HARV: Harnessing Hybrid Virtualization to Improve Instance (Re)Usage in Public Cloud

Silvery Fu, Yifei Zhu, Ryan Shea, and Jiangchuan Liu
School of Computing Science, Simon Fraser University
Email: {dif, yza323, rws1, jcliu}@cs.sfu.ca

Abstract—**In the public cloud market, there has been a constant battle over the billing options of the cloud instances between their providers and their users. The users generally have to pay for the entire billing cycle even on fractional usage. Ideally, the residual life-cycles should be resalable by the users, which demands efficient resource consolidation and multiplexing; otherwise, the revenue and use cases are confined by the transient nature of the instances. This paper presents HARV, a novel cloud service that facilitates the management and trade of cloud instances through a third-party platform to run buyers' tasks. The platform relies on *hybrid virtualization*, an infrastructure layout integrating both the hypervisor-based virtualization and lightweight containerization. It further incorporates a truthful online auction mechanism for instance trading and resource allocation. Our design achieves efficient resource consolidation with no need for provider-level support, and we have deployed a prototype of HARV on the Amazon EC2 public cloud. Our evaluations on both micro-benchmarks and real-life workloads reveal that applications experience negligible performance overhead when hosted on HARV. Trace-driven simulations further show that HARV can achieve substantial cost savings.**

I. Introduction

IaaS (Infrastructure as a Service) has been a major form of public cloud service deployment, and it is estimated the IaaS market will grow from \$15.1B in 2014 to \$126.2B by 2026 [1]. State-of-the-art IaaS cloud providers generally offer resources to users as *virtual machine* (VM) instances, and a user has to pay for the full billing cycle of an instance even if only a fraction of the cycle is to be used. Existing studies have shown that this partial usage issue exists extensively among cloud tasks [2]. As a matter of fact, 79.8% of cloud users use less than 20% of billing cycle according to the previous analysis [2]. There have been pioneer efforts toward fine-grained resource provisioning and pricing to offer instances that better match the user demands [2][3][4]. Unfortunately, as we will show later, there is a trade-off in terms of cost-effectiveness between a cloud provider and the users since the former generally resists to refining the instance granularity.

An attractive alternative is to allow users to re-sell their unused instances [5]. Having a cloud market allowing this not only improves the utilization of cloud resources but is beneficial for building a healthier cloud ecosystem [6]. This is however easy said than done. To generate re-usable resources, it is necessary to aggregate and consolidate the partially used instances. Early solutions on resource consolidation are mostly done from the provider-side, e.g., how to allocate virtual machines given the limited number of available physical machines [7][8][9]. To achieve similar goals from the user-side in the public cloud, global knowledge of the physical machine cluster will be needed, together with such operations as VM live migration. It is hardly possible or feasible for a public cloud provider to expose those low-level interfaces to its users given concerns from security and network/system management. Such third-party solutions as *Cloud Brokerage* [10] suggest that a wholesaler may purchase a large volume of instances from the cloud provider and re-sell them to users at discounted prices. While they do not require infrastructure changes to the public cloud provider, the broker still operates at the VM level; thereby the *usage waste problem* of the cloud instances remains exist.

Moreover, maximizing the (re)usage efficiency demands effective resource-multiplexing, i.e., allowing workloads from more than one user to run together on an instance. Without a proper implementation, this will lead to nested virtualization that can introduce considerable performance overhead [11]. Maximizing the (re)usage efficiency also calls for novel pricing mechanism beyond those offered by the public cloud provider. Facing the ever changing availability of partially used instances and the arrival patterns of their potential users, a dynamic online solution is naturally expected.

In this paper, we show strong evidence that the partially used instances are valuable resources, which, if properly recycled, can remarkably improve the cost-effectiveness of public cloud users. We also demonstrate that such an *instance recycling* service is doable with limited overhead to both cloud users and providers. In particular, we design and implement HARV, a third-party platform that **HAR**nesses hybrid **V**irtualization to both recycle cloud instances and manage their users' tasks. The hybrid virtualization seamlessly combines existing hybervisor-based virtualization and containerization, and does not require any change to the infrastructure of the existing public cloud providers. We present a two-level scheduling policy in HARV to simplify cluster resource management and ensure its applicability with a public cloud. It also incorporates a truthful online auction mechanism to determine the allocation of requests and the corresponding recycling price. We have implemented HARV and deployed it with the Amazon EC2 public cloud. Extensive experiments with real-world benchmarks and large-scale simulations verify that HARV is highly scalable and cost-effective. It achieves cost savings up to 24% on a typical 1-hour billing cycle, and 19% on the 15-minute billing cycle.

The remainder of this paper is organized as follows. We first

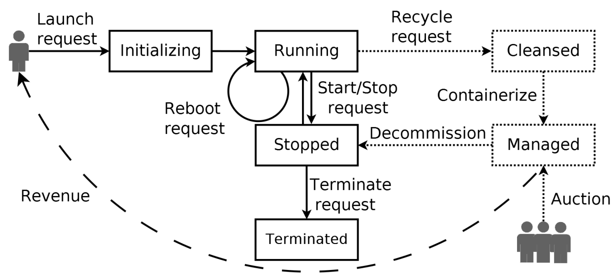


Fig. 1: Augmented instance life cycle

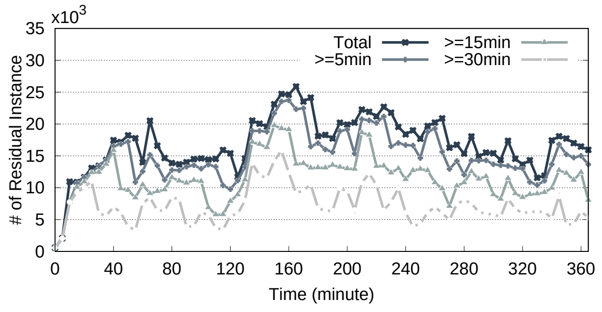


Fig. 2: The number of residual instances

explore the public cloud cost-effectiveness issues and present a system overview in Section II. In Section III, we present details about our system design, including its scheduling and pricing policies. Extensive experiments and evaluations can be found in Section IV. We review related literature in Section VI, and finally conclude our work in Section VII.

II. Background and Motivation

We start from investigating the (in)efficiency of state-of-the-art billing options offered by IaaS cloud and how it affects users' cost-effectiveness. We argue that a recycling mechanism is necessary for utilizing the residual time of cloud instances, and suggest that hybrid virtualization is the key toward real-world implementation and deployment.

A. Billing Inefficiency: Cause and Consequence

In Fig. 1, we depict the state transitions in a typical public cloud instance's life-cycle (the ones in solid lines). In general, the provisioned instance is considered in the same billing cycle as long as it stays in the *Running phase*; however, if the user *stops* a running instance, a new billing cycle will begin when it is restarted. It is because the cloud provider needs to release the computing resources held by the instance (CPU cores, memory, IP, etc.) when handling the stop request. As such, when the user "restart" the instance, a new group of resources has to be re-provisioned for it. Consequently, the user will be charged for the newly provisioned resources, even if (in terms of time) it still falls into the same billing cycle. Let T_{actual} be the actual time a user utilizes an instance, and T_{cycle} be the duration of the billing cycle, the *residual instance time* T_{RI} can be calculated as:

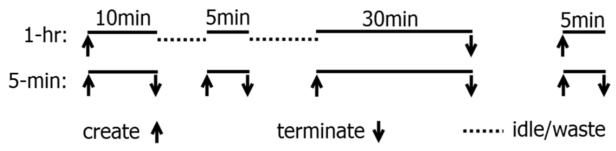


Fig. 3: Billing cycle: 1-hour vs. 5-minute

$$T_{RI} = T_{cycle} - (T_{actual} \bmod T_{cycle}) \tag{1}$$

In Fig. 2, we depict the number of potential residual instances ($T_{cycle} = 1hr$) per time slot during a ≈ 6.5 hours record duration in a real-world cluster. Each of the four lines denotes an assumed T_{RI} range of the residual instances. The data are extracted from one of the Google's publicly accessible traces [12]. Those Google-cluster traces have also been widely used in other recent cloud resource provisioning studies as well [7][10][13]. As shown, despite the fluctuation, there is a constant supply of residual instances: about 10,000 to 20,000 with $T_{RI} \geq 15min$, and the ones with $T_{RI} \geq 30min$ account for nearly half of the total.

Define the *waste ratio*:

$$W_{ratio} = T_{RI}/T_{actual} \tag{2}$$

which is bounded by T_{cycle}/T_{actual}; given the actual usage time is usually unpredictable, the smaller the T_{cycle}, the less likely a cloud user will overpay the billing. Had the cloud providers adopted an ideal "per-second billing", the billing would have been efficient. Unfortunately, *per-hour* cycle is still the dominant billing model in the current IaaS market (e.g. Amazon EC2)[1]. Although there exist cloud providers who offer per-minute billing after an initial time interval[2] such as Microsoft Azure Cloud and Google Compute Engine, their resource offerings are different from EC2's. For example, EC2's instance grants users nearly full control over the software stack including even the kernel [14]. In addition, we conjecture it is a business decision for the per-minute billing cloud platforms to offer more competitive pricing schemes than their market opponents [15] even if those schemes could yield a lower profit margin as we will explain in what follows.

The above analysis raises the question: why IaaS providers favor long billing cycles? In addition to other potential reasons, we conjecture that a longer billing cycle will help compensate and reduce cloud providers' operational costs, especially the costs for instance provisioning (e.g., instance creation, decommission, VM image transfer, boot-time operations, scheduling costs, etc.). To be specific, first, we can infer from Fig. 2 that the duration of user jobs vary substantially with the majority being short-term ones. We then extract the history a user's job requests during a 75-minute interval from the trace as depicted in Fig. 3. Supposing this user will create an instance and runs

[1]As shown later in this paper, even when the billing cycle is much shortened (e.g. 15 minutes), our solution can still provide substantial cost savings.

[2]The initial time interval is usually 10 to 15 minutes which also leads to potential billing inefficiency problems.

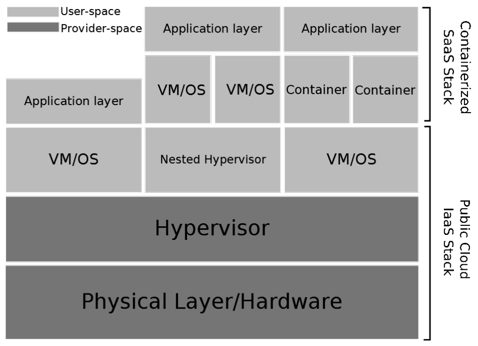


Fig. 4: Hybrid virtualized layering in public cloud

several jobs spanning across our examined time interval, when the billing cycle is an hour, the user may subsequently create two instances with the first one covering three jobs in the first hour (with a waste ratio $1/3$). When the billing cycle is shorter (e.g., 5 minutes), the user is allowed to timely terminate the instance to avoid unnecessary billing cycle charges and create a new instance upon the arrival of the next job (with a zero waste ratio). On the provider side, however, the shortened billing cycle leads to 2x more instance creations and thereby surged provisioning costs.

Despite it being an ideal case for users who have precise cost management, we can expect most of the users would follow such a pattern to avoid unnecessary billing if shorter billing cycle were available. Hence, a longer billing cycle could help reduce potential provisioning costs for cloud providers. It transfers the complexity of consolidating workloads in the time dimension to cloud users, which could lead to the billing inefficiency problem if not addressed.

B. Recycling Instances with Third Party

Given the resistance from the cloud service provider on shortening billing cycles, a better alternative is to consolidate user-supplied residual instances, trade their computing resources, and generate revenue. For brevity, we refer to the cloud users who "recycle" their instances as sellers; those who purchase resources as buyers. As illustrated in Fig. 1, the instance life-cycles can be augmented with additional states and transitions represented in dotted lines. Before a seller decides to stop an instance, it can launch a recycle request, with the necessary information to take over the instance and immediately clean its states. Once the instance reaches the *cleansed* state, it can be added back to the cluster management. Buyers can purchase resources from the cluster at a market-driven price, deploy their applications, and the resulting revenue goes to the sellers. Each managed instance is associated with a decommission deadline to ensure the seller not to be charged for another full billing cycle.

While there are instances available for recycling as shown earlier, the recycling is nontrivial to accomplish from both the system's perspective and the pricing perspective. Fig. 4 describes a hybrid virtualized layering structure in the public cloud. At the very bottom sits the physical layer with bare-metal machines, on top of which hypervisor is placed to abstract and manage the underlying hardware. These two layers are marked as the provider-space, as only the cloud providers have access to resource management on these layers for security reasons. As such, public cloud users are not allowed to access these provider-space utilities, making resource consolidation difficult at the user-space. Even worse, the heterogeneity and highly transient nature of recycled instances may significantly limit the compatible workload types. In short, we are facing the following challenges:

- Managing a large amount of residual instances;
- Utilizing transient cloud resources efficiently;
- Identifying target workloads and providing platform-level supports accordingly;
- Determining the resource price and scheduling policies.

To address the first two challenges, there is a need for an additional virtualization layer on top of the existing one. It will allow tenant isolation on the same recycled instance as to achieve resource multiplexing. It is also a resource management layer where residual instances can be consolidated even with no support from the provider. We emphasize here that a *third-party* solution that does not rely on the provider for recycling is necessary: 1. The instance's billing cycle is fully paid regardless of whether the owner chooses to recycle it or not. 2. Providers could have higher operational costs when residual instances are recycled, since those instances will consume more resources as compared to when they are idle. As such, without explicit incentives, providers themselves are less likely to offer the recycling service on their own. More discussion on the potential incentives is offered in Sec. V.

C. Why Hybrid-virtualization?

There are two potential candidates for building the additional layer, namely *nested virtualization*[3] and *hybrid-virtualization*. As depicted in Fig. 4, in the original user-space (left side of the figure), applications are run directly in the provider-managed VM. With nested virtualization, the applications are placed in the VMs managed by a nested hypervisor, which is run on top of the original VM. By doing so, each application in the same VM can now have their own virtualized resource pool and isolated runtime environment. This is seemingly a natural choice to facilitate resource consolidation in the user-space [16][17]. However, placing a hypervisor on top of another often results in excessive overhead and application performance penalties [11]. Further optimization would require tuning the underlying hypervisor

[3]Nested Virtualization in Xen: http://wiki.xenproject.org/wiki/Nested_Virtualization_in_Xen

TABLE I: Performance of hybrid-virtualization

Resource Type (Benchmark)	Bare-VM	Hybrid-VM
CPU (7z Compression)	92.18Mbytes/s	92.26Mbytes/s
Memory (Sysbench, Read)	10.84Gbytes/s	10.85Gbytes/s
Memory (Sysbench, Write)	10.49Gbytes/s	10.08Gbytes/s
Disk (Bonnie++, Rewrite)	119.95Mbytes/s	118.22Mbytes/s
Network (Iperf, TCP Send)	126.60Mbytes/s	126.59Mbytes/s
Network (Iperf, TCP Recv)	126.53Mbytes/s	126.50Mbytes/s

resides in the provider space, which unfortunately is not allowed in the public cloud in general.

On the contrary, the alternative technique leverages both the traditional virtualization and the emerging lightweight containerization, which we refer to as hybrid-virtualization. In hybrid-virtualization, application containers are placed insides virtual machines. The container, in its simplest form, is a collection of OS kernel utilities (e.g. `cgroups`) configured to manage the resources that an application uses. With containers, resources are monitored and managed through efficient function hooking devised in only the non-performance critical execution paths, thereby incurring much lower overhead.

To validate this, we provisioned four m4.2xlarge general purpose instances from Amazon EC2 cloud, powered by 8x vCPU on Intel Xeon Haswell processor, 32 GB memory, high-throughput SSD storage, and enhanced networking. In the non-containerized test (the baseline), benchmarks were run directly in the host VM. For container virtualization, we installed the latest version of *docker*[4], the mostly widely used container implementation. In Table. I, we present the experimental results. As we can see, for CPU, containers are able to attain the compression speed within $\pm 0.1\%$ against the native VM. A closer look at the MIPS number confirmed that container does not consume more CPU cycles in compression. Similar observations can be made on the disk, memory, and network tests. These results indicate that hybrid-virtualization is able to complement today's public cloud infrastructure with another lightweight resource management layer, and thereby has the potential of supporting the instance recycling framework with limited performance penalties.

III. HARV: System Design, Optimization, and Implementation

We designed and implemented HARV, a third-party platform that **HAR**nesses hybrid **V**irtualization to realize the instance recycling mechanism. In this section, we first illustrate the design considerations of HARV. We show how HARV uses a two-level scheduling policy to simplify cluster resource management while improving its applicability. We show how it handles workloads with different persistence and duration requirements. Further, we design a truthful online auction mechanism to complement our system.

A. Cluster Architecture

In Fig. 5, we describe the architectural design of the HARV. Our cluster consists of recycled/residual instances from contributors, a *state manager* module in charge of cluster state updates, and a *Tier-1 scheduler* handles container allocation. *Tier-2 scheduler* and load balancer are two complementary modules incorporating the two-level scheduling policies, and can be customized by the buyers themselves. The state manager holds a consistent state information of the cluster, including details on each active residual instance, the available resources, and their decommission deadline. It is also in charge of detecting failures of residual instances and containers through keep-alive messaging. Upon the arrival, failure, or decommission of each residual instance, the state manager updates the state table and notifies Tier-1 scheduler.

[4]Docker Container: https://www.docker.com/

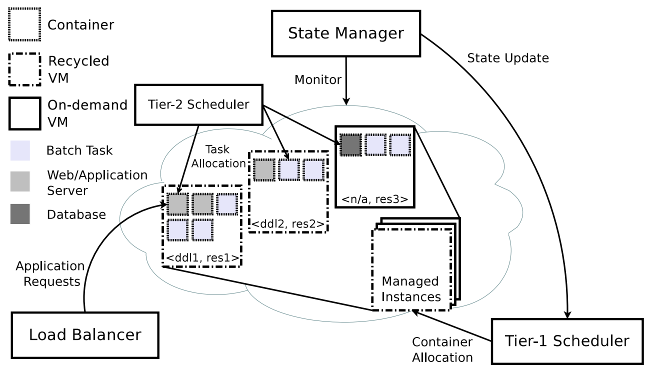


Fig. 5: Cluster architecture with two-level schedulers

Containers are allocated based on our auction mechanism. As shown in Fig. 5, five containers are placed in the leftmost instance, including two for web servers and three for batch tasks. Those containers may have different arrival time, duration, and ownerships. Meanwhile, each component of an application is encapsulated in different containers and scheduled across the cluster. By allowing such, we can achieve not only resource consolidation but also better resource multiplexing and statistical multiplexing in the user-space.

B. Two-level Scheduling

Algorithm 1 Request dispatch algorithm

```
1: while Receiving request $q_i$ do
2:   $u_i = q_i.getUserID()$
3:   if $q_i.isContainerRequest()$ then
4:     $S = MS.getCurrentInstances()$
5:     $\tau_i = R_i.getResourceVector()$
6:     if $s_i = \emptyset$ then
7:       $T1.declineRequest(q_i)$
8:     else
9:       $MS.update(s_i, \tau_i)$
10:    end if
11:  else
12:    $T2_i = MS.getT2Scheduler(u_i)$
13:    $T2_i.scheduleWorkload(q_i)$
14:  end if
15: end while
```

Another advantage brought by the hybrid-virtualization is allowing us to separate cluster-level scheduling and application-specific scheduling. Specifically, HARV does not schedule user-provided jobs directly. Instead, it only decides the container allocation on residual instances, improving resource utilization, and optimizing recycling efficiency. We employ the auction algorithm, described in Sec. III-C, as the cluster-level scheduler (Tier-1) for this purpose.

Meanwhile, the application-specific scheduler (Tier-2) enables buyers to deploy customized scheduling policies. This is because, intuitively, users are the ones who ultimately decide how to effectively use their provisioned containers,

Symbol	Description
MS	Cluster state management service
$T1, T2_i$	Tier-1 scheduler and Tier-2 scheduler supplied by user i
s_i	Instance i
τ_i	Resource vector $< ddl_i, \vec{res}_i >$
b_i	Bid with request i (when use auction-based scheduling)
u_i	The utility when b_i is satisfied
q_i	Request i: $< u_i, \tau_i, (b_i) >$
R_j^r	Capacity of resource type r in an instance j
S	Total number of instances
R	Number of resource types
T	The expected maximum running time of the system
L_r, U_r	Lower and upper bound of per unit resource valuation
t_{min}	Minimum requested time of all bids
t_i, t_j	Requested time for bid i and residual time for instance j
d_i^r	Demand of resource type r in a bid i

TABLE II: Table of notation

Algorithm 2 Online auction algorithm (OA)

```
 1: Initiate $\lambda_{r,j} = \frac{t_{min}L^r}{2TRS}, x_{i,j} = 0, L^r, U^r$
 2: while Receiving bid $i$ do
 3:     Calculate utility:$u_i = b_i - \sum_r \lambda_{r,j} d_i^r$
 4:     if $u_i > 0$ and $t_i \leq t_j$ then
 5:         $j^* = \arg\max_j (b_i - \sum_r \lambda_{r,j} d_i^r)$, $x_{i,j^*} = 1$
 6:         $p_i = \sum_r \lambda_{r,j^*} d_i^r$
 7:         Update dual variable: $\beta_{j^*}^r = \frac{R_j^r - \sum_i x_{i,j} d_i^r}{R_j^r}$,
 8:         $\lambda_{r,j^*} = U^r \frac{t_{min}L^r}{2TRSU^r}^{\beta}$
 9:     else
10:         $x_{i,j} = 0$
11:     end if
12: end while
```

since the usage pattern is best understood by themselves. Upon receiving a request, the cluster will run the dispatch algorithm to determine whether it is a container request or an application request, and consult to the (Tier-1 or -2) scheduler accordingly. We give the details of the request dispatch process in Algorithm. 1, with symbols described in Table II.

Following the two-level scheduling, buyers need to submit their requests with specifications on containers, by which the Tier-1 scheduler decides where to allocate them, considering both the decommission deadline and resource constraint. Sample specifications are listed in Table III with specifications for database, web server, and batch job containers. Here CPU is expressed in relative units, where the higher the amount, the more CPU share the container can obtain. The maximum unit that can be specified for a CPU/vCPU is 1024.

C. Tier-1 Scheduling and Instance Trading

An integral component of our cloud system is this container allocation scheduler (Tier-1) as well as a market *mechanism* to facilitate the trading of recycled instances. To this end, we built an auction-based instance trading module to meet both requirements. Current pricing scheme in cloud markets is still fixed price dominant which usually does not lead to an efficient market. Auctions have been widely used to determine the clearing prices that reflect the demand and supply relationship in the market [8]. Our system differs from the previously studied scenarios in that (1) the resources are inherently constrained by each instance that are holding them. Treating each type of resource as a monolithic resource pool like previous works did is not applicable to our system; (2) Instance pool in our system is dynamic. To this end, we carefully modify the state of art auction mechanism [18] into our problem. The detailed algorithm is presented in Algo. 2, where the symbols used are listed in Table. II.

We set binary variable $x_{i,j}$ equal 1 if a container request i is allocated to instance j, otherwise, this container request will not be satisfied by HARV. Our unit price updating function is defined as $\lambda_{r,j} = U^r \frac{t_{min}L^r}{2TRSU^r}^{\beta}$, where $L^r = \min_i \frac{b_i}{d_i^r}$, $U^r = \max_i \frac{b_i}{d_i^r}$. The intuition behind this pricing function such is that the smaller the β, the fewer resources are left in the system. Once β equals zero, the marginal price is set to be the upper bound of the user's value per unit of resources. Under such circumstance, no bid can win the auction, guaranteeing the capacity constraint is satisfied. We are allowing as many requests as we can to be satisfied by the platform in the beginning, and becoming more conservative with the diminishing of resources.

Though the competitive ratio claimed in the original mechanism cannot be guaranteed anymore[5], our modified mechanism can still guarantee *truthfulness* and *individual rationality*, two important economical properties of a good auction. The individual rationality is guaranteed by our designed algorithm by ensuring that the utility for each selected requests is nonnegative. Since the pricing scheme of this mechanism falls into the family of *sequential posted price mechanisms* [19] in which truthful bid reporting is a dominant strategy, our algorithm guarantees the truthfulness in bid value.

D. Details in Resource Allocation and Sharing

HARV relies on the `cgroups` kernel feature to enforce the resource allocation decision made by the Tier-1 scheduler, and the `namespace isolation` kernel feature to enable sharing of resources among multiple buyers on the same residual instance. Specifically, each buyer's workload is encapsulated in a container which is associated with a resource vector $\vec{res}_i$ given in the buyer's request. Through container management tools, HARV translates the resource vector into corresponding *control groups* (cgroups), a collection of kernel controllers for system resources including CPU, memory, network and disk I/O. These controllers are assigned to the container runtime in the form of function hooking. When the container starts running, its resource access will trigger the corresponding hooks to ensure that the container does not use more than its resource share. Further, each container will be assigned a unique set of resource identifiers for its PID, IPC, network, and file system etc., providing it a runtime environment isolated from other co-located containers'. HARV creates a software

[5]While we leave the design of a more competitive mechanism as future works, our current mechanism is able to achieve high social welfare and cost savings, as we will show in Sec. IV.

Job Type	Duration (min)	Persistence	CPU Units	Memory (MB)	Network (Mbps)	Storage (GB)	Port
Web Front-end Server	35	No	256	100	200	0.1	80
Database	n/a	Yes	256	500	200	50	n/a
Sysbench	35	Yes	1024	512	100	0.5	n/a

TABLE III: Sample container request specifications given job types

bridge to allow co-located containers to share the host VM's network (with packet forwarding, NAT, and DNS configured).

E. Relocating/Migrating Containers

HARV is able to handle long-term task/containers through container migration. It configures the Linux `CRIU` (Checkpoint/Restore In Userspace) utility to checkpoint a running container, creates image files, sends those files to the next running destination, rebuilds and restarts the container. An advantage of this approach is that applications usually have dependencies such as OS binaries, third-party packages etc., while the container is able to encapsulate those runtime dependencies, making it convenient to restart an application without manually reconfiguring underlying host instance. Besides, buyers themselves (or the Tier-2 scheduler) can handle the instance decommission through data migration.

For batch tasks, migration can be efficient given it preserves computed results. For applications such as web front-end server, when instance decommission occurs, instead of migrating containers, a perhaps more efficient way is to simply treat it as container failures, and reassign the job to containers launched in other instances. It is worth noting that, in current version of our platform, there will be a service downtime from a few seconds to minutes depending on the check-pointed image size. Although batch tasks should not be affected much, those service downtime may not be tolerable for some user-facing applications.

F. Target Workloads

HARV is an ideal platform for running short jobs or the ones with limited persistent data. A variety of applications fit in this category, either in data processing including MapReduce accelerator [20], or the web front-end servers. We categorize the potential workloads for HARV and handling approaches based on their persistence and duration (long-term when user-specified duration exceeds the maximum allowable residual hour) as follows.

Long-term Stateless and Short-term Stateless HARV runs them in recycled instances and handles instance decommission through migrating or replicating containers (treats the decommission as an instance failure).

Short-term Stateful A migration deadline will be set for tasks of this kind. The larger the amount of state data, the earlier it is set prior to decommission deadline.

Long-term Stateful Since frequently migrating these jobs can be cost-prohibitive, our current version of HARV handles them by provisioning on-demand or reserved instances from the cloud provider, e.g. the database server shaded in dark gray in Fig. 5. Buyers can also launch those jobs in their own (non-recycled) public cloud instances while linking them to the accelerators deployed in recycled instances.

IV. Evaluation

In this section, we present the results of system-level benchmarking and trace-driven simulation on HARV. We show that HARV is able to attain high application performance with substantial cost savings.

A. System-level Evaluation

1) Prototype and Benchmarks Setup: We deployed a prototype of HARV with Amazon EC2 public cloud. We used *docker* as the containerization tool for the hybrid-virtualization setup. We run the master node that accepts instance recycle requests and hosts Tier-1 scheduler in an On-demand `m4.2xlarge` instance. We configured Amazon ECS service[6] to handle cluster state management (namely the state manager module). We modified its agent program to integrate it with the Tier-1 scheduler. Notably, except for the state management module, no other EC2 services were used in our system. Since such a module is commonly available in major cloud providers[7], HARV can be easily ported to other cloud platforms. Our testing cluster contains a maximum of a hundred residual `m4.large` instances. We chose the following two representative types of workloads:

Multi-tier Web Service: We used the RuBBoS[8] on-line forum benchmark to model the multi-tier application service. The number of containers provisioned for running the web and application servers is equal to the initial amount of recycled instance whose specifications are shown in Table III. We set up a load balancer for the web servers and deployed an emulated HTTP client[9] on an on-demand `m4.large` instance to request web pages from the server with different numbers of concurrent connections. We selected the average request rate and average request completion time as the performance metrics. We also sampled and calculated the average queue length in the load balancer.

Batch Task: We created batch workloads by devising a script that runs `sysbench` multi-threaded benchmark repeatedly (with a short sleep time between each run). We provisioned its container as specified in Table III.

We began the test by running only one recycled instance with one web server container and one batch task container requested. We used the HTTP client to launch page requests

[6]Amazon ECS: https://aws.amazon.com/ecs/

[7]Azure Container Service Cluster: https://azure.microsoft.com/en-us/documentation/articles/container-service-deployment/

[8]RUBBoS Bulletin Board Benchmark: http://jmob.ow2.org/

[9]Apache Benchmark: https://httpd.apache.org/docs/2.4/programs/ab.html

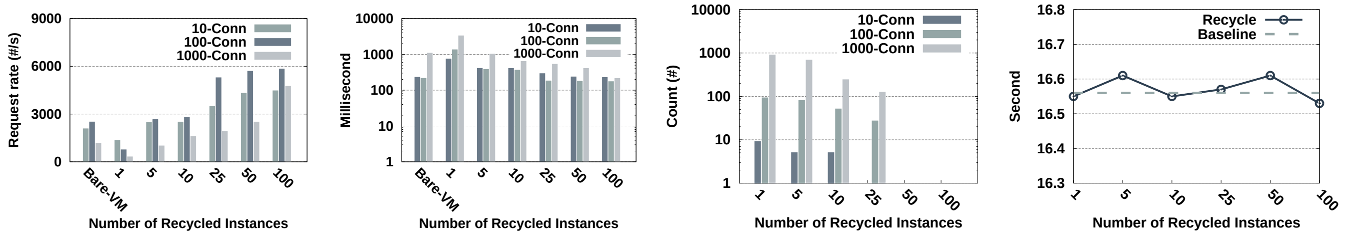
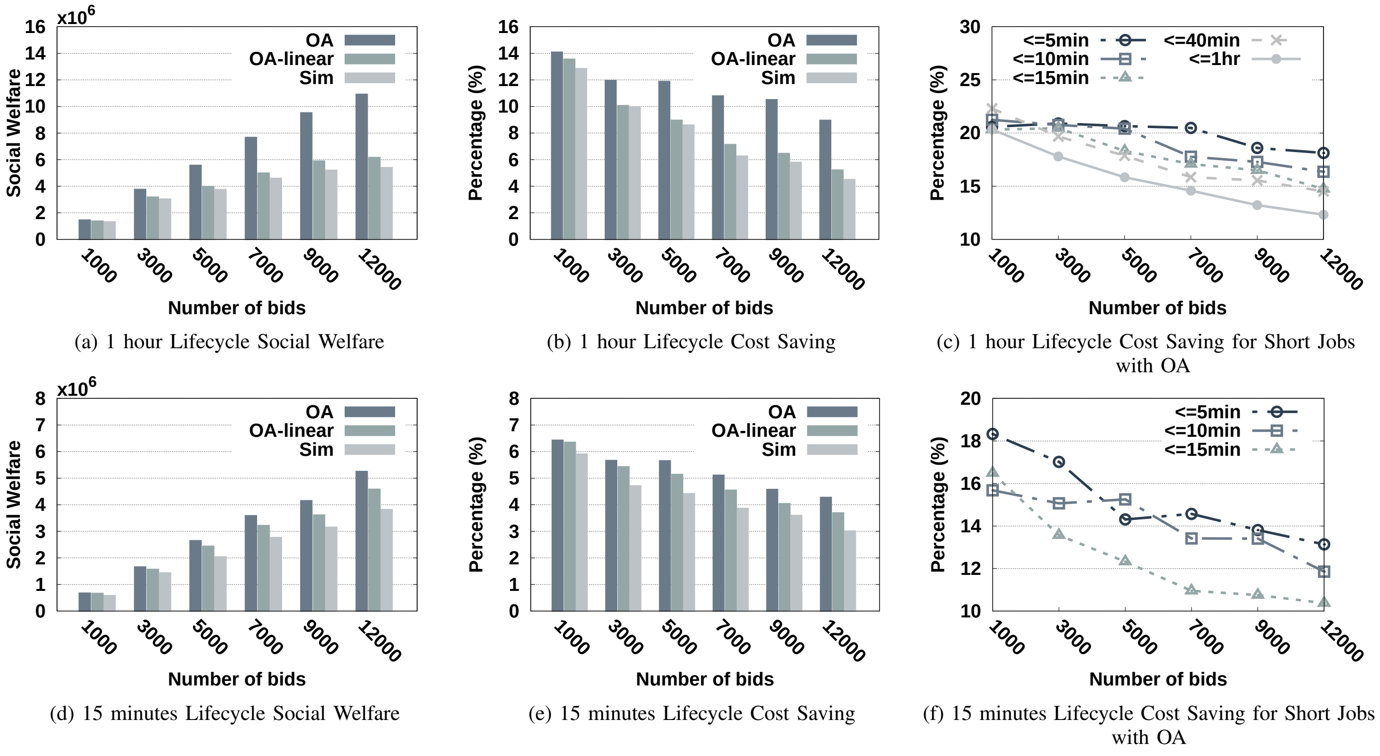

(a) Average Request Rate (b) Average Request Completion Time (c) Average Queue Length in Load Balancer (d) Average Batch Task Completion Time

Fig. 6: Real-world web application performance on HARV


(a) 1 hour Lifecycle Social Welfare (b) 1 hour Lifecycle Cost Saving (c) 1 hour Lifecycle Cost Saving for Short Jobs with OA

(d) 15 minutes Lifecycle Social Welfare (e) 15 minutes Lifecycle Cost Saving (f) 15 minutes Lifecycle Cost Saving for Short Jobs with OA

Fig. 7: Comparison of different online mechanisms in social welfare and cost saving

with 10, 100, 1000 concurrent connections consecutively. We collected the average request rate, throughput, request completion time as well as the number of queued requests in the load balancer. To ensure fairness, we waited until the load balancer queue was emptied before starting each new test. The client emulator is placed on an on-demand instance within the same cloud region in order to minimize the interference from the network. We then changed the number of recycled instance, the number of server container, and the number of batch task container to 5, 10, 25, 50, and 100, and repeat these tests. Finally, we obtained the baseline performance for both benchmarks by running each of them in a single, non-containerized `m4.large` instance.

2) Results: We present the benchmark results in Fig. 6. As shown in Fig. 6a, the baseline performance with a single `m4.large` instance (the "Bare-VM") is considerably higher than the single recycled instance case. This is due to, in the former case, the web servers being allowed to use all of the VM resources; whereas in the latter the servers are run in containers, and they have to share the resources with other co-located containers, thereby experiencing lower performance. Nonetheless, the average request rate immediately catches up with the baseline when there are five recycled instances and more. The effect of scaling out is also significant when there are more concurrent connections. For example, when the connection is 1000, an additional 50 recycled instances doubles request rate from around 2500 per second with 50 instances to near 5000 with 1000 instances.

Similarly, in Fig. 6b, the more recycled instances joining the HARV cluster the less the time to complete requests, with the average request completion time dropping from above 3000 ms the highest to 300 ms the lowest for 1000 connections. Particularly, when the number of recycled instances is higher than 10, the request completion time stays lower than the baseline across all concurrent connection settings. To confirm the effect of scaling, we depicted the average queue length in the load balancer in Fig. 6c. As can be seen, when the number of recycled instances is lower than 10, substantial amount of requests are buffered in the load balancer's queue, especially when the system experiences high concurrent connections. This high-buffering leads to the excessive delay in the request completion as observed in Fig. 6b. With more recycled instances and more web server container allocated, the queue length plummets, because requests can be immediately dispatched to available or idle servers. These results indicate that the additional container layer poses a minimal impact on user-perceived application performance. Considering the low (monetary) cost of HARV containers, HARV is a good choice for web service providers to provision for demand peaks.

For the performance of the batch tasks in Fig. 6d, the average job completion time stays almost unaffected throughout the experiments. As aforementioned, we assigned a better part of VM resources to the Sysbench container. This shows that in HARV, even if applications (with different resource usage patterns) share the same residual instances, HARV can still maintain their performance through differentiating the resources usage priorities. We attribute this achieved performance isolation to the use of containerization with each running container assigned an independent OS namespace and resource control groups (e.g. cgroups in Linux).

Finally, we measured the management module overhead. We found that when the recycled instance cluster is not trivially small, the overhead of running HARV is negligible. The management overhead (excluding the user-level overhead, e.g., the load balancer and Tier-2 scheduler) originates from the master node, state management module, Tier-1 scheduler, and the agent program on each recycled instance. In terms of monetary costs, the master node (running on `m4.2xlarge` On-demand instance in our prototype) costs \$0.479 per hour, an affordable price (in the real deployment, HARV can transfer some of the revenue to cover this cost) that can be further reduced by provisioning the cheaper reserved instance as the module will be running constantly. There is no additional charge for the state management module from EC2[10]. In terms of performance overhead, HARV takes 67.5 seconds (averaged over 100 instances) to setup a recycled instance, a process that includes instance cleansing, agent program installation, and containerization; and 4.2 seconds to handle a request (the time between receiving the request and starting the container; averaged over 5000 requests). For the agent program, HARV consumes less than 5% of CPU, limited memory footprint and network bandwidth. The results indicate that the time HARV takes to manage a recycled instance is considerably shorter than the billing cycle, leaving most of the residual instance time available to recycle.

B. Large-scale Trace-driven Simulations

1) Experimental Settings: In this part, we conduct simulations to evaluate the effectiveness and scalability of our trading module (Tier-1 scheduler). We select the publicly accessible Google Cluster trace [12] (also used in Sec. II), consisting of 3,535,030 entries, reporting each tasks' ID, active time, normalized resource demand (CPU, Memory), as well as task types, in an approximately 6 hours period. The time interval between each report update is 5 minutes. We identified 176,580 unique tasks after removing the reported anomalies, combining different entries that belong to the same task and calculating their durations.

To simulate the residual instances, we firstly assume each task request will be handled by a single on-demand VM/instance, and compute the corresponding residual instance information, including the time it being recycled as well as the residual hour according to Formula 1. Our event-driven simulator read the entries sequentially while checking the "current time" of the cluster; it adds and removes a residual instance to simulate the recycling and decommissioning process. Each requests are submitted to the scheduler; unsatisfiable requests are simply omitted.

2) Performance Metrics: We use cost saving and social welfare as our performance metrics. Cost saving is defined as the percentage of saving can be achieved by using our trading system compared with directly buying on-demand instances. Social welfare is the sum of utilities of all users and the auctioneer (i.e., $\sum_i (b_i - \sum_r \lambda_{r,j} d_i^r + \sum_r \lambda_{r,j} d_i^r) = \sum_i b_i$), as defined in Algo. 2, an indicator on how efficiently our system allocates resources to users who want them most. We test the system performance under different instance lifecycle. We choose 1 hour since it is one of the current prevalent instance life time settings. We also choose 15 minutes as the lifecycle to reflect current trends in designing fine-grained resource provisioning scheme in academia.

3) Trading Systems Compared: We compare our instance trading system with other one-off sale markets where an instance will only be sold once (i.e., no recycling will be involved). We implemented the online auction algorithm (Sec. III-C) in our trading system, whereas we adopted two other allocation algorithms in the one-off market. First, we adopted the similarity-based scheduling policy (Sim). In Sim, the vector similarity is computed between the container request vector and the instance vectors, and the instance with the highest similarity score is chosen to satisfy the request. Sim is a representative heuristic that being used frequently in designing cluster scheduling algorithms [21]. The second algorithm is the online auction mechanism with linear dual variable updates (OA-linear). In OA-linear, we change the dual variable update function in Algo. 2 to a linear function to validate the effectiveness of the original, exponential function. Finally, the pricing scheme in Sim is fixed, whereas OA and OA-linear both implement dynamic pricing.

4) Results: We present the experimental results in Fig. 7. First, as shown in Fig. 7a, given 1-hour lifecycle (current EC2 billing cycle setting) the proposed trading system with OA algorithm consistently achieves higher social welfare than the other two methods in a flat-rate market. Further, similar observations can be made in Fig. 7d where the lifecycle

[10]Amazon ECS Pricing: https://aws.amazon.com/ecs/pricing/

is reduced to 15 minutes. Notice that social welfare using 15 minutes lifecycle decreases when compared with the 1-hour counterpart is because we have fewer requests in the trace to be satisfied by the 15 minutes long instance. Though smaller billing cycle results in fewer resources available on the market to accommodate container requests, OA still achieves considerably higher performance than the other two methods.

Second, OA is able to maintain the cost savings even when the resource contention is high. In Fig. 7b, the cost saving of OA achieves 14% at 1,000 bids, and sustains over 10% except for the 12,000 bids scenario. For the other two methods, however, the cost savings drop from nearly 14% (OA-linear) and 13% (Sim) to below 10% when there are more than 3,000 bids, and plummet to around 5% at 12,000 bids. Cost savings of OA in 15 minutes lifecycle in Fig. 7e also exhibit similar superiority. Notably, even OA-linear implements a dynamic pricing scheme and Sim a fixed one, OA-linear achieves no better social welfare and cost savings than Sim. This confirms the importance and superiority of the pricing function in OA.

Third, in Fig. 7c and Fig. 7f, we intend to show the cost saving effects of our system to the jobs with different lengths. In the 1 hour lifecycle, our system constantly brings over 15% cost saving gains to all jobs with less than 40 minutes duration in all tested scenarios. Jobs with less than 5 minutes duration maintain around 20% cost saving. As we have explained before, the reduction of lifecycle brings less space for requests consolidating, which leads to smaller cost savings. However, jobs with less than 10 minutes duration still can benefit from 15% cost saving in 1000 bids to 12% cost savings in 12000 bids. As a conclusion, our auction-based trading system can achieve significant performance gain as compared to those one-off markets with either flat-rate or dynamic-rate.

V. Discussion

Before concluding our paper, we discuss the following issues pertaining to the adoption and practicality of the instance recycling service.

Provider's Incentives and Support: Although HARV tackles the general, third-party instance recycling problem where we assume the absence of cloud providers' support, there are indeed incentives for providers to support such service. Similar to Amazon EC2's spot instances (or Google Compute Engine's preemptible VMs), recycled instances is a cost-effective choice for certain types of workloads (see Sec. III-F). They both allow users to buy non-standard computing resources with a (likely) much lower price. On the other hand, there are major differences between recycled and spot instances. First, recycled instances can not only save costs for users who want to buy resources but also those who sell them, i.e., the residual instances owners. Second, unlike spot instances, recycled instances do not preempt workloads which allows users to run workloads that are not interruptible. Third, the resource offering of recycled instances are containers as opposed to VMs in spot instances. As such, while spot instances has become a widely used service, cloud providers can exploit recycled instances as another form of differentiated, value-added service to attract diverse user groups, gain extra revenue, and further improve their resource utilization. Cloud providers can either cooperate with a third-party instance recycling platform or build one themselves. Within the extent of our knowledge, HARV is the first work that addresses the motivation and technical challenges for building such service.

Trust and security issues: Trust and security issues have been one of the biggest concerns over cloud computing in general [22]. On the one hand, HARV targets for major public cloud deployment only and thereby these issues are not as pronounced as in other platforms relying on private, customer-supplied resources (e.g., from a private cloud or PCs) [23]. On the other hand, instance recycling indeed introduces new trust and security challenges. For example, residual instances suppliers and buyers should not have each other's data. In addition, malicious workloads should be prevented from sabotaging other co-located workloads. HARV relies on containers to provide resource isolation as discussed in Sec. III-D. At the policy level, the supplier is required to grant root privileges to HARV in order to successfully submit a residual instance to HARV (and they can choose to wipe out their data beforehand). HARV will then drop the supplier's root privileges and limit the local container manager to root access only (i.e., the docker daemon in our prototype). Notably, even if suppliers give up the root privileges, they will still be able to terminate the instance or modify its running through the cloud providers' API. Although HARV can blacklist the untrustworthy instance suppliers, a well-rounded solution in this case would require cloud providers' support. Moreover, advancements on cryptography allow more types of privacy-sensitive workloads to be run on third-party platforms such as HARV. For example, Order Preserving Encryption has been effectively used to preserve client's confidentiality for middlebox workloads running on the third-party cloud [24]. While in this paper we focus on other design dimensions of instance recycling, we will continue to address the trust and security issues in our future works.

VI. Related Work

Both cloud providers and users are faced with the resource inefficiency problem. While cloud providers have the luxury of improving their resource provisioning methods, cloud users may only leverage existing pricing options and/or application-level scheduling to alleviate the issue. HARV provides an alternative solution for cloud users by enabling them to resell their underutilized resources.

Resource Provisioning: Facing the resource inefficiency in current IaaS cloud, a substantial works have been done on designing fine-grained resource provisioning methods [8][25][26]. Most of these works focus on solving resource allocation problems from the provider's perspective, and they all operate on VM level. HARV can be treated as a cloud provider. Different from these works, HARV operates on containers for resource provisioning and does not belong to the IaaS category. The advancement of containerization techniques opens opportunities for cloud providers to supply flexible and efficient cloud resource offering. Containers bring less CPU consumption, less reboot time, smaller image size as compared to hypervisor based VMs [27]. They also introduce little application performance overhead as shown in our paper.

Pricing Options: Extensive works tried to improve cost-effectiveness for cloud users by leveraging and improving the existing pricing options [20][28][10]. Chohan *et al.* [20] explored the Spot Instance option to accelerate MapReduce jobs with greatly reduced monetary cost. Wang *et al.* in [10] exploited the Reserve option, and proposed a dynamic instance acquisition scheme that minimizes the broker's cost to accommodate given demands. Instead of exploiting existing billing options, we tackle this issue by introducing a new cloud instance type, which can be used jointly with those existing frameworks, too.

Customer-supplied Cloud: Wang *et al.* [23] studied a customer-supplied cloud (SpotCloud), where resources are provided from user's own physical machine instead of the public cloud. HARV can be viewed as a customer-supplied cloud, too. The major difference between HARV and SpotCloud is that HARV's resources are from the public cloud only. Compared to the residual instance we studied, a SpotCloud machine could introduce greater performance variance and trust issues.

VII. Conclusion and Future Works

In this paper, we proposed an instance recycling mechanism to address the prevalent partial usage waste problem faced by users in public IaaS cloud. We designed and implemented a system (HARV) to enable efficient instance recycling. HARV incorporates a container-virtualized layer to enable resource orchestration without provider-level supports. HARV adopts a two-level scheduling policy which offloads application-specific scheduling to its buyers while it only handles container allocations. Further, we designed an instance trading module, with an online auction to determine the market price. Evaluation on real-world workloads demonstrates HARV's practicality and scalability; and the large-scale simulation shows it can achieve considerable cost savings, even when the life-cycle is significantly shortened.

For future works, we plan to extend HARV to handle those non-residual instances. Cloud instances are often underutilized [7], if not completely idle, and those underused portion of resources can as well being reused. This will result in a generalized definition of residual instances, i.e. the instances with residual resources. New challenges will emerge from this more general problem, including how to estimate residual resources with precision and design pricing scheme with finer-granularity. Nevertheless, with HARV demonstrating the feasibility and benefits of instance recycling, we believe those challenges are worth addressing.

Acknowledgment

This research is supported by an Industrial Canada Technology Demonstration Program (TDP) grant, an NSERC Discovery Grant, and an E.W.R. Steacie Memorial Fellowship.

References

[1] R. Finos. Public cloud market forecast 2015-2026. [Online]. Available: http://wikibon.com/public-cloud-market-forecast-2015-2026/

[2] H. Jin, X. Wang, S. Wu, S. Di, and X. Shi, "Towards optimized fine-grained pricing of iaas cloud platform," *IEEE Trans. Cloud Computing*, vol. 3, no. 4, 2015.

[3] A. A. Hossain and E.-N. Huh, "Refundable service through cloud brokerage," in *Proc. IEEE CLOUD*, 2013.

[4] Y. Song, M. Zafer, and K.-W. Lee, "Optimal bidding in spot instance market," in *Proc. IEEE INFOCOM*, 2012.

[5] O. Agmon Ben-Yehuda, M. Ben-Yehuda, A. Schuster, and D. Tsafrir, "The rise of raas: the resource-as-a-service cloud," *Communications of the ACM*, vol. 57, no. 7, 2014.

[6] A. Bestavros and O. Krieger, "Toward an open cloud marketplace: Vision and first steps," *IEEE Internet Computing*, vol. 18, no. 1, 2014.

[7] L. Chen and H. Shen, "Consolidating complementary vms with spatial/temporal-awareness in cloud datacenters," in *Proc. IEEE INFOCOM*, 2014.

[8] L. Zhang, Z. Li, and C. Wu, "Dynamic resource provisioning in cloud computing: A randomized auction approach," in *Proc. IEEE INFOCOM*, 2014.

[9] F. Hao, M. Kodialam, T. Lakshman, and S. Mukherjee, "Online allocation of virtual machines in a distributed cloud," in *Proc. IEEE INFOCOM*, 2014.

[10] W. Wang, D. Niu, B. Li, and B. Liang, "Dynamic cloud resource reservation via cloud brokerage," in *Proc. IEEE ICDCS*, 2013.

[11] D. Williams, H. Jamjoom, and H. Weatherspoon, "The xen-blanket: virtualize once, run everywhere," in *Proc. ACM EuroSys*, 2012.

[12] J. L. Hellerstein, "Google cluster data," Google research blog, Jan 2010, posted at http://googleresearch.blogspot.com/2010/01/google-cluster-data.html.

[13] X. Zhang, Z. Huang, C. Wu, Z. Li, and F. C. Lau, "Online auctions in iaas clouds: Welfare and profit maximization with server costs," in *Proc. ACM SIGMETRICS*, 2015.

[14] M. Armbrust *et al.*, "Above the clouds: A berkeley view of cloud computing," *Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS*, vol. 28, no. 13, 2009.

[15] Gartnermq 2016. AWS's a leader in the IaaS market. [Online]. Available: https://aws.amazon.com/resources/gartner-2016-mq-learn-more/

[16] P. Sharma, S. Lee, T. Guo, D. Irwin, and P. Shenoy, "Spotcheck: Designing a derivative iaas cloud on the spot market," in *Proc. ACM Eurosys*, 2015.

[17] M. Ben-Yehuda *et al.*, "The turtles project: Design and implementation of nested virtualization." in *Proc. USENIX OSDI*, vol. 10, 2010.

[18] N. R. Devanur and Z. Huang, "Primal dual gives almost optimal energy efficient online algorithms," in *Proc. ACM-SIAM SODA*, 2014.

[19] S. Chawla, J. D. Hartline, D. L. Malec, and B. Sivan, "Multi-parameter mechanism design and sequential posted pricing," in *Proc. ACM Symposium on Theory of Computing (STOC)*, 2010.

[20] N. Chohan *et al.*, "See spot run: Using spot instances for mapreduce workflows," in *Proc. USENIX HotCloud*, 2010.

[21] R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella, "Multi-resource packing for cluster schedulers," in *Proc. ACM SIGCOMM*, 2014.

[22] S. Pearson and A. Benameur, "Privacy, security and trust issues arising from cloud computing," in *Proc. IEEE CloudCom*, 2010.

[23] H. Wang, F. Wang, J. Liu, and J. Groen, "Measurement and utilization of customer-provided resources for cloud computing," in *Proc. IEEE INFOCOM*, 2012.

[24] C. Lan, J. Sherry, R. A. Popa, S. Ratnasamy, and Z. Liu, "Embark: Securely outsourcing middleboxes to the cloud," in *Proc. USENIX NSDI*, 2016.

[25] G. Feng, S. Garg, R. Buyya, and W. Li, "Revenue maximization using adaptive resource provisioning in cloud computing environments," in *Proc. ACM/IEEE Grid*, 2012.

[26] M. Hadji and D. Zeghlache, "Minimum cost maximum flow algorithm for dynamic resource allocation in clouds," in *Proc. IEEE CLOUD*, 2012.

[27] L. Li, T. Tang, and W. Chou, "A rest service framework for fine-grained resource management in container-based cloud," in *Proc. IEEE CLOUD*, 2015.

[28] L. Zheng, C. Joe-Wong, C. W. Tan, M. Chiang, and X. Wang, "How to bid the cloud," in *Proc. ACM SIGCOMM*, 2015.