Opportunistic Service Provisioning in the Cloud

Mathias Björkqvist
IBM Research Zurich Lab
Rüschlikon, Switzerland

Lydia Y. Chen
IBM Research Zurich Lab
Rüschlikon, Switzerland

Walter Binder
University of Lugano
Lugano, Switzerland

***Abstract*—There is an emerging trend to deploy services in cloud environments due to their flexibility in providing virtual capacity and pay-as-you-go billing features. Cost-aware services demand computation capacity such as virtual machines (VMs) from a cloud operator according to the workload (i.e., service invocations) and pay for the amount of capacity used following billing contracts. However, as recent empirical studies show, the performance variability, i.e., non-uniform VM performance, is inherently higher than in private hosting platforms, since cloud platforms provide VMs running on top of typically heterogeneous hardware shared by multiple clients. Consequently, the provisioning of service capacity in a cloud needs to consider workload variability as well as varying VM performance. We propose an opportunistic service replication policy that leverages the variability in VM performance, as well as the on-demand billing features of the cloud. Our objective is to minimize the service provisioning costs by keeping a lower number of faster VMs, while maintaining target system utilization. Our evaluation results on traces collected from in-production systems show that the proposed policy achieves significant cost savings and low response times.**

I. Introduction

Deploying services in cloud environments is an attractive solution, due to cost and ease of management advantages. Hosting services in a cloud relieves the service provider from maintaining an expensive computing infrastructure. Thanks to on-demand virtual resource provisioning, cloud operators such as Amazon Elastic Compute Cloud (Amazon EC2) [4] provide on-demand computing capacity, enabling elastic service provisioning. Another advantage of cloud environments is their pay-as-you-go billing feature. The service provider can thus request the necessary computing capacity in the unit of Virtual Machines (VMs) from the cloud operator, according to the workload. Consequently, hosting services in a cloud — in conjunction with an effective service replication policy — can achieve significant cost savings for the service provider.

Recent studies [6], [7], [9], [13], [19] report empirical experiences of migrating various applications onto cloud platforms and point out a common weakness of cloud environments — higher performance variability than on private platforms. In particular, VMs with the same specification (i.e., incurring the same costs for the user) show significant performance variability in terms of throughput[1]; some VMs are faster and some are slower. This can be explained by the fact that the cloud operators may consolidate multiple VMs on the same physical machine, resulting in resource sharing and performance interference. The effects of resource sharing are dynamically changing depending on varying workloads and on workload management actions taken by the cloud operator, such as VM consolidation and VM migration. Furthermore, hardware features such as dynamic frequency scaling can have an impact on performance depending on the workloads and on VM consolidations. As a consequence, the computing capacity of individual VMs fluctuates, and so does the aggregated capacity of all provisioned VMs of a service provider.

In addition to the performance variability, another distinguishing difference between private and cloud platforms is the cost structure and restrictions imposed by the billing contract. On a private platform, turning a VM on and off is not restricted by any billing contract, whereas VMs requested in a cloud are typically charged for pre-defined billing periods; e.g., one hour is currently the smallest billing period in Amazon EC2. Therefore, in a cloud it can be wasteful to turn VMs on and off without considering billing constraints. Moreover, frequently turning VMs on and off may cause not only additional costs but also some capacity loss because of the time overhead associated with the VM control actions. System performance (i.e., service response times) can fluctuate greatly during the transition of turning VMs on and off.

On the one hand, cloud platforms provide several cost advantages for elastic service provisioning. On the other hand, system dynamics become much more complex than in private platforms and pose several new challenges. Purely workload-driven service replication policies have been shown effective on private platforms [2], [10], [16], [17], implementing simple control actions such as turning service replicas on and off. However, such policies can fall short in optimizing the trade-off between cost and performance in a cloud, due to the lack of consideration of the variability in VMs' performance and billing contracts. For example, in a cloud, a lower number of faster VMs may have the same aggregate capacity as a higher number of slower VMs, but typically cost less, particularly if the faster and the slower VMs are not distinguished by the billing contracts. To optimize service provisioning costs and service performance simultaneously, the service replication policy in the cloud needs to choose not only the right number of VMs but also the VMs with better performance. As such, a broad range of criteria, such as workload, heterogeneity of VM performance, and billing contracts, needs to be taken into consideration when designing service replication algorithms for cloud environments.

[1]We interchangeable use the performance variability and throughput variability when refereing to the non-uniform performance of VMs in the cloud.

In this paper we develop an opportunistic replication policy for elastic service provisioning on cloud platforms. Our objective is to leverage the variability in VM performance and their billing contracts in a cloud such that the VM costs of all services hosted by a provider are minimized, while maintaining given system utilization. Our policy takes several control actions in a slotted window: turning VMs on and off, replacing slower VMs in the hope of getting faster ones, and reconfiguring VMs from one type of service to another. All these actions are associated with non-negligible time overhead. The criteria are the predicted workload, estimated VM performance, target system utilization, and billing contract periods. In particular, we deploy VM controllers for all types of services to collect statistics required by aforementioned the criteria. We implement an opportunistic policy within a VM broker, which coordinates the VM control actions across different services. To evaluate the cost and performance gain of our proposed policy, we built a detailed simulator for services hosted in a cloud, driven by service utilization traces collected from real IBM production systems. Our evaluation results show that the proposed opportunistic replication policy achieves significantly lower service provisioning costs than workload-oblivious or purely workload-driven policies.

The original scientific contribution of this paper is a novel service replication policy, which is specially designed to explore the variability of VM performance on cloud platforms. In contrast to existing replication polices, we optimize the cost and performance not only for a single service but also for the entire system, by an augmented set of control actions, in particular replacing and reconfiguring VMs. Our evaluation environment encompasses a large number of different parameters, such as different time overheads associated with each control action. The proposed opportunistic replication policy is shown to achieve lower cost and better performance for services hosted in the cloud, compared to replication policies oblivious to the unique characteristics in the cloud.

This paper is organized as follows: The system architecture is explained in Section II. The proposed opportunistic replication policy in Section III. Section IV contains the experimental results. Related studies are summarized in Section V. Section VI concludes this paper.

II. System Overview

A. System Architecture and Dynamics

Figure 1 illustrates the system architecture considered in this paper. A service provider deploys I types of services S_i ($1 \leq i \leq I$) in a cloud. The services considered here are simple atomic ones (i.e., we do not focus on composite services that invoke other services). At any given moment, there are $n_i \geq 1$ VMs running a service of type i; we also say there are n_i replicas of service i in the cloud. The values n_i may change over time according to the actions taken by the policy presented in this paper. However, there is always at least one replica for each service.

To limit the scope of this study, we assume that all services are CPU-bound and multi-threaded, that is, capable of handling several concurrent invocation requests in parallel. We also assume that the service execution time is not significantly influenced by the input parameters passed upon service invocation.

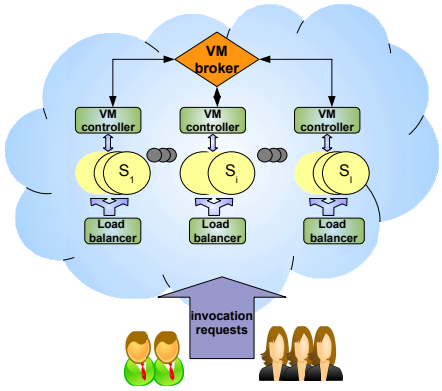


Fig. 1. Schematics of services system deployed in a Cloud platform.

For each service, there is a corresponding load balancer and VM controller that are deployed in the cloud, too. The load balancer distributes incoming invocation requests to the replicas of the requested service (i.e., to the currently active VMs running a service of the corresponding type) with the fewest outstanding requests. We assume that the size of invocation requests varies, following an exponential distribution, and thus the execution times of requests follow an exponential distribution for a given VM throughput. The VM controller monitors active VMs and keeps tracks of statistics about the invocation rate, VM performance, and the billing periods of the active VMs. All controllers communicate the statistics to the VM broker, on which the proposed opportunistic policy and the control actions are implemented.

The throughput of a VM (i.e., its performance) is not fixed but changes over time, due to the possibly heterogeneous infrastructure used by the cloud operator, hardware optimizations that result in performance fluctuations, performance interference of multiple VMs consolidated on the same physical machine, and VM migrations. In this paper, we assume that the average performance of VMs with the same specification fluctuates in the discrete range of values. The specific values of VM performance can be estimated by observing the completed service requests. Each VM is bound to a contract that defines the billing period (e.g., one or several hours). That is, releasing a VM before the end of a billing period would be wasteful for the service provider who would still have to pay until the end of the period.

B. VM Replica Provisioning

Here, the VM replication provisioning is implemented in slotted windows. The length of the control windows depends on the dynamics of the workload and the parameterization of the service replication policy. We assume that the billing period is a multiple of the algorithm's execution interval. In our simulation, we use a billing period of one hour. To dynamically provide VM replicas in a cost-effective manner,

the VM broker considers four kinds of control actions: (1) turn on a new VM; (2) turn off a VM (i.e., terminating the contract at the end of a billing period); (3) replace a VM at the end of a billing period, if the VM is suspected of not performing well; (4) reconfigure a (previously allocated) VM to run a service of a different type. The first three actions are requests towards the cloud operator, while the fourth action is transparent to the cloud operator.

There are some time overheads associated with each action. The turning on of a new VM is assumed to take υ seconds to load and start the required service. The VMs that are about to be turned off need to immediately stop receiving invocation requests and complete the remaining invocations. As for VMs reconfigured from one service to another one, they no longer receive invocations of the former service and start serving invocations of the new service right after completing the remaining requests and after the reconfiguration process. Here, similar to the process of loading services, we assume the configuration time takes also υ seconds. The newly turned-on and reconfigured VMs are published as "available" VMs after the completion of their loading/configuration process. Note that previous related studies [2], [10], [16], [17] tend to overlook the overhead structure and lead to a simplified replication policy.

III. Opportunistic Replication Policy

Following the rule of thumb practiced in today's resource management [2], [20], we provide sufficient VMs to each service such that the VMs' aggregate capacities are well utilized. A typical target utilization could be around 80% [15], for handling temporary workload variation. Here, we aim at achieving better performance metrics, e.g., response time, and maintain target utilization at a lower cost, by leveraging a pay-as-you-go billing model and the variability in VMs' performance in the cloud.

We develop an opportunistic replication policy and implement it in the VM broker. In contrast to replication policies in private platforms, our proposed policy decides not only on the number of active VMs per service but also intends to acquire VMs with better performance. The general idea of our opportunistic policy is that the VM broker first decides on the number of VMs for each service, based on the information monitored/collected in VM controllers. The second step is to select specific VMs, using appropriate control actions. The selection criteria considered are the billing period, the difference in the number of VMs in adjacent windows, and the performance of active VMs. Each controller then executes the decisions made by the broker. In the following text, we first describe the control timing of the broker, the algorithm to decide on the number of VMs for each service, and finally the algorithm to select VMs.

A. Control Window and Overhead

Herein, we consider a control window of fixed length, τ minutes, indexed by t. Due to the time overhead associated with each control actions, the VM broker queries the required

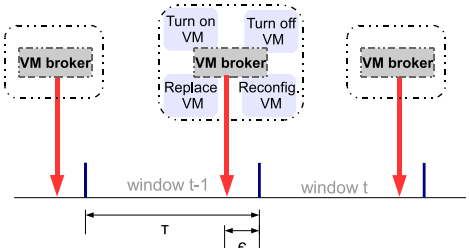


Fig. 2. Timing of control actions and windows for the VM broker.

statistics from controllers at ε seconds before the beginning of every control window. The schematics are depicted in Fig. 2. Controllers of services immediately send back their invocation rate, VM performance, and their billing periods. Using the collected information and algorithms described in the following two subsections, the broker computes and broadcasts decision of VM replication and selection to all controllers. We assume that such a decision process takes a negligible time and no time overhead occurs. We choose such a value of ε that there is a sufficient time for turned-on VMs to load the services, turned-off VMs to complete the remaining service invocation, replaced VMs to complete pending requests and replacement VMs to load the new service, and reconfigured VMs to complete pending requests and load the new service.

B. Number of Replica VMs

To proactively provide a sufficient number of well-utilized VMs at the beginning of every control window, the broker needs to know the utilization of active service VMs by the estimates of the average invocation rates, and the aggregate capacity.

We define the utilization of active service i VMs, U_i, as the invocation rate, λ_i, divided by the aggregate capacity of active VMs, i.e., $\sum_{j=1}^{n_i} \mu_{ij} = n_i \mu_i$, where $\mu_i j$ denotes the throughput of VM j for service i and μ_i denotes the average throughput per active VMs,

$$U_i = \frac{\lambda_i}{\sum_{j=1}^{n_i} \mu_{ij}} = \frac{\lambda_i}{n_i \mu_i}. \tag{1}$$

Note that as there can be multiple threads in a replica VM, the performance of a VM corresponds to the summation of all the threads. Our objective of VM provisioning is that the effective utilization of every service at every window is less than the target value, $U_i(t) < U^*, \forall i,t$. To that end, we first need to estimate the average invocation rate and average capacity for each coming control window. We propose to use a simple last value prediction for the invocation rate,

$$\widehat{\lambda_i(t)} = \lambda_i(t-1), \tag{2}$$

and for the average throughput,

$$\widehat{\mu_i(t)} = \mu_i(t-1) = \frac{\sum_{j=1}^{n_i(t-1)} \mu_{ij}(t-1)}{n_i(t-1)}. \tag{3}$$

Substituting the estimated values of Eq. 2 and 3 into Eq. 1, the broker estimates the utilization of service i when deploying

$n_i(t)$ VMs at the beginning of window t,

$$U_i(t) = \frac{\widehat{\lambda_i(t)}}{n_i(t)\widehat{\mu_i(t)}}. \quad (4)$$

After straightforward algebraic manipulation, the broker controls $n_i(t)$ such that $U_i(t) \leq U^*$, as follows,

$$n_i(t) = \lceil \frac{\widehat{\lambda_i(t)}}{U^*\widehat{\mu_i(t)}} \rceil, \forall i,t. \quad (5)$$

Once the broker obtains the values of $n_i(t)$, it proceeds to decided on which VMs to be turned off, replaced, reconfigured, and how many new VMs to be turned on.

C. Turning on-off, Replacing, and Reconfiguring VMs

The objective of selecting VMs is to maintain as few VMs as possible and to keep as many fast VMs as possible, such that the return on payment for active VMs is maximized. Consequently, the broker only turns off the expiring VMs, whose billing contracts end, and only turns on new VMs for services when there is no spare capacity from other services. In general, the broker is greedy in maximizing the "benefit" of individual services, rather than the global welfare of all services, when it comes to decreasing VMs. The broker also increases VMs in a collaborative manner, as elaborated in the following.

The decision process of the broker is structured into two parts. The first part focuses on the services which need to reduce VMs, and the second part focuses on services which need to increase VMs, from their current provision. Critical parameters considered are the number of expiring contracts, $E_i(t)$, the difference in the number of VMs in adjacent windows, $\delta_i(t) = n_i(t) - n_i(t-1)$, and the performance of the VMs. Expiring VMs can be either turned off, reconfigured to other services, or replaced by other VMs, whereas non-expiring VMs can only be reconfigured. When $\delta_i(t) > 0$, the service i demands more VMs at window t, whereas when $\delta_i(t) < 0$, the service i tries to reduce VMs by turning off or reconfiguring, given the possibility. It implies that not all services can always reduce VMs as the workload decreases shown in Eq. 5, due to the billing periods and no available services requiring more VMs.

To facilitate selecting control actions for increasing and decreasing VMs, we keep two lists, an expiring list and a reconfiguration list, recording expiring VMs and reconfigurable VMs, respectively. The lists are filled up during the "decreasing" part of the policy and flushed out during the "increasing" part of the policy. Both lists are maintained in a slowest-first manner. For example, the broker always selects the slowest expiring VMs first onto the expiring list and distributes the slowest VMs first to the services with $\delta_i(t) > 0$.

1) Decreasing VMs: For services with $\delta_i(t) < 0$, the broker greedily optimizes the aggregate VM capacity of individual services by turning off expiring VMs or replacing the slow expiring VMs with faster ones.

When there are more expiring VMs than reduced VMs, $E_i(t) > |\delta_i(t)|$,, where $|*|$ denotes the absolute value and $|\delta_i|$ is the number of VMs needs to be reduced. the broker first turns off $|\delta_i(t)|$ expiring and slowest VMs. Then, the broker tries to replace remaining expiring VMs by comparing the corresponding cost and benefit. The cost of replacing VMs is the unavailability of its capacity during the time a replacement VM is being configured. The potential benefit is the chance of obtaining a VM with better performance. We thus derive the quantitative cost of replacing a VM j of service i as

$$C_{ij} = \varepsilon\mu_{ij}. \quad (6)$$

On the other hand, we derive the benefit to replace expiring VM j of service i as the expected capacity gain, which sums the product of probabilities of throughput levels, the throughput differences, and the control window length. Assuming a VM has K different levels of throughput and the probability of receiving a VM with throughput level k is P_k, one can write

$$B_{ij} \sum_{k \neq j}^{K} P_k(\mu_{ik} - \mu_{ij})\tau. \quad (7)$$

When the benefit of replacing VM j of service i is greater than the cost, $B_{ij} > C_{ij}$, the broker replaces VM j by a new VM. Note that the replacing decision may not necessarily lead to a faster VM. From Eq. 6, one can see that probabilistically speaking, it is beneficial to replace VMs especially when currently expiring VMs are slow and the control window is longer.

When there is not a sufficient number of expiring VMs to be reduced, i.e., $E_i(t) < |\delta_i(t)|$, the broker first turns off $E_i(t)$ expiring VMs. Then, among the non-expiring VMs, it chooses the $\{|\delta_i(t)| - E_i(t)\}$ slowest VMs and adds them to the reconfiguration list, Ψ. Such a list is first filled up by services with $\delta_i(t) < 0$ in a sequential order of service index, and then flushed out by services with $\delta_i(t) > 0$ in a round robin fashion for reasons of fairness. As such, the time overheads associated with replacing and reconfiguring VMs can be minimized.

2) Increasing VMs: Once the broker completes the process of increasing VMs, it proceeds to the services requiring additional VMs. The broker first tries to distribute any available VMs on the reconfiguration list, and then turn on new VMs where required. Let the total number of VMs on the reconfiguration list be $n^{\Psi}(t)$, and total number of additional VMs from services with $\delta_i(t) > 0$ be $\Delta = \sum_{i \in \{\delta_i(t) > 0\}} \delta_i(t)$. When $n^{\Psi}(t) > \Delta$, it implies a sufficient number of VMs can be reconfigured and then distributed to other services. The broker distributes the slowest Δ VMs on the reconfiguration list to services with $\delta_i(t) > 0$ in a round-robin fashion. Then, the remaining $\{n^{\Psi}(t) - \Delta\}$ VMs on the reconfiguration list are returned to their original services. On the other hand, when there is not a sufficient number of VMs on the reconfiguration list to meet the requirement for an increasing number of VMs, the broker only distributes $n^{\Psi}(t)$ VMs in a round-robin fashion and turns on additional VMs according to unfulfilled demands. We summarize the opportunistic replication policy implemented on the broker in Algorithm 1.

Algorithm 1 Opportunistic Replication Policy on the Broker

Compute $n_i(t)$ as in Eq.5 and $\delta_i = n_i(t) - n_i(t-1)$, $\forall i$.
for $i = 1$ to I services, with $\delta_i(t) \leq 0$ **do**
  if $E_i(t) > |\delta_i|$ **then**
    Turn off $|\delta_i|$ expiring VMs
    Replace up to $\{E_i(t) - |\delta_i(t)|\}$ servers based on Eq. 7, and 6.
  else
    Remove $E_i(t)$ expiring VMs
    Add **slowest** $\{|\delta_i(t)| - E_i\}$ VMs to the reconfiguration list, Ψ.
  end if
end for
for For services with $\delta_i(t) > 0$ **do**
  if $n^{\Psi}(t) > \sum_i \delta_i(t) = \Delta$ **then**
    Distribute Δ VMs on the reconfiguration list round-robinly
    Return remaining $\{n^{\Psi}(t) - \Delta\}$ VMs back to original services
  else
    Distribute $n^{\Psi}(t)$ VMs on the reconfiguration list in round-robin
    Add $\{\Delta - n^{\Psi}(t)\}$ servers and distribute to the corresponding services in round-robin
    Empty the reconfiguration list
  end if
end for

IV. Evaluation

In this section, we use trace driven simulation to evaluate the proposed opportunistic replication policy for service systems deployed in the cloud. The performance metrics evaluated are the VM costs, the average normalized throughput of VMs, the average utilization of active VMs, and the average response time of an invocation. To show the effectiveness of the VM broker, we also present the detailed statistics of control actions. We benchmark our policy against a workload oblivious replication policy and a purely workload driven policy. Our evaluation results, based on the average of ten simulation runs, show that the VM broker can significant reduce the cost by acquiring a smaller number and faster VMs in a collaborative manner, while maintaining the system utilization slightly lower than the target values, $U^* = 80\%$, and achieving low average response times of invocations.

A. System Configuration

We built a trace-driven simulator of service-oriented systems in the cloud using Java. Invocation requests are generated for each service, following a Poisson process with time varying arrival rates. Each VM replica is configured to have one thread, independent of service types. Moreover, we assume a VM can have three different performance to process requests of each service in our simulated cloud environment. To ease the analysis, we express VM throughput as a multiplier of the minimum throughput of each service, $\alpha\mu_i$. The specific values are $\alpha = \{1, 1.2, 1.5\}$, i.e., a VM has an average throughput of μ_i, $1.2\mu_i$ and $1.5\mu_i$ for processing request of service i. The probabilities of obtaining VMs with different α values are 0.5, 0.3, 0.2 respectively. The aforementioned values can be configured according to values measured in different cloud platforms.

The VM controller collects the required statistics $\varepsilon = 20$ seconds before every control window of length $\tau = 5$ minutes and the VM broker immediately computes and implements the control actions, some of which have time overhead of $\upsilon = 20$ seconds. For a fair comparison, the timing of control actions in policy II are synchronized with the broker. The specific length of the control window is chosen according to workload characteristics and prediction schemes. The discussion of the optimality of those values is beyond the scope of this paper.

Cost Calculation

We follow the convention in today's commercial cloud [4] and use an hour as the billing period. The actual cost per billing period is different between cloud providers, and also varies depending on the requested VM specifications. We present our results in terms of relative cost savings, and the results are therefore not bound to any cloud provider in particular.

The total cost is the summation of all requested VM hours. When VMs are turned off before the end of billing period, they still need to pay for the remaining minutes. We add the cost for any possibly remaining periods immediately onto the windows when those VM are turned off.

B. The Workloads: Invocation Requests

Following approaches used in [2], [12], [20], we adopt the utilization traces from current IBM production systems as workload input for each service, i.e., to generate the invocation requests. Based on the basic utilization law [8], the utilization multiplied by a normalized constant reflects the request rate, especially when the load is below 100% utilization.

We collect utilization traces from four large multi-processor servers engaging in web services in financial, airline and media industries, in late January, 2012. The trace from one server is considered as one service here. The utilization values are the average computed over 15 minutes. To obtain the request rate per second, we multiply the utilization values with the processing power of the server, i.e., the number of cores. We illustrate the rationale by an example: Let the utilization value be 35% for a 16 core server. This implies that, on average, 5.6 $(0.35 \cdot 16)$ cores are busy. We further assume that a core is occupied by a single request and such a value corresponds to the request arrival rate for a small granularity, i.e., second. As such, we obtain the request rates for four services, shown in Fig. 3. One can clearly see that the workloads are time-varying.

The execution times of each service follow the exponential distribution with mean $\frac{1}{\mu_1} = \frac{1}{1}, \frac{1}{\mu_2} = \frac{1}{1}, \frac{1}{\mu_3} = \frac{1}{10}, \frac{1}{\mu_4} = \frac{1}{8}$ seconds respectively. Note that due to the performance variability of VMs, the execution times can be scaled down by the multiplying factor, α, by 1.2 or 1.5.

Due to limitations in the granularity in collecting utilization we are unable to collect the higher moment statistics and further fit the empirical distribution of utilization. Consequently, we assume that the arrivals of the requests follow Poisson processes for each 15 minutes and that their means fluctuate according to Fig. 3. Once requests are generated, they are then immediately forwarded to the corresponding and available service load balancers.

We compare the proposed policy against the following policies, which are oblivious to the variability of the workload, heterogeneity of VM throughput, or billing periods:

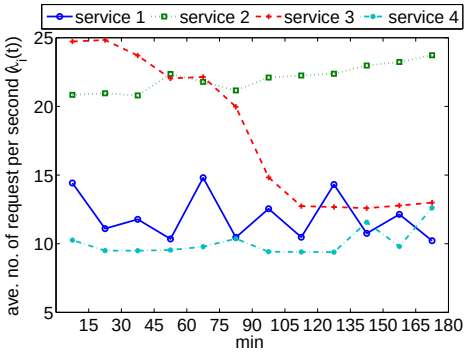
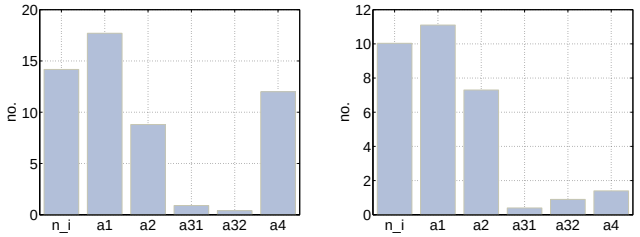


Fig. 3. Average request rates of services, $\lambda_i(t)$.



(a) Service 1 (b) Service 2

Fig. 4. Average number of active VMs per window(n_i) and average number of VM used in each control action: turn off (a1), replace (a2), reconfigure off (a31), reconfigure on (a32), turn on (a4).

- Policy I: statically providing maximum number of VMs for each service. This is a workload oblivious policy.
- Policy II(o), Policy II(p): dynamically providing VMs for each service, according to the workload only. This is a purely workload-driven policy. The number of VMs is decided by Eq. 5, but based on the minimum VM throughput only. We provide two versions of Policy II, namely II(a), and II(p). The former one uses the actual request rate information and the later one uses the predicted ones in Eq.2.

We also use the actual and predicted invocation rates in the proposed broker, and denote them broker(a) and broker(p) in the following. The difference between these two versions comes from the performance degradation due to the inaccurate workload prediction.

C. Two Services

We first evaluate the VM broker on a system with two services, namely service 1 and 2 shown in Fig. 3. We summarize the results in terms of cost saving, average utilization, average response time, and average normalized VM throughput in Table I. The cost savings are compared with the cost of policy I, where the provisioning costs are the highest. The normalized VM throughput is calculated from the observed VM throughput divided by the minimum throughput for each service.

Clearly, static provisioning of VMs in policy I incurs high costs, and results in low utilization of VMs, as well as lower response times. Policy II saves costs for both services, compared to the policy I, because of frequently turning VMs on and off, and being oblivious to the performance variability of VM throughput. However, policy II has a much lower cost saving than the broker, especially for service 1, whose workloads are more stable and the cost savings from dynamic VM provisioning is smaller. In contrast, the broker can leverage the control knob of ”replacing” VM for less varying workloads and acquire faster VMs, which in turn results in a lower number of VMs provisioned and subsequently lower cost. As for the utilization, the broker maintains the utilization at roughly 70 %, which is slightly lower than the target value of 80 %. Overall, the broker has the highest cost savings, medium utilization, the lowest response time, and the highest normalized VM throughput.

Furthermore, we present average statistics about control actions and the number of VMs provisioned in Fig. I. We denote a1, a2, a31, a32, and a4 as the number of VM, which are turned off, replaced, and reconfigured to another services, reconfigured from another service VM, and turned on, respectively. The first three actions are associated with ”decreasing VMs”, while the latter two are for ”increasing” VMs. One can see that there is a higher number of reconfiguration and lower number of replacing occurring for service 2, because of a higher oscillation in workloads compared to service 1. Due to a higher variety of control actions taken for service 2, the broker is able to obtain VMs with higher throughput, i.e., the average normalized throughput of service 2 is higher than service 1. We conclude that the broker can apply different control actions according to different dynamics of workloads, and further gain cost savings without sacrificing the performance.

D. Four Services

Secondly, we evaluate the VM broker on a system with four services, namely service 1-4 shown in Fig. 3. We present the cost savings, average utilization, average response time, and average normalized VM throughput under different policies in Table II. The statistics of the control actions used in the proposed broker are summarized in Fig. 5. Similar to the observations made in previous sections, we can see that the proposed VM broker can achieve significantly higher costing savings than other policies, while adhering to the utilization target and attaining lower response times. For the services with less varying workloads, i.e., services 1, 3 and 4, the broker can gain cost saving by replacing slower VM with faster VMs, whereas policy II can only gain marginal cost savings, compared to the static provisioning. For service 2, although policy II and the broker can gain good cost savings, the broker still obtains twice as high savings as policy II due to effective replacing and reconfiguration of VMs.

As the broker applies VM control actions in a collaborative manner, especially through the reconfiguration, a better performance gain can be achieved by the broker in systems with a higher number of services. The overall performance in the case of four services is better than in the case of two services. One can observe this especially by comparing service

TABLE I
PERFORMANCE OF DIFFERENT POLICIES: TWO SERVICES.

Policy	Total	S1				S2			
	CS[%]	CS[%]	U[%].	RT[s]	AT	CS[%]	U[%].	RT[s]	AT
I	0	0	55.8	0.91	1.14	0	49.4	0.45	1.15
II(o)	14	7	74.6	1.08	1.16	21	74.9	0.55	1.17
II(p)	14	7	74.1	2.18	1.17	21	73.7	0.55	1.16
Brok.(o)	32	27	68.9	0.96	1.22	38	72.7	0.52	1.23
Brok.(p)	31	25	68.0	1.75	1.22	37	71.6	0.50	1.23

cs=normalized cost savings, U=utilization,
RT=average response time of invocation, AT=average normalized throughput of VMs

1 and 2 in both system scenarios. In particular, the average utilization of VMs for each service increases slightly and gets closer to the target value (80%). Both services also achieve higher cost savings in the four services scenario, because of more opportunities to reconfigure VMs to different services. From our evaluation, we believe our proposed policy can opportunistically acquire a fewer and faster VMs for different workloads, and its effectiveness grows with the scale of the system, i.e., with a higher number of different services.

Discussion: We would like to point out a few limitation of our study. First, this study considers only atomic services, and modeling dependencies among services (i.e., composite services) will be our future work. Second, we adopt preset values for modeling the variability of VM performance. We note that the cost savings and performance metrics presented here can change depending on those values. We plan to conduct extensive measurements in a commercial cloud environment to confirm that our simulation results can be carried over to real systems.

V. RELATED STUDIES

Cloud computing is an merging platform for commercial and scientific applications, due to advantages in the pay-as-you-go business model and elasticity capacity provision. Various studies [6], [7], [9], [13], [19] present performance studies and report their experiences on migration of various applications to commercial cloud platforms. A common observation is the high variability in the quality of service. Kossmann et al. [9] present a comprehensive evaluation of database applications under different cloud architectures. They conclude that the cost and performance of the services vary significantly depending on the workload. Jackson et al. [6], [7] port various scientific applications, such as SNFactory pipeline, to the Amazon EC2 [4]. Their results show that the performance of EC2 is more variable and slower than non-cloud computing platforms, due to the limitation of interconnects on the EC2. Ueda and Nakatani evaluate a wiki workload and Apache daytrader using two open-source cloud platforms, OpenNebula [14] and Eucalyptus [13]. The two platforms give very different performance results, e.g., in terms of VM provisioning, response time, and throughput, compared to Amazon EC2 .

Because of the advantages of cloud computing, there is an emerging trend to migrate service-oriented applications from existing system to the cloud. The focus of related studies are widely spread: from a summary of the practical experiences [1], to frameworks for automating and easing the migration process [11], and cost optimization [5], [18]. Chauhan and Babar [1] report practical experiences of migrating an open source software framework, Hackystat, to the cloud. One of the key findings is that it is easier to migrate software systems consisting of stateless components to IaaS clouds.

Various service replication strategies are developed and evaluated for guaranteeing the reliability [3], [21] or performance under time-varying workloads [2], [10], [15]–[17]. To deliver highly dependable service systems, Zhen and Lyu [21] compare different combinations of replication strategies, using their proposed evaluation framework. Their objective is to select a suitable strategy such that the performance threshold and failure threshold are met. Dustdar and Juszczyk [3] developed a passive replication strategy on mobile ad-hoc networks, whose topologies vary over time, and validated it on a simulation prototype. As for workload driven replication strategy, both single-tier [2], [10] and multiple-tier [16], [17] web server systems in a non-cloud platform are well addressed. Petrucci et al. [15] implement a dynamic service provisioning policy to optimize power consumption on a heterogeneous cluster. While most provisioning studies monitor the request rate, Singh et al. [16] monitor not only the request rate but also the mix of applications.

Motivated by the emerging practice of migrating service systems to the cloud, we study the opportunistic replication strategy, which leverage the pay-as-you-go billing feature and high system variability in the cloud. Our study is based on several observations made in the prior art. In contrast to existing replication policies for private platforms, our proposed replication policy is driven by the workload dynamics as well as the performance reliability, and most importantly designed for the cloud platform. Moreover, we are able to not only handle elastic workload demands of different services, but also maintain a very satisfactory performance at a lower cost.

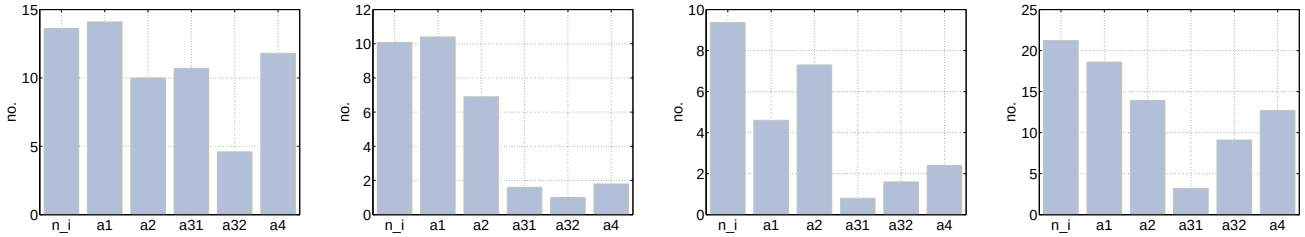
VI. CONCLUSION

In this paper, we propose an opportunistic replication policy, specially designed for services deployed in a cloud. The objective of our work is to leverage the variability in VM performance and pay-as-you-go billing contracts in the cloud, such that the number of VMs for each service is minimized and opportunistically provisioned with better performing VMs. Our policy is based on comprehensive workload and system

TABLE II
PERFORMANCE OF UNDER DIFFERENT POLICIES: FOUR SERVICES

Policy	Total	S1				S2				S3				S4			
	CS[%]	CS[%]	U[%]	RT	AS	CS[%]	U[%]	RT	AS	CS[%]	U[%]	RT	AS	CS[%]	U[%]	RT	AS
I	0.00	0.00	55.8	0.91	1.14	0.00	49.4	0.45	1.15	0.00	63.8	0.37	1.16	0.00	58.2	1.74	1.17
II(a)	12.4	7.95	74.6	1.09	1.15	20.2	74.9	0.56	1.15	7.18	75.0	0.44	1.16	13.3	76.0	1.98	1.17
II(p)	11.2	5.61	73.9	2.37	1.16	19.7	73.9	0.56	1.15	7.01	75.1	0.44	1.18	12.0	76.7	2.17	1.16
Brok.(a)	29.7	30.1	71.6	0.99	1.22	38.8	73.2	0.52	1.24	19.6	73.1	0.42	1.23	28.6	74.9	1.86	1.23
Brok.(p)	29.5	28.4	71.6	2.34	1.20	37.0	71.8	0.51	1.23	21.9	74.0	0.42	1.25	29.3	75.6	2.03	1.23

CS=normalized cost saving, U=utilization, RT=average response time of invocation, AT=average normalized throughput of VMs



(a) Service 1 (b) Service 2 (c) Service 3 (c) Service 4

Fig. 5. Average number of active VMs per window(n_i) and average number of VM used in each control action: turn off (a1), replace (a2), reconfigure off (a31), reconfigure on (a32), turn on (a4).

characteristics, i.e., time variability of workloads, VM variability, invocation variability, and billing periods. Moreover, we consider a complex set of control actions, i.e., turning on and off, replacing and reconfiguring VMs, with detailed modeling of the respective overhead. The proposed policy not only optimizes the cost of a single service but also the welfare of all services in a collaborative manner.

Our evaluation results using production traces show that the proposed policy achieves lower cost and better performance in terms of VM utilization and response time, compared to existing replication policies that are oblivious to performance and billing characteristics of the cloud.

Regarding ongoing research, we are exploring more complex service-oriented systems including composite services. Moreover, we are conducting further evaluations on commercial and research cloud platforms.

REFERENCES

[1] M. A. Chauhan and M. A. Babar. Migrating service-oriented system to cloud computing: An experience report. In *IEEE CLOUD*, pages 404–411, 2011.

[2] Y. Chen, A. Das, W. Qin, A. Sivasubramaniam, Q. Wang, and N. Gautam. Managing Server Energy and Operational Costs in Hosting Centers. In *Proceedings of ACM SIGMETRICS*, pages 303–314, 2005.

[3] S. Dustdar and L. Juszczyk. Dynamic replication and synchronization of web services for high availability in mobile ad-hoc networks. *Service Oriented Computing and Applications (SOCA)*, 1(1):19–33, 2007.

[4] A. EC2. http://www.amazon.com.

[5] C. Fehling, F. Leymann, and R. Mietzner. A framework for optimized distribution of tenants in cloud applications. In *IEEE CLOUD*, pages 252–259, 2010.

[6] K. R. Jackson, L. Ramakrishnan, K. Muriki, S. Canon, S. Cholia, J. Shalf, H. J. Wasserman, and N. J. Wright. Performance analysis of high performance computing applications on the amazon web services cloud. In *CloudCom*, pages 159–168, 2010.

[7] K. R. Jackson, L. Ramakrishnan, K. J. Runge, and R. C. Thomas. Seeking supernovae in the clouds: a performance study. In *HPDC*, pages 421–429, 2010.

[8] L. Kleinrock. *Queueing Systmes*. Wiley, 1976.

[9] D. Kossmann, T. Kraska, and S. Loesing. An evaluation of alternative architectures for transaction processing in the cloud. In *SIGMOD Conference*, pages 579–590, 2010.

[10] M. Lin, A. Wierman, L. L. H. Andrew, and E. Thereska. Dynamic Right-sizing for Power-proportional Data Centers. In *Proceedings of IEEE INFOCOM*, 2011.

[11] R. Meersman, T. S. Dillon, and P. Herrero, editors. *Cafe: A Generic Configurable Customizable Composite Cloud Application Framework*, volume 5870 of *Lecture Notes in Computer Science*. Springer, 2009.

[12] X. Meng, C. Isci, J. Kephart, L. Zhang, E. Bouillet, and D. Pendarakis. Efficient Resource Provisioning in Compute Clouds via VM Multiplexing. In *Proceedings of International Conference on Autonomic Computing*, ICAC, pages 11–20, 2010.

[13] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov. The eucalyptus open-source cloud-computing system. In *In Proceedings of 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 124–131, 2009.

[14] OpenNebula. http://opennebula.org.

[15] V. Petrucci, E. V. Carrera, O. Loques, J. C. B. Leite, and D. Mossé. Optimized management of power and performance for virtualized heterogeneous server clusters. In *CCGRID*, pages 23–32, 2011.

[16] R. Singh, U. Sharma, E. Cecchet, and P. Shenoy. Autonomic mix-aware provisioning for non-stationary data center workloads. In *Proceeding of the 7th international conference on Autonomic computing (ICAC)*, 2010.

[17] C. Stewart, T. Kelly, and A. Zhang. Exploiting nonstationarity for performance prediction. In *Proceedings of EuroSys*, pages 31–44, 2007.

[18] I. Trummer, F. Leymann, R. Mietzner, and W. Binder. Cost-optimal outsourcing of applications into the clouds. In *CloudCom*, pages 135–142, 2010.

[19] Y. Ueda and T. Nakatani. Performance variations of two open-source cloud platforms. In *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC)*, pages 1–10, 2010.

[20] A. Verma, U. Sharma, R. Jain, and K. Dasgupta. Compass: Cost of migration-aware placement in storage systems. In *Integrated Network Management*, pages 50–59, 2007.

[21] Z. Zheng and M. R. Lyu. A qos-aware fault tolerant middleware for dependable service composition. In *DSN*, pages 239–248, 2009.