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

Chuan Luo, Bo Qiao, Wenhuan Xing, Xin Chen, Pu Zhao, Chao Du, Randolph Yao, Hongyu Zhang, Wei Wu, Shaowei Cai, Bing He, Saravanakumar Rajmohan, Qingwei Lin

1Microsoft Research, China
2Microsoft 365, United States
3Microsoft Azure, United States
4The University of Newcastle, Australia
5L3S Research Center, Leibniz University Hannover, Germany
6State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, China
7School of Computer Science and Technology, University of Chinese Academy of Sciences, China
{chuan.luo, boqiao, v-xich15, puzhao, chaodu, ranyao, saravar, qlin}@microsoft.com,
hongyu.zhang@newcastle.edu.au, william.third.wu@gmail.com, {caisw, hebing}@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 solve the predictive VM provisioning (PreVMP) problem in which the VM provisioning plans are optimized based on the predicted demands of different VM types, to ensure rapid provisioning 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 reliability of the cloud services provided by Microsoft 365.

Introduction
Cloud computing has become a popular and powerful computing paradigm (Wang et al. 2015). 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.

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 arising 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 corre-
loration 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 AzSCC 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).

Notations Related to VMs: A virtual machine (VM) type $v_i = (\alpha_i, B_i)$ is a virtual tuple, where $\alpha_i$ denotes the number of CPU cores of $v_i$, and $B_i = \{b_{i1}, \ldots, b_{i\beta}\}$ ($|B_i| = \beta$) denotes the amount of $\beta$ resource types (e.g., memory and disk) of $v_i$. Given a set of $n$ VM types, the historical demands for all VM types are represented by $D = \{d_{it} | i \in \{1, \ldots, n\}, t \in \{1, \ldots, T\}\}$, where each $d_{it}$ is a non-negative integer and represents the demand of VM type $v_i$ during the time period $[t-1, t]$. Moreover, $D_i = \{d_{i1}, \ldots, d_{iT}\}$ denotes the historical demands of VM type $v_i$; and $D^t = \{d_{1t}, \ldots, d_{nt}\}$ refers to the demands of all VM types in the period $[t-1, t]$. Finally, $Y^* = \{y_{1t}, \ldots, y_{nt}\}$ represents the real demands for all VM types in the time period $[T, T+1]$, which are 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 cores of $p_j$, and $H_j = \{h_{j1}, \ldots, h_{j\beta}\}$ ($|H_j| = \beta$) denotes the amount of $\beta$ resource types of $p_j$. At any time stamp $t$, the cloud system can provide a collection of $m^t$ PMs, i.e., $P^t = \{p_{1t}, \ldots, p_{mt}\}$, to provision VMs during the time period $[t, t+1]$. For simplicity, notations $P$, $p_j$ and $m$ represent $P^T$, $p_{jt}$ 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 cannot exceed $C_j$; for each of $\beta$ 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 $A = \{a_{i, j} | i \in \{1, \ldots, n\}, j \in \{1, \ldots, m\}\}$ to denote $n \\ m$ 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 $X = \{x_i | i \in \{1, \ldots, n\}\}$ 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.

Optimistic: It is 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 $u_i = \min\{x_i, y_{it}\} \cdot \alpha_i$, and the total number of utilized CPU cores for all VM types is $u = \sum_{i=1}^{n} u_i$. Thus, 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.

\[
\text{maximize} \quad u = \sum_{i=1}^{n} \min \{ x_i, y_i^* \} \cdot \alpha_i \\
\text{s.t.} \quad \sum_{i=1}^{n} a_{i,j} \cdot \alpha_i \leq C_j, \quad j \in J \\
\sum_{i=1}^{n} a_{i,j} \cdot b_{i,k} \leq h_{j,k}, \quad (j,k) \in J \times K \quad (1) \\
a_{i,j} \geq 0, \quad (i,j) \in I \times J \\
a_{i,j} \text{ is an integer}, \quad (i,j) \in I \times J
\]

where \( I = \{1, \ldots, n\} \), \( J = \{1, \ldots, m\} \) and \( K = \{1, \ldots, \beta\} \) (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., \( \{ y_i^* \mid i \in \{1, \ldots, n\} \} \)) 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., \( \{ y_i^* \mid i \in \{1, \ldots, n\} \} \)) 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 \( \{ y_i^* \mid 1 \leq i \leq n \} \) 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 (Wildér, 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), Decision-NN (Wildér, 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.

### 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; Gao, Bagdouri, and Oard 2016). 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.,

\[
\rho(v_i, v_j) = \frac{\text{cov}(D_i,D_j)}{\sigma(D_i) \cdot \sigma(D_j)} \quad (-1 \leq \rho(v_i, v_j) \leq 1),
\]

where \( \text{cov}(\cdot, \cdot) \) 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 \( \eta \) 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.

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.

High-level 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 its estimated performance through a neural network based performance regressor.

Prediction Phase: In this phase, for each partitioned VM type group \( g \), CAHS jointly predicts the future demands for all VM types in \( g \), 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 \( Cfg^* \) is 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 \( \kappa \) (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 \( \tau \) (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 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 $g_s$, is identified, and those VM types in $g_s$ 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 $g_s$ containing more than one VM type, all VM types in $g_s$ 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 with the best performance, estimated by $Cfg^*$. 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 iteratively 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 $g_s$ ($D_{g_s} = \{d_{s, t} \mid v_{s, t} \in g_s, t \in \{1, \ldots, T\}\}$). Further, for group $g_s$ and time stamp $t$, we use $D_{s, t}$ to denote the demands for all VM types in $g_s$ during the time period $[t−1, t]$ ($D_{s, t}$ is a vector). In addition, for group $g_s$, we use $\Gamma_s^T(D_{g_s})$ to denote a submatrix of $D_{g_s}$, with the first time stamp of $t$ and the length of $l$ ($\Gamma_s^T(D_{g_s}) = \{D_{s, t}^l, D_{s, t}^{l+1}, \ldots, D_{s, T}^l\}$).

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 (Box et al. 2015) to split the group demand $D_{g_s}$ into a set of sequential matrices, i.e., $\{\Gamma_s^T(D_{g_s}) \mid 1 \leq t \leq T−l+1\} \cup \{D_{s, T}^l\}$, where $l$ is a configurable parameter and can be specified by $Cfg^*$. For each sequential matrix $\Gamma_s^T(D_{g_s})$, CAHS considers the last demand vector $D_{g_s}^{t−l}$ as the label vector, and then forecasts $D_{s, t}^{l+1}$, through a predictor trained on $\Gamma_s^T(D_{g_s}) \cup \{D_{s, T}^l\}$, 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 $\epsilon_{g_s}$ by fitting a multi-variant Gaussian Mixture Model (GMM) (McLachlan and Peel 2000), 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 (Pelagatti 2015).

---

**Algorithm 2: Configuration Selection**

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

For each time period \([t - 1, t]\), this phase solves a corresponding VMP problem subject to the resource constraints and 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 the neglect of 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 \(\theta_s\) 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 \(f_{g_s}\). Then the (unknown) real demand \(y^*_s = \tilde{d}^*_s + \theta_{s_i}\) 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., \(E(u)\)).

As discussed in the Problem Definition section, the optimization problem in our context is an NP-hard problem. To solve this challenge, we design a new heuristic search algorithm based on a novel, effective scoring function (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:

\[
\text{score}(v_i) = \alpha_i \cdot \left(1 - \text{CDF}_{y^*_i|g_s\setminus\{v_i\}}(x_i)\right) + \sum_{j\neq i} \left(\alpha_j \cdot \text{CDF}_{y^*_j|g_s\setminus\{v_i,v_j\}}(y_j - x_j | y^*_i = x_i + 1)\right) - \sum_{j\neq i} \left(\alpha_j \cdot \text{CDF}_{y^*_j|g_s\setminus\{v_i,v_j\}}(y_j - x_j | y^*_i = x_i)\right) + \sum_{j\neq i} \left(\alpha_j \cdot x_j \cdot \text{CDF}_{y^*_j|g_s\setminus\{v_i,v_j\}}(x_j | y^*_i = x_i + 1)\right) - \sum_{j\neq i} \left(\alpha_j \cdot x_j \cdot \text{CDF}_{y^*_j|g_s\setminus\{v_i,v_j\}}(x_j | y^*_i = x_i)\right). \tag{1}
\]

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.

Discussion

**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.

\(^2\)https://github.com/Azure/AzurePublicDataset
The Azure-2017 benchmark contains 110 VM types and summarizes VM provisioning workloads of Microsoft 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.

State-of-the-art 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), Decision-NN (Wildler, 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) (Hyndman and Athanasopoulos 2013), time series decomposition based forecasting approach (TSDec) (Hyndman and Athanasopoulos 2013), automatic autoregressive integrated moving average (AutoARIMA) (Hyndman and Athanasopoulos 2013), 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, Decision-NN 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 $\kappa$ and $\tau$ 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

Improvement over State-of-the-art 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 CAHS-alt2 only considers demand correlation in its predic-

<table>
<thead>
<tr>
<th>Approach</th>
<th>Azure-2017 avg. r ± SD (time (sec))</th>
<th>Azure-2019 avg. r ± SD (time (sec))</th>
<th>INDU-2020 avg. r ± SD (time (sec))</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAHS</td>
<td>0.871 ± 0.064 ± 0.333 ± 0.032</td>
<td>0.872 ± 0.032 ± 0.032</td>
<td></td>
</tr>
<tr>
<td>CAHS-alt1</td>
<td>0.822 ± 0.094 ± 0.084 ± 0.047</td>
<td>0.820 ± 0.047 ± 0.047</td>
<td></td>
</tr>
<tr>
<td>CAHS-alt2</td>
<td>0.839 ± 0.087 ± 0.091 ± 0.047</td>
<td>0.837 ± 0.047 ± 0.047</td>
<td></td>
</tr>
<tr>
<td>CAHS-alt3</td>
<td>0.842 ± 0.063 ± 0.051 ± 0.047</td>
<td>0.853 ± 0.060 ± 0.051</td>
<td></td>
</tr>
<tr>
<td>CAHS-alt4</td>
<td>0.847 ± 0.051 ± 0.072 ± 0.047</td>
<td>0.863 ± 0.037 ± 0.037</td>
<td></td>
</tr>
<tr>
<td>CAHS-alt5</td>
<td>0.844 ± 0.077 ± 0.084 ± 0.058</td>
<td>0.853 ± 0.055 ± 0.058</td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Results of CAHS, CAHS-alt1, CAHS-alt2, CAHS-alt3, CAHS-alt4 and CAHS-alt5 on all benchmarks.
CAHS clearly outperforms alt1 and i.e., well-known clustering algorithms (performance than CAHS – 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.

Effect of Hyper-parameter Settings: Since CAHS introduces two hyper-parameters \( \kappa \) and \( \tau \), the experimental results of CAHS with different hyper-parameter settings of \( \kappa \) and \( \tau \) are presented in Tables 3 and 4, respectively. Table 3 presents that CAHS can achieve better performance with larger \( \kappa \), and Table 4 demonstrates that CAHS can perform better when \( \tau \) 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 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.

Table 3: Results of CAHS with different hyper-parameter settings of \( \kappa \) on all benchmarks.

<table>
<thead>
<tr>
<th>( \kappa )</th>
<th>Azure-2017</th>
<th>Azure-2019</th>
<th>INDU-2020</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>avg. r ± SD</td>
<td>avg. r ± SD</td>
<td>avg. r ± SD</td>
</tr>
<tr>
<td></td>
<td>time (sec)</td>
<td>time (sec)</td>
<td>time (sec)</td>
</tr>
<tr>
<td>1</td>
<td>0.837 ± 0.081</td>
<td>0.802 ± 0.094</td>
<td>0.833 ± 0.050</td>
</tr>
<tr>
<td>10</td>
<td>0.851 ± 0.061</td>
<td>0.832 ± 0.045</td>
<td>0.850 ± 0.040</td>
</tr>
<tr>
<td>25</td>
<td>0.861 ± 0.060</td>
<td>0.842 ± 0.039</td>
<td>0.859 ± 0.038</td>
</tr>
<tr>
<td>50</td>
<td>0.871 ± 0.064</td>
<td>0.851 ± 0.033</td>
<td>0.872 ± 0.032</td>
</tr>
<tr>
<td>100</td>
<td>0.877 ± 0.048</td>
<td>0.854 ± 0.033</td>
<td>0.872 ± 0.038</td>
</tr>
<tr>
<td>150</td>
<td>0.881 ± 0.047</td>
<td>0.855 ± 0.035</td>
<td><strong>0.873 ± 0.033</strong></td>
</tr>
<tr>
<td>200</td>
<td><strong>0.883 ± 0.047</strong></td>
<td><strong>0.858 ± 0.036</strong></td>
<td>0.872 ± 0.039</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( \tau )</th>
<th>Azure-2017</th>
<th>Azure-2019</th>
<th>INDU-2020</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>avg. r ± SD</td>
<td>avg. r ± SD</td>
<td>avg. r ± SD</td>
</tr>
<tr>
<td></td>
<td>time (sec)</td>
<td>time (sec)</td>
<td>time (sec)</td>
</tr>
<tr>
<td>0</td>
<td>0.825 ± 0.099</td>
<td>0.818 ± 0.057</td>
<td>0.829 ± 0.064</td>
</tr>
<tr>
<td>0.1</td>
<td>0.847 ± 0.065</td>
<td>0.825 ± 0.049</td>
<td>0.853 ± 0.064</td>
</tr>
<tr>
<td>0.2</td>
<td>0.853 ± 0.061</td>
<td>0.821 ± 0.057</td>
<td>0.855 ± 0.064</td>
</tr>
<tr>
<td>0.3</td>
<td>0.841 ± 0.074</td>
<td>0.825 ± 0.050</td>
<td>0.862 ± 0.065</td>
</tr>
<tr>
<td>0.4</td>
<td>0.846 ± 0.067</td>
<td>0.826 ± 0.047</td>
<td>0.865 ± 0.062</td>
</tr>
<tr>
<td>0.5</td>
<td>0.846 ± 0.062</td>
<td>0.828 ± 0.051</td>
<td>0.868 ± 0.034</td>
</tr>
<tr>
<td>0.6</td>
<td>0.848 ± 0.076</td>
<td>0.836 ± 0.039</td>
<td><strong>0.873 ± 0.030</strong></td>
</tr>
<tr>
<td>0.7</td>
<td>0.855 ± 0.065</td>
<td>0.842 ± 0.034</td>
<td>0.871 ± 0.039</td>
</tr>
<tr>
<td>0.8</td>
<td><strong>0.871 ± 0.064</strong></td>
<td><strong>0.851 ± 0.033</strong></td>
<td>0.872 ± 0.032</td>
</tr>
<tr>
<td>0.9</td>
<td>0.853 ± 0.077</td>
<td>0.845 ± 0.041</td>
<td>0.854 ± 0.053</td>
</tr>
<tr>
<td>1.0</td>
<td>0.822 ± 0.094</td>
<td>0.820 ± 0.084</td>
<td>0.820 ± 0.047</td>
</tr>
</tbody>
</table>

Table 4: Results of CAHS with different hyper-parameter settings of \( \tau \) on all benchmarks.
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 Klewein, Yingnong Dang, Gil Lapid Shafiriri, Murali Chintalapati, Sushant Rewaskar, Thomas Moscibrod, 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


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


