unique in its formalism-independent design and flexibility. To do so, we use the steady-state simulation applying the batch means method with a confidence level of 0.95. The minimum and maximum number of batches are set to 1000 and 10,000, respectively. When the error given by the ratio between the size of the confidence interval and the mean value for each reward variable falls below 10% the simulation is terminated. The execution of each set of batches took around 35 min on average on an Intel® Core™ i7 @ 2.00 GHz system with 4GB RAM.

In this set of experiments, the request arrival rate ($\lambda_{in}$) is set to [1000, 5000] req/h. The size of global ($Q_d$), local ($Q_{r_i}$), and federation ($Q_f$) queues are set to 100, 15, and 25, respectively. Moreover, the multiplexing factor ($L$), the number of racks ($N_r$) and the number of PMs inside each rack ($N_{\text{pm}_i}$) are set to 4, 20, and 30, respectively. The values of the other input parameters for simulation of the proposed model of Fig. 3 are the same as those reported in Table 4. Based on this configuration, which represents a cloud system in the scale of modular data centers [61][62], the proposed policies are compared in terms of total power consumption ($Pow_{total}$). The results are shown in Fig. 9. Like the results obtained from analytical modeling approach presented in Fig. 8 (d), best-fit policies outperform worst-fit ones in terms of total power consumption all over the analyzed range, but herein, *policy 1* performs better than *policy 3*, especially when the arrival rate grows. Moreover, *policy 5* consumes the most power in the whole range of the request arrival rate. The curve of all the policies becomes almost plane, when the request arrival rate becomes higher than 3000 req/h. It shows that system saturates when the arrival rate of requests exceeds 3000 per hour.

## 7. Conclusions and future work

In this work, we presented a SAN model for an autonomous group of PMs in an IaaS cloud data center, called a rack. The model encompasses several aspects of a real computing environment, including different PM power consumption states, different strategies to move PMs between these states, VM provisioning and serving steps, VM multiplexing, and failure/repair behavior of PMs. This base model was then used in a hierarchical way to model a whole cloud data center composed of several racks. Using this unified SAN model, we are able to examine different energy-aware and structure-aware dispatching policies. In order to compare different dispatching policies under evaluation in this paper, several performance, availability, and power consumption metrics were introduced on the proposed SAN. Among the output metrics introduced, the power consumed by cloud system was partitioned into several fine-grained metrics. It allows cloud providers to realize the effect of those metrics on the total power consumption when different resource management policies are applied. Although, in most situations, the total power consumption is the metric of interest, this decomposition can be especially helpful for providers to give different power cost weights to different components.

While the adopted dispatching policies behaved quite similarly for certain metrics, e.g. the power consumption and unavailability of PMs, the behavior is quite distinct for other metrics, such as the federation probability, the mean size of local queues, the mean waiting time and the power consumption of ToR switches and cooling systems. Results showed that none of the introduced policies gives the best results with regard to all performance measures or workloads. The analysis preformed in this paper for different dispatching policies makes it possible, in practice, to choose the suitable policy depending on the preferred SLA metrics, or even switch between them when the characteristics of the workload change. It can be also concluded that the total power consumption of a data center can be significantly reduced by switching idle PMs, ToR switches, and cooling systems to the standby mode. Furthermore, switching to standby mode can notably improve the availability of servers, and in most of cases, the federation probability.

One of the limitations of the model proposed in this paper is the scalability. Although we have mitigated this issue by simulating a large system using simulation engine of the Möbius tool, well-known approximation methods can still be used to improve the scalability of the model. Decomposition technique, at both system and model levels, can be applied to improve a SAN model to handle more complex situations. As an interesting extension to the model proposed in this paper, one can apply the folding method or fixed-point iteration technique to decompose the proposed monolithic model in order to reduce the number of states in the state space. Moreover, hierarchical modeling, as one of the system-level decomposition techniques, can be used to reduce the state space. The SAN model proposed in this paper only considers homogeneous PMs. Therefore, one interesting extension to this work is to support heterogeneous PMs. By considering heterogeneous PMs, one can define different classes of servers each one having its own failure and repair rates, processing speed, memory and storage capacities, and power consumption. Colored extensions of PNs and SANs could be used for such an extension. Colored versions of PNs and SANs can also be used to detect SLA violations in clouds. If different colors of tokens are used inside places, representing different requests and machines, it would be possible to determine percentile service level agreements.

Considering more sophisticated VM placement strategies and taking into account VM migrations that highly influence power consumption and performance of cloud data centers, one can propose a more comprehensive analytical model. In such a model, algorithms of live migration and mechanisms to select candidate VMs/PMs can be modeled and evaluated. In the architecture investigated in this work, two layers of network switches, access and aggregate layers, have been considered in a tree structure. To be able to increase the number of network layers, or to support more complex architectures, e.g. Fat-Tree and DCell, the proposed model can be divided into sub-models, and appropriate techniques can be applied to solve the problem. In addition

**Table 6**
Comparison of the total power consumption obtained with the proposed analytic model and the CloudSim framework (W).

| Policy | Analytic model | | | | CloudSim | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Request arrival rate ($\lambda_{in}$) | | | | Request arrival rate ($\lambda_{in}$) | | | | | | | | | | | |
| | 1 | 5 | 10 | 15 | 20 | 1 | PE | SD | 5 | PE | SD | 10 | PE | SD | 15 | PE | SD | 20 | PE | SD |
| ***policy 1*** | 844 | 1987 | 3165 | 4114 | 4789 | 845 | 3.4 | 0.1 | 1989 | 6.2 | 0.1 | 3167 | 6.8 | 0.1 | 4106 | 23.5 | 0.2 | 4777 | 33.8 | 0.3 |
| ***policy 2*** | 921 | 2608 | 3777 | 4452 | 4935 | 919 | 5.8 | 0.2 | 2598 | 30.7 | 0.4 | 3781 | 11.4 | 0.1 | 4449 | 9.1 | 0.1 | 4923 | 35.9 | 0.2 |
| ***policy 3*** | 843 | 1902 | 3105 | 4150 | 4864 | 844 | 2.8 | 0.1 | 1931 | 83.2 | 1.5 | 3152 | 133.7 | 1.5 | 4188 | 108.1 | 0.9 | 4894 | 84.3 | 0.6 |
| ***policy 4*** | 863 | 2167 | 3434 | 4388 | 5017 | 864 | 3.1 | 0.1 | 2171 | 12.5 | 0.2 | 3435 | 4.4 | 0.0 | 4393 | 15.4 | 0.1 | 5021 | 12.4 | 0.1 |
| ***policy 5*** | 5105 | 5594 | 5683 | 5712 | 5728 | 5105 | 1.1 | 0.0 | 5594 | 0.3 | 0.0 | 5680 | 8.8 | 0.1 | 5712 | 2.8- | 0.0 | 5728 | 3.5 | 0.0 |

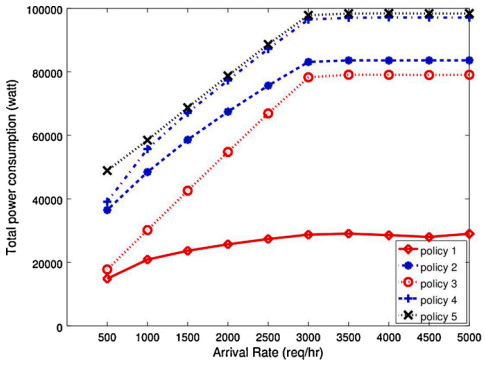



**Fig. 9.** The power consumption of a real sized sample data center for different dispatching policies; results were obtained with the Möbius simulation engine.

to the directions mentioned above, one can investigate scheduling policies at LM components of a data center. In this way, the LM will be able to decide, based on the optimization goals, on what PM the requesting VM should be deployed. For example, it may choose the hosting PM to maximize the number of powered-off PMs and to reduce the corresponding power consumption of PMs in the rack. Dynamic scheduling algorithms can also be modeled by capturing VM migration and data center consolidation aspects, which play important roles in energy saving policies.

## Conflict of interest

None declared.

## Declaration of Competing Interest

The authors report no declarations of interest.

## References

[1] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, A view of cloud computing, Commun. ACM 53 (4) (2010) 50–58.
[2] R. Buyya, C.S. Yeo, S. Venugopal, J. Broberg, I. Brandic, Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th utility, Fut. Gen. Comput. Syst. 25 (6) (2009) 599–616.
[3] F. Machida, D.S. Kim, K.S. Trivedi, Modeling and analysis of software rejuvenation in a server virtualized system with live VM migration, Perform. Eval. 70 (3) (2013) 212–230.
[4] Y. Han, J. Chan, T. Alpcan, C. Leckie, Using virtual machine allocation policies to defend against co-resident attacks in cloud computing, IEEE Trans. Depend. Secure Comput. 14 (1) (2017) 95–108.
[5] A. Kansal, F. Zhao, J. Liu, N. Kothari, A.A. Bhattacharya, Virtual machine power metering and provisioning, in: The 1st ACM symposium on Cloud computing, Indianapolis, IN, 2010, pp. 39–50.
[6] A.M. Sampaio, J.G. Barbosa, R. Prodan, PIASA: A power and interference aware resource management strategy for heterogeneous workloads in cloud data centers, Simul. Modell. Pract. Theory 57 (2015) 142–160.
[7] M. Gaggero, L. Caviglione, Model predictive control for energy-efficient, quality-aware, and secure virtual machine placement, IEEE Trans. Autom. Sci. Eng. 16 (1) (2019) 420–432.
[8] D. Bruneo, A stochastic model to investigate data center performance and QoS in IaaS cloud computing systems, IEEE Trans. Parallel Distrib. Syst. 25 (3) (2014) 560–569.
[9] K. Bilal, S.U.R. Malik, S.U. Khan, A.Y. Zomaya, Trends and challenges in cloud datacenters, IEEE Cloud Comput. 1 (1) (2014) 10–20.
[10] Y. Chen, L. Wang, X. Chen, R. Ranjan, A.Y. Zomaya, Y. Zhou, S. Hu, Stochastic workload scheduling for uncoordinated datacenter clouds with multiple QoS constraints, IEEE Trans. Cloud Comput. PP (99) (2016).
[11] Y. Sharma, B. Javadi, W. Si, On the reliability and energy efficiency in cloud computing, in: The 13th Australasian Symposium on Parallel and Distributed Computing, Sydney, Australia, 2015, pp. 111–114.
[12] A.M. Sampaio, J.G. Barbosa, A comparative cost analysis of fault-tolerance mechanisms for availability on the cloud, Sustain. Comput.: Informatics Syst. 19 (2018) 315–323.
[13] C. Jiang, Y. Wang, D. Ou, Y. Li, J. Zhang, J. Wan, B. Luo, W. Shi, Energy efficiency comparison of hypervisors, Sustain. Comput.: Informatics Syst. 22 (2019) 311–321.

[14] C. Nadjahi, H. Louahlia, S. Lemasson, A review of thermal management and innovative cooling strategies for data center, Sustain. Comput.: Informatics Syst. 19 (2018) 14–28.
[15] M. Koot, F. Wijnhoven, Usage impact on data center electricity needs: a system dynamic forecasting model, Appl. Energy 291 (2021) 116798.
[16] Y. Sharma, B. Javadi, W. Si, D. Sun, Reliability and energy efficiency in cloud computing systems: survey and taxonomy, J. Netw. Comput. Appl. 74 (2016) 66–85.
[17] AWS and Sustainability. https://aws.amazon.com/about-aws/sustainability/, (Accessed August 2021).
[18] R. Entezari-Maleki, L. Sousa, A. Movaghar, Performance and power modeling and evaluation of virtualized servers in IaaS clouds, Inform. Sci. 394–395 (2017) 106–122.
[19] Y. Xia, M. Zhou, X. Luo, S. Pang, Q. Zhu, A stochastic approach to analysis of energy-aware DVS-enabled cloud datacenters, IEEE Trans. Syst. Man Cybern.: Syst. 45 (1) (2015) 73–83.
[20] J. Zhang, X. Wang, H. Huang, S. Chen, Clustering based virtual machines placement in distributed cloud computing, Fut. Gen. Comput. Syst. 66 (1) (2017) 1–10.
[21] C.-S. Li, H. Franke, C. Parris, B. Abali, M. Kesavan, V. Chang, Composable architecture for rack scale big data computing, Fut. Gen. Comput. Syst. 67 (2017) 180–193.
[22] R. Entezari-Maleki, K.S. Trivedi, A. Movaghar, Performability evaluation of grid environments using stochastic reward nets, IEEE Trans. Depend. Secure Comput. 12 (2) (2015) 204–216.
[23] E. Ataie, R. Entezari-Maleki, L. Rashidi, K.S. Trivedi, D. Ardagna, A. Movaghar, Hierarchical stochastic models for performance, availability, and power consumption analysis of IaaS clouds, IEEE Trans. Cloud Comput. 7 (4) (2019) 1039–1056.
[24] Z. Ding, H. Qiu, R. Yang, C. Jiang, M. Zhou, Interactive-control-model for human-computer interactive system based on petri nets, IEEE Trans. Autom. Sci. Eng. 16 (4) (2019) 1800–1813.
[25] T. Tapia-Flores, E. López-Mellado, A.P. Estrada-Vargas, J.-J. Lesage, Discovering petri net models of discrete-event processes by computing T-invariants, IEEE Trans. Autom. Sci. Eng. 15 (3) (2018) 992–1003.
[26] A. Movaghar, J.F. Meyer, Performability modeling with stochastic activity networks, in: The 1984 Real-Time Systems Symposium, Austin, TX, USA, 1984, pp. 215–224.
[27] J.F. Meyer, A. Movaghar, W.H. Sanders, Stochastic activity networks: structure, behaviour, and application, in: The International Workshop on Timed Petri Nets, Torino, Italy, 1985, pp. 106–115.
[28] W.H. Sanders, J.F. Meyer, Stochastic activity networks: Formal definitions and concepts, in: E. Brinksma, H. Hermanns, J.-P. Katoen (Eds.), Lectures on Formal Methods and Performance Analysis, vol. 2090 of Lecture Notes in Computer Science (LNCS), Springer, 2001, pp. 315–343.
[29] D. Daly, D.D. Deavours, J.M. Doyle, P.G. Webster, W.H. Sanders, Möbius: an extensible tool for performance and dependability modeling, in: B.R. Haverkort, H. C. Bohnenkamp, C.U. Smith (Eds.), Computer Performance Evaluation: Modelling Techniques and Tools, vol. 1786 of Lecture Notes in Computer Science (LNCS), Springer, 2000, pp. 332–336.
[30] R.N. Calheiros, R. Ranjan, A. Beloglazov, C.A.F.D. Rose, R. Buyya, CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms, Software: Pract. Experience 41 (1) (2011) 23–50.
[31] R. Ghosh, K.S. Trivedi, V.K. Naik, D.S. Kim, End-to-end performability analysis for Infrastructure-as-a-Service cloud: an interacting stochastic models approach, in: The IEEE 16th Pacific Rim International Symposium on Dependable Computing, Tokyo, Japan, 2010, pp. 125–132.
[32] R. Ghosh, V.K. Naik, K.S. Trivedi, Power-performance trade-offs in IaaS cloud: a scalable analytic approach, The IEEE/IFIP 41st International Conference on Dependable Systems and Networks Workshops, Hong Kong (2011) 152–157.
[33] R. Ghosh, F. Longo, V.K. Naik, K.S. Trivedi, Modeling and performance analysis of large scale IaaS clouds, Fut. Gen. Comput. Syst. 29 (5) (2013) 1216–1234.
[34] B. Liu, X. Chang, Z. Han, K. Trivedi, R.J. Rodríguez, Model-based sensitivity analysis of IaaS cloud availability, Fut. Gen. Comput. Syst. 83 (2018) 1–13.
[35] R. Ghosh, F. Longo, F. Frattini, S. Russo, K.S. Trivedi, Scalable analytics for IaaS cloud availability, IEEE Trans. Cloud Comput. 2 (1) (2014) 57–70.
[36] H. Khazaei, J. Misic, V.B. Misic, A fine-grained performance model of cloud computing centers, IEEE Trans. Parallel Distrib. Syst. 24 (11) (2013) 2138–2147.
[37] H. Khazaei, J. Misic, V.B. Misic, Performance of an IaaS cloud with live migration of virtual machines, in: The IEEE Global Telecommunications Conference, Atlanta, GA, 2013, pp. 2289–2293.
[38] A.N. Asadi, M.A. Azgomi, R. Entezari-Maleki, Analytical evaluation of resource allocation algorithms and process migration methods in virtualized systems, Sustain. Comput.: Informatics Syst. 25 (2020) 100370.
[39] H. Khazaei, J. Misic, V.B. Misic, Performance analysis of cloud computing centers using M/G/m/m+r queuing systems, IEEE Trans. Parallel Distrib. Syst. 23 (5) (2012) 936–943.
[40] X. Qiu, Y. Dai, Y. Xiang, L. Xing, A hierarchical correlation model for evaluating reliability, performance, and power consumption of a cloud service, IEEE Trans. Syst. Man Cybern.: Syst. 46 (3) (2016) 401–412.
[41] E. Sousa, F. Lins, E. Tavares, P. Cunha, P. Maciel, A modeling approach for cloud infrastructure planning considering dependability and cost requirements, IEEE Trans. Syst. Man. Cybernet.: Syst. 45 (4) (2015) 549–558.
[42] R. Entezari-Maleki, A. Movaghar, Availability modeling of grid computing environments using SANs, in: The 19th International Conference on Software, Telecommunications and Computer Networks, Split, Croatia, 2011, pp. 1–6.
[43] S. Sebastio, R. Ghosh, T. Mukherjee, An availability analysis approach for deployment configurations of containers, IEEE Trans. Serv. Comput. 14 (1) (2021) 16–29.
[44] B. Addis, D. Ardagna, B. Panicucci, M.S. Squillante, L. Zhang, A hierarchical approach for the resource management of very large cloud platforms, IEEE Trans. Depend. Sec. Comput. 10 (5) (2013) 253–272.
[45] M. Sedaghat, F. Hernandez-Rodriguez, E. Elmroth, Decentralized cloud datacenter reconsolidation through emergent and topology-aware behavior, Fut. Gen. Comput. Syst. 56 (2016) 51–63.
[46] I. Hwang, M. Pedram, Hierarchical virtual machine consolidation in a cloud computing system, in: The IEEE 6th International Conference on Cloud Computing, Santa Clara, CA, 2013, pp. 196–203.
[47] D. Bruneo, A. Lhoas, F. Longo, A. Puliafito, Modeling and evaluation of energy policies in green clouds, IEEE Trans. Parallel Distrib. Syst. 26 (11) (2015) 3052–3065.
[48] Y. Xia, M. Zhou, X. Luo, S. Pang, Q. Zhu, Stochastic modeling and performance analysis of migration-enabled and error-prone clouds, IEEE Trans. Ind. Informatics 11 (1) (2015) 495–504.
[49] D. Bruneo, F. Longo, A. Puliafito, Evaluating energy consumption in a cloud infrastructure, in: The IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks, Lucca, Italy, 2011, pp. 1–6.
[50] D. Bruneo, A. Lhoas, F. Longo, A. Puliafito, Analytical evaluation of resource allocation policies in green IaaS clouds, in: The 3rd International Conference on Cloud and Green Computing, Karlsruhe, Germany, 2013, pp. 84–91.
[51] K. Dunlap, N. Rasmussen, Choosing Between Room, Row, and Rack-Based Cooling for Data Centers, APC White paper 130, Rev. 2.
[52] M. Sedaghat, E. Wadbro, J. Wilkes, S.D. Luna, O. Seleznjev, E. Elmroth, DieHard: reliable scheduling to survive correlated failures in cloud data centers, in: The 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, Cartagena, Colombia, 2016, pp. 52–59.
[53] Y. Xiang, V. Aggarwal, Y.-F. Chen, T. Lan, Differentiated latency in data center networks with erasure coded files through traffic engineering, IEEE Trans. Cloud Comput. (2016).
[54] H. Shen, Z. Li, New bandwidth sharing and pricing policies to achieve a win-win situation for cloud provider and tenants, IEEE Trans. Parallel Distrib. Syst. 27 (9) (2016) 2682–2697.
[55] E. Ataie, R. Entezari-Maleki, S.E. Etesami, B. Egger, D. Ardagna, A. Movaghar, Power-aware performance analysis of self-adaptive resource management in IaaS clouds, Fut. Gen. Comput. Syst. 86 (2018) 134–144.
[56] G. Ciardo, A. Blakemore, P.F. Chimento, J.K. Muppala, K.S. Trivedi, Automated generation and analysis of markov reward models using stochastic reward nets, in: C.D. Meyer, R.J. Plemmons (Eds.), Linear Algebra, Markov Chains, and Queueing Models, vol. 48 of The IMA Volumes in Mathematics and its Application, Springer, 1993, pp. 145–191.
[57] K.S. Trivedi, Probability and Statistics With Reliability, Queuing and Computer Science Applications, 2nd ed., John Wiley and Sons, 2002.
[58] S. Fakhrolmobasheri, E. Ataie, A. Movaghar, Modeling and evaluation of power-aware software rejuvenation in cloud systems, Algorithms 11 (10) (2018) 1–15.
[59] D. Meisner, B.T. Gold, T.F. Wenisch, PowerNap: Eliminating server idle power, in: The 14th International Conference on Architectural Support for Programming Languages and Operating Systems, Washington, DC, 2009, pp. 205–216.
[60] S. Esfandiarpoor, A. Pahlavan, M. Goudarzi, Structure-aware online virtual machine consolidation for datacenter energy improvement in cloud computing, Comput. Electr. Eng. 42 (2015) 74–89.
[61] Sun Modular Datacenter S20/D20 Overview. https://docs.oracle.com/cd/E19115-01/mod.dc.s20/820-5770-10/820-5770-10.pdf (Accessed August 2021).
[62] IBM Portable Modular Data Center Overview and Case Studies. https://www.yumpu.com/en/document/view/30176243/ibm-portable-modular-data-center-overview-and-case-studies (Accessed August 2021).