Optimizing Multi-tenant Cloud Resource Pools via Allocation of Reusable Time Slots

Leonardo P. Tizzei, Marco A. S. Netto, and Shu Tao

IBM Research
{ltizzei, mstelmar}@br.ibm.com
shutao@us.ibm.com

Abstract. Typical pricing models for IaaS cloud providers are slotted, using hour and month as time units for metering and charging resource usage. Such models lead to financial loss as applications may release resources much earlier than the end of the last allocated time slot, leaving the cost paid for the rest of the time unit wasted. This problem can be minimized for multi-tenant environments by managing resources as pools. This scenario is particularly interesting for universities and companies with various departments and SaaS providers with multiple clients. In this paper we introduce a tool that creates and manages resource pools for multi-tenant environments. Its benefit is the reduction of resource waste by reusing already allocated resources available in the pool. We discuss the architecture of this tool and demonstrate its effectiveness, using a seven-month workload trace obtained from a real multi-tenant SaaS financial risk analysis application. From our experiments, such tool reduced resource costs per day by 13% on average in comparison to direct allocation of cloud provider resources.

Keywords: cloud computing, multi-tenancy, resource allocation, elasticity, SaaS, charging models, financial cost saving

1 Introduction

Infrastructure as a Service (IaaS) providers usually follow a pricing model where customers are charged based on their utilization of computing resources, storage, and data transfer [7]. Typical pricing models for IaaS providers are slotted, using hour and month as time units for metering and charging the resource usage. For several IaaS providers, the smallest billing unit is one hour [7], which means that if clients utilize cloud resources for only 1 hour and 30 minutes, for example, they have to pay for 2 hours and waste 30 minutes of the remaining time.

Some IaaS providers, such as Amazon, offer a service called *spot instances* [1], which allows clients to bid for instances provided by other clients in auctions. If the bid price is greater than the current spot price for the specified resource, the request is fulfilled. However, the utilization of these resources might be interrupted if the spot price rises above the bid price. Such interruption is not tolerable for many Software as a Service (SaaS) applications, especially for critical enterprise applications—such as those in financial sector.

For multi-tenant environments, resource waste can be minimized by managing resources as pools. In particular, large organizations can benefit from this resource pool, which would enable their subdivisions (*e.g.*, departments) to share these resources. Similarly, SaaS providers with multiple clients could benefit from such pools. A resource manager can allocate cloud resources released by a tenant for consumption by another tenant thus minimizing resource waste. Furthermore, such allocation avoids the need for provisioning new cloud resources and might accelerate the acquisition time since the allocation of cloud resources is generally faster than their provision.

In this paper we study how the reuse of allocation time slots in a multi-tenant environment can save IT costs. The contributions of the paper are:

- A motivation scenario for reusing resources among multiple tenants based on real data from tenants accessing a financial risk analysis application over a seven-month period (§ 2);
- Description of a tool for managing cloud resources as pools for multiple tenants, which can be executed on both simulation (single machine) and real (cloud environment) modes (§ 3);
- Evaluation of the tool using a seven-month period workload from multiple tenants (§ 4).

2 Motivation and Problem Description

SaaS applications rely on elasticity to offer a service to its clients without under-utilization of its cloud resources. Elasticity becomes essential in scenarios such as when a subset of workload tasks of a SaaS application is compute-intensive. Then, the SaaS application scales-out its infrastructure to run this subset of compute-intensive tasks. When their execution ends, the SaaS application scales-in the infrastructure. Thus, the provider of such application scales-out and scales-in its infrastructure, because under-provisioning of resources might cause Service Level Agreement (SLA) violations and over-provisioning of resources causes resource-waste [13] and, consequently, additional costs to SaaS providers.

We define resource-waste as the amount of time a cloud resource is not utilized by a SaaS application after its acquisition. For instance, two clients A and B submit workflows (i.e., in this case, a set of compute-intensive tasks that are executed in a predefined order without human intervention) to the SaaS application, which provisions one cloud resource to execute compute-intensive tasks. The execution of client A workflow tasks lasts 3 hours and 40 minutes and the execution of client B workflow tasks lasts 2 hours and 10 minutes. When both executions end, the SaaS application does not have anything else to do in remaining time for each cloud resource so it cancels both of them and wastes 70 (20 + 50) minutes. If several of these workflows run in parallel, resource-waste can be reduced by pooling together cloud resources.

We extracted data from a seven-month workload trace of a real cloud-based risk analysis application (further described in Section 4.1) to illustrate the problem of resource-waste. Figure 1 depicts when executions of compute-intensive

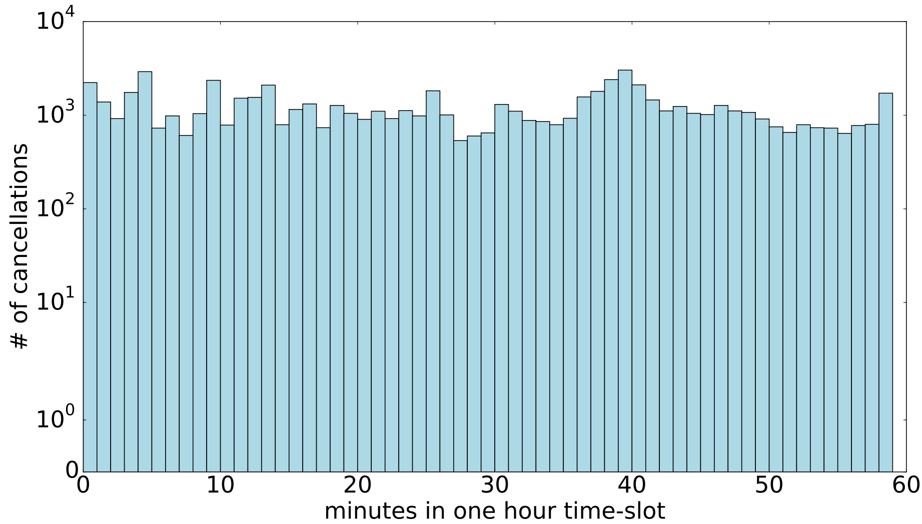

Fig. 1. Histogram of number of executions of compute-intensive tasks that end in each minute of one hour time-slot

tasks end within one hour time-slot. The closer to the zero minute the greater the resource-waste. For instance, a resource release at minute 10 of the last allocated time slot means that 50 minutes were wasted. This figure shows a significant number of cloud resources were under-utilized, which might increase operational costs for the SaaS provider.

SaaS applications would only be able to allocate cloud resources from one client to another if several clients submitted workflows constantly. Otherwise, the SaaS application would have to maintain cloud resources between two workflows that are far apart. Furthermore, clients have different profiles for the time they submit their workflows and for the duration of these workflows. The data from the SaaS application contains 32 clients and they submit workflows in different periods of the day and their workflows have different durations. Figure 2 illustrates the profile of four of these 32 clients. Each histogram shows the number of tasks of the workflows that are submitted by clients in each hour of a day. These charts show that, for this SaaS application, clients submit workflows in different periods of the day. Since there are several clients that have heterogeneous profile of utilization, this scenario is suitable for allocating cloud resources from one client to another.

Thus, the research question we address in this paper is: *Can a resource manager reduce resource-waste for multi-tenant cloud-based applications by reusing already allocated resources*?

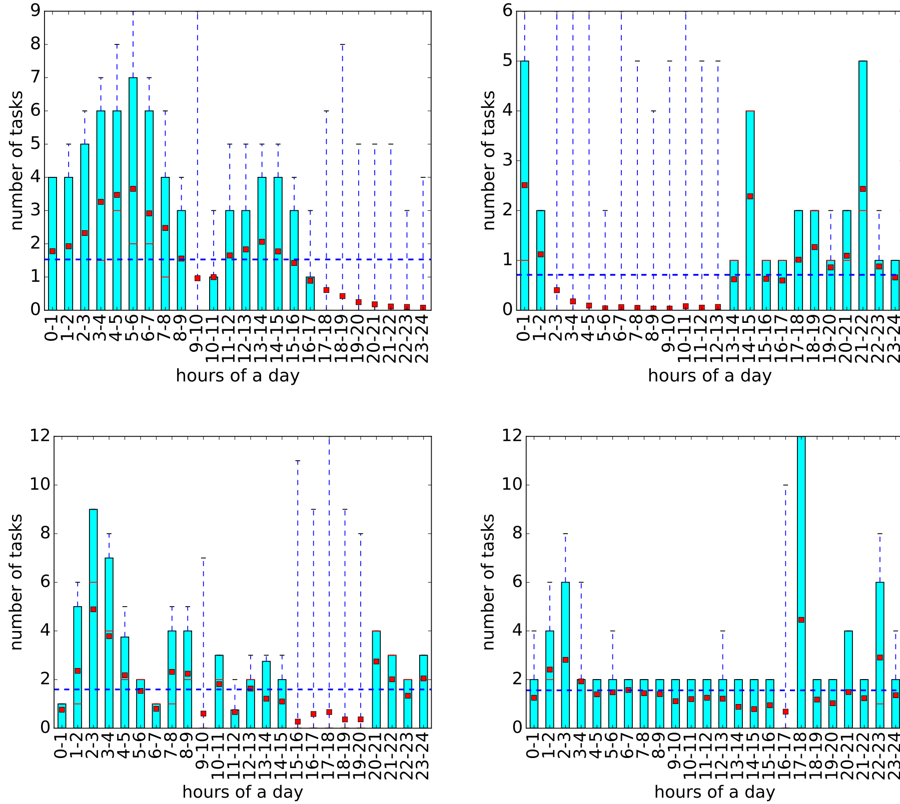

Fig. 2. Boxplots describing the number of tasks executed in each hour of a day over a period of seven months. Each plot shows the profile of one client.

3 PoolManager: a Cloud Resource Manager Tool

This section describes how we designed the software architecture and developed a tool, called PoolManager, to address the resource-waste problem mentioned in Section 2.

3.1 Overview

The PoolManager tool has two goals: (i) to reduce financial cost for SaaS applications by minimizing resource-waste; (ii) to support SaaS applications to meet SLA by minimizing acquisition time. These goals are overlapping because the minimization of resource-waste by allocating resources from one client to another might also reduce provision times of new resources. In order to achieve these goals, we designed PoolManager to lie between SaaS and IaaS providers to control the access to cloud resources. This control is necessary to create a pool

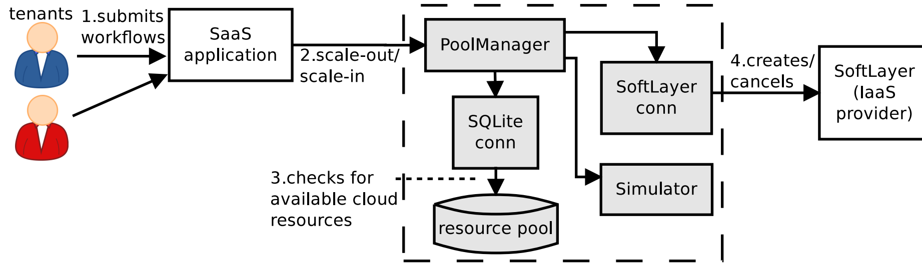

Fig. 3. Overview of the PoolManager architecture

of resources aiming to minimize resource-waste. Figure 3 provides an overview of our solution and defines the scope of our contributions inside the dashed square.

`Tenants` submit workflows to the `SaaS application` provider, which might need to scale-out its infrastructure to meet SLAs. Then, it requests the `PoolManager` for cloud resources, which checks the `resource pool` to decide whether it is necessary to create new resources. If so, `PoolManager` submits a request via a cloud connector (*e.g.*, `SoftLayer conn`)[1] for resources to the `IaaS provider`. Otherwise, it checks the policy for allocating resources, erases existing data of selected resources, and configures those according to the client. The number of resources available in the pool might be insufficient for the request, thus new resources might be created alongside to the allocation of existing ones.

After expanding its infrastructure, `SaaS application` provider will probably shrink it to reduce operational costs, which will trigger a similar flow of messages as described above. The `Simulator` plays the role of an "artificial" cloud connector and it was built for helping developers to explore data and create resource allocation policies, as described in Section 4.

3.2 Operations and Software Architecture

PoolManager has two main operations, where SaaS application providers: (i) request for cloud resources; and (ii) request to cancel cloud resources that were used to execute client workflows. Figure 4 describes these two operations using a UML activity diagram notation [15].

In the first operation (Figure 4a), PoolManager receives a request for creating cloud resources. Then, it checks if there are cloud resources available in the pool. If so, it allocates available resources to the SaaS application according to allocation policies (policies are further described in Section 3.3). It might be the case that the number of available resources is not sufficient to fulfill the request, so it submits a request for creating new cloud resources to the IaaS provider.

For the second operation (Figure 4b), PoolManager receives a request for canceling cloud resources that were allocated to a client. Then, the PoolManager checks if any policy applies, so it can later be allocated to another client.

[1] Other cloud connectors could be created to access resources from various other cloud providers, similar to the concept of broker in grid computing.

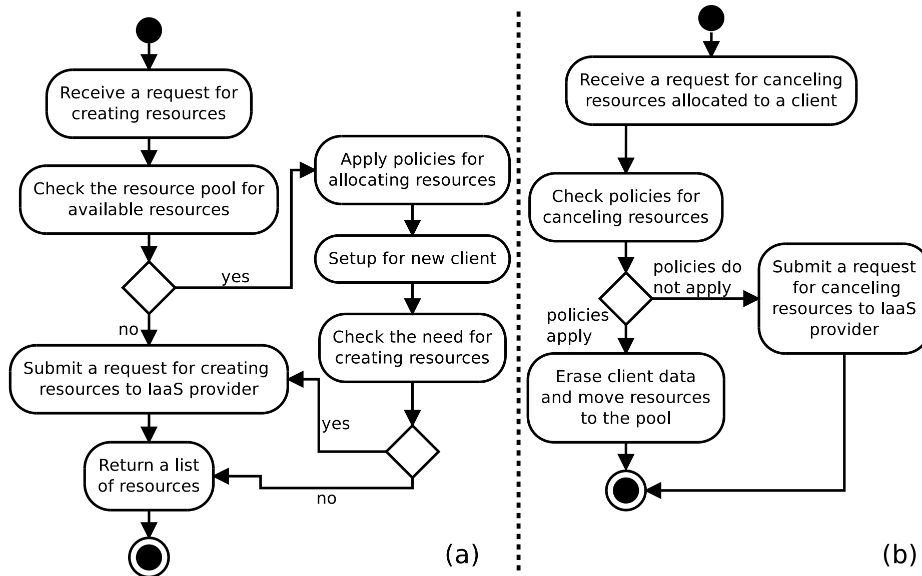

Fig. 4. Two main operations: (a) request for creating cloud resources and (b) request for canceling cloud resources

If policies apply, then all data of the previous client is erased. Otherwise, PoolManager submits a request to the IaaS provider to cancel cloud resources. In some situations, some resources are canceled, whereas others have their data erased and moved to the pool. Note that after receiving a request for canceling resources, they can be either canceled or maintained by PoolManager, but they cannot be allocated directly from one client to another.

The architecture of this solution is similar to the Proxy architectural pattern [5], because the PoolManager plays the role of a placeholder that aims to reduce provisioning time of cloud resources. In such manner, it mediates the access from SaaS applications to the IaaS provider to allocate resources released by one client to another. From our experiments, we observed reductions of 2-10 minutes of cloud provisioning time to a few seconds for cleaning up the data of an existing instance of a client to deliver it to another client. To perform such a process we rely on the cloud provider, in our case SoftLayer, to refresh a VM. However, additional scripts to delete data could be provided.

PoolManager was implemented using Python2.7, it has around 2.2 KLOC, and uses SQLite3 database and SoftLayer [3] IaaS provider, but other databases and IaaS providers could be used as well.

3.3 Time-based Allocation and Cancellation Policies

There are several types of resource allocation policies for cloud [9, 22, 24]. We implemented two policies considering time-slots: one is related to the request for

creating resources and the other is related to the request for canceling resources. Exploring the best optimization policies is out of the scope of this paper and we leave it as future work.

Both policies aim to save costs by reusing already allocated resources. In the first policy PoolManager allocates resources that are the closest, in terms of time, to be canceled. For example, if PoolManager receives a request for creating two cloud resources and it has five in the pool, the two cloud resources closest to be canceled will be allocated. We also investigated other allocation policies, including, for instance, offering resources that still have a long time to complete the hour. We observed that the most cost-effective solutions lie when offering resources near the initial of the hour or closer to be canceled. Near the initial hour has the benefit of reusing most of the already allocated hour, whereas close to the cancellation time avoids re-provisioning of machines.

The second policy aims to minimize resource waste. It defines that a resource should not be canceled if it was utilized for less than 50 minutes or more than 59 minutes within one hour time-slot. By "one hour time-slot" we mean the last of one hour time-slots of cloud resources that have been already paid. For instance, a cloud resource is created and it is utilized to execute the workflow which lasted 2 hours and 45 minutes. Within the last (*i.e.*, the third) of one hour time-slots, such cloud resource was utilized by 45 minutes. The rationale behind this policy is to maximize the chances of a client to reuse an existing instance. The 50th minute has been chosen empirically, which leaves 10 minutes before the next billing hour. Our experience has shown that such time is usually enough to cancel cloud resources even in the presence of faults (*e.g.*, temporary lack of network connection). Given that, we believe 50 minutes is a conservative number, though it can be easily changed in order to enhance the optimization of resources. Note that after the 59th minute the resource should not be canceled, because there is a risk it is already too late to cancel it before the next billing hour.

4 Case Study: Financial Risk Analysis in the Cloud

4.1 Goal and Target SaaS Application

The goal of this case study is to assess whether PoolManager can minimize resource-waste. We used Risk Analysis in the Cloud (RAC) as target SaaS application (the application name is fictitious due to the contract between RAC provider and its clients). RAC manages risk and portfolio construction for financial organizations, such as banks and trading desks. We were provided with a real workload that was computed by RAC application. This workload describes the workflow execution of 32 clients, which submitted 74,753 tasks—including 8,933 compute-intensive tasks—over a period of 214 days (seven months). Each compute-intensive task demands the creation of eight cloud resources in order to increase performance thus meeting SLA while non-compute-intensive tasks are executed on on-premise environment.

4.2 Planning and Operation

We defined two metrics: (i) *resource-waste per day*, which is the number of minutes that cloud resources were idle per day; (ii) *financial gain per day*, which is the subtraction of the amount paid to IaaS provider from the amount clients pay to SaaS provider per day.

First, we defined resource-waste per day metric as presented in Equation 1.

$$W = \sum_{k=1}^{T} 60 - (U_k \bmod 60) \tag{1}$$

where cloud resource-waste is W, utilization of cloud resource in minutes is U and T is the total number of times that any cloud resource was utilized by the SaaS provider. If the same cloud resource was utilized twice, *e.g.*, for executing the task of one client and afterwards of another, it is represented by two k values.

In order to assess how PoolManager influenced resource-waste in terms of time, we compared the PoolManager tool against the direct-access between SaaS and IaaS providers (hereafter called *direct-access* approach, for short). To collect resource-waste per day for the direct-access approach, we implemented a parser for the workload, which identified the beginning and end of compute-intensive tasks that demand the creation of cloud resources. Thus, in the beginning of each of these tasks, eight cloud resources would be created and when these tasks ended, resources would be canceled if we were running the real application. Since we are interested in measuring the resource-waste, the parser computed the utilization of cloud resources as the time between the beginning and the end of a compute-intensive task. Then, for each one of them, it calculated resource-waste per day according to Equation 1.

To collect resource-waste per day metric for the PoolManager approach, we replaced its connector to a real cloud provider by an artificial one, which simulates the creation and cancellation of cloud resources, because the creation of approximately 9,000 cloud resources for such workload was out of our budget. Furthermore, such simulator enabled us to run faster the workflow (further information about the simulator below). Then, we implemented another parser for the workload that identified compute-intensive tasks and generated a script that simulates clients submitting requests to RAC application. Finally, we executed this script and stored all information related to the creation and cancellation of cloud resources into a database, which was parsed to collect the metric defined by Equation 1.

The second metric assesses the amount of money that can be saved by pooling cloud resources. SaaS providers, such as RAC provider, must pay per usage of cloud resources that are offered by IaaS providers. Equation 2 represents how can one measure the costs of these cloud resources.

$$C_{IaaS} = \sum_{k=1}^{T} \lceil U_k / 60 \rceil \tag{2}$$

where C_{IaaS} is the cost of IaaS in units of money, U represents cloud resource utilization measured in minutes, T represents the number of times cloud resources were utilized. This equation is a simplified version of the real cost, because it does not consider the cost variation according to the type of cloud resource. Only the time aspect is being considered.

In order to measure the financial benefits of pooling cloud resources from the SaaS provider's perspective, it is also necessary to measure its income. However, we did not have access to the real contract between the RAC application and its clients. Thus, we defined that the clients of SaaS application follows the same pay-per-usage model between SaaS and IaaS providers—using time slots. That is, the amount of money each client pays to the SaaS provider is proportional to the duration of workflow execution. Equation 3 defines how it is measured:

$$C_{SaaS} = \sum_{c=1}^{N} \sum_{k=1}^{T} \lceil U_{ck}/60 \rceil \tag{3}$$

where C_{SaaS} is the cost of SaaS in units of money, U_{ck} measures cloud resource utilization k to execute the workflow of a client c. N is the total number of clients and T represents the number of times cloud resources were utilized by a client c. Based on Equations 2 and 3, we can define the *financial gain* metric as follows:

$$G = C_{SaaS} - C_{IaaS} \tag{4}$$

Discrete Event Simulator In order to replicate the execution of a seven-month workload in a feasible manner, we developed a discrete event simulator. When this simulator receives creation or cancellation requests, instead of actually creating or canceling cloud resources, it executes the allocation policies and stores decisions and resource related information into the database exactly the same way that would be stored if the real connector was executed. The idea is to log all request information so it can be later parsed to extract the metrics mentioned in Section 4.2. For instance, if a request for creating cloud resources demands a creation of a new cloud resource, then the simulator will 'accelerate' time, instead of waiting for the average provisioning time. Another example: if two requests are apart for more than one hour and there is no request between them, the simulator will cancel all cloud resources that are in the pool, because the PoolManager would not maintain a cloud resource for such a period of time.

4.3 Result Analysis

The results for the comparison between the *direct-access* approach against PoolManager are presented in Figure 5, which shows resource-waste (in minutes) over the days. Results show PoolManager minimized *resource-waste per day* significantly. Both curves in Figure 5 have an erratic behavior with peaks, which represent work days, and valleys, which represent weekends. Despite this similarity, the curve that represents the direct-access approach has more intense peaks.

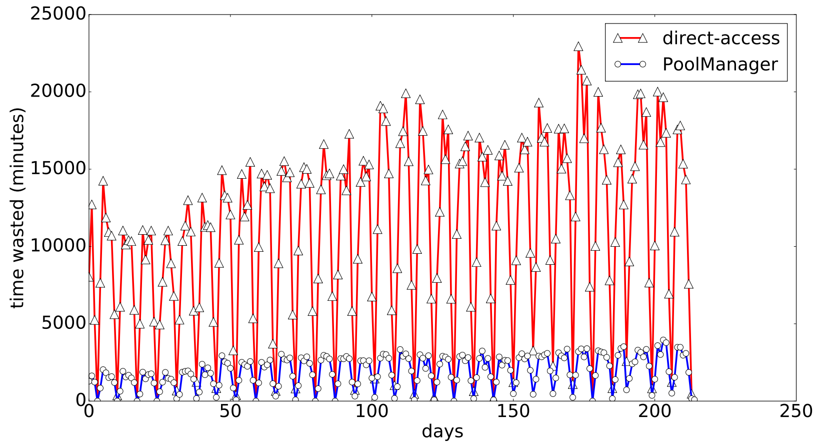

Fig. 5. Resource-waste per day for direct-access approach versus PoolManager

For all days, PoolManager reduced resource-waste in comparison to direct-access approach and such difference is even higher in the peaks of both curves. For instance, in the day 173, the direct-access approach wasted 22,424 minutes while PoolManager wasted 2,936 minutes. This is consequence of the reuse of the VMs among several clients.

Figure 6 illustrates why PoolManager minimized resource-waste as it presents a histogram of the number of times cloud resources were canceled in each minute of one hour time-slot. Note that PoolManager did not cancel any cloud resource before the 50th minute of one hour time-slot, according to time-based cancellation policy. The figure also shows a peak around 50 minutes which is a result of the cancellation policy. An administrator having a detailed understanding of the workload and client needs could be more aggressive and move the cancellations to 55 minutes or even further to obtain further cost reductions, but increasing the risk of delaying the cancellation process and having to allocate another entire hour—this trade-off needs to be carefully analyzed to configure the cancellation policy.

For the *financial gain per day* metric, the results show that PoolManager saved an average of 238 monetary units per day for the SaaS provider (see the blue dashed line in Figure 7). This means that the allocation of cloud resources—according to allocation policy—from one client to another has been effectively performed to minimize cloud resource provisioning. Furthermore, cancellation policy supported cost reduction, because resources that were utilized once to execute the workflow of a client were maintained and often re-utilized to execute the workflow of other clients. From our experiments, PoolManager reduced resource costs per day by 13% on average in comparison to direct allocation from cloud provider resources: Direct-access cost per day of 545.83 on average (+/-285.22) against PoolManager cost per day of 473.45 on average (+/-254.92).

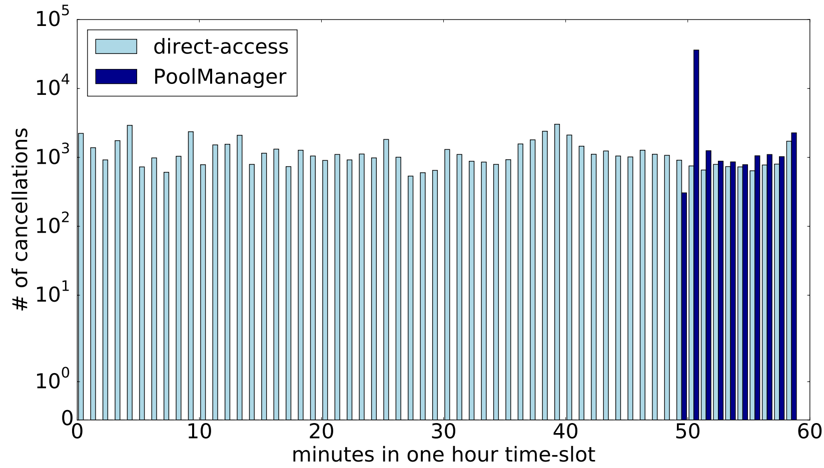

Fig. 6. Comparing resource utilization using direct-access approach and PoolManager

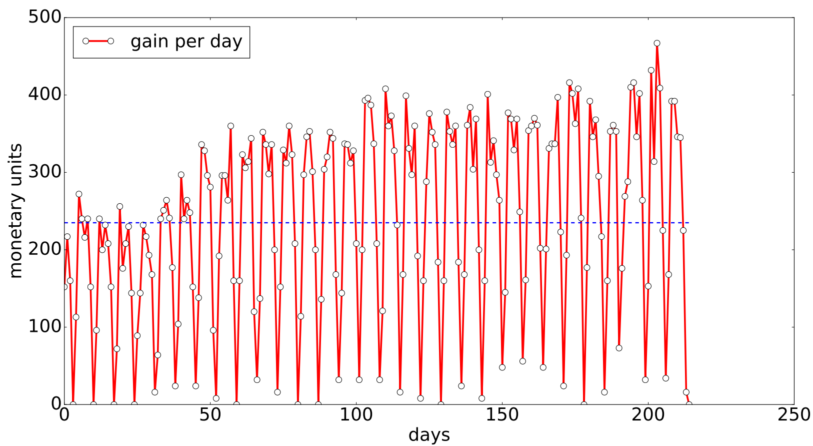

Fig. 7. Gain per day

Pooling resources minimized the need for provisioning cloud resources and, since allocation time is usually shorter than provisioning time, such approach can also contribute to reduce acquisition time thus supporting the SaaS provider to meet SLAs. PoolManager allocated around 49% of all cloud resources that were utilized. If our assumption that allocation time is shorter than provisioning time is correct (which we observed in our experiments: 2-10 minutes of provisioning versus seconds for cleaning up VMs), such results can significantly reduce acquisition time.

Therefore, given the aforementioned research question, the results provided evidence that PoolManager succeeds in minimizing resource-waste.

4.4 Threats to Validity

Threats to internal validity are influences that can affect the independent variable, which in this study is the PoolManager tool, with respect to causality, without the researcher's knowledge [23]. One threat to internal validity that we identified is related to the instrumentation of this case study. We implemented two parsers to collect data and they might have bugs, which can cause erroneous results. We minimized this threat by testing both parsers and by manually analyzing parts of their execution.

Threats to external validity are conditions that limit our ability to generalize the results of our experiment to industrial practice [23]. We identified two threats to external validity: (i) RAC application might not be representative of SaaS applications and (ii) the workload sample that was analyzed in this study might not be representative of a typical SaaS application workload. Regarding the first threat to external validity, we cannot totally avoid it, but our target application is a real industry application, which was *not* developed by any of the authors. Regarding the second threat to external validity, the workload sample corresponds to seven-month of execution logs, which were generated by submissions of several clients. Based on our knowledge of such application and on conversations we had with its experts, we believe that such period is long enough to be representative.

5 Related Work

Resource sharing is a concept that became popular in grid computing [4,6,11,12,14], where participants of autonomous organizations shared computing resources among themselves. Usually this resource sharing did not involve any monetary costs perceived by end-users.

With cloud, monetary costs are perceived by end-users and therefore they are careful on how they allocate resources [22]. For instance, Shen et al. [21] introduced a system called CloudScale to manage elasticity of resources for multi-tenant cloud environments. Their work focused on resource and energy savings and explored resource demand predictions and accuracy of those predictions. Gong et al. [13] also used predictions to scale resource capacity according to applications' demands. Other resource efforts [17, 24] considered dynamic resource allocation using auctions. In addition projects have been exploring resource sharing in clouds [8, 20] via the use of virtual currency.

Nathani *et al.* [19] proposed a scheduling algorithm for allocating virtual machines on cloud environments. The multi-tenancy aspect is explored by reorganizing allocation requests according to user deadlines. Lin *et al.* [18] focused their studies on resource allocation for multiple users considering the application level instead of trying to map physical to virtual resources.

León and Navarro [16] introduced an incentive mechanism for users to report their actual resource requirements inorder to save energy costs and meet QoS demands on multi-tenant clusters. Espadas *et al.* [10] proposed a resource

allocation model for SaaS applications considering multi-tenancy with the aim to avoid under and over resource utilization.

The main difference between these research efforts and ours is that we focused on services that need to manage resources for multiple tenants considering the hourly slotted-time charging model and exclusive use of resources for each tenant. We also show the benefits of managing resource pools instead of having individuals renting resources directly to IaaS providers.

6 Conclusions

Multi-tenant SaaS applications need to meet SLA and provide low prices to be competitive. These applications might benefit from pooling cloud resources together because it enables the allocation of cloud resources from one client to another, thus minimizing cloud provisioning. We developed a tool called PoolManager, which manages a pool of cloud resources through configurable policies, aiming to minimize resource waste. We evaluated PoolManager in a case study which involved a real risk analysis application. Based on the seven-month workload of this application, we compared the benefits of using PoolManager against its absence. The main lessons of this case study are the following: (i) it is important to understand how resources are consumed using historical data in order to understand how much resource-time is being wasted before designing any pooling optimization policy; and (ii) clients have different resource demands and arranging the resources as pools can bring great benefits to the entire group of clients and to the SaaS provider managing the resources for these clients.

As cloud providers reduce the billing unit (e.g., Google's GCE by-the-minute billing [2]), resource waste decreases and thus the benefit of using PoolManager is minimized. Even though, PoolManager's policies can be easily adapted to such billing unit aiming to increase reuse of cloud resources in order to minimize the number of fails during their provisioning.

As future work, we will explore how the configuration of existing and new policies affect the results regarding resource waste. In particular, how to use resource demand predictions to determine, in a proactive way, the size of the resource pool.

Acknowledgment

We would like to thank Xin Hu and Miguel Artacho from IBM Analytics team for their valuable help with the application used in this paper. We would like to thank Anshul Gandhi's contribution in initial analysis on the workload, David Wu, Alexei Karve, Chuck Schulz for discussions and environment setup, and the anonymous reviewers for their comments on this paper. This work has been supported and partially funded by FINEP / MCTI, under subcontract no. 03.14.0062.00.

References

1. Amazon elastic compute cloud: How spot instances work. http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/how-spot-instances-work.html, accessed: Ago/2015
2. Google cloud platform. https://cloud.google.com/compute/#pricing, accessed: Ago/2015
3. IBM SoftLayer. www.softlayer.com, accessed: Ago/2015
4. Andrade, N., Cirne, W., Brasileiro, F., Roisenberg, P.: Ourgrid: An approach to easily assemble grids with equitable resource sharing. In: Proceedings of the International Workshop on Job Scheduling Strategies for Parallel Processing. pp. 61–86. Springer (2003)
5. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M., Sommerlad, P., Stal, M.: Pattern-oriented software architecture, volume 1: A system of patterns. John Wiley and Sons (1996)
6. Buyya, R., Abramson, D., Giddy, J., Stockinger, H.: Economic models for resource management and scheduling in grid computing. Concurrency and computation: practice and experience 14(13-15), 1507–1542 (2002)
7. Buyya, R., Broberg, J., Goscinski, A.M.: Cloud computing: Principles and paradigms, vol. 87. John Wiley & Sons (2010)
8. Chard, K., Bubendorfer, K., Caton, S., Rana, O.F.: Social cloud computing: A vision for socially motivated resource sharing. IEEE Transactions on Services Computing 5(4), 551–563 (2012)
9. Endo, P.T., de Almeida Palhares, A.V., Pereira, N.N., Goncalves, G.E., Sadok, D., Kelner, J., Melander, B., Mangs, J.E.: Resource allocation for distributed cloud: concepts and research challenges. Network, IEEE 25(4), 42–46 (2011)
10. Espadas, J., Molina, A., Jiménez, G., Molina, M., Ramírez, R., Concha, D.: A tenant-based resource allocation model for scaling software-as-a-service applications over cloud computing infrastructures. Future Generation Computer Systems 29(1), 273–286 (2013)
11. Foster, I., Kesselman, C.: The Grid 2: Blueprint for a new computing infrastructure. Elsevier (2003)
12. Frey, J., Tannenbaum, T., Livny, M., Foster, I., Tuecke, S.: Condor-g: A computation management agent for multi-institutional grids. Cluster Computing 5(3), 237–246 (2002)
13. Gong, Z., Gu, X., Wilkes, J.: Press: Predictive elastic resource scaling for cloud systems. In: Proceedings of the International Conference on Network and Service Management. IEEE (2010)
14. Grimshaw, A., Ferrari, A., Knabe, F., Humphrey, M.: Wide area computing: resource sharing on a large scale. Computer 32(5), 29–37 (1999)
15. Larman, C.: Applying UML and patterns: an introduction to object-oriented analysis and design and iterative development, 3/e. Pearson Education India (2005)
16. León, X., Navarro, L.: Incentives for dynamic and energy-aware capacity allocation for multi-tenant clusters. In: Proceedings of International Conference on the Economics of Grids, Clouds, Systems, and Services, pp. 106–121. Springer (2013)
17. Lin, W.Y., Lin, G.Y., Wei, H.Y.: Dynamic auction mechanism for cloud resource allocation. In: Proceedings of the International Conference on Cluster, Cloud and Grid Computing. IEEE (2010)
18. Lin, W., Wang, J.Z., Liang, C., Qi, D.: A threshold-based dynamic resource allocation scheme for cloud computing. Procedia Engineering 23, 695–703 (2011)

19. Nathani, A., Chaudhary, S., Somani, G.: Policy based resource allocation in IaaS cloud. Future Generation Computer Systems 28(1), 94–103 (2012)
20. Punceva, M., Rodero, I., Parashar, M., Rana, O., Petri, I.: Incentivising resource sharing in social clouds. Concurrency and Computation: Practice and Experience 27(6), 1483–1497 (2015)
21. Shen, Z., Subbiah, S., Gu, X., Wilkes, J.: Cloudscale: elastic resource scaling for multi-tenant cloud systems. In: Proceedings of the Symposium on Cloud Computing. p. 5. ACM (2011)
22. Vinothina, V.V., Sridaran, R., Ganapathi, P.: A survey on resource allocation strategies in cloud computing. International Journal of Advanced Computer Science and Applications 3(6) (2012)
23. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: Experimentation in software engineering. Springer Science & Business Media (2012)
24. Zhang, Q., Zhu, Q., Boutaba, R.: Dynamic resource allocation for spot markets in cloud computing environments. In: Proceedings of the International Conference on Utility and Cloud Computing. IEEE (2011)