INDICES: Exploiting Edge Resources for Performance-aware Cloud-hosted Services

Shashank Shekhar*, Ajay Dev Chhokra*, Anirban Bhattacharjee*, Guillaume Aupy†, Aniruddha Gokhale*
*Vanderbilt University, Nashville, TN 37235, USA and †INRIA-Bordeaux, Bordeaux, France
* Email: {shashank.shekhar,ajay.d.chhokra,anirban.bhattacharjee,a.gokhale}@vanderbilt.edu and †guillaume.aupy@inria.fr

***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 paper makes three contributions to address these limitations: (a) determining when to reprovision, (b) identifying the appropriate MDC and its host from among multiple choices such that the performance considerations of the applications are met, and (c) ensuring that the cloud service provider continues to meet customer service level objectives while keeping its operational and energy costs low. Empirical results validating the claims are presented using a setup comprising a cloud data center and multiple MDCs composed of heterogeneous hardware.**

***Index Terms*—Cloud Computing, Cloud latency, Micro Data Center, Cloudlet, Edge Computing, Fog Computing, Performance Interference, Resource Management.**

I. 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 users mandate lower latencies for the applications. Prior works [1], [2] 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 [3]. 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. All 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 round trip latencies must focus on reducing the network delays and the server processing time.

In recent years, edge computing, cloudlets [4] or Micro Data Centers (MDCs) [5] have emerged as one of the key mechanisms to manage and bound the transit latency $t_{transit}$ by supporting cloud-based services closer to the clients. MDCs[1] can be viewed as "a data center in a box," which act as the middle tier in the emerging "mobile device–MDC–cloud" hierarchy [4]. MDCs possess key attributes of soft states, sufficient compute power and connectivity, and proximity to clients, and conform to standard cloud technologies.

Recent efforts [6], [7] 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 conserve resources on the mobile client endpoints. Nonetheless, less efforts have focused on 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. Efforts that consider performance of MDCs, however, make very simplistic assumptions regarding their performance models.

In this paper, our focus is on performance of MDCs, specifically the key contributing factors in performance degradation of MDCs and data centers in general. A fundamental system property that is often overlooked is performance interference, which is caused by co-located applications in virtualized data centers [8], [9]. Performance interference being an inherent property of any virtualized system, it manifests itself in MDCs also and therefore must be factored in any approach that is performance-aware. Thus, we focus on a "just-in-time and performance-aware" service migration approach for moving cloud-based interactive services hosted in the centralized cloud data center to a MDC.

A number of challenges including the heterogeneity in the hardware, and difficulty in measuring performance interferences and other system and network performance metrics must be overcome. We address these challenges in the context of providing a ubiquitous deployment approach that spans the cloud-edge spectrum and make the following contributions:

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

- We present a technique to estimate the performance of a cloud application on different hardware platforms subjected to performance interference stemming from various co-located applications.
- We formulate server selection as an optimization problem that finds an apt server among micro data centers to migrate an application to, so it can meet its performance needs while minimizing the deployment cost to the service provider.
- We describe the INDICES (**IN**telligent **D**eployment for ub**I**quitous **C**loud and **E**dge **S**ervices) framework that codifies our 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 paper is organized as follows: Section II presents the system model and assumptions; Section III describes the problem formulation we address in this research; Section IV delves into details of our solution including the design and implementation; Section V presents empirical proof that validates our claims; Section VI compares related work with our work; and finally Section VII presents concluding remarks alluding to lessons learned and future work.

II. System Model and Assumptions

This research 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, virtual desktop etc. by improving application response times. To that end they exploit micro data centers. In doing so, however, cost considerations and energy savings for the PaaS provider in operating and managing the resources beyond the traditional data centers are critical issues while ensuring that such an approach provides an additional source of revenue to the PaaS provider. In this paper, however, we do not discuss revenue generation issues.

A. Architectural Model

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), $M = \{m_1, m_2, .., m_n\}$. 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 that CDC provider.[2]

The CDC contains a global manager gm, which is responsible for detecting and mitigating global SLO violations. We assume that for all $m \in M$, there exist links to the CDC with a backhaul bandwidth of b_m. Each MDC m 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

[2]Sharing of MDCs across different CDC providers is a dimension of our future work.

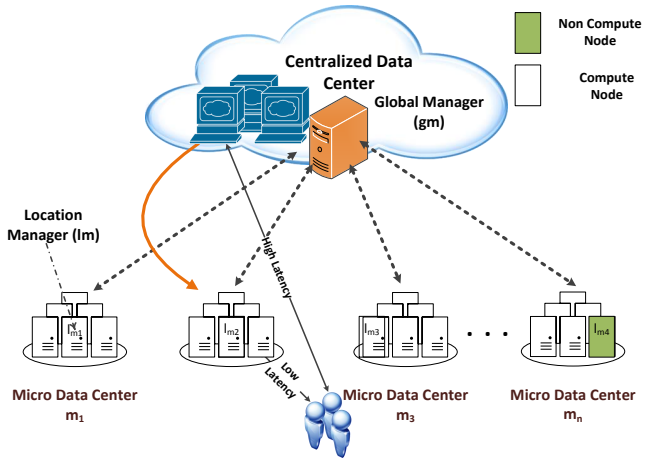

Fig. 1. Architectural Model

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.

B. Application Model

For this work, we consider a set $Apps$ of latency-sensitive applications that can be collaborative or single user and interactive or streaming in nature. Each application $a \in Apps$ is initially deployed in a CDC, with U_a number 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, e.g., in a collaborative educational application [10].

Each application a can be hosted on any active host in CDC, $\eta \in H$, where H is the set of all active hosts that provide virtualization using a hypervisor or virtual machine monitor (VMM). We 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. Each interactive or streaming step of a is assumed to take 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. Finally, for all users $u \in U_a$, let $el_{a,\eta,u}$ represent the estimated round-trip network latency and ϕ_a be the application-defined bounds on acceptable response time for each interactive step of the application. Formally, the SLO for each application a hosted on host η can be characterized by:

$$eet_{a,\eta} + \max_{u \in U_a}(el_{a,\eta,u}) \leq \phi_a \quad (1)$$

Over time, a subset PA from the set of applications $Apps$ are identified by the system as suffering from performance degradation such that each application $p \in PA$ has a subset

of one or more users, $U'_p \subseteq U_p$ experiencing SLO violations. SLO violation can be noticed either by the client-side instrumentation capability included with the "app" that the user installs or via a predictive capability used by the CDC based on user profile and location.

Our objective is thus to minimize the SLO violations, which is achieved by identifying and migrating application p to a MDC host $h \in H_m$ that will provide significantly improved performance. Since any application migration will involve state transfer, we assume that application p has the snapshot of current state which has to be transferred as part of the migration over the backhaul network from CDC to MDC m. Moreover, $ci_{p,h}$ is the initialization cost of the migrated application p on host h_m before the application can start processing requests on the MDC host. However, 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 p. For this paper we do not consider further consolidation of resources where applications migrate back to the CDC. The transfer cost $transfer_{p,h}$ incurred while transferring application p from CDC to host h of a MDC, and associated constraint are defined in the following equations:

$$transfer_{p,h} = \frac{s_p}{b_m} + ci_{p,h} \tag{2}$$

$$transfer_{p,h} \ll eed_p \tag{3}$$

where, b_m is the backhaul bandwidth, s_p is the size of the snapshot of the application's state, and eed_p is the remaining expected execution duration of application p's usage by the client. Equation 3 is a necessary condition for the motivation to use the edge and our solution to be relevant. To ensure that Equation 3 holds, we do not require transferring entire images of the VM and its containers. 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 IV-D. This assumption is realistic because we surmise that a MDC is either owned entirely or leased exclusively by a CDC provider. We also ensure Equation 3 holds by considering δ_p as a tolerance percentage value for the application user before (s)he starts to observe the improved response time:

$$transfer_{p,h}/eed_p \leq \delta_p \tag{4}$$

Finally, another critical issue we must account for is that any migration of a new application from CDC to a MDC should not violate the SLOs of existing applications in that MDC. To capture this aspect, let J_h represent the set of all applications currently running on a MDC host h, $eet_{j,h}$ be the estimated execution time for each application $j \in J_h$, which must be updated when we make a decision to migrate p to the same host, and $el_{j,h,u}$ be their corresponding measured round-trip network latency. These quantities must satisfy:

$$\forall j \in J_h, eet_{j,h} + \max_{u \in U_j}(el_{j,h,u}) \leq \phi_j \tag{5}$$

III. Problem Statement and its Formulation

We now formally present the problem statement, which includes two parts: determining which users are experiencing SLO violations, and making decisions to migrate the impacted application from CDC to apt MDC host.

A. Performance Estimation Problem and 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 [8], [9]. Below we describe their role in the performance estimation problem:

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 with 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 for different client mobile devices. Thus, we consider each workload as a different application setting, which is reflected in the application response time.

2) Heterogeneity: Our CDC and MDCs consist of heterogeneous hardware and hence each application's performance can vary significantly from one hardware platform to another [8]. Therefore, we need an accurate benchmark of performance for each hardware platform.

3) Performance Interference: Although hypervisors/VMMs provide a high degree of security, fault, and environment isolation, the level of isolation is inadequate when it comes to performance isolation for the following reasons:

- *Presence of non-partitionable shared resources:* On-chip resources including cache spaces, cache and DRAM bandwidths, and interconnect networks are difficult to partition [11]. Although, Intel has introduced Cache Allocation Technology [12] to partition the last level cache (LLC), it is still not widely used and cannot be applied to older generation servers. The load imposed on these shared resources by one application is detrimental to all the cache- and memory-sensitive applications [13].
- *Resource overbooking:* Resource overbooking is common in cloud data centers, which precludes strict CPU reservations and can lead to lower level caches (L1 and L2) getting shared. Overbooking beyond the server capacity can lead to significant performance issues for the applications.

B. Cost Estimation and Objective Formulation

The objective of the framework is to assure the SLOs for all the identified applications $p \in PA$ while minimizing the overall deployment cost. Each MDC host h 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, the running servers have operational costs, such as need for power and cooling. Thus, the provider wants to use as few MDC servers as possible and hence the deployment cost depends on the duration for which the MDC server is on.

This cost $\widetilde{T}_h$ for deploying $p \in PA_h$ applications on host h is the extra duration for which the server has to be turned on and can be represented as:

$$\widetilde{T}_h = \begin{cases} 0, & \text{if } \max\limits_{p \in PA_h}(eed_p) < \max\limits_{j \in J_h}(eed_j), \\ \max\limits_{p \in PA_h}(eed_p) - \max\limits_{j \in J_h}(eed_j), & \text{otherwise} \end{cases} \tag{6}$$

We define a constant α_h denoting the cost of powering on the MDC server, and constant β_h denoting the cost for transferring the state to host h. Their values depend on the host h and its corresponding MDC. The cost for deployment on host h is thus defined as:

$$C(h) = \alpha_h * \widetilde{T}_h + \beta_h * \sum_{p \in PA_h} transfer_{p,h} \tag{7}$$

The optimization problem we solve for this research can then be formulated as:

$$\begin{aligned} \underset{h \in H}{\text{minimize}} & \sum C(h) \\ \text{subject to} \quad & eet_{p,h} + \max_{u \in U_p}(el_{p,h,u}) \leq \phi_p, \\ & \forall j \in J_h, eet_{j,h} + \max_{u \in U_j}(el_{j,h,u}), \\ & transfer_{p,h}/eed_p \leq \delta_p \end{aligned} \tag{8}$$

IV. Design of INDICES

We now present the design of the INDICES framework, which solves the optimization problem from Equation 8. The remainder of this section describes the INDICES architecture and the techniques used to solve the optimization problem.

A. INDICES Architecture and Implementation

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 as shown in Figure 1.

Each MDC is composed of a management node and several servers on which the applications residing on virtual machines execute. 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, latency estimator, performance predictor and cost estimator.

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 [14] 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 message queuing protocol. The lm_m runs a server developed in the Go programming language, which persists the data in the InfluxDB database, which is designed specifically for time series data.

B. Estimating Execution Time

The constraints in Equation 8 require an accurate understanding of the predicted execution time duration of an application if it were to execute at a 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 [15] when co-located with other applications on different hardware platforms given the hardware heterogeneity across the CDC and MDCs. Although prior efforts [13], [16] have used retired instructions per cycle (IPC) or last-level cache (LLC) miss rate as the performance indicators, Lo et. al [17] 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 [13], [15], [18], [19] is a property that identifies the degree to which that application will (a) degrade the performance of other running applications on the host – known as *pressure* – and (b) how much its own performance suffers due to interference from other applications – known as *sensitivity*. Several prior efforts have used pairwise application execution to estimate their sensitivity and pressure [13], [18], [19], however, these solutions are not viable for a data center given the significantly large number of hosted applications. Some other efforts [16] pause non-critical applications to measure pressure and sensitivity of live applications, which may not be a realistic solution.

Thus, for a given application p, its performance on a host with hardware configuration w is modeled by Equation 9, where Y is the execution time, X is a vector of system-level metrics that quantify the state of the host, and the function $f_p^1()$ models the relation between the state of the host machine and performance of the application p. Moreover, the information needed by the second constraint of Equation 8 is obtained through Equation 10, which depicts the change in the state of the host with hardware configuration w if application p were to be hosted on it. Equation 10 is 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^{new} into Equation 9 and solving it for each running application.

$$Y = f_p^1(X_w) \tag{9}$$

$$X_w^{new} = f_p^2(X_w^{old}) \tag{10}$$

Another required step is to identify the right system level metrics to use. Previous works [8], [15], [20] have identified several sources of interference including caches, prefetchers,

TABLE I
SERVER ARCHITECTURES

Config	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

memory, network, disk, translation lookaside buffers (TLBs), and integer and floating point processing units. Although modern hardware architectures provide access to many performance counters, which is not the case with older architectures, to provide a broadly applicable and easily reproducible approach, we have selected the following metrics:

- **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. In regard to LLC and memory bandwidth, due to their strong correlation that we observed during experimentation and the ease of instrumenting the latter, we have used only memory bandwidth. Table I lists the hardware counter-based equations for memory bandwidth, which are derived from [21], [22].
- **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 9 and 10 using the following sequence of steps:

1) **Feature Selection:** We adopted the Recursive Feature Elimination (RFE) approach using Gradient Boosted Regression Trees [23] as a way to select the optimal set of features and reduce training time. We performed RFE in a cross-validation loop to find the optimal number of features that minimizes a loss function (mean squared).
2) **Correlation Analysis:** To further reduce the training time by decreasing the dimensions of the feature vector, we used the Pearson Coefficient to eliminate highly dependent metrics with a threshold of ± 0.8.
3) **Regression Analysis:** We have used the off-the-shelf Gradient Tree Boosting curve fitting method due to its ability to handle heterogeneous features and its robustness to outliers.

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

1) Offline Phase: Whenever the data center receives a request for migrating an as yet un-profiled application, 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 above listed three steps or 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 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 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.

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 for estimating 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 9. The pressure on existing applications J_h on the host h is calculated by first applying Equation 10 on the target application and then Equation 9 for existing applications.

C. Network Latency Estimation

The constraints of the optimization problem of Equation 8 require an accurate understanding of the network latencies incurred by the clients, specifically the worst among all the clients of each application. To that end we must determine the clients who suffer SLO violations from Equation 1. 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 need 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.

Although multiple MDCs may be available for migrating the impacted application, to satisfy latency concerns, only those eligible MDCs closer to the end user must be chosen. 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 still sufficient for us as we use the client location to reduce the set of MDCs that we need to query. The client's geolocation and consequently its region is derived from the IP address.

The next step is measuring 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. The client app selects the SLO latency (e.g., usually 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.

D. State Transfer

The final constraint of Equation 8 requires estimating the cost of state transfer. The local managers calculate the state transfer cost using Equation 2 and use it in local decision making. Once the gm selects the h_m for migrating an application p, the application 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 [4], [24]–[26]. 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 h_m and is synthesized with the base image for execution. Equation 2 displays the cost.

Once the application starts running, it informs the gm and all the application clients are redirected to the new application URL. This happens for a custom client 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 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 [28] live migration feature.

E. Solving the Optimization Problem at Runtime

The final piece of the puzzle is solving the optimization problem in Equation 8. The optimization problem described in III-B cannot be solved offline due to the changing dynamics of the system, and moreover, being an NP-Hard problem, we employ a heuristics-based algorithm described in Algorithm 1 that selects aptly suited servers in a MDC while minimizing the overall deployment cost for the entire system.

Algorithm 1 Deployment Server Selection Algorithm

1: **Input** $\leftarrow (Apps)$
2: **for all** $a \in Apps$ **do**
3: $t_{a,CDC} \leftarrow \max(U_a)$ ▷ t is response time
4: **if** $t_{a,CDC} > \phi_a$ **then**
5: $PA.insert(a)$
6: **if** $PA = \emptyset$ **then return** ▷ Do nothing
7: **for all** $p \in PA$ **do**
8: $eed_p \leftarrow GetExpectedExecutionDuration(p)$
9: $clientLoc \leftarrow GetLocation(max(U'_p))$
10: $nearbyMDCs \leftarrow FindNearbyMDCs(clientLoc)$
11: **for all** $m \in nearbyMDCs$ **do**
12: $lm_m \leftarrow LocalManager(m)$
13: $el_{p,lmTransit} \leftarrow GetLatency(lm_m, clientLoc)$
14: $H_m \leftarrow GetServerList(m)$
15: **for all** $h_m \in H_m$ **do**
16: **if** $transfer_{p,h_m} > \delta_p$ **then**
17: **skip** h_m ▷ Constraint Violated
18: $perf_{p,h_m} \leftarrow PredictPerfInterf(h_m, p)$
19: **for all** $j \in J_{h_m}$ **do**
20: $eet_{j,h_m} \leftarrow EstExecTime(perf_{p,h_m}, j)$
21: **if** $el_{j,h_m} + eet_{j,h_m} > \phi_j$ **then**
22: **skip** h_m ▷ Constraint Violated
23: $eet_{p,h_m} \leftarrow EstExecTime(perf_{h_m}, p)$
24: **if** $el_{p,lmTransit} + eet_{p,h_m} > \phi_p$ **then**
25: **skip** h_m ▷ Constraint Violated
26: $transfer_{p,h_m} \leftarrow EstTransDur(h_m, p)$
27: $C_{p,H_m}.insert(EstCost(transfer_{p,h_m}, eed_p))$
28: $C_{p,m}.insert(min(C_{p,H_m}))$
29: $minC_{p,h} \leftarrow min(C_{p,m})$

The algorithm consists of two phases. First, we identify the applications suffering SLO violations (Line 5). In the

second phase we select the suitable server. For the identified applications in PA, we find the location and address of the client that suffers the worst latency (Line 9). That location is used to perform a lookup for nearby MDCs (Line 10). We then identify the server within the identified MDCs that provide the best performance. This step is carried out in parallel across all the identified MDCs (Loop starting at Line 11). The client measures the latency to the local manager lm_m of each nearby MDC and if it is within the acceptable application response time threshold ϕ_p, then we select that MDC and fetch the corresponding list of servers (Line 14).

For each such server, we predict the performance interference and estimate the execution time of the application p were it to execute on that host (Line 18), and update the estimated execution time of existing applications J on that host (Loop starting at Line 19). We then calculate the cost according to Equation 7 if the constraints defined in Equation 8 can be met (Line 27). The minimum cost server is identified for each MDC (Line 28). Finally the minimum cost server is selected across all identified MDCs (Line 29) and the application is migrated and clients are redirected to the migrated application. Due to the distributed nature of our framework, the algorithm can be solved in $\mathcal{O}(H_m * J_h)$ for each application p.

V. Experimental Validation

We now present results of evaluating INDICES in the context of a latency senstive application use case.

A. Experimental Setup

Table I 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.

We use PARSEC and Splash-2 benchmarks [29] to generate the training data. As described in section IV-B, to preclude profiling every new application on all the hardware, we need some training data. PARSEC targets Chip-Multiprocessors composing virtualized data centers, and provides a rich set of applications with different instruction mix, cache and memory utilization, needed for stressing different system subcomponents. We selected 20 tests from the benchmarks for data generation and validation. Due to lack of access to servers in different geographical regions, we used the network emulation tool, *netem*, and hierarchy token bucket based traffic control, *tc-htb*, for emulating the desired network latencies and bandwidth among the client, CDC and different MDCs.

B. Application Use Case

Our use case is an image processing application that performs feature detection, e.g., facial recognition in computer vision. We use the well-known Scale Invariant Feature Transform (SIFT) [30] to find the scale and rotation independent features. The client-side interface of the application continuously streams frames from a video or a web camera at a fixed rate of a frame per 200 milliseconds. The video 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 within this duration, 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 defined in Table I, the base execution times, eet_a, were measured to be 86, 91, 146, 157 ms, respectively. Table II displays the emulated ping latency el_a 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, was set as 1 hour and the SLO was set to 95%.

TABLE II
CDC and MDC set up for use case (Section V-B)

Conf	Distance	Ping Latency ($\pm$20%) ms	TCP latency(ms)	HTTP latency(ms)	Servers
1	1 hop	$<$1	2	6	1C + 1D
2	2 hops	5	14	28	1A + 2D
3	Multi hops	20	54	96	1B + 2D
4	Multi hops	30	76	142	1A + 3D
5	Central	50	127	220	1D

C. Evaluating the Performance Estimation Model

We first benchmarked 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 IV-B1, was set to 10% error. However, as illustrated in Figure 2, none of the existing applications met the criteria. Thus, we decided not to use any of the existing estimators for the use case application and benchmarked the application on all hardware configurations to develop its estimators. Figure 2 confirms that the estimation errors were high for all the hardware types requiring us to develop its estimators. We also found that the mean estimation error for our use case application to be less than 4% on all the platforms with low standard deviations as depicted in Figure 3. We can also account for this estimation error in our response time constraint (Equation 1) for stricter SLO adherence.

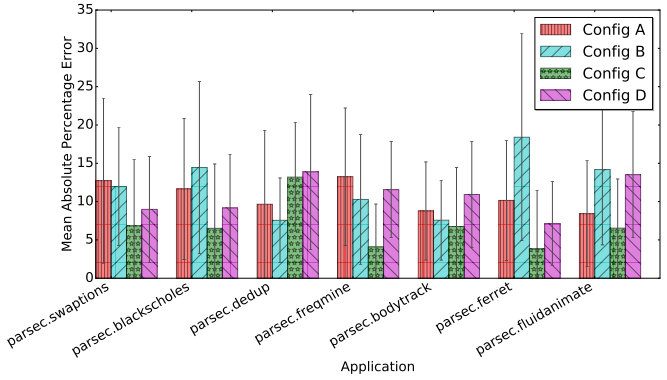

Fig. 2. Estimation of SIFT Profile Similarity with Parsec Benchmark

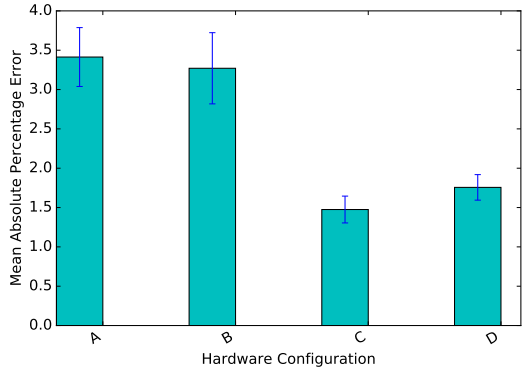

Fig. 3. SIFT Application Performance Estimation Error

D. Evaluating the Server Selection Algorithm

We compare our server selection algorithm results against two approaches: server selection algorithms based on minimum number of hops and least loaded server (among reachable MDCs). From Table II we observe that the minimum hop is 1. There are 2 servers in the minimum hop MDC 1 with hardware configuration types C and D. We create interference load on both the servers but ensured 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 considered the server with least existing allocated resources, i.e. containing only a single VM. We did not consider a server with no existing load as it results in acquiring a new server and thus causes additional cost to the service provider. We found the server of hardware type D with MDC configuration 4 to be least loaded.

Applying SLO from Equation 1, INDICES found 2 servers of type A and D from MDC 2 and one server of type B from MDC 3 to be suitable for which we plot their response times for eed_a of one hour. Figure 4 displays the comparison of each of the suitable servers found by INDICES against the least loaded server. We observe that in this scenario, the least loaded server had 100% SLO violation because of network latency. However, the servers found by INDICES met their deadline 100%, 99.38% and 98.94%, respectively, which was well over the target SLO of 95%. Also, the minimum hop servers met the deadline only 66.64% and 60.64% of times due to performance interference shown in Figure 5.

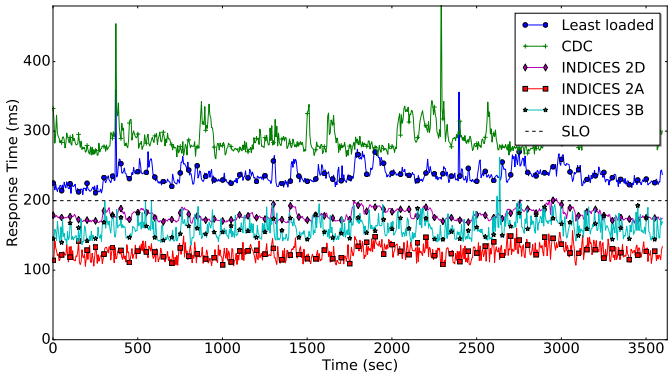

Fig. 4. INDICES vs Least Loaded

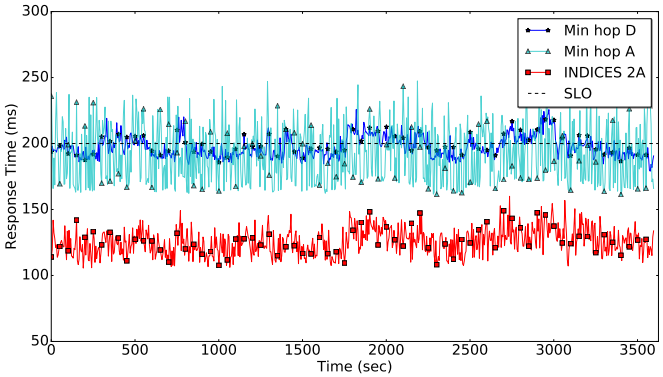

Fig. 5. INDICES vs Minimum Hop

Applying Algorithm 1 further, INDICES found the server of type B from MDC 3 to be most suitable since our objective is to select the minimum cost server to the service provider if it can meet the SLO. Thus, it preferred a server which already had an application that was going to run longer and had better bandwidth from the CDC server for migration. Figure 6 compares 3 migration scenarios (a) an overlay with the software stack already present on the target server and the bandwidth is 10 Mbps, (b) same as previous but with bandwidth 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 sec, however, the migration and initialization takes 32s, 56s and 190s respectively for a, b and c scenarios. Thus, the overlay based image transfer should be the preferred methodology wherever applicable.

VI. Related Work

In this section we compare and contrast our work with related work along three dimensions: network latency-based

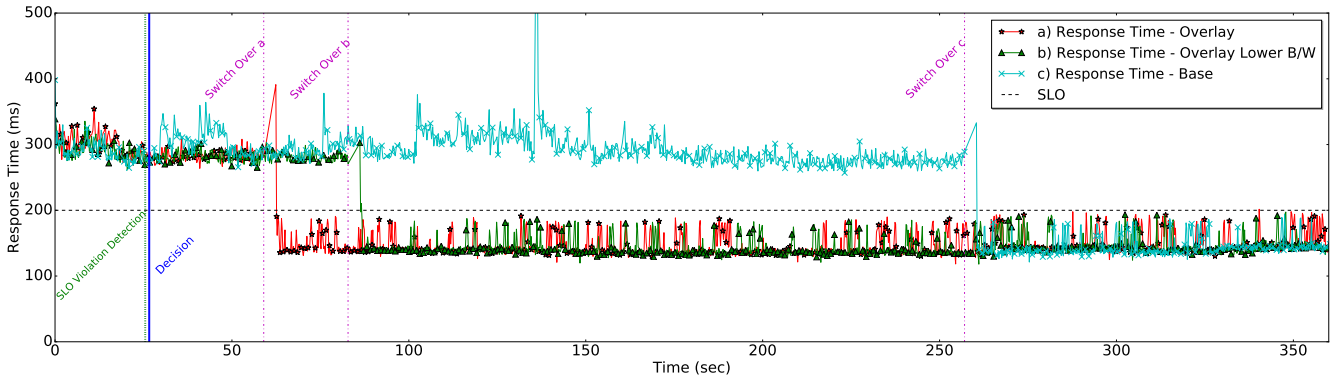

Fig. 6. Application Switch-Over Performed by INDICES under 3 Different Scenarios

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.

A. Network Latency-based Server Selection

DONAR [31] 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. Dealer [32] 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. [33] 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.

B. Performance Interference-aware Server Selection

Paragon [8] 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. SMiTe [18] designed rulers for estimating sensitivity and degree of contention between applications when they are collocated. Bubble-Flux [16] 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.

Our prior work [9] 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 [17] 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.

C. Performance-aware Edge Computing

Zhou et al. [7] 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 [6] 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. [34] 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 [35] 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.

VII. Conclusions

This paper presents an approach 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 are improved. In doing so, our algorithm 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 evaluating our framework called INDICES support our claims.

This work has provided deep insights that require further research. Some of these include the need for readily available

benchmarks, better approaches to collecting measurements, workload consolidation across MDCs and CDCs, revenue generation and energy saving issues, and MDCs that are shared across different CDC providers. Going forward, we will also expand on our assumptions and limitations such as trust, security, workload variations and user mobility.

All scripts, source code, and experimental results for INDICES are available for download from https://github.com/shekharshank/indices.

ACKNOWLEDGMENTS

This work is supported in part by the AFOSR DDDAS FA9550-13-1-0227 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] M. Jarschel, D. Schlosser, S. Scheuring, and T. Hoßfeld, "Gaming in the clouds: Qoe and the users perspective," *Mathematical and Computer Modelling*, vol. 57, no. 11, pp. 2883–2894, 2013.

[2] R. Kohavi, R. M. Henne, and D. Sommerfield, "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*. ACM, 2007, pp. 959–967.

[3] Y. A. Wang, C. Huang, J. Li, and K. W. Ross, "Estimating the performance of hypothetical cloud service deployments: A measurement-based approach," in *INFOCOM, 2011 Proceedings IEEE*. IEEE, 2011, pp. 2372–2380.

[4] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The Case for VM-Based Cloudlets in Mobile Computing," *Pervasive Computing, IEEE*, vol. 8, no. 4, pp. 14–23, 2009.

[5] V. Bahl, "Cloud 2020: Emergence of micro data centers (cloudlets) for latency sensitive computing (keynote)," *Middleware 2015*, 2015.

[6] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "ThinkAir: Dynamic Resource Allocation and Parallel Execution in the Cloud for Mobile Code Offloading," in *INFOCOM, 2012 Proceedings IEEE*, March 2012, pp. 945–953.

[7] B. Zhou, A. V. Dastjerdi, R. N. Calheiros, S. N. Srirama, and R. Buyya, "A context sensitive offloading scheme for mobile cloud computing service," in *Cloud Computing (CLOUD), 2015 IEEE 8th International Conference on*. IEEE, 2015, pp. 869–876.

[8] C. Delimitrou and C. Kozyrakis, "Paragon: Qos-aware scheduling for heterogeneous datacenters," in *ACM SIGPLAN Notices*, vol. 48, no. 4. ACM, 2013, pp. 77–88.

[9] F. Caglar, S. Shekhar, A. Gokhale, and X. Koutsoukos, "An Intelligent, Performance Interference-aware Resource Management Scheme for IoT Cloud Backends," in *1st IEEE International Conference on Internet-of-Things: Design and Implementation*. Berlin, Germany: IEEE, Apr. 2016, pp. 95–105.

[10] F. Caglar, S. Shekhar, A. Gokhale, S. Basu, T. Rafi, J. Kinnebrew, and G. Biswas, "Cloud-hosted Simulation-as-a-Service for High School STEM Education," *Elsevier Simulation Modelling Practice and Theory: Special Issue on Cloud Simulation*, vol. 58, no. 2, pp. 255–273, Nov. 2015. [Online]. Available: http://dx.doi.org/10.1016/j.simpat.2015.06.006

[11] S. Govindan, J. Liu, A. Kansal, and A. Sivasubramaniam, "Cuanta: quantifying effects of shared on-chip resource interference for consolidated virtual machines," in *Proceedings of the 2nd ACM Symposium on Cloud Computing*. ACM, 2011, p. 22.

[12] "Cache allocation technology improves real-time performance," http://www.intel.com/content/www/us/en/communications/cache-allocation-technology-white-paper.html.

[13] J. Mars, L. Tang, R. Hundt, K. Skadron, and M. L. Soffa, "Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations," in *44th annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2011, pp. 248–259.

[14] F. Forster, "Collectd - The System Statistics Collection Daemon." http://collectd.org.

[15] M. S. Islam, M. Gibson, and A. Muzahid, "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)*. IEEE, 2015, pp. 74–81.

[16] H. Yang, A. Breslow, J. Mars, and L. Tang, "Bubble-flux: Precise online qos management for increased utilization in warehouse scale computers," in *ACM SIGARCH Computer Architecture News*, vol. 41, no. 3. ACM, 2013, pp. 607–618.

[17] D. Lo, L. Cheng, R. Govindaraju, P. Ranganathan, and C. Kozyrakis, "Improving resource efficiency at scale with heracles," *ACM Transactions on Computer Systems (TOCS)*, vol. 34, no. 2, p. 6, 2016.

[18] Y. Zhang, M. A. Laurenzano, J. Mars, and L. Tang, "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*. IEEE Computer Society, 2014, pp. 406–418.

[19] W. Kuang, L. E. Brown, and Z. Wang, "Modeling cross-architecture co-tenancy performance interference," in *Cluster, Cloud and Grid Computing (CCGrid), 2015 15th IEEE/ACM International Symposium on*. IEEE, 2015, pp. 231–240.

[20] C. Delimitrou and C. Kozyrakis, "ibench: Quantifying interference for datacenter applications," in *Workload Characterization (IISWC), 2013 IEEE International Symposium on*. IEEE, 2013, pp. 23–33.

[21] Detecting memory bandwidth saturation in threaded applications. [Online]. Available: https://software.intel.com/en-us/articles/detecting-memory-bandwidth-saturation-in-threaded-applications

[22] P. J. Drongowski and B. D. Center, "Basic performance measurements for amd athlon 64, amd opteron and amd phenom processors," *AMD whitepaper*, vol. 25, 2008.

[23] J. Elith, J. R. Leathwick, and T. Hastie, "A working guide to boosted regression trees," *Journal of Animal Ecology*, vol. 77, no. 4, pp. 802–813, 2008.

[24] F. Travostino, P. Daspit, L. Gommans, C. Jog, C. De Laat, J. Mambretti, I. Monga, B. Van Oudenaarde, S. Raghunath, and P. Y. Wang, "Seamless live migration of virtual machines over the man/wan," *Future Generation Computer Systems*, vol. 22, no. 8, pp. 901–907, 2006.

[25] R. Bradford, E. Kotsovinos, A. Feldmann, and H. Schiöberg, "Live wide-area migration of virtual machines including local persistent state," in *3rd international conference on Virtual execution environments*. ACM, 2007, pp. 169–179.

[26] T. Wood, K. Ramakrishnan, P. Shenoy, and J. Van der Merwe, "Cloudnet: dynamic pooling of cloud resources by live wan migration of virtual machines," in *ACM Sigplan Notices*, vol. 46, no. 7. ACM, 2011, pp. 121–132.

[27] K. Ha, Y. Abe, Z. Chen, W. Hu, B. Amos, P. Pillai, and M. Satyanarayanan, "Adaptive vm handoff across cloudlets," Technical Report CMU-CS-15-113, CMU School of Computer Science, Tech. Rep., 2015.

[28] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," *Linux Journal*, vol. 2014, no. 239, p. 2, 2014.

[29] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The parsec benchmark suite: characterization and architectural implications," in *17th international conference on Parallel architectures and compilation techniques*. ACM, 2008, pp. 72–81.

[30] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.

[31] P. Wendell, J. W. Jiang, M. J. Freedman, and J. Rexford, "Donar: decentralized server selection for cloud services," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 4, pp. 231–242, 2010.

[32] M. Hajjat, D. Maltz, S. Rao, K. Sripanidkulchai *et al.*, "Dealer: application-aware request splitting for interactive cloud applications," in *8th international conference on Emerging networking experiments and technologies*. ACM, 2012, pp. 157–168.

[33] M. Kwon, Z. Dou, W. Heinzelman, T. Soyata, H. Ba, and J. Shi, "Use of network latency profiling and redundancy for cloud server selection," in *2014 IEEE 7th International Conference on Cloud Computing*. IEEE, 2014, pp. 826–832.

[34] D. Fesehaye, Y. Gao, K. Nahrstedt, and G. Wang, "Impact of cloudlets on interactive mobile cloud applications," in *Enterprise Distributed Object Computing Conference (EDOC), 2012 IEEE 16th International*. IEEE, 2012, pp. 123–132.

[35] W. Zhang, Y. Hu, Y. Zhang, and D. Raychaudhuri, "Segue: Quality of service aware edge cloud service migration," in *8th IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE, 2016.