Predictive Job Scheduling under Uncertain Constraints in Cloud Computing

Hang Dong[1] , **Boshi Wang**[1,3] , **Bo Qiao**[1] and **Wenqian Xing**[1] , **Chuan Luo**[1,*] , **Si Qin**[1] ,
Qingwei Lin[1,*] , **Dongmei Zhang**[1] , **Gurpreet Virdi**[2] , **Thomas Moscibroda**[2]
[1]Microsoft Research, China
[2]Microsoft Azure, United States
[3]The Ohio State University, United States
{hangdong, boqiao, v-wenxing, chual, siqin, qlin, dongmeiz}@microsoft.com
{v-boshiwang, gurvir, moscitho}@microsoft.com

Abstract

Capacity management has always been a great challenge for cloud platforms due to massive, heterogeneous on-demand instances running at different times. To better plan the capacity for the whole platform, a class of cloud computing instances have been released to collect computing demands beforehand. To use such instances, users are allowed to submit jobs to run for a pre-specified uninterrupted duration in a flexible range of time in the future with a discount compared to the normal on-demand instances. Proactively scheduling those pre-collected job requests considering the capacity status over the platform can greatly help balance the computing workloads along time. In this work, we formulate the scheduling problem for these pre-collected job requests under uncertain available capacity as a Prediction + Optimization problem with uncertainty in constraints, and propose an effective algorithm called *Controlling under Uncertain Constraints (CUC)*, where the predicted capacity guides the optimization of job scheduling and job scheduling results are leveraged to improve the prediction of capacity through Bayesian optimization. The proposed formulation and solution are commonly applicable for proactively scheduling problems in cloud computing. Our extensive experiments on three public, industrial datasets shows that *CUC* has great potential for supporting high reliability in cloud platforms.

1 Introduction

The cloud computing paradigm manages resources as a shared pool of configurable virtual machines (VMs), and offers an opportunity for users to allocate or release VMs whenever necessary. Deploying workloads on large-scale cloud platforms, such as Amazon Web Service, Microsoft Azure, Google Cloud, has thus gained significant popularity in recent years. The total market of worldwide public cloud will grow to $364B in 2022 based on Gartner forecasts[1].

For cloud platforms, it remains a huge challenge to properly allocate the computing resources with a high utilization rate and with a high reliability level at the same time due to the plug-and-play and pay-as-you-go styles in cloud computing [Mesbahi *et al.*, 2018; Gawali and Shinde, 2018]. From an operational perspective, the total demand in a public cloud is in the order of millions of VMs per day. This high request rate must be satisfied by the platform while maintaining fast response time, and therefore would result in a waste of resources along the fluctuations between peaks and valleys of customers' demands.

To reduce the resource waste due to demand fluctuations, a stream of VM products with pre-collected demands has been released by major cloud platforms, where customers can book resources in a coming time window by specifying the required VM type, the duration in hours, as well as the earliest and latest time for the job to start. Collecting the demands for these VMs and scheduling these jobs properly can greatly shave peaks and fill valleys of resource utilization along time, and can therefore achieve a higher utilization rate and higher overall revenue for the platform.

However, it is challenging to intelligently schedule these pre-collected job requests due to the uncertain available resources (normally referred to as capacity in cloud computing) in the future and not affect the system's high reliability level at the same time. More specifically, deploying jobs violating the capacity will largely reduce system reliability in cloud computing. To this end, we resort to a Prediction + Optimization framework to tackle this difficulty in this paper. More specifically, we provide a general formulation for the scheduling problem on the pre-collected job requests under uncertain capacity, and propose a solution called *Controlling under Uncertain Constraints (CUC)* for this Prediction + Optimization problem. With *CUC*, we can flexibly control the violation level on capacity constraints in the application, and maximize the overall utilization for the pre-collected job requests under the pre-specified violation level. Extensive experiments on three public datasets from Microsoft Azure show that *CUC* can strictly control the violation level to the available capac-

[*]Corresponding authors

[1]https://www.gartner.com/en/newsroom/press-releases/2020-07-23-gartner-forecasts-worldwide-public-cloud-revenue-to-grow-6point3-percent-in-2020

ity and outperforms existing state-of-the-art methods .

We list the main contributions of this work as follows:

- We provide an effective formulation for a class of scheduling problems with uncertain constraints to be predicted, and furthermore introduce a violation level on the constraints to be controlled.
- We propose *CUC*, a novel approach for solving the job scheduling problem under uncertain constraints, and can flexibly control the violation level on uncertain constraints to fulfill system requirements.
- *CUC* is demonstrated to be able to strictly control the violation level on the uncertain constraints with industrial datasets on extensive experiments and meet rigorous reliability requirements in cloud computing.

2 Related Work

The pre-collected job scheduling problem in cloud computing can be treated as a variant of the general machine scheduling problem [Chen *et al.*, 1998]. Even when the future capacity is known, this problem is known to be an NP-hard problem and usually formulated as a mixed integer programming (MIP) problem [Fanjul-Peyro *et al.*, 2017]. Solving this type of problems often needs to resort to heuristics or other approximation algorithms [Fleszar and Hindi, 2018].

Another type of problem related to the pre-collected job scheduling problem in cloud computing is the general resource allocation problem. Most of the works on resource allocation focus on considering demand or workload uncertainty with known capacity [Calzarossa *et al.*, 2019]. However, in many application scenarios, the capacity itself may suffer from uncertainty, such as the case in appointment scheduling and room allocation in health-care systems [Gupta and Denton, 2008; Zhou *et al.*, 2020]. Rather than just assuming the future available capacity to be known, resource (capacity) prediction based planning methods have been explored both by academia [Huang *et al.*, 2018] and industry. Further more, the uncertainty considered in these works is in the form of a fixed probability distribution, which we can hardly well capture in reality. Instead, we usually need to develop a statistical distribution of the unknown value from historical data. Therefore, the actual performance of the optimization problems under this 'uncertain uncertainty' would largely depend on the prediction performance on the uncertain value.

The literature [Verderame *et al.*, 2010] has reviewed several ways to take into parameter uncertainty in an explicit manner, including two-stage stochastic programming, parametric programming, fuzzy programming, chance constraint programming, robust optimization, and conditional value-at-risk frameworks. Moreover, a group of methods have been raised recently dealing with the Prediction + Optimization problem [Demirovic *et al.*, 2019; Luo *et al.*, 2020; Mandi *et al.*, 2020], where the optimization objective contains unknown parameters that needs to be predicted before optimization. Currently, these works have not considered the uncertainty in constraints and are therefore not applicable for the pre-collected job scheduling problem under uncertain capacity constraints considered in this work.

3 Problem Formulation

In this section, we formally formulate the job scheduling problem under uncertain capacity constraints as follows.

Definitions on job requests. Given a set of pre-collected job requests $B = \{b_1, \dots, b_N\}$, each job request includes basic requirements for the job to run. Specifically, for a job request $b_i \in B, i \in \{1, \dots, N\}$, it can be represented as $b_i = (c_i, d_i, e_i, l_i)$, where c_i is the requested resource capacity for job i, d_i is the duration for the job, e_i and l_i are the earliest start time and latest start time for the job, respectively. In practice, the most valuable computing resource is the CPU cores in cloud computing [Luo *et al.*, 2020], and it can be any other type of resources according to the specific application. Here d_i is restricted to be an integer hours of duration, e_i and l_i are restricted to be the beginning of each hour, and $e_i < l_i$ for $i \in \{1, \dots, N\}$. For each of the job requests, it is only valid between its earliest start time and latest start time and should be deployed during this period, or it is treated as failed otherwise.

Definitions on predicted capacity. From the view of the supply side, the cloud platform can only deploy these pre-collected job requests on the remaining capacity, which is obtained by subtracting the capacity occupied by on-demand job requests from the whole available resource pool in the cloud platform. Therefore, due to the uncertainty on the on-demand workloads, the available capacity for deployment of these pre-collected jobs also suffers from the uncertainty, and needs to be predicted at a regular basis for scheduling these pre-collected jobs. For each scheduling scope containing T hours, the predicted capacity is a time series $\hat{A} = \{\hat{a}_1, \hat{a}_2, ..., \hat{a}_T\}$ where $\hat{a}_i, i \in \{1, ..., T\}$ is the predicted distribution of available capacity for job requests measured in the unit of CPU cores.

Problem formulation. The job scheduling process is running in a regular basis, covering the time period of T time units (e.g., $T = 24$ hours in the application scenario considered in this work). Each scheduling run needs to determine for the following T time units, which job requests should start at the beginning of each time unit. The process can be formally formulated as a combinatorial optimization problem. The decision variables are $\{X_{it}\}$, $i = 1, ..., N$, $t = 1, ..., T$, where $X_{it} = 1$ if job request i is scheduled to start at time t, and $X_{it} = 0$ otherwise. The goal of the job scheduling practice is to maximize the overall computing resources multiplied by the duration time for the deployed job requests and try not to affect the on-demand workloads. This combinatorial optimization problem formulation is as follows:

$$
\begin{aligned}
\text{Maximize} \quad & u = \sum_{i=1}^{N} \sum_{t=1}^{T} c_i d_i X_{it} \\
\text{s.t.} \quad & \textstyle\sum_{t=1}^{T} X_{it} \leq 1 && \text{for } 1 \leq i \leq N \\
& \textstyle\sum_{t<e_i \vee t>l_i} X_{it} = 0 && \text{for } 1 \leq i \leq N \\
& \textstyle\sum_{i=1}^{N} \sum_{t'=t-d_i}^{t} c_i X_{it'} \leq a_t && \text{for } 1 \leq t \leq T \\
& X_{it} \in \{0, 1\} && \text{for } 1 \leq i \leq N, 1 \leq t \leq T
\end{aligned}
\tag{1}
$$

where a_t is the real available capacity for the pre-collected jobs at time t, $t \in \{1, ..., T\}$. Note that $A = \{a_1, ..., a_T\}$ remains unknown when solving this scheduling problem, we can only estimate A with $\hat{A}$ where $\hat{A}$ is the output of a specific time series prediction method using historical capacity data.

In essence, the above combinatorial optimization problem cannot be directly solved since A is unknown. In this case of not knowing the real available capacity, we cannot guarantee any scheduled plan would not affect the on-demand workloads unless no request for the pre-collected jobs is actually deployed. Therefore, we propose a new formulation (2) which allows a specific low level of impact on the normal on-demand workloads, and tries to control the impact so that it is below this specific level. The goal has thus turned to maximizing the benefits from these scheduled pre-collected jobs on the condition that the violation frequency for on-demand capacity is controlled under a pre-specified level p by adjusting the capacity constraints in the optimization problem. In an ideal situation, the predicted distribution $\hat{A}$ would be the true distribution of A, then the following optimization problem would be sufficient to achieve our goal:

$$
\begin{aligned}
&\text{Maximize} \quad u = \sum_{i=1}^{N}\sum_{t=1}^{T} c_i d_i X_{it} \\
&\text{s.t.} \quad \sum\nolimits_{t=1}^{T} X_{it} \le 1 && \text{for } 1 \le i \le N \\
&\quad \sum\nolimits_{t<e_i \vee t>l_i} X_{it} = 0 && \text{for } 1 \le i \le N \\
&\quad \sum\nolimits_{i=1}^{N} \sum\nolimits_{t'=t-d_i}^{t} c_i X_{it'} \le \tilde{a}_t && \text{for } 1 \le t \le T \\
&\quad \tilde{a}_t = \sup\{\gamma | \mathbb{P}(\hat{a}_t \ge \gamma) \ge 1-p\} && \text{for } 1 \le t \le T \\
&\quad X_{it} \in \{0,1\} && \text{for } 1 \le i \le N, 1 \le t \le T
\end{aligned} \tag{2}
$$

However, there is always error between our predicted distribution $\hat{A}$ and the true distribution of A. Therefore, we still need to improve our prediction based on the result from the optimization step to achieve better performance. Formally, we use a two-layer optimization formulation for this scheduling problem with predicted uncertainty:

$$
\begin{aligned}
&\text{Minimize} \quad \max(v-p, 0) \\
&\text{s.t.} \quad X = \arg\max \sum\nolimits_{i=1}^{N} \sum\nolimits_{t=1}^{T} c_i d_i X_{it} \\
&\quad \sum\nolimits_{t=1}^{T} X_{it} \le 1 && \text{for } 1 \le i \le N \\
&\quad \sum\nolimits_{t<e_i \vee t>l_i} X_{it} = 0 && \text{for } 1 \le i \le N \\
&\quad \sum\nolimits_{i=1}^{N} \sum\nolimits_{t'=t-d_i}^{t} c_i X_{it'} \le \tilde{a}_t && \text{for } 1 \le t \le T \\
&\quad \tilde{a}_t = \sup\{\gamma | \mathbb{P}(\hat{a}_t \ge \gamma) \ge 1-p\} && \text{for } 1 \le t \le T \\
&\quad X_{it} \in \{0,1\}
\end{aligned} \tag{3}
$$

where p is the pre-specified violation level, and v is the actual violation rate comparing the scheduled capacity with the true capacity:

$$
v = \sum_{t=1}^{T} \mathbb{1}\left(\sum_{i=1}^{N}\sum_{t'=t-d_i}^{t} c_i X_{it'} > a_t\right) / T. \tag{4}
$$

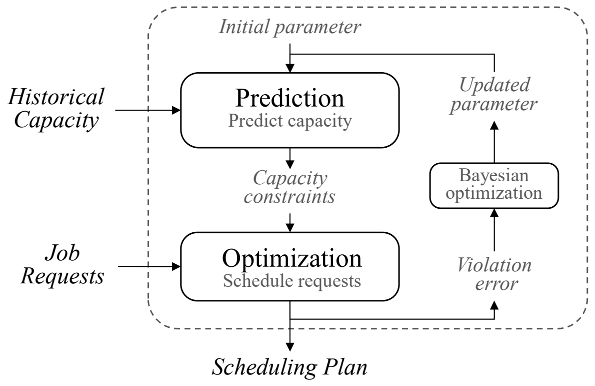

Figure 1: Overall design of the proposed *CUC* algorithm.

In this formulation (3), the objective is to 1) achieve the highest utilization under the capacity constraints that is predicted to meet the pre-specified violation level p; 2) improve the predictive performance on future capacity according to the performance of the optimization result in the training stage, which will be further explained in Section 4.

4 Controlling under Uncertain Constraints

In this section, we introduce *Controlling under Uncertain Constraints (CUC)*, our scheduling algorithm under uncertain constraints for the Prediction + Optimization problem for pre-collected job requests.

4.1 Overall Design of *CUC*

The illustration of the overall design of *CUC* is shown in Figure 1. As stated before, we try to schedule the pre-collected job requests in a coming period under unknown capacity. To deal with this uncertainty, the capacity is predicted using historical capacity so as to get an estimation on the constraints under which the job requests should be scheduled. Then the predicted distribution of future capacity is used to determine the capacity constraints according to the pre-specified violation level for the inner level optimization in (3), which gives a scheduled plan for the job requests. To improve the prediction as well as the scheduling performance, the actual violation rate and the error between that and the pre-specified violation level are calculated and sent to the Bayesian optimization module to update parameters for the prediction model. After several iterations when the violation error is well controlled, we get the optimized scheduling plan as well as the improved parameters for the prediction model in the specific dataset.

4.2 Technical Details of *CUC*

More specifically, the use of the proposed *CUC* can be divided into the training stage and the testing stage. In the training stage of *CUC*, the major objective is to learn better prediction model parameters for real prediction applications. The training procedures of *CUC* are summarized in Algorithm 1.

To obtain the predicted distribution for the capacity in the next T time units, we adopt the time series prediction model based on sliding windows [Box and Jenkins, 1976] and fit

Algorithm 1: Training Procedures of *CUC*

```
Input: Historical capacity $A^-$, Requests
       $B = \{b_i\}_{i \in \{1,...,N\}}$, Violation level $p$,
       Converge threshold $\gamma$, Max iteration steps $k$;
Output: Schedule $\{X_{it}\}_{i \in \{1,...,N\}, t \in \{1,...,T\}}$,
        Prediction parameters $\theta$;
1 Set iter_num = 1, initialize parameters $\theta = \theta_0$ for the
  prediction model;
2 while $\max(v - p, 0) > \gamma$ or iter_num $\leq k$ do
3     Get predicted distribution $\hat{A}$ with $\theta$ and historical
      capacity $A^-$ using Algorithm 2;
4     $\tilde{A} \leftarrow \Phi_{\hat{A}}^{-1}(p)$;
5     Conduct the greedy constructive algorithm with
      capacity constraints $\tilde{A}$, get
      $\{X_{it}^{(0)}\}_{i \in \{1,...,N\}, t \in \{1,...,T\}}$;
6     Calculate the violation rates according to (4);
7     Update the parameters $\theta$ in the prediction through
      BO according to $\max(v - p, 0)$;
8     iter_num $\leftarrow$ iter_num $+ 1$;
9 return $X, \theta$;
```

Algorithm 2: Prediction Method in *CUC*

```
Input: Prediction parameters $\theta$, Historical capacity
       $A^- = \{a_1^-, ..., a_L^-\}$ ;
Output: Predicted capacity distribution
        $\hat{A} = \{\hat{a}_1, ..., \hat{a}_T\}$;
1 for $t \leftarrow 1$ to $L - 2T + 1$ do
2     Get prediction $\dot{a}_{t+T}^-, ...\dot{a}_{t+2T-1}^-$ trained on
      previous $L - l$ capacity values ;
3     Calculate $\delta_i \leftarrow a_i^- - \dot{a}_i^-$ for
      $i \in \{t + T, ..., t + 2T - 1\}$;
4 $\dot{A} = \{\dot{a_1}, ...\dot{a_T}\} \leftarrow$
  predict with previous capacities $\{a_{L-T+1}^-, ..., a_L^-\}$;
5 $\epsilon = \{\epsilon_1, ..., \epsilon_T\} \leftarrow$
  fit the error distribution by GMM using
  $\{\delta_{L-T+1}, ..., \delta_{L-1}\}$;
6 $\hat{A} = \dot{A} + \epsilon$;
7 return $\hat{A}$
```

the distribution of prediction error similar to the prediction component in [Luo *et al.*, 2020]. The detail of the prediction method is shown in Algorithm 2. We first construct prediction samples using a sliding window, and then calculate the prediction errors on the samples and fit the distribution of prediction uncertainty using Gaussian mixture model (GMM) [Luo *et al.*, 2020] with these calculated prediction errors. Although *CUC* can work with any parametric time series prediction model since it treats the prediction function as a black box, in this work we use a time series decomposition based forecasting approach [Hyndman and Athanasopoulos, 2018] considering its performance and computational efficiency for the experiments in this work.

The predicted distributions for future capacity in T consecutive time units are then used to calculate the expected capacity constraints $\tilde{A} = \Phi_{\hat{A}}^{-1}(p)$ where $\Phi_{\hat{A}}^{-1}(\cdot)$ is the inverse cumulative distribution function (CDF) for distribution $\hat{A}$. Ideally, the predicted distribution $\hat{A}$ is close enough to the true distribution, and then setting the constraints $\tilde{A}$ according to this formula would better control the violation rate.

For our imperfect prediction results, the model parameters θ for the prediction model are then iteratively updated in the training stage so as to achieve the optimization goal in (3), which is to control the actual violation rate v of capacity at a specific low level and maximize the utilization of all deployed jobs. A greedy constructive method described in Algorithm 3 is designed to tackle with the inner-level optimization problem with high computational efficiency under the capacity constraints $\tilde{A}$.

Note that in (3) the inner layer optimization problem is a standard MIP problem which is known to be NP-hard, and therefore we propose a new, greedy constructive method to meet the requirement on computational cost in our scheduling application. It has been demonstrated both theoretically [Du and Pardalos, 1998] and practically [Cai *et al.*, 2017] that greedy strategy is a major way of designing algorithms for NP-hard optimization problems. Moreover, experiments in Section 5 also demonstrate the effectiveness of our greedy constructive algorithm.

The violation error, i.e., the difference between the actual violation rate v and the pre-specified level p when v is larger than p, is then calculated as the objective for choosing prediction model parameters. In practice, there are usually many samples in the training set, and in this case the error on the violation level is calculated by: $\sum_{s=1}^{S} \max(v_s - p, 0) / S$, where S is the number of samples in the training set. The violation error is then sent to the Bayesian optimization (BO) [Mockus and Mockus, 1991] module which basically builds and maintains a surrogate model to select better parameters for reducing the violation error. The prediction and optimization steps conducted in an iterative manner until the actual violation rate $\bar{v}$ is still smaller but close enough to the pre-specified violation level p, or the number of iteration steps is large enough with regard to the system requirements. The schedule X for the final optimization step is then output as the scheduled plan, and the parameters θ is output as the prediction model parameters to be adopted in the testing stage or in production environment.

For the testing stage, the prediction model is run with the learned parameters and then the corresponding optimization problem is solved only once for each sample in the testing set with Algorithm 3. The output scheduled plan X is then output as the actual deployment plan for the jobs in application. The testing procedures of *CUC* are concluded in Algorithm 4.

5 Experiments

To evaluate the performance of our proposed scheduling algorithm *CUC*, we conduct extensive experiments on three public datasets and one synthetic dataset. Then we illustrate the effects of both the BO module and the uncertainty prediction

Algorithm 3: Greedy Construction Method in *CUC*

```
Input: Capacity Constraints $\tilde{A} = \{\tilde{a}_t\}_{t\in\{1,\dots,T\}}$;
       Requests $B = \{b_i\}_{i\in\{1,\dots,N\}}$;
Output: Schedule $\{X_{it}\}_{i\in\{1,\dots,N\},t\in\{1,\dots,T\}}$;
1  Sort the set of request by descending order of $c_i/d_i$;
2  for $i \leftarrow 1$ to $N$ do
3      $\Delta \leftarrow$ initialize an empty list of capacity gaps;
4      $ST_i \leftarrow \mathbf{min}(l_i, T - d_i + 1)$, the latest start time;
5      for $t \leftarrow e_i$ to $ST_i$ do
6          $\delta_t \leftarrow \min_{\tilde{t}\in\{t,t+d_i\}} \left(\hat{a}_{\tilde{t}} - c_i - \sum_{j=1}^{i-1} c_j \sum_{u=\tilde{t}-d_j+1}^{\tilde{t}} X_{ju}\right)$;
7          add element $\delta_t$ to the list $\Delta$;
8      $\delta_{t^*} \leftarrow$ find the largest $\delta_t \in \Delta$;
9      if $\delta_{t^*} > 0$ then
10         $X_{it^*} \leftarrow 1$, deploy job $i$ at time $t^*$;
11     else
12         continue, skip job $i$;
13 return $\{X_{it}\}_{i\in\{1,\dots,N\},t\in\{1,\dots,T\}}$;
```

Algorithm 4: Testing Procedures of *CUC*

```
Input: Requests $\{b_i\}_{i\in\{1,\dots,N\}}$, Prediction parameters
       $\theta$, Feasibility level $p$;
Output: Schedule $\{X_{it}\}_{i\in\{1,\dots,N\},t\in\{1,\dots,T\}}$;
1 Get predicted distribution $\hat{A}$ according to $\theta$;
2 Calculate the corresponding capacity constraints $\tilde{A}$
    according to the inverse cdf of $\hat{A}$;
3 Solve the optimization problem (2) with Algorithm 3,
    get $\{X_{it}\}_{i\in\{1,\dots,N\},t\in\{1,\dots,T\}}$;
4 return $X$
```

module, as well as the power of controlling violation level in our proposed formulation for the pre-collected jobs.

5.1 Datasets

The experiments in this work are conducted on three public datasets, which are all collected from Microsoft Azure. In particular, two public datasets are introduced in the literature [Cortez *et al.*, 2017] and each of both contains the representative VM workload of Microsoft Azure across 30 consecutive days; the other dataset is described in [Hadary *et al.*, 2020] and includes the representative VM workload of Microsoft Azure across 14 consecutive days. To make these datasets applicable with our algorithm and consistent for evaluation, we extract the job requests and the corresponding available capacity for each hour to obtain three datasets with 14 days of data for each dataset. The job requests in these datasets are the jobs with duration from 1 to 6 hours. The pre-processed dataset will also be publicly available in order to provide benchmarking datasets for this type of scheduling problems.

Each of the three datasets is divided into a training set and a testing set. For the first 5 days out of the 14 days, we only use the hourly available capacity for fitting our prediction model. Then a sliding window is adopted to generate samples, with each sample containing past hourly capacity for 5 days (i.e., 120 data points for prediction), future capacity for $T = 24$ hours, and the job requests in the next 24 hours. The last 2 days are left as the testing set, and therefore there are altogether 144 samples for Day 6 to Day 11. Job requests in Day 12 are left out to avoid data leakage in constructing the dataset.

5.2 State-of-the-art Competitors and Setup

We consider the following methods as our state-of-the-art competitors. The first one is the classical two-stage method where we use several classical time series prediction models to predict the future capacity at once and then solve the correspondent MIP problem with Gurobi, a professional solver for linear and non-linear optimization problems. The prediction methods in our experiments include linear regression (LR), time series decomposition based forecasting approach (TS-Dec) [Hyndman and Athanasopoulos, 2018], automatic autoregressive integrated moving average (AutoARIMA) [Hyndman and Athanasopoulos, 2018], long short-term memory (LSTM) [Luo *et al.*, 2019], un-observed component model (UCM) [Durbin and Koopman, 2012] and Prophet [Taylor and Letham, 2018]. Besides, we also compare our scheduling algorithm with OptNet [Amos and Kolter, 2017] who solves differentiable optimization problems through a layer in neural networks. Note that it has been claimed in [Amos and Kolter, 2017] that OptNet is not practical with a too heavy computational burden when the number of hidden variables in an OptNet layer is larger than 1000, and our preliminary experiments also confirm that OptNet is not applicable for the data of 100k-scale decision variables in the three datasets we use. Therefore, we have down-sampled the job requests and capacity correspondingly in the Azure 2020 dataset with a factor of 0.01 to make a fourth synthetic data for incorporating OptNet as a competitor of our algorithm.

In this work, all experiments were conducted on a machine with Intel Xeon E5-2690 v4 CPU, 112GB memory and NVIDIA Tesla P100 GPU. In *CUC*, we set the number of BO steps as $k = 50$ as suggested in [Wu *et al.*, 2019] if not explicitly presented in the table.

5.3 Experiment Results

Scheduling Performance against Competitors

The experiment results for comparing the proposed *CUC* method with the other 7 state-of-the-art scheduling methods are shown in Table 1. We record the utility ratio on the testing set (denoted by 'Util'), which is calculated as follows:

$$\text{Util} = \frac{\sum_{i=1}^{N}\sum_{t=1}^{T} c_i d_i X_{it}}{\sum_{i=1}^{N}\sum_{t=1}^{T} c_i d_i X_{it}^*} \times 100\% \tag{5}$$

where X_{it}^* is the optimal schedule obtained by actually solving the MIP problem with all the capacity known by Gurobi solver on the testing set. Besides, the actual violation rate on the testing set (denoted by 'Vio') is also computed by $\text{Vio} = \sum_{s=1}^{S} v_s/S \times 100\%$, where S is the number of samples in the testing set. Note that, when a scheduling plan

Method	Azure 2017			Azure 2019			Azure 2020			Synthetic		
	Util(%)	Vio(%)	Time(s)	Util(%)	Vio(%)	Time(s)	Util(%)	Vio(%)	Time(s)	Util(%)	Vio(%)	Time(s)
LR+Gurobi	100.11	55.03	5017	100.98	62.33	3537	99.03	37.85	5065	99.01	19.97	19
TSDec+Gurobi	99.75	49.22	3050	99.72	45.31	3177	99.10	33.25	12269	98.90	11.55	20
AutoARIMA+Gurobi	101.60	80.90	3024	98.63	44.44	3300	101.20	82.47	13375	100.74	38.37	20
LSTM+Gurobi	99.52	42.62	2205	98.01	21.96	2363	98.98	12.33	5769	98.76	27.08	21
UCM+Gurobi	99.30	41.15	2007	99.02	36.11	3209	97.70	11.11	6764	97.97	0	21
Prophet+Gurobi	101.59	80.73	2184	100.33	59.90	3694	101.15	81.86	7889	100.74	35.59	21
OptNet	–	–	–	–	–	–	–	–	–	94.45	0	20
CUC ($p = 0.1\%$)	93.52	0	31	90.94	0.09	33	92.17	0	53	92.04	0	1.44

Table 1: Performance comparison for *CUC* with other 7 competitors on 3 real datasets and 1 synthetic dataset, including the utility ratio, violation rate and running time in seconds. OptNet is not applicable in real datasets due to its heavy computational cost.

violates the capacity constraints (i.e., Vio is greater than 0), the resulted utility ratio might be larger than 1.

Considering the cloud computing application scenario of *CUC*, the violation rate should be well controlled at a low level (we set $p = 0.1\%$ here according to the common reliability requirements in cloud computing) in order to avoid severe impact for on-demand workloads. Therefore, all the scheduling methods compared in this experiment should be able to stably achieve a low violation rate on testing set and provide a satisfactory utility ratio as well.

The comparison results are shown in Table 1, from which we can obtain the following observations. First, *CUC* provides the best performance in controlling the violation rate, and at the same time does not sacrifice much on the utility ratio. According to [Luo *et al.*, 2020], achieving the utility ratio around 80% is already favorable in production. Moreover, *CUC* performs around 100 times faster than any other compared method in the computational time for scheduling.

Discussion. We note that, a production cloud platform must achieve a high reliability, so a scheduling plan with high violation rate is not applicable. As demonstrated in Table 1, all our competitors exhibit high violation rates, while *CUC* can achieve high utilization rate ($> 90\%$) while keep low violation rate ($< 0.1\%$), indicating that the superiority of *CUC* over all competitors in real-world applications.

Ablation Analysis

To demonstrate the effectiveness of the uncertainty modeling and the Bayesian optimization module in *CUC*, an ablation analysis is conducted. Here we compare our proposed *CUC* algorithm with its two variants: in the first variant we conduct prediction on the capacity value without error distribution considered, and thus just set constraints as the predicted capacity; for the second variant, we set the iteration times of BO as $k = 1$ to avoid iterated update on the prediction model parameters, and set the violation level parameter as $p = 0.1\%$ to keep consistent with the setting of *CUC*. The related results are reported in Table 2, and clearly show that both the uncertainty modeling and the BO module contributed to the good performance of *CUC* on reducing the violation rate.

Controlling Violation Level by p

Another appealing property of *CUC* is that the violation rate can be flexibly controlled according to the required system

Method	Azure 2017		Azure 2019		Azure 2020	
	Util(%)	Vio(%)	Util(%)	Vio(%)	Util(%)	Vio(%)
W/o Uncertainty	99.34	44.44	98.90	45.13	98.07	24.13
W/o BO ($p = 0.1\%$)	97.58	15.36	95.52	6.51	95.74	0.43
CUC ($p = 0.1\%$)	93.52	0	90.94	0.09	92.17	0

Table 2: Results of ablation experiments on the uncertainty module and BO module.

Method	Azure 2017		Azure 2019		Azure 2020	
	Util(%)	Vio(%)	Util(%)	Vio(%)	Util(%)	Vio(%)
$p = 0.1\%$	93.52	0	90.94	0.09	92.17	0
$p = 0.2\%$	93.92	0	91.27	0.17	92.24	0
$p = 0.5\%$	94.49	0	91.45	0.17	92.58	0
$p = 1\%$	94.50	0	92.18	0.34	93.27	0
$p = 2\%$	95.47	0	92.96	0.69	93.93	0
$p = 5\%$	96.24	0.52	94.15	3.82	94.79	0.95
$p = 10\%$	96.92	3.65	95.34	9.55	95.51	2.86

Table 3: Performance with different violation level p.

reliability level. We have compared the violation rate under different pre-specified violation levels p in Table 3. The results demonstrate that by setting different p values in *CUC*, the actual violation rate can be well controlled under p and the utility ratio can still be satisfactory. This study further demonstrates that *CUC* can be flexibly and efficiently applied in a wide range of job scheduling applications with different reliability requirements.

6 Application in Practice

Microsoft Azure has been continuously evolving to provide products and services to meet emerging demands from customers. The feature of pre-collecting VM requests with a discount is such an example. In practice, we allow customers to submit jobs in advance which will run continuously for 1-6 hours. The proposed method has supported the scheduling plan for this type of pre-collected VM requests in Microsoft Azure. According to the evaluation result during a pilot study, we observe a 65% increase in utilization ratio with the proposed scheduling method compared to the traditional first-in-first-out fashion, with the average waiting time for the VM requests not significantly prolonged.

7 Conclusion

To utilize the computing resources more effectively and efficiently, it is of critical importance for cloud platforms to develop intelligent scheduling methods for pre-collected demands under uncertain available capacity in the near future, such as the job format considered in this work. We formally formulate the job scheduling problem under uncertain capacity constraints as a Prediction + Optimization problem , and propose a novel and effective algorithm *CUC* for solving this type of scheduling problem. Our proposed algorithm *CUC* can effectively control the violation level according to specific requirements of the whole resource management system and maximize the resource utilization at the same time. The effectiveness of *CUC* has been validated through extensive experiments on three public, real-world application datasets.

For future work, we plan to consider the optimization problem with uncertainty in both constraints and objective functions, which can be applied in a broader range of real-world scenarios. Moreover, although currently reinforcement learning techniques cannot be directly applied to the predictive job scheduling problem, it is still a promising direction to explore how to adopt advanced models on reinforcement learning on this problem.

Acknowledgements

We would like to thank Yunus Mohammed, Marcus Fontoura, Girish Bablani, Lidong Zhou, and Hsiao-Wuen Hon for their great support and sponsorship. We would also like to thank Shandan Zhou, Kevin Yang, Harrison Zhao, Karthikeyan Subramanian, Marko Lalic, and Rajeesh Ramachandran for the collaboration on the application of the scheduling algorithm into Microsoft Azure.

References

[Amos and Kolter, 2017] Brandon Amos and J. Zico Kolter. OptNet: Differentiable optimization as a layer in neural networks. In *Proceedings of ICML*, pages 136–145, 2017.

[Box and Jenkins, 1976] George.E.P. Box and Gwilym M. Jenkins. *Time Series Analysis: Forecasting and Control*. Holden-Day, 1976.

[Cai *et al.*, 2017] Shaowei Cai, Jinkun Lin, and Chuan Luo. Finding A small vertex cover in massive sparse graphs: Construct, local search, and preprocess. *Journal of Artificial Intelligence Research*, 59:463–494, 2017.

[Calzarossa *et al.*, 2019] Maria Carla Calzarossa, Marco L Della Vedova, and Daniele Tessera. A methodological framework for cloud resource provisioning and scheduling of data parallel applications under uncertainty. *Future Generation Computer Systems*, 93:212–223, 2019.

[Chen *et al.*, 1998] Bo Chen, Chris N. Potts, and Gerhard J. Woeginger. *A Review of Machine Scheduling: Complexity, Algorithms and Approximability*, pages 1493–1641. Springer US, Boston, MA, 1998.

[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.

[Demirovic *et al.*, 2019] Emir Demirovic, 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.

[Du and Pardalos, 1998] Dingzhu Du and Panos M Pardalos. *Handbook of combinatorial optimization*, volume 4. Springer Science & Business Media, 1998.

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

[Fanjul-Peyro *et al.*, 2017] Luis Fanjul-Peyro, Federico Perea, and Rubén Ruiz. Models and matheuristics for the unrelated parallel machine scheduling problem with additional resources. *European Journal of Operational Research*, 260(2):482 – 493, 2017.

[Fleszar and Hindi, 2018] Krzysztof Fleszar and Khalil S. Hindi. Algorithms for the unrelated parallel machine scheduling problem with a resource constraint. *European Journal of Operational Research*, 271(3):839 – 848, 2018.

[Gawali and Shinde, 2018] Mahendra Bhatu Gawali and Subhash K Shinde. Task scheduling and resource allocation in cloud computing using a heuristic approach. *Journal of Cloud Computing*, 7:4, 2018.

[Gupta and Denton, 2008] Diwakar Gupta and Brian Denton. Appointment scheduling in health care: Challenges and opportunities. *IIE transactions*, 40(9):800–819, 2008.

[Hadary *et al.*, 2020] Ori Hadary, Luke Marshall, Ishai Menache, Abhisek Pan, Esaias E Greeff, David Dion, Star Dorminey, Shailesh Joshi, Yang Chen, Mark Russinovich, and Thomas Moscibroda. Protean: VM allocation service at scale. In *Proceedings of OSDI 2020*, pages 845–861, 2020.

[Huang *et al.*, 2018] X. Huang, J. Cao, and Y. Tan. A prediction based server cluster capacity planning strategy. In *Proceedings of PIC 2018*, pages 296–304, 2018.

[Hyndman and Athanasopoulos, 2018] Robin John Hyndman and George Athanasopoulos. *Forecasting: Principles and Practice*. OTexts, 2nd edition, 2018.

[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.

[Luo *et al.*, 2020] Chuan Luo, Bo Qiao, Xin Chen, Pu Zhao, Randolph Yao, Hongyu Zhang, Wei Wu, Andrew Zhou, and Qingwei Lin. Intelligent virtual machine provisioning in cloud computing. In *Proceedings of IJCAI 2020*, pages 1495–1502, 2020.

[Mandi *et al.*, 2020] Jayanta Mandi, Emir Demirovic, Peter J. Stuckey, and Tias Guns. Smart predict-and-optimize for hard combinatorial optimization problems. In *Proceedings of AAAI 2020*, pages 1603–1610, 2020.

[Mesbahi *et al.*, 2018] Mohammad Reza Mesbahi, Amir Masoud Rahmani, and Mehdi Hosseinzadeh. Reliability and high availability in cloud computing environments: a reference roadmap. *Human-centric Computing and Information Sciences*, 8:20, 2018.

[Mockus and Mockus, 1991] J. B. Mockus and L. J. Mockus. Bayesian approach to global optimization and application to multiobjective and constrained problems. *Journal of Optimization Theory and Applications*, 70:157–172, 1991.

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

[Verderame *et al.*, 2010] Peter M Verderame, Josephine A Elia, Jie Li, and Christodoulos A Floudas. Planning and scheduling under uncertainty: a review across multiple sectors. *Industrial & Engineering Chemistry Research*, 49(9):3993–4017, 2010.

[Wu *et al.*, 2019] Jia Wu, Xiu-Yun Chen, Hao Zhang, Li-Dong Xiong, Hang Lei, and Si-Hao Deng. Hyperparameter optimization for machine learning models based on bayesian optimization. *Journal of Electronic Science and Technology*, 17(1):26–40, 2019.

[Zhou *et al.*, 2020] Liping Zhou, Na Geng, Zhibin Jiang, and Xiuxian Wang. Public hospital inpatient room allocation and patient scheduling considering equity. *IEEE Transactions on Automation Science and Engineering*, 17(3):1124–1139, 2020.