A Virtual Machine Re-packing Approach to the Horizontal vs. Vertical Elasticity Trade-off for Cloud Autoscaling

Mina Sedaghat
Dept. of Computing Science
Umeå University, Sweden
mina@cs.umu.se

Francisco Hernandez-Rodriguez
Dept. of Computing Science
Umeå University, Sweden
francisco@cs.umu.se

Erik Elmroth
Dept. of Computing Science
Umeå University, Sweden
elmroth@cs.umu.se

ABSTRACT

An automated solution to horizontal vs. vertical elasticity problem is central to make cloud autoscalers truly autonomous. Today's cloud autoscalers are typically varying the capacity allocated by increasing and decreasing the number of virtual machines (VMs) of a predefined size (horizontal elasticity), not taking into account that as load varies it may be advantageous not only to vary the number but also the size of VMs (vertical elasticity). We analyze the price/performance effects achieved by different strategies for selecting VM-sizes for handling increasing load and we propose a cost-benefit based approach to determine when to (partly) replace a current set of VMs with a different set. We evaluate our repacking approach in combination with different auto-scaling strategies. Our results show a range of 7% up to 60% cost saving in total resource utilization cost of our sample applications and workloads.

Categories and Subject Descriptors

C.2.4 [**Distributed Systems**]: Cloud Computing
C.4 [**Performance of Systems**]: Performance attributes

General Terms

Algorithms, Measurement, Performance

Keywords

Cloud computing, Autoscaling, Autonomous computing, Vertical elasticity, Horizontal elasticity

1. INTRODUCTION

Autonomous management systems are key to the realization of future elastic cloud infrastructures, both due to the scale and complexity of the infrastructures and due to the fact that management actions often may need to be performed with only seconds or minutes notice. Key to such an autonomous management system is the ability to automatically scale the amount of resources allocated to an application, depending on its load, in order to appropriately handle a hosted application's load peaks without over-provisioning when load is low. The scaling can be done either by changing the number of VMs, *horizontal elasticity*, or by changing the size of VMs, *vertical elasticity*. Recently, most attention has been given to horizontal elasticity management, partly due to the fact that vertical elasticity is much more limited as it cannot scale outside single physical machines.

However, as the horizontal elasticity decisions also have to consider what size VMs to initiate or terminate, the problems are not fully detached. For a certain amount of capacity there is some set of VMs (of possibly varying but normally predefined sizes) that most cost-efficiently provides that capacity. Even if during a sequence of scale-up operations the autoscaler determines the most cost-efficient way to provide the extra capacity needed at each step, the sequence of such operations may result in a far from optimal VM set for the aggregate capacity. Hence, at some point in time the set of VMs allocated would benefit from being replaced by a new, more optimized set of VMs providing the same capacity, here called *repacking* of VMs. Alternatively, each autoscaling decision could take into account not only adding VMs but also replacing some VMs with VMs of other sizes. As frequent turning on and off VMs is costly, this latter approach is not suitable in scenarios where frequent scaling operations are performed.

In this contribution, we as part of a holistic cloud management system [21], investigate the problem of when and how a set of allocated VMs should be repacked to a new optimal set of VMs and propose a model to provide the required capacity more cost-efficiently. The solution takes into account the price and capacity of a number of predefined VMs, where capacity most commonly is based on the specific application's performance on a VM of that type. Of course, price-changes may occur over time and that by itself is a reason for re-allocation. Moreover, we analyze the different parameters that affect the repacking decisions, including *price/performance* ratio of an application, cost of reconfigurations, behavior of different autoscalers, and the relation of repacking with expected life-span of the reconfigured VM set before scaling down or before the next reconfiguration.

2. BACKGROUND AND MOTIVATION

Applications with capacity demands with significant variations over time benefit the most when deployed in cloud environments, since they can benefit from the elasticity provided by clouds. In this contribution we assume that elasticity is handled through autoscaling.

Public cloud providers offer a variety of instance types, each characterized by different sizes and different prices. For example, Amazon EC2 [1] currently offers 17 instance types that range from standard instances to application specific instances offering high I/O, high CPU, or high memory. For private clouds we also assume that VMs of different sizes, prices, and price/performance ratios can be provisioned.

Different autoscaling mechanisms use different techniques to estimate and predict the capacity required to serve an application with a changing demand [22, 20, 3, 14, 15]. However, current autoscalers decide on when and how to change the capacity for a changing demand, but normally not evaluate which instance-type to choose. For example, autoscalers like those offered by Amazon or Rightscale [2] only consider the number of VMs to provision. In these cases, the user has to specify what VM-type to add or remove when a condition given by a threshold is met [14]. Thresholds and rules that act upon them must be defined by a user with a good understanding of both workload and application. With more information available, more advanced autoscalers can make improved decisions such as finding a cost efficient resource set [9, 7]. However, even for advanced autoscalers, when demand changes prior optimal combination of VMs becomes sub-optimal.

Notably, the cost-efficiency for different VM-types depends on pricing and application performance. In case there are preferences, e.g., on keeping the total number of VMs low, we can assume, without loss of generality, that this is included in an application's price/performance ratio. In case the best price/performance ratio is provided by the smallest VM-type, taking into account that this also leads to the large number of VMs, the VM-type selection problem can in practice be avoided by always selecting the smallest VM type and never make any re-configurations. Of more interest, from a re-configuration point of view, is when the price/performance ratio is more advantageous for large VM-types. In particular, this effect can be large when there is a penalty associated with having large number of VMs. For such scenarios, there is an inherent conflict in avoiding costly over-provisioning by allocating too much capacity and by allocating more costly small VMs.

In this contribution we consider different strategies for balancing this trade-off through selection of VM-type to add when more capacity is needed and with reconfiguration of the total VM-set as it has become less cost-efficient after a series of small capacity increases. It should be remarked that this contribution makes no assumptions on whether the autoscaler used is reactive[2, 1] or pro-active [3, 9]. It only considers how to most cost-efficiently provide the requested capacity, regardless of how that request has been determined.

3. REPACKING APPROACH

We formulate the problem of determining how to reconfigure the VM set as follows:
There is always a cost per time-unit associated with resource usage, regardless if the hosting is done on external resources and the cost is simply the explicit given price or if the price is a combination of investment and running cost on local machines in a private cloud. For a given VM-type, there is also an associated capacity, which may, preferably, be expressed in application specific terms. We assume, without loss of generality, that there are n VM-types VM$_i$, $i = 1, \ldots, n$ with an associated price p_i per time unit, during which they are able to perform work represented by the capacity c_i and the application requires at least the capacity C_{Req}. Assume that there is currently a set S of n VMs that provides the aggregated capacity $C \geq C_{Req}$ at the total price $P = \sum_{i=1}^{n} p_i$. We are interested to find a new set $\hat{S}$ with $\hat{n}$ VMs, that provides a capacity $\hat{C} \geq C_{Req}$ at a minimum price $\hat{P}$, and $\hat{P} < P$, while considering an overhead cost of reconfiguration R. The reconfiguration cost is to compensate the impact of reconfiguration on the application performance and may, if so required, include other costs than only the pure resource cost. Finally, given an expected stability duration d before the capacity requires another change, we want to determine if $\hat{P}$ is sufficiently much smaller than P to motivate the cost for the reconfiguration. Hence, the problem in its most basic form is:

$$1 - [(\hat{P} \times d + R)/(P \times d)] > K \quad (1)$$

where K is gain factor that defines the minimum benefit that a human manager defines as the threshold to perform reconfiguration. The intention of considering K is to avoid unnecessary costly changes for a marginal gain.

In this section we introduce a model for determining when to perform *Repacking* (or re-configuration) of the VM-set used to provide a certain capacity. Repacking of an application with a changing workload is a process that should be performed repeatedly and it is not a one time decision, so the following steps are executed repeatedly.

3.1 Finding the optimal Set

The goal of repacking is to find a configuration of resources that minimizes the resource utilization cost, given current configuration, a number of different VM-types with different price/performance characteristics, a cost-function for performing repacking, and an expected time (duration) for which a new configuration is expected to be used.

The first step is to find an optimal configuration defined in terms of number of VMs of each type that are required to serve the application with the desired QoS and with the minimum cost. In other words, we need to find how many instances are required and of what type they should be to serve the load with a minimal resource provisioning cost. The problem is formulated as a combinatorial optimization problem:

$$Minimize \quad Price = \sum_{i=1}^{n} n_i \times p_i$$

Subject to:

$$\sum_{i=1}^{n} n_i \times c_i \geq C_{Req}$$

$$n_i \in N$$

where p_i and c_i are respectively the price and the capacity of VM type i; n is the number of VM types; C_{Req} is the capacity required to provide the desired performance, and n_i is the number allocated instances of VM type i. The capacity of a VM, c_i, is presented as the average capacity that each VM can serve for the particular hosted application and it is calculated as the amount of work that saturates the instance. We assume that each instance type can serve a portion of load defined in terms of number of requests and proportional to its capacity (i.e., memory and CPU capacity). In case there is a cost associated with running many small VM-types, that is assumed to be included in the capacity figure. The price of a VM, p_i, is defined as all possible costs of maintaining a VM from a customer perspective, regardless if this is specified costs from a public provider or costs somehow determined for using a private infrastructure. In order to find the optimal set, we formulate the problem as an integer programming problem.

3.2 Deciding on transitioning policy

The knowledge about an optimal set of VMs for a certain capacity only gives a set of VMs that minimizes the cost, without acknowledging the overhead costs of altering the configuration. Therefore, we need to calculate the costs of performing this reconfiguration, in order to decide whether or not it is beneficial to enforce the reconfiguration despite the overhead costs. To calculate the reconfiguration costs, we first specify how to move from the current set to the optimized set. The transition policy defines a plan for changing the resource set from its current setup S to its new target configuration $\hat{S}$. Depending on application type and cloud platform, the transition can be done in several ways, e.g., by shutting down the VM, copying the state to the new VM, and starting the new VM; or by keeping both VMs running while new VMs boot and become ready to serve; or a combination of both by shutting down some VMs and keeping some other running. Transitions also can be carried out as a single action or they can be done through a series of consecutive actions, e.g., first start a VM and then shut down extra VMs one by one. For either alternative, there is an associated cost, called reconfiguration cost.

In our experiments, the transition policy is based on the notion of always maintaining the required capacity by keeping all instances running, to restrict the performance loss during transition. In order to do that, we start all new instances while maintaining the previous configuration running. Only after the new instances are launched and ready to serve, we shut down the extra instances from the previous configuration.

3.3 Calculating the reconfiguration costs

Reconfigurations, no matter the transition policy chosen, require changes in the resource set and a number of VM startups and shutdowns. Although cloud VM instances are expected to be very rapidly provisioned, the total time before they are ready to serve the application may be substantial. In our case, this time is the reconfiguration time. The minimum reconfiguration time is the time it takes for booting the VM and registering it to the load balancer until it becomes available. This time can be much longer when we add the time for application specific configurations, copying the data files and startups. For longer reconfigurations, the cost of preserving the performance increases, e.g., if a reconfiguration action requires copying a large database to a new VM, both VMs need to be up while the database is being copied, leading to extra payments for having two VMs in use. During the reconfiguration time a performance impact can be expected and this impact is associated with a cost. This cost can be defined in terms of performance degradations, SLA violations or simply the additional costs required to preserve the QoS during transition time. We calculate these reconfiguration costs as follows:

$$R = C_{extra} \times repacking\ time$$

where C_{extra} is the overhead cost associated with the extra allocation needed to preserve the performance during the repacking time.

3.4 Decision making

Deciding when to reconfigure is based on a cost-benefit analysis that depends on the following factors:

1. The difference between the current configuration's cost and the optimal configuration's cost.

2. The cost of reconfiguration.

3. The time period that we expect the new configuration to last.

Each of these factors is part of a decision function that weights the potential benefits of reconfiguration against the reconfiguration cost and the decision evaluates to true if the condition in Equation (1) is met.
Here $\hat{P}$ and P are respectively the runtime costs of the new and current configuration, d is the expected duration, or stability interval, which is the time we expect the new configuration be durable. If this duration is long, even expensive reconfigurations are worth performing. R is the reconfiguration cost and K is the gain factor which defines the minimum benefit expected for performing the reconfiguration. K represents hidden costs different from explicit reconfiguration costs, e.g., additional cost due to software licenses, or additional human resources for maintenance. These costs are not easily measured, but still need to be considered. We address these costs by assuming that a human manager configures the controller based on knowledge about such costs, gained from experience.
As it can be seen in (1), for large values of d, $\hat{P} \times d$ becomes large enough, that R can be neglected. This means that if we expect the new optimized configuration to last long enough, the reconfiguration costs can be neglected and the repacking controller can proceed with reconfiguration of the resource set. Although for large values of d, the maximum cost saved is bounded by $1 - (\hat{P}/P)$.

4. AUTOSCALING STRATEGIES

Autoscalers adapt the resource set to fast and transient changes in load. Different autoscalers have different VM selection policies and decide based on different inputs. Hence they propose different resource sets, which affect the decisions and costs of repacking. The autoscaler calculates the required capacity either by taking *total load* or $\Delta load = total\ load - current\ capacity$ as input.

Additionally, the policy for selecting VM types to provide the capacity can also be done at least in two sensible ways. It can either select the *optimal set of VMs*, or select a specific VM type preferred by the application or specific to a workload behavior, we call this prefered VM a *base instance*.

Repacking can be applied on autoscalers using $\Delta load$ as input. So using a repacking approach, two new strategies

Algorithm 1 Repacking Algorithm

Input: Required Capacity C at time t; Current Resource Set (S), with associated cost P.
Output: Decision on Repacking

```
 1: Find new optimal Resource Set, $\hat{S}$.
 2: Calculate cost ($\hat{P}$)
 3: if $\hat{P} < P$ then
 4:     Select a Transition Policy
 5:     Calculate Reconfiguration Cost:
        $R = C_{extra} \times repacking\ time$.
 6: end if
 7: if $1 - [(\hat{P} \times d + R)/(P \times d)] > k$ then
 8:     Reconfigure to $\hat{S}$
 9:     return;
10: else
11:     Repacking is not feasible
12:     return;
13: end if
```

are introduced, $S_{delta\_Repacked}$ and $S_{base\_Repacked}$. This results in 5 different autoscaling strategies in total, that are presented as follows:

1. S_{delta}:
S_{delta} calculates an optimal resource set for providing capacity corresponding to $\Delta load$. Some applications are performance sensitive and may require quick adaptations to changes and their priority is to maintain performance even by paying higher rental costs. S_{delta} is an autoscaling strategy suitable for this kind of applications. It does not perform reconfigurations, since it only adds (or removes) the VMs to the previous allocation. However, if the workload increases in small steps, S_{delta} allocates one more small VM for each time the load changes. The output of S_{delta} over time is therefore typically a resource set comprises of a set of small instances. Although each addition is optimal with respect to $\Delta load$, it is still a sub-optimal set with respect to the *total load*.

2. S_{base}:
The S_{base} strategy similarly takes $\Delta load$ as input for scaling the capacity. However, contrary to the first strategy, S_{base} does not adjust to the exact optimal extra capacity, i.e., a small VM for a small change, but it adds a VM of a predefined size, *base instance*, that a priori, has been judged to be a good size for that specific workload characteristics, such as volume and its oscillations in a specific time span. The *base instance* does not necessarily fit the capacity perfectly, as it may overprovision a little to avoid the frequent additions and removals that lead to a set of small expensive VMs in S_{delta}. So *base instance* can be selected by considering a tradeoff between overprovisioning cost and the cost due to cost inefficiency of smaller VMs. Another reason for using S_{base} is that a VM type may be more suitable for a certain application, e.g., a compute intensive application benefits from a high CPU VM type. The resource set shaped by this strategy is homogeneous consisting of a set of *base instances* for the duration that workload volume and its changes are within a specific range.

3. S_{Full}:
The previous two strategies take $\Delta load$ as input and find the appropriate extra VMs to satisfy $\Delta load$. In both cases the previous resource set remains unchanged and the new VMs are added to keep up with the changes in load. S_{Full}, on the contrary, uses *total load* to find the optimal set to serve the application demand. By considering *total load*, S_{Full} not only adds extra capacity but it may also replace VMs with more cost efficient ones, during any scaling decision. In this way, S_{Full} maintains the resource set optimal during application runtime. However, replacements may require frequent shutdowns and start ups that are costly and failure prone. The cost of running the application with this strategy is the sum of the resource set cost plus the cost of reconfigurations at each timestep.

The drawback of S_{Full} is that it blindly performs the reconfiguration as long as the new resource set is not optimal, without taking reconfiguration cost into account. We include S_{Full} as a comparison to show how doing a cost benefit analysis optimizes the number of reconfigurations as they are considered to be costly. We investigate the impact of smarter reconfigurations on the total cost against blind reconfigurations by comparing our result against S_{Full}.

4. $S_{base\_Repacked}$ and $S_{delta\_Repacked}$:
Repacking is applicable on autoscalers with $\Delta load$ as input. Hence, we applied our repacking model on the resource sets suggested by S_{delta} and S_{base} that leads to two new scaling strategies called $S_{delta\_Repacked}$ and $S_{base\_Repacked}$ repacking. The repacking model uses each of the resource sets proposed by S_{delta} and S_{base} to investigate which reconfigurations are required to reshape the set to optimal and to discover the cost of these reconfigurations. This information is used to perform a cost benefit analysis to evaluate the worthiness of the repacking. If the evaluation is positive, the previous resource set is replaced by the new optimal one. The reconfiguration actions for S_{delta} are mostly exchanging smaller instances with larger ones, whereas for S_{base} the actions involve replacing the set with an optimal one formed of different types of instances, instead of the original set of a single VM type.

Table 1 summarizes the autoscaling strategies employed, their inputs and specifications.

Scaling strategy	Input	VM selection policy	Repacking
S_{delta}	$\Delta load$	optimal set	Never
S_{base}	$\Delta load$	*base instance*	Never
S_{Full}	*total load*	optimal set	always
$S_{delta\_Repacked}$	*total load*	optimal set	when beneficial
$S_{base\_Repacked}$	*total load*	optimal set	when beneficial

Table 1: Autoscaling strategies.

5. REPACKING AND WORKLOADS

The need and effects of repacking of VMs are different for different workload patterns. Mao and Humphrey [15] characterize four types of workloads for cloud environments as *Stable*, *Increasing (Growing)*, *Seasonal (Cycle/Bursting)* and *On-and-Off*. Each of these patterns represents a specific application or a scenario.
Stable and **On-and-Off workloads**: The behavior of these workloads are similar to batch workloads. They have a definite lifetime and resource requirements, e.g., many repetitive

transactions to a database, with some heavy computational work for each. They are also active for a short period, *on-and-of*, or a long period of time, *Stable*, but their load is constant and not changing overtime.

Increasing workloads: Workloads of this category are often seen in companies that have high growth trends. Social networking services such as Facebook, Twitter, or file sharing applications such as Dropbox are good samples of *increasing* workloads. The growth trend for these websites shows a common pattern, as they all start with slow growth followed by small oscillations that become larger as service become popular.

Seasonal (Cycle/Bursting) workloads: The increase in load in seasonal workloads occurs in patterns that are repeating themselves, e.g., daily, weekly or monthly periods. These increases can be based on predictable events (in time and possibly in magnitude).

Neither of the *Stable* and *On-and-Off* workloads is of particular relevance to repacking of VM sets as the optimal set can be chosen from start and remain optimal since there are no load variations during the execution. Therefore, we focus on the *increasing* and *seasonal* workloads as they have load variations where series of autoscaling actions over time may lead to suboptimal VM set.

6. PERFORMANCE EVALUATION

6.1 Experiment setup

We evaluate our repacking approach by simulating a cloud with 6 types of VMs. In order to study the effect of repacking for different price/performance ratios between VM-types we consider two application types, A and B. Each application benefit differently from the different types of VMs, e.g., by replacing 2 small VMs with a medium VM, application A can serve 1.2 times more request for the same price, while this number for application B is 1.4 times more request for the same reconfiguration.

For clarity, we have assumed that allocated resources can be used and paid for any period of time and not on, e.g., a per-hour basis. This scenario is most relevant to a private cloud but the approach can equally well be applied to infrastructures with hourly payment schemes, although the time for repacking should then be aligned with payment periods to make full use of resources already paid for.

Table 2 shows the resource utilization for each application on each VM type.

In our evaluations we consider seasonal and increasing

Application	VM Type	Price($/min)	Capacity(req/min)
Application A	small1	0.003	500
	small2	0.006	1197
	med1	0.012	2867
	med2	0.024	6868
	large1	0.048	16449
	large2	0.09	39396
Application B	small1	0.003	500
	small2	0.006	1310
	med1	0.012	3434
	med2	0.024	9000
	large1	0.048	23588
	large2	0.09	61819

Table 2: Server configurations and their prices.

workload patterns. We perform the evaluation by generating two synthetic workloads of each type, as shown in Figure 1. Our 4 workloads were different in volume and duration with 168, 182, 215 and 338 hours length.

Since the time required for actuating a reconfiguration contributes with overhead costs that impacts the decision of when it is beneficial to perform reconfiguration, we include two different reconfiguration durations, 4 minutes and 20 minutes. We also set the gain factor $K = 1$ % which implies that the expected benefit should be higher than 1% in order for repacking to be performed. Notably, 1% may seem like a small number but it should be remarked that it is with respect to total infrastructure cost.

For each application runtime, we measure the following parameters:

1. **Total cost of running the application.** The total cost is the aggregated resource utilization cost, from start to the end. We compare the total cost of an application when using different scaling strategies. This helps us to reason about the cost efficiency of each of them.

2. **Number of reconfigurations performed.** The number of reconfigurations during the application lifetime provides insight into the effects of reconfiguration cost and stability interval of the reconfiguration. It also shows how the cost benefit analysis filters the unnecessary reconfigurations and result a reduction in total cost.

VM types are each associated with a price p_i for resources consumed during a time unit. Each VM type serves a maximum capacity C_i^A and C_i^B for application A and B respectively. *Total load* is measured as the aggregated load at time t where $\Delta load = total\ load_t - capacity_{t-1}$.

We evaluate five strategies, S_{delta}, $S_{delta\_Repacked}$, S_{base}, $S_{base\_Repacked}$ and S_{Full}. We compare the total cost of running an application for each of these five strategies. For the scaling strategies that are selecting *optimal set* of VMs, i.e, $S_{delta\_Repacked}$, $S_{base\_Repacked}$ and S_{Full}, we formulate the problem of finding the optimal set as an Integer Linear Programming (ILP) model and used Gurobi solver to solve this problem. The result is an instance set, that represents the VMs with a sufficient capacity and minimum cost.

6.2 Repacking and scaling algorithms

To demonstrate the effect of repacking, we compare the autoscaling strategies presented in Section 4 when using our proposed repacking method $S_{delta\_Repacked}$ and $S_{base\_Repacked}$ against the same strategies but without repacking (S_{delta}, S_{base}, S_{Full}).

Table 3 and Figure 2 show the total cost of running applications A and B for four workloads. In Figure 2 each set of bars compares the total cost of running each application, on *Y axis*, for five autoscaling strategies and two different reconfiguration costs (*Cheap R* and *Expensive R*). The first and third set of bars show the result of repacking when the reconfiguration cost is cheap whereas the second and fourth set present the result for expensive reconfigurations. The following are observations after analyzing these 5 autoscaling strategies:

1. Comparison of non-repacked scaling mechanisms against their repacked counterparts (S_{delta} with $S_{delta\_Repacked}$ and S_{base} with $S_{base\_Repacked}$) shows that the total cost is reduced when repacking is applied.

2. Comparison of repacking strategies ($S_{delta\_Repacked}$ and $S_{base\_Repacked}$) against S_{Full} shows that the total cost obtained when applying the repacking strategy is lower than

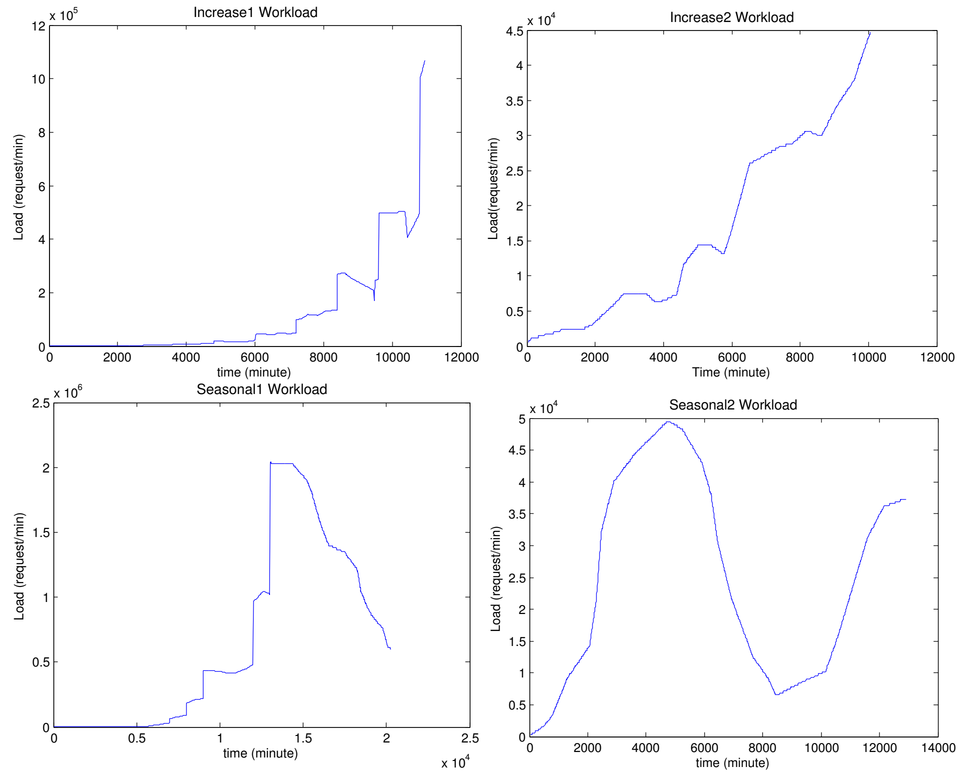
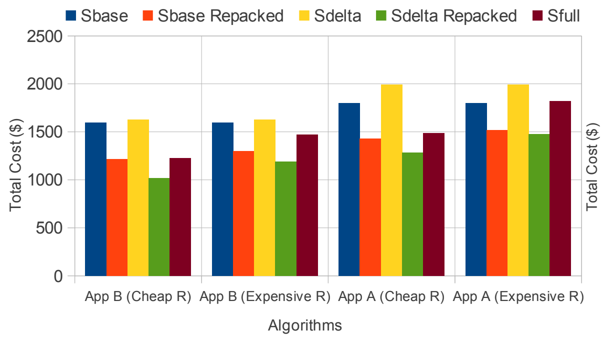
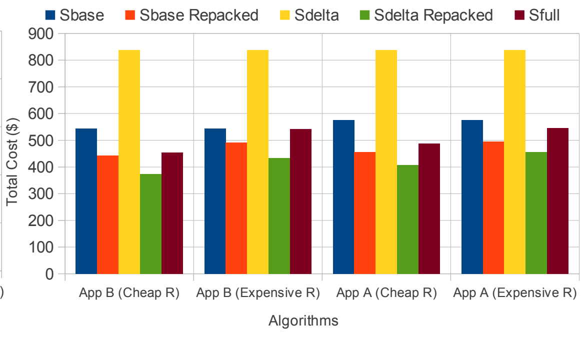


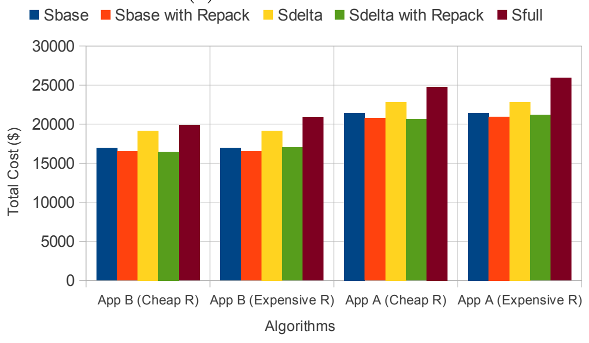
Figure 1: The workloads used in experiments.



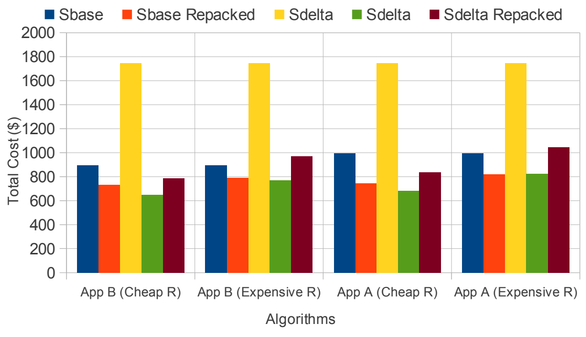
(a) Increase1 workload



(b) Increase2 workload



(c) Seasonal1 workload



(d) Seasonal2 workload

Figure 2: Comparison between total costs of running applications resulted by different scaling algorithms and different workloads when $d = 1000$.

the cost of S_{Full}. Lower cost is a result of performing reconfigurations at the appropriate time avoiding unnecessary costly reconfigurations thus reducing the total cost of running the application.

3. $S_{delta\_Repacked}$ (green bars) outperforms all other scaling strategies, producing the lowest total cost for running the applications. S_{delta} by itself is not an efficient autoscaler but it generates a resource set comprised of various VM types that increases the possibility of reducing reconfiguration costs. Repacking tries to keep common VMs in the VM set, adding extra VMs of other types only when they are not available. When repacking S_{delta}, reconfiguration costs are low because it is very probable to reuse an already started VM. This reduces the total cost after repacking.

4. $S_{base\_Repacked}$ (red bars) shows the second best result among the scaling strategies tested. S_{base} produces a VM set of mostly the same type of VMs, so transitioning to an optimal set requires starting several VMs of other types. This leads to higher reconfiguration costs making this strategy less efficient than $S_{delta\_Repacked}$.

5. S_{delta} (yellow bars) is the most expensive and least cost efficient strategy among all. The reason is that S_{delta} produces a resource set that mostly consists of a large number of small instances with high *price/performance* ratios.

6. S_{base} is a more efficient strategy than S_{delta} since it is less sensitive to small spikes in load, making it a more stable strategy. S_{base} can handle those spikes by previously over-provisioned resource sets. However, for this to happen, *base instance* must be correctly determined with respect to workload volume and oscillations, which by itself is a non-trivial problem.

We also studied the case when an application owner wants to use only one instance type. In this case a new *base instance* is used for repacking instead of repacking to optimal. This strategy results in expensive reconfiguration costs, as the entire resource set should be replaced with the new resource set that only uses the new *base instance*. However, repacking still produces a benefit since it performs a cost-benefit analysis before doing the reconfigurations. In this case the number of reconfigurations performed is low due to the high reconfiguration costs. However, from a pure cost-perspective this approach is not competitive.

6.3 Reconfiguration cost and stability intervals

Repacking decisions are closely dependent on the reconfiguration cost. This extra cost needs to be carefully considered before deciding to repack to evaluate if repacking is actually beneficial. We defined this cost as a function of reconfiguration time and it is the time required for the new servers to become available, or the time that both old and new resource sets are concurrently alive in order to preserve performance. We studied the behavior of the repacking model when the reconfiguration cost is cheap and expensive.

Before repacking we need to estimate the expected stability interval, d, of the new configuration. In order to show the effects of stability intervals on reconfiguration numbers, we assumed that there is uncertainty in predicting stability intervals. In this experiment we have assumed a rough prediction of $d = 180$ minutes for *Increase1, Seasonal1*, and *Seasonal2* workloads, and $d = 100$ minutes for *Increase2* workload. The $d = 100$ minutes and $d = 180$ minutes in Table 4 highlight the effects of short stability intervals on repacking decisions with expensive reconfigurations by showing a decrease in number of reconfigurations. We also have a more accurate estimation of $d = 1000$ minutes for our workloads, which shows the behavior of the repacking model for long stability intervals. In our case, we set the values of d by observing the workload pattern beforehand, although this value can be forecasted using statistical models such as ARIMA [23]. With this setup we make the following observations.

1. Increasing the reconfiguration cost increases the total cost of running the application in S_{Full}, $S_{base\_Repacked}$ and $S_{delta\_Repacked}$.

2. Doing expensive reconfigurations for a short stability interval (d) is not reasonable since the next reconfiguration takes place before the overhead cost of a previous one can be amortized. As presented in Table 4, when reconfiguration cost is expensive the repacking algorithm decides to perform 13 reconfigurations less compared to when reconfigurations are cheap in *Increase*1 workload with $d = 180$. This argument is also valid for *Seasonal1*, *Seasonal2*, and *Increase2* workloads, with 20, 12, and 10 less reconfigurations respectively.

3. If a reconfiguration lasts long enough even costly reconfigurations can be beneficial. In this case the number of reconfigurations are almost equal for both cheap and expensive scenarios.

Workload	**d=180 min**		**d=1000 min**	
Reconfig cost	Cheap	Expensive	Cheap	Expensive
Increase1	46	33	46	46
Seasonal1	87	67	89	89
Seasonal2	50	38	52	49
Workload	**d=100 min**		**d=1000 min**	
Reconfig cost	Cheap	Expensive	Cheap	Expensive
Increase2	22	12	23	23

Table 4: Number of reconfigurations with respect to different stability intervals (d).

Our experiments show that for cheaper reconfigurations more reconfiguration actions are triggered and the resource set is more frequently repacked to the optimal with a low reconfiguration cost. In contrast, for high reconfiguration costs some reconfigurations are avoided since the cost benefit analysis does not show any benefit for doing costly reconfigurations.

Table 5 shows the effect of the number of beneficial reconfigurations on the total cost of running the application. A reduced number of beneficial reconfigurations is either due to a short stability interval or to a high reconfiguration cost and it leads to an overall higher resource cost, the more reconfigurations, the more opportunities for tuning the resource set to optimal. However, the number of reconfigurations is not the only factor for reaching an optimal allocation. A more important factor for total cost reduction is the difference between the optimal cost and the cost for the old VM set for each reconfiguration opportunity.

	Reconfiguration cost	Cheap	Expensive	Cheap	Expensive
Workload type	Scaling strategy	App A		App B	
Increase1	S_{base}	1797.62	1797.62	1598.3	1598.3
	$S_{base\_Repacked}$	1428.86	1518.42	1218.03	1296.65
	S_{delta}	1989.62	1989.62	1630.12	1630.12
	$S_{delta\_Repacked}$	1283.17	1475.6	1020.55	1189.86
	S_{Full}	1488.69	1819.13	1224.64	1472.33
Increase2	S_{base}	576.27	576.27	544.86	544.86
	$S_{base\_Repacked}$	456.84	495.73	443.18	490.94
	S_{delta}	838.31	838.31	838.29	838.29
	$S_{delta\_Repacked}$	407.32	455.56	373.43	433.09
	S_{Full}	488.79	545.9	454.06	542.85
Seasonal1	S_{base}	21393.74	21393.74	16961.23	16961.23
	$S_{base\_Repacked}$	20732.84	20969.79	16502.5	16517.3
	S_{delta}	22801.98	22801.98	19152.16	19152.16
	$S_{delta\_Repacked}$	20579.89	21193.29	16476.5	17003.44
	S_{Full}	24740.29	25936.8	19839.8	20857.36
Seasonal2	S_{base}	993.19	993.19	894.91	894.91
	$S_{base\_Repacked}$	743.83	820.75	730.58	791.78
	S_{delta}	1745.86	1745.86	1744.65	1744.65
	$S_{delta\_Repacked}$	681.06	824.12	645.35	769.06
	S_{Full}	835.6	1043.87	784.48	970.67

Table 3: Total cost ($) of running the servers with different scaling models and different workloads when $d = 1000$.

Workload		**Cheap**		**Expensive**	
	Stability interval	d=180	d=1000	d=180	d=1000
Increase1	Number of reconfigs	46	46	33	46
	Total cost	1020.55	1020.55	1209.32	1189.86
Seasonal1	Number of reconfigs	87	89	67	89
	Total cost	16490.54	16476.5	17017.66	17003.44
Seasonal2	Number of reconfigs	50	52	38	49
	Total cost	646.38	645.35	857.48	824.12
Workload		**Cheap**		**Expensive**	
	Stability interval	d=180	d=1000	d=180	d=1000
Increase2	Number of reconfigs	22	23	12	23
	Total cost	374.01	373.43	468.79	433.09

Table 5: Total cost of running the application for different Stability interval, d values.

6.4 Price/performance ratio

When running our two applications, *App A* and *App B*, we can choose from 6 VM types. Each VM$_i$ can serve application A with performance C_i^A and application B with performance C_i^B at a cost p_i. A larger i represents a larger VM so that $C_{i+1} > C_i$. We use the *price/performance* ratio (p_i/C_i) as metric for cost efficiency of VM$_i$ for the specific application. The repacking assumption is that the *price/performance* ratio of larger VMs is lower than for smaller ones, so repacking smaller instances becomes reasonable.

To understand which *price/performance* ratios make repacking of interest, in our experiments, we assume that our applications are different in relative performance ratio according to VM sizes. So, for the same p_{i+1}/p_i, $C_{i+1}^A/C_i^A < C_{i+1}^B/C_i^B$, i.e., for the same price, larger VMs are capable of serving more requests for application B in comparison to application A. Table 2 shows the *price* and *capacity* of different VMs of each application used in our experiments.

With these parameters, repacking is more beneficial for applications with a larger difference between p_{i+1}/C_{i+1} and p_i/C_i, in our case application B. As seen in Table 6, the total cost of running application A with *Increase1* workload when the reconfiguration costs are cheap is reduced by 35.51%, while this number is 37.39% for application B that has a lower *price/performance* ratio. This reduction is 25.84% for application A, while it is 27.01% for application B that has high reconfiguration costs. The same pattern can be seen when the applications encounter other workload types or have high reconfiguration costs. The reason is that for applications with low *price/performance* ratios, larger instances are capable of serving more requests for a lower cost, which is a good motivation for repacking the instances to larger VMs in order to reduce the total cost of running the application.

Workload	**Cheap R**		**Expensive R**	
	App A	**App B**	**App A**	**App B**
Increase1	35.51 %	37.39 %	25.84%	27.01%
Increase2	51.41%	55.45%	45.66%	48.34%
Seasonal1	9.74%	13.97%	7.06%	11.22%
Seasonal2	60.99%	63.04%	52.8%	55.92%

Table 6: Improvement percentage achieved by using repacking for applications A and B.

7. RELATED WORK

There are several contributions that relate to our work. The first group of related studies focus on capacity autoscaling in clouds [7, 3, 14, 5, 16, 2]. Their objective is typically to adjust the allocated capacity to demand so that the required performance is provided, and their main concern is to predict the capacity as accurate as possible to ensure that the capacity is available when needed. Methods such as, static threshold based controllers [16, 2], control theory [18, 19], queuing theory [3, 5], and time series analysis of workloads [9, 11] are used to handle autscaling problem in clouds.

Another group of related studies focuses on cost efficient resource provisioning in cloud environments. Cost efficiency in clouds can be discussed from an infrastructure provider or from an application owner point of view. The difference is that infrastructure providers are more concerned with high resource utilization and energy efficiency of their infrastructure, so they try to avoid overprovisioning as much as possible by efficient consolidation strategies [17, 8] or overbooking policies [25, 4]. On the other hand, application owners are concerned with the performance of their applications and budget constraints, so their focus is on cost aware scheduling strategies [20, 6, 26] and smart resource acquisition [15] to handle the job with the minimum budget. However, in all the aforementioned works, planning the capacity is normally performed solely based on load, regardless of considering an optimal combination of VM types. To the best of our knowledge, only few studies taking VM sizes and their performances into account but then mostly for the optimal placement of VMs [12, 13, 24] rather than autoscaling.

Runtime reconfiguration techniques and measuring their associated costs is also a topic of interest when studying repacking. Jung et al., [10], proposes an adaptation engine for runtime reconfiguration in multi tier applications. They evaluate the impact of 5 reconfiguration actions on application response time and developed a middleware to generate cost-aware adaptation actions. However, they approached the problem of on-demand reconfiguration from an infrastructure provider perspective. Although the overall idea of reconfiguration is similar for both infrastructure providers and application owners, the target problem is different. Infrastructure providers often perform reconfigurations to ensure availability and isolation of demand fluctuations in colocated VMs, whereas application owners employ reconfigurations to reduce cost and increase performance.

Sharma et al, [22], introduce a cost aware provisioning system that takes into account price differentials of server types in order to minimize the rental cost of the application. They considered cost aware reconfiguration of resources as part of the autoscaling mechanism. While this is the closest study to our work, a main difference is that we decouple reconfiguration from autoscaling so that more freedom is given to the application owner when choosing autoscaler. The result is that application owners can choose the autoscaler that better fits the application's needs while at the same time benefiting from the advantages of repacking.

8. CONCLUSION

In this paper we have shown that combining the benefits of *vertical* and *horizontal* elasticity to scale in terms of both the number and the size of VMs increases the cost efficiency of the resource set used to serve an application. We propose a cost-benefit based approach called *repacking*, which takes cost and stability of a reconfiguration into account to determine the appropriate trade-off between horizontal and vertical scaling and acts accordingly. The goal of the repacking method is to find a configuration of resources that minimizes the resource utilization cost, given a current configuration, a number of different VM-types with different price/performance characteristics, a cost-function for performing repacking, and an expected time (duration) for which the new configuration is expected to be used.

Through experimental evaluations we have shown that by performing a cost-benefit analysis we can decide when and how to replace a non-optimal set by a new optimal set, and that decision can reduce the total cost of resource utilization during the application's lifetime. Our results show a range of 7% up to 60% cost saving in total resource utilization cost of our sample applications and workloads.

We also show that different applications benefit differently from the repacking approach. Applications with larger differences in *price/performance* ratio among their VM types are the ones that benefit most. Moreover, we studied the effects of cheap and costly reconfigurations on repacking behavior. Our results show that costly reconfigurations become reasonable to perform when the expected stability of the reconfiguration is long enough to amortize the overhead cost of reconfiguration. Hence, our approach automatically avoids unnecessary reconfigurations if a short stability interval for that reconfiguration is expected.

9. ACKNOWLEDGMENT

Financial support for this work is provided in part by the European Commissions Seventh Framework Programme ([FP7/ 2001 -2013]) under grant agreement number 257115, OPTIMIS and Swedish Research Council (VR) under contract number C0590801 for the project Cloud Control.

10. REFERENCES

[1] Amazon EC2: http://aws.amazon.com/ec2/instance-types/.

[2] Rightscale cloud management. http://www.rightscale.com/, 2012.

[3] A. Ali-Eldin, J. Tordsson, and E. Elmroth. An adaptive hybrid elasticity controller for cloud infrastructures. In *Network Operations and Management Symposium (NOMS), 2012 IEEE*, pages 204–212. IEEE, 2012.

[4] D. Breitgand, Z. Dubitzky, A. Epstein, A. Glikson, and I. Shapira. Sla-aware resource over-commit in an iaas cloud. In *8th international conference on systems virtualiztion management (svm) Network and service management (cnsm), 2012*, pages 73–81, Oct.

[5] R. Chi, Z. Qian, and S. Lu. A game theoretical method for auto-scaling of multi-tiers web applications in cloud. In *Proceedings of the Fourth Asia-Pacific Symposium on Internetware*, Internetware '12, pages 3:1–3:10, New York, NY, USA, 2012. ACM.

[6] T. N. B. Duong, X. Li, R. Goh, X. Tang, and W. Cai. Qos-aware revenue-cost optimization for latency-sensitive services in iaas clouds. In *16th International Symposium on Distributed Simulation and Real Time Applications (DS-RT), 2012 IEEE/ACM*, pages 11–18, Oct.

[7] H. Ghanbari, B. Simmons, M. Litoiu, C. Barna, and G. Iszlai. Optimal autoscaling in a IaaS cloud. In *Proceedings of the 9th international conference on Autonomic computing*, pages 173–178. ACM, 2012.

[8] I. Goiri, J. Guitart, and J. Torres. Characterizing cloud federation for enhancing providers' profit. In *3rd International Conference on Cloud Computing (CLOUD), 2010 IEEE*, pages 123–130. IEEE, 2010.

[9] Z. Gong, X. Gu, and J. Wilkes. Press: Predictive elastic resource scaling for cloud systems. In *International Conference on Network and Service Management (CNSM), 2010*, pages 9–16. IEEE, 2010.

[10] G. Jung, K. Joshi, M. Hiltunen, R. Schlichting, and C. Pu. A cost-sensitive adaptation engine for server consolidation of multitier applications. *Middleware 2009*, pages 163–183, 2009.

[11] J. Kupferman, J. Silverman, P. Jara, and J. Browne. Scaling into the cloud. *CS270-Advanced Operating Systems*, 2009.

[12] W. Li, J. Tordsson, and E. Elmroth. Modeling for dynamic cloud scheduling via migration of virtual machines. In *Proceedings of Third International Conference on Cloud Computing Technology and Science*, CLOUDCOM '11, pages 163–171, 2011.

[13] W. Li, J. Tordsson, and E. Elmroth. Virtual machine placement for predictable and time-constrained peak loads. In *Proceedings of the 8th international conference on Economics of Grids, Clouds, Systems, and Services*, GECON'11, pages 120–134, Berlin, Heidelberg, 2012. Springer-Verlag.

[14] T. Lorido-Botrán, J. Miguel-Alonso, and J. A. Lozano. Auto-scaling techniques for elastic applications in cloud environments. Technical report, Department of Computer Architecture and Technology, UPV/EHU, 2012.

[15] M. Mao and M. Humphrey. Auto-scaling to minimize cost and meet application deadlines in cloud workflows. In *Proceedings of International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 49:1–49:12, New York, NY, USA, 2011.

[16] M. Maurer, I. Brandic, and R. Sakellariou. Enacting slas in clouds using rules. *Euro-Par 2011 Parallel Processing*, pages 455–466, 2011.

[17] M. Mazzucco, D. Dyachuk, and R. Deters. Maximizing cloud providers' revenues via energy aware allocation policies. In *3rd International Conference on Cloud Computing (CLOUD), 2010 IEEE*, pages 131–138. IEEE, 2010.

[18] P. Padala, K.-Y. Hou, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, and A. Merchant. Automated control of multiple virtualized resources. In *Proceedings of the 4th ACM European conference on Computer systems*, pages 13–26. ACM, 2009.

[19] T. Patikirikorala and A. Colman. Feedback controllers in the cloud. *Swinburne University*, 2011.

[20] M. Salehi and R. Buyya. Adapting market-oriented scheduling policies for cloud computing. *Algorithms and Architectures for Parallel Processing*, pages 351–362, 2010.

[21] M. Sedaghat, F. Hernández, and E. Elmroth. Unifying cloud management: Towards overall governance of business level objectives. In *11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 591–597. IEEE, 2011.

[22] U. Sharma, P. Shenoy, S. Sahu, and A. Shaikh. Kingfisher: A system for elastic cost-aware provisioning in the cloud. *Dept. of CS, UMASS, Tech. Rep. UM-CS-2010-005*, 2010.

[23] J. M. Tirado, D. Higuero, F. Isaila, and J. Carretero. Multi-model prediction for enhancing content locality in elastic server infrastructures. In *18th International Conference on High Performance Computing (HiPC)*, pages 1–9. IEEE, 2011.

[24] J. Tordsson, R. S. Montero, R. Moreno-Vozmediano, and I. M. Llorente. Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers. *Future Gener. Comput. Syst.*, 28(2):358–367, Feb. 2012.

[25] B. Urgaonkar, P. Shenoy, and T. Roscoe. Resource overbooking and application profiling in shared hosting platforms. *ACM SIGOPS Operating Systems Review*, 36(SI):239–254, 2002.

[26] M. Zhu, Q. Wu, and Y. Zhao. A cost-effective scheduling algorithm for scientific workflows in clouds. In *31st International on Performance Computing and Communications Conference (IPCCC), 2012 IEEE*, pages 256–265, Dec.