INDICES: Applying DDDAS Principles for Performance Interference-aware Cloud-to-Fog Application Migration

Shashank Shekhar and Ajay Dev Chhokra and Anirban Bhattacharjee and Yogesh Barve and Shweta Khare and Guillaume Pallez and Hongyang Sun and Aniruddha Gokhale and Gabor Karsai

Abstract An increasing number of interactive applications and services, such as online gaming and cognitive assistance, are being hosted in the cloud because of its elastic properties and cost benefits. Despite these benefits, the longer and often unpredictable end-to-end network latencies between the end user and the cloud can be detrimental to the response time requirements of the applications. Although technology enablers, such as Cloudlets or Micro Data Centers (MDCs), are increasingly being leveraged by cloud infrastructure providers to address the network latency concerns, existing efforts in re-provisioning services from the cloud to the MDCs seldom focus on ensuring that the performance properties of the migrated services are met. This chapter demonstrates the application of DDDAS principles to address these limitations by: (a) determining when to re-provision; (b) identifying the appropriate MDC and a suitable host within that MDC that meets the performance considerations of the applications; and (c) ensuring that the cloud service provider continues to meet customer service-level objectives while keeping its operational

Shashank Shekhar
Siemens Corporate Technology, Princeton, NJ, USA, e-mail: shashankshekhar@siemens.com
and Ajay Dev Chhokra
Vanderbilt University, Nashville, TN, USA e-mail: ajay.d.chhokra@vanderbilt.edu
and Anirban Bhattacharjee
Vanderbilt University, Nashville, TN, USA e-mail: anirban.bhattacharjee@vanderbilt.edu
and Yogesh Barve
Vanderbilt University, Nashville, TN, USA e-mail: yogesh.d.barve@vanderbilt.edu
and Shweta Khare
Vanderbilt University, Nashville, TN, USA e-mail: shweta.p.khare@vanderbilt.edu
and Guillaume Pallez
INRIA, Bordeaux, France e-mail: guillaume.pallez@inria.fr
and Hongyang Sun
Vanderbilt University, Nashville, TN, USA e-mail: hongyang.sun@vanderbilt.edu
and Aniruddha Gokhale
Vanderbilt University, Nashville, TN, USA e-mail: a.gokhale@vanderbilt.edu
and Gabor Karsai
Vanderbilt University, Nashville, TN, USA e-mail: gabor.karsai@vanderbilt.edu

and energy costs low. Empirical evaluations using a setup comprising a cloud data center and multiple MDCs composed of heterogeneous hardware are presented to validate our claims.

1 Introduction

The cloud has become an attractive hosting platform for a variety of interactive and soft real-time applications, such as cloud gaming, cognitive assistance, health monitoring systems and collaborative learning, due to its elastic properties and cost benefits. Despite these substantial advantages, the response time considerations of the users mandate lower latencies for the applications. Prior work [31, 36] have shown that in highly interactive applications, latencies exceeding 100 milliseconds (ms) may be too high for acceptable user experience. However, real-world experiments have shown that the latencies experienced by geographically distributed users of an interactive service may tend to be on the order of several hundreds of milliseconds [52]. Consequently, there is a need to bound the resulting response times within acceptable limits.

For any cloud-hosted interactive application, the key factors that affect the round trip latencies are the network delay between the client and the cloud, particularly the roundtrip delay between the nearest access point of the client and the cloud, and the time it takes to serve the client request in the cloud. Other factors, such as the time taken by the thin client, the time to reach the nearest access point or time for the load balancer at the cloud front-end are negligible. Thus, any improvement in response times must focus on reducing the network delays and the server processing time.

In recent years, edge computing, cloudlets [47] or Micro Data Centers (MDCs) [3] have emerged as one of the key mechanisms to manage and bound the transit latency by supporting cloud-based services closer to the clients. MDCs can be viewed as "data center in a box," which act as the middle tier in the emerging "mobile device–MDC–cloud" hierarchy [47].[1] MDCs possess key attributes of soft states, sufficient compute power and connectivity, proximity to clients, and conformance to standard cloud technologies.

Recent efforts [13,14,37,60] have leveraged the cloud, MDCs and mobile ad-hoc networks by focusing primarily on cyber foraging, where tasks are offloaded from mobile devices to the cloud/MDCs for faster execution and conserving resources on the mobile client endpoints. However, only recently has there been an increasing interest in moving tasks from the central clouds to the MDCs. Those that do, however, have seldom considered the resulting application performance because these efforts tend to overlook the fact that servers within the MDC may themselves get overloaded, thereby worsening the user experience as compared to that of a traditional cloud-hosted interactive service. On the other hand, efforts that consider performance of MDCs make very simplistic assumptions regarding their performance models.

[1] In the rest of the chapter, we will use the term MDC to represent all emerging mechanisms, such as Cloudlets, Micro Datacenters (MDCs), Locavore infrastructures, etc.

In this chapter, we focus on the performance of MDCs, specifically on the key factors contributing to performance degradation of applications running in MDCs specifically and data centers in general. One fundamental system property that is often overlooked in prior works is performance interference, which is caused by co-located applications in virtualized infrastructures [10, 17, 33, 35]. Performance interference, being an inherent property of any virtualized system, manifests itself in MDCs also and therefore must be factored in by any approach that is performance-aware. Thus, we consider a "just-in-time and performance-aware" service migration approach for migrating cloud-based interactive services hosted in a centralized cloud data center to an MDC.

To support such as vision, a number of challenges manifest themselves as described below that must be addressed by any just-in-time and performance-aware cloud-to-fog application migration solution:

- *Hardware heterogeneity:* Differences in hardware configurations of the servers in a traditional data center and in an MDC will provide different performance profiles, and hence should be accounted for in the analyses.
- *Performance Interference:* Noisy neighbors [11] cause performance issues, which must be considered in both traditional data centers and MDCs. However, since an MDC is orders of magnitude smaller than a traditional data center, performance interference may be more pronounced and manifests more rapidly than in traditional data centers.
- *Network performance measurements:* Accurate latency and bandwidth measurements are required to reliably work over Wide Area Networks (WANs). This is important, since accurately estimating the value for the transit latency is critical in our problem formulation and solution.
- *System performance measurements:* Accurate application and server performance measurement and logging techniques are required to accurately measure the service execution (i.e., service execution time on the hardware of the data centers or MDCs).

The cloud-fog resource spectrum is a highly dynamic system and hence no *a priori,* statically-defined solution is going to address these multitude of challenges all at once. Instead, solutions that rely on dynamic data-driven techniques such as that envisioned by the Dynamic Data Driven Applications Systems (DDDAS) paradigm [6, 15] offer the most promise. DDDAS proposes an approach where data-driven models of a system are learned through dynamic instrumentation of the system, the models are simulated to conduct what-if analysis, and in turn these insights are used in a feedback loop to steer the system along the desired trajectory. In this chapter we apply the DDDAS approach to address these aforementioned challenges and present a just-in-time and performance-aware cloud-to-fog application migration approach that involves the following contributions:

- We present a data-driven modeling technique to estimate the performance of a cloud application on different hardware platforms subject to performance interference stemming from various co-located applications.

- We formulate server selection as an optimization problem that finds an apt server among multiple micro data centers to migrate an application to, so that its performance needs can be met while minimizing the deployment cost of the service provider.
- We describe INDICES (**IN**telligent **D**eployment for ub**I**quitous **C**loud and **E**dge **S**ervices), which is a framework that implements our DDDAS-based algorithms for online performance monitoring, performance prediction, network performance measurements, server selection and application migration.
- We show experimental results to validate our claims and evaluate the efficacy of the INDICES framework.

The rest of the chapter is organized as follows: Section 2 presents the system model and assumptions; Section 3 describes the problem formulation we address in this research; Section 4 delves into details of our solution including the design and implementation; Section 5 presents empirical results that validate our claims; Section 6 compares related work with our work; and finally Section 7 presents concluding remarks alluding to lessons learned and future work.

2 System Model and Assumptions

In this section, we formally describe the system model used in our study, and also introduce the assumptions we make for formulating the problem.

2.1 Components of Application Response Times

Typically, the total end-to-end latency t_{total} or response time experienced by cloud-hosted interactive applications consists of several parts as shown in Equation (1) below and which forms the motivation for our work :

$$t_{total} = t_{client} + t_{access} + t_{transit} + t_{datacenter} + t_{server} \tag{1}$$

where

- t_{client} is the processing delay at the client endpoint;
- t_{access} is the sum of inbound and outbound message transmission delays between the client and its nearest network access point;
- $t_{transit}$ is the sum of inbound and outbound communication delays between the network access point and the cloud data center;
- $t_{datacenter}$ is the communication delay from the data center front end (e.g., a web server and load balancer) to the target server in the data center that actually handles the request in both directions;
- t_{server} is the processing delay at the target server.

Among these costs, t_{client} and t_{access} cannot be controlled and managed by the cloud service provider. Also, since $t_{datacenter}$ is usually less than 1 ms [12], it is practically negligible. On the other hand, a cloud provider can control and manage $t_{transit}$ and t_{server}, both of which are key factors in meeting the response time requirements of interactive applications. Note that $t_{transit}$ is governed by the number of hops incurred by the application messages to traverse the wide area network to reach the cloud data center and for the responses to traverse back to the user.

2.2 Architectural Model

The work presented in this chapter is geared towards platform-as-a-service (PaaS) cloud providers who seek to meet service-level objectives (SLOs) of soft real-time applications, such as online gaming, augmented reality, or virtual desktop, by improving application response times via the exploitation of micro data centers (MDCs). In doing so, cost considerations and energy savings for the PaaS provider in operating and managing the resources beyond the traditional data centers are also critical issues while ensuring that such an approach provides an additional source of revenue to the PaaS provider. Revenue generation issues are, however, beyond the scope of this chapter.

Figure 1 depicts our architectural model that consists of a Centralized Data Center (CDC), owned by a PaaS cloud provider. The CDC is connected to a group of Micro Data Centers (MDCs), denoted by $M = \{m_1, m_2, \ldots, m_n\}$. These MDCs are deployed at the edge, and are either owned by the CDC provider or leased from an edge-based third party MDC provider. A leased MDC is assumed to be exclusively under the control of the CDC provider.[2] Once an MDC is leased, all its resources are considered to be part of the CDC provider, and hence customers of the CDC can be transparently diverted to the MDC using their CDC-based security credentials.

The CDC comprises a set of compute servers, H_{cdc}, that can be used to execute applications. The CDC also contains a global manager gm, which is responsible for detecting and mitigating global SLO violations. We assume that, for each MDC $m \in M$, there exist links to the CDC with a backhaul bandwidth of b_m. Each MDC m also comprises a set of compute servers, H_m, that can be allocated to the CDC for its operations at a specified cost. One of the hosts from H_m or a specially designated MDC host acts as the local manager (lm_m) for that MDC and is responsible for data collection, performance estimation, latency measurements and MDC-level decision making. This decision-making logic is deployed at the MDC by the CDC provider. For convenience, we let $H_{mdc} = \bigcup_{m \in M} H_m$ denote the set of all servers in the MDCs, and let $H_{total} = H_{mdc} \bigcup H_{cdc}$ denote the set of all servers from both the CDC and the MDCs. Table 1 lists the key notations in the architecture model.

[2] The sharing of MDCs across different CDC providers is not addressed in this work and forms a dimension of our future work.

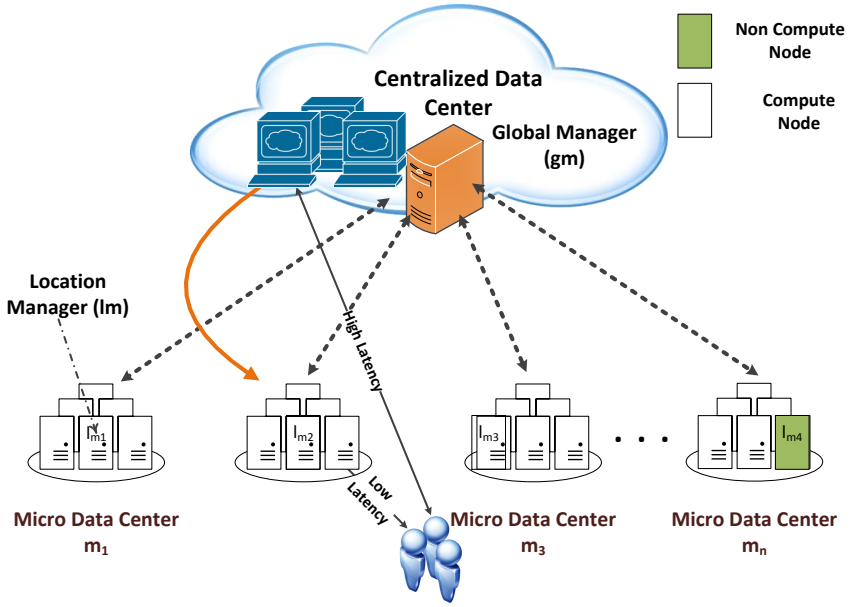

Fig. 1 Architectural Model

Table 1 Key notations in the architectural model

Notation	Description
CDC	Centralized Data Center, located at the cloud for execution of applications
M	Set of Micro Data Centers, located at the edge for execution of applications
H_{cdc}	Set of compute servers (hosts) located at the CDC
H_m	Set of compute servers (hosts) located at an MDC $m \in M$
H_{total}	Set of all compute servers (hosts) located at both the CDC and the MDCs
H_{mdc}	Set of all compute servers (hosts) located at the MDCs
b_m	Backhaul bandwidth to transfer data from the CDC to hosts of H_m

2.3 Application Model

We consider a set of latency-sensitive applications, denoted as $Apps$, that can be collaborative or single user and interactive or streaming in nature. Each application $a \in Apps$ is initially deployed in the CDC, with a set U_a of users, and is assumed to be containerized inside a virtual machine (VM). We assume that for a collaborative application a, its users are located in proximity of each other where they incur similar round trip latencies. These scenarios are common when we consider collaborative educational applications such as [9] where the users are a group of students working from a school library or a coffee shop, computer vision-based

applications in museum and stadium settings [19], or a single user system such as augmented reality-based assisted industrial troubleshooting [28] where image processing operations are performed in the cloud. Table 2 lists the key notations used in the application model.

2.3.1 Application Performance

Each application $a \in Apps$ can be hosted on any active host in CDC or MDC, i.e., $\eta \in H_{total}$ that provides virtualization using a hypervisor or virtual machine monitor (VMM), such as KVM [34] and Xen [2]. Let eed_a represent the expected execution duration for which the application will be used by the end-user clients. An interactive or streaming application comprises multiple individual interactions between the user and the application that we call *streaming steps*.

Each interactive or streaming step of application a consists of both a latency and an execution time. A step takes an estimated execution time $eet_{a,\eta}$ on host η; for collaborative applications, it indicates the time needed for all users to have completed that step. Section 4.2 discusses in detail a systematic way of estimating these per-step execution times. In addition, for each user $u \in U_a$ of the application, let $el_{a,\eta,u}$ represent the estimated round-trip network latency. Hence, if we denote by $ess_{a,\eta}$ the total expected response time of a streaming step, we have:

$$ess_{a,\eta} = eet_{a,\eta} + \max_{u \in U_a} el_{a,\eta,u} \tag{2}$$

For an application a, we define ϕ_a to be a bound on the acceptable response time for each interactive step of the application. Formally, the SLO for application a hosted on host η should satisfy:

$$ess_{a,\eta} \leq \phi_a \tag{3}$$

2.3.2 Migration Cost

During normal execution, some applications may suffer from performance degradation. We denote by PA the subset of $Apps$ running on the CDC that suffer from performance degradation. These impacted applications can be identified reactively by the end-user client, which notices missed deadlines using special instrumentation features supplied in the client-side "app" installed as part of the PaaS platform and notifies the CDC service. Alternatively, they can be identified proactively via a predictive decision based on the existing user profiles, where the system predicts that the users are likely to experience SLO violations if they had connected from their profiled location during a certain time period. Our goal is to minimize the number of SLO violations.

Table 2 Key notations in the application model

Notation	Description
$Apps$	Set of applications initially deployed on the CDC
PA	Subset of applications in $Apps$ that suffer from performance degradation
U_a	Set of users executing application $a \in Apps$
eed_a	Expected execution duration of application $a \in Apps$
$eet_{a,\eta}$	Estimated execution time of a streaming step of application $a \in Apps$ executed on host $\eta \in H_{total}$
$el_{a,\eta,u}$	Estimated round-trip latency experienced by user $u \in U_a$ from a streaming step of application $a \in Apps$ executed on host $\eta \in H_{total}$
$ess_{a,\eta}$	Total expected response time of a streaming step of application $a \in Apps$ executed on host $\eta \in H_{total}$
ϕ_a	Bound on acceptable response time for each streaming step of application $a \in Apps$
s_a	Size of the snapshot of application $a \in PA$ to be migrated from CDC to MDC
$ci_{a,\eta}$	Initialization cost of migrating application $a \in PA$ to host η
$transfer_{a,\eta}$	Transfer time for migrating application $a \in PA$ from CDC to host η in MDC and for initializing it
δ_a	Bound on acceptable transfer time for migrating and initializing application $a \in PA$

To do this, the system can decide to migrate those applications from the CDC to the hosts in the MDCs. Migrating (or transferring) an application incurs two costs: the cost to transfer a snapshot of the application on the new host, and the initialization cost to start the application on the new host.

For an application $a \in PA$ transferred to host $\eta \in H_m$, we denote by s_a the size of the snapshot of the application (independent of the host). It is transferred to host η using the corresponding MDC's backhaul bandwidth b_m. Furthermore, let us denote by $ci_{a,\eta}$ the initialization cost of migrating application a to host η before the application can start processing requests on the MDC host. Once the user-specific state has been transferred, there is minimal interaction between the CDC-based server and the MDC-based server for the remainder of the functioning of application a. In this chapter, we do not consider further consolidation of resources where applications migrate back to the CDC. The transfer time $transfer_{a,\eta}$ incurred while migrating application a from the CDC to host η of an MDC is therefore defined as follows:

$$transfer_{a,\eta} = \frac{s_a}{b_m} + ci_{a,\eta} \tag{4}$$

In general, to migrate the application to the new machine, we need the transfer duration to be small compared to the application's remaining expected execution duration eed_a. This is a necessary condition to motivate the use of MDC resources and for our solution to be relevant. To ensure this, we do not require sending entire images of the VM or the container from the CDC to MDC. Instead, we use a layered file system architecture at the MDC that is pre-populated with base images used at the CDC as described in Section 4.4. This assumption is realistic because we surmise that an MDC is either owned entirely or leased exclusively by the CDC provider.

We also ensure that the transfer duration is within a threshold δ_a defined by the application users before they start to observe improved response time. We concretize these requirements with the following constraint:

$$transfer_{a,\eta} \leq \delta_a \ll eed_a \tag{5}$$

Finally, another critical issue we must account for is that any migration of a new application from the CDC to an MDC should not violate the SLOs of the existing applications in that MDC. To capture this aspect, let $Apps_\eta$ represent the set of all applications currently running on an MDC host η. Then, for each application $b \in Apps_\eta$, we must verify that its response time bound remains satisfied, i.e., $ess_{b,\eta} \leq \phi_b$, after the migration of the new application.

3 Problem Statement and its Formulation

We now formally present the problem statement. Recall that our objective is to improve response times for cloud-hosted interactive applications that are experiencing performance degradation by migrating them to MDCs. To that end, we must address two key problems. First, we must have a systematic approach for understanding the causes of performance degradation and for determining if an application is impacted. Second, we must find an effective approach by which an application can be migrated from the CDC to an MDC without impacting existing MDC-based applications while minimizing the cost incurred by the cloud provider.

3.1 Performance Estimation Challenges

The performance of an application depends on several factors, including the workload, the hardware hosting platform, and co-located applications that cause performance interference [10, 17, 33]. It is thus important for any solution to account for all of these factors in order to accurately estimate an application's performance in both the CDC and the MDCs. Below, we describe the roles of these factors and the challenges in performance estimation.

3.1.1 Workload Estimation

For the cloud-hosted interactive applications of interest to us, we assume that the workload variation is not significant within a single user session of the service. However, different sessions may have different workloads. For example, in an image processing application, the quality and hence the size of the captured and relayed image may vary depending on different clients' mobile devices. Thus, we consider

each workload as a different application setting, which is reflected by the application-specific response time (as described in Section 2.3.1).

3.1.2 Hardware Heterogeneity

The CDC and MDCs may consist of heterogeneous hardwares and hence each application's performance can vary significantly from one hardware platform to another [17]. Therefore, we need an accurate benchmark of performance for each hardware platform.

3.1.3 Performance Interference

Server virtualization platforms such as KVM [34] and Xen [2] provide high degree of security, fault and environment isolations for applications running in virtualized containers, i.e., virtual machines (VMs). However, the level of isolation is inadequate when it comes to performance isolation even though the cloud providers have well-defined resource sharing mechanisms. This happens due to two primary reasons:

- *Presence of non-partitionable shared resources:* VMs can provide isolation guarantees by applying strict CPU reservations and static partitioning of disk and memory spaces. There are solutions available to limit the network bandwidth too. Yet, on-chip resources including cache spaces, cache and DRAM bandwidths, and interconnect networks are difficult to partition [26]. Recently, Intel has introduced Cache Allocation Technology [8] to partition the last level cache (LLC). However, it is still not widely used and cannot be applied to older generation servers. In addition, the presence of shared storage disk is a leading cause of performance degradation [48]. The load imposed on these shared resources by one application is detrimental to all the cache-, memory- and I/O-sensitive applications [42].
- *Hypervisor overhead:* The virtual machine monitor or hypervisor has its own overhead. In traditional hypervisors such as KVM and Xen, each virtual machine runs its own operating system which leads to overhead. In addition, the virtual CPUs (vCPUs) can be de-scheduled and virtual RAM can be swapped out without notification. This leads to performance anomalies. In recent years, Linux Container-based virtualization techniques such as Docker and LXC has grained transaction due to its low overhead. These resource sharking mechanisms share the kernel space from the host machine, which alleviates some of the performance concerns but do not provide the same level of performance and security isolations as the VMs [45]. In addition, to maximize the server utilization in shared clusters, cloud providers tend to overbook resources such as CPU cores. This precludes strict CPU reservations and leads to even the lower level caches (L1 and L2) getting shared. In addition, if the overbooked workload goes beyond the server capacity, contention takes place and the applications suffer from performance issues.

3.2 Optimization Problem Formulation

The objective of the framework is to assure the SLOs for all the identified applications in PA by migrating them to the MDC hosts, while minimizing the overall deployment cost.

3.2.1 Objective Function

To formalize the optimization problem, we define the following binary variables to indicate the decision for deploying the applications in the set PA onto the hosts in H_{mdc}.

$$x_{a,\eta} = \begin{cases} 1 & \text{if } a \in PA \text{ is deployed on } \eta \in H_{mdc} \\ 0 & \text{otherwise} \end{cases}$$

Thus, the total number of decision variables is $|PA| \times |H_{mdc}|$.

The total cost of deploying applications consists of two parts as shown in Equation (6), where the first part indicates a fixed cost incurred due to extending the lease of a host η (if the host is selected), and the second part indicates the cost of deploying an application a to host η, which includes both transfer and initialization costs.

$$C^{total} = \sum_{\eta \in H_{mdc}} C_{\eta}^{fixed} + \sum_{\eta \in H_{mdc}} \sum_{a \in PA} x_{a,\eta} \cdot C_{a,\eta}^{deploy} \tag{6}$$

Each MDC host η involves a monetary allocation cost as it is either leased or could be leased to other providers if owned by the centralized cloud. In addition, running servers involves operational costs, such as the need for power and cooling. Thus, the provider wants to use an MDC server for the shortest duration possible and hence the deployment cost depends on the duration for which the MDC server is used. This cost can be modeled as the extra duration for which the server has to be turned on due to the deployment of the applications in PA and it can be represented by a nonlinear function as shown in Equation (7), where the constant α_{η} denotes the cost gradient for powering on host η.

$$C_{\eta}^{fixed} = \alpha_{\eta} \cdot \max\left(0, \max_{a \in PA}(x_{a,\eta} \cdot eed_a) - \max_{b \in Apps_{\eta}} eed_b\right) \tag{7}$$

The other objective for the cloud provider is to select hosts which bear minimum transfer and initialization costs. This cost is represented as a linear function of transfer and initialization time, as shown in Equation (8), where the constant $\beta_{a,\eta}$ represents the cost gradient for transferring application a to host η.

$$C_{a,\eta}^{deploy} = \beta_{a,\eta} \cdot transfer_{a,\eta} \tag{8}$$

3.2.2 Optimization and Constraints

Using the decision variables and the objective function, we can now formulate the optimization problem as an Integer Non-Linear Program (INLP) as follows:

$$\text{minimize} \quad C^{total}$$

$$\text{subject to} \quad \sum_{\eta \in H_{mdc}} x_{a,\eta} = 1, \forall a \in PA \tag{9}$$

$$\sum_{\eta \in H_{mdc}} x_{a,\eta} \cdot ess_{a,\eta} \leq \phi_a, \forall a \in PA \tag{10}$$

$$\sum_{\eta \in H_{mdc}} x_{a,\eta} \cdot transfer_{a,\eta} \leq \delta_a, \forall a \in PA \tag{11}$$

$$ess_{b,\eta} \leq \phi_b, \forall b \in Apps_\eta, \forall \eta \in H_{mdc} \tag{12}$$

$$x_{a,\eta} \in \{0,1\}, \forall a \in Apps_\eta, \forall \eta \in H_{mdc} \tag{13}$$

The following explains the constraints in the above formulation based on the architectural and application models:

- Constraint (9) restricts each application $a \in PA$ to be deployed on only one MDC host.
- Constraint (10) enforces the response time constraint for each application $a \in PA$ (i.e., Inequality (3)).
- Constraint (11) enforces the deployment constraint for each application $a \in PA$ (i.e., Inequality (5)).
- Constraint (12) ensures the response time constraint for each existing application $b \in App_\eta$ in each MDC host $\eta \in H_{mdc}$.
- Constraint (13) restricts the decision variables to be binary.

Due to the NP-hardness of the above INLP formulation, we rely on a greedy-based heuristic to solve it. Section 4.5 describes the proposed heuristic for server selection.

4 Design of INDICES

We now present the design of our INDICES framework, which solves the optimization problem from Section 3.2. To that end, our solution depends on accurately and reliably estimating: (a) the execution time of the impacted application and network latencies suffered by its clients; (b) similar parameters for the already running applications on different hosts of different MDCs, which are then used in selecting the appropriate host to migrate an impacted application; and (c) the transfer time for migrating the state of the impacted application. This section describes the framework architecture and the detailed techniques for solving the optimization problem.

4.1 INDICES Architecture and Implementation

Before delving into the detailed techniques used to solve the optimization problem, we first present a high-level architecture of INDICES. Given the scale of the system, a centralized approach to performance prediction and cost estimation for every application hosted in the CDC/MDC and its clients is infeasible. Thus, we take a hierarchical approach, where individual MDCs with their local managers and the global manager of the CDC participate in a two-level decision making process as shown in Figure 1.

Figure 2 shows the local decision making part of INDICES. Each MDC is composed of a management node and several servers on which the applications execute within virtual machines. Each individual host in the system has a performance monitoring component that logs the data at the local manager lm_m. The local manager consists of a data collector, a latency estimator, a performance predictor and a cost estimator.

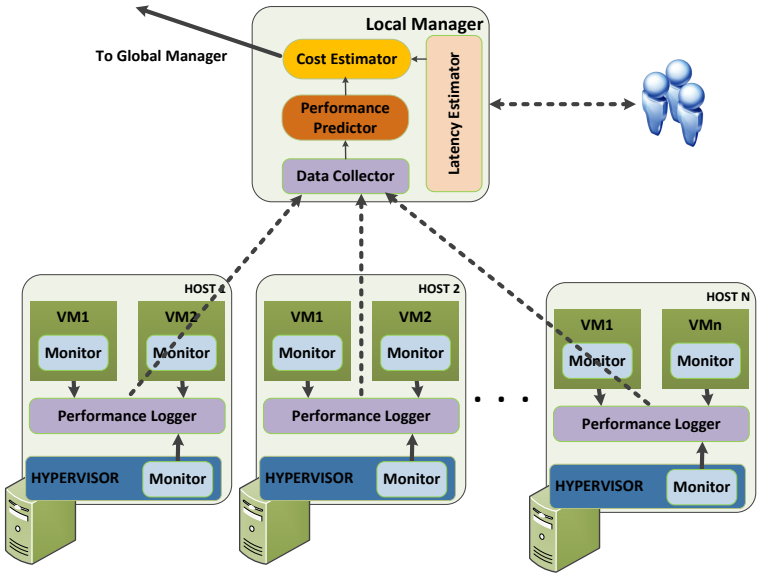

Fig. 2 Local Decision

The performance monitor instruments the host and collects system level metrics such as CPU, memory and network utilizations, as well as micro architectural metrics such as retired instructions per second (IPS) and cache misses. This information is periodically logged to the local manager for processing. The performance monitoring framework is based on the collectd [23] system performance statistics collection tool. To collect micro architectural performance metrics, we developed a Python plugin

for collectd using Linux perf. This plugin detects if the hardware platform is known, and accordingly executes code that collects hardware specific performance counter statistics. The information is then forwarded to the lm_m using AMQP [51] message queuing protocol. The lm_m runs a server developed in the Go programming language, which persists the data in the InfluxDB database, designed specifically for time-series data.

4.2 Execution Time Estimation

The constraints in the optimization problem require an accurate understanding of the predicted execution time duration of an application if it were to execute at an MDC, as well as the execution times of the existing applications executing on the hosts of the MDCs. Hence, we build an application's expected performance profile and in turn its interference profile [30] when co-located with other applications on different hardware platforms given the hardware heterogeneity across the CDC and MDCs. Although prior efforts [42, 56, 58] have used retired instructions per cycle (IPC) or last-level cache (LLC) miss rate as the performance indicators, Lo et. al [40] have shown the limitations of these metrics for latency-sensitive applications. Thus, we consider *execution time* as the primary indicator of performance.

The *interference profile* of an application [38, 42, 55, 59] is a property that identifies: (a) the degree to which the application will degrade the performance of other running applications on the host – known as *pressure* – and (b) how much the application's own performance suffers due to interference from other applications – known as *sensitivity*. The performance degradation of an application depends, to varying degrees, on different system components and architectures, and other co-located applications. Several prior efforts have used pairwise application execution to estimate their sensitivity and pressure [38, 42, 55, 59]. However, these solutions are not viable for a data center given the significantly large number of hosted applications. Some other efforts [56] pause non-critical applications to measure pressure and sensitivity of live applications, which may not be a realistic solution.

Thus, for a given application a, its performance on a host with hardware configuration w is modeled by Equation (14), where Y is the execution time, X is a vector of system-level metrics that quantify the state of the host, and the function f_a^1 models the relation between the state of the host machine and the performance of the application a. Moreover, the system-level information needed is obtained through Equation (15), which depicts through function f_a^2 the change in the state of the host with hardware configuration w if application a were to be hosted on it. Equation (15) provides an indirect measure of performance interference, since its output can be used to calculate the change in execution time of an already running application by plugging the new state vector X_w^{new} into Equation (14) and solving it for each running application.

$$Y = f_a^1(X_w) \tag{14}$$

$$X_w^{new} = f_a^2(X_w^{old}) \tag{15}$$

Table 3 Server Architectures

Server	Hardware Model	Sockets/Cores/ Threads/GHz	L1/L2/L3 Cache (KB)	Mem Type-/MHz/GB	Memory Bandwidth	Count
A	i7 870	1/4/2/2.93	32/256/8192	DDR3/1333/16	64 *(UNC_IMC_NORMAL_READS.ANY + UNC_IMC_WRITES.FULL.ANY) / time in sec	2
B	Xeon W3530	1/4/2/2.8	32/256/8192	DDR3/1333/6	64 *(UNC_IMC_NORMAL_READS.ANY + UNC_IMC_WRITES.FULL.ANY) / time in sec	1
C	Core2Duo Q9550	1/4/1/2.83	32/6144/-	DDR2/800/8	64 * BUS_TRANS_MEM.ALL_AGENTS *1e9 * CPUFrequency / CPU_CLK_UNHALTED.CORE	1
D	Opteron 4170HE	2/6/1/2.1	64/512/5118	DDR3/1333/32	64 * SamplingPeriod * DRAM_ACCESSES_PAGE.ALL / time in sec	9

Another required step is to identify the right system-level metrics to use. Previous works [16, 17, 30] have identified several sources of interference including caches, prefetchers, memory, network, disk, translation lookaside buffers (TLBs), and integer and floating point processing units. Both Intel and AMD architectures provide hardware counters to monitor the performance of micro-architectural components. However, not all the sub-components can always be monitored. Moreover, the list of available counters is significantly smaller for older generation servers. Due to these constraints and driven by the need to support a broadly applicable solution, we select the following host metrics for performance monitoring:

- **System Metrics:** CPU utilization, memory utilization, network I/O, disk I/O, context switches, page faults.
- **Hardware Counters:** Retired instructions per second (IPS), cache utilization, cache misses, last-level cache (LLC) bandwidth and memory bandwidth. The bandwidth metrics are not directly available and the counters can vary from one hardware to other. In our analysis, we found that the LLC bandwidth and memory bandwidth are highly correlated and hence we select the memory bandwidth and not LLC bandwidth due to its easier availability on different architectures and versions. Table 3 lists the hardware counter-based equations for calculating memory bandwidth, which are derived from [1, 20].
- **Hypervisor Metrics:** Scheduler wait time, scheduler I/O wait time, scheduler VM exits. These metrics are the summation for all the executing virtual machines for the KVM hypervisor.

By applying standard supervised machine learning techniques on the collected metrics, we estimate the functions in Equations (14) and (15) using the following sequence of steps:

1. **Feature Selection:** Feature selection is the process of finding relevant features in order to shorten the training times and reduce errors due to over-fitting. We

have adopted the Recursive Feature Elimination (RFE) approach using Gradient Boosted Regression Trees [21]. We perform RFE in a cross-validation loop to find the optimal number of features that minimizes a loss function (mean squared error in our case).
2. **Correlation Analysis:** To remove the linearly dependent features, correlation analysis is required. This step further reduces the training time by decreasing the dimensions of the feature vector. We use the Pearson Coefficient to eliminate highly dependent metrics with a threshold of ±0.8.
3. **Regression Analysis:** In this step, curve fitting is performed using ensemble methods. We have used standard off-the-shelf Gradient Tree Boosting method, which is widely used in the areas of web page ranking and ecology. The primary advantage of this method lies in its ability to handle heterogeneous features and its robustness to outliers.

The performance estimation of the applications consists of two phases: (1) Offline Phase, and (2) Online Phase. The offline phase occurs at the CDC to find estimators, while the online phase is performed by the local manager (lm_m) of the MDCs to estimate the performance of the target application and also to estimate the performance degradation of the running applications. The two phases are described below.

4.2.1 Offline Phase

Whenever the data center receives a request for migrating an application that has not been profiled, it is benchmarked on a single host with a given hardware configuration and then co-located with other applications to develop its interference profile. However, since the number of profiling configurations can be huge, we select a uniformly distributed subset of possible co-location combinations for profiling. The estimators can be found either by following the three steps above listed or by choosing an existing estimator of some application based on similarity between the projected performance and the actual performance. We use a hybrid approach, which first predicts the performance of the new application and its interference profile using estimators of an existing application for the same hardware specifications. If the difference between the measured performance and the estimated performance are within a pre-defined threshold, then we consider the new application to be similar in performance to the existing application. Among all such similar applications, the estimator of the application with the least error is selected for all MDC hardware configurations. This saves profiling time and cost. However, if there is no match, the application profile is developed by performing feature pruning followed by model fitting on each unique hardware platform maintained by the data center.

4.2.2 Online Phase

The learned models are then exported and forwarded to the MDC local manager lm_m for the available hardware platforms in the MDC to estimate the performance

of any application to be deployed in the MDC. Since each MDC is small in size and typically illustrates limited heterogeneity in the supported hardware, the number of estimation models will be small. On receiving a request from the global manager gm, the local manager lm_m estimates the performance of an application by feeding the estimator with presently logged data set using estimation function 14. The pressure on the existing set of applications App_η on host η is calculated by first applying Equation (15) on the target application and then Equation (14) for the existing applications.

4.3 Network Latency Estimation

The constraints of the optimization problem also require an accurate understanding of the network latencies incurred by the clients, specifically the worst among all the clients of each application. This information is needed for identifying the appropriate host in the appropriate MDC to which an impacted application can be migrated such that it satisfies the SLOs for the worst suffering client while not unduly affecting the existing applications of the MDC hosts.

Thus, estimating the latency to different MDC servers is another key component for achieving the targeted SLOs. To that end, we must determine the clients who suffer SLO violations from Equation (3). In each client, the instrumented "app" that is installed by the user as part of the client application periodically reports to gm the application response time it is observing. To not overwhelm the gm, such data logging needs not occur directly on the gm; instead, it can be logged on an ensemble of servers that then report to the gm, or the application server can itself gather data and forward the information when SLO violations occur.

Since there could be multiple MDC choices to migrate an impacted application to, the first step in our algorithm requires reducing the target set of MDCs for latency estimation to decrease the load and amount of time for server selection. To that end, we use the logged performance data from the clients to extract its IP address in order to determine the closest MDCs to that client. The extracted client IP address may not be accurate since often internet users have private addresses and the reported external address is that of the network router or one from the pool of network provider's addresses in case the connection is via a cellular network. However, this information is sufficient, as we use the client location to reduce the set of MDCs we need to query. The client's geo-location and consequently its region is derived from the IP address.

The next step is measuring the latencies to the nearby MDCs. To obtain a reliable latency estimate, we use HTTP-based and TCP socket-based latency measurement techniques for HTTP-based and plain TCP-based cloud applications, respectively. We can easily add additional protocols to this list based on the protocol used. Subject to the collected information, the gm forwards to the client app a list of "nearby" MDC gateway servers that are also the local managers lm_m, each hosting a server for the purpose of latency measurement. The client then posts n requests to each lm_m with

a file that it typically posts to the cloud for processing (e.g., an image for image processing application) and also the average size of the response it receives from the application. The server responds with a response for the same number of bytes. For each of the n interactions, the client records the elapsed time and thus measures $t_{client}+t_{access}+t_{transit}$. The client app selects the SLO latency (e.g., 95^{th} percentile) from the n latencies for each lm_m and reports it to gm. This approach also accounts for the delay due to bandwidth size as we transfer the actual request data instead of a ping. It is similar to the *speed test* for measuring the download/upload speeds of an internet provider.

4.4 State Transfer Estimation

The final constraint of the optimization problem requires estimating the cost of the state transfer. The local managers calculate the state transfer cost using Equation (4) and use it in local decision making. Once the gm selects a host for migrating an application, the application's state has to be transferred before the clients can be switched to the new server location. In this regard, there exist several solutions available for WAN-scale virtual machine migration [7, 47, 50, 54]. We leverage the cloud virtual disk format such as qcow2 features for WAN migration. The VM disk is composed of a base image and can contain several overlays on top of it for change sets. The VM overlay when combined with the base image constructs the VM that needs to run for serving the clients.

This base image can contain just an operating system such as Ubuntu or an entire software stack such OpenCV for image processing. The base image is assumed to be present on MDC hosts to save on migration costs and can be shared by multiple VMs. For the target application, overlays are created using external snapshots. The VM overlay is the state that gets transferred to the host and is synthesized with the base image for execution.

Once the application starts running, it informs the gm and all the application clients are redirected to the new application URL. This happens for custom clients by forwarding the new location to the clients which can then use the new URL for processing. However, for browser-based clients, the communication with the gm occurs via application server due to cross-domain restriction and the existing application issues HTTP-redirect to the new location. In the future, we will enhance our solution to support live migration of VMs using solutions such as Elijah cloudlet [27] or the recently introduced Docker Linux container's live migration feature [44].

4.5 Solving the Optimization Problem at Runtime

The final piece of the puzzle is to solve the optimization problem described in Section 3.2. The problem cannot be solved offline due to the changing dynamics of

the system. Moreover, due to the non-linearity and NP-hardenss of the problem, we employ a heuristics-based algorithm as described in Algorithm 1 to solve it efficiently in an online setting. The algorithm selects aptly a suited server in an MDC while minimizing the overall deployment cost for the entire system.

Algorithm 1 Deployment Server Selection

1: **Input**: set $Apps$ of all applications running on the CDC
2: **Output**: a server on an MDC for migrating each application $a \in PA \subseteq Apps$ that experiences performance degradation
3: **for all** $a \in Apps$ **do**
4: $ess_{a,cdc} \leftarrow EstTotalExecTime(a, CDC)$
5: **if** $ess_{a,cdc} > \phi_a$ **then**
6: $PA.insert(a)$
7: **if** $PA = \emptyset$ **then return** ▷ Do nothing
8: **for all** $a \in PA$ **do**
9: $eed_a \leftarrow GetExpectedExecutionDuration(a)$
10: $clientLoc \leftarrow GetLocation(U_a)$
11: $nearbyMDCs \leftarrow FindNearbyMDCs(clientLoc)$
12: **for all** $m \in nearbyMDCs$ **do**
13: $H_m \leftarrow GetServerList(m)$
14: **for all** $\eta \in H_m$ **do**
15: $transfer_{a,\eta} \leftarrow EstTransDur(\eta, a)$
16: **if** $transfer_{a,\eta} > \delta_a$ **then**
17: **skip** η ▷ Constraint violated
18: $perf_{a,\eta} \leftarrow PredictPerfInterf(\eta, a)$
19: **for all** $b \in App_\eta$ **do**
20: $ess_{b,\eta} \leftarrow EstTotalExecTime(b, \eta, perf_{a,\eta})$
21: **if** $ess_{b,\eta} > \phi_b$ **then**
22: **skip** η ▷ Constraint violated
23: $ess_{a,\eta} \leftarrow EstTotalExecTime(a, \eta, perf_{a,\eta})$
24: **if** $ess_{a,\eta} > \phi_a$ **then**
25: **skip** η ▷ Constraint violated
26: $C^{total}_{a,\eta} \leftarrow EstTotalCost(transfer_{a,\eta}, eed_a)$
27: $[C^{total}_{min}, \eta_{min}] \leftarrow \min_\eta \{C^{total}_{a,\eta}\}$
28: migrate application a to server η_{min}

There are two phases in the algorithm. In the first phase, we identify the set PA of applications suffering from SLO violations (Lines 3–6). In the second phase, we select a suitable server for migrating each of these applications (Lines 8–28), unless the set PA is empty, in which case the algorithm simply returns (Line 7). Otherwise, for each application $a \in PA$, we find the locations of the clients of the application (Line 10), which are used to perform a lookup for the nearby MDCs (Line 11). We then identify the server within the nearby MDCs that provides the best performance. This step is carried out in parallel across all the nearby MDCs (for-loop starting from Line 12).

For each such server η, we first check the deployment constraint as shown by Constraint (11) (Lines 15–17). We then predict the performance interference of

application a were it to execute on the server (Line 18). This interference is used to check the response times (Constraint (12)) for the existing set App_η of applications on that server (Lines 19–22), as well as the response time (Constraint (10)) for application a itself (Lines 23–25). Finally, we calculate the total cost according to Equation (6) if all of these constraints can be met (Line 26). The server with the minimum cost is then selected across all identified MDCs (Line 27), and the application is migrated to the selected server and clients are redirected to the migrated application.

5 Experimental Validation

In this section, we present experimental results for validate the INDICES framework in the context of a latency sensitive application use case.

5.1 Experimental Setup

Table 3 illustrates the hardware platforms and their counts used in our experiments. The CDC uses Openstack cloud OS version 12.0.2 where the guests receive their own public IP addresses. The MDC servers are managed directly by libvirt virtualization APIs and the guests communicate via port forwarding on the host. Each machine has Ubuntu 14.04.03 64-bit OS, QEMU-KVM hypervisor version 2.3.0 and libvirt version 1.2.16. Guests are configured with 2 GB memory, 10 GB disk, Ubuntu 14.04.03 64-bit OS and either 1 or 2 vCPUs. Since we are not concerned with VM migration within a CDC, we do not depict the CDC heterogeneity.

As described in Section 4.2, to preclude profiling every new application on all the hardware, we need initial training data. We use PARSEC, Splash-2 [5] and Stream [43] benchmarks to generate the training data. PARSEC and Splash-2 target chip-multiprocessors that typically comprise the modern day data centers, and provide a rich set of applications with different instruction mix, cache and memory utilization, needed for stressing different system subcomponents. We select 20 tests from these benchmarks for data generation and validation. We also use the Stream benchmark that specifically targets cache/memory bandwidth, which is one of the key sources of performance interference.

Due to lack of access to servers in different geographical regions, we use the network emulation tool, *netem*, and hierarchy token bucket based traffic control, *tc-htb*, for emulating the desired network latencies from the client to the CDC and different MDCs.

5.2 Application Use Case

We use an image processing application to validate the efficacy of our framework. The application performs feature detection, which is a critical and expensive part of any of the computer vision problem such as object detection, facial recognition [24], etc. We use the well-known Scale Invariant Feature Transform (SIFT) [41] to find the scale and rotation independent features. We consider an augmented reality application as the use case, where the client device is capturing video frames at a continuous rate. The application augments information to the captured frame by processing the frame. We use a Minnowboard Turbot as the client device connected to a webcam that continuously captures frames at a rate of 5 frames per second (fps). The frame resolution is 640x360 pixels and average frame size is 56 KB. The server comprises a Python-based application that receives frames over a TCP socket, processes it, and responds with the identified features along with the processing time. The client expects to receive a response before capturing the next frame, implying that 200 ms is the deadline for the application. Although our use case considers the performance for a single client connected to the cloud-hosted application, it can easily be extended to multiple clients residing in a similar latency region.

When the image processing application is submitted for hosting in our cloud, we execute it on different hardware platforms in isolation to find its base execution times. For hardware platforms A, B, C, D shown in Table 3, the base execution times are measured to be 86, 91, 146, 157 ms, respectively. Table 4 displays the emulated ping latency from this client to CDC or different MDCs in the same region as the client. The table also lists their server composition, and the measured 95^{th} percentile network latency while sending TCP/IP and HTTP post requests of 56 KB size and receiving a response of size less than 1 KB. The expected duration for which the client needs to perform the image processing, eed_a, is set as 1 hour and the SLO is set to 95%.

Table 4 CDC and MDC setup for application use case

MDC	Distance	Ping latency (±20%)(ms)	TCP latency(ms)	HTTP latency(ms)	Servers
H_1	1 hop	<1	2	6	1C + 1D
H_2	2 hops	5	14	28	1A + 2D
H_3	Multi hops	20	54	96	1B + 2D
H_4	Multi hops	30	76	142	1A + 3D
H_5	Central	50	127	220	1D

5.3 Evaluating the Performance Estimation Model

We first benchmark our use case application on hardware platform D in order to develop its performance estimators. The threshold to discern applications with similar interference performance profile, as described in Section 4.2.1, is set to 10% error. However, as illustrated in Figure 3, none of the existing applications meets the criteria. Thus, we decide not to use any of the existing estimators for the use case application and benchmark the application on all hardware configurations to develop their estimators. With this approach, we find that the mean estimation error for our use case application is less than 4% on all the platforms with low standard deviations as depicted in Figure 4. We can account for this estimation error in our response time constraint (Equation (3)) for stricter SLO adherence.

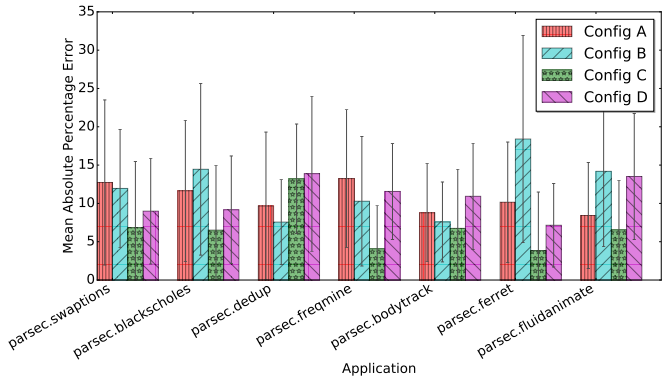

Fig. 3 Estimation of SIFT Profile Similarity with PARSEC Benchmark

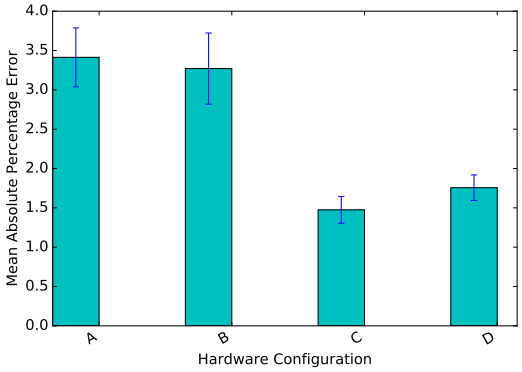

Fig. 4 SIFT Application Performance Estimation Error

5.4 Evaluating the Server Selection Algorithm

We compare our server selection algorithm against two other approaches based on minimum number of hops and least loaded server (among reachable MDCs), respectively. From Table 4, we observe that the minimum hop is 1. There are 2 servers in the minimum hop MDC H_1 with hardware configuration types C and D. We create interference load on both servers but ensure that the total load on the server does not exceed its capacity in terms of memory and vCPUs to eliminate unrealistic performance deteriorations. For the least-loaded server algorithm, we consider the server with least existing allocated resources, i.e., containing only a single VM. We do not consider servers with no existing load as it results in acquiring a new server and thus causing additional cost to the service provider. In this case, we find the server of hardware type D in MDC H_4 to be least loaded. Applying SLO from Equation (3) and our server selection algorithm, INDICES finds 2 servers of type A and D from MDC H_2 and one server of type B from MDC H_3.

Figure 5 shows the response time comparison of each of the suitable servers found by INDICES against the least loaded server for the expected duration eed_a (one hour) of the application. We observe that, in this scenario, the least loaded server has 100% SLO violation because of the network latency. However, the servers found by INDICES meet the SLO 100%, 99.38% and 98.94%, respectively, which are all well over the target SLO of 95%. Figure 6 shows the corresponding response time for the minimum hop algorithm. The 2 servers found by the algorithm meet the SLO only 66.64% and 60.64% of times due to performance interference.

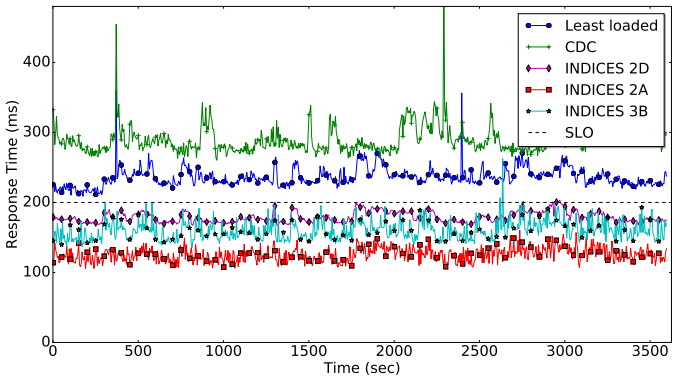

Fig. 5 INDICES vs Least Loaded Server Selection

Applying Algorithm 1 further, INDICES finds the server of type B from MDC H_3 to be most suitable, since our objective is to select a server with the minimum cost to the service provider if it can meet the SLO. Thus, it prefers a server which already has an application that is going to run longer and has better bandwidth from the CDC server for migration. Figure 7 compares 3 migration scenarios: (a) an overlay

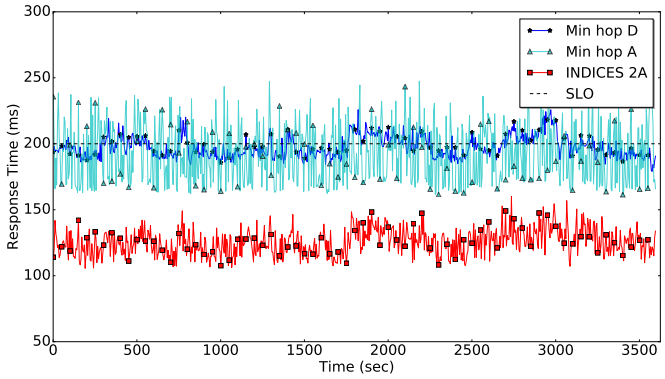

Fig. 6 INDICES vs Minimum Hop Server Selection

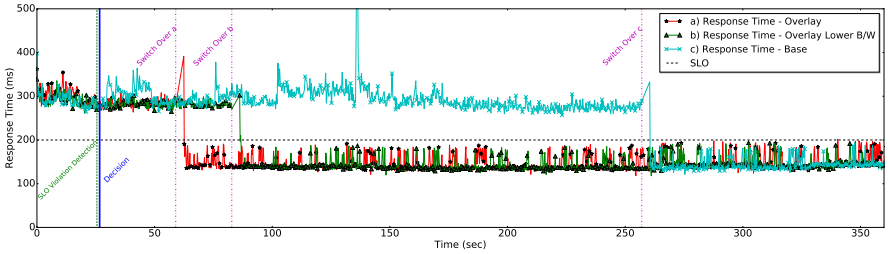

Fig. 7 Application Switch-Over Performed by INDICES under 3 Different Scenarios

with the software stack already present on the target server and the bandwidth is 10 Mbps; (b) same as the previous scenario but with a bandwidth of 1 Mbps; (c) overlay is not present on the target server and the compressed file of size 938 MB has to transferred over 10 Mbps bandwidth. In all the scenarios, the application overlay and configuration files have to be transferred and the application has to be initialized. We observe that the server selection takes $\approx$ 1 second, but the migration and initialization take 32 seconds, 56 seconds and 190 seconds, respectively, for scenarios (a), (b) and (c). Thus, the overlay-based image transfer should be the preferred methodology whenever applicable.

6 Related Work

In this section, we compare and contrast our work with related work along three dimensions: network latency-based server selection, performance interference-based server selection and performance-aware edge computing. Unlike our work, our survey has found that existing works seldom consider all dimensions holistically.

6.1 Network Latency-based Server Selection

DONAR [53] addresses the global replica selection problem using a decentralized, selection algorithm where the underlying protocol solves an optimization problem that takes into account client performance and server load. CloudGPS [18] is a server selection scheme that considers network performance, inter-domain transit traffic and server workload for decision making. This work also reduces the network distance measurement costs. Dealer [29] targets geo-distributed, multi-tier and interactive applications to meet their stringent deadline constraints by monitoring individual component replicas and their communication latencies, and selects the combination that provides the best performance. Kwon et al. [39] applied network latency profiling and redundancy for cloud server selection while suggesting using cloudlets. We contend that these efforts consider simplistic models of server workload and their impact on performance, and do not cater to edge resource management.

6.2 Performance Interference-aware Server Selection

Paragon [17] identified the sources of interference that impact application performance and developed micro benchmarks for heterogeneous hardware. The system benchmarks applications and classifies them to find collocation patterns for scheduling. Sherlock [32] developed approach to estimate the performance interference effect due to last level cache contention in containerized environment. The system profiles application and defines a metric *IScore*, that measures the degree of performance degradation of the application. It does not rely on the hardware counter of the system, and does not take into account the heterogeneity in the hardware platform.

MEDEA [25] allows the users to define anti-affinity application constraints, which captures resource interference characteristics of the applications. These constraints utilized by the server scheduler for optimizing placement decisions of the services to be deployed in the cluster. SMiTe [59] designed rulers for estimating sensitivity and degree of contention between applications when they are collocated. Bubble-Flux [56] assures QoS for latency-sensitive applications by dynamic interference profiling of shared hardware resources and collocating latency-sensitive applications with batch applications. These works, however, do not apply to virtualized data centers where the hypervisor places its own overhead on the resources and impacts performance. Moreover, our framework requires virtualized environments to support migration of applications on heterogeneous platforms. DeepDive [46] first identifies an abnormal behavior using a warning system and employs an interference analyzer by cloning the target VM and running synthetic benchmarks. Such an approach can be a costly runtime operation.

Our prior work [10] designed a performance interference-aware resource management framework that benchmarks applications residing in virtual machines and applies a neural network-based regression mechanism that estimates a server's performance interference level. However, hardware heterogeneity and per application

performance were not considered. Heracles [40] mitigates performance interference issues for latency-sensitive applications by partitioning different shared resources. However, partitioning for resources, such as memory bandwidth is still not available, and moreover, cache partitioning is only available on newer hardware which cannot be applied to existing hardware.

6.3 Performance-aware Edge Computing

Zhou et al. [60] described a multi attribute decision analysis algorithm to offload tasks amongst mobile ad-hoc network, cloudlet and public cloud. Their work performs cost estimation considering execution time, power consumption, bandwidth and channel conjunction level which is utilized by the decision making algorithm. The approach utilizes ThinkAir [37] for offloading the tasks. However, they target only Java-based tasks and the solution is not catered to latency-sensitive applications such as those targeted by us.

Fesehaye et al. [22] described a design to select between cloudlets and central cloud server for interactive mobile cloud applications based on the number of hops, mobility and latency. SEGUE [57] is an edge cloud migration decision system that applies state-based Markov Decision Process (MDP) model incorporating network and server states. Both the approaches have not been evaluated on real systems and the results are only simulation-based.

SmartRank. [49] is a tool for offloading facial recognition from mobiles to cloudlets, and the scheduling is performed based on the round trip time and CPU utilization. In our approach, we optimize the cost to the cloud provider while maintaining the end user SLOs.

7 Conclusions

This chapter presents INDICES, a framework for dynamic cloud resource management that exploits the available edge/fog resources in the form of micro data centers, which are used to migrate cloud-hosted applications closer to the clients so that their response times can be improved. In doing so, the INDICES framework ensures that existing edge-deployed services are not unduly impacted in terms of their performance nor are the operational and management costs for the cloud provider overly affected. These objectives are met using an online optimization problem, which is solved using a two-level cooperative and online process between system-level artifacts we have developed and deployed at both the micro data centers and centralized cloud data center. Our experimental results support the claims by showing the efficacy of the INDICES framework using a realistic application use case.

This work has opened up many new challenges and directions, which form our future work. These challenges are presented below:

- **Lack of benchmarks:** There is a general lack of open source and effective benchmarking suites that researchers can use to conduct edge/fog computing studies.
- **Collecting metrics under hardware heterogeneity:** The plethora of deployed hardware configurations with different architectures and versions makes it hard to collect various performance metrics. Modern architectures are making it easier to collect more fine-grained performance metrics, but much more research is needed in identifying effective approaches to control the hardware and to derive the best performance out of them.
- **Workload consolidation and migration across MDCs:** In this work, once an application is migrated to an MDC, it will complete its operation until termination. Our future work will consider dynamic server consolidation across MDCs and CDCs.
- **Reconciling application state:** We have assumed that once the application state is transferred to the MDC, there is no additional state that accumulates at the CDC. However, for a broader set of applications, not all application states may be transferrable to the MDC and may have to be reconciled periodically with the CDC, which gives rise to interesting consistency versus availability tradeoffs.
- **Distributed user base:** We have assumed that all distributed users of an interactive applications are located in close proximity to each other. However, for applications such as online gaming, this assumption may not hold for which additional research will be necessary.
- **Energy savings and revenue generation:** In this work, we did not discuss revenue generation issues stemming from the use of edge resources. Moreover, energy savings is only indirectly referred to through our experimental results. Addressing these limitations forms dimensions of our future work.
- **Shared micro data centers:** We have assumed that an MDC is exclusively controlled by a CDC provider. In the future, it is likely that MDC providers may lease their resources to multiple different CDCs. Additional research is needed to address situations where MDCs are shared including those that address security and isolation guarantees.

All scripts, source code, and experimental results for INDICES are available for download from an extended framework we recently developed called FECBench [4], which is available from `https://github.com/doc-vu/fecbench`.

Acknowledgments

This work is supported in part by the AFOSR DDDAS FA9550-13-1-0227 and FA9550-18-1-0126, and NSF US Ignite CNS 1531079. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of AFOSR and NSF.

References

1. Detecting memory bandwidth saturation in threaded applications. URL https://software.intel.com/en-us/articles/detecting-memory-bandwidth-saturation-in-threaded-applications
2. Abels, T., Dhawan, P., Chandrasekaran, B.: An overview of xen virtualization. Dell Power Solutions **8**, 109–111 (2005)
3. Bahl, V.: Cloud 2020: Emergence of micro data centers (cloudlets) for latency sensitive computing (keynote). Middleware 2015 (2015)
4. Barve, Y., Shekhar, S., Khare, S., Bhattacharjee, A., Kang, Z., Sun, H., Gokhale, A.: FECBench: A Lightweight Interference-aware Approach for Application Performance Modeling. In: IEEE International Conference on Cloud Engineering (IC2E), pp. 211–221. Prague, Czech Republic (2019)
5. Bienia, C., Kumar, S., Singh, J.P., Li, K.: The parsec benchmark suite: characterization and architectural implications. In: 17th international conference on Parallel architectures and compilation techniques, pp. 72–81. ACM (2008)
6. Blasch, E., Ravela, S., Aved, A.: Handbook of Dynamic Data Driven Applications Systems. Springer (2018)
7. Bradford, R., Kotsovinos, E., Feldmann, A., Schiöberg, H.: Live wide-area migration of virtual machines including local persistent state. In: 3rd international conference on Virtual execution environments, pp. 169–179. ACM (2007)
8. Cache allocation technology improves real-time performance. http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/cache-allocation-technology-white-paper.pdf
9. Caglar, F., Shekhar, S., Gokhale, A., Basu, S., Rafi, T., Kinnebrew, J., Biswas, G.: Cloud-hosted Simulation-as-a-Service for High School STEM Education. Elsevier Simulation Modelling Practice and Theory: Special Issue on Cloud Simulation **58**(2), 255–273 (2015). URL http://dx.doi.org/10.1016/j.simpat.2015.06.006
10. Caglar, F., Shekhar, S., Gokhale, A., Koutsoukos, X.: An Intelligent, Performance Interference-aware Resource Management Scheme for IoT Cloud Backends. In: 1st IEEE International Conference on Internet-of-Things: Design and Implementation, pp. 95–105. IEEE, Berlin, Germany (2016)
11. Caglar, F., Shekhar, S., Gokhale, A., Koutsoukos, X.: Intelligent, performance interference-aware resource management for iot cloud backends. In: 2016 IEEE First International Conference on Internet-of-Things Design and Implementation (IoTDI), pp. 95–105. IEEE (2016)
12. Choy, S., Wong, B., Simon, G., Rosenberg, C.: The brewing storm in cloud gaming: A measurement study on cloud to end-user latency. In: Proceedings of the 11th annual workshop on network and systems support for games, p. 2. IEEE Press (2012)
13. Chun, B.G., Ihm, S., Maniatis, P., Naik, M., Patti, A.: Clonecloud: elastic execution between mobile device and cloud. In: Proceedings of the sixth conference on Computer systems, pp. 301–314. ACM (2011)
14. Cuervo, E., Balasubramanian, A., Cho, D.k., Wolman, A., Saroiu, S., Chandra, R., Bahl, P.: Maui: making smartphones last longer with code offload. In: Proceedings of the 8th international conference on Mobile systems, applications, and services, pp. 49–62. ACM (2010)
15. Darema, F.: Dynamic data driven applications systems: A new paradigm for application simulations and measurements. In: International Conference on Computational Science, pp. 662–669. Springer (2004)
16. Delimitrou, C., Kozyrakis, C.: ibench: Quantifying interference for datacenter applications. In: Workload Characterization (IISWC), 2013 IEEE International Symposium on, pp. 23–33. IEEE (2013)
17. Delimitrou, C., Kozyrakis, C.: Paragon: QoS-aware Scheduling for Heterogeneous Datacenters. In: ACM SIGPLAN Notices, vol. 48, pp. 77–88. ACM (2013)

18. Ding, C., Chen, Y., Xu, T., Fu, X.: Cloudgps: a scalable and isp-friendly server selection scheme in cloud computing environments. In: IEEE 20th International Workshop on Quality of Service, p. 5. IEEE Press (2012)
19. Drolia, U., Guo, K., Gandhi, R., Narasimhan, P.: Edge-caches for vision applications. In: Edge Computing (SEC), IEEE/ACM Symposium on, pp. 91–92. IEEE (2016)
20. Drongowski, P.J., Center, B.D.: Basic performance measurements for amd athlon™ 64, amd opteron™ and amd phenom™ processors. AMD whitepaper **25** (2008)
21. Elith, J., Leathwick, J.R., Hastie, T.: A working guide to boosted regression trees. Journal of Animal Ecology **77**(4), 802–813 (2008)
22. Fesehaye, D., Gao, Y., Nahrstedt, K., Wang, G.: Impact of cloudlets on interactive mobile cloud applications. In: Enterprise Distributed Object Computing Conference (EDOC), 2012 IEEE 16th International, pp. 123–132. IEEE (2012)
23. Forster, F.: Collectd - The System Statistics Collection Daemon. `http://collectd.org` (2017)
24. Forsyth, D.A., Ponce, J.: Computer vision: a modern approach. Prentice Hall Professional Technical Reference (2002)
25. Garefalakis, P., Karanasos, K., Pietzuch, P., Suresh, A., Rao, S.: Medea: Scheduling of long running applications in shared production clusters. In: Proceedings of the Thirteenth EuroSys Conference, EuroSys, vol. 18, p. 4 (2018)
26. Govindan, S., Liu, J., Kansal, A., Sivasubramaniam, A.: Cuanta: quantifying effects of shared on-chip resource interference for consolidated virtual machines. In: Proceedings of the 2nd ACM Symposium on Cloud Computing, p. 22. ACM (2011)
27. Ha, K., Abe, Y., Chen, Z., Hu, W., Amos, B., Pillai, P., Satyanarayanan, M.: Adaptive vm handoff across cloudlets. Tech. rep., Technical Report CMU-CS-15-113, CMU School of Computer Science (2015)
28. Ha, K., Pillai, P., Lewis, G., Simanta, S., Clinch, S., Davies, N., Satyanarayanan, M.: The impact of mobile multimedia applications on data center consolidation. In: Cloud Engineering (IC2E), 2013 IEEE International Conference on, pp. 166–176. IEEE (2013)
29. Hajjat, M., Maltz, D., Rao, S., Sripanidkulchai, K., et al.: Dealer: application-aware request splitting for interactive cloud applications. In: 8th international conference on Emerging networking experiments and technologies, pp. 157–168. ACM (2012)
30. Islam, M.S., Gibson, M., Muzahid, A.: Fast and qos-aware heterogeneous data center scheduling using locality sensitive hashing. In: 2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom), pp. 74–81. IEEE (2015)
31. Jarschel, M., Schlosser, D., Scheuring, S., Hoßfeld, T.: Gaming in the clouds: Qoe and the users' perspective. Mathematical and Computer Modelling **57**(11), 2883–2894 (2013)
32. Joshi, K., Raj, A., Janakiram, D.: Sherlock: Lightweight detection of performance interference in containerized cloud services. In: 2017 IEEE 19th International Conference on High Performance Computing and Communications; IEEE 15th International Conference on Smart City; IEEE 3rd International Conference on Data Science and Systems (HPCC/SmartCity/DSS), pp. 522–530 (2017). DOI 10.1109/HPCC-SmartCity-DSS.2017.68
33. Kambadur, M., Moseley, T., Hank, R., Kim, M.A.: Measuring interference between live datacenter applications. In: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, p. 51. IEEE Computer Society Press (2012)
34. Kivity, A., Kamay, Y., Laor, D., Lublin, U., Liguori, A.: kvm: the Linux virtual machine monitor. In: Proceedings of the Linux Symposium, vol. 1, pp. 225–230 (2007)
35. Koh, Y., Knauerhase, R., Brett, P., Bowman, M., Wen, Z., Pu, C.: An analysis of performance interference effects in virtual environments. In: Performance Analysis of Systems & Software, 2007. ISPASS 2007. IEEE International Symposium on, pp. 200–209. IEEE (2007)
36. Kohavi, R., Henne, R.M., Sommerfield, D.: Practical guide to controlled experiments on the web: listen to your customers not to the hippo. In: 13th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 959–967. ACM (2007)
37. Kosta, S., Aucinas, A., Hui, P., Mortier, R., Zhang, X.: ThinkAir: Dynamic Resource Allocation and Parallel Execution in the Cloud for Mobile Code Offloading. In: INFOCOM, 2012 Proceedings IEEE, pp. 945–953 (2012). DOI 10.1109/INFCOM.2012.6195845

38. Kuang, W., Brown, L.E., Wang, Z.: Modeling Cross-Architecture Co-Tenancy Performance Interference. In: 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), pp. 231–240. IEEE (2015)
39. Kwon, M., Dou, Z., Heinzelman, W., Soyata, T., Ba, H., Shi, J.: Use of network latency profiling and redundancy for cloud server selection. In: 2014 IEEE 7th International Conference on Cloud Computing, pp. 826–832. IEEE (2014)
40. Lo, D., Cheng, L., Govindaraju, R., Ranganathan, P., Kozyrakis, C.: Improving resource efficiency at scale with heracles. ACM Transactions on Computer Systems (TOCS) **34**(2), 6 (2016)
41. Lowe, D.G.: Distinctive image features from scale-invariant keypoints. International journal of computer vision **60**(2), 91–110 (2004)
42. Mars, J., Tang, L., Hundt, R., Skadron, K., Soffa, M.L.: Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations. In: 44th annual IEEE/ACM International Symposium on Microarchitecture, pp. 248–259. ACM (2011)
43. McCalpin, J.D.: A survey of memory bandwidth and machine balance in current high performance computers. IEEE TCCA Newsletter **19**, 25 (1995)
44. Merkel, D.: Docker: lightweight linux containers for consistent development and deployment. Linux Journal **2014**(239), 2 (2014)
45. Morabito, R., Kjällman, J., Komu, M.: Hypervisors vs. lightweight virtualization: a performance comparison. In: Cloud Engineering (IC2E), 2015 IEEE International Conference on, pp. 386–393. IEEE (2015)
46. Novaković, D., Vasić, N., Novaković, S., Kostić, D., Bianchini, R.: DeepDive: Transparently Identifying and Managing Performance Interference in Virtualized Environments. In: USENIX Conference on Annual Technical Conference, USENIX ATC'13, pp. 219–230. USENIX Association, Berkeley, CA, USA (2013). URL `http://dl.acm.org/citation.cfm?id=2535461.2535489`
47. Satyanarayanan, M., Bahl, P., Caceres, R., Davies, N.: The Case for VM-Based Cloudlets in Mobile Computing. Pervasive Computing, IEEE **8**(4), 14–23 (2009). DOI 10.1109/MPRV.2009.82
48. Shafer, J.: I/o virtualization bottlenecks in cloud computing today. In: Proceedings of the 2nd conference on I/O virtualization, pp. 5–5. USENIX Association (2010)
49. Silva, F.A., Maciel, P., Matos, R.: Smartrank: a smart scheduling tool for mobile cloud computing. The Journal of Supercomputing pp. 1–24 (2015)
50. Travostino, F., Daspit, P., Gommans, L., Jog, C., De Laat, C., Mambretti, J., Monga, I., Van Oudenaarde, B., Raghunath, S., Wang, P.Y.: Seamless live migration of virtual machines over the man/wan. Future Generation Computer Systems **22**(8), 901–907 (2006)
51. Vinoski, S.: Advanced message queuing protocol. IEEE Internet Computing **10**(6), 87–89 (2006). DOI doi.ieeecomputersociety.org/10.1109/MIC.2006.116
52. Wang, Y.A., Huang, C., Li, J., Ross, K.W.: Estimating the performance of hypothetical cloud service deployments: A measurement-based approach. In: INFOCOM, 2011 Proceedings IEEE, pp. 2372–2380. IEEE (2011)
53. Wendell, P., Jiang, J.W., Freedman, M.J., Rexford, J.: Donar: decentralized server selection for cloud services. ACM SIGCOMM Computer Communication Review **40**(4), 231–242 (2010)
54. Wood, T., Ramakrishnan, K., Shenoy, P., Van der Merwe, J.: Cloudnet: dynamic pooling of cloud resources by live wan migration of virtual machines. In: ACM Sigplan Notices, vol. 46, pp. 121–132. ACM (2011)
55. Xu, C., Chen, X., Dick, R.P., Mao, Z.M.: Cache contention and application performance prediction for multi-core systems. In: Performance Analysis of Systems & Software (ISPASS), 2010 IEEE International Symposium on, pp. 76–86. IEEE (2010)
56. Yang, H., Breslow, A., Mars, J., Tang, L.: Bubble-Flux: Precise Online QoS Management for Increased Utilization in Warehouse Scale Computers. In: ACM SIGARCH Computer Architecture News, vol. 41, pp. 607–618. ACM (2013)
57. Zhang, W., Hu, Y., Zhang, Y., Raychaudhuri, D.: Segue: Quality of service aware edge cloud service migration. In: 8th IEEE International Conference on Cloud Computing Technology and Science (CloudCom). IEEE (2016)

58. Zhang, X., Tune, E., Hagmann, R., Jnagal, R., Gokhale, V., Wilkes, J.: Cpi2: Cpu performance isolation for shared compute clusters. In: Proceedings of the 8th ACM European Conference on Computer Systems, EuroSys '13, pp. 379–391. ACM, New York, NY, USA (2013)
59. Zhang, Y., Laurenzano, M.A., Mars, J., Tang, L.: Smite: Precise qos prediction on real-system smt processors to improve utilization in warehouse scale computers. In: 47th Annual IEEE/ACM International Symposium on Microarchitecture, pp. 406–418. IEEE Computer Society (2014)
60. Zhou, B., Dastjerdi, A.V., Calheiros, R.N., Srirama, S.N., Buyya, R.: A context sensitive offloading scheme for mobile cloud computing service. In: Cloud Computing (CLOUD), 2015 IEEE 8th International Conference on, pp. 869–876. IEEE (2015)