Correlation-Aware Heuristic Search for Intelligent Virtual Machine Provisioning in Cloud Systems

Chuan Luo,[1] Bo Qiao,[1] Wenqian Xing,[1,2] Xin Chen,[1,2] Pu Zhao,[1] Chao Du,[1] Randolph Yao,[3] Hongyu Zhang,[4] Wei Wu,[5] Shaowei Cai,[6] Bing He,[1] Saravanakumar Rajmohan,[2] Qingwei Lin[1,*]

[1]Microsoft Research, China [2]Microsoft 365, United States [3]Microsoft Azure, United States
[4]The University of Newcastle, Australia [5]L3S Research Center, Leibniz University Hannover, Germany
[6]State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, China
{chuan.luo, boqiao, v-wexin, v-xich15, puzhao, chaodu, ranyao, v-hebi, saravar, qlin}@microsoft.com,
hongyu.zhang@newcastle.edu.au, william.third.wu@gmail.com, caisw@ios.ac.cn

Abstract

The optimization of resource is crucial for the operation of public cloud systems such as Microsoft Azure, as well as servers dedicated to the workloads of large customers such as Microsoft 365. Those optimization tasks often need to take unknown parameters into consideration and can be formulated as Prediction+Optimization problems. This paper proposes a new Prediction+Optimization method named Correlation-Aware Heuristic Search (*CAHS*) that is capable of accounting for the uncertainty in unknown parameters and delivering effective solutions to difficult optimization problems. We apply this method to solving the predictive virtual machine (VM) provisioning (PreVMP) problem, where the VM provisioning plans are optimized based on the predicted demands of different VM types, to ensure rapid provisions upon customers' requests and to pursue high resource utilization. Unlike the current state-of-the-art PreVMP approaches that assume independence among the demands for different VM types, *CAHS* incorporates demand correlation when conducting prediction and optimization in a novel and effective way. Our experiments on two public benchmarks and one industrial benchmark demonstrate that *CAHS* can achieve better performance than its nine state-of-the-art competitors. *CAHS* has been successfully deployed in Microsoft Azure and significantly improved its performance. The main ideas of *CAHS* have also been leveraged to improve the efficiency and the reliability of the cloud services provided by Microsoft 365.

Introduction

Cloud computing has become a popular and powerful computing paradigm. Virtualization is one of the key technologies used in cloud systems. A physical machine (PM) can host multiple virtual machines (VMs) and a cloud system manages a large pool of configurable VM resources (Cortez et al. 2017). For instance, Microsoft Azure provides public cloud services to host VMs from general customers and also maintains dedicated AzSC (Azure on Substrate Compute) servers for the large number of Office workloads owned by Microsoft 365.

[*]Qingwei Lin is the corresponding author of this work.

Copyright © 2021, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

In order to manage large cloud systems effectively, advanced algorithms are needed to guide the creation, scheduling, allocation and maintenance of the large number of VMs. Among all those different tasks, VM provisioning, as a core problem in cloud computing, has attracted considerable attention in both academia and industry (Zhang et al. 2014; Zhao et al. 2018). In practice, a cloud system not only has to provision an enormous quantity of VMs each day, but also needs to provision VMs rapidly upon customers' requests since provisioning VMs from scratch could take a fairly long time (Mao and Humphrey 2012). In order to overcome these challenges, recent literature formulates the above scenario as the predictive VM provisioning (PreVMP) problem, where the future demands are predicted first and then the VM provisioning plan is optimized based on the predicted demands (Luo et al. 2020).

Similar to many other problems arsing from the study of cloud systems such as Microsoft Azure and AzSC, the PreVMP problem can be formulated as a typical prediction and optimization (Prediction+Optimization) problem (Balghiti et al. 2019). Still, as the objective in PreVMP problem depends on unknown parameters (*i.e.,* actual demands), common optimization approaches are inapplicable here (Demirović et al. 2019a). Unfortunately, existing PreVMP approaches (Wilder, Dilkina, and Tambe 2019; Demirović et al. 2019a; Luo et al. 2020) assume that the demands for different VM types are independent, and do not explore demand correlation. This limitation prevents the existing approaches from forming truly optimized VM provisioning plans, since ignoring demand correlation would result in inaccurate modeling of prediction uncertainty and incur performance degradation in optimization.

In this paper, we are devoted to proposing a novel, effective Prediction+Optimization approach called *Correlation-Aware Heuristic Search* (*CAHS*), which leverages the correlation among demands to improve the practical performance. While we focus on the application of *CAHS* on solving PreVMP problem, this approach is applicable to the general Prediction+Optimization problem, including other problems emerging from the intelligent management of cloud systems. The ideas used in this method, such as correlation exploration, uncertainty modeling and heuristic search, have also been incorporated into a number of practical methods for managing Microsoft 365's workloads on the cloud. Those practical methods include the B2 autopilot container allocation algorithm for the load balance (Luo et al. 2020), the AzSC packing algorithm for improving packing efficiency (Luo et al. 2020), and DADRAC (Data Driven Admission Control) for the capacity management.

In the following, we first formulate the PreVMP problem and conduct empirical study on application PreVMP benchmarks. Our investigation shows that the demands of different VM types often exhibit significant correlations; as a result, the limitation of existing state-of-the-art PreVMP approaches would incur performance degradation. To address this serious issue, *CAHS* first partitions all VM types into different groups based on demand correlation; then, for each partitioned group, *CAHS* jointly predicts the future demands for all VM types in this group, and models the prediction uncertainty considering demand correlation; finally, *CAHS* optimizes the provisioning plan based on the predicted demands and the jointly modeled prediction uncertainty. In this way, *CAHS* incorporates demand correlation when conducting prediction and optimization.

Our experiments on two public benchmarks and one industrial benchmark, all of which are collected from industrial public cloud systems, show that *CAHS* achieves much better performance than existing state-of-the-art PreVMP approaches. Our results indicate that *CAHS* can bring benefits in practice. More encouragingly, *CAHS* has been successfully deployed in Microsoft Azure, and the VM provisioning time has been significantly shortened. The main ideas behind *CAHS* have also been applied to managing Microsoft 365's workloads with considerable success.

We summarize our major contributions as follows.

- We provide empirical evidence that the demands of different VM types often exhibit significant correlations.
- In order to solve the PreVMP problem, as well as other Predict+Optimization problems in general, we propose a novel, effective approach called *CAHS*, which explores the correlation among demands for different VM types and incorporates demand correlation when conducting prediction and optimization.
- Our experimental results on two public benchmarks and one industrial benchmark clearly demonstrate the superiority of *CAHS* over current state-of-the-art approaches. More encouragingly, *CAHS* has been deployed in Microsoft Azure and brought significant performance improvement in production.

Problem Definition

In this section, we formally describe the predictive virtual machine provisioning (PreVMP) problem (Luo et al. 2020), and introduce necessary notations used in this paper.

Notations Related to VMs: A *virtual machine (VM) type* $v_i = (\alpha_i, B_i)$ is a pairwise tuple, where α_i denotes the number of *CPU cores* of VM type v_i, and $B_i = \{b_{i,1}, \dots, b_{i,\beta}\}$ ($|B_i| = \beta$) denotes the amount of β resource types (*e.g.,* memory and disk) of VM type v_i. Given a set of n VM types, the *historical demands* for all VM types are represented by $D = \{d_i^t \mid i \in \{1, \dots, n\}, t \in \{1, \dots, T\}\}$, where each d_i^t is a non-negative integer and represents the demand of VM type v_i during the time period $[t-1, t]$. Moreover, $D_i = \{d_i^1, \dots, d_i^T\}$ denotes the historical demands of VM type v_i; and $D^t = \{d_1^t, \dots, d_n^t\}$ refers to the demands of all VM types in the period $[t-1, t]$. Finally, we use notation $Y^* = \{y_1^*, \dots, y_n^*\}$ to denote the *real demand*s for all VM types in the time period $[T, T+1]$, which remains *unknown* till the time stamp $T+1$.

Notations Related to PMs: A *physical machine (PM)* p_j is a pairwise tuple, *i.e.,* $p_j = (C_j, H_j)$, where C_j denotes the number of *CPU core*s of p_j, and $H_j = \{h_{j,1}, \dots, h_{j,\beta}\}$ ($|H_j| = \beta$) denotes the amount of β resource types of p_j. At any time stamp t, the cloud system can provide a collection of m^t PMs, *i.e.,* $P^t = \{p_1^t, \dots, p_m^t\}$, to provision VMs during the time period $[t, t+1]$. For simplicity, notations P, p_j and m represent P^T, p_j^T and m^T (the PM related information queried at last time stamp T), respectively.

Resource Constraints: Since a VM can only be provisioned on a PM with enough resource, for each PM $p_j = (C_j, H_j)$, the total number of CPU cores required by all VMs hosted on p_j cannot exceed C_j; also, for each of β resource types, the total amount of the corresponding resource type required by all VMs hosted on p_j cannot exceed the amount of that resource type of p_j.

Decision Variables: We use notation $A = \{a_{i,j} \mid i \in \{1, \dots, n\}, j \in \{1, \dots, m\}\}$ to denote $n \cdot m$ integer decision variables, and each $a_{i,j}$ represents the number of VMs with VM type v_i to be hosted on PM p_j. Also, notation $X = \{x_i \mid i \in \{1, \dots, n\}\}$ denotes n integer decision variables, and $x_i = \sum_{j=1}^{m} a_{i,j}$ represents the number of VMs with VM type v_i to be provisioned on all PMs.

Provisioning Plan: A *provisioning plan* is a complete assignment to A. Given a provisioning plan, it is *valid* if all resource constraints are satisfied under this provisioning plan; otherwise it is *invalid*.

Objective: It is well recognized that optimizing VM provisioning at CPU core level can lead to good performance for cloud systems in practice (Mann 2016; Zhao et al. 2018; Luo et al. 2020). We follow this standard practice and set the objective in this work as optimizing CPU core utilization ratio, which is defined as the ratio between the total number of CPU cores utilized by all VMs and the total number of CPU cores in all physical machines. In our notations, the *number of utilized CPU cores* for VM type v_i is $u_i = \min\{x_i, y_i^*\} \cdot \alpha_i$ and the *total number of utilized CPU cores* for all VM types is $u = \sum_{i=1}^{n} u_i$. Hence, following the above notations, the *CPU core utilization ratio* can be represented as $r = u / (\sum_{j=1}^{m} C_j)$.

The PreVMP Problem: At time stamp T, given a set of n VM types V, historical demands D, and a set of m PMs, the **Predictive Virtual Machine Provisioning** (PreVMP) problem aims to find a valid provisioning plan so that the highest CPU core utilization ratio r can be achieved. As the number of total CPU cores in all physical machines (*i.e.,* $\sum_{j=1}^{m} C_j$) can be regarded as a constant, maximizing r is equivalent to maximizing the total number of utilized CPU cores for all VM types u. Hence, the PreVMP problem is formally described as below.

$$\begin{aligned} \text{maximize} \quad & u = \textstyle\sum_{i=1}^{n} \min\{x_i, y_i^*\} \cdot \alpha_i \\ \text{s.t.} \quad & \textstyle\sum_{i=1}^{n} a_{i,j} \cdot \alpha_i \leqslant C_j, & j \in J \\ & \textstyle\sum_{i=1}^{n} a_{i,j} \cdot b_{i,k} \leqslant h_{j,k}, & (j,k) \in J \times K \\ & a_{i,j} \geqslant 0, & (i,j) \in I \times J \\ & a_{i,j} \text{ is an integer}, & (i,j) \in I \times J \end{aligned} \tag{1}$$

where $I = \{1, \dots, n\}$, $J = \{1, \dots, m\}$ and $K = \{1, \dots, \beta\}$ (recalling that $x_i = \sum_{j=1}^{m} a_{i,j}$). We would like to note that, at time stamp T, the real demands of all VM types during the following time period $[T, T+1]$ (*i.e.,*, $\{y_i^* \mid i \in \{1, \dots, n\}\}$) are unknown.

Actually, the PreVMP problem is a very challenging problem (Luo et al. 2020); even if all real demands during the time period $[T, T+1]$ (*i.e.,*, $\{y_i^* \mid i \in \{1, \dots, n\}\}$) were known, the problem is still NP-hard (Hbaieb, Khemakhem, and Jemaa 2017; Zhao et al. 2018; Luo et al. 2020).

Related Work

When all the real demands $\{y_i^* \mid 1 \leqslant i \leqslant n\}$ are known, the PreVMP formulated in Equation 1 reduces to the classic VM provisioning (VMP) problem. While the classic VMP problem has been well studied (Mann 2016; Hbaieb, Khemakhem, and Jemaa 2017; Zhao et al. 2018; Liu et al. 2018), approaches for solving classic VMP problem cannot handle the scenario where real demands are unknown.

As discussed before, the PreVMP problem is a typical Prediction+Optimization problem (Balghiti et al. 2019). A common solution is the two-stage approach, which first predicts the real demands based on historical demands, and then optimizes the provisioning plan by treating predicted demands as real ones (Demirović et al. 2019a). Recently, there is a growing body of work on solving specific scenarios of the Prediction+Optimization problem, including the cases when the optimization objective is linear (Elmachtoub and Grigas 2017; Mandi et al. 2020), when the optimization problem is a ranking problem (Demirović et al. 2019b), and when the optimization problem can be solved by dynamic programming (Demirović et al. 2020). However, due to the form of the optimization objective in the PreVMP problem, these new approaches are not applicable. More importantly, as prediction errors are unavoidable (Wilder, Dilkina, and Tambe 2019), the two-stage method would exhibit inferior performance for solving PreVMP (as can be observed in our experiments). Also, robust optimization (Bertsimas, Gupta, and Kallus 2018) and stochastic optimization (Spall 2003) cannot be applied to the Prediction+Optimization problem, since they do not have a clear way to use feature data and predict unknown parameters in optimization objective from data (Elmachtoub and Grigas 2017; Demirović et al. 2019a).

In the literature, current state-of-the-art approaches for solving the PreVMP problem are *Semi-direct* (Demirović et al. 2019a), *NN-Decision* (Wilder, Dilkina, and Tambe 2019) and *UAHS* (Luo et al. 2020). However, these approaches assume that the demands for different VM types are independent and do not explore correlation among demands. As we will show later, this limitation would incur performance degradation.

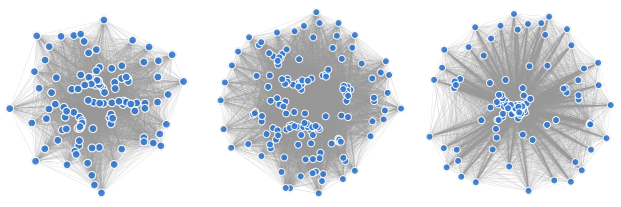

Figure 1: Visualizations to demonstrate the VM demand correlation for the three benchmarks.

Empirical Study on VM Demand Correlation

Since our *CAHS* approach leverages the VM demand correlation for optimizing VM provisioning plans, here we conduct empirical study to investigate whether the correlation appears frequently or sporadically among the demands of different VM types in application scenarios.

For the empirical study, we adopt two public benchmarks (*i.e.,* `Azure-2017` and `Azure-2019`) and one industrial benchmark (*i.e.,* `INDU-2020`).[1] It is well recognized that Pearson's correlation coefficient (PCC) (Rodgers and Nicewander 1988) is an effective metric to assess the correlation, and has been adopted in many fields (Xiong et al. 2004). Hence, in this paper, we adopt PCC as the metric to measure demand correlation. Given two different VM types v_i and v_j, as well as their historical demands D_i and D_j, the demand correlation between v_i and v_j, denoted by $\rho(v_i, v_j)$, can be calculated as the PCC between D_i and D_j, *i.e.,* $\rho(v_i, v_j) = \frac{cov(D_i, D_j)}{\sigma(D_i) \cdot \sigma(D_j)}$ $(-1 \leqslant \rho(v_i, v_j) \leqslant 1)$, where $cov(\cdot, \cdot)$ and $\sigma(\cdot)$ denote covariance and standard deviation, respectively.

We visualize the demand correlations among all VM types for the `Azure-2017`, `Azure-2019` and `INDU-2020` benchmarks in Figure 1, where we adopt $\lambda = \sqrt{1 - \rho^2}$ as our distance metric (Innocenti and Materassi 2008). In Figure 1, each point represents a VM type; for two different VM types v_i and v_j, the smaller the distance (*i.e.,* the $\lambda(v_i, v_j)$ value), the stronger the demand correlation between v_i and v_j. As can be seen from Figure 1, it is apparent that the demand correlations are strong among a large number of VM types in all application benchmarks.

In this empirical study, given two VM types v_i and v_j, we say that the pair (v_i, v_j) is *significantly correlated* if $v_i \neq v_j$ and $|\rho(v_i, v_j)| > 0.5$ and the correlation is statistically significant at a significance level of 0.05. For each VM type v_i, we use $\eta(v_i)$ to denote the number of significantly correlated pairs where v_i appears. Our statistics show that the values of the averaged η across all VM types for the `Azure-2017`, `Azure-2019` and `INDU-2020` benchmarks are 6.7, 6.4 and 45.4, respectively, which further confirms that significant demand correlation among VM types can be frequently observed in all application benchmarks.

[1]The `Azure-2017`, `Azure-2019` and `INDU-2020` benchmarks, which are all collected from industrial public cloud systems, are introduced in the Experiments section.

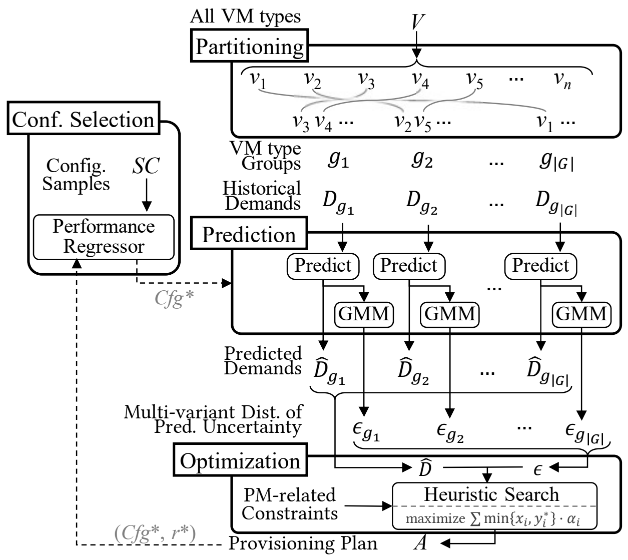
All VM types
V
Partitioning
v_1 v_2 v_3 v_4 v_5 ··· v_n
v_3 v_4 ··· v_2 v_5 ··· v_1 ···
Conf. Selection
Config. Samples
SC
VM type Groups
g_1 g_2 ··· $g_{|G|}$
Historical Demands
D_{g_1} D_{g_2} ··· $D_{g_{|G|}}$
Performance Regressor
Prediction
Predict
GMM
Cfg^*
Predicted Demands
$\widehat{D}_{g_1}$ $\widehat{D}_{g_2}$ ··· $\widehat{D}_{g_{|G|}}$
Multi-variant Dist. of Pred. Uncertainty
ϵ_{g_1} ϵ_{g_2} ··· $\epsilon_{g_{|G|}}$
Optimization
$\widehat{D}$
ϵ
PM-related Constraints
Heuristic Search
maximize $\sum \min\{x_i, y_i^*\} \cdot \alpha_i$
(Cfg^*, r^*)
Provisioning Plan
A

Figure 2: High-level framework of *CAHS*. The iteration process is terminated once the limit κ is reached.

There are many possible reasons for the existence of demand correlation. For example, we observe from our data that many requests are submitted to deploy web services. To accomplish a web service deployment request, many VMs with different VM types are required for front-end, business logic, database, *etc.* As another example, a version upgrade (*e.g.,* the release of a new operating system version) could also lead to the correlations in VM demands – more users would request those VM types with the new operating system version, and meanwhile less users would request those VM types with the old operating system version.

Correlation-Aware Heuristic Search

This section presents *Correlation-Aware Heuristic Search* (*CAHS*), a novel approach for solving the PreVMP problem.

Framework

The main idea of *CAHS* is to explore and incorporate the VM demand correlation when conducting prediction and optimization. Inspired by the success of the iterative framework underlying *UAHS* for solving PreVMP (Luo et al. 2020), the high-level framework of our *CAHS* approach is also an iterative one. There are four critical phases in our *CAHS* approach: 1) partitioning phase; 2) configuration selection phase; 3) prediction phase; 4) optimization phase. The high-level framework of *CAHS* is illustrated in Figure 2. According to Figure 2, *CAHS* conducts the configuration selection, prediction and optimization phases in an iterative manner.

Partitioning Phase: This phase partitions all VM types into different VM type groups, based on the demand correlation. This phase ensures the sufficient strength of correlations among demands of the VM types in the same group.

Configuration Selection Phase: This phase selects an effective configuration Cfg^* for the prediction phase, based on

Algorithm 1: VM Type Partitioning

```
Input: $V$: a set of all VM types;
Output: $G$: the set of partitioned VM type groups;
1  $G \leftarrow \varnothing, R \leftarrow V$;
2  while $|R| > 1$ do
3      $v_i, v_j \leftarrow \underset{v_i \in R, v_j \in R\ (v_i \neq v_j)}{\arg\max} |\rho(v_i, v_j)|$;
4      $g \leftarrow \{v_i\}, R \leftarrow R \backslash \{v_i\}$;
5      sort $R$ by the $|\rho|$ value w.r.t. $v_i$ in a descending order;
6      foreach $v_j \in R$ do
7          $p \leftarrow \min\{|\rho(v_j, w)| \mid w \in g\}$;
8          if $p > \tau$ then $g \leftarrow g \cup \{v_j\}, R \leftarrow R \backslash \{v_j\}$;
9      $G \leftarrow G \cup \{g\}$;
10 if $|R| == 1$ then $G \leftarrow G \cup \{R\}$ ;
11 return $G$;
```

its estimated performance through a neural network based performance regressor.

Prediction Phase: In this phase, for each partitioned VM type group g_s, *CAHS* jointly predicts the future demands for all VM types in g_s, and models the prediction uncertainty as a joint distribution by considering demand correlation.

Optimization Phase: For each time period $[t-1, t]$, this phase takes all VM types' predicted demands during $[t-1, t]$ and all VM type groups' prediction uncertainties as inputs, and conducts optimization to produce an effective provisioning plan subject to resource constraints regarding the PM status queried at time stamp $t-1$.

In each iteration, after the provisioning plans for all time periods are produced, the average utilization ratio r^* across all produced provisioning plans and the chosen configuration Cfg^* are combined as a new sample, *i.e.,* (r^*, Cfg^*), to iteratively enhance the performance regressor in the configuration selection phase. Through this way, the configuration selection, prediction and optimization phases form an effective feedback loop, which can lead to significant performance improvement. For the high-level framework, the iteration process is terminated once the iteration limit κ (which is a hyper-parameter) is reached.

Partitioning Phase

As demonstrated in our empirical study, different VM types often form a cluster where all pairwise demand correlations are relatively strong. This observation in our empirical study motivates us to propose effective methods for partitioning all VM types into different groups, based on demand correlation. As a result, the core problem in the partitioning phase is how to effectively identify a group of VM types with strong demand correlation.

Given a group g containing multiple VM types, we will say all VM types in g are correlated if the PCC metric $|\rho|$ for all pairs of VM types in group g are greater than a threshold τ (which is a hyper-parameter).

Here we propose a new, greedy VM type partitioning algorithm, which is outlined in Algorithm 1. In the beginning, the group set G is initialized as an empty set, and the set of remaining VM types R is initialized as the set of all VM

Algorithm 2: Configuration Selection

```
Input: Reg: an NN-based performance regressor;
Output: Cfg*: the selected configuration;
1 $Cfg^* \leftarrow$ a randomly sampled configuration;
2 $Perf^* \leftarrow$ the performance of $Cfg^*$ estimated by $Reg$;
3 $SC \leftarrow$ a set of randomly sampled configurations;
4 foreach $Cfg \in SC$ do
5     $Perf \leftarrow$ the performance of $Cfg$ estimated by $Reg$;
6     if $Perf^*$ is worse than $Perf$ then
7         $Cfg^* \leftarrow Cfg$, $Perf^* \leftarrow Perf$;
8 return $Cfg^*$;
```

types V (Line 1 in Algorithm 1). Then the algorithm works in an iterative manner. In each iteration, a group of correlated VM types satisfying our proposed criterion, denoted by g, is identified, and those VM types in g are removed from R (Lines 3–9 in Algorithm 1). The iteration process is terminated once R contains no more than one VM type (Line 2 in Algorithm 1).

Our partitioning algorithm splits all VM types into a set of groups, and guarantees that, for each group g containing more than one VM type, all VM types in g are correlated.

Configuration Selection Phase

The role of the configuration selection phase is to provide a configuration for the prediction phase. Thus, the core problem in this phase is how to select an effective configuration.

To address this challenge, it is necessary to develop a model that can estimate the performance of any given configuration. This model would then help us choose the most promising configuration for the next phase. Due to the fact that neural network (NN) has strong learning ability and can approximate any function (Hastie, Friedman, and Tibshirani 2001), we adopt an NN-based regressor for this task.

Furthermore, we propose a new algorithm for configuration selection (outlined in Algorithm 2). Following an effective sampling method called Best from Multiple Selections (BMS) (Cai 2015; Cai, Lin, and Luo 2017), our algorithm first randomly samples a set of configurations; then, from the sampled configuration set, the algorithm selects the configuration *Cfg** with the best performance, estimated by the NN-based regressor; finally the selected configuration is output as the one recommended to the prediction phase.

Our algorithm can work with any NN-based regressor. In this paper, to keep the training time at a minimum, we use multi-layer perceptron (MLP) (Hastie, Friedman, and Tibshirani 2001) to estimate configuration performance.

Prediction Phase

Within each group g_s, the VM types' demands are correlated. Hence, the core problem in the prediction phase is how to predict the demands for all VM types in g_s and to model the prediction uncertainty while considering the correlation.

Here we introduce several necessary notations regarding VM type groups. For a VM type group g_s, we use notation D_{g_s} to denote the matrix of historical demands for group g_s ($D_{g_s} = \{d^t_{s_i} \mid v_{s_i} \in g_s, t \in \{1, \ldots, T\}\}$). Further, for

Algorithm 3: Prediction and Uncertainty Modeling

```
Input: Cfg*: a selected configuration;
       $D_{g_s}$: historical demands for VM type group $g_s$;
Output: $\hat{D}_{g_s}$: predicted demand data for group $g_s$;
        $\epsilon_{g_s}$: the multi-variant distribution of prediction
        uncertainty for group $g_s$;
1 specify $l$ and predictor's parameters by $Cfg^*$;
2 for $t \leftarrow 1$ to $T - l + 1$ do
3     $\hat{D}^{t+l-1}_{g_s} \leftarrow$ the demands for $g_s$ at $t + l - 1$ forecast by
      predictor trained on $\Gamma^t_l(D_{g_s}) \backslash \{D^{t+l-1}_{g_s}\}$;
4     $E^{t+l-1}_{g_s} \leftarrow D^{t+l-1}_{g_s} - \hat{D}^{t+l-1}_{g_s}$;
5 $\hat{Y}_{g_s} \leftarrow$ the real demands for $g_s$ (at time stamp $T + 1$)
  forecast by predictor trained on $\Gamma^{T-l+2}_{l-1}(D_{g_s})$;
6 $\hat{D}_{g_s} \leftarrow \{\hat{D}^l_{g_s}, \hat{D}^{l+1}_{g_s}, \ldots, \hat{D}^T_{g_s}, \hat{Y}_{g_s}\}$;
7 $\epsilon_{g_s} \leftarrow$ fit multi-variant GMM based on $\{E^l_{g_s}, \ldots, E^T_{g_s}\}$;
8 return $\hat{D}_{g_s}$, $\epsilon_{g_s}$;
```

group g_s and time stamp t, we use $D^t_{g_s}$ to denote the demands for all VM types in g_s during the time period $[t-1, t]$ ($D^t_{g_s}$ is a vector). In addition, for group g_s, we use $\Gamma^t_l(D_{g_s})$ to denote a submatrix of D_{g_s}, with the first time stamp of t and the length of l ($\Gamma^t_l(D_{g_s}) = \{D^t_{g_s}, D^{t+1}_{g_s}, \ldots, D^{t+l-1}_{g_s}\}$).

To address the challenge in the prediction phase, compared to previous approaches that assume the demand of each VM type is independent, we propose a novel prediction and uncertainty modeling algorithm which is capable of exploiting demand correlation. Our new proposed algorithm is listed in Algorithm 3. For each VM type group g_s, *CAHS* first applies the sliding window based method (Luo et al. 2020) to split the group demand D_{g_s} into a set of sequential matrices, *i.e.*, $\{\Gamma^t_l(D_{g_s}) \mid 1 \leqslant t \leqslant T-l+1\} \cup \Gamma^{T-l+2}_{l-1}(D_{g_s})$, where l is a configurable parameter and can be specified by *Cfg**. For each sequential matrix $\Gamma^t_l(D_{g_s})$, *CAHS* considers the last demand vector $D^{t+l-1}_{g_s}$ as the label vector, and then forecasts $\hat{D}^{t+l-1}_{g_s}$, through a *predictor* trained on $\Gamma^t_l(D_{g_s}) \backslash \{D^{t+l-1}_{g_s}\}$, and the prediction error vector for each sequential matrix can be obtained.

For each group g_s, once the prediction error vectors for all sequential matrices are computed, we need to model the prediction uncertainty for group g_s. Previous approaches (Luo et al. 2020) use single-variant distributions to model the prediction uncertainty, and cannot handle the correlated case. In the context of PreVMP, for each group g_s, we are the first to model the prediction uncertainty ϵ_{g_s} by fitting a multi-variant Gaussian Mixture Model (GMM) (Adams and Beling 2019), which can well capture the correlation among multiple variants (Verbeek, Vlassis, and Kröse 2003). Besides, the *predictor* underlying *CAHS* is the Unobserved Component Model (UCM) (Durbin and Koopman 2012), since it is recognized that UCM can forecast multi-variant time series both efficiently and effectively (Young 2011).

Optimization Phase

For each time period $[t - 1, t]$, this phase solves a corresponding VMP problem subject to the resource constraints regarding the PM status queried at time stamp $t-1$. If we directly solve the original combinatorial optimization problem in Equation 1 by simply replacing the unknown demands with the predicted demands, this would incur performance degradation due to ignoring prediction uncertainty. Therefore, the core problem in this phase is how to find an effective way to incorporate the prediction uncertainty into optimization.

For a given group g_s, we treat the prediction uncertainty θ_{s_i} for VM type $v_{s_i} \in g_s$ as a random variable, and model all prediction uncertainties regarding group g_s (*i.e.,* $\theta_{s_1}, \theta_{s_2}, \ldots, \theta_{s_{|g_s|}}$) with the joint distribution ϵ_{g_s}. Then the (unknown) real demand $y^*_{s_i} = \hat{d}^t_{s_i} + \theta_{s_i}$ can be treated as a random variable. From Equation 1, the optimization objective $u = \sum_{i=1}^{n} \min\{x_i, y^*_i\} \cdot \alpha_i$ is a random variable depending on $y^*_1, \ldots, y^*_n$. Hence, an effective way to incorporate prediction uncertainty into optimization is to maximize the mathematical expectation of objective u (*i.e.,* $\mathbb{E}(u)$).

As discussed in the Problem Definition section, the optimization problem in our context is an NP-hard problem. To address this challenge, we design a new heuristic search algorithm, since heuristic search is known to show effectiveness in solving NP-hard combinatorial optimization problems (Karp 2011; Cai and Su 2013; Luo et al. 2015b,a, 2019). For heuristic search algorithms, the most critical component is the scoring function (Luo et al. 2014; Cai et al. 2013; Luo et al. 2017), which is designed to evaluate the potential of each candidate variable.

Specifically, given a provisioning plan A and its corresponding objective u, if we provision a new VM with VM type v_i, resulting in a new objective u', the *score* of v_i is the increment to the objective *i.e.,* $\mathbb{E}(u') - \mathbb{E}(u)$. In this paper, we propose a novel, efficient scoring function in Lemma 1, which can well capture demand correlation.[2]

Lemma 1. *Given a provisioning plan A and VM type v_i, the* score *of v_i can be calculated as:*

$$\begin{aligned}
&\mathit{score}(v_i) = \alpha_i \cdot (1 - \mathit{CDF}_{y^*_i|g_s\backslash\{v_i\}}(x_i)) \\
&\quad + \Sigma_{j\neq i}(\alpha_j \cdot \mathbb{E}_{y^*_j|g_s\backslash\{v_i,v_j\}}(y_j \leq x_j \mid y^*_i = x_i + 1)) \\
&\quad - \Sigma_{j\neq i}(\alpha_j \cdot \mathbb{E}_{y^*_j|g_s\backslash\{v_i,v_j\}}(y_j \leq x_j \mid y^*_i = x_i)) \\
&\quad + \Sigma_{j\neq i}(\alpha_j \cdot x_j \cdot \mathit{CDF}_{y^*_j|g_s\backslash\{v_i,v_j\}}(x_j \mid y^*_i = x_i)) \\
&\quad - \Sigma_{j\neq i}(\alpha_j \cdot x_j \cdot \mathit{CDF}_{y^*_j|g_s\backslash\{v_i,v_j\}}(x_j \mid y^*_i = x_i + 1)).
\end{aligned}$$

Actually, the computation of *score* is efficient, because it is known that conditional distributions can be simulated very fast (Cong, Chen, and Zhou 2017), and the computational complexity for remaining calculations is $O(|g_s|)$.

Based on the new scoring function, we can design a new, heuristic search algorithm for optimization. Our algorithm first initializes an empty provisioning plan (*i.e.,* setting each $a_{i,j} \in A$ to 0), and then works in an iterative manner: in each iteration, the VM type v_i with the greatest *score* is selected, and a PM p_j is randomly chosen; then *CAHS* assigns one VM with VM type v_i to PM p_j (*i.e.,* $a_{i,j} \leftarrow a_{i,j} + 1$). The iteration process is terminated once there is no VM type with positive *score*. Finally, our algorithm reports the provisioning plan A as its output.

[2]For simplicity, we utilize term 'CDF' to denote cumulative distribution function, and use term '$y^*_j \mid g_s\backslash\{v_i, v_j\}$' to denote a random variable y^*_j. In particular, y^*_j is with a bi-variant distribution $\chi(v_i, v_j)$, and $\chi(v_i, v_j)$ is derived from a multi-variant distribution conditioned on $\{y^*_k = x_k \mid v_k \in g_s\ (v_k \neq v_i, v_k \neq v_j)\}$.

Discussions

Discussion on the Differences Between *CAHS* and *UAHS*: The major differences between *CAHS* and *UAHS* are listed as follows: a) *CAHS* explores demand correlation and partitions VM types into different groups based on demand correlation, while *UAHS* does not consider demand correlation; b) For prediction, *CAHS* jointly predicts the future demands for those VM types in a group and models the prediction uncertainty as a joint distribution, while *UAHS* treats each VM type individually (*UAHS* predicts the future demand and models the prediction uncertainty as a single-variant distribution for each VM type independently). c) For optimization, *CAHS* optimizes the provisioning plan based on a novel, effective scoring function (Lemma 1), while *UAHS* optimizes the provisioning plan using a simple, greedy method.

Discussion on the General Applicability of *CAHS*: It is worthy noting that *CAHS* can be easily adapted for solving the general Prediction+Optimization problem. This method is applicable to other Prediction+Optimization problems as long as the following two conditions hold: a) Historical samples of the unknown parameters are available to train the predictor in the prediction phase; b) The scoring function representing the increment to the objective function during heuristic search is available and can be easily computed based on the simulated conditional distributions in the optimization phase. The high-level framework of *CAHS* (*i.e.,* the iteration process involving the partitioning, configuration selection, prediction and optimization phases) can also be easily modified to incorporate any probability model of unknown parameters and any heuristic search algorithm for solving the general Prediction+Optimization problem.

Experiments

In this section, in order to evaluate the effectiveness of *CAHS*, we first perform thorough experiments on two public benchmarks and one industrial benchmark to compare *CAHS* against nine state-of-the-art approaches. Then we conduct more empirical evaluations to analyze the effects of our proposed new algorithmic ideas underlying *CAHS*.

Benchmarks

In the context of PreVMP, two public PreVMP benchmarks[3] (Cortez et al. 2017), *i.e.,* `Azure-2017` and `Azure-2019`, are commonly used to evaluate the performance of PreVMP approaches (Luo et al. 2020). Both public PreVMP benchmarks are collected from Microsoft Azure. The `Azure-2017` benchmark contains 110 VM types and summarizes VM provisioning workloads of Microsoft

[3]https://github.com/Azure/AzurePublicDataset

Approach	Azure-2017 avg. r ± SD time (sec)	Azure-2019 avg. r ± SD time (sec)	INDU-2020 avg. r ± SD time (sec)
LR	0.684 ± 0.187 1089.8	0.697 ± 0.135 4849.6	0.714 ± 0.050 3166.1
TSDec	0.756 ± 0.084 1051.7	0.721 ± 0.123 4137.1	0.739 ± 0.036 3252.4
AutoARIMA	0.769 ± 0.140 2229.5	0.760 ± 0.122 4408.9	0.749 ± 0.042 8212.2
LSTM	0.757 ± 0.084 3434.2	0.711 ± 0.090 9054.5	0.754 ± 0.054 14067.4
UCM	0.784 ± 0.122 1386.7	0.767 ± 0.106 3226.7	0.760 ± 0.038 4254.3
Prophet	0.762 ± 0.141 800.2	0.704 ± 0.142 3071.3	0.725 ± 0.054 2293.9
Semi	0.790 ± 0.120 1030.6	0.781 ± 0.084 4174.1	0.751 ± 0.039 3649.8
NN	0.794 ± 0.116 2722.1	0.791 ± 0.097 4818.0	0.772 ± 0.043 1891.5
UAHS	0.818 ± 0.103 255.1	0.816 ± 0.102 394.8	0.803 ± 0.053 507.2
CAHS	**0.871 ± 0.064** 284.5	**0.851 ± 0.033** 450.6	**0.872 ± 0.032** 584.2

Table 1: Results of *CAHS* and its state-of-the-art competitors on all benchmarks. We use '*LR*', '*TSDec*', '*AutoARIMA*', '*LSTM*', '*UCM*', '*Prophet*', '*Semi*' and '*NN*' to denote '*LR+ACO*', '*TSDec+ACO*', '*AutoARIMA+ACO*', '*LSTM+ACO*', '*UCM+ACO*', '*Prophet+ACO*', '*Semi-direct*' and '*NN-Decision*', respectively.

Approach	Azure-2017 avg. r ± SD time (sec)	Azure-2019 avg. r ± SD time (sec)	INDU-2020 avg. r ± SD time (sec)
CAHS-alt1	0.822 ± 0.094 261.9	0.820 ± 0.084 412.6	0.820 ± 0.047 541.9
CAHS-alt2	0.839 ± 0.087 283.4	0.826 ± 0.091 444.1	0.837 ± 0.047 564.6
CAHS-alt3	0.842 ± 0.063 461.1	0.824 ± 0.051 678.9	0.853 ± 0.060 1033.8
CAHS-alt4	0.847 ± 0.051 344.7	0.827 ± 0.047 544.6	0.863 ± 0.037 998.9
CAHS-alt5	0.844 ± 0.077 351.1	0.834 ± 0.038 521.5	0.853 ± 0.055 789.5
CAHS	**0.871 ± 0.064** 284.5	**0.851 ± 0.033** 450.6	**0.872 ± 0.032** 584.2

Table 2: Results of *CAHS*, *CAHS-alt1*, *CAHS-alt2*, *CAHS-alt3*, *CAHS-alt4* and *CAHS-alt5* on all benchmarks.

Azure across 30 days in 2017. Similarly, the Azure-2019 benchmark includes 150 VM types and contains VM provisioning workloads of Microsoft Azure across 30 days in 2019. For each of the Azure-2017 and Azure-2019 benchmarks, the demand for each VM type is recorded every 4 hours, so it has 180 time stamps across 30 days. Each of the Azure-2017 and Azure-2019 benchmarks consists of 42 PreVMP benchmarking instances.

Also, we encode and use an industrial benchmark called INDU-2020, which is gathered from Microsoft Azure. The INDU-2020 benchmark has 154 VM types and records VM provisioning workloads in 60 days in 2020; it has 360 time stamps across 60 days. To align with those two public benchmarks, the INDU-2020 benchmark is processed to contain 42 PreVMP benchmarking instances.

Competitors

Following the evaluation setup adopted in the literature (Luo et al. 2020), we compare *CAHS* against a baseline comprised of 9 state-of-the-art PreVMP competitors, including 6 two-stage methods, *Semi-direct* (Demirović et al. 2019a), *NN-Decision* (Wilder, Dilkina, and Tambe 2019) and *UAHS* (Luo et al. 2020). Those 6 two-stage methods are *LR+ACO*, *TSDec+ACO*, *AutoARIMA+ACO*, *LSTM+ACO*, *UCM+ACO* and *Prophet+ACO*, which integrate 6 representative, effective prediction methods, *i.e.,* linear regression (LR) (Luo et al. 2020), time series decomposition based forecasting approach (TSDec) (Luo et al. 2020), automatic autoregressive integrated moving average (AutoARIMA) (Hyndman and Khandakar 2008), long short-term memory (LSTM) (Luo et al. 2019), unobserved component model (UCM) (Durbin and Koopman 2012) and Prophet (Taylor and Letham 2018), with the ACO (Ant Colony Optimization) algorithm (Zhao et al. 2018) (which is the current state of the art in solving the classic VMP problem), respectively. *Semi-direct*, *NN-Decision* and *UAHS* are three state-of-the-art approaches for solving PreVMP (Luo et al. 2020).

Experimental Setup

All the experiments in this paper were performed on a computing server with 2.50GHz Intel Xeon E7-8890 v3 CPU and 1.0TB memory, running the operating system of Ubuntu 18.04. For our *CAHS* approach, the hyper-parameters κ and τ are set to 50 and 0.8, respectively; the effects of different hyper-parameter settings are discussed later. For each approach for solving each benchmark, we report the average CPU core utilization ratio ('avg. r'), the standard deviation ('SD') and the average runtime ('time') in seconds. The experimental results highlighted in **boldface** indicate the best performance for each benchmark.

Experimental Results

Comparing *CAHS* with Competitors: Table 1 reports the comparative results of *CAHS* and its 9 state-of-the-art competitors on all benchmarks. As can be clearly seen from Table 1, *CAHS* can achieve much better utilization ratio than all its competitors. In particular, the improvement of *CAHS* on average utilization ratio over the second best approach *UAHS* is 0.053, 0.035 and 0.069 on the Azure-2017, Azure-2019 and INDU-2020 benchmarks, respectively. Since those benchmarks are all gathered from industrial public cloud systems, the significant improvement on computing resource utilization achieved by *CAHS* indicates that *CAHS* is able to greatly decrease the hardware resources required by predictive VM provisioning, and in turn can considerably reduce operational expenses.

Effect of Incorporating Demand Correlation: Based on *CAHS*, we develop its two alternative versions *CAHS-alt1* and *CAHS-alt2*: *CAHS-alt1* does not consider demand correlation in both prediction and optimization phases, while

κ	Azure-2017	Azure-2019	INDU-2020
	avg. r ± SD time (sec)	avg. r ± SD time (sec)	avg. r ± SD time (sec)
1	0.837 ± 0.081	0.802 ± 0.094	0.833 ± 0.050
	4.0	7.9	12.4
10	0.851 ± 0.061	0.832 ± 0.045	0.850 ± 0.040
	45.8	59.3	118.8
25	0.861 ± 0.060	0.842 ± 0.039	0.859 ± 0.038
	121.8	144.0	300.3
50	0.871 ± 0.064	0.851 ± 0.033	0.872 ± 0.032
	284.5	450.6	584.2
100	0.877 ± 0.048	0.854 ± 0.033	0.872 ± 0.038
	475.3	754.2	1249.9
150	0.881 ± 0.047	0.855 ± 0.035	**0.873 ± 0.033**
	727.8	995.7	1953.0
200	**0.883 ± 0.047**	**0.858 ± 0.036**	0.872 ± 0.039
	961.6	1393.9	2734.1

Table 3: Results of *CAHS* with different hyper-parameter settings of κ on all benchmarks.

CAHS-alt2 only considers demand correlation in its prediction phase. The results of comparing *CAHS* with *CAHS-alt1* and *CAHS-alt2* are presented in Table 2. From Table 2, *CAHS* clearly outperforms *CAHS-alt1* and *CAHS-alt2* on all benchmarks, which confirms that incorporating demand correlation can lead to significant performance improvement.

Effect of VM Type Partitioning: To show the effectiveness of our algorithm for VM type partitioning, based on *CAHS*, we replace our partitioning algorithm with three well-known clustering algorithms (*i.e., k-means* (Lloyd 1982), *Spectral Clustering* (Shi and Malik 2000) and *DBSCAN* (Ester et al. 1996)), resulting in *CAHS-alt3*, *CAHS-alt4* and *CAHS-alt5*, respectively. The experimental results of *CAHS*, *CAHS-alt3*, *CAHS-alt4* and *CAHS-alt5* are shown in Table 2. According to Table 2, *CAHS* achieves better performance than *CAHS-alt3*, *CAHS-alt4* and *CAHS-alt5* on all benchmarks, which indicates the effectiveness of our partitioning algorithm.

Effects of Hyper-Parameter Settings: Since *CAHS* introduces two hyper-parameters κ and τ, the experimental results of *CAHS* with different hyper-parameter settings of κ and τ are presented in Tables 3 and 4, respectively. Table 3 presents that *CAHS* can achieve better performance with larger κ, and Table 4 demonstrates that *CAHS* can perform better when τ is around 0.8.

Applications in Practice

In this section, we present the applications of *CAHS* in practice. In particular, we first describe the success story in Microsoft Azure, and then introduce the applications in Microsoft 365.

Success Story in Microsoft Azure

More encouragingly, our *CAHS* approach has been successfully applied to the Pre-Provisioning Service (PPS) (Luo et al. 2020) in Microsoft Azure. For PPS, demand predictions are obtained from *CAHS* through an intermediary service system called Resource Central (Cortez et al. 2017), and *CAHS* has significantly improved the performance of VM

τ	Azure-2017	Azure-2019	INDU-2020
	avg. r ± SD time (sec)	avg. r ± SD time (sec)	avg. r ± SD time (sec)
0	0.825 ± 0.099	0.818 ± 0.057	0.829 ± 0.064
	936.4	1669.0	2210.5
0.1	0.847 ± 0.065	0.825 ± 0.049	0.853 ± 0.064
	478.3	824.6	1025.5
0.2	0.853 ± 0.061	0.821 ± 0.057	0.855 ± 0.064
	464.3	738.6	912.5
0.3	0.841 ± 0.074	0.825 ± 0.050	0.862 ± 0.065
	446.5	695.5	849.5
0.4	0.846 ± 0.067	0.826 ± 0.047	0.865 ± 0.062
	430.6	634.9	771.6
0.5	0.846 ± 0.062	0.828 ± 0.051	0.868 ± 0.034
	411.3	594.3	720.6
0.6	0.848 ± 0.076	0.836 ± 0.039	**0.873 ± 0.030**
	379.1	551.4	653.5
0.7	0.855 ± 0.065	0.842 ± 0.034	0.871 ± 0.039
	331.3	488.2	614.8
0.8	**0.871 ± 0.064**	**0.851 ± 0.033**	0.872 ± 0.032
	284.5	450.6	584.2
0.9	0.853 ± 0.077	0.845 ± 0.041	0.854 ± 0.053
	275.0	428.1	548.0
1.0	0.822 ± 0.094	0.820 ± 0.084	0.820 ± 0.047
	261.9	412.6	541.9

Table 4: Results of *CAHS* with different hyper-parameter settings of τ on all benchmarks.

provisioning for Microsoft Azure. Through our statistics on the collected data, *CAHS* benefits the majority of VM provisioning processes that are eligible for PPS, and the median of the VM provisioning time has been significantly shortened.

Applications in Microsoft 365

Although our discussion has been mainly focused on solving the PreVMP problem, our proposed *CAHS* method provides a general framework for dealing with a broad range of Prediction+Optimization problems, including many common issues in managing large number of workloads hosted on cloud systems. The ideas used in *CAHS*, including correlation exploration, uncertainty modeling, heuristic search and the unified Prediction+Optimization framework, have also been leveraged for developing intelligent algorithms to improve the efficiency and the reliability of Microsoft 365's workloads hosted on the cloud. In this subsection, we will discuss the general applicability of *CAHS* and provide a brief overview on the relevant applications in managing Microsoft 365's cloud services.

Methods and ideas explored in *CAHS* can be extremely valuable for managing dedicated cloud servers hosting workloads owned by large enterprises and organizations such as Microsoft 365 as well as public cloud systems. Similar to the scenario of intelligent VM provisioning, the management of cloud services needs to handle unknown quantities corresponding to the fluctuating demands of customers, and requires to solve various challenging optimization problems under multiple hard and soft constraints. For instance, one of the most frequent tasks is the allocation of VMs to physical machines. Given objective functions such as the packing density or machine load balance, a difficult combinatorial optimization problem has to be solved to obtain the best allocation plan under resource constraints such as CPU and memory. A better allocation policy also needs to plan for unknown factors such as the life time, priority, and actual resource utilization of a VM, which can only be predicted with models trained on historical data. Such application scenarios occur during both the initial onboarding of VMs and run time adjustment such as live migration. Moreover, practical tasks such as capacity planning, VM scheduling, and power management also require to solve different combinatorial optimization problems while taking the unknown future workload demand into consideration.

In addition to VM provisioning, we have leveraged ideas in *CAHS* to develop intelligent algorithms for managing cloud workloads owned by Microsoft 365. The B2 autopilot container allocation algorithm adopts the similar heuristic search method used in *CAHS* to guide the allocation of containers hosting Office workloads in Microsoft 365 data centers under multiple resource constraints. This algorithm has significantly improved the load balance across physical machines and significantly reduced the number of hot-spot machines. The AzSC packing algorithm deployed on AzSC, the dedicated clusters for Microsoft 365, also relies on the heuristic search method to improve the packing efficiency. This algorithm has resulted in an increment of the overall packing density and a reduction of the number of stranded cores in the fleet. In order to improve the capacity efficiency, the DADRAC (Data-Driven Admission Control) system utilizes predicted demand growth at cluster level to dynamically set the optimal buffer threshold for each cluster. This approach has freed considerable computing resources and contributed to the effort of easing the resource shortage during the outbreak of COVID-19. All those examples rely on heuristic search algorithms and prediction-based planning inspired by *CAHS* to improve the efficiency and reliability of cloud services provided by Microsoft 365. These successful applications present the application potential of Prediction+Optimization methods such as *CAHS*, and the future development of intelligent cloud management system will continuously benefit from new advancements in this area.

Conclusion

In this paper, we aim to advance the state of the art in solving the predictive virtual machine provisioning (PreVMP) problem, which is a typical Prediction+Optimization problem and is critical in cloud computing. We conduct empirical study on application benchmarks and provide empirical evidence showing the prevalence of significant correlation among demands of different VM types. Then, we propose a novel approach dubbed *CAHS* for solving the PreVMP problem. The main advantage of *CAHS* is that *CAHS* incorporates demand correlation into prediction and optimization. Our extensive experiments on two public benchmarks and one industrial benchmark clearly demonstrate the superiority of *CAHS*. Furthermore, *CAHS* has been successfully deployed in Microsoft Azure and achieved significant performance improvement for that cloud system, which indicates that *CAHS* can bring considerable performance improvement in production.

Acknowledgments

We express our deep gratitude to Dongmei Zhang, Girish Bablani, Robert Gu, Jim Kleewein, Yingnong Dang, Gil Lapid Shafriri, Murali Chintalapati, Sushant Rewaskar, Thomas Moscibroda, Melur Raghuraman, Marcus Fontoura and Andrew Zhou for their great support and sponsorship. Besides, we would like to thank Daud Howlader, Anusha Kowdeed, Chandramouleswaran Ravichandran, Bowen Xu and Tri Tran for the collaboration on the prediction work from Pre-Provisioning Service in Microsoft Azure, and to thank Raphael Ghelman and Eli Cortez for servicing the prediction in the Resource Central service system in Microsoft Azure. Also, we would like to thank Yanan Wang, Johnny Li and Mohit Datta for the collaboration on the project of B2 autopilot container allocation in Microsoft 365, and to thank Senthil kumaran Baladhandayutham, Chetan Bansal, Victor Rühle, Camille Couturier, Silvia Yu, Zhou Zhou, Si Qin, Jue Zhang, Neil Yan, Hang Dong, Mike Hung, Qingchuan Zhang, Boris Chen, Anyang Yu, Tianqi Zhao, Maosen Huang and Yiqu Liu for the collaboration on the project of AzSC packing in Microsoft 365.

References

Adams, S. C.; and Beling, P. A. 2019. A survey of feature selection methods for Gaussian mixture models and hidden Markov models. *Artificial Intelligence Review* 52(3): 1739–1779.

Balghiti, O. E.; Elmachtoub, A. N.; Grigas, P.; and Tewari, A. 2019. Generalization Bounds in the Predict-then-Optimize Framework. In *Proceedings of NeurIPS 2019*, 14389–14398.

Bertsimas, D.; Gupta, V.; and Kallus, N. 2018. Data-driven robust optimization. *Mathematical Programming* 167: 235–292.

Cai, S. 2015. Balance between Complexity and Quality: Local Search for Minimum Vertex Cover in Massive Graphs. In *Proceedings of IJCAI 2015*, 747–753.

Cai, S.; Lin, J.; and Luo, C. 2017. Finding A Small Vertex Cover in Massive Sparse Graphs: Construct, Local Search, and Preprocess. *Journal of Artificial Intelligence Research* 59: 463–494.

Cai, S.; and Su, K. 2013. Local search for Boolean Satisfiability with configuration checking and subscore. *Artificial Intelligence* 204: 75–98.

Cai, S.; Su, K.; Luo, C.; and Sattar, A. 2013. NuMVC: An Efficient Local Search Algorithm for Minimum Vertex Cover. *Journal of Artificial Intelligence Research* 46: 687–716.

Cong, Y.; Chen, B.; and Zhou, M. 2017. Fast Simulation of Hyperplane-Truncated Multivariate Normal Distributions. *Bayesian Analysis* 12(4): 1017–1037.

Cortez, E.; Bonde, A.; Muzio, A.; Russinovich, M.; Fontoura, M.; and Bianchini, R. 2017. Resource Central: Understanding and Predicting Workloads for Improved Resource Management in Large Cloud Platforms. In *Proceedings of SOSP 2017*, 153–167.

Demirović, E.; Stuckey, P. J.; Bailey, J.; Chan, J.; Leckie, C.; Ramamohanarao, K.; and Guns, T. 2019a. An Investigation into Prediction + Optimisation for the Knapsack Problem. In *Proceedings of CPAIOR 2019*, 241–257.

Demirović, E.; Stuckey, P. J.; Bailey, J.; Chan, J.; Leckie, C.; Ramamohanarao, K.; and Guns, T. 2019b. Predict+Optimise with Ranking Objectives: Exhaustively Learning Linear Functions. In *Proceedings of IJCAI 2019*, 1078–1085.

Demirović, E.; Stuckey, P. J.; Guns, T.; Bailey, J.; Leckie, C.; Ramamohanarao, K.; and Chan, J. 2020. Dynamic Programming for Predict+Optimise. In *Proceedings of AAAI 2020*, 1444–1451.

Durbin, J.; and Koopman, S. J. 2012. *Time Series Analysis by State Space Methods*. Oxford University Press.

Elmachtoub, A. N.; and Grigas, P. 2017. Smart "Predict, then Optimize". *CoRR* abs/1710.08005.

Ester, M.; Kriegel, H.; Sander, J.; and Xu, X. 1996. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proceedings of KDD 1996*, 226–231.

Hastie, T.; Friedman, J. H.; and Tibshirani, R. 2001. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer.

Hbaieb, A.; Khemakhem, M.; and Jemaa, M. B. 2017. Using Decomposition and Local Search to Solve Large-Scale Virtual Machine Placement Problems with Disk Anti-Colocation Constraints. In *Proceedings of AICCSA 2017*, 688–695.

Hyndman, R. J.; and Khandakar, Y. 2008. Automatic Time Series Forecasting: The forecast Package for R. *Journal of Statistical Software* 27(3): 1–22.

Innocenti, G.; and Materassi, D. 2008. Econometrics as Sorcery. *CoRR* abs/0801.3047.

Karp, R. M. 2011. Heuristic algorithms in computational molecular biology. *Journal of Computer and System Sciences* 77(1): 122–128.

Liu, X. F.; Zhan, Z.; Deng, J. D.; Li, Y.; Gu, T.; and Zhang, J. 2018. An Energy Efficient Ant Colony System for Virtual Machine Placement in Cloud Computing. *IEEE Transactions on Evolutionary Computation* 22(1): 113–128.

Lloyd, S. P. 1982. Least Squares Quantization in PCM. *IEEE Transactions on Information Theory* 28(2): 129–136.

Luo, C.; Cai, S.; Su, K.; and Huang, W. 2017. CCEHC: An efficient local search algorithm for weighted partial maximum satisfiability. *Artificial Intelligence* 243: 26–44.

Luo, C.; Cai, S.; Su, K.; and Wu, W. 2015a. Clause States Based Configuration Checking in Local Search for Satisfiability. *IEEE Transactions on Cybernetics* 45(5): 1014–1027.

Luo, C.; Cai, S.; Wu, W.; Jie, Z.; and Su, K. 2015b. CCLS: An Efficient Local Search Algorithm for Weighted Maximum Satisfiability. *IEEE Transactions on Computers* 64(7): 1830–1843.

Luo, C.; Cai, S.; Wu, W.; and Su, K. 2014. Double Configuration Checking in Stochastic Local Search for Satisfiability. In *Proceedings of AAAI 2014*, 2703–2709.

Luo, C.; Hoos, H. H.; Cai, S.; Lin, Q.; Zhang, H.; and Zhang, D. 2019. Local Search with Efficient Automatic Configuration for Minimum Vertex Cover. In *Proceedings of IJCAI 2019*, 1297–1304.

Luo, C.; Qiao, B.; Chen, X.; Zhao, P.; Yao, R.; Zhang, H.; Wu, W.; Zhou, A.; and Lin, Q. 2020. Intelligent Virtual Machine Provisioning in Cloud Computing. In *Proceedings of IJCAI 2020*, 1495–1502.

Mandi, J.; Demirović, E.; Stuckey, P. J.; and Guns, T. 2020. Smart Predict-and-Optimize for Hard Combinatorial Optimization Problems. In *Proceedings of AAAI 2020*, 1603–1610.

Mann, Z. Á. 2016. Multicore-Aware Virtual Machine Placement in Cloud Data Centers. *IEEE Transactions on Computers* 65(11): 3357–3369.

Mao, M.; and Humphrey, M. 2012. A Performance Study on the VM Startup Time in the Cloud. In *Proceedings of IEEE CLOUD 2012*, 423–430.

Rodgers, J. L.; and Nicewander, W. A. 1988. Thirteen Ways to Look at the Correlation Coefficient. *The American Statistician* 42(1): 59–66.

Shi, J.; and Malik, J. 2000. Normalized Cuts and Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22(8): 888–905.

Spall, J. C. 2003. *Introduction to Stochastic Search and Optimization: Estimation, Simulation, and Control*. Wiley.

Taylor, S. J.; and Letham, B. 2018. Forecasting at Scale. *The American Statistician* 72(1): 37–45.

Verbeek, J. J.; Vlassis, N. A.; and Kröse, B. J. A. 2003. Efficient Greedy Learning of Gaussian Mixture Models. *Neural Computation* 15(2): 469–485.

Wilder, B.; Dilkina, B.; and Tambe, M. 2019. Melding the Data-Decisions Pipeline: Decision-Focused Learning for Combinatorial Optimization. In *Proceedings of AAAI 2019*, 1658–1665.

Xiong, H.; Shekhar, S.; Tan, P.; and Kumar, V. 2004. Exploiting a Support-Based Upper Bound of Pearson's Correlation Coefficient for Efficiently Identifying Strongly Correlated Pairs. In *Proceedings of KDD 2004*, 334–343.

Young, P. C. 2011. Unobserved Component Models. In *Recursive Estimation and Time-Series Analysis*, 99–136.

Zhang, Z.; Li, Z.; Wu, K.; Li, D.; Li, H.; Peng, Y.; and Lu, X. 2014. VMThunder: Fast Provisioning of Large-Scale Virtual Machine Clusters. *IEEE Transactions on Parallel and Distributed Systems* 25(12): 3328–3338.

Zhao, H.; Wang, J.; Liu, F.; Wang, Q.; Zhang, W.; and Zheng, Q. 2018. Power-Aware and Performance-Guaranteed Virtual Machine Placement in the Cloud. *IEEE Transactions on Parallel and Distributed Systems* 29(6): 1385–1400.