





Article

Bouncer: A Resource-Aware Admission Control Scheme for Cloud Services

Aaqif Afzaal Abbasi ¹, Mohammed A. A. Al-qaness ², Mohamed Abd Elaziz ³,
Hassan A. Khalil ⁴ and Sunghwan Kim ^{5,*}

¹ Department of Software Engineering, Foundation University, Islamabad 46000, Pakistan

² School of Computer Science, Wuhan University, Wuhan 430072, China

³ Department of Mathematics, Faculty of Science, Zagazig University, Zagazig 44519, Egypt

⁴ Computer Department, Deanship of Preparatory Year and Supporting Studies, Imam Abdulrahman Bin Faisal University, Dammam 31441, Saudi Arabia

⁵ School of Electrical Engineering, University of Ulsan, Ulsan 44610, Korea

* Correspondence: sungkim@ulsan.ac.kr; Tel.: +82-52-259-1401

Received: 14 August 2019; Accepted: 22 August 2019; Published: 24 August 2019



Abstract: Cloud computing is a paradigm that ensures the flexible, convenient and on-demand provisioning of a shared pool of configurable network and computing resources. Its services can be offered by either private or public infrastructures, depending on who owns the operational infrastructure. Much research has been conducted to improve a cloud's resource provisioning techniques. Unfortunately, sometimes an abrupt increase in the demand for cloud services results in resource shortages affecting both providers and consumers. This uncertainty of resource demands by users can lead to catastrophic failures of cloud systems, thus reducing the number of accepted service requests. In this paper, we present Bouncer—a workload admission control scheme for cloud services. Bouncer works by ensuring that cloud services do not exceed the cloud infrastructure's threshold capacity. By adopting an application-aware approach, we implemented Bouncer on software-defined network (SDN) infrastructure. Furthermore, we conduct an extensive study to evaluate our framework's performance. Our evaluation shows that Bouncer significantly outperforms the conventional service admission control schemes, which are still state of the art.

Keywords: admission control; cloud architecture; cloud computing; cloud services; distributed systems; resource-awareness; SDN; services computing; virtualization

1. Introduction

Cloud computing has the potential to transform a large part of the IT industry and to make the software even more attractive as a service as well as to shape the way IT hardware is designed, utilized, and purchased [1]. The success of any cloud management software critically depends on the flexibility, scale, and efficiency with which it can utilize the underlying hardware resources while providing necessary performance isolation [2]. Therefore, resource management at the cloud-scale requires the management platform to provide a rich set of resource controls that can balance the quality of service (QoS) of tenants with the overall resource efficiencies of the datacenters. In this regard, a comprehensive overview of high-performance computing applications in virtualized environments is presented in [3].

Although virtualization technologies provide key infrastructure for the consolidation of many distributed web services into large data centers, there is a major issue in the provisioning and deployment of resources to virtual machines (VMs) in virtualized environments, namely admission control of workloads and service requests. Unfortunately, most of the presently available control

methods do not provide a unified approach to address these concerns [4]. This has led to a variety of demands and time-dependent constraints. Inevitably, data center administrators try to either copy a complete environment in a test laboratory or mock-up responsive systems by coding their own versions for test purposes. This has become a costly and time-consuming endeavor.

More importantly, most of the traditional modeling approaches rely on queuing theory and its related methods [5]. Queuing-based control methods are most suitable for steady-state systems, where the system properties do not change with time, but these are not fit for dynamic systems, such as in datacenter traffic engineering. Based on the recent characterization of virtualization environments, many proposals have been presented to address these challenges. Some services delivery control theories are presented in [6–8] but these do not consider a system's constraints; while, on the other hand, the studies presented in [9] and admission-control techniques involving static resource allocation reject a large proportion of requests in order to avoid overloading. Admission-control literature, as presented in [10] also does not guarantee system stability or robustness.

Cloud services are offered by a cloud computing system hosted on top of a data center infrastructure. On the other hand, Virtualization is the enabling technology behind cloud computing. When it comes to cloud services, the benchmark criteria to interpret a cloud service is a service-level agreement [11]. As the service-level agreement is a legal contract defining the terms of services, our proposed scheme strives to achieve excellence in terms of achieving the required resources from the data center (cloud infrastructure). Virtualization is an important aspect of cloud computing as it is the enabling technology behind the cloud computing paradigm. Bouncer is a policy-based approach which was implemented on a virtualized environment managed by software-defined hardware infrastructure to implement a pre-defined cloud service policy in a controlled environment. The policy is later evaluated against three conventional SLA approaches for its effectiveness.

Application awareness is the capacity of a system to maintain information about connected applications to optimize their operation and that of any subsystems that they run or control. The concept of application-awareness has been in use for decades [12,13]. However, in recent years, software-defined network (SDN) emerged as one of the state-of-the-art technology to introduce application-awareness in data center applications by taking resource management control functions and implementing them through user-defined policies. In this paper, we use the term “application-awareness” to highlight the use of SDN centric environment to administer, manage and control cloud services.

In this paper, we present Bouncer, a model framework for services admission control in cloud environments. The idea behind services admission control in modern scalable systems provides a sense of infinite control over available resources of a data center. Guaranteed control uses a mechanism to set limits for some or all of the QoS parameters such as delay, variable delay, throughput, and packet loss [14]. This is achieved by admission control itself and by decisions of admission control in the access network by the management of throughput and traffic policies. SDNs can be used to significantly enhance network and service management by enabling separated control and data planes. The centralized position of an OpenFlow controller with a global vision of network states offers a promising approach to realizing flow-based admission control. The presented framework, Bouncer, uses a resource-aware admission control technique to improve workload-admission of cloud services. Bouncer considers required resources of a host machine as a composite of central processing unit (CPU), memory and storage resources. It rejects services request that is non-compliant to the statutory provisions of the system. This helps abridge the semantic gap between the required and the available cloud resources. The main contributions of this paper are three-fold. First, we identify resource administration challenges in virtualized environments. The objective of which highlights the importance of cloud system resources and the challenges in achieving high system throughput, improve load balance and minimal meta-job processing time. Second, we present Bouncer, a resource administration framework for virtualized environments. The proposed scheme uses SDN-enabled infrastructure to manage multiple admission control services and requests. The objective for choosing SDN-enabled infrastructure in Bouncer is to introduce application-awareness concepts in the workload

admission process of data centers. We also showcase its workload-admission control schemes for addressing workload congestion issues. Finally, we rigorously evaluated a prototype of Bouncer to verify its effectiveness through a set of experiments. The evaluated criterion was CPU resource utilization and system throughput estimation under varying workloads. The main objective of the evaluation was to analyze the proposed scheme for varying workloads and data traffic patterns.

The rest of the paper is organized as follows. Section 2 provides a brief discussion of related developments. Section 3 briefly outlines the background details and challenges in virtualized environments. Next; we extensively elaborate and discuss the design architecture of Bouncer and its components in Section 4. Section 5 provides a comprehensive evaluation of Bouncer. Finally, Section 6 concludes the paper.

2. Related Work

The work presented in the paper is related to cloud services admission control. Below we present a short synopsis of the related developments in the area of cloud services and delivery management.

Several research works have addressed the issues related to cloud service optimization. The admission control mechanism for cloud services has been examined as a mixed-integer optimization problem in [15]. The suggested solution also proposes to outsource the services and admission tasks from local data centers to external cloud services. In [16], an energy-efficient solution is presented to enhance the optimal allocation of resources by cloud providers. An efficient solution presented in [17] proposes ways to utilize the resources in a system that have been freed by preceding tasks. The multi-objective approach presented in the paper uses an approximation approach to perform this process.

A probability-based solution proposing the overbooking of resources in variable time frames is given in [18]. A model framework focusing on issues pertaining to hosting of elastic services is given in [19]. It proposes a way to estimate end-to-end deadline of cloud services. Several mathematical models are compared in [20] to compute the probability density function of services. In [21], authors came up with a novel idea of a prediction system which enables a scheduler to discard a service request if its available resource capability is unable to complete a request before its due time.

Admission of services to ensure QoS guarantees at cloud service provider level is elaborated in [22]. A learning-based cloud services admission control framework is proposed in [23] which admit service requests only after ensuring that they are likely to pass over the required threshold value set by the cloud service provider. Just like computing systems, cloud systems are highly non-linear. Therefore, a robust mechanism is required especially when control theories are employed for services admission control issues. In this regard, an adaptive controller design feedback system is developed [24] for a queuing model. This approach not only eases the experimentation design but also made the admission system more tolerant and stable to residual faults.

In distributed computing systems, such as cloud computing systems, various procedures or their combinations have been proposed to improve the handling of admission control requests and to improve their processing. These include frameworks proposing various control methods, optimization solutions, resource reservation, artificial intelligence-based solutions, etc. These proposed techniques can be improved to further refine the admission control schemes. Below we discuss recent developments in admission constraint-oriented admission control techniques.

In model-free methods, the models are segregated on the basis of the accompanying classes and their attributes. In this perspective, authors in [25] presented a standardized solution based on execution complexity of the received admission control requests. By using pre-defined patterns, a request can be classified as a simple or a complex task and thereby resource allocation was decided for the application. For a model-free approach in [26], authors present a framework to address the rigidity issues in admission control procedures. A learning-based admission control solution was presented in [27] which use the user-experience to grant access to required resources. A standard-based

solution is presented in [28] to provide adaptability-related solutions to applications which require priority in the execution process.

In model-based approaches, more information is needed to make an admission control model. These models work in two steps, i.e., evaluation and management. In the first stage, the model is evaluated under a control hypothesis [29]. A control hypothesis is a state-space model [30] which precisely highlights the area of activity of the admission control scheme. In the next phase, the selected state-space is populated with optimized task values. The resultant schemes provide a modular approach to handle admission control solution.

In [31], the authors investigate the resource management and access control mechanisms in distributed computing systems. It further explores the methodologies used in investigating resource admission scheme for cloud data centers. In [32], a feedback admission controller is used which uses a linear time-invariant (LTI) model to regulate CPU utilization for the job tasks. In [33], admission control resource allocation (ACRA) is presented which utilizes LTI models to tackle admission control and resource allocation challenges. ACRA also suggested handling scalability issues for increasing number of admission control requests for consolidated cloud services. A combinatorial approach was developed in [34] to jointly address admission control and resource allocation challenges by using modeling controllers. The proposed technique utilized a set of operating points to ensure the vertical integration and scaling for the stability of workload admission. The same technique was utilized in [35] to control a local admission controller task response. However, the proposed scheme only works under controlled environments and lack scalability for an increasing number of applications.

3. Background and Challenges

In this section, we first discuss the basic concepts of system virtualization and then highlight the major challenges in this field.

3.1. Workload Admission in Virtualized Environments

The deciding factor for accepting or rejecting a new elastic service request is one of the key decisions of an infrastructure provider. Usually, at workload admission time, it is not known how the elasticity requirements will affect the future capacity needs of an application. Sometimes, admitting new services increases the risk of the already deployed ones failing. Similarly, a strict acceptance policy for incoming services may result in an increased number of rejections. The elasticity decisions, therefore, should ideally be able to forecast in advance a change in a load of a service. It should also be able to react to the unexpected load changes faster than the rate of change of the load [36]. This requires fast and reliable mechanisms for managing usage and future usage predictions.

Figure 1 demonstrates a generalized view of a system virtualization architecture, in which a layer (the hypervisor or virtual machine monitor (VMM) is introduced between the guest operating systems and the underlying hardware. This virtualization technology is broadly employed in cloud data centers. Workload-admission and control services in virtualized environments decide whether a set of services can be admitted in a virtualized infrastructure. These schemes are based on a vast number of factors, including the services, infrastructure, QoS, policies, and costs, etc. In order to effectively utilize the underlying network resources, virtualized systems control and facilitate resource administration capabilities by:

- Allowing the efficient use of resources.
- Initiating support for upcoming requests.
- Assigning requests their respective resource requirements.
- Mapping various user requests with different QoS parameters to VMs.

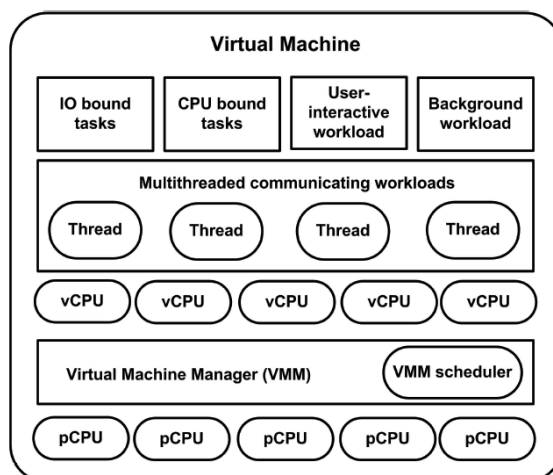


Figure 1. Workload admission and processing in virtualized environments.

3.2. Admission Control Challenges in System Virtualization

Due to the increasing size of computing systems, cloud admission control schemes are growing in complexity to a phenomenal degree. Below we highlight a few of the many challenges faced by these admission control schemes.

- **Resource contention:** Virtualized services and environments consistently need to verify whether there are enough resources available to satisfy the VM-level reservation (without interrupting VM kernel operations) or the VM-level reservations of other VMs running on that host.
- **Resource pre-emption:** To maximize profits, service providers are keen to accept as many services as possible. This develops a need to define the effective number of requests that can be accepted by a resource provider while ensuring that QoS violations are minimized. Efficient mechanisms are required to provide these services and involve using pre-emption aware schemes.
- **Oversubscription:** While oversubscription can leverage underutilized capacity in the cloud, it can also lead to overload. Workload-admission control avoids oversubscription effects by minimizing congestion, packet loss, and possible degradation of the user service experience.
- **Overhead and overbooking:** VM contention issues on shared computing resources in data centers bring noticeable performance overheads and can affect the VM performance for tenants. Efficient resource consolidation strategies can address these issues by using various overhead mitigation techniques; a number of these are presented in [37].

4. System Design

Fine-grained network resource provisioning requires knowledge of the instantaneous traffic demands, and subsequent harnessing of intelligent resource admission control as well as exploiting the rich path redundancy of the underlying data center network. However, achieving such provisioning using existing legacy mechanisms is faced with two fundamental challenges: First, estimating network load based on historical traffic demands (i.e., predictions) is dubious, since these change rapidly in data center environments and different patterns emerge over diverse timescales [38]. Second, the existing routing protocols such as equal-cost multi-path (ECMP) fail to support dynamic applications since they are load-agnostic and operate solely on packet header contents [39].

The output of the optimizer is a set of active components to both the power control and the routing modules. The power control is responsible for toggling the power states of switches, ports and line cards. The routing is responsible for flow admission and installs the computed routes into the network.

Data centers are built on top of legacy hardware and software technologies currently deployed within ISP networks. Cloud operators often assume high similarity between the two environments and hence employ similar resource management principles—static resource admission

and over-provisioning [40]. However, there are fundamental differences that are becoming apparent relatively early in the cloud data centers' lifetime and will only intensify as their utilization and commoditization increase. This is the fundamental reason behind using SDN-based infrastructure in the proposed scheme. Thus far, we have focused on our system's design and principles. Below though, we give explicit details on its policies and functions.

4.1. Design Considerations

Our proposed design of Bouncer strictly complies with the following design requirements.

- **Prioritization of services:** The prioritization of services minimizes or eliminates the need for a detailed and well-rehearsed plan. In order to enable the prioritization of services, Bouncer enforces prioritization levels at the workload-admission control and scheduling phases. This classification does not violate QoS rules and is also in line with the optimization requirements of the system.
- **Capacity Awareness:** To avoid overloading, Bouncer uses service profiling through SDN-based control (implemented during evaluation). This allows a system to define and extend its process capacity to avoid any service degradation.
- **Coordination of functions:** Admission-control systems should be based on the controlled-time-sharing principle implemented on VMs [41]. The fundamental reason for this is that time-sharing if controlled properly, allows an elastic response to a wide variety of disturbances affecting the workload performance. In Bouncer, we, therefore, assume that the requested time (by services) is accurate.

4.2. Workload Distribution and Overload Avoidance Policy for Workload-Admission Control

Bouncer considers a scenario similar to [23] where the submitted tasks (or the service requests) are represented by a row vector $t = [t_1, t_2, t_3, \dots, t_n]$. The resources required by these tasks is represented by a row vector $r = [r_1, r_2, r_3, \dots, r_n]$. We mapped these two-row vectors together in such a way that every individual member of row vector r represents the resource consumption for its respective element in row vector t . To ensure overload avoidance, we need to add a requirement whereby the system cannot accept requests that would lead it to exceed the available compute nodes of Bouncer. Therefore, for T number of total available compute nodes.

$$t \cdot r = \sum_{i=1}^n t_i r_i \leq T \quad (1)$$

where t_c and r_c represent the cumulative tasks and resources utilization, respectively. Our analysis goal is to develop a function where the chances of rejecting requests are minimal. Therefore, total response time R_t for a cluster Q can be derived as an objective function given by Equation (2).

$$R_t(Q) = W_t(Q_{inprocess}) + W_t(Q_{waiting}) \leq W_{threshold} \quad (2)$$

The performed analysis works as a gateway for the submitted requests. It streams and segregates requests by considering their threshold capacity (see Figure 2). A detailed commentary on the variables used in Equations (1) and (2) is presented in Table 1. The performed analysis improves the queuing and resource provisioning constraints in two ways:

- By mapping the requests along with their resource usage, which provides a clear description of all the services and their occupied resources by using Equation (1).
- By analyzing R_t for individual requests and ensuring that the results do not exceed $W_{threshold}$, which helps to ensure that the deadline violations of the requests can be minimized by using Equation (2).

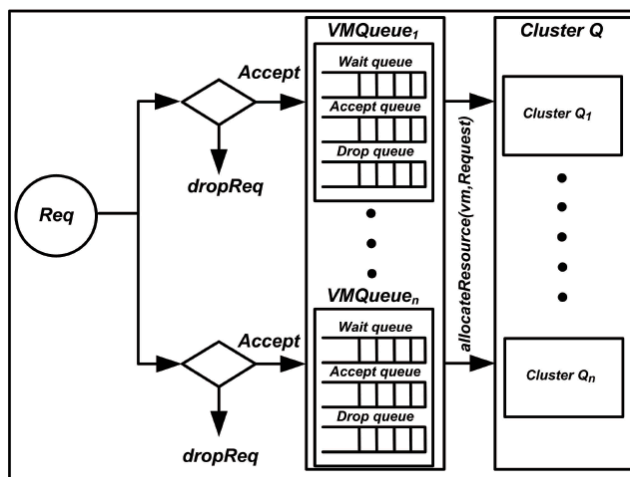


Figure 2. Analysis of resource provisioning in Bouncer.

Table 1. Workload distribution and overload avoidance policy analysis.

Variable	Description
r, t	Row vectors
T	Total number of available compute nodes
T_c	Cumulative tasks
R_c	Resource utilization
P_t	Number of permissible tasks
$R_t (c)$	Total response time
$Wt (Q_{inprogress})$	Waiting time of in-process requests of cluster Q
$Wt (Q_{waiting})$	Waiting time of waiting requests of cluster Q
$Wt (Q_{threshold})$	Threshold value for waiting time of cluster Q

In order to preserve the QoS levels, the workload distribution and overload avoidance policy assign applications to hosts with the required resource capacity. Here, we assume that the resources required by an individual application are a composite of the memory, CPU, and storage resources. We, therefore, consider every incoming application requirement on the basis of these three aspects. The admission-control policy thus looks up VMs with sufficient available resources to entertain the incoming resource demands. For this, we consider a queue “ i ” containing a number of applications, where each application has its own set of resource demands. We then consider another queue “ j ”, containing available VMs. In order to address the utility maximization concerns for queue structures [5], we ensure that every incoming application is provided with its required resource demands.

Bouncer ensures that applications are admitted into the system only once it is ensured that enough resources exist to process them. The working of the workload distribution and overload avoidance policy for workload-admission is outlined in Figure 2. Upon joining a network, our system assigns these applications to their potential VMs. This gives way to a free queued structure for the workload-admission system, where if resources are not contentious, applications can easily enter into a network.

However, if the required resource requirements exceed the available VM resource threshold capacity, applications are queued and the process is repeated until all the applications in “ i ” are either admitted to the VMs or is dropped beforehand.

In cloud services, taking a decision on the basis of existing utilization of network links is critically important. This not only facilitates the incoming services but also facilitates in accommodating futuristic network demands. The acceptance decision is based on the service consumption comparison. The acceptance decision of a cloud decision is dependent on the cost of accepting the request (resource consumption) and the maximum throughput of the system. Therefore, the choice between T_c and

$R_i(c)$ ensures a ratio of $W_i(Q_{threshold})$ for a sequence of admission control services. Therefore, in the worst-case scenario, the number of accepted service requests would be smaller than the required number of requests. The rationale behind the performed study is to derive strategies for developing demand patterns. We summarize the approach in Algorithm 1.

Algorithm 1. Workload distribution and overload avoidance policy.

```

Input: reqQueue: The queue of requesting services; vmQueue: The list of available VMs; reqResource:
1: The required resources for executing a VM request; vmResource: Cumulative available VM resource;
   waitQueue: The waiting queue of (not satisfactory) applications.
2: Output: Capacity-aware admission control and forwarding decision
3: /* We implement 3 major conditions to be satisfied before an application can be allocated to a VM. The
   wait and drop Queues amass the pending and rejected applications */
4: while reqQueue != NULL do
5:     Request ← DeQueue(reqQueue);
6:     if vmQueue == NULL then
7:         dropReq (request);
8:         continue;
9:     end if
10:    while vmQueue != NULL do
11:        vm ← getVM (vmQueue);
12:        /* We consider required resources of a VM as a composite of CPU, Memory and Storage resources. VM
           must ensure that it have enough resources to handle the resource demands */
13:        if vmResource[vm] ≥ reqResource[request] then
14:            allocateResource(vm, request);
15:            DeqQueue(vmQueue, vm);
16:            /* Traverse VM queue, and consider next resource demand */
17:            break;
18:        end if
19:        vmQueue ≥ vmQueue.next;
20:        /* If none of available VM can satisfy the service requirements, append requests */
21:        if vmQueue == NULL then
22:            EnQueue(waitQueue, request);
23:        end if
24:    end while
25: end while

```

5. Performance Evaluation and Results

In virtualized environments, requests generated by different services may cause synchronization issues. The basic idea behind Bouncer is thus to provide a strategy to keep all the functions synchronized and online at the same time. It is easy to apply our framework in practice. We first define the benchmark strategies which we use for evaluation. An explicit detail on the implementation of testbed with hardware and software configurations is also described below.

5.1. Baseline Strategy

In the current literature, we observe that most performance analysis of cloud service admission control schemes was carried out under the assumption that the queue holding times for new services and waiting services are identically distributed (some with exponential distribution), i.e., all cloud services calls were assumed to be identically distributed with the same parameter. This mechanism presents several characteristics inherited from different well-known approaches, such as the Uni-path reservation [42] and a unique reservation threshold [43]. In admission control schemes, baseline strategies are considered as a benchmark for the performance of a particular task. However, keeping in view the cloud services provisioning systems, we compare Bouncer with three schemes which are

similar to the service-level agreement (SLA) based approaches in conventional cloud systems. In this paper, we consider the task execution of a unit cloud service as the minimal baseline parameter. We compare Bouncer against the following three strategies.

- **Maximal Admittance Policy (MAP):** The strategy allows all incoming requests without filtering them. Its objective is to improve and maximize the resource admittance rate in a system. The acceptance queue for this strategy in our system is theoretically set to maximum (infinite) but we limited our experimentation value up to 2500 requesting jobs in the job queue. This policy is similar to High Availability Admission Control Setting and Policy [44], often used in conventional SLAs.
- **Smart Job Admission Policy (SJAP):** In SJAP, we consider a scenario where the request time R_t for a job Q_i measured on the basis of the waiting processes $Q_{inprocess}$. Therefore, in order to get selected for admission in SJAP queue, a request must satisfy the condition of $R_t \geq Q_{inprocess}$ and is similar to [45] except that it does not perform the slicing for function isolation.
- **Load Balancing and Control Policy (LBCP):** LBCP identify the incoming requests based on their impact on system resources by performing isolation on individual requests. The policy is a limited version of the load balancing policy presented in [46].

5.2. Characteristics of the Proposed Methodology

In this paper, we present a resource-aware cloud admission control scheme for cloud services which ensures that a minimal resource level is maintained before the grant of admission control services to cloud applications. Below we gave a short discussion on the uniqueness, advantages, and shortcomings of the presented approach.

The uniqueness of the proposed scheme is that uses application-awareness as benchmark for the application acceptance of resources. Looking at the current state of affairs in the cloud and data center industry, the use of proprietary hardware and switching equipment discourages the free and open access to hardware resources of the data center. Therefore, the application-awareness is required to achieve transparency for performing network statistics.

By offloading the control plane to computing platforms running on centrally-controlled, software-controlled commodity hardware, SDN unleashes the potential to operate computation-intensive machine learning tools and solve complex optimization problems in a centralized fashion. Therefore, we use application-awareness concepts and employ those using software-defined networking devices in the experimentation process.

The application-awareness concepts when introduced to the field of cloud services admission control not only help in revisiting the traditional cloud services admission control concepts but also helps in exploring new avenues for managing service level deliverance requirements under varying network traffic conditions. The presented approach will not only facilitate the management of data center resources but will also facilitate in achieving the desired level of service for end-users.

In terms of the limitations for the proposed technique, the services admission controller (actually the SDN controller) might face bottleneck issues when confronting numerous service requests simultaneously. Apart from the bottleneck issue is the security challenges for the services. These challenges can be resolved by limiting the number of services or by employing multiple scalable SDN controllers. For the security issues, third party security services and restriction-based solutions can be implemented.

5.3. Testbed and Topology

The testbed used to evaluate the performance of the hierarchical control framework used three identical cluster servers with tree-structured consolidated applications. The controller sent the necessary information to achieve an equilibrium state. In order to analyze the performance of the proposed scheme, we used a real-life data used in [47]. The traffic generation matrix used Poisson demand arrivals at a rate of λ demands/s with a size of 100 Mb/s and distributed mean time value of 30s.

In high-performance computing systems, applications are categorized on the basis of their priority. This categorization is performed to ensure that the resource allocation methods are well-defined and serve a clear objective. The same case applies to Bouncer where all incoming service requests follow a Poisson arrival scheme [48]. Employing Poisson regression ensures that all events are independent and they don't hinder the path of newly arriving services.

All cloud service requests follow a Poisson arrival model in the system with mean arrival rates of $\lambda_{i,j}$ ($\lambda_{1,1}$, $\lambda_{1,2}$, $\lambda_{2,1}$ and $\lambda_{2,2}$, respectively). Each service request i,j has four associated parameters (for access to resource type, its queue, service time limit denoted with a constant value D_{max} which is related to the prescribed SLA requirement. The total service arrival rate in a system can be denoted as $\lambda = \sum_{i,j} \lambda_{i,j}$. Each connection request i,j has a negative connection time (service time distribution) with an average rate of $1/\mu_{i,j}$. On the basis of the mentioned scenario, the load for each service request i,j represented as $\rho_{i,j}$ and is given by $\rho_{i,j} = \mu_{i,j}^{-1} \lambda_{i,j}$. The total load intensity for the system can be calculated as $\rho = \sum_{i,j} \mu_{i,j}^{-1} \lambda_{i,j}$.

We run a number of experiments to evaluate the performance of Bouncer. The implementation required simple modifications to be made to the VM services and policies. A private infrastructure was deployed at Wuhan University, Wuhan, China, and was used here for evaluation purposes. The systems used in the experimental evaluation use an Intel Xeon CPU (E5-2650 v2) with a clock speed of 2.60GHz. Using the state-of-the-art Ubuntu v18.04 environment and VM v15.0.4, we employ two eight slot SDN enabled HP 12508E switches, with a 10.8 Tbps maximum switching capacity. The reason for implementing SDN-enabled infrastructure in our experiments is that Bouncer employs application-awareness features (through SDNs) for services admission control. We fine-tuned the number of services received and sent by way of the hosts using SDN-enabled switches. The workload used in the experiments was sampled from Wuhan University's local data center.

5.4. Results and Discussion

In virtualized environments like cloud data centers, there is a linear relationship between the average response time and the workload of a system. Bouncer resolves system overloading by dropping excessive requests. It also maintains average response time under the required range. For cloud services requesting admission into the system, have a linear relation. We compared Bouncer's average response time with an average response time of three important admission control strategies. It is clearly visible from Figure 3a that the average response time of Bouncer is effective than the other compared schemes. The curve line showing our framework performance also indicates that it resists a steady increase in average response time of request completion even after an increase in the number of job requests.

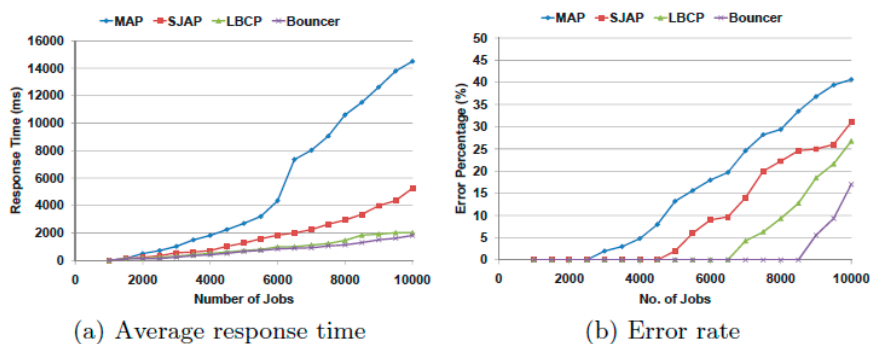


Figure 3. Response time and error rate estimation. (a) Average response time; (b) error rate.

We compared the error rate of all four admission control schemes on our testbed. The experiment is conducted to determine the validity of Bouncer performance with other schemes. From the results in Figure 3b, it is clear that Bouncer serves the most batch jobs with least error percentage whereas LBCP,

MAP and SJAP having higher error rates. We believe that Bouncer has better performance in reducing error percentage because it delivers special priority to resource-hungry admission control schemes. The acquired error rate also indicates that Bouncer serves the maximum number of successful requests.

The performance overheads in virtualized environments are difficult to handle. It is because large data centers are often spread across multiple geo-distributed dominions. Conventional resource isolation techniques have also failed to address these issues. Although the tested workload used in the evaluation of Bouncer is realistic, we consider a case where CPU demands of all the evaluated schemes do not have any communication overheads. We compared the CPU utilization percentage results in Figure 4a. We also calculated and compared the service monitoring overhead costs in Table 2. Results demonstrate that Bouncer incurs fewer overheads than MAP and SJAP whereas its overheads are slightly higher than LBCP. It is mainly because LBCP performs isolation functions on requests and segregates them to achieve higher results. Although this technique improves the results, it slows down the service admission in queued architectures. In contrast, Bouncer’s service admission and acceptance process do not constraint system resources. It’s simple yet efficient request handler and resource allocation process consumes less CPU time than MAP, SJAP, and LBCP. Our experiments for comparing CPU resource utilization demonstrate that Bouncer achieves higher throughput and acceptable response time with the compared workload.

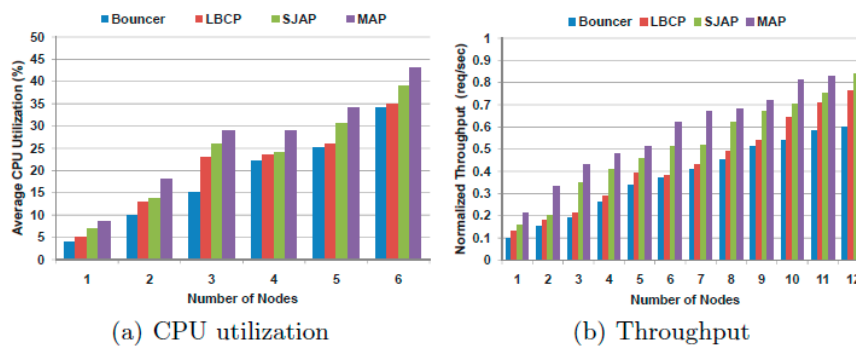


Figure 4. CPU resource utilization and system throughput estimation. (a) CPU utilization; (b) throughput.

Table 2. Monitoring overhead cost (in ms) for evaluated service requests.

CPU Utilization	10%	20%	30%	40%	50%
Bouncer	2.93	3.45	3.90	4.22	4.53
Load Balancing and Control Policy (LBCP)	2.68	3.12	3.87	4.14	4.36
Smart Job Admission Policy (SJAP)	3.16	3.58	3.98	4.29	4.83
Maximal Admittance Policy (MAP)	3.46	3.83	4.32	4.65	5.23

In virtualized environments, an admission control algorithm has a critical role in handling the scalability concerns of a system. If a system is scalable, it is able to handle more requests per unit time. Therefore, to compare the scalability of our model framework, we conduct a simple experiment. We generate a large number of requests and map them on our test bed’s processing nodes. The aim of this experiment is to evaluate the throughput of Bouncer. We considered the monitoring results and process execution time and compared them with their respective overheads to calculate the overall system’s scalability. The normalized throughput in performed experimentation is measured in requests per second unit. The results in Figure 4b demonstrate that Bouncer algorithm achieves better throughput for the compared number of nodes of a cluster.

It is important to mention here that Bouncer is an SLA-aware implementation of policy-based cloud services admission. In case of distributed concurrent applications, such as block chain systems or multi-parallel processing queue service streaming, heuristic solutions have been proposed in [49–51] which provide scalability-related solutions to reduce the gap between the service provider and the

clients by using a utility factor. A utility factor in case of cloud services admission may be referred to time (scheduling, response time) and budget (pay as you go service usage policy, etc.). Resource allocation challenges are often addressed on basis of an optimization related constraint; therefore, in order to ensure the service quality assurance on a multi-platform scale such as block-chain technology, we measure the services output success on basis of the SLA principles of success. These service levels are ensured by server-client coordination constraints and are not included in the local optimization problem on an individual server in the decentralized method.

We performed resource consumption for core switches under the pretext of traffic utilization. In this concern, two scenarios namely low traffic level and high traffic level were implemented. The purpose of the evaluation was to evaluate the effect of cloud services admission control scheme on network switches. The switching load functions were evaluated by using an SDN-enabled evolved packet core (EPC) gateway which can extract switching statistics from the 8 slot SDN enabled HP 12508E SDN-enabled switch. From the results in Figure 5, we can see that Bouncer's core switch utilization has relatively little impact on overall data usage whereas the other three benchmark criteria are relatively consuming more switching resources for performing the same task. We believe that the Poisson arrival scheme and associated parameters reduced the overall switch utilization which is absent for the other 3 benchmark services and can also handle scalability related challenges.

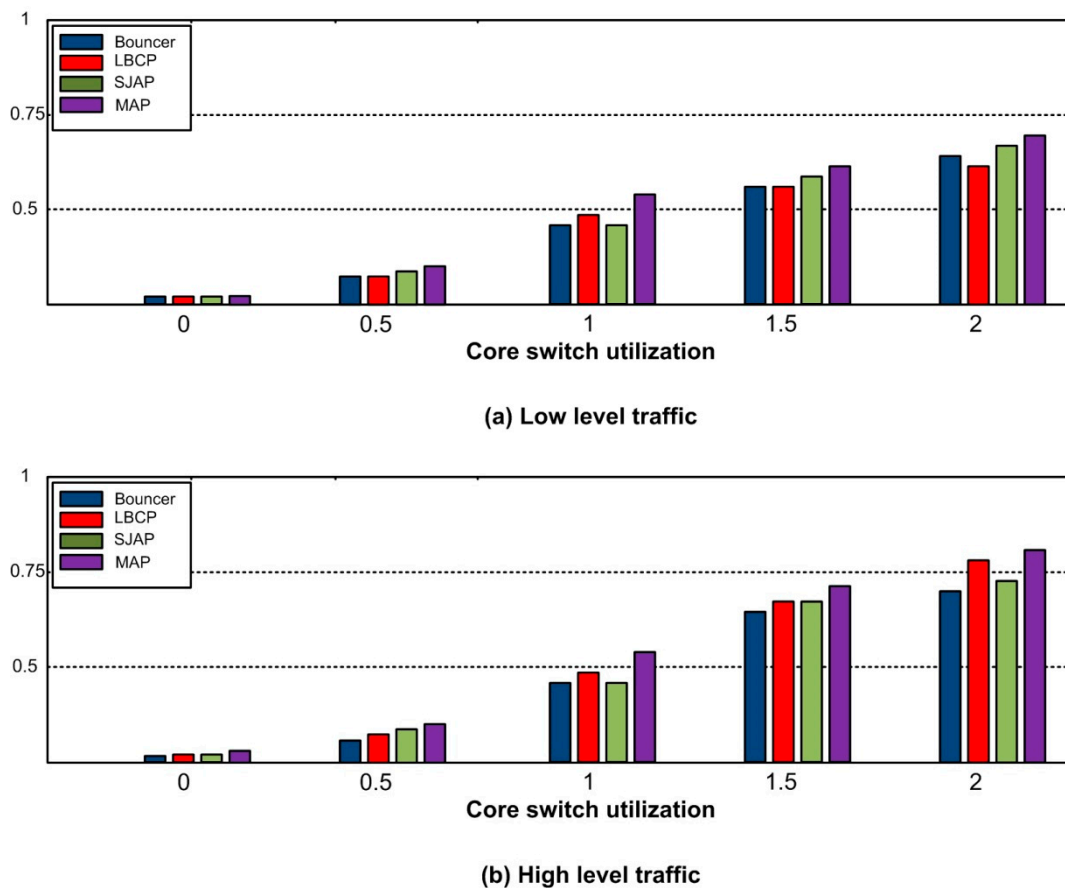


Figure 5. Core switch utilization comparison and estimation. (a) Low level traffic; (b) high level traffic.

In summary, the Bouncer algorithm provides a simple, easy and horizontal scaling solution for admitted service requests through a standardized and logical resource allocation method.

6. Conclusion and Future Work

Cloud systems receive numerous service requests in a very short period of time which may cause service congestion and delay in service response. This can frustrate users' interests and can also lead to

significant losses in revenue and reputation of the cloud services provider. This work proposes Bouncer, a model framework for addressing the resource allocation challenges in virtualized environments. We briefly identified the major challenges in admission control strategies and highlighted the target areas that need significant attention for improvement. Bouncer ensures that services in a cloud environment are only accepted if the system has enough available resources to handle them. Its workload distribution and overload avoidance policy admit services request after ensuring that their requirements can be met by the system. By generating a large number of service requests through the selected number of nodes, we implemented and compared a prototype of Bouncer with three conventional admission control policies. The results obtained from the evaluated workloads confirm that Bouncer outperforms the compared schemes. The results demonstrate significant improvements, specifically in terms of service response time, reduced errors percentage for workload execution and memory overheads. In the future, we plan to develop a performance isolation scheme for Bouncer.

Author Contributions: A.A.A. designed the study, M.A.A.A. designed the experiments, M.A.E. and H.A.K. performed the experiments, A.A.A., M.A.A.A., M.A.E. and H.A.K. wrote the manuscript, A.A.A. revised the manuscript, and S.K. supervised the research study. All authors have read and approved the manuscript.

Funding: Following are results of a study on the “Leaders in INdustry-university Cooperation+” project, supported by the Ministry of Education and National Research Foundation of Korea.

Acknowledgments: The authors thank the anonymous reviewers for their thorough feedback and suggestions, which were crucial to improving the contents and presentation of the article.

Conflicts of Interest: The authors declare no conflicts of interest.

References

- Xu, M.; Buyya, R. Brownout approach for adaptive management of resources and applications in cloud computing systems: A taxonomy and future directions. *ACM Comput. Surv. (CSUR)* **2019**, *52*, 8. [[CrossRef](#)]
- Zahavi, E.; Shpiner, A.; Rottenstreich, O.; Kolodny, A.; Keslassy, I. Links as a Service (LaaS): Guaranteed tenant isolation in the shared cloud. *IEEE J. Sel. Areas Commun.* **2019**, *37*, 1072–1084. [[CrossRef](#)]
- Wang, S.; Zhao, Y.; Xu, J.; Yuan, J.; Hsu, C.H. Edge server placement in mobile edge computing. *J. Parallel Distrib. Comput.* **2019**, *127*, 160–168. [[CrossRef](#)]
- Lu, Q.; Xu, X.; Liu, Y.; Weber, I.; Zhu, L.; Zhang, W. uBaaS: A unified blockchain as a service platform. *Future Gener. Comput. Syst.* **2019**, *101*, 564–575. [[CrossRef](#)]
- Abbasi, A.A.; Abbasi, A.; Shamshirband, S.; Chronopoulos, A.T.; Persico, V.; Pescapè, A. Software-Defined Cloud Computing: A Systematic Review on Latest Trends and Developments. *IEEE Access* **2019**, *7*, 93294–93314. [[CrossRef](#)]
- Leontiou, N.; Dechouniotis, D.; Denazis, S.; Papavassiliou, S. A hierarchical control framework of load balancing and resource allocation of cloud computing services. *Comput. Electr. Eng.* **2018**, *67*, 235–251. [[CrossRef](#)]
- Sikora, T.D.; Magoulas, G.D. Neural adaptive admission control framework: SLA-driven action termination for real-time application service management. *Enterp. Inf. Syst.* **2019**, 1–41. [[CrossRef](#)]
- Ari, A.A.A.; Damakoa, I.; Titouna, C.; Labraoui, N.; Gueroui, A. Efficient and Scalable ACO-Based Task Scheduling for Green Cloud Computing Environment. In Proceedings of the 2017 IEEE International Conference on Smart Cloud (SmartCloud), New York, NY, USA, 3–5 November 2017; pp. 66–71.
- Nayak, B.; Padhi, S.K.; Pattnaik, P.K. Static Task Scheduling Heuristic Approach in Cloud Computing Environment. In *Information Systems. Design and Intelligent Applications*; Springer: Singapore, 2019; pp. 473–480.
- Jlassi, S.; Mammari, A.; Abbasi, I.; Graiet, M. Towards correct cloud resource allocation in FOSS applications. *Future Gener. Comput. Syst.* **2019**, *91*, 392–406. [[CrossRef](#)]
- Zhou, H.; Ouyang, X.; Ren, Z.; Su, J.; de Laat, C.; Zhao, Z. A Blockchain Based Witness Model for Trustworthy Cloud Service Level Agreement Enforcement. In Proceedings of the IEEE INFOCOM 2019-IEEE Conference on Computer Communications, Paris, France, 29 April–2 May 2019; pp. 1567–1575.
- Jin, H.; Abbasi, A.A.; Wu, S. Pathfinder: Application-aware distributed path computation in clouds. *Int. J. Parallel Program.* **2017**, *45*, 1273–1284. [[CrossRef](#)]

13. Abbasi, A.; Jin, H. v-Mapper: An Application-Aware Resource Consolidation Scheme for Cloud Data Centers. *Future Internet* **2018**, *10*, 90. [[CrossRef](#)]
14. Shen, D.; Junzhou, L.; Dong, F.; Jin, J.; Zhang, J.; Shen, J. Facilitating Application-aware Bandwidth Allocation in the Cloud with One-step-ahead Traffic Information. *IEEE Trans. Serv. Comput.* **2019**. [[CrossRef](#)]
15. Wei, X.; Tang, C.; Fan, J.; Subramaniam, S. Joint Optimization of Energy Consumption and Delay in Cloud-to-Thing Continuum. *IEEE Internet Things J.* **2019**, *6*, 2325–2337. [[CrossRef](#)]
16. Sun, H.; Yu, H.; Fan, G.; Chen, L. Energy and time efficient task offloading and resource allocation on the generic IoT-fog-cloud architecture. *Peer Peer Netw. Appl.* **2019**, 1–16. [[CrossRef](#)]
17. Liu, C.F.; Bennis, M.; Debbah, M.; Poor, H.V. Dynamic task offloading and resource allocation for ultra-reliable low-latency edge computing. *IEEE Trans. Commun.* **2019**, *67*, 4132–4150. [[CrossRef](#)]
18. Mou, D.; Li, W.; Li, J. A network revenue management model with capacity allocation and overbooking. *Soft Comput.* **2019**, 1–10. [[CrossRef](#)]
19. Chen, M.; Li, W.; Fortino, G.; Hao, Y.; Hu, L.; Humar, I. A Dynamic Service Migration Mechanism in Edge Cognitive Computing. *ACM Trans. Internet Technol. (TOIT)* **2019**, *19*, 30. [[CrossRef](#)]
20. Calzarossa, M.C.; Della Vedova, M.L.; Tessera, D. A methodological framework for cloud resource provisioning and scheduling of data parallel applications under uncertainty. *Future Gener. Comput. Syst.* **2019**, *93*, 212–223. [[CrossRef](#)]
21. Zhang, Y.; Zhang, L.; Du, L.; Liu, C.; Chen, Y. Distributed energy-efficient target tracking algorithm based on event-triggered strategy for sensor networks. *IET Control. Theory Appl.* **2019**, *13*, 1564–1570. [[CrossRef](#)]
22. Gavvala, S.K.; Jatoth, C.; Gangadharan, G.R.; Buyya, R. QoS-aware cloud service composition using eagle strategy. *Future Gener. Comput. Syst.* **2019**, *90*, 273–290. [[CrossRef](#)]
23. Abbasi, A.A.; Jin, H.; Wu, S. A software-Defined Cloud Resource Management Framework. In *Asia-Pacific Services Computing Conference*; Springer: Cham, Switzerland, 2015; pp. 61–75.
24. Zhang, J.; Peng, C.; Xie, X.; Yue, D. Output Feedback Stabilization of Networked Control Systems Under a Stochastic Scheduling Protocol. *IEEE Trans. Cybern.* **2019**. [[CrossRef](#)]
25. Tan, Z.; Qu, H.; Zhao, J.; Ren, G.; Wang, W. Low-complexity networking based on joint energy efficiency in ultradense mmWave backhaul networks. *Trans. Emerg. Telecommun. Technol.* **2019**, *30*, e3508. [[CrossRef](#)]
26. Harishankar, M.; Pilaka, S.; Sharma, P.; Srinivasan, N.; Joe-Wong, C.; Tague, P. Procuring Spontaneous Session-Level Resource Guarantees for Real-Time Applications: An Auction Approach. *IEEE J. Sel. Areas Commun.* **2019**, *37*, 1534–1548. [[CrossRef](#)]
27. Bega, D.; Gramaglia, M.; Banchs, A.; Sciancalepore, V.; Costa-Perez, X. A machine learning approach to 5G infrastructure market optimization. *IEEE Trans. Mobile Comput.* **2019**. [[CrossRef](#)]
28. Momenzadeh, Z.; Safi-Esfahani, F. Workflow scheduling applying adaptable and dynamic fragmentation (WSADF) based on runtime conditions in cloud computing. *Future Gener. Comput. Syst.* **2019**, *90*, 327–346. [[CrossRef](#)]
29. Gürsu, H.M.; Vilgelm, M.; Alba, A.M.; Berioli, M.; Kellerer, W. Admission Control Based Traffic-Agnostic Delay-Constrained Random Access (AC/DC-RA) for M2M Communication. *IEEE Trans. Wirel. Commun.* **2019**, *18*, 2858–2871. [[CrossRef](#)]
30. Al-qaness, M.A.A.; Elaziz, A.A.; Kim, S.; Ewees, A.A.; Abbasi, A.A.; Alhaj, Y.A.; Hawbani, A. Channel State Information from Pure Communication to Sense and Track Human Motion: A Survey. *Sensors* **2019**, *19*, 3329. [[CrossRef](#)]
31. Nam, J.; Jo, H.; Kim, Y.; Porras, P.; Yegneswaran, V.; Shin, S. Operator-Defined Reconfigurable Network OS for Software-Defined Networks. *IEEE/ACM Trans. Netw.* **2019**, *27*, 1206–1219. [[CrossRef](#)]
32. Avgeris, M.; Dechouniotis, D.; Athanasopoulos, N.; Papavassiliou, S. Adaptive resource allocation for computation offloading: A control-theoretic approach. *ACM Trans. Internet Technol. (TOIT)* **2019**, *19*, 23. [[CrossRef](#)]
33. Chen, G.; Zhang, Y.; Shi, Y.; Zeng, Q. Joint resource allocation and admission control mechanism in software defined mobile networks. *China Commun.* **2019**, *16*, 33–45.
34. Taleb, T.; Afolabi, I.; Samdanis, K.; Yousaf, F.Z. On Multi-domain Network Slicing Orchestration Architecture and Federated Resource Control. *IEEE Netw.* **2019**. [[CrossRef](#)]
35. Abeni, L.; Biondi, A.; Bini, E. Hierarchical scheduling of real-time tasks over Linux-based virtual machines. *J. Syst. Softw.* **2019**, *149*, 234–249. [[CrossRef](#)]

36. Kyung, Y.; Park, J. Prioritized admission control with load distribution over multiple controllers for scalable SDN-based mobile networks. *Wirel. Netw.* **2019**, *25*, 2963–2976. [[CrossRef](#)]
37. Bhushan, K.; Gupta, B.B. Network flow analysis for detection and mitigation of Fraudulent Resource Consumption (FRC) attacks in multimedia cloud computing. *Multimed. Tools Appl.* **2019**, *78*, 4267–4298. [[CrossRef](#)]
38. Qin, Y.; Guo, D.; Luo, L.; Cheng, G.; Ding, Z. Design and optimization of VLC based small-world data centers. *Front. Comput. Sci.* **2019**, *13*, 1034–1047. [[CrossRef](#)]
39. Wang, S.; Li, X.; Qian, Z.; Yuan, J. Distancer: A Host-Based Distributed Adaptive Load Balancer for Datacenter Traffic. In *International Conference on Algorithms and Architectures for Parallel Processing*; Springer: Cham, Switzerland, 2008; pp. 567–581.
40. Shen, H.; Chen, L. Resource demand misalignment: An important factor to consider for reducing resource over-provisioning in cloud datacenters. *IEEE/ACM Trans. Netw.* **2018**, *26*, 1207–1221. [[CrossRef](#)]
41. Caballero, P.; Banchs, A.; de Veciana, G.; Costa-Pérez, X.; Azcorra, A. Network slicing for guaranteed rate services: Admission control and resource allocation games. *IEEE Trans. Wirel. Commun.* **2018**, *17*, 6419–6432. [[CrossRef](#)]
42. Martini, B.; Gharbaoui, M.; Adami, D.; Castoldi, P.; Giordano, S. Experimenting SDN and Cloud Orchestration in Virtualized Testing Facilities: Performance Results and Comparison. *IEEE Trans. Netw. Serv. Manag.* **2019**. [[CrossRef](#)]
43. Massaro, A.; de Pellegrini, F.; Maggi, L. Optimal Trunk-Reservation by Policy Learning. In *Proceedings of the IEEE INFOCOM 2019—IEEE Conference on Computer Communications*, Paris, France, 29 April–2 May 2019; pp. 127–135.
44. Rothenberg, C.E.; Roos, A. A review of policy-based resource and admission control functions in evolving access and next generation networks. *J. Netw. Syst. Manag.* **2008**, *16*, 14–45. [[CrossRef](#)]
45. Han, B.; Sciancalepore, V.; di Feng Costa-Perez, X.; Schotten, H.D. A Utility-Driven Multi-Queue Admission Control Solution for Network Slicing. In *Proceedings of the IEEE INFOCOM 2019—IEEE Conference on Computer Communications*, Paris, France, 29 April–2 May 2019; pp. 55–63.
46. Lin, Y.-D.; Wang, C.C.; Lu, Y.; Lai, Y.; Yang, H.-S. Two-tier dynamic load balancing in SDN-enabled Wi-Fi networks. *Wirel. Netw.* **2018**, *24*, 2811–2823. [[CrossRef](#)]
47. Gao, K.; Xu, C.; Qin, J.; Zhong, L.; Muntean, G.M. A Stochastic Optimal Scheduler for Multipath TCP in Software Defined Wireless Network. In *Proceedings of the ICC 2019-2019 IEEE International Conference on Communications (ICC)*, Shanghai, China, 20–24 May 2019; pp. 1–6.
48. Ferdouse, L.; Anpalagan, A.; Erkucuk, S. Joint Communication and Computing Resource Allocation in 5G Cloud Radio Access Networks. *IEEE Trans. Veh. Technol.* **2019**. [[CrossRef](#)]
49. Zhang, W.; Sun, H.; Zhao, D.; Xu, L.; Liu, X.; Zhou, J.; Ning, H.; Guo, Y.; Yang, S. A Streaming Cloud Platform for Real-Time Video Processing on Embedded Devices. *IEEE Trans. Cloud Comput.* **2019**. [[CrossRef](#)]
50. Bhimani, J.; Mi, N.; Leaser, M.; Yang, Z. New Performance Modeling Methods for Parallel Data Processing Applications. *ACM Trans. Model. Comput. Simul. (TOMACS)* **2019**, *29*, 15. [[CrossRef](#)]
51. Kiss, T.; DesLauriers, J.; Gesmier, G.; Terstyanszky, G.; Pierantoni, G.; Oun, O.A.; Taylor, S.J.; Anagnostou, A.; Kovacs, J. A cloud-agnostic queuing system to support the implementation of deadline-based application execution policies. *Future Gener. Comput. Syst.* **2019**, *101*, 99–111. [[CrossRef](#)]

