H-0331 (HAI1801-005) January 16, 2018
Computer Science

IBM Research Report

Heterogeneous Resource Reservation

Ofer Biran, David Breithand, Dean Lorenz, Michael Masin, Eran Raichstein, Avi Weit
IBM Research Division
Haifa Research Laboratory
Mt. Carmel 31905
Haifa, Israel

Ilyas Iyoob
IBM GTS
11501 Burnet Road
Austin, TX 78758-3400, USA

Research Division
Almaden – Austin – Beijing – Cambridge – Dublin - Haifa – India – Melbourne - T.J. Watson – Tokyo - Zurich

Heterogeneous Resource Reservation

Ofer Biran, David Breithand, Dean Lorenz,
Michael Masin, Eran Raichstein, and Avi Weit
IBM Research – Haifa
Email: {biran,davidbr,dean,michaelm,eranra,weit}@il.ibm.com

Ilyas Iyoob
IBM
Email: iiyoob@us.ibm.com

***Abstract*— Given a large variety of resources and billing contracts offered by today's cloud providers, customers face a non-trivial optimization challenge for their application workloads.**

A number of works are dealing with either billing contracts selection optimization or resource types selection. We argue that the largest cost savings to elastic workloads result from jointly optimizing heterogeneous resources and billing contracts selection. To this end, we introduce a novel cloud control and management framework and formulate a novel optimization problem called Heterogeneous Resource Reservation (HRR).

We evaluate our solution through a thorough simulation study using publicly available cloud workload data as well as internal anonymous customer data. For these data we demonstrate dramatic cost savings are attainable through our proposed approach.

I. Introduction

At the dawn of the cloud era, on-demand resource procurement model for relatively limited resource repertoire was expected to dominate, because of its simplicity and the promise to eliminate resources rightsizing and long term commitments and upfront capital investments, effectively relieving a customer from capacity planning overhead. However, very soon providers started differentiating themselves both on resource heterogeneity and procurement models offering sustained usage discount schemes coming both as explicit capacity reservations models and implicit *a posteriori* discount. These schemes aim at setting monetary incentives for the customers for committing to specific levels of demand to help providers with capacity planning cycle[1]

As a result of this evolutionary processes, the current cloud computing landscape became very complex. A typical cloud today offers a variety of heterogeneous resources and procurement models (i.e., contracts) to choose from. The problem becomes even more complex when workloads should be deployed and operated on different clouds and brokering decisions should be made for specific workloads, suggesting the best clouds to host them. This complexity is both a promise and a challenge. Essentially, a cloud customer faces a reincarnation of a capacity planning problem, where the new complexity stems from proliferation of the cloud procurement and billing models as opposed to the well studied traditional capacity planning problem arising in the context of a data center owned by the application provider.

In this paper, we argue that it is highly beneficial to consider both resources heterogeneity and long term capacity reservation within a unified optimization framework. We demonstrate that this approach allows to dramatically slash operational costs while maintaining application Quality of Service (QoS). Naturally, long term commitment implied by capacity reservation, requires some form of forecasting to predictively provision resources. A number of works on various forms of predictive elasticity have appeared recently (see Section II). Also, optimally allocating heterogeneous resources to application workloads has received considerable attention [1]. However, to the best of our knowledge, no unified framework that combines optimal predictive capacity reservation with multi-dimensional heterogeneous cloud resources procurement, has been presented thus far. This paper intends to fill this void by developing such a unified generic cost optimization model that can be applied uniformly across all major cloud providers.

Essentially, the problem comprises two sub-problems:

Demand Forecasting. Forecast demand of an application in terms of resource requests for the forecasting horizon, which is comparable to the billing contract commitment (if such a commitment exists). In general, precise forecasting of time series for mid-term and long term horizons is very difficult. We work around this problem by predicting an empiric distribution of the demand, i.e., relative frequencies of request combinations rather than their exact timing.

Reservation and Allocation. Given a forecast of the resources demand, a catalog of VM types, and a catalog of billing contracts, find an optimal allocation of demand to instances of heterogeneous VM types along with an optimal selection of the billing contracts to be used with these VM types. In Section IV, we formulate this problem as a mixed integer linear problem, which can be efficiently solved by existing solvers even for the very large problem instances as demonstrated by our evaluation experiments (millions of decision variables and tens of thousands of constraints).

Today's State Of The Art (SOTA) in cloud operations cost management can be summarized as follows. Application requests are statically mapped on VM types with matching CPU/memory ratio. A forecast mechanism is used to predict future application demand. Based on this prediction, long term commitments are made either explicitly (i.e., through

The research leading to these results was partially funded by the European Community's Seventh Framework Programme(FP7/2007-2013) under grant agreement n 610802

[1]Note that in a pure on-demand model, a provider discovers customer's demand reactively, which considerably complicates capacity planning.

reservations) or implicitly (i.e., by keeping VMs up to obtained planned sustained usage discount). Excess demand not accounted by the forecast mechanism is covered by on-demand VM procurement.

Using Google data, we demonstrate that, on average, HRR achieves 55% higher cost savings by its joint optimization strategy compared to the SOTA above. This almost doubles cost savings achievable through separately optimizing selection of heterogeneous resources and billing contracts. Moreover, we demonstrate that even when SOTA methodology is used with perfect knowledge of application requests distribution for the next month (which is clearly a theoretical setting), HRR still outperforms SOTA by 48%, on average. Likewise, we show that when VMs are procured on-demand, even when assuming perfect knowledge of exact time series of application requests, HRR still outperforms this strategy by 59%. Also, it is important to stress that our results are within 2-3% gap from an optimal solution.

II. Related Work

An approach conceptually similar to our own is presented in [2]. The authors jointly optimize heterogeneous VM selection and billing contracts using an advanced two-stage integer stochastic programming problem. This model is being transformed into Equivalent Deterministic Formulation (DEF) using explicit knowledge of application resource demand and cloud resources pricing distributions. They apply Sample Average Approximation (SAA) for obtaining DEF size susceptible to solving using a linear solver. Besides different problem formulation techniques, there are two main differences between our work and [2]. First, [2] uses static mapping between VM types and application request types (we term this "dedicated" requests to VMs mapping). We demonstrate that by making this decision more than once, one can almost double cost savings. Second, billing contracts modeling of [2] is rather rudimentary, considering only three basic contract types: on-demand, one year and three years reservations. At the time of [2] being published many modern contract types, such as AWS convertible reserved instances or Google sustained usage discounts have not been invented yet. In our model, we show how these new contract types can be explicitly modeled to allow seamless brokerage across clouds [3].

An IP formulation and a heuristic-based solution for optimizing billing contracts selection for single VM type are developed in [4]. In [5], the joint optimization problem of billing contracts and heterogeneous VMs is studied for single resource dimension (CPU). In [6], competitive deterministic and randomized online algorithms for optimizing capacity reservation for VM of the same type are presented (based on extending the Bahncard problem [7]). In [8], contract optimization for homogeneous VMs using load prediction, is studied.

Workload prediction attracted considerable attention in the literature. In [9] HMM and ARIMA are used to predict application workload fluctuation. In [10] application of ARIMA to cloud workload prediction and its application to resource management is studied. In [8] a number of workload prediction techniques, including ARIMA, Seasonal ARIMA (S-ARIMA), Exponential Smoothing, Holt-Winters and other methods are explored and compared. A two level ensemble method based on regression and correlation between VMs is used in [11] to capture highly transient demand. PRESS [12] uses a method combining signal processing and discrete-time Markov chain to predict workload. Neural networks and linear regression are used in [13] to predict resource consumption. In [14] a workload prediction technique based on past pattern matching is developed.

III. End-to-End System

A. Approach

All major cloud vendors support dynamic workload allocation over homogeneous resources through *Auto-Scaling Groups (ASG)*[2]. The amount of resources in each group is dynamically adjusted to fit the overall workload Heterogeneity is desired, as application workload requests are often unbalanced with respect to resource consumption. In essence, given an expectation of a specific application workload in terms of request types combination distribution over some forecasting horizon, the problem is to find a mapping of application requests (i.e., resource demand vectors) on heterogeneous VM instances (heterogeneous bins) resulting in a minimal cost. This is a difficult problem since it effectively amounts to bin packing, which is NP-hard even in case of the homogeneous bins. On top of that, there is a need to optimize selection of billing contracts associated with VMs. For each billing contract type and for each VM type, decision whether VM instances of that type will be provided on-demand or using some reservation.

Algorithm 1 captures the overall framework. The algorithm continuously solves the HRR problem using a forecast of application request combinations distribution. A reader interested in the details of the forecast mechanism used is referred to [3].

Algorithm 1: Overall Algorithmic Framework

```
Input: VM Types, Billing Contracts, History of Resource Requests
Output:
        (1) Optimized Selection of VM types
        (2) Optimized Selection of Billing Contracts
        (3) Optimized Requests Allocation to VM types
History ← accumulate request combination data points;
repeat
    forecast ← ECDF(History);
    ((1), (2), (3)) ← HRR(forecast);
        configure Load Balancer weights;
        update Reservations Inventory;
        control Auto-Scaling Groups sizes;
    History ← History ⋃ new data points (if needed);
until TRUE;
```

B. Architecture

We utilize existing cloud features to execute the output of Algorithm 1. We assume that a cloud workload (i.e., an application) is characterized by a resource footprint that comprises

[2]We adopt the AWS terminology.

heterogeneous cloud resources. We create a *Heterogeneous ASG* by grouping multiple homogeneous ASGs under a top level LB with configurable weights[3]. This hierarchical structure is not supported by cloud providers out of the box, but it is readily achievable with the existing cloud offerings.

Figure 1 depicts our proposed cloud cost optimization architecture. Our system creates a plan that defines three control points: (1) which homogeneous ASGs should be used and what should be their size (2) which contracts should be used and how many resources of each type should be reserved (3) what weights should the top-level LB use to balance requests across the ASGs. As described above, the two sub-problems solved are *demand forecasting* and *optimal reservation and allocation*. A brief description of the system components follows.

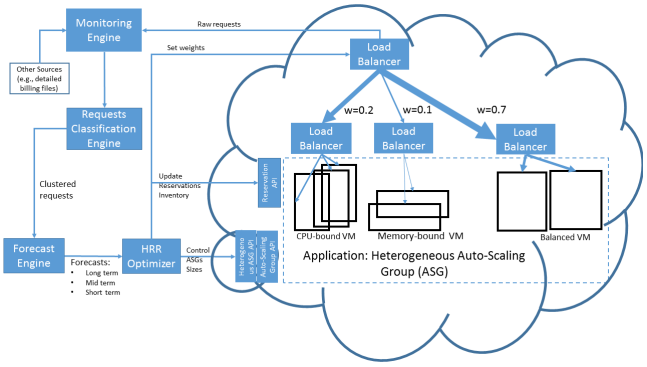

Fig. 1: Conceptual architecture

Monitoring Engine: samples the ASGs comprising the application at regular intervals assuming standard tools available for Layer 7 load balancers (e.g., NGINX).

Request Classification Engine: clusters similar application requests. A specific clustering method is pluggable. As explained in more details in Section V, one clustering method uses a ratio between CPU and memory of the application requests as a label.

Forecast Engine: uses labeled historical data to build a Empiric Cumulative Distribution Function (ECDF) of requests combinations with a target confidence band.

Optimization Engine: this component continuously solves HRR (see the formulation in Section IV). In our reference implementation, we use ILOG CPLEX [17] linear solver.

Top Level Load Balancer: implements an optimized mapping of application requests via configuring appropriate weights for each sub-ASG in a hierarchical LB structure.

IV. Optimal Heterogeneous Resource Reservation

In this section we provide the notations and formal definition of the HRR as a mixed linear problem. The notations are summarized in Table I.

[3]Most modern LBs have configurable weights feature. As representative examples, consider HAproxy [15] and NGINX [16].

TABLE I: Notation summary

Notation	Description
Sets	*Description and default index*
$\mathbf{R}$	request types, $r \in \mathbf{R}$
$\mathbf{V}$	virtual machine types, $v \in \mathbf{V}$
$\mathbf{B}$	billing contracts, $b \in \mathbf{B}$
$\mathbf{Q}$	resource types, $q \in \mathbf{Q}$
$\mathbf{H_t}$	forecasted $\bar{\mathbf{r}}[h]$, $h \in [t, t+1]$
Sequences	
$\mathbf{T}$	time slots $t \in \mathbf{T} = 1, \ldots, T$
Parameters	
$r[h]$	workload demand of request type r at time $h \in [t, t+1]$
$w_q(r)$	required resource q for a unit of request type r
$w_q(r_{\rightarrow v})$	number of v type VMs required per an r type request
$c_q(v)$	capacity of resource q in one virtual machine type v
$s^b(v)$	normalized setup fee per VM type v with contract b
T^b	reservation period for contract b
$u^b(v)$	hourly usage rate per one VM type v with contract b
Variables	
$a_{r_{\rightarrow v}}[t]$	fraction of type r requests served by type v VMs at time interval $[t, t+1]$
$y_v^b[t]$	number of type v VMs initialized under contract b at time t
$z_v^b[t]$	number of type v VMs used under contract b at time t
$x_v^b[h]$	number of type v VMs used under contract b at time $h \in [t, t+1]$

A. Input Parameters

There is a set of *virtual machines types* $\mathbf{V}$ and each virtual machine $v \in \mathbf{V}$ is characterized by its capacity along a set of consumable *resource types* $\mathbf{Q}$: with each $v \in \mathbf{V}$ we associate a vector $\bar{\mathbf{c}}(v) = \langle c_q(v) \rangle_{q \in \mathbf{Q}}$, where $c_q(v)$ is the capacity of a single instance of virtual machine v for resource dimension q, e.g., memory, CPU, disk, etc.

An application serves multiple request types, where each *request type* $r \in \mathbf{R}$, is characterized by its demand along the same resource dimension set $\mathbf{Q}$: each request type is associated with a vector $\bar{\mathbf{w}}(r) = \langle w_q(r) \rangle_{q \in \mathbf{Q}}$, where $w_q(r)$ is the capacity requirements of a single instance of request type r. We define $w_q(r_{\rightarrow v})$ as the capacity requirements of virtual machines of type v that are needed to satisfy the requirement of a single request of type r for resource q.

Discrete decisions on how many VMs of each type under each billing contract to keep active are being made at times h, $h \in H_t$, for some t, and VM configurations stay put throughout intervals $[h, h+1]$. Discrete decisions about purchasing new VMs are made at times t, which are multiples of h. T, which is a multiple of t denotes the overall planning horizon (e.g., h might correspond to hours, t might correspond to weeks, and T might correspond to months resolution). Application workloads are collected at discrete times, called *samples*: $\bar{\mathbf{r}}[h] = \langle r[h] \rangle_{r \in \mathbf{R}}$ denotes the combination of request demands at sampling time h. In general, accurate prediction of time series $\bar{\mathbf{r}}[h]$ for $h \in H_t$ is a very difficult task that requires application specific knowledge. Our optimization does not require an accurate prediction of time series. Rather than predicting the value of each $\bar{\mathbf{r}}[h]$ for every request type at every point in time, we only need to predict the counts of each value of $\bar{\mathbf{r}}[h]$ over the entire window H_t. In other words, we are only interested in the overall *number of hours* each $\bar{\mathbf{r}}$ is served and not the *exact times* it was served. Historic data points from time interval H_{t-1} are used to predict data points that will occur in any order in time interval $[t, T]$ from the reservation time t for the whole time horizon T. The order is immaterial, because we are only interested in the number of

occurrences of a particular resource request combination.

A *billing contract* $b \in \mathbf{B}$ defines an effective hourly rate for each virtual machine type. The effective hourly rate for each contract, b, per a particular virtual machine type, v, is comprised of a *usage* rate $u^b(v)$ and a *flat setup* fee $s^b(v)$. The usage rate is paid per each h per each virtual machine used (we will to h as *hour* even though this does not have to be an actual hour). The setup fee is paid per *payment cycle*, T^b, by the number of initialized virtual machines; that is, if $y_v^b[t]$ virtual machines of type v were initialized at time t then the flat setup cost for the entire pay cycle $[(t+1),(t+2),\ldots,(t+T^b)]$ would be $s^b(v) \cdot y_v^b[t]$. If the flat setup fee is not zero then a virtual machine can be used only if its setup fee was paid.

B. Optimization goal and output

Our goal is to minimize the overall cost of virtual machines required for providing elastic application on a cloud.

For a given workload, we seek an *execution plan* that defines the number of virtual machines of each type that we should use in any interval $[t_i, t_{i+1}]$ and the billing contracts by which these machines should be procured. Furthermore, our execution plan needs to be complemented by an *allocation plan* that specifies how many instances of each VM type are allocated to serve which requests.

The *execution plan* for each virtual machine type is denoted by two time series. The first series, $(y_v^b[1], y_v^b[2], \ldots y_v^b[T])$, defines how many virtual machines of type v are initialized under billing contract b at each point in time t. The second series, $(z_v^b[1], z_v^b[2], \ldots, z_v^b[T])$, defines how many virtual machines of type v are expected to be used under billing contract b during period t.

Although we defined above that $t = m \cdot h$, the forecast H_t might include more data points corresponding to "hours" h than m. This is due to bootstrapping and trending. To normalize $x_v^b[h]$ to m number of hours we use $\rho_t = \frac{m}{|H_t|}$: $z_v^b[t] = \rho_t \sum_{h \in H_t} x_v^b[h] \quad \forall v \in \mathbf{V}, b \in \mathbf{B}, t \in T$.

The *allocation plan* is denoted by the time series $(a_{r \to v}[1], a_{r \to v}[2], \ldots a_{r \to v}[T])$, where $a_{r \to v}[t]$ defines the fraction of request of type r that should be served by virtual machines of type v at time t. At run time, the allocation $a_{r \to v}[t]$ is realized by a load-balancer that supports weight-based distribution simply by using $a_{r \to v}[t]$ as the weights. The load-balancer needs to set different weights per request type. Therefore, it must be able to classify requests as they are being served. In general, the weights should be adjusted per time slot t, to match $a_{r \to v}[t]$. However, if dynamic updates cannot be supported by the underlying infrastructure, then we add an additional constraint that $a_{r \to v}[t]$ remains static, *i.e.*, $a_{r \to v}[t] = a_{r \to v}$ for all $t \in T$.

C. Basic Model (B-HRR)

We can now formally define the model for the HRR problem.

Given a workload:

$$W[t, t+1] = (\bar{\mathbf{r}}[1], \bar{\mathbf{r}}[2], \ldots \bar{\mathbf{r}}[m * h]), t \in [1, T] \qquad (1)$$

Minimize the total cost:

$$\textit{TotalCost} = \sum_{t \in T v, \in \mathbf{V}, b \in \mathbf{B}} \left(s^b(v)\, y_v^b[t] + u^b(v)\, z_v^b[t] \right) \qquad (2)$$

Subject to the following constraints:

$$\sum_{v \in \mathbf{V}} a_{r \to v}[t] = 1 \qquad \begin{matrix} \forall r \in \mathbf{R} \\ \forall t \in T \end{matrix} \qquad (3)$$

$$\sum_{b \in \mathbf{B}} c_q(v)\, x_v^b[h] \geq \sum_{r \in \mathbf{R}} a_{r \to v}[t] r[h] w_q(r \to v) \qquad \begin{matrix} \forall v \in \mathbf{V} \\ \forall q \in \mathbf{Q} \\ \forall t \in T \\ \forall h \in H_t \end{matrix} \qquad (4)$$

$$x_v^b[h] \leq \sum_{\tau = t - T^b + 1}^{t} y_v^b[\tau] \qquad \begin{matrix} \forall \{b \in \mathbf{B} | s^b(v) > 0\} \\ \forall v \in \mathbf{V} \\ \forall t \in T \\ \forall h \in H_t \end{matrix} \qquad (5)$$

$$z_v^b[t] = \rho_t \sum_{h \in H_t} x_v^b[h] \qquad \begin{matrix} \forall v \in \mathbf{V} \\ \forall b \in \mathbf{B} \\ \forall t \in T \end{matrix} \qquad (6)$$

$$a_{r \to v}[t], z_v^b[t] \geq 0; \quad y_v^b, x_v^b \in Z^* \qquad \begin{matrix} \forall v \in \mathbf{V} \\ \forall b \in \mathbf{B} \\ \forall q \in \mathbf{Q} \\ \forall t \in T \end{matrix} \qquad (7)$$

The *TotalCost* (2) summarizes setup and usage costs of all machines under all contracts. Constraint (3) ensures that the allocation plan is valid and can be rewritten as $\sum_{v \in \mathbf{V}} a_{r \to v} = 1$ $\forall r \in \mathbf{R}$ to support a static allocation.

The main constraint (4) ensures that the total capacity of the virtual machines is enough to support the total demand of the workload. It can be written in vector form as

$$\sum_{b \in \mathbf{B}} x_v^b[h] \bar{\mathbf{c}}(v) \geq \sum_{r \in \mathbf{R}} a_{r \to v}[t] r[h] \bar{\mathbf{w}}(r), \quad \begin{matrix} \forall v \in \mathbf{V} \\ \forall t \in T \\ \forall h \in H_t \end{matrix} \qquad (8)$$

where the LHS is the capacity of virtual machines of type v at time t and the RHS is the fraction of the overall demand that is served by those virtual machines.

The reservation constraint (5) is optional and applies only to billing contracts that offer reduced effective hourly rate if the virtual machines are reserved and paid for in advance. Such contracts have a non-zero setup cost and may have zero usage rate. The RHS is the total number of virtual machines that were reserved and are still available at time t. In the special case, where reservations are only made once, (5) becomes

$$x_v^b[h] \leq y_v^b. \qquad \forall v \in \mathbf{V}, \forall b \in \mathbf{B}, \forall t \in T, \forall h \in H_t.$$

Constraints (6) calculate $z_v^b[t]$ based on $x_v^b[h]$ and synchronization coefficient ρ_t. Constraints (7) define the allowed domain of the decision variables.

V. Evaluation

We study performance of our proposed model using trace-driven simulations. We use two data sets: a publicly available Google's clusterdata-2011-2 trace [18] and an obfuscated data of a customer who prior to migrating to IBM's SoftLayer has run its workload on AWS.

A. Google Trace Data

The clusterdata-2011-2 trace represents 29 days of information about compute jobs submitted to a Google comprising about 12.5K machines, in May 2011. The job records are

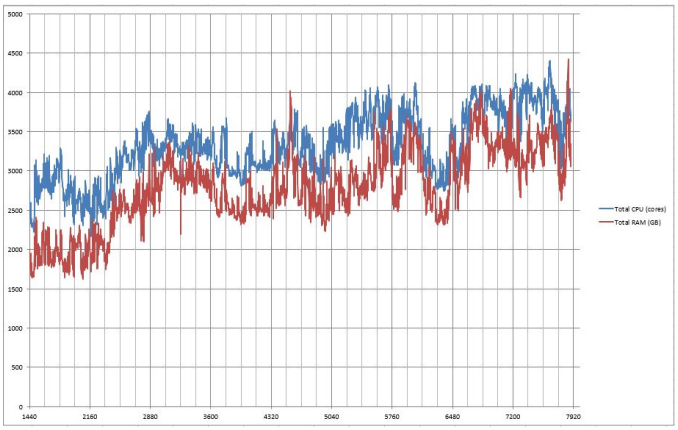

Fig. 2: A synthetic workload inspired by clusterdata-2011-2 trace

comprised of task records with each task record having normalized information on its resource requirements (e.g., CPU and memory). There are over 46M of individual tasks in the trace. We discard approximately 0.3% of tasks due to missing information, inconsistent time stamps or other similar problems.

Having only one month of data precludes us from using the trace as-is for exploring the role of the quantity discounts over long periods of time. To cope with this obstacle, we scaled the one month worth of the data to be a year long trace with one virtual "hour" of the scaled trace being $h = \frac{end\_time\_stamp}{365*24}$, which came out to be $h \approx 4.8$ min. This scaling resulted in approximately 5K tasks per "hour" as opposed to approximately 67K tasks per hour in the original trace. No claim is made that the actual one year worth of the trace data would look exactly as our scaled trace. Rather we use clusterdata-2011-2 trace as inspiration and a proxy for a massive realistic workload that can be executed on a cloud.

Another transformation of the original clusterdata-2011-2 trace that we perform deals with classifying application requests (i.e., tasks) into *types*. We compute a request type as $\frac{CPU}{mem}$ ratio with one digit after the decimal point to keep the number of types relatively small. This results resulting in 11 request types memory bound request types (the ratio is smaller than 1) and a similar number of CPU bound request types (the ratio is greater than 1). For each hour of the scaled trace, we calculate the number of requests of each type along with their total CPU and memory demand. The resulting time series is used as an input for our model.

Figure 2 shows the time series at resolution of one "hour" between "Month 3" and "Month 12", where demand is expressed in terms of total CPU and total memory requested by different requests at each "hour". As one can observe, the workload exhibits variability with peaks and valleys of demand. There is a moderate upward trend. The trace also indicates a two-month cyclicity.

B. AWS Customer Data

Our second data set comes from a real customer running its workload on AWS. This is a mid-size workload with hundreds of VMs deployed in 8 regions of AWS at any given time. The bulk of VM expenses occurs in just three regions: us-east, us-west1, and us-west2. Thus, in our experiments we focus on these three regions. The workload exhibits variability in resource demand. There is a moderate upward trend in consumption. There is no cyclicity in the workload[4].

Our input data comes from the standard AWS detailed billing files, which provide information about active VMs at hourly resolution. In this trace, we do not have information about resource consumption at the application request level. Rather we treat the CPU and memory parameters of a VM as a single "request". The VMs are then mapped on bare metal servers. For the sake of an experiment, we assume an inventory of the bare metal servers, which is identical in its assortment of configurations to that of the AWS VMs. It should be stressed that the resulting costs should not be treated literally by the reader, because, for one, AWS does not provide bare metal machines as a service to cloud consumers. This set of experiments demonstrates a potential value for cloud provider that would be interested to provide user's workload on the minimal number of physical servers.

C. Experiments

All our experiments have been performed on a standard laptop Lenovo TP W530 i7-3740QM 2.7 GH having 16 GB RAM and 8 cores. We used IBM ILOG CPLEX 12.6.3 [17].

Maximum running time for experiments performed on the clusterdata-2011-2 inspired trace was set 45 at minutes. For AWS trace maximum running time of the optimizer was configured to be 15 minutes. Relative optimal gap threshold was set at 0.1%[5].

Table II summarizes our experiments. First thing to notice is that the actual experiments have completed much faster than the maximal execution time configured. In all experiments, a one month forecast of ECDF of application requests combinationswas used. Table III and Table IV summarize the results of our evaluation study for the clusterdata-2011-2 inspired and AWS traces respectively. In the first set of experiments, summarized in Table III, the minimal history window used to calculate the next month ECDF forecast has been varied. The results corresponding to each history window are shown in a separate row. Table IV has similar structure to that of Table III, but each row represents a different AWS region. Minimal history used in this set of experiments was one month. Both tables show absolute total cost of resource consumption for each experiment and their relative performance with respect to HRR[6]. Relative overhead of a strategy A vs HRR is represented by the intersection of the

[4]Due to the lack of space we do not present graphs.

[5]Optimization stops if the gap between a candidate solution and the lower bound is smaller than the threshold.

[6]We used AWS VM inventory and prices to obtain concrete figures. These numbers are probably outdated since prices are subject to change.

TABLE II: Experiments Summary

Experiment	Acronym Meaning	Description
HRR	Heterogeneous Resource Reservation (HRR)	HRR with monthly forecast of application requests combinations distribution and dynamic mapping of request types to VM types. Demand unmatched by the reservations is addressed by on-demand elasticity of the cloud.
HRR-DK	HRR with full requests Distribution Knowledge	HRR with knowledge of the actual application requests distribution per month instead of a forecast.
HRR-TK	HRR with full requests Time series Knowledge	HRR with full knowledge of exact time series of requests for the entire planning horizon.
SOTA	The current State-Of-The-Art	HRR with static mapping of request types to VM types minimizing cost (i.e., matching CPU/memory ratios).
SOTA-DK	SOTA with full requests Distribution Knowledge	SOTA with knowledge of the actual application requests distribution per month instead of a forecast.
PE	Pure Elasticity	Request types are statically mapped to VM types minimizing cost. All VM instances deployed on demand.
E-TK	Elasticity with full requests Time series Knowledge	Request types are dynamically mapped to VM types with full knowledge of exact time series of requests for the next hour. All VM instances are deployed on demand.

TABLE III: Summary of experimental results for a synthetic workload inspired by thhe clusterdata-2011-2 trace

History	HRR	PE	SOTA	E-TK	SOTA-DK	HRR-DK	HRR-TK
1 week	5697	12614 (121%)	8795 (54%)	9108 (60%)	8476 (49%)	5681 (0%)	5518 (-3%)
2 weeks	5707	12614 (121%)	8656 (52%)	9108 (60%)	8476 (49%)	5681 (0%)	5518 (-3%)
1 month	5745	12614 (120%)	8654 (51%)	9108 (59%)	8476 (48%)	5681 (-1%)	5518 (-4%)
4 months	5815	12614 (117%)	9444 (62%)	9108 (57%)	8476 (46%)	5681 (-2%)	5518 (-5%)
(Average %)		(120%)	(55%)	(59%)	(48%)	(-1%)	(-4%)

TABLE IV: Summary of experimental results for the AWS detailed billing data trace

Region	HRR	PE	SOTA	E-TK	SOTA-DK	HRR-DK	HRR-TK
us-east1	6551	8753 (34%)	7038 (7%)	8327 (27%)	6353 (-3%)	6114 (-7%)	5972 (-9%)
us-west2	4442	6324 (42%)	4955 (12%)	5795 (31%)	4584 (3%)	3908 (-12%)	3768 (-15%)
us-west1	4556	6571 (44%)	4906 (8%)	6181 (36%)	4487 (-2%)	4260 (-6%)	4194 (-8%)
(Average %)		(40%)	(9%)	(31%)	(-1%)	(-8%)	(-11%)

A column and a row describing the experiment. The relative overhead RO is calculated as $RO = \frac{A-HRR}{HRR}100\%$.

Table III shows that, on average, HRR outperforms PE, SOTA, E-TK, and and SOTA-DK by 120%, 55%, 59%, and 48%, respectively. Table IV shows results for AWS trace. The gains are smaller (but still significant), because the trace already embodies best static mapping of requests to VMs.

To make it easier to navigate our experimentations, we explain the intuition behind them. HRR-DK, HRR-TK and E-TK are theoretical algorithms in a sense that they assume knowledge of the future. HRR-DK measures importance of precisely forecasting application request distribution. On average, on the data at hand, it performs only 1% better than HRR, even though it knows the exact distribution for the next month. While, it's easy to construct adversarial scenarios, our experiment demonstrates that there are reasonable inputs where exact knowledge of distribution is not a big advantage and simple non-parametric estimation works well enough.

HRR-TK's role is to explore importance of knowing the exact time series of requests vs distribution. HRR-TK improves over HRR and HRR-DK. This experiment emphasizes the importance of being able to configure Load Balancing every hour, which amounts to an additional 3% efficiency increase over HRR-DK, on average. E-TK illustrates the advantage of dynamic resource diversification. Dynamic mapping of request types to VM types reduces relative overhead by a factor of 2 compared to PE. SOTA and SOTA-DK demonstrate a similar opportunity offered by reserving heterogenous resources with static mapping of requests to resources.

A dramatic improvement happens when we unleash the full power of joint optimization of capacity reservation and resource diversification through HRR, where application request types are dynamically mapped on the VM types. As one can see, HRR is 48% better than SOTA-DK, on average. Its relative overhead compared to HRR-DK and HRR-TK is between $0\% - 3\%$ and $3\% - 5\%$ respectively. Moreover, we observe that HRR is a robust algorithm, with minimal history window size having only minor effect on its performance.

Table IV presents similar results for the AWS trace. Note that AWS trace is a more difficult use case for HRR, since we treat requests for VM procurement as "requests". Naturally, these requests are already "optimized" for the VM inventory of AWS. Hence, dynamic requests mapping has smaller effect on the overall performance. Still, because HRR reuses previously reserved servers and there are many situations when more than one VM request can be mapped on the same server (consider these to be physical servers for the sake of discussion), HRR is able to significantly improve results obtained by the PE and E-TK, and SOTA.

VI. Conclusions and Future Work

We presented a novel Heterogeneous Resource Reservation (HRR) problem. Future directions include combining short-term workload patterns prediction with HRR to optimize QoS. In particular, short term prediction allows to power VMs up/down in advance to avoid cold boot, while HRR allows to select optimized VM types and contracts to slash overall operational costs.

REFERENCES

[1] I. Hwang and M. Pedram, "Portfolio theory-based resource assignment in a cloud computing system," in *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*. IEEE, 2012, pp. 582–589.

[2] S. Chaisiri, B. Lee, and D. Niyato, "Optimization of resource provisioning cost in cloud computing," *IEEE Trans. Services Computing*, vol. 5, no. 2, pp. 164–177, 2012. [Online]. Available: http://dx.doi.org/10.1109/TSC.2011.7

[3] O. Biran, D. Breitgand, D. Lorenz, M. Masin, E. Raichstein, and A. Weit, "Heterogeneous Resource Reservation," http://domino.watson.ibm.com/library/CyberDig.nsf/Home, IBM, Tech. Rep. H-0331, 2018.

[4] S. Khatua, P. K. Sur, R. K. Das, and N. Mukherjee, "Heuristic-based optimal resource provisioning in application-centric cloud," *CoRR*, vol. abs/1403.2508, 2014. [Online]. Available: http://arxiv.org/abs/1403.2508

[5] S. Shen, K. Deng, A. Iosup, and D. H. J. Epema, "Scheduling jobs in the cloud using on-demand and reserved instances," in *Euro-Par 2013 Parallel Processing - 19th International Conference, Aachen, Germany, August 26-30, 2013. Proceedings*, ser. Lecture Notes in Computer Science, F. Wolf, B. Mohr, and D. an Mey, Eds., vol. 8097. Springer, 2013, pp. 242–254. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-40047-6_27

[6] W. Wang, B. Liang, and B. Li, "Optimal online multi-instance acquisition in iaas clouds," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 12, pp. 3407–3419, 2015. [Online]. Available: http://dx.doi.org/10.1109/TPDS.2014.2385697

[7] R. Fleischer, "On the bahncard problem," *Theor. Comput. Sci.*, vol. 268, no. 1, pp. 161–174, 2001. [Online]. Available: http://dx.doi.org/10.1016/S0304-3975(00)00266-8

[8] R. Van den Bossche, K. Vanmechelen, and J. Broeckhove, "Iaas reserved contract procurement optimisation with load prediction," *Future Gener. Comput. Syst.*, vol. 53, no. C, pp. 13–24, Dec. 2015. [Online]. Available: http://dx.doi.org/10.1016/j.future.2015.05.016

[9] S. Mistry, A. Bouguettaya, H. Dong, and A. K. Qin, "Predicting dynamic requests behavior in long-term iaas service composition," in *2015 IEEE International Conference on Web Services, ICWS 2015, New York, NY, USA, June 27 - July 2, 2015*, J. A. Miller and H. Zhu, Eds. IEEE Computer Society, 2015, pp. 49–56. [Online]. Available: http://dx.doi.org/10.1109/ICWS.2015.17

[10] R. N. Calheiros, E. Masoumi, R. Ranjan, and R. Buyya, "Workload prediction using ARIMA model and its impact on cloud applications' qos," *IEEE Trans. Cloud Computing*, vol. 3, no. 4, pp. 449–458, 2015. [Online]. Available: http://dx.doi.org/10.1109/TCC.2014.2350475

[11] Y. Jiang, C. Perng, T. Li, and R. N. Chang, "ASAP: A self-adaptive prediction system for instant cloud resource demand provisioning," in *11th IEEE International Conference on Data Mining, ICDM 2011, Vancouver, BC, Canada, December 11-14, 2011*, D. J. Cook, J. Pei, W. Wang, O. R. Zaïane, and X. Wu, Eds. IEEE Computer Society, 2011, pp. 1104–1109. [Online]. Available: http://dx.doi.org/10.1109/ICDM.2011.25

[12] Z. Gong, X. Gu, and J. Wilkes, "PRESS: predictive elastic resource scaling for cloud systems," in *Proceedings of the 6th International Conference on Network and Service Management, CNSM 2010, Niagara Falls, Canada, October 25-29, 2010*. IEEE, 2010, pp. 9–16. [Online]. Available: http://dx.doi.org/10.1109/CNSM.2010.5691343

[13] S. Islam, J. Keung, K. Lee, and A. Liu, "Empirical prediction models for adaptive resource provisioning in the cloud," *Future Generation Comp. Syst.*, vol. 28, no. 1, pp. 155–162, 2012. [Online]. Available: http://dx.doi.org/10.1016/j.future.2011.05.027

[14] E. Caron, F. Desprez, and A. Muresan, "Forecasting for grid and cloud computing on-demand resources based on pattern matching," in *Cloud Computing, Second International Conference, CloudCom 2010, November 30 - December 3, 2010, Indianapolis, Indiana, USA, Proceedings*. IEEE Computer Society, 2010, pp. 456–463. [Online]. Available: http://dx.doi.org/10.1109/CloudCom.2010.65

[15] "HAproxy," http://www.haproxy.org/, 2017.

[16] "NGINX," https://www.nginx.com/resources/wiki/, 2017.

[17] "IBM ILOG CPLEX 12.1," http://www-01.ibm.com/software/integration/optimization/cplex-optimizer, 2017.

[18] J. Wilkes and C. Reiss, "Clusterdata2011_2 traces," 2011.