Fast and Fine-grained Autoscaler for Streaming Jobs with Reinforcement Learning

Mingzhe Xing\(^1\), Hangyu Mao\(^2\) and Zhen Xiao\(^1\)

\(^1\)School of Computer Science, Peking University
\(^2\)Huawei Noah’s Ark Lab
mzxing@stu.pku.edu.cn, maohangyu1@huawei.com, xiaozhen@pku.edu.cn

Abstract

On computing clusters, the autoscaler is responsible for allocating resources for jobs or fine-grained tasks to ensure their Quality of Service. Due to a more precise resource management, fine-grained autoscaling can generally achieve better performance. However, the fine-grained autoscaling for streaming jobs needs intensive computation to model the complicated running states of tasks, and has not been adequately studied previously. In this paper, we propose a novel fine-grained autoscaler for streaming jobs based on reinforcement learning. We first organize the running states of streaming jobs as spatio-temporal graphs. To efficiently make autoscaling decisions, we propose a Neural Variational Subgraph Sampler to sample spatio-temporal subgraphs. Furthermore, we propose a mutual-information-based objective function to explicitly guide the sampler to extract more representative subgraphs. After that, the autoscaler makes decisions based on the learned subgraph representations. Experiments conducted on real-world datasets demonstrate the superiority of our method over six competitive baselines.

1 Introduction

At present, massive streaming data, e.g., video stream, is generated incessantly on service-based applications, and processed by streaming jobs [Sun et al., 2019]. Each job is comprised of multiple tasks that execute specific computing operations. In order to minimize response latency and to improve the usage efficiency of computing resources, it is critical to design an autoscaler [Nguyen et al., 2020], which is responsible for allocating computing resources to jobs or fine-grained tasks, namely job-level and task-level autoscaling, respectively. Previous work has demonstrated that fine-grained resource management can generally achieve performance improvement in various computing scenarios due to a more precise resource allocation, e.g., 11-x faster execution speed for web services [Qiu et al., 2020] and 35% gain on GPU utilization [Yu et al., 2018]. It is valuable and promising to design an effective task-level autoscaler for streaming jobs.

Designing an optimal autoscaler is known to be NP-hard [Gari et al., 2021]. An easy approach is to use heuristics, namely heuristic-based autoscalers [Verma et al., 2015]. They are typically based on heuristic rules meticulously tuned by experts, which is not only a time-consuming process but also subject to human cognitive bias. To remedy these drawbacks, another line of work [Mao et al., 2016] formulates the autoscaling process as Markov Decision Process (MDP), and adopt Reinforcement Learning (RL) to train the autoscalers, namely RL-based autoscalers.

Despite the superiority of RL-based autoscalers, how to autoscale streaming jobs at task-level with RL remains a challenging problem and has not been adequately studied. As shown in Fig. 1, DeepRM [Mao et al., 2016], DREAM [Ni et al., 2020] and DeepWave [Sun et al., 2020] used RL to autoscale batch jobs that process deterministic size of input data and cannot deal with streaming jobs with time-variant workloads. Although TVW-RL [Mondal et al., 2021] considered the temporal patterns of dynamic workloads, it focused on job-level autoscaling rather than task-level, which neglected the topological task dependencies within a job and thus caused poor performance as shown in the experiments. Another obstacle is the large temporal dimension issue. Typically, the streaming jobs will be running online for months or even years [Hueske and Kalavri, 2019] and produce massive records of job states. It brings heavy computation overhead to model the recorded fine-grained task states, which will harm the autoscaling efficiency of time-critical stream computing.

To address the above issues, we propose a novel task-level autoscaler for stReaiming jobs with reinforcement lEarning (named SURE). First, we give an MDP formulation that can accurately describe the autoscaling process. More
specifically, as a streaming job can normally be encoded as a Directed Acyclic Graph (DAG) [Hueske and Kalavri, 2019], we organize the running states of streaming jobs as Spatio-Temporal Graphs (STGs), which take both the topological task dependencies and temporal workloads evolution into consideration and can better model the dynamic task states. Second, due to the large temporal dimension issue, it is inefficient to learn the entire STG. To alleviate this problem, we further design a novel Neural Variational Subgraph Sampler to extract informative subgraphs. While the subgraph sampler can be implicitly optimized during policy training, we propose an objective function based on mutual information maximization, which can explicitly enhance and guide the sampler to extract more representative subgraphs. After that, we leverage Graph Neural Network (GNN) to learn the sampled subgraphs as the representation of the entire STG, which can accelerate the model inference speed significantly. Finally, RL is applied to train the autoscaler.

To our best knowledge, we are the first to utilize RL to learn task-level autoscaler for streaming jobs. Extensive experiments conducted on real-world datasets demonstrate that our method outperforms six baselines. In addition, ablation study, parameter sensitive analysis and visualized case study are provided for better understanding of our work.

2 Related Work

Heuristic-based Autoscaler. HPA [Nguyen et al., 2020] periodically configured the number of replicas based on the monitored metrics, e.g., CPU utilization ratio, and the corresponding desired metric values. Graphene [Grandl et al., 2016] identified tasks that cost a long time to complete, and selected the schedule order with the minimum running time. Voilà [Fahs et al., 2020] scale-up or scale-down pods by identifying high network latency or overloaded replicas.

Reinforcement-learning-based Autoscaler. Although heuristic-based methods have been widely deployed in industry, they are sub-optimal by nature. To tackle this issue, DeepRM modeled the resource states as bitmap, and applied RL to schedule resources. DeepWeave employed GNN to process DAG information and rewarded solutions with faster Job Completion Time. DREAM designed an Encoder-Decoder framework with GNN and RL to schedule jobs according to their dependencies. However, the aforementioned methods can only schedule batch jobs with static workloads. TVW-RL exploited the temporal patterns of time-varying workloads and used RL to improve the metrics for operational excellence.

Spatio-temporal Graph Modeling. Spatio-temporal graph neural network is becoming growingly important in modeling time evolutionary spatial data. ASTGCN [Guo et al., 2019] consisted of three independent components to model temporal properties, where each component contained the spatio-temporal convolution and attention mechanism to effectively capture the dynamic spatio-temporal correlations. CCRNN [Ye et al., 2021] constructed learnable adjacency matrices in different layers and used a layer-wise coupling mechanism to capture the multi-level spatial dependence and temporal dynamics simultaneously.

3 Problem Definition

To begin with, we give a formal definition of task-level autoscaling for streaming jobs. Given a streaming job $j$, it can be abstracted as a DAG, denoted as $\langle V, E, P \rangle$, where $V$ is the node set and each node denotes a task in job $j$, and $E$ represents the dependencies between tasks, and $P$ denotes how many units of resources are allocated to task nodes, namely parallelism. Job $j$ continuously receives and processes time-varying size of input data, e.g., video stream. As shown in Fig. 2, for every $L$ minutes (named rescaling span), the autoscaler needs to allocate more resources (scale-up parallelism) for task nodes if the workloads increase, and release free resources (scale-down parallelism) for tasks with low workloads. In the meantime, a monitor records the job DAG states every minute, which can produce $L$ state snapshots $[G_1, G_2, \ldots, G_L]$ during a rescaling span, where $G_t$ is the state of job DAG at the $t$-th minute in rescaling span. In $G_t$, the $i$-th task node, denoted as $v_{i,t}$, is associated with a feature vector $s_{i,t} \in \mathbb{R}^D$, where the elements in $s_{i,t}$ are system metrics recorded by monitor, including average and summation of received data size, CPU usage, parallelism and average computation time of $v_{i,t}$, and $D$ is the feature dimension. There are two typical performance metrics for streaming jobs, namely latency and resource utilization ratio. Specifically, the latency is defined as the duration of a unit of streaming data completely processed by the job, and the resource utilization ratio denotes the proportion of resource usage.

In order to train the autoscaler with RL, we use a tuple $(State, Action, Transition, Reward)$ to formulate the autoscaling process as Markov Decision Process (MDP) that can accurately describe the dynamic task states based on the above notations. This MDP tuple can be specified as follows:

- **State.** At the $t$-th step, by connecting the consecutive $L$ snapshots recorded from the $(t-1)$-th to $t$-th step, we can build a spatio-temporal graph $G_{t-1}$ as illustrated in Fig. 2, where each node $v_{i,t}$ is associated with feature vector $s_{i,t}$.

- **Action.** Action $a_{i,t}$, a decimal in $[0.0, 2.0]$, is defined as the ratio between desired parallelism $p_{i,t}$ and current parallelism $p_{i,t-1}$ of task node $v_i$. This is based on an empirical setting [Taherizadah and Grobelnik, 2020] that, for each rescaling operation, the parallelism value cannot be reduced to zero or scaled up to more than twice of previous value.

- **Transition.** Given the autoscaling decision $a_{i,t}$ conditioned on $G_{t-1}$, task node $v_i$ changes its parallelism to $p_{i,t}$. The streaming job then continues running for $L$ minutes, and generates a spatio-temporal graph $G_t$ as the new state.
In this part, we aim to tackle the

4.1 Neural Variational Subgraph Sampler
ploy RL to train the autoscaler policy. Reduced by mutual information maximization. Finally, we represent subgraphs, we propose an objective function de-

making. Furthermore, in order to explicitly extract more rep-
dating correlation of spatial and temporal domains, we aim to

learn the joint spatial and temporal importance weights dis-
stribution $v_i$, where $\theta_i$ is the importance weights along temporal dimension of task node $v_i$, and $\phi_i$ is the spatial sampling weights of $v_i$ at the $l$-th snapshot, and $S_i$ denotes the running states of $v_i$. First, for node $v_i$, we build an input tensor $S_i \in \mathbb{R}^{L \times (1+K) \times D}$ from the $L$ recorded snapshots, where $K = |N_i(v_i)|$ is the number of $c$-hop neighbor nodes of $v_i$. As shown in Fig. 3, the first column (in gray) of $S_i$ is the feature vectors $[s_{i,1}, s_{i,2}, \ldots, s_{i,L}]^T$ of nodes $\{v_{i,1}, v_{i,2}, \ldots, v_{i,L}\}$, and the last $K$ elements (in blue) at the $l$-th row are the feature vectors of the $c$-hop neighbors of $v_{i,l}$ (for brevity, we hide the feature dimension $D$ in Fig. 3, and set $L = 5$ and $K = 4$, so the shape of $S_i$ is $5 \times (1+4)$).

① Input Tensor Construction. Our goal is to derive the temporal and spatial sampling probability distributions $p(\theta_i|S_i)$ and $p(\phi_i|S_i)$, where $\theta_i$ is the importance weights along temporal dimension of task node $v_i$, and $\phi_i$ is the spatial sampling weights of $v_i$ at the $l$-th snapshot, and $S_i$ denotes the running states of $v_i$. First, for node $v_i$, we build an input tensor $S_i \in \mathbb{R}^{L \times (1+K) \times D}$ from the $L$ recorded snapshots, where $K = |N_i(v_i)|$ is the number of $c$-hop neighbor nodes of $v_i$. As shown in Fig. 3, the first column (in gray) of $S_i$ is the feature vectors $[s_{i,1}, s_{i,2}, \ldots, s_{i,L}]^T$ of nodes $\{v_{i,1}, v_{i,2}, \ldots, v_{i,L}\}$, and the last $K$ elements (in blue) at the $l$-th row are the feature vectors of the $c$-hop neighbors of $v_{i,l}$ (for brevity, we hide the feature dimension $D$ in Fig. 3, and set $L = 5$ and $K = 4$, so the shape of $S_i$ is $5 \times (1+4)$).

② Sampling Distribution Inference. To capture the underlying correlation of spatial and temporal domains, we aim to learn the joint spatial and temporal importance weights distribution $p(\Omega_i|S_i) = p(\theta_i, \phi_i|S_i)$ instead of learning $p(\theta_i|S_i)$ and $p(\phi_i|S_i)$ separately. In specific, we design a variational inference network to obtain $p(\Omega_i|S_i)$. Here, we assume $p(\Omega_i|S_i)$ follows Gaussian distribution $N(\mu_i, \Sigma_i)$, and derive the mean $\mu_i \in \mathbb{R}^{L \times (1+K)}$ and co-variance $\Sigma_i \in \mathbb{R}^{L \times (1+K) \times (1+K)}$:

$$\mu_i = \sigma(W_1^T S_i), \quad \Sigma_i = \sigma(W_2^T S_i),$$

where $\sigma$ is activation function, and $W_1$ and $W_2$ are learnable parameters. Then, the joint spatio-temporal sampling distribution $\Omega_i \in \mathbb{R}^{L \times (1+K)}$ can be drawn from $N(\mu_i, \Sigma_i)$. We

More specifically, motivated by the weighted video stream sampling models [Zhi et al., 2021], we assume that there is an underlying importance weights distribution of snapshots for each task node along the temporal dimension, and only a subset of snapshots is salient and valuable. Analogously, from the spatial perspective, we also assume that only a subset of spatial neighbors is most relevant with a specific node inspired by Graph Attention Network [Velickovic et al., 2018]. Aside from lower computation cost, another benefit of introducing the weighted sampling mechanism is to reduce the noise or irrelevant features that could lead to performance degradation. Based on the above assumptions, we design a novel Neural Variational Subgraph Sampling (named NVSS) strategy, consisting of the following four steps.

① Input Tensor Construction. Our goal is to derive the temporal and spatial sampling probability distributions $p(\theta_i|S_i)$ and $p(\phi_i|S_i)$, where $\theta_i$ is the importance weights along temporal dimension of task node $v_i$, and $\phi_i$ is the spatial sampling weights of $v_i$ at the $l$-th snapshot, and $S_i$ denotes the running states of $v_i$. First, for node $v_i$, we build an input tensor $S_i \in \mathbb{R}^{L \times (1+K) \times D}$ from the $L$ recorded snapshots, where $K = |N_i(v_i)|$ is the number of $c$-hop neighbor nodes of $v_i$. As shown in Fig. 3, the first column (in gray) of $S_i$ is the feature vectors $[s_{i,1}, s_{i,2}, \ldots, s_{i,L}]^T$ of nodes $\{v_{i,1}, v_{i,2}, \ldots, v_{i,L}\}$, and the last $K$ elements (in blue) at the $l$-th row are the feature vectors of the $c$-hop neighbors of $v_{i,l}$ (for brevity, we hide the feature dimension $D$ in Fig. 3, and set $L = 5$ and $K = 4$, so the shape of $S_i$ is $5 \times (1+4)$).

② Sampling Distribution Inference. To capture the underlying correlation of spatial and temporal domains, we aim to learn the joint spatial and temporal importance weights distribution $p(\Omega_i|S_i) = p(\theta_i, \phi_i|S_i)$ instead of learning $p(\theta_i|S_i)$ and $p(\phi_i|S_i)$ separately. In specific, we design a variational inference network to obtain $p(\Omega_i|S_i)$. Here, we assume $p(\Omega_i|S_i)$ follows Gaussian distribution $N(\mu_i, \Sigma_i)$, and derive the mean $\mu_i \in \mathbb{R}^{L \times (1+K)}$ and co-variance $\Sigma_i \in \mathbb{R}^{L \times (1+K) \times (1+K)}$:

$$\mu_i = \sigma(W_1^T S_i), \quad \Sigma_i = \sigma(W_2^T S_i),$$

where $\sigma$ is activation function, and $W_1$ and $W_2$ are learnable parameters. Then, the joint spatio-temporal sampling distribution $\Omega_i \in \mathbb{R}^{L \times (1+K)}$ can be drawn from $N(\mu_i, \Sigma_i)$. We

Fig. 3: The overall architecture of our proposed approach. It shows an example to sample a subgraph for task node $v_2$, and then make autoscaling decision for this task node. $L$, $K$, $k_1$ and $k_2$ are set as 5, 4, 3 and 2 in this example. “FFN” denotes the feed forward network. The steps labeled with ①, ②, ③ and ④ correspond to the four steps introduced in Section 4.1.

- **Reward.** The reward $r_t$ is designed to minimize latency $l_t$ and maximize resource utilization ratio $u_t$:

$$r_t = -\lambda l_t + (1 - \lambda)u_t,$$

where $\lambda \in [0, 1]$ controls the importance of each metric and can be tuned according to specific jobs.

With the above MDP formulation, the autoscaler can make decisions based on the topological task dependencies and temporal workloads dynamics implied in the spatio-temporal graphs. However, since the rescaling span $L$ is usually large for streaming jobs [Nguyen et al., 2020], i.e., the large temporal dimension issue, the recorded job snapshots are massive, which makes the spatio-temporal graph too huge to be directly learned with conventional Spatio-Temporal GNNs [Ye et al., 2021], and brings heavy computation overhead for state modeling and decision-making. Therefore, we need to design a graph learning method, which can accelerate the decision inference speed without performance degradation.

4 Methodology
In this section, we present the learning details of our ap-

proach. First, we propose a novel Neural Variational Sub-

graph Sampler. By learning the sampled informative sub-

graphs, it can reduce the time cost of autoscaling decision-

making. Furthermore, in order to explicitly extract more re-

desonable subgraphs, we propose an objective function de-

duced by mutual information maximization. Finally, we em-

ploy RL to train the autoscaler policy.

4.1 Neural Variational Subgraph Sampler
In this part, we aim to tackle the large temporal dimension issue and effectively learn the task node representations from giant Spatio-Temporal Graph (STG). Specifically, for each task node in a job DAG, we propose to sample a subgraph from STG, which maintains the most informative features of this task node in STG. By only learning the subgraph instead of the entire STG, it can greatly save the time cost of decision making. Huang and Zitnik also have proved that the information loss of operating GNN on a subgraph rather than the entire graph is bounded by an exponentially decaying term with respect to the node number of subgraph.
can obtain the \( \theta_i \) and \( \phi_i \) from \( \Omega_i \) as follows:

\[
\theta_i = (\Omega_{m,n})_{1 \leq m \leq L}, \quad \phi_i = (\Omega_{m,n})_{n = 1 \leq n \leq K + 1},
\]

where \( \theta_i \in \mathbb{R}^L \) and \( \phi_i \in \mathbb{R}^K \) are the first column and the last \( K \) elements at the \( l \)-th row from \( \Omega_i \), respectively.

3. Subgraph Sampling. Next, we sample a subgraph for node \( v_i \) using \( \theta_i \) and \( \phi_i \). First, \( k_1 \) informative snapshots can be sampled from Multinomial(\( \theta_i \)) for node \( v_i \). We name the \( v_i \) in the corresponding sampled snapshots as temporal anchor nodes, e.g., \( v_{2,1}, v_{2,4}, v_{2,5} \) (green nodes) in Fig. 3. Second, for each temporal anchor node \( v_{il} \), \( k_2 \) spatial neighbor nodes (yellow nodes) can be sampled from Multinomial(\( \phi_i \)) in the \( l \)-th snapshot, where the spatial neighbor nodes are from the \( c \)-hop neighbors set of \( v_i \), i.e., \( \mathcal{N}_c(v_i) \). In this way, we can obtain the sampled temporal anchor nodes, and the corresponding spatial neighbor nodes. By connecting temporal anchor nodes as depicted in Fig. 3, we can reconstruct the sampled subgraph \( g_i \) for task node \( v_i \). Under this sampling procedure, the marginal likelihood of \( g_i \) is:

\[
p(g_i|S_i) = \prod_{l=1}^{k_1} \prod_{s=1}^{k_2} p(v_{il}|\phi_i)p(v_{il}|\theta_i)p(\theta_i, \phi_i|S_i), \tag{2}
\]

where \( v_{il} \) and \( v_{is} \) are the sampled spatial neighbor and temporal anchor nodes, respectively.

Note that the sampling processes include drawing \( \Omega_i \) from \( \mathcal{N}(\mu_i, \Sigma_i) \) and sampling nodes from \( p(\theta) \) and \( p(\phi) \), but they are undifferentiable, making back-propagation inapplicable. To overcome this issue, we adopt the Reparameterization Trick [Kingma et al., 2015] that can be specified as follows:

\[
\begin{align*}
x & \sim \mathcal{N}(0, \text{diag}(I)) \quad y \sim \text{Gumbel}(0,1) \\
\Omega_i &= \log(x \times \Sigma_i + \mu_i) + y,
\end{align*}
\]

where \( x \) and \( y \) are sampled random variables from Gaussian and Gumbel distribution respectively, and \( \text{diag}(I) \) is a diagonal matrix with all-one diagonal elements. In this way, we can transform the undifferentiable factor to \( x \) and \( y \), while \( \mu_i \) and \( \Sigma_i \) are differentiable and can be optimized during training.

4. Autoscaling Decision Making. After obtaining the sampled subgraph \( g_i \), we utilize GraphSAGE [Hamilton et al., 2017] to learn subgraph representation as the state of task node \( v_i \), which can save considerable amount of state modeling time compared with learning the entire STG (detailed time complexity analysis can be found in Appendix B). The message passing process of GraphSAGE can be specified as:

\[
h_{ei} = \sigma(W_3^{t}h_{ei-1}^{t-1}, \sum_{u \in \mathcal{M}(i)} h_{ui}^{t-1})_{[\mathcal{M}(i)]},
\]

where \( W_3^t \) is the learnable parameter at the \( e \)-th layer of GraphSAGE, and \( \mathcal{M}(i) \) is the neighbor nodes set of \( v_i \). The input \( h_{1} \) of the first layer of GraphSAGE is the node features \( \{s_u\}_{u \in g} \). Recall that subgraph \( g_i \) is sampled with respect to node \( v_i \), we can regard \( h_{ei} = \text{Readout}(h_{ei}^{t}|v_i \in g_i) \) as the representation of \( v_i \), where the Readout function is average pooling and \( E \) is the number of GraphSAGE layers.

Finally, a linear layer acts as the agent to make autoscaling decision for \( v_i \) based on the state representation \( h_{ei} \):

\[
a_i = \sigma(W_4^t h_{ei}),
\]

where \( a_i \) is the action decision of autoscaling (introduced in Section 2), and \( W_4^t \) is a learnable parameter.

4.2 Enhancing NVSS via Mutual Information

In the previous section, NVSS can be implicitly learned to sample informative spatio-temporal subgraphs, due to the existing of reward signal from RL. In this section, we explicitly encourage the sampled subgraphs to reveal the most representative topological information of the entire STG, and enhance NVSS to extract more representative subgraphs.

Specifically, we would like to minimize the semantic divergence between the sampled subgraph \( g_i \) and the entire STG \( G \). Since the Mutual Information (MI) [Kraskov et al., 2004] measures the mutual dependence between two variables, and a larger MI means that the two variables are more correlated, we propose an objective function from the MI perspective as follows (we omit the subscript of \( g_i \) for concision):

\[
\max I(f(g), f(G)) = H(f(g)) - H(f(g)|f(G)), \tag{4}
\]

where \( H \) denotes the entropy of the given distribution, and \( f \) denotes the GraphSAGE that extracts features for graphs.

With the likelihood distribution of subgraph \( q \) (Eq. 2), the lower bound of objective function (Eq. 4) can be derived as follows (please see Appendix A for detailed derivation):

\[
Y(g, G) = \log q_{g \sim p(g|S)}(f(g), f(G)) \geq \sum_g \left( -\mathcal{KL}(q(\Omega|S)||p(\Omega|g, S)) - \frac{1}{2} \log |\Sigma| + (\Omega - \mu)^T \Sigma^{-1} (\Omega - \mu) + CE(\Omega, \hat{\Omega}) \right) + \mathbb{E}_q \log I(f(g), f(G)),
\]

where \( \mathcal{KL} \) represents KL-divergence [Anzai, 2012], and \( CE \) denotes Cross Entropy loss, and \( \Omega \) is the joint distribution of actually sampled temporal anchor and spatial neighbor nodes.

The objective function (Eq. 6) includes three parts, i.e., KL-divergence, log-likelihood of prior distributions and log of MI, which are intuitive. The KL-divergence term attempts to minimize the distance between the estimated distribution \( q \) and the true distribution \( p \). The second term is regularization term of prior distributions. The log of MI ensures that the sampled subgraphs contain the most representative information of the entire STG. The first and second terms in Eq. 6 are easy to compute. As for the MI term, we adopt the Jensen-Shannon MI estimator [Nowozin et al., 2016] as follows:

\[
I(f(g), f(G)) = \mathbb{E}_p[-\text{sp}(D_w(f(g), f(G)))] - \mathbb{E}_q[\text{sp}(D_w(f(g), f(G)))]
\]

where \( D_w \) is a discriminator that takes a subgraph and STG embeddings pair as input and determines whether the subgraph is sampled from the STG, and \( \text{sp}(z) = \log(1 + e^z) \) is the softplus function, and \( g' \) is negative sampled subgraph from \( p = p \). In practice, we generate negative subgraphs by using all possible combinations of STG and subgraph embeddings across all graph instances in a mini-batch.

Indeed, there have been a few studies that used subgraph sampling and mutual information techniques simultaneously, however, our method has two major differences. First, NVSS can capture the underlying correlation of spatial and temporal domains, and be end-to-end optimized, while the samplers in previous work are heuristic-based [Sun et al., 2021] or cannot perform sampling for a specific node along the temporal
dimension [Yu et al., 2020], which cannot be applied in this scenario and compared with our method. Second, we seek for a seamless integration of the subgraph sampling and mutual information rather than a simple combination of the two techniques. With the derived objective function (Eq. 6), the two parts can be jointly trained and mutually optimized. Moreover, the objective function is also intuitive and interpretable.

4.3 Training with Reinforcement Learning

Here, we use RL to optimize the autoscaler policy. The expected sum of rewards is $R = \mathbb{E}_\pi [\sum_{t=0}^{\infty} \gamma^t r_t]$, where $\pi$ represents the policy that makes autoscaling decisions with Eq. 3, and $\gamma \in (0, 1]$ is a discount factor that reduces future rewards relative to current reward, and $r_t$ is the reward function as defined in Section 2. The objective function that needs to be maximized for optimizing the autoscaler policy is: $J(\psi) = \frac{1}{N} \sum_{n=1}^{N} \log \pi_{\psi} R$, where $\psi$ is the model parameters. Combined with the MI-based objective function (Eq. 6), the total loss of our model is $L = -J - \lambda_1 \sum_{i=1}^{V} Y_i$, where $\lambda_1$ is a hyperparameter that adjusts the weight of the MI-based loss. More learning details can be found in Appendix C.

5 Experiments

5.1 Experimental Setup

Simulation Environment. Following the streaming job execution logic described in Section 2, we implement a simulated computing system for streaming jobs and register a monitor for each job to record its state snapshots periodically.

Dataset. We use Clarknet Trace \(^1\) as workloads, which describes the number of HTTP requests to the servers recorded in 20,000 minutes. The workloads are highly varying in time, and show periodicity characteristics. As for computing jobs, since we focus on long-running streaming jobs, we only keep the jobs in Alibaba Cluster Dataset \(^2\) that were running for more than 2,000 minutes following Mondal et al.. The details of datasets can be found in Appendix D.1.

Experiment Settings. To evaluate the performance of our method on jobs with different task numbers, we randomly sample six jobs and divide them into three sets, namely Small, Medium and Large. More specifically, the task numbers of Small-1, Small-2, Medium-1, Medium-2, Large-1 and Large-2 are 6, 16, 25, 32, 40 and 46, respectively. Each job acts as an individual simulation environment and receives a subsequence of workloads that lasts for $7 \times 24$ hours in ClarkNet Trace in each episode. For each job, we first train an autoscaler and then use it to perform testing on this job for ten times and take the average reward as the final result.

Baseline. We compare our method with baselines from three categories: (1) heuristic-based autoscaler, i.e., HPA; (2) RL-based autoscaler, i.e., DeepWave, DREAM and TVW-RL; and (3) spatio-temporal GNN, i.e., ASTGCN and CCRNN. Due to space limitation, we skip their details that have been presented in related work. For fair comparisons, we use the same features for all baselines, and make slight modifications for them to fit for our scenario. Following DREAM, DeepWave and TVW-RL, we use REINFORCE [Williams, 1992] to train our autoscaler. The details of parameter settings and modifications of all compared methods are in Appendix D.2. The code and Appendix are available at https://github.com/xmzzyo/sure.

5.2 Performance Comparison

We present the standardized rewards of all baselines and our method on Small, Medium, Large job settings in Table 1.

In general, RL-based autoscalers perform better than HPA on most jobs, because their strategies can be optimized regarding specific job environments, while HPA uses heuristic rules and cannot be well adapted to different jobs and dynamic workloads. TVW-RL performs the best among RL-based methods on average, since it can model the temporal patterns of workloads, which is critical when autoscaling streaming jobs. Among all the baselines, CCRNN achieves the best performance on average. It uses a layer-wise coupling mechanism to capture the multi-level dependence of temporal and spatial domains. Our method consistently outperforms all the competitors on all jobs. It strongly supports our claim that the proposed subgraph sampling strategy can extract representative subgraphs, which imply the joint spatial and temporal correlations of the entire STG. By only learning the subgraphs, it can derive effective task representations and make good decisions. Moreover, our sampling mechanism can also alleviate the interference of redundancy information or noise, which can benefit the model performance.

5.3 Ablation Study

To examine the effects of different modules, three variants of our method are compared, including: (A) w/o NVSS denotes using an uniformly random sampler instead of NVSS; (B) w/o

<table>
<thead>
<tr>
<th>Heuristic-based</th>
<th>HPA</th>
<th>DeepWave</th>
<th>DREAM</th>
<th>TVW-RL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-0.17</td>
<td>-2.77</td>
<td>0.50</td>
<td>0.26</td>
</tr>
<tr>
<td>RL-based</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>-1.16</td>
<td>-1.23</td>
<td>-0.23</td>
<td>0.66</td>
</tr>
<tr>
<td>Spatial-temporal GNN</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.26</td>
<td>-0.66</td>
<td>0.36</td>
<td>0.97</td>
</tr>
<tr>
<td>Ours</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.52</td>
<td>1.41</td>
<td>1.19</td>
<td>1.81</td>
</tr>
</tbody>
</table>

Table 1: Performance comparison with baselines on Small, Medium and Large job settings, respectively. The best, second best and third best results are in bold, underline and gray cell, respectively.

\(^1\)ftp://ita.ee.lbl.gov/html/contrib/ClarkNet-HTTP.html

\(^2\)https://github.com/alibaba/clusterdata
Fig. 4(c) and 4(d), we can see that our approach can achieve lower latency and competitive utilization efficiency on most λ settings. Note that we cannot always have improvements on both metrics, since a lower latency would generally cost more or even redundant resources. Nevertheless, the comprehensive performances of our model, i.e., the reward and inference time, are consistently better than the best baseline.

5.5 Case Study

In this part, we show qualitative cases based on Small-1 job to give intuitive impressions of our model. The DAG structure of Small-1 job can be found in Appendix D.4.

Fig. 5(a) presents the heatmap of sampling distribution Ω of task node v1. From the first column of Ω (i.e., θ), we can observe that the most recent snapshots (framed in green) are more likely to be sampled along temporal dimension, which is reasonable and intuitive since they are more related to the states at the next step. Besides, the nearest (1-hop) neighbor nodes of v1, i.e., v2, v3, and v4 (framed in blue), have larger probabilities to be sampled, since they are more topological relevant to v1. In Fig. 5(b), we use t-SNE to visualize the embeddings of subgraphs sampled for task nodes and the entire STG in ten autoscaling steps. As depicted, the embeddings are grouped into four clusters. The v2, v3 and v4 are grouped together as they are 1-hop neighbors of v1, and v5 and v6 are 2-hop and 3-hop neighbors of v1. It demonstrates that our graph embeddings can effectively present the topological features of nodes, and the subgraph embeddings are centralized and closed with the embedding of the entire STG G.

6 Conclusion

In this paper, we propose a novel approach to autoscale streaming jobs at task-level with reinforcement learning. We first organize the job states as spatio-temporal graphs and give a formal MDP formulation of autoscaling process. To efficiently learn the giant spatio-temporal graphs, we design a Neural Variational Subgraph Sampler, which can greatly save the graph learning time. Furthermore, we propose an objective function based on mutual information to guide the sampler to extract more representative subgraphs. Our experiments demonstrate the superior performance and interpretability of our approach. In the future, we will apply our method to solve other classical spatio-temporal graph modeling tasks, such as traffic forecasting and pose detection, which also suffer from the large temporal dimension issue.

<table>
<thead>
<tr>
<th></th>
<th>Small-1</th>
<th>Small-2</th>
<th>Medium-1</th>
<th>Medium-2</th>
<th>Large-1</th>
<th>Large-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>w/o NVSS</td>
<td>0.46</td>
<td>-1.30</td>
<td>0.21</td>
<td>-0.82</td>
<td>-1.27</td>
<td>0.29</td>
</tr>
<tr>
<td>w/o MI</td>
<td>0.47</td>
<td>-0.78</td>
<td>0.22</td>
<td>-0.82</td>
<td>0.78</td>
<td>0.33</td>
</tr>
<tr>
<td>SURE</td>
<td><strong>0.52</strong></td>
<td><strong>1.41</strong></td>
<td><strong>1.19</strong></td>
<td><strong>1.81</strong></td>
<td><strong>1.02</strong></td>
<td><strong>0.95</strong></td>
</tr>
</tbody>
</table>

Table 2: Ablation study of our model. The best results are in bold.
Acknowledgments

We thank Jianye Hao, Dong Li, Chen Chen, and the anonymous reviewers for their insightful comments on this paper. This work was supported by the National Natural Science Foundation of China under Grant No. 61872397. Zhen Xiao is the corresponding author.

References


