Coupling QoS Co-Simulation with Online Adaptive Arrival Forecasting

Yichong Chen
Department of Computing
Imperial College London
London, United Kingdom
yichong.chen119@imperial.ac.uk

Manuel Roveri
DEIB
Politecnico di Milano
Milano, Italy
manuel.roveri@polimi.it

Shreshth Tuli
Happening Technology Ltd.
London
United Kingdom
shreshth.tuli@happening.xyz

Giuliano Casale
Department of Computing
Imperial College London
London, United Kingdom
g.casale@imperial.ac.uk

***Abstract*—Coupled simulation, also known as co-simulation, has been proposed to provide more information to a task scheduler by simulating at runtime the Quality of Service (QoS) arising from a scheduling action. To do so, co-simulation algorithms run the simulation assuming a static set of arrival time series, restricting the diversity of the traffic scenarios. To ensure the co-simulator can provide valuable and representative results, we present an online adaptive arrival forecasting framework that contains a change-point detection module and a probabilistic transformer model to couple co-simulators with arrival series forecasting. The framework can also update the prediction model to adapt to dynamic environments. Our experiments show that our online adaptive forecasting framework has lower forecasting errors than established prediction models, such as autoregressive processes, and lower on real-world traces the co-simulator prediction error by up to 27% on average response time and 39% on average service-level agreement (SLA) violation.**

***Index Terms*—fog computing, time series forecasting, change point detection, co-simulation**

I. Introduction

Fog computing is a recently proposed paradigm that combines the benefits of Edge and Cloud Computing. Compared to the latter, Fog computing places latency-sensitive applications on edge devices to reduce latency and protect user privacy. Meanwhile, resource-consuming applications are deployed on the Cloud to reduce the cost of consuming servers and hardware. Reliability and performance of a fog node are restricted by its compute capacity, storage, and battery, raising the need for effective resource allocation and task scheduling. Many studies propose various scheduling strategies, such as heuristic, artificial intelligence/machine learning (AI/ML) and deep learning (DL) based algorithms, to optimize Quality-of-Service (QoS), defined as the level of performance, availability and reliability provided by the hosts [1]. Most of these studies assume that the topology, arrival process, bandwidth, etc., remain unchanged. However, these assumptions may be violated in a real-world system. For example, the topology of a Fog system can be changed by loading or offloading hosts, or concept drift exists in the arrival process of a real-world application.

Co-simulation, which simulates the response of a complex system to real-time changes, has been used as a digital-twin of a Fog system to help scheduling algorithms gain knowledge of the system and boost the scheduler to achieve better QoS [2], [3]. Existing works have examined the benefits of co-simulation in a static setting where the parameters of the arrival processes do not vary over time. However, the static setting of the parameters of the arrival process cannot mimic concept drifts in a real-world system which impedes the co-simulator from providing valuable estimations. The goal of this paper is to endow a mechanism to update the arrival traces with which the co-simulator is parameterized so that the co-simulator can well represent the present system.

To ensure that the co-simulator provides a better estimation of QoS parameters, we dynamically generate an arrival series, represented as a time series of counts for each job type, as the input trace to the co-simulator using a probabilistic transformer. Furthermore, we present an online adaptive framework to adapt the arrival generation model to the dynamic environment. The adaptive framework uses a change point detection module to monitor the arrival series online and fine-tune the transformer model after change points are detected in the arrival series.

We propose, in particular, a hierarchical change detection structure based on the theory developed in [4] as the detection module. We find that the existing hierarchical framework can only detect changes on the mean, which is also named location shift, in a time series. However, real-world arrival traces, such as Alibaba trace [5], not only contain location shifts but also changes in the variability, which is also called scale shifts. In order to complement the loss of scale shift detection, we present an extended data-depth plus CUSUM (DD$^+$-CUSUM) algorithm to detect both increase and decrease change in scale and form the hierarchical change point detection algorithm (HCPD). Experiments on real-world datasets from an Alibaba cluster indicate that our HCPD forecasting framework can ensure the co-simulator gives more accurate simulation results than with offline artificial parameterization.

In particular, our experiments show that HCPD can detect and estimate change points with lower detection delay and lower mean absolute error (MAE) on change point estimation compared to the Bayesian online change point detection. Further, the probabilistic transformer forecasting model can provide valuable arrival series prediction to the co-simulator to estimate QoS result with lower MAE compared to other time series forecasting methods. Moreover, the online adaptive

978-3-903176-59-1

framework can further boost the transformer model to give predictions that have lower MAE on the QoS estimation running on the co-simulator compared to the model without the online adaptive framework.

In summary, the key contributions of this paper are:

- We define a DD$^+$-CUSUM algorithm to detect both increase and decrease change in scale and present a non-parametric online change point detection algorithm to detect location and scale shift and estimate the change point.
- We design a probabilistic transformer model to forecast the arrival series with only one forward step needed.
- We present an online adaptive arrival forecasting framework in order to provide valuable arrival series prediction for the co-simulator to achieve better QoS estimation under real-world scenarios.

This paper is structured as follows. We begin by discussing related work in Section II. We defined the adaptive prediction problem in Section III and, in Section IV, we provide a brief explanation of the techniques used in the proposed model. The proposed online adaptive arrival series prediction model is presented in Section V in detail. Experiment settings and results are presented in Section VI to show the performance of the proposed model, and we conclude in Section VII.

II. Related Work

A. Co-simulation & Scheduling

The resource allocation and task scheduling problem aims to identify optimal task-node pairs that satisfy constraints, such as fulfilling an SLA, and optimizes the considered QoS parameters, such as response time. However, [6] illustrates that finding the optimal scheduling policy for a set of tasks even in basic scheduling problems is normally NP-hard. Rather than seeking the globally optimal scheduling, most studies employ heuristics search to find near-optimal solutions that meet constraints and ensure high QoS. For example, [7], [8] present task scheduling algorithms to minimize the operational costs of the system, such as the energy consumption and devices usage fee, without violating SLA.

AI/ML methods have also been applied to resource allocation and task scheduling in fog computing. Several works apply deep reinforcement learning (RL) to adapt to stochastic workloads in fog computing [9], [10]. The policy gradient-based algorithm in [9] takes the environment state, which consists of tasks and hosts parameters and history assignment of task-node pairs, as the input of the model. It predicts the rewards of each scheduling policy and selects the optimal scheduling decision. However, these RL models tend to be slow to adapt to real-world application scenarios [2].

Co-simulation is also used in modelling distributed systems [2], [3]. These prior works select the optimal scheduling decision using a simulation model that predicts QoS by simulating a decision execution. For example, [2] proposes a co-simulation framework, named COSCO, which provides a dynamic interactive environment for AI models to optimize QoS. Moreover, the proposed DL model GOBI* takes the simulation result to boost the prediction, making a better performance than the model without using the simulation result. Follow-up works, such as [3], illustrate an extension to the approach to increase the robustness of decisions.

B. Change Detection Test & Change Point Detection

Change detection tests (CDT) are applied to determine a time series as containing abrupt changes to one or more properties, such as the mean, variance or correlation. The cumulative sum (CUSUM) has been widely used as an online change detection algorithm [11]. Traditionally, CUSUM assumes that the distribution family of the time series and the parameters of the distribution are known as a prior. However, this assumption does not always hold in practice. [12] therefore presents a distribution-free CUSUM control chart for change detection. However, it can only detect changes that happen in the location parameters of the distribution. [13] introduces a data-depth (DD) CUSUM control chart to detect the scale change. However, this algorithm only detects the change when the scale increases.

Change point detection (CPD) is more challenging compared to change detection since it aims both at detecting and locating the occurrence of a change in a finite sequence of data. [14] detects a change point by separating the time series into two sub-series and finding the split point with the maximum difference. Instead of using statistical tests, [15] applies kernel function, which projects the features to infinite dimension space, to measure the difference between two sub-series. [16] introduces the Bayesian online change point detection (BOCPD) to identify the changes in an online fashion. However, this method requires assumptions on the distribution family of the time series.

C. AI/ML-Based Time Series Forecasting

Forecasting has been widely applied in resource management and scheduling [1]. Autoregressive integrated moving average (ARIMA) is a traditional statistical model that has been widely used in time series forecasting [17]. It assumes a linear correlation between the past and present observations at different time lags and it is also able to capture trends in non-stationary time series. However, real-world time series might not always be captured effectively by the recursive form for the time series data assumed by autoregressive processes.

DL models are also used in time series analysis. The recurrent neural network (RNN) structures, such as vanilla RNN, long short term memory (LSTM) and gated recurrent unit (GRU) [18], are commonly used in time series prediction. They recurrently process the time series in order to capture correlation features. DeepAR [19] uses RNN structure to model probabilistic features of the series and predict parameters of the distribution. Although RNN models are effective in modelling the time series data, the RNN structures are slow to train, and their performance of forecasting decreasing in the length of input data increases.

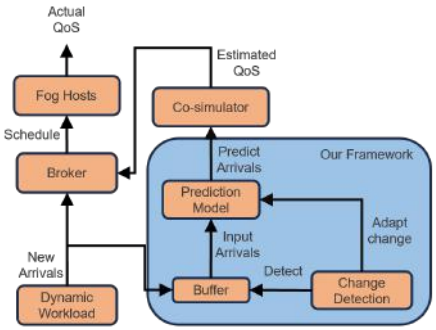

Fig. 1: A dynamic Fog system with an adaptive digital-twin co-simulator

The transformer model [20] is another popular structure in time series forecasting. The attention mechanism in transformers captures the correlation between each time instance. Moreover, a transformer is faster to train compared to RNN models. Some transformer models, such as LongFormer [21], SparseTransformer [22], and TransformerXL [23], are designed for long sequence forecasting. However, the standard transformer models are slow during the inference procedure because they predict one future instance at a time and take the newly predicted value to make the next prediction.

III. Problem Formulation

We consider a dynamic Fog system that contains a digital-twin co-simulator, which is used to estimate the QoS of the system, as shown in Figure 1. The Fog system receives tasks from a gateway that collects tasks and sends them to the broker. The scheduling decisions are made by the broker based on the QoS estimates from the co-simulator. The tasks are executed on the assigned hosts, and the actual QoS is computed.

The distribution of the arrival process can change over time. We wish to design a framework that predicts the number of arrivals in L step ahead with the last t_0 observation of arrivals and adapt the prediction model when concept drift occurs on the distribution of the arrivals. The prediction and adaptation mechanism form our proposed framework.

We define the number of arrivals at time t as $\mathbf{x}_t = (x_t^1, \ldots, x_t^d) \in \mathbb{Z}^{*d}$, where x_t^i is the number of class ith tasks at time t and d is the number of types of jobs. Therefore, we try to predict the future arrival series $\mathbf{X}_{t_0+1:t_0+L} = \{\mathbf{x}_t, t = t_0+1, t_0+2, \ldots, t_0+L\}$ denoting as prediction series, with the latest observations series $\mathbf{X}_{0:t_0} = \{\mathbf{x_t}, t = 0, 1, \ldots t_0\}$ denoting as condition series.

Instead of predicting the exact value of the prediction series, which is usually difficult to achieve for counting data, we model the distribution of the prediction series $P(\mathbf{X}_{t_0+1:t_0+L}|\mathbf{h})$ with $\mathbf{h}$ as the parameters of the distribution. We predict $\mathbf{h}$ with a neural network f_θ from the condition series $\mathbf{X}_{0:t_0}$ so that $\mathbf{h} = f_\theta(\mathbf{X}_{0:t_0})$, where θ refers to the parameters in the neural network.

Therefore, the goal of our time series forecasting is to find θ that maximizes the likelihood of the prediction series given the condition series. The maximizing problem is formulated as

$$\max_\theta \prod_{i=0}^{N} P(\mathbf{X}^i_{t_0+1:t_0+L}|f_\theta(\mathbf{X}^i_{0:t_0})), \tag{1}$$

where N is the number of training series and i is the ith sample.

The prediction model is trained on a history arrival trace. However, the model may fail to give accurate predictions when the distribution of the arrivals is shifted. Therefore, we introduce a change point detection mechanism in the framework to detect changes in the arrival series. When a change is detected, the prediction model is fine-tuned using the most recent arrivals.

Considering an arrival series $\mathbf{X}_{1:T}$ with length T, we assume that it is possible to separate the time series into two non-overlapping partitions such that the distribution of each partition is different. The time instances that divide the time series is the change point. Under this scenario, the change point detection problem can be formulated as follow:

$$\mathbf{x}_t \sim \begin{cases} \Phi_0 & t < T^* \\ \Phi_1 & t \geq T^* \end{cases} \tag{2}$$

where Φ_0 is the distribution of the arrivals before the change occurs, Φ_1 is the distribution after the location or scale shift, and T^* is the change point we want to identify.

IV. Background

A. Hierarchical Change Detection Test

CUSUM-type change detection algorithms are effective and have been long used in online change detection [11]. The standard CUSUM-type algorithm cumulatively sums the log-likelihood ratio at each time point. The paradigm detects changes when the cumulative sum exceeds the preset control limits. However, these CUSUM-type change detection algorithms cannot estimate the exact change point in the series.

Hypothesis tests can be used to estimate the location of the change point and are named change point methods. These change point methods split the time series into two sub-series and compute the statistical test value of the difference between the sub-series. Hypothesis testing is performed on the splitting using the maximum statistical test value. The splitting point is estimated as the change point if the hypothesis test rejects the null hypothesis. However, these methods cannot be applied in online detection because performing the hypothesis test in an online setting increases the probability of making false positive detection, known as sequential testing issues [24].

To address this, we propose a hierarchical change detection test (HCDT) framework proposed in [4] combines CUSUM change detection algorithms and hypothesis change point methods. HCDT monitors the time series with the CUSUM algorithm. If the new coming data trigger the CUSUM detection, HCDT performs a hypothesis test with the hypothesis change point method to validate the detection. The HCDT framework

does not perform the hypothesis test in every step, which prevents sequential testing issues. Meanwhile, validating the CUSUM detection with the hypothesis change point method can further reduce the false positive detection. In this work, we develop a novel method within the HCDT paradigm for estimating change points in the proposed prediction framework.

B. Probsparse Attention

The transformer model has been widely used in processing time series data and has achieved great success in many fields. The attention mechanism introduced in [20] plays a vital role in the transformer model. The attention module takes query $\mathbf{Q} \in \mathbb{R}^{L_Q \times m}$, key $\mathbf{K} \in \mathbb{R}^{L_K \times m}$ and value $\mathbf{V} \in \mathbb{R}^{L_V \times m}$ as inputs, where L_Q, L_K and L_V represent the length of the corresponding series and m is the dimension of an instance in the input series. The attention function is formulated as

$$\mathcal{A}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d}}\right)\mathbf{V} \tag{3}$$

The output of the attention function $\mathcal{A}$ is the weighted sum of $\mathbf{V}$, where the weights are calculated from the dot-product between $\mathbf{Q}$ and $\mathbf{K}$, followed by the softmax function applied column-wise.

Some of the previous works [21], [22] have noted that the attention weights of query q_i have potential sparsity, which means only a few dot-product pairs (q_i, k_j) with large weight contribute to the attention, where q_i and k_j are the ith row and jth row of query $\mathbf{Q}$ and key $\mathbf{K}$ respectively. In this scenario, the attention function should extract key features from the value $\mathbf{V}$ with dominant weights. On the contrary, the attention function becomes calculating the mean of the value $\mathbf{V}$ when the attention weights are identical. The calculation of attention becomes trivial in this circumstance, which reduces the processing speed and increases memory usage in processing long sequences.

The *ProbSparse* attention defines a query sparsity measurement to evaluate the potential sparsity of a query q_i [25]. Following the formulation in [26], the attention weight of a dot-product pair (q_i, k_j) can be written as a conditional probability $p(k_j|q_i)$. If sparsity does not exist in the attention weights, the attention weights would be close to a uniform distribution, i.e., $q(k_j|q_i) = 1/L_K$. The Kullback-Leibler divergence $KL(q||p)$ can be used to measure the distance between attention probability $p(k_j|q_i)$ and the uniform distribution $q(k_j|q_i)$. KL divergence increases when sparsity exists in the attention weights, and it decreases when the attention weights are identical. Therefore, KL divergence can represent the sparsity of the attention weights for q_i. The sparsity measurement of q_i is defined as

$$M(q_i, \mathbf{K}) = \max_j \left(\frac{q_i k_j^T}{\sqrt{d}}\right) - \frac{1}{L_K}\sum_{j=1}^{L_K}\frac{q_i k_j^T}{\sqrt{d}} \tag{4}$$

To reduce the calculation on dot-product pairs, *ProbSparse* attention randomly selects $L_K \ln L_Q$ pairs to calculate the sparsity measurement for a query q_i. The queries with the top $c \ln L_Q$ measurement are selected and form the new query metric $\bar{\mathbf{Q}}$ and replace the original query $\mathbf{Q}$ in Eq. 3, where c is a tunable factor.

V. Proposed Model

In this section, we present a hierarchical framework to predict a long-time arrival series for the co-simulator to estimate the performance metrics of the new arrival tasks with the default scheduling policy.

A. Non-parametric Hierarchical Change Point Detection

As we mention in Section IV-A, HCDT is an effective framework to detect the changes in an online scenario and has the ability to estimate change points. However, the original work [4] does not evaluate the performance of change point estimation. Furthermore, our framework requires knowledge of when the change happens in order to adapt the prediction with the latest valid data. Therefore, we extend its usage to detect the change point in our prediction framework and rename it hierarchical change point detection (HCPD).

To detect both location and scale shifts in the time series, we apply MCUSUM and DD$^+$-CUSUM change detection algorithms. MCUSUM, presented in [27], is a non-parametric location shift detection algorithm. It replaces the log-likelihood ratio with the difference between the new observation and the sample mean and draws the control chart on the summation of this difference. DD-CUSUM [13] can be used to detect the scale increase. However, the original work does not consider decreasing in scale shift. To address this issue, we propose an extended version of DD-CUSUM named data-depth plus CUSUM (DD$^+$-CUSUM) to detect both increase and decrease shifts in scale.

The data depth *R* statistic measures the distance between an observation and the mean of a distribution. Given observed arrivals $\mathbf{x}_t$, the data depth can be formulated as

$$DD_{\Phi_0}(\mathbf{x}_t) = 1 - ||E_{\Phi_0}(U(\mathbf{x}_t - \mathbf{y}))||, \tag{5}$$

where $\mathbf{y} \sim \Phi_0$ is data from distribution Φ_0, and the operation $U(x) = \frac{x}{||x||}$. Thus, $DD_{\Phi_0}(\mathbf{x})$ is close to 0 if the observation $\mathbf{x}$ is far away from the center of the distribution Φ_0. On the contrary, $DD_{\Phi_0}(\mathbf{x})$ becomes large and attains the maximum value 1 if the observation is near the center of Φ_0. In a change detection problem with a given historical data set $\mathbf{y}_1, \ldots, \mathbf{y}_m$, the sample data depth is defined as

$$DD_{\hat{\Phi}_0}(\mathbf{x}_t) = 1 - \frac{1}{m}\left\|\sum_{i=1, \mathbf{y}_i \neq \mathbf{x}_t}^{m} U(\mathbf{x}_t - \mathbf{y}_i)\right\| \tag{6}$$

where $\hat{\Phi}_0$ represents the empirical distribution of the data.

The original DD-CUSUM algorithm only considers the increase of the scale by cumulatively summing the estimated *R* calculated from

$$R^+_{\hat{\Phi}_0}(\mathbf{x}_t) = \frac{\sum_{i=1}^{m} I(DD_{\hat{\Phi}_0}(\mathbf{y}_i) \leq DD_{\hat{\Phi}_0}(\mathbf{x}_t))}{m} \tag{7}$$

which is the count of the historical data with a large data depth compared to the new observation. If the new observation $\mathbf{x}_t$ is

near the center of the historical data set $\mathbf{y}_i$, the data depth of $\mathbf{x}_t$ would be greater than most of the historical data, which leads to a large R statistic value. Hence, $1 - R^+_{\hat{\Phi}_0}(\mathbf{x}_t)$ can represent the distance between $\mathbf{x}_t$ and the empirical distribution of the historical data.

In order to detect the decrease of the scale, we first propose to estimate the negative R statistic measures by counting the historical data, which have a smaller data depth than the new observation. The negative estimate can be represented as

$$R^-_{\hat{\Phi}_0}(\mathbf{x}_t) = \frac{\sum_{i=1}^{m} I(DD_{\hat{\Phi}_0}(\mathbf{y}_i) \geq DD_{\hat{\Phi}_0}(\mathbf{x}_t))}{m} \tag{8}$$

Then, DD$^+$-CUSUM cumulatively sums both positive and negative R statistic measures and the control chart is performed on the one with the maximum value. The complete DD$^+$-CUSUM can be formed as follow:

$$S^+_t = \max(0, S^+_{t-1} + (1 - R^+_{\hat{\Phi}_0}(\mathbf{x}_t)) - k) \tag{9}$$

$$S^-_t = \max(0, S^-_{t-1} + (1 - R^-_{\hat{\Phi}_0}(\mathbf{x}_t)) - k) \tag{10}$$

$$S_t = \max(S^+_t, S^-_t) \tag{11}$$

DD$^+$-CUSUM detects a change when $S_t > h$.

The MCUSUM and DD$^+$-CUSUM monitor the arrival simultaneously and store the new observation in a buffer with a prefix size. The buffer drops the oldest observation and adds the newest one if the buffer reaches its size limit. HCPD detects a potential change if either one of these two CUSUM algorithms raises a detection flag.

Once the detection algorithms trigger the alarm, a validation module uses the Lepage-type (LP) hypothesis test [28], which can detect both location and scale shifts, to validate the change and estimate the change point with the stored data in the buffer. We calculate the LP test value on every possible splitting and performance hypothesis test on the one with the maximum. If the LP test confirms the change exists, we consider the splitting point with the maximum statistical test value as the estimated change point.

B. Arrival Series Forecasting

We design a transformer based encoder-decoder model with the *ProbSparse* attention as the self-attention module. to predict the long-term arrival series. The overall structure of our transformer model is shown in Fig 2.

1) Encoder: The encoder takes the input arrival series and the time features as the input. The time features of the arrival series are represented in vector form and concatenated with the input arrival series as extra features. The input series is first processed with a 1-D convolution layer to project the input series $\mathbf{X} \in \mathbb{R}^{L_{enc} \times d}$ into a high dimensions space so that $\mathbf{X}' \in \mathbb{R}^{L_{enc} \times d_{model}}$. The projected series is processed with a positional encoding layer to encode order features into the series.

The encoder contains multiple self-attention blocks. In each self-attention block, the input is processed with a multi-head self-attention module with *ProbSparse* attention. The multi-head self-attention projects the encoded input to multiple queries and key-value pairs so that these projected vectors contain different information that the attention should focus on. The output of the multi-head self-attention module is added to the input of the attention module as a residual block [29] so that the loss can back-propagate through the attention module or directly pass to the input.

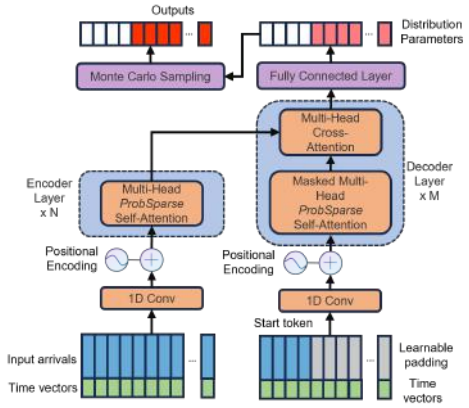

Fig. 2: The architecture of the probabilistic transformer

2) Decoder: In a standard encoder-decoder transformer model, the decoder takes the output of the encoder and a start token or its previous predictions as the input and makes the next prediction. However, the previous predictions are unavailable at the beginning under the co-simulation scenario. Meanwhile, the model is required to have fast inference speed to avoid long scheduling time. Unlike [25], which adds zero paddings to the input of the decoder to make prediction simultaneously, we set a sequence of learnable parameters with the length of maximum prediction L_{max} to replace the zero paddings. In the meantime, a subsequence of the input series of the encoder with length L_{start} works as the start token of the decoder. Therefore, the start subsequence and the learnable padding parameters form the input series of the decoder $\mathbf{X}_{dec} \in \mathbb{R}^{(L_{start}+L_{max}) \times d}$. The timestamps are also added as the time features to the corresponding input.

The decoder also has multiple layers. In each layer, X_{dec} is first processed with a multi-head masked *ProbSparse* attention. The masked dot product prevents each position from getting information from the future position. The output of the self-attention acts as the query in the cross-attention layer, and the output of the encoder forms the key-value pairs. The decoder can, therefore, receive the information of the history series and process it with the decoder input to predict future arrivals.

3) Distribution Prediction: The arrival series represents the number of tasks that are integers at each timestamp. Commonly, standard outputs of a DL model are continuous real numbers. To address this issue, one can round the output to its nearest integer, but the rounding procedure could lose information and ignore the distribution feature of count data. Instead of directly predicting the number of future arrivals, we predict the parameters of the distribution of the future arrival. Since the arrival series is a sequence of positive count data,

Algorithm 1 Adaptive Arrival Prediction Framework

Require: History arrivals $\mathbf{X}_{his}$, input length L_{in}, prediction length L_{pred}, fine-tuning length L_{fine}, buffer $\mathbf{B}$ with prefix length, sampling number N_{sample}

1: **Initialization:** HCPD, Probabilistic Transformer f_θ
2: Apply HCPD to detect change point on $\mathbf{X}_{his}$
3: Sample change-point free series from $\mathbf{X}_{his}$
4: Train f_θ with change-point free series
5: Fine-tuning flag $g \leftarrow False$
6: **while** new arrivals $\mathbf{x}_t$ is available at t **do**
7: Update $\mathbf{B}$ with $\mathbf{x}_t$
8: **if** HCDP detects a change point with $\mathbf{x}_t$ **then**
9: Estimated change point $\hat{T}$
10: $g \leftarrow True$
11: Update $\mathbf{B} \leftarrow \mathbf{B}[\hat{T} : t]$
12: **end if**
13: **if** g **then**
14: **if** len($\mathbf{B}$) $> L_{fine}$ **then**
15: Sampling fine-tuning data from $\mathbf{B}$
16: Fine-tuning f_θ
17: $g \leftarrow False$
18: **end if**
19: **end if**
20: Predict $\hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\alpha}} \leftarrow f_\theta(\mathbf{x}_{t-L_{in}:t})$
21: Sample N_{sample} arrival series with $\hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\alpha}}$
22: **end while**

our transformer model predicts the parameters of a negative binomial distribution. The probability of the arrival number of class ith tasks at time t can be represented as

$$p(x_t^i|\mu,\alpha) = \frac{\Gamma(x_t^i + \frac{1}{\alpha})}{\Gamma(x_t^i+1)\Gamma(\frac{1}{\alpha})}\left(\frac{1}{1+\alpha\mu}\right)^{\frac{1}{\alpha}}\left(\frac{\alpha\mu}{1+\alpha\mu}\right)^{x_t^i} \tag{12}$$

where $\mu \in \mathbb{R}^+$ is the mean and $\alpha \in \mathbb{R}^+$ is the shape parameter of the corresponding negative binomial distribution. Therefore, the output of the decoder is passed to two independent fully-connected linear layers with softplus activation function to predict $\hat{\mu}$ and $\hat{\alpha}$. The transformer model is trained by minimizing the negative log-likelihood function.

C. Adaptive Arrival Series Prediction Framework

In real-world systems, the distribution of task arrivals may change over time. A DL model trained with history arrivals may struggle to adapt its predictions to unseen arrival distributions without additional assistance. The prediction on the new arrival distribution might not be as good as the seen one. This can reduce the simulation accuracy and negatively affect the scheduling to make migration and allocation decisions. One solution is to fine-tune and update the DL model when a new observation becomes available.

The procedure of the online adaptive arrival prediction framework is present in Algorithm 1. Suppose we have a dataset of the history arrivals $\mathbf{X}_{his}$. Before the online procedure (lines 1-5), HCPD is used to find the change points in $\mathbf{X}_{his}$. The training series is generated from $\mathbf{X}_{his}$, excluding those containing change points. The series with change points contains arrivals from different distributions, which increases the difficulty for the transformer to learn to predict the parameters of future arrivals because the transformer needs to recognize the position of the change point and only focus on the data after the change point. Therefore, we sample the series, which is change-free, to enhance the training procedure.

During the online prediction (lines 6-22), the framework observes a new arrival $\mathbf{x}_t$ and saves it in the buffer. Once HCPD detects a change point $\hat{T}$, the framework updates the buffer $\mathbf{B}$ to contain the arrivals after $\hat{T}$ and raises the flag for the transformer to execute the fine-tuning procedure. In the fine-tuning procedure, the framework first checks the length of the buffer (line 14). If the length of the buffer contains enough data, the framework generates fine-tuning sample series from the buffer and fine-tunes the transformer (lines 15-16). Otherwise, the framework postpones the fine-tuning procedure to gather additional data.

The transformer predicts the parameters $\hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\alpha}}$ of the future distributions with series $\mathbf{x}_{t-L_{in}:t}$ generated from $\mathbf{B}$ (line 20). Finally, the framework uses Monte Carlo sampling method to sample N_{sample} arrival series with the predicted parameters and the length of each predicted series is L_{pred} (line 21). The co-simulator runs simulations with these arrival series and estimates the QoS with the mean of the simulation results.

VI. Experimental Results

In this section, we quantify the performance of the proposed model with three experiments.

A. Dataset

1) Synthetically Generated Dataset: The synthetically generated dataset contains a sequence with 12000 observations and 20 change points, derived from a negative binomial distribution. By adjusting the count number n and the shape factor p, we can control the mean and variance of the observations. The gap between two consecutive changes is at least 400 steps and at most 1000 steps. The n and p are initialized as 50 and 0.5, respectively, and n is increased or decreased by 25 or 50 at each location shift. The shape factor is chosen from 0.1 and 0.5 to simulate the scale shift. We also include trend and seasonal data using linear and sine functions. Each observation represents arrivals gathered by the gateway over a 60-second period considered as a step.

2) Alibaba Trace Dataset: We use the 2018 Alibaba arrival trace [5] as the real-world application dataset to evaluate our model since the arrival of tasks exhibits random changes. This dataset records data from over 4000 machines across 8 days. To simplify without losing generality, we focus on arrivals to the first 50 machines by ID. The trace comprises 12 classes of tasks. We count the arrivals for every 60 second as one observation, forming a trace of 11219 observations.

B. Models settings

1) COSCO Simulation Framework: We use the COSCO framework presented in [2] as the co-simulator. In COSCO,

tasks are generated based on given arrival series and are built using the BitBrain workload trace [30]. These tasks are allocated by a scheduling broker using a first-come-first-service (FCFS) strategy, directing them to the most resource-available host. COSCO executes the assigned tasks in the respective simulated hosts within a specified interval and records the QoS metrics. Simulations are run on an Intel i7-9750H CPU with 16 GB of memory.

2) Baseline Models: In the first experiment, we consider the Bayesian online change point detection method (BOCPD) [31] and the kernel change point detection with RBF kernel [32] as the baselines to evaluate the performance of our HCPD. By adjusting the thresholds of the baseline models on change-free sequence, we aimed for an average run length (ARL) of 500, denoting the expected steps before a false positive detection.

In the second experiment, we select three prediction models: mean prediction, moving average (MA) and auto-regressive integrated moving average (ARIMA) [33], as our baselines in the arrival forecasting experiment. The most recent 50 observations are used as the training data for these models, referred to as the reference series. These models are fitted to the reference series to predict the next 25 steps.

3) Adaptive Framework: The threshold of HCPD is also selected to achieve ARL 500 with the same experiments as the baseline change point detection models. The probabilistic transformer contains 6 layers of self-attention blocks in the encoder and 6 decoder layers in the decoder, with a hidden dimension of 512. The model receives a history sequence with a length of 50 as the reference series and takes the last 25 steps as the start subsequence to forecast the parameters of the arrivals in the next 25 steps. For both synthetic and Alibaba traces, the initial 7000 steps are used to pretrain the model on a single Nvidia Quadro RTX 4000 8GB GPU, with the remaining steps used for evaluation. In the online process, fine-tuning lengths, batch size and epoch are set to 150, 32 and 5, respectively, with Adam [34] as the optimizer. With these settings, the checkpoint of the probabilistic transformer is about 120MB and the resident memory as 451MB on synthetic trace and 471MB on Alibaba trace.

C. Evaluation Metrics

We define three performance metrics to evaluate the change point detection algorithms, which are false positive rate (FPR), mean detection delay (MDD) and mean absolute error (MAE). The false positive rate is defined as the inverse of the mean time of the algorithm given a false positive detection

$$\text{FPR} = \frac{1}{\frac{1}{N_{fp}} \sum_{i=1}^{N_{fp}} s_i} \tag{13}$$

where s_i is the number of steps between the ith false detection and the last detected change, and N_{fp} is the number of false positive detection. MDD is defined as the mean difference between the change occurring and the time when the algorithm detects the change. It is formulated as

$$\text{MDD} = \frac{1}{N_{cp}} \sum_{i=1}^{N_{cp}} (t_i^{detect} - t_i^{cp}) \tag{14}$$

TABLE I: Performance metric of change point detection algorithms on synthetically generated dataset

Methods	FPR (%)	MDD	MAE
HCPD	0.25	**106.88**	**75.29**
BOCPD	**0.23**	127.24	117.24
KCPD	0.24	228.82	151.45

where N_{cp} is the number of change points, t_i^{cp} is the time of the ith change point occurs, and t_i^{detect} is the detection time for the ith change point. The MAE for the change point detection is defined as follows

$$\text{MAE}_{cp} = \frac{1}{N_{cp}} \sum_{i=1}^{N_{cp}} |\hat{t}_i^{cp} - t_i^{cp}|$$

where $\hat{t}_i^{cp}$ is the estimated change point for the ith change point.

When we evaluate the performance of the prediction models, we calculate the absolute difference between the QoS of the simulation running with the predicted traces and the simulation with the true arrivals. We consider two types of QoS performance in our experiments which are the average response time (ART) and the average service level agreement violation (ASLAV) of the tasks that arrive at the last observation time of the reference sequence. The ART is defined as

$$\text{ART} = \frac{1}{N_{task}} \sum_{i=1}^{N_{task}} rt_i \tag{15}$$

where N_{task} is the number of tasks that arrive at the last observation time of the reference sequence, and rt_i is the response time of the ith task. The ASLAV is formulated as

$$\text{ASLAV} = \frac{1}{N_{task}} \sum_{i=1}^{N_{task}} \max(rt_i - sla_i, 0) \tag{16}$$

where sla_i is the maximum response time contractually stipulated in the SLA of the ith task.

D. Results

1) HCPD evaluation: We first evaluate the performance of HCPD on the synthetic dataset with BOCPD and KCPD as the baseline models. Table I shows the performance metrics of the CPD models. While BOCPD has the lowest FPR at 0.23%, the FPR difference among the three models is minimal, with KCPD at 0.24% and HCPD at 0.25%. Conversely, the MDD of HCPD is 106.88, which is 16% and 53% lower than for BOCPD and KCPD, respectively. Also, HCPD has the lowest MAE of the distance between the estimated change point and the true change point compared to BOCPD and KCPD.

2) Framework evaluation: To evaluate the performance of our proposed online adaptive framework, we run simulations with the synthetically generated trace and the Alibaba arrival trace after the 7000th step, using ART and ASLAV as sample labels at each step. The framework and the other three baseline models make forecasting at each step. We also evaluate the effect of the adaptive mechanism by removing HCPD and fine-tuning the procedure in the framework and refer to it as the

TABLE II: The MAE of ART and ASLAV estimated with forecasting models on Synthic trance and Alibaba trace

Dataset		Synthetic trace	Alibaba trace
Mean	ART	24.97	40.33
	ASLAV	4.58	18.59
MA	ART	24.17	52.84
	ASLAV	7.45	23.57
ARIMA	ART	26.16	47.87
	ASLAV	4.27	22.96
Our Model	ART	**15.28**	**31.74**
	ASLAV	**3.09**	**11.21**
Model w/o adapt.	ART	24.72	34.87
	ASLAV	5.38	12.68

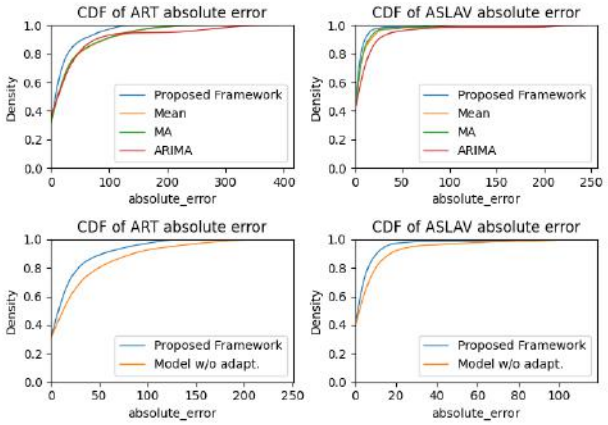

Fig. 3: The CDFs of the absolute error of the QoS metrics on synthetically generated trace

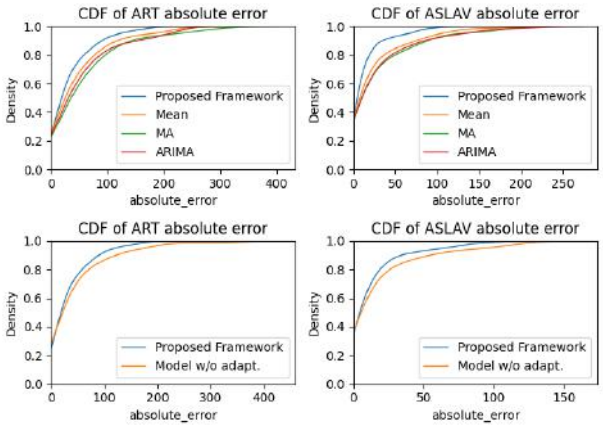

Fig. 4: The CDFs of the absolute error of the QoS metrics on Alibaba trace

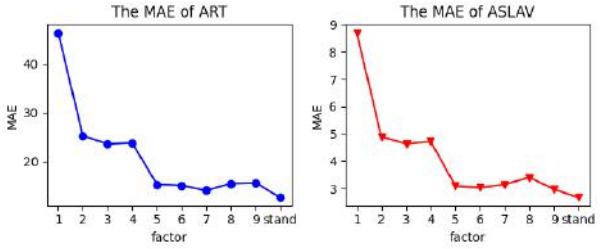

Fig. 5: The MAE of ART and ASLAV with different factors c of the *ProbSparse* attention testing on synthetically generated trace. *Stand* represent the prediction model with standard attention function

model without adaptation. The digital-twin co-simulator runs simulations with the predicted arrivals to estimate the ART and ASLAV of the corresponding step. The performance of the model is measured by the absolute difference between the labelled estimated ART and ASLAV.

Table II shows the MAE of the ART and ASLAV estimated by the co-simulator with arrivals estimated by the corresponding models. Our model outperforms the other baseline models with 37% lower MAE of ART and 28% lower MAE of ASLAV testing on the synthetic trace. In addition, the proposed model also achieves the best performance with 27% and 39% lower MAE on ART and ASLAV, respectively, on the Alibaba trace.

Figure 3 and 4 show the cumulative distribution function (CDF) of the absolute error of ART and ASLAV on synthetic trace and Alibaba trace. These figures present that most of the traces predicted by the proposed model can well represent future arrivals. Therefore, the QoS estimated by the simulator using the adaptive framework is more accurate compared to the other baseline methods.

These tables and figures also show that the adaptation mechanism can help the arrivals forecasting model to adapt to the concept drift in the arrival trace. The MAE of ART and ASLAV estimated with the adaptive framework are 38% and 42% lower than the one without the adaption mechanism on the synthetic trace and 8% and 12% lower when testing on Alibaba trace. The adaptive framework is also more robust compared to the one without adaptation.

We evaluate the complexity of our framework by its average prediction time, inclusive of fine-tuning. The MA and ARIMA models take 0.074 and 0.083 seconds respectively on the synthetic trace, and 0.081 and 0.087 seconds on the Alibaba trace. Our adaptive framework takes 0.072 seconds on the synthetic trace and 0.094 seconds on the Alibaba trace. The processing speed of our framework is comparable to MA, 15% faster than ARIMA on the synthetic trace, but 7% slower than MA on the Alibaba trace due to more frequent change detection and fine-tuning.

3) Sensitivity Analysis: A sensitivity analysis is conducted on the factor c in the $ProbSparse$ attention function, which determines the number of queries used in the attention calculations and impacts the fine-tuning and inference speed of the prediction model. We also compare the $ProbSparse$ attention with the standard attention function to evaluate the effectiveness of the $ProbSparse$ attention. We set the batch size to 32, and the fine-tuning epoch is set to 5 during the fine-tuning procedure.

Figure 5 indicates that as the factor increases, the MAE of ART and ASLAV decreases due to more queries influencing the weighted sum via attention instead of simply averaging

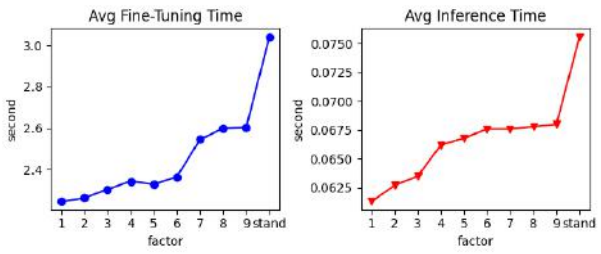

Fig. 6: The average fine-tuning time and inference time consumed by the framework with different factors of the ProbSparse attention testing on synthetically generated trace.

the series. However, Figure 6 shows increasing computation speed during fine-tuning and inference with a higher factor, as more queries require attention computation. Despite negligible improvement in MAEs with standard attention beyond a factor of 5 and slower speeds, the proposed framework using $ProbSparse$ attention with this factor ensures efficient arrivals estimation and quick inference.

VII. Conclusion

In this paper, we have presented an online adaptive arrivals prediction framework, based on hierarchical change point detection, to monitor and detect concept drift and a probabilistic transformer model to forecast arrivals. The proposed framework can leverage a co-simulator to adapt the concept drift in arrival traces and make better QoS estimation for scheduling. The experiments on synthetic trace and real-world arrival trace demonstrate that the proposed online adaptive framework has the ability to adapt the concept drift in the trace and provide valuable arrivals to the co-simulator to achieve better QoS estimation.

Future work may apply the proposed framework in the digital-twin co-simulation to assist gradient base schedulers, such as GOBI and GOBI* [2], to make better scheduling decisions in a dynamic Fog system. The framework can also be used in other scenarios, where the digital-twin model of a system requires adaptation to a dynamic environment.

Acknowledgments

We thank Yingzhen Li (Imperial College London) for providing assistance and comments on this work.

References

[1] D. Ardagna, G. Casale, M. Ciavotta, J. F. Pérez, and W. Wang, "Quality-of-service in cloud computing," *JISA*, Sept. 2014.
[2] S. Tuli, S. R. Poojara, S. N. Srirama, G. Casale, and N. R. Jennings, "COSCO: Container Orchestration Using Co-Simulation and Gradient Based Optimization for Fog Computing Environments," *IEEE TPDS*, Jan. 2022.
[3] S. Tuli, G. Casale, and N. R. Jennings, "GOSH: Task Scheduling Using Deep Surrogate Models in Fog Computing Environments," *IEEE TPDS*, Nov. 2022.
[4] C. Alippi, G. Boracchi, and M. Roveri, "Hierarchical Change-Detection Tests," *IEEE TNNLS*, Feb. 2017.
[5] A. Inc., "Alibaba production cluster data v2018," 2018. Accessed: 2023-06-01.
[6] J. Ullman, "NP-complete scheduling problems," *JCSS*, June 1975.
[7] A. Beloglazov and R. Buyya, "Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in Cloud data centers," *Concurrency and Computation*, 2012.
[8] K. Han, Z. Xie, and X. Lv, "Fog Computing Task Scheduling Strategy Based on Improved Genetic Algorithm," *Computer Science*, Apr. 2018.
[9] S. Tuli, S. Ilager, K. Ramamohanarao, and R. Buyya, "Dynamic Scheduling for Stochastic Edge-Cloud Computing Environments Using A3C Learning and Residual Recurrent Neural Networks," *IEEE TMC*, Mar. 2022.
[10] Z. Tang, X. Zhou, F. Zhang, W. Jia, and W. Zhao, "Migration Modeling and Learning Algorithms for Containers in Fog Computing," *IEEE TSC*, Sept. 2019.
[11] S. Kuvattana, S. Sukparungsee, P. Busababodhin, and Y. Areepong, "Performance Comparison of Bivariate Copulas on the CUSUM and EWMA Control Charts," p. 5, 2015.
[12] A. Mukherjee, M. A. Graham, and S. Chakraborti, "Distribution-Free Exceedance CUSUM Control Charts for Location," *CS-SC*, May 2013.
[13] J. Li, X. Zhang, and D. R. Jeske, "Nonparametric multivariate CUSUM control charts for location and scale changes," *Journal of Nonparametric Statistics*, Mar. 2013.
[14] D. M. Hawkins, P. Qiu, and C. W. Kang, "The Changepoint Model for Statistical Process Control," *JQT*, Oct. 2003.
[15] Z. Harchaoui, E. Moulines, and F. Bach, "Kernel Change-point Analysis," in *Advances in NeurIPS*, vol. 21, 2008.
[16] R. P. Adams and D. J. C. MacKay, "Bayesian Online Changepoint Detection," Oct. 2007.
[17] G. E. P. Box, *Time series analysis : forecasting and control.* Wiley series in probability and statistics, Wiley, 2016 - 2016.
[18] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
[19] D. Salinas, V. Flunkert, J. Gasthaus, and T. Januschowski, "DeepAR: Probabilistic forecasting with autoregressive recurrent networks," *IJF*, July 2020.
[20] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is All you Need," in *Advances in NeurIPS*, 2017.
[21] I. Beltagy, M. E. Peters, and A. Cohan, "Longformer: The Long-Document Transformer," Dec. 2020.
[22] R. Child, S. Gray, A. Radford, and I. Sutskever, "Generating Long Sequences with Sparse Transformers," Apr. 2019.
[23] Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. Le, and R. Salakhutdinov, "Transformer-XL: Attentive Language Models beyond a Fixed-Length Context," in *Proc. of the 57th AM of the ACL*, July 2019.
[24] A. I. Goldman, "Issues in designing sequential stopping rules for monitoring side effects in clinical trials," *Controlled Clinical Trials*, Dec. 1987.
[25] H. Zhou, S. Zhang, J. Peng, S. Zhang, J. Li, H. Xiong, and W. Zhang, "Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting," *Proc. of AAAI*, May 2021.
[26] Y.-H. H. Tsai, S. Bai, M. Yamada, L.-P. Morency, and R. Salakhutdinov, "Transformer Dissection," in *Proc. of the EMNLP-IJCNLP*, Jan. 2019.
[27] R. B. Crosier, "Multivariate Generalizations of Cumulative Sum Quality-Control Schemes," *Technometrics*, Aug. 1988.
[28] G. J. Ross, D. K. Tasoulis, and N. M. Adams, "Nonparametric Monitoring of Data Streams for Changes in Location and Scale," *Technometrics*, Nov. 2011.
[29] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *Proc. of the CVPR*, pp. 770–778, IEEE, June 2016.
[30] S. Shen, V. Van Beek, and A. Iosup, "Statistical Characterization of Business-Critical Workloads Hosted in Cloud Datacenters," in *Proc. of the CCGC*, May 2015.
[31] A. H. Gee, J. Chang, J. Ghosh, and D. Paydarfar, "Bayesian Online Changepoint Detection Of Physiological Transitions," in *Proc. of the IEEE EMBC*, pp. 45–48, July 2018.
[32] F. Desobry, M. Davy, and C. Doncarli, "An online kernel change detection algorithm," *IEEE TSP*, Aug. 2005.
[33] G. E. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.
[34] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," Jan. 2017.