Intelligent Virtual Machine Provisioning in Cloud Computing

Chuan Luo[1] , **Bo Qiao**[1] , **Xin Chen**[1] , **Pu Zhao**[1] , **Randolph Yao**[2] ,
Hongyu Zhang[3] , **Wei Wu**[4] , **Andrew Zhou**[5] and **Qingwei Lin**[1,*]
[1]Microsoft Research, China
[2]Microsoft Azure, United States
[3]The University of Newcastle, Australia
[4]University of Technology Sydney, Australia
[5]Microsoft Office, China
{chuan.luo, boqiao, v-xich15, puzhao, ranyao, azhou, qlin}@microsoft.com,
hongyu.zhang@newcastle.edu.au, william.third.wu@gmail.com

Abstract

Virtual machine (VM) provisioning is a common and critical problem in cloud computing. In industrial cloud platforms, there are a huge number of VMs provisioned per day. Due to the complexity and resource constraints, it needs to be carefully optimized to make cloud platforms effectively utilize the resources. Moreover, in practice, provisioning a VM from scratch requires fairly long time, which would degrade the customer experience. Hence, it is advisable to provision VMs ahead for upcoming demands. In this work, we formulate the practical scenario as the predictive VM provisioning (PreVMP) problem, where upcoming demands are unknown and need to be predicted in advance, and then the VM provisioning plan is optimized based on the predicted demands. Further, we propose *Uncertainty-Aware Heuristic Search* (*UAHS*) for solving the PreVMP problem. *UAHS* first models the prediction uncertainty, and then utilizes the prediction uncertainty in optimization. Moreover, *UAHS* leverages Bayesian optimization to interact prediction and optimization to improve its practical performance. Extensive experiments show that *UAHS* performs much better than state-of-the-art competitors on two public datasets and an industrial dataset. *UAHS* has been successfully applied in Microsoft Azure and brought practical benefits in real-world applications.

1 Introduction

Recently, cloud computing has emerged as a new computing paradigm that offers a variety of services [Wang *et al.*, 2015]. Virtual machine (VM) provisioning is a common and critical problem in cloud computing. In practice, a huge number of VMs are provisioned per day in industrial cloud platforms, and the cloud platform may not efficiently utilize the resources by merely adopting any simple provisioning plan. However, due to its complicated resource constraints, the VM provisioning problem is computationally hard and urgently calls for effective solutions [Hbaieb *et al.*, 2017].

In current industrial practice, when a customer requests for a VM, the VM is provisioned from scratch, which usually costs much time [Mao and Humphrey, 2012]. The long delay in VM provisioning significantly degrades the customer experience [Zhang *et al.*, 2014]. However, most steps in VM provisioning are independent to a particular request, which can be done before request. Hence, it is advisable to provision VMs ahead, and to serve requests with provisioned VMs.

In this work, we formulate the industrial scenario described above as the predictive VM provisioning (PreVMP) problem, where upcoming demands for various VM types are unknown and need to be predicted in advance, and then the VM provisioning plan is optimized using the predicted demands. The PreVMP problem can be treated as a problem of prediction with optimization [Wilder *et al.*, 2019]. Since the optimization objective of PreVMP contains unknown parameters (*i.e.,* real demands), conventional optimization methods are not able to handle such a problem.

We propose *Uncertainty-Aware Heuristic Search* (*UAHS*), which is a novel approach for solving PreVMP. In particular, *UAHS* not only predicts future demands for all VM types, but also models the prediction uncertainty for all VM types. Then *UAHS* utilizes the predicted demands and the modeled prediction uncertainty to optimize the VM provisioning plan. Moreover, *UAHS* leverages Bayesian optimization to effectively interact prediction and optimization, in order to improve its performance. In this way, *UAHS* solves PreVMP by combining prediction and optimization in an integral way.

Extensive experiments on two public datasets and an industrial dataset demonstrate that *UAHS* can perform much better than existing state-of-the-art methods. Furthermore, we have successfully applied *UAHS* in Microsoft Azure, which is an industrial public cloud platform, and observed that the median of the VM provisioning time has been reduced by around 42% after the deployment of *UAHS*.

Our main contributions in this work are as follows:

- We formulate the industrial scenario as the PreVMP problem, which shows benefits in practice.

*Corresponding author.

- We propose *UAHS*, a novel approach for PreVMP. *UAHS* models and utilizes the prediction uncertainty to conduct optimization. Also, *UAHS* leverages Bayesian optimization to effectively interact prediction and optimization.
- Extensive experiments on two public datasets and an industrial dataset indicate the effectiveness of *UAHS*. Furthermore, *UAHS* has been applied in Microsoft Azure and brought practical benefits in real-world applications.

2 Problem Formulation

In this section, we formally formulate the problem of predictive virtual machine provisioning (PreVMP) as follows.

Definitions about virtual machines. Given a set of n *virtual machine types* $V = \{v_1, \ldots, v_n\}$, each virtual machine (VM) type contains three attributes, *i.e.,* $v_i = (\alpha_i, \beta_i, \gamma_i)$, where α_i, β_i and γ_i are the amount of *CPU cores*, *memory* and *storage* of VM type v_i, respectively. We have the *historical demands* for all n VM types $D = \{d_i^t \mid i \in \{1, \ldots, n\}, t \in \{1, \ldots, T\}\}$, where t denotes a time stamp, and d_i^t is a non-negative integer and denotes the demand of VM type v_i accumulated within the time period $[t-1, t]$. Also, we use the notation D_i to denote the historical demand for VM type v_i, *i.e.,* $D_i = \{d_i^1, \ldots, d_i^T\}$, which is an equally spaced time series; we use the notation D^t to denote the set of all VM types' demands in the time period $[t-1, t]$, *i.e.,* $D^t = \{d_1^t, \ldots, d_n^t\}$. We use $Y^* = \{y_1^*, \ldots, y_n^*\}$ to denote the real demands for all VM types in the time period $[T, T+1]$, where y_i^* is a non-negative integer and denotes the real demand for VM type v_i in the time period $[T, T+1]$; hence, Y^* is *unknown* before the time stamp $T+1$.

Definitions about physical machines. At any time stamp t, based on the cloud platform status, the cloud platform can provide a set of m^t *physical machines* $P^t = \{p_1^t, \ldots, p_m^t\}$ to provision VMs. In this work, we use notations m, P and p_j to represent m^T, P^T and p_j^T (the physical machine related status information at time stamp T), respectively. Each physical machine (PM) contains three attributes, *i.e.,* $p_j = (C_j, M_j, S_j)$, where C_j, M_j and S_j denote the available resource of *CPU cores*, *memory*, and *storage* of PM p_j, respectively. A VM must be provisioned on a PM with sufficient resource: For each PM p_j, the VMs provisioned on p_j must satisfy the resource constraints that the total amount of CPU cores, memory, and storage requested by all VMs provisioned on PM p_j cannot exceed C_j, M_j and S_j, respectively.

Necessary notations. The notation A denotes a set of $n \cdot m$ integer decision variables, *i.e.,* $A = \{a_{i,j} \mid i \in \{1, \ldots, n\}, j \in \{1, \ldots, m\}\}$, where $a_{i,j}$ denotes the number of VMs with VM type v_i to be provisioned on PM p_j. Also, the notation $X = \{x_1, \ldots, x_n\}$ denotes a set of n decision variables, where x_i denotes the number of VMs with VM type v_i to be provisioned, *i.e.,* $x_i = \sum_{j=1}^{m} a_{i,j}$. A *provisioning plan* is an assignment to A; a plan is *feasible* if all resource constraints are satisfied. For VM type v_i, the amount of the utilized CPU cores is calculated as $h_i = \min\{x_i, y_i^*\} \cdot \alpha_i$. For all VM types, the overall amount of utilized CPU cores is $h = \sum_{i=1}^{n} h_i$, and the overall CPU core utilization ratio is $r = h / (\sum_{j=1}^{m} C_j)$.

Problem formulation. At time stamp T, given a set of n VM types V and its historical demands D, as well as, a set of m PMs P, the problem of **predictive virtual machine provisioning** (PreVMP) is to find a feasible provisioning plan, which maximizes the overall CPU core utilization ratio r, since core-level based provisioning can achieve good performance for the cloud platform [Zhao *et al.*, 2018]. Maximizing the overall CPU core utilization ratio r is equivalent to maximizing the overall amount of utilized CPU cores h, since the denominator (*i.e.,* $\sum_{j=1}^{m} C_j$) is a constant. We give the formal formulation as follows.

$$
\begin{aligned}
\text{maximize} \quad & h = \textstyle\sum_{i=1}^{n} \min\{x_i, y_i^*\} \cdot \alpha_i & & \\
\text{s.t.} \quad & \textstyle\sum_{i=1}^{n} a_{i,j} \cdot \alpha_i \le C_j, & & j \in J \\
& \textstyle\sum_{i=1}^{n} a_{i,j} \cdot \beta_i \le M_j, & & j \in J \\
& \textstyle\sum_{i=1}^{n} a_{i,j} \cdot \gamma_i \le S_j, & & j \in J \\
& a_{i,j} \ge 0, & & (i,j) \in I \times J \\
& a_{i,j} \text{ is an integer}, & & (i,j) \in I \times J
\end{aligned}
\tag{1}
$$

where $x_i = \sum_{j=1}^{m} a_{i,j}$, $I = \{1, \ldots, n\}$ and $J = \{1, \ldots, m\}$. Regarding Equation 1, recall that, for $i \in I$, y_i^* (*i.e.,* the real demand for VM type v_i in the time period $[T, T+1]$) remains unknown at time stamp T.

Remarks. In theory, the predictive VM provisioning problem is a computationally hard problem; even if Y^* was known, the deterministic VM provisioning problem is NP-hard [Hbaieb *et al.*, 2017; Zhao *et al.*, 2018].

3 Related Work

The deterministic VM provisioning (VMP) problem aims to find a feasible provisioning plan that can optimize the allocations of VMs to PMs under resource constraints. A number of methods have been proposed to solve deterministic VMP [Hbaieb *et al.*, 2017; Zhao *et al.*, 2018], and the Ant Colony Optimization (ACO) algorithm [Zhao *et al.*, 2018] exhibits state-of-the-art performance. However, these methods are only applicable when the real VM demands are known.

For PreVMP, cloud platforms need to provision VMs according to the upcoming demands, which unfortunately are previously unknown. Hence, the PreVMP problem can be treated as a problem of prediction with optimization (Prediction+Optimization) [Wilder *et al.*, 2019]. A straightforward two-stage method first predicts unknown parameters and then directly conducts optimization based on the predicted results. An improved two-stage method called *Semi-direct* [Demirović *et al.*, 2019a] is proposed by modifying the loss function according to the characteristics of the optimization problem. Unfortunately, such two-stage methods assume that the predicted results are accurate, but in practice the prediction errors are inevitable [Wilder *et al.*, 2019].

Recently, a few methods are proposed to deal with several restricted variants of the Prediction+Optimization problem: 1) the optimization problem is a ranking problem [Demirović *et al.*, 2019b]; 2) the optimization problem needs to be solvable by dynamic programming [Demirović *et al.*, 2020]; 3) the optimization objective is linear [Elmachtoub and Grigas,

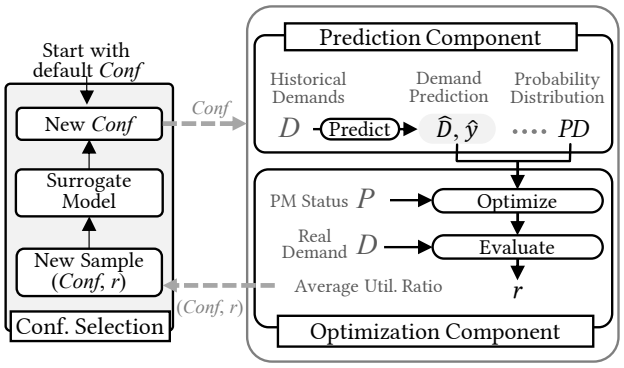

Figure 1: Top-level design of *UAHS*. The iteration process is terminated when the number of iterations reaches the limit $\boldsymbol{max\_iter}$.

Algorithm 1: Algorithm for searching configuration

Input: $\boldsymbol{prob}$: probability balancing exploitation and exploration;
Output: $\boldsymbol{Conf}$: selected configuration;

```
CS ← a set of randomly sampled configurations;
if with probability prob then
    Conf ← the configuration with the largest EI evaluated by the surrogate model from CS using the BMS sampling method;
else
    Conf ← the configuration with the largest variance evaluated by the surrogate model from CS using the BMS sampling method;
return Conf;
```

2017; Mandi *et al.*, 2020]. Unfortunately, the PreVMP problem does not satisfy any of those restrictions, so such methods are not applicable to the PreVMP problem.

Besides, a recent method called *Decision-NN* [Wilder *et al.*, 2019] based on neural network is proposed, and integrates prediction with optimization by directly using the gradient of the optimization objective function in the training phase of a machine learning model. To obtain the gradient of the discrete optimization objective function, a continuous relaxation mechanism is employed during the training phase. However, the continuous relaxation mechanism makes the discrete optimization objective function derivable but might incur inferior performance.

4 Uncertainty-Aware Heuristic Search

In this section, we first introduce the top-level design of *UAHS*, and then describe each key component of *UAHS*.

4.1 Top-level Design of *UAHS*

We first present the top-level design of *UAHS*, which consists of three key components: 1) configuration selection component, 2) prediction component, and 3) optimization component. We illustrate the top-level design of *UAHS* in Figure 1. *UAHS* works in an iterative framework.

Configuration selection component. This component selects a promising configuration[1] $Conf$ (*i.e.,* settings of internal configurable parameters) for the prediction component and the optimization component, through a surrogate model which relates the effect of configurations on performance.

Prediction component. This component is a configurable component, and is specified by the selected configuration $Conf$. For each VM type, it first utilizes the sliding window based time series analysis approach [Box *et al.*, 2015] to split the time series (*i.e.,* historical demand data) into a set of sub-time series; then it predicts the future VM demand for each sub-time series of each VM type, denoted by $\hat{d}_i^t$, and models the distribution of prediction uncertainty for each VM type, denoted by ϵ_i.

[1]For the first iteration, since the surrogate model is not built, the default configuration is selected.

Optimization component. For each time period $[t-1, t]$, it takes the predicted VM demands accumulated within $[t-1, t]$ for all VM types, denoted by $\hat{D}^t = \{\hat{d}_1^t, \ldots, \hat{d}_n^t\}$, and the distributions of prediction uncertainty for all VM types, denoted by $PD = \{\epsilon_1, \ldots, \epsilon_n\}$, as inputs; then it conducts optimization via heuristic search, subject to resource constraints regarding the PM status queried at time stamp $t-1$, denoted by P^{t-1}, to produce a provisioning plan. Once the plan is obtained, then the average utilization ratio r across all provisioning plans along with the selected configuration $Conf$ is treated as a new sample, denoted by $(Conf, r)$, to refine the surrogate model in the configuration selection component.

Remarks. During the iteration process of *UAHS*, the optimal configuration is recorded whenever it is found. *UAHS* conducts the iteration process until the number of iterations exceeds the limit max_iter, which is a hyper-parameter. Once the iteration process of *UAHS* is terminated, for each VM type, the prediction component configured with the optimal configuration is called to predict the (unknown) real demands, and models the distribution of prediction uncertainty. Then the predicted demands for all VM types, denoted by $\hat{D}^{T+1}$, and the distributions of prediction uncertainty for all VM types, denoted by PD, are handed to the optimization component. Finally, the optimization component configured with the optimal configuration, which takes $\hat{D}^{T+1}$ and PD as inputs, produces the final provisioning plan.

4.2 Configuration Selection Component

Bayesian optimization (BO) [Mockus, 1989] shows effectiveness in algorithm configuration [Hutter *et al.*, 2011]. Hence, we adopt BO in the configuration selection component. The main idea of BO is to build and maintain a surrogate model which relates the effect of configurations on performance, and then to use that surrogate model to iteratively select promising configurations. *UAHS* utilizes Gaussian process (GP) [Shahriari *et al.*, 2016] as its surrogate model. Given a configuration, GP can evaluate its potential benefits using *expected improvement* (EI) [Jones *et al.*, 1998] and can evaluate its diversification property using *variance* [Shahriari *et al.*, 2016].

As discussed in Section 4.1, a new sample (*i.e.,* the pair of the selected configuration and its performance) is obtained in

each iteration. GP can be updated using the samples obtained in previous iterations. Then we need to address how to search promising configurations using GP.

The algorithm underlying *UAHS* for searching promising configurations is outlined in Algorithm 1, and works between the exploitation mode and the exploration mode. In the exploitation mode, our algorithm focuses on known good parts of the configuration space; in the exploration mode, our algorithm tends to gather more information in the unknown parts. Since it is critical to make trade-off between exploitation and exploration [Hutter *et al.*, 2011; Cai *et al.*, 2016; Luo *et al.*, 2017], our algorithm utilizes a hyper-parameter $prob$ to balance exploitation and exploration. Initially, our algorithm constructs a candidate set CS consisting of configurations, and each configuration in CS is randomly sampled from the configuration space. Then, with probability $prob$, our algorithm selects the configuration with the largest EI evaluated by GP from CS using a sampling method called Best from Multiple Selections (BMS) [Cai, 2015]; otherwise (with probability $1 - prob$), it selects the configuration with the largest *variance* evaluated by GP from CS using the BMS sampling method. The BMS sampling method was proposed in the *FastVC* algorithm for solving the problem of minimum vertex cover, and is able to select a high-quality candidate with theoretical guarantee [Cai, 2015].

4.3 Prediction Component

Before introducing the prediction component, we first define a necessary operator f_l^t as follows: Given a VM type v_i and its historical demand $D_i = \{d_i^1, \ldots, d_i^T\}$, the operator f_l^t can be applied to D_i, and $f_l^t(D_i)$ denotes a sub-time series of D_i, with the first time index of t and the length of l, *i.e.,* $f_l^t(D_i) = \{d_i^t, d_i^{t+1}, \ldots, d_i^{t+l-1}\}$.

As discussed in Section 4.1, the prediction component is configurable, and its parameters can be specified by the selected configuration $Conf$. In the prediction component, *UAHS* needs to process the historical data for all VM types. For each VM type v_i, *UAHS* calls the function *Predict*, which takes the selected configuration $Conf$ and v_i's historical data D_i as inputs. The function *Predict* is shown in Algorithm 2.

For the function *Predict*, *UAHS* first utilizes the sliding window based time series analysis approach [Box *et al.*, 2015] to split the time series (*i.e.,* historical demand data) into a set of $T - l + 2$ sub-time series, denoted by $\{f_l^t(D_i) \mid 1 \leq t \leq T - l + 1\} \cup \{f_{l-1}^{T-l+2}(D_i)\}$, where l denotes the length of each sub-time series[2] and is an internal configurable parameter. For each sub-time series $f_l^t(D_i)$ of each VM type, the last demand d_i^{t+l-1} is treated as the label, and all other previous demands $f_l^t(D_i) \backslash d_i^{t+l-1}$ are regarded as the training set; then *UAHS* uses a *predictor*, which is trained on the training set, to predict the last demand, resulting in the predicted value of the last demand $\hat{d}_i^{t+l-1}$; after that, the prediction error of *predictor* for sub-time series $f_l^t(D_i)$, denoted by e_i^{t+l-1}, can be calculated by $d_i^{t+l-1} - \hat{d}_i^{t+l-1}$. After the prediction errors of *predictor* for all sub-time series are obtained,

[2]Except the last sub-time series, the length of the last sub-time series is $l - 1$, since y_i^* is unknown.

Algorithm 2: Function *Predict*

Input: $\boldsymbol{Conf}$: selected configuration;
$\boldsymbol{D_i}$: historical demand data for VM type v_i;
Output: $\boldsymbol{\hat{D}_i}$: predicted demand data for VM type v_i;
$\boldsymbol{\epsilon_i}$: distribution of prediction uncertainty;

specify l and *predictor*'s parameters by $Conf$;
for $t \leftarrow 1$ ***to*** $T - l + 1$ **do**
  $\hat{d}_i^{t+l-1} \leftarrow$ use *predictor* trained on $f_l^t(D_i) \backslash d_i^{t+l-1}$ to predict the demand at $t + l - 1$;
  $e_i^{t+l-1} \leftarrow d_i^{t+l-1} - \hat{d}_i^{t+l-1}$;
$\hat{y}_i \leftarrow$ use *predictor* trained on $f_{l-1}^{T-l+2}(D_i)$ to predict the real demand;
$\hat{D}_i \leftarrow \{\hat{d}_i^l, \hat{d}_i^{l+1}, \ldots, \hat{d}_i^T, \hat{y}_i\}$;
$\epsilon_i \leftarrow$ fit the distribution by GMM using $\{e_i^l, e_i^{l+1}, \ldots, e_i^T\}$;
return $\hat{D}_i$, ϵ_i;

we can fit the distribution of prediction uncertainty, denoted by ϵ_i, through Gaussian mixture model (GMM) [McLachlan and Peel, 2000] using all obtained prediction errors, since it is well recognized that GMM can reasonably approximate a wide class of distributions [Verbeek *et al.*, 2003].

UAHS treats *predictor* as a black-box tool, and *predictor* can be instantiated with any time series forecasting method.[3]

4.4 Optimization Component

As introduced in Section 4.1, for each time period $[t-1, t]$, the optimization component solves the corresponding VM provisioning problem within $[t-1, t]$. The details of the optimization component are described as follows.

For time period $[t-1, t]$, given 1) the predicted VM demands accumulated within $[t-1, t]$ for all VM types, denoted by $\hat{D}^t = \{\hat{d}_1^t, \ldots, \hat{d}_n^t\}$, 2) the distributions of prediction uncertainty for all VM types, denoted by $PD = \{\epsilon_1, \ldots, \epsilon_n\}$, and 3) the PM status P^{t-1} queried at time stamp $t-1$, the task is to solve the corresponding optimization problem listed in Equation 1. However, the original problem in Equation 1 does not consider prediction uncertainty; directly solving the original problem might result in inferior practical performance, because for time series forecasting methods, the prediction errors are inevitable [Wilder *et al.*, 2019]. Since *UAHS* utilizes probability distributions to model the prediction uncertainty, a novel idea is to leverage the distributions of prediction uncertainty during the optimization process.

As discussed before, for each VM type v_i, the prediction uncertainty can be expressed as a random variable u_i with the distribution ϵ_i. We can assume the (unknown) v_i's real demand $y_i^* = \hat{d}_i^t + u_i$, which can be seen as a random variable.[4]

[3]In this work, *UAHS* instantiates *predictor* using a time series decomposition based forecasting approach [Hyndman and Athanasopoulos, 2013] for its efficiency.

[4]Since y_i^* can only be non-negative integer, we treat the y_i^* as a discrete random variable, whose cumulative distribution function (CDF) equals to the continuous version.

Algorithm 3: Function *Optimize*

Input: $\boldsymbol{Conf}$: selected configuration;
$\hat{\boldsymbol{D}}^{\boldsymbol{t}}$: predicted VM demands accumulated with $[t-1, t]$ for all VM types;
$\boldsymbol{PD}$: distributions of prediction uncertainty for all VM types;
$\boldsymbol{P^{t-1}}$: PM status queried at time stamp $t-1$;
Output: $\boldsymbol{A^t}$: provisioning plan for time period $[t-1, t]$;

specify the trade-off probability b by $Conf$;
$a_{i,j} \leftarrow 0$ for all possible pairs of (i, j);
$FT \leftarrow$ the set of all feasible tuples (v_i, p_j);
while $FT \neq \varnothing$ **do**
  $CV \leftarrow \{v_i \mid \exists\, p_j \in P, (v_i, p_j) \in FT\}$;
  if *with probability* b **then**
    $v^* \leftarrow$ random VM type $v_i \in CV$;
  else
    $v^* \leftarrow$ VM type $v_i \in CV$ with the largest $score$;
  $CP \leftarrow \{p_j \mid (v^*, p_j) \in FT\}$;
  $p^* \leftarrow$ PM $p_j \in CP$ with the largest $utility$;
  $a_{i,j} \leftarrow a_{i,j} + 1$ where $v_i = v^*$ and $p_j = p^*$;
  remove any infeasible tuples (v_i, p_j) from FT;
$A^t \leftarrow \{a_{i,j} \mid i \in \{1, \ldots, n\}, j \in \{1, \ldots, m\}\}$;
return A^t;

Recall that the overall amount of the utilized CPU cores is $h = \sum_{i=1}^{n} \min\{x_i, y_i^*\} \cdot \alpha_i$; hence, h can also be regarded as a random variable which depends on $Y^* = \{y_1^*, \ldots, y_n^*\}$. Intuitively, the expected value of h, *i.e.,* $\mathbb{E}(h)$, represents the average amount of utilized CPU cores, across many independent realizations of real demands Y^*. Therefore, the optimization problem is to maximize $\mathbb{E}(h) = \sum_{i=1}^{n} \mathbb{E}(h_i)$.

Given a provisioning plan A and the PM status P^{t-1} queried at time stamp $t-1$, if a VM with VM type v_i can be provisioned on a PM p_j without violating any resource constraints in Equation 1, then the tuple (v_i, p_j) is feasible; otherwise, the tuple (v_i, p_j) is infeasible. The feasible tuple can be evaluated by two metrics (*i.e., score* and *utility*), for assessing the qualities of the related VM type and the related PM, respectively. We introduce the metrics as below.

Given a provisioning plan A and its corresponding objective h, if we provision a new VM with VM type v_i, resulting in a new objective h', the *score* of v_i is the increment to the objective *i.e.,* $\mathbb{E}(h') - \mathbb{E}(h)$, and how to calculate *score* of v_i is described in Lemma 1.

Lemma 1. *Given a provisioning plan A and VM type v_i, the* score *of v_i can be calculated as $score(v_i) = (1 - CDF_{y_i^*}(x_i)) \cdot \alpha_i$.*

Proof. We denote the notations h and h' as the objectives before and after adding a new VM with VM type v_i to provisioning plan A, respectively. Also, we denote the notations h_i and h_i' as the amounts of the utilized CPU cores for VM type v_i before and after adding a new VM with VM type v_i to provisioning plan A, respectively.

Since adding a new VM with VM type v_i would only impact the amount of the utilized CPU cores for VM type v_i, so it is easy to obtain $\mathbb{E}(h') - \mathbb{E}(h) = \mathbb{E}(h_i') - \mathbb{E}(h_i)$. The *score* of v_i can be calculated as follows.

$$
\begin{aligned}
score(v_i) &= \mathbb{E}(h') - \mathbb{E}(h) = \mathbb{E}(h_i') - \mathbb{E}(h_i) \\
&= \textstyle\sum_{k=0}^{+\infty} \min\{x_i + 1, k\} \cdot \alpha_i \cdot P(y_i^* = k) \\
&\quad - \textstyle\sum_{k=0}^{+\infty} \min\{x_i, k\} \cdot \alpha_i \cdot P(y_i^* = k) \\
&= \textstyle\sum_{k=x_i+1}^{+\infty} P(y_i^* = k) \cdot \alpha_i \\
&= P(y_i^* \geq x_i + 1) \cdot \alpha_i \\
&= (1 - P(y_i^* \leq x_i)) \cdot \alpha_i \\
&= (1 - CDF_{y_i^*}(x_i)) \cdot \alpha_i
\end{aligned}
$$

□

For any PM p_j, the *utility* of p_j measures the remaining resource capacity [Hbaieb *et al.*, 2017], and is calculated as $utility(p_j) = CR_j \cdot MR_j \cdot SR_j$, where $CR_j = (\sum_{i=1}^{n} a_{i,j} \cdot \alpha_i)/C_j$, $MR_j = (\sum_{i=1}^{n} a_{i,j} \cdot \beta_i)/M_j$ and $SR_j = (\sum_{i=1}^{n} a_{i,j} \cdot \gamma_i)/S_j$. Actually, CR_j, MR_j and SR_j represent the utility status of CPU cores, memory and storage on PM p_j, respectively. The reason why *UAHS* prefers to select a PM with larger *utility* is based on the intuition that in the optimal provisioning plan all PMs tend to be fully utilized in practice [Hbaieb *et al.*, 2017].

To make our function *Optimize* efficient, inspired by the high efficiency of the two-mode heuristic search framework [Hoos and Stützle, 2004; Cai *et al.*, 2013; Luo *et al.*, 2015; Luo *et al.*, 2019], we design a new two-mode heuristic search algorithm to optimize the provisioning plan. The function *Optimize* introduces a trade-off probability b (which is an internal configurable parameter) to effectively balance the greediness and the randomness, in order to better escape local optima during the search process.

The function *Optimize* underlying *UAHS* is listed in Algorithm 3. In the beginning, the set of all feasible tuples FT is initialized. Then the function *Optimize* conducts the optimization process iteratively. In each iteration, the function *Optimize* switches between the random mode and the greedy mode, in order to select a tuple (v^*, p^*) from FT, where v^* and p^* denote the VM type and the PM chosen in this iteration, respectively. Regarding choosing the VM type v^*, with probability b, the function *Optimize* works in the random mode, *i.e.,* selecting a VM type randomly as v^*; otherwise (with probability $1-b$), the function *Optimize* works in the greedy mode, *i.e.,* selecting the VM type with the largest *score* as v^*. Regarding choosing the PM p^*, the function *Optimize* selects the PM with the largest *utility* as p^*. Then the provisioning plan is updated accordingly. At the end of each iteration, those infeasible tuples would be removed from FT. The iteration process is terminated once FT becomes empty. Once the iteration process terminates, the current provisioning plan is reported as the output of the function *Optimize*.

5 Experiments

To study the performance of *UAHS*, we conduct extensive experiments on two public datasets and an industrial dataset to compare *UAHS* against 8 state-of-the-art competitors.

Competitors	Azure-2017	Azure-2019	Industrial
	avg. r ± SD time	avg. r ± SD time	avg. r ± SD time
LR+ACO	0.684 ± 0.184 1177.7	0.692 ± 0.118 5204.5	0.724 ± 0.055 3353.5
TSDec+ACO	0.711 ± 0.162 1260.8	0.686 ± 0.140 4722.7	0.701 ± 0.066 3458.8
AutoARIMA+ACO	0.771 ± 0.146 3161.2	0.760 ± 0.121 9637.8	0.746 ± 0.043 12208.4
LSTM+ACO	0.763 ±0.101 9166.5	0.735 ±0.113 20362.3	0.729 ±0.032 19549.3
UCM+ACO	0.723 ± 0.148 1162.6	0.752 ± 0.118 4976.6	0.736 ± 0.046 2810.5
Prophet+ACO	0.762 ± 0.139 1431.7	0.696 ± 0.154 4696.0	0.738 ± 0.040 3272.6
Decision-NN	0.792 ± 0.106 970.9	0.762 ± 0.128 4504.9	0.759 ± 0.032 3193.2
Semi-direct	0.788 ± 0.118 1139.6	0.764 ± 0.087 3946.2	0.745 ± 0.058 3590.3
UAHS-alt1	0.791 ± 0.144 6.7	0.784 ± 0.124 11.0	0.776 ± 0.049 10.5
UAHS-alt2	0.764 ± 0.122 355.2	0.723 ± 0.116 517.3	0.719 ± 0.052 542.6
UAHS	**0.833 ± 0.092** 336.5	**0.821 ± 0.084** 548.0	**0.813 ± 0.044** 523.7

Table 1: Results of *UAHS*, *UAHS-alt1*, *UAHS-alt2* and their competitors on all datasets.

5.1 Datasets

In order to evaluate the performance of *UAHS*, we utilize two public datasets[5] [Cortez *et al.*, 2017] collected from Microsoft Azure, dubbed `Azure-2017` and `Azure-2019`. After obtaining the `Azure-2017` and `Azure-2019` datasets, we pre-process, filter and aggregate those two datasets in order to make them applicable in our experiments. After pre-processing, filtering and aggregating those two datasets, the `Azure-2017` dataset has 110 different VM types and contains a representative trace of VM workload of Microsoft Azure across 30 consecutive days in 2017; the `Azure-2019` dataset has 150 different VM types and includes a similar trace of Microsoft Azure across 30 consecutive days in 2019. For each processed dataset of `Azure-2017` and `Azure-2019`, the demand for each VM type is recorded every 4 hours, so it has 42 time stamps per week and 180 time stamps in total across the 30 days. In order to capture the weekly characteristics, we construct 42 instances by taking all sub-time series in length of 139 (*i.e.,* $180 - 42 + 1$) time stamps using the sliding window based time series analysis approach [Box *et al.*, 2015], each of which has the demand on the last time stamp as label. Since both the `Azure-2017` and `Azure-2019` datasets lack the PM status, for each instance in the `Azure-2017` and `Azure-2019` datasets, the PM status is generated synthetically to simulate a practical setup.

Besides those two public datasets described above (*i.e.,* the `Azure-2017` and `Azure-2019` datasets), we additionally adopt an industrial dataset called `Industrial`, which is gathered from Microsoft Azure, to evaluate the practical performance of *UAHS*. The `Industrial` dataset records a recent trace of VM workloads on the cloud platform with a subset of VM types. In order to be aligned with the `Azure-2017` and `Azure-2019` datasets, the `Industrial` dataset is processed similarly to construct 42 instances.

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

max_iter	Azure-2017	Azure-2019	Industrial
	avg. r ± SD time	avg. r ± SD time	avg. r ± SD time
1	0.791 ± 0.144 6.7	0.784 ± 0.124 11.0	0.776 ± 0.049 10.5
10	0.811 ± 0.106 67.2	0.810 ± 0.109 109.6	0.781 ± 0.045 104.8
25	0.826 ±0.094 168.2	0.815 ±0.094 274.0	0.799 ±0.050 261.9
50	0.833 ± 0.092 336.5	0.821 ± 0.084 548.0	0.813 ± 0.044 523.7
100	0.836 ± 0.081 673.0	0.825 ± 0.083 1096.0	0.810 ± 0.054 1047.5
150	**0.839 ± 0.081** 1009.5	0.826 ± 0.082 1644.0	0.814 ± 0.054 1571.3
200	0.835 ± 0.086 1346.0	**0.830 ± 0.079** 2192.0	**0.816 ± 0.055** 2095.1

Table 2: Results of *UAHS* with different hyper-parameter settings of $\boldsymbol{max\_iter}$ on all datasets.

5.2 Competitors

UAHS is compared against 8 state-of-the-art competitors, including 6 straightforward two-stage methods, *Decision-NN* [Wilder *et al.*, 2019] and *Semi-direct* [Demirović *et al.*, 2019a]. For all competitors, the ACO algorithm [Zhao *et al.*, 2018] is adopted as the optimization algorithm, since ACO exhibits the state-of-the-art performance in solving the deterministic VM provisioning problem [Zhao *et al.*, 2018].

We integrate *ACO* with 6 effective time series forecasting methods, including 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], to construct 6 straightforward two-stage methods, dubbed *LR+ACO*, *TSDec+ACO*, *AutoARIMA+ACO*, *LSTM+ACO*, *UCM+ACO* and *Prophet+ACO*, respectively. As introduced in Section 3, *Decision-NN* [Wilder *et al.*, 2019] and *Semi-direct* [Demirović *et al.*, 2019a] are two state-of-the-art methods for solving the problem of prediction with optimization and can be applicable to the PreVMP problem.

5.3 Experimental Setup

In this work, all experiments were conducted on a machine with Intel Xeon E5-2673 CPU and 256 GB memory, running GNU/Linux. For *UAHS*, the hyper-parameter settings of max_iter and $prob$ are set to 50 and 0.9, respectively. The effects of different hyper-parameter settings of max_iter and $prob$ are discussed in Section 5.4. For each method on each dataset, we report the average utilization ratio ('avg. r'), the standard deviation of the utilization ratios across all instances ('SD'), and the average run time ('time') in second. For each dataset, we use **boldface** to indicate the best results.

5.4 Experimental Results

Comparisons against state-of-the-art competitors. The comparative results of *UAHS* and its 8 state-of-the-art competitors on all 3 datasets are presented in Table 1. It is clear

$prob$	Azure-2017 avg. r $\pm$ SD time	Azure-2019 avg. r $\pm$ SD time	Industrial avg. r $\pm$ SD time
0.1	0.827 ± 0.091 334.9	0.821 ± 0.088 558.6	0.809 ± 0.047 520.9
0.2	0.832 ± 0.086 335.2	0.820 ± 0.077 563.7	0.805 ± 0.041 531.8
0.3	0.830 ± 0.079 338.7	0.822 ± 0.090 570.0	0.808 ± 0.052 522.7
0.4	0.812 ± 0.085 324.6	0.821 ± 0.090 549.2	0.810 ± 0.053 530.6
0.5	0.820 ± 0.084 338.7	0.818 ± 0.095 553.1	0.804 ± 0.047 532.8
0.6	0.833 ± 0.092 314.3	0.811 ± 0.108 569.6	0.803 ± 0.045 532.8
0.7	0.827 ± 0.100 338.9	$\mathbf{0.825 \pm 0.081}$ 566.1	0.801 ± 0.049 532.3
0.8	0.826 ± 0.089 336.1	0.815 ± 0.094 547.8	0.804 ± 0.051 521.2
0.9	0.833 ± 0.092 336.5	0.821 ± 0.084 548.0	$\mathbf{0.813 \pm 0.044}$ 523.7
1.0	$\mathbf{0.837 \pm 0.087}$ 339.0	0.823 ± 0.084 553.0	0.799 ± 0.059 529.7

Table 3: Results of *UAHS* with different hyper-parameter settings of $\boldsymbol{prob}$ on all datasets.

that *UAHS* stands out as the best method in terms of average utilization ratio and average run time. In particular, on all 3 datasets *UAHS* achieves the utilization ratio more than 0.8, while the figures for its all competitors are less than 0.8. When we focus on the metric of average run time, *UAHS* runs much faster than its all competitors. The results in Table 1 indicate both the effectiveness and the efficiency of *UAHS*.

Effectiveness of Bayesian optimization. To evaluate the effectiveness of Bayesian optimization underlying *UAHS*, we modify *UAHS* by removing the Bayesian optimization process, resulting in an alternative version called *UAHS-alt1*. The comparative results of *UAHS* and *UAHS-alt1* are reported in Table 1. The results present that *UAHS* achieves higher average utilization ratio than *UAHS-alt1*, demonstrating the effectiveness of Bayesian optimization underlying *UAHS*.

Effectiveness of prediction uncertainty modeling. To assess the effectiveness of prediction uncertainty modeling, we develop another alternative version of *UAHS* which works without prediction uncertainty modeling, dubbed *UAHS-alt2*. The comparative results of *UAHS* and *UAHS-alt2* are shown in Table 1, and *UAHS* performs much better than *UAHS-alt2* in terms of average utilization ratio, indicating the effectiveness of prediction uncertainty modeling underlying *UAHS*.

Robustness of *UAHS*. The results of *UAHS* with different hyper-parameter settings of max_iter and $prob$ are reported in Tables 2 and 3, respectively. Table 2 presents that *UAHS* achieves better performance with relatively larger max_iter. Besides, Table 3 demonstrates that *UAHS* exhibits stable performance with different hyper-parameter settings of $prob$. The results in Tables 2 and 3 provide clear evidence that *UAHS* shows robustness and can achieve state-of-the-art performance with different hyper-parameter settings.

6 Applications in Practice

Pre-Provisioning Service (PPS) in Microsoft Azure brings VM deployment reliability and latency benefits by creating pre-provisioned VMs. In practice, we have successfully applied our *UAHS* approach to PPS in Microsoft Azure and significantly improved the performance of the VM provisioning on the cloud platform. PPS obtains demand predictions from *UAHS* through an intermediary service system called Resource Central [Cortez *et al.*, 2017]. In original VM provisioning process, it could cost fairly long time to provision a VM from scratch before the VM is accessible. With *UAHS*, the original VM provisioning process could be done in reduced time via the utilization of pre-provisioned VMs. Based on the data collected from the cloud platform before and after the deployment of *UAHS*, about 93% of customer requests that prediction is targeted for are successfully served by pre-provisioned VMs. Moreover, the median of the VM provisioning time has been reduced by around 42%, indicating that *UAHS* can bring practical benefits in real-world applications. Furthermore, part of *UAHS* has been successfully applied in Microsoft Office, like the projects of B2 Autopilot Container Allocation and AZSC Packing, and considerable performance improvements are obtained.

7 Conclusions

In cloud computing, virtual machine (VM) provisioning is a common and critical problem and it is advisable to provision VMs ahead for customer experience. In this work, we formulated the predictive VM provisioning (PreVMP) problem, and proposed a novel approach, dubbed Uncertainty-Aware Heuristics Search (*UAHS*), for solving the PreVMP problem. *UAHS* models and utilizes the prediction uncertainty to conduct optimization. Moreover, *UAHS* leverages Bayesian optimization to interact the prediction component and the optimization component, to achieve performance improvement. Extensive experiments on two public datasets and an industrial dataset show that *UAHS* performs much better than state-of-the-art competitors. Furthermore, *UAHS* has been successfully applied in Microsoft Azure and brought practical benefits in real-world applications.

Acknowledgements

We would like to thank Dongmei Zhang, Girish Bablani, Yingnong Dang, Gil Lapid Shafriri, Murali Chintalapati, Saravanakumar Rajmohan, Jim Kleewein, Sushant Rewaskar, Thomas Moscibroda and Marcus Fontoura for their great support and sponsorship. We would also like to thank Daud Howlader, Anusha Kowdeed, Chandramouleswaran Ravichandran and Bowen Xu 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.

References

[Box *et al.*, 2015] George E. P. Box, Gwilym M. Jenkins, Gregory C. Reinsel, and Greta M. Ljung. *Time Series Analysis: Forecasting and Control*. Wiley, 2015.

[Cai *et al.*, 2013] Shaowei Cai, Kaile Su, Chuan Luo, and Abdul Sattar. NuMVC: An efficient local search algorithm

for minimum vertex cover. *Journal of Artificial Intelligence Research*, 46:687–716, 2013.

[Cai *et al.*, 2016] Shaowei Cai, Chuan Luo, Jinkun Lin, and Kaile Su. New local search methods for partial MaxSAT. *Artificial Intelligence*, 240:1–18, 2016.

[Cai, 2015] Shaowei Cai. Balance between complexity and quality: Local search for minimum vertex cover in massive graphs. In *Proceedings of IJCAI 2015*, pages 747–753, 2015.

[Cortez *et al.*, 2017] Eli Cortez, Anand Bonde, Alexandre Muzio, Mark Russinovich, Marcus Fontoura, and Ricardo Bianchini. Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms. In *Proceedings of SOSP 2017*, pages 153–167, 2017.

[Demirović *et al.*, 2019a] Emir Demirović, Peter J. Stuckey, James Bailey, Jeffrey Chan, Chris Leckie, Kotagiri Ramamohanarao, and Tias Guns. An investigation into prediction + optimisation for the knapsack problem. In *Proceedings of CPAIOR 2019*, pages 241–257, 2019.

[Demirović *et al.*, 2019b] Emir Demirović, Peter J. Stuckey, James Bailey, Jeffrey Chan, Christopher Leckie, Kotagiri Ramamohanarao, and Tias Guns. Predict+optimise with ranking objectives: Exhaustively learning linear functions. In *Proceedings of IJCAI 2019*, pages 1078–1085, 2019.

[Demirović *et al.*, 2020] Emir Demirović, Peter J. Stuckey, James Bailey, Jeffrey Chan, Christopher Leckie, Kotagiri Ramamohanarao, and Tias Guns. Dynamic programming for predict+optimise. In *Proceedings of AAAI 2020*, 2020.

[Durbin and Koopman, 2012] James Durbin and Siem Jan Koopman. *Time Series Analysis by State Space Methods*. Oxford University Press, 2012.

[Elmachtoub and Grigas, 2017] Adam N. Elmachtoub and Paul Grigas. Smart "Predict, then optimize". *CoRR*, abs/1710.08005, 2017.

[Hbaieb *et al.*, 2017] Ameni Hbaieb, Mahdi Khemakhem, and Maher Ben Jemaa. Using decomposition and local search to solve large-scale virtual machine placement problems with disk anti-colocation constraints. In *Proceedings of AICCSA 2017*, pages 688–695, 2017.

[Hoos and Stützle, 2004] Holger H. Hoos and Thomas Stützle. *Stochastic Local Search: Foundations & Applications*. Elsevier / Morgan Kaufmann, 2004.

[Hutter *et al.*, 2011] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Proceedings of LION 2011*, pages 507–523, 2011.

[Hyndman and Athanasopoulos, 2013] Rob J. Hyndman and George Athanasopoulos. *Forecasting: Principles and Practice*. OTexts, 2013.

[Jones *et al.*, 1998] Donald R. Jones, Matthias Schonlau, and William J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global Optimization*, 13(4):455–492, 1998.

[Luo *et al.*, 2015] Chuan Luo, Shaowei Cai, Wei Wu, Zhong Jie, and Kaile Su. CCLS: An efficient local search algorithm for weighted maximum satisfiability. *IEEE Transactions on Computers*, 64(7):1830–1843, 2015.

[Luo *et al.*, 2017] Chuan Luo, Shaowei Cai, Kaile Su, and Wenxuan Huang. CCEHC: An efficient local search algorithm for weighted partial maximum satisfiability. *Artificial Intelligence*, 243:26–44, 2017.

[Luo *et al.*, 2019] Chuan Luo, Holger H. Hoos, Shaowei Cai, Qingwei Lin, Hongyu Zhang, and Dongmei Zhang. Local search with efficient automatic configuration for minimum vertex cover. In *Proceedings of IJCAI 2019*, pages 1297–1304, 2019.

[Mandi *et al.*, 2020] Jaynta Mandi, Emir Demirović, Peter. J Stuckey, and Tias Guns. Smart predict-and-optimize for hard combinatorial optimization problems. In *Proceedings of AAAI 2020*, 2020.

[Mao and Humphrey, 2012] Ming Mao and Marty Humphrey. A performance study on the VM startup time in the cloud. In *Proceedings of IEEE CLOUD 2012*, pages 423–430, 2012.

[McLachlan and Peel, 2000] Geoffrey McLachlan and David Peel. *Finite Mixture Models*. Wiley, 2000.

[Mockus, 1989] Jonas Mockus. *Bayesian Approach to Global Optimization: Theory and Applications*. Kluwer Academic Publishers, 1989.

[Shahriari *et al.*, 2016] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P. Adams, and Nando de Freitas. Taking the human out of the loop: A review of Bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2016.

[Taylor and Letham, 2018] Sean J. Taylor and Benjamin Letham. Forecasting at scale. *The American Statistician*, 72(1):37–45, 2018.

[Verbeek *et al.*, 2003] Jakob J. Verbeek, Nikos A. Vlassis, and Ben J. A. Kröse. Efficient greedy learning of Gaussian mixture models. *Neural Computation*, 15(2):469–485, 2003.

[Wang *et al.*, 2015] Changjun Wang, Weidong Ma, Tao Qin, Xujin Chen, Xiaodong Hu, and Tie-Yan Liu. Selling reserved instances in cloud computing. In *Proceedings of IJCAI 2015*, pages 224–231, 2015.

[Wilder *et al.*, 2019] Bryan Wilder, Bistra Dilkina, and Milind Tambe. Melding the data-decisions pipeline: Decision-focused learning for combinatorial optimization. In *Proceedings of AAAI 2019*, pages 1658–1665, 2019.

[Zhang *et al.*, 2014] Zhaoning Zhang, Ziyang Li, Kui Wu, Dongsheng Li, Huiba Li, Yuxing Peng, and Xicheng Lu. VMThunder: Fast provisioning of large-scale virtual machine clusters. *IEEE Transactions on Parallel and Distributed Systems*, 25(12):3328–3338, 2014.

[Zhao *et al.*, 2018] Hui Zhao, Jing Wang, Feng Liu, Quan Wang, Weizhan Zhang, and Qinghua Zheng. Power-aware and performance-guaranteed virtual machine placement in the cloud. *IEEE Transactions on Parallel and Distributed Systems*, 29(6):1385–1400, 2018.