SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

Reducing Idleness in Financial Cloud via Multi-objective Evolutionary Reinforcement Learning based Load Balancer

Peng YANG[1,2*], Laoming ZHANG[2,3], Haifeng LIU[4] & Guiying LI[2,5]

[1]*Department of Statistics and Data Science, Southern University of Science and Technology, Shenzhen* 518055*, China;*
[2]*Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen* 518055*, China;*
[3]*Academy for Advanced Interdisciplinary Studies, Southern University of Science and Technology, Shenzhen* 518055*, China;*
[4]*Guangdong OPPO Mobile Telecommunications Corp., Ltd, Shenzhen* 518052*, China;*
[5]*Research Institute of Trustworthy Autonomous Systems, Southern University of Science and Technology, Shenzhen* 518055*, China*

Abstract In recent years, various companies started to shift their data services from traditional data centers onto cloud. One of the major motivations is to save operation costs with the aid of cloud elasticity. This paper discusses an emerging need from financial services to reduce idle servers retaining very few user connections, without disconnecting them from the server side. This paper considers this need as a bi-objective online load balancing problem. A neural network based scalable policy is designed to route user requests to varied numbers of servers for elasticity. An evolutionary multi-objective training framework is proposed to optimize the weights of the policy. Not only the new objective of idleness is reduced by over 130% more than traditional industrial solutions, but the original load balancing objective is slightly improved. Extensive simulations help reveal the detailed applicability of the proposed method to the emerging problem of reducing idleness in financial services.

Keywords Evolutionary Reinforcement Learning, Evolutionary Multi-objective Optimization, Load Balance, Cloud Computing

Citation

1 Introduction

Elasticity forms a cornerstone of cloud computing that helps it succeed in the last decade [1]. In many fields, the number of concurrent users of a cloud service system may vary significantly over time, while the cloud system can elastically reduce the idle servers when the users become fewer and increase the servers and vice versa [2]. This feature has greatly helped save millions of dollars per year for many industries who run their services on the cloud. In financial scenarios (especially stock trading), it is common that the number of concurrent users at the close time of the market is over 10 times smaller than that at the opening time [3]. Traditional financial services focus more on stability rather than elasticity, until recently the cloud native offers new possibility for achieving both in the same architecture [4].

Unfortunately, no industrial solution has been given to readily reduce idle servers in the financial cloud services as it is still a non-trivial task. Ideally, if a server serves zero user connection, it can be shutdown to reduce idleness. In practice, when the user concurrency largely decreases, the servers that become idle may still retain a small number of user connections. Other services like entertainment and office can simply disconnect all these users and re-connect them to the busy servers in a short time. In this regard, these idle servers become truly empty and thus can be shutdown to save costs. However, in the financial services, the availability of the connections is extremely sensitive, as even a millisecond of disconnection

* Corresponding author (email: yangp@sustech.edu.cn, Peng YANG)

may cause a heavy loss of investment and damage to the financial market [5]. This means that an idle server in the financial service must passively wait for all its users (though very few) to disconnect and then can be shutdown afterward, while the waiting time directly induces wastes of the idle servers.

We propose that the above issue is mainly caused by the improper routing decisions of the user requests over the servers. Specifically, given that a server can only be shutdown when it serves no connection, we can know that the shutdown time of a server is decided by the latest disconnection time among all its user connections. Once we can route the user connections with similar disconnection time in the same server, this server can be shutdown in time and the idleness can be greatly reduced. On this basis, this paper focuses on the load balancer module within the cloud computing architecture, as it is responsible for routing user requests to different servers for constructing connections.

As its name suggests, the traditional routing algorithms inside the industrial load balancers mainly focus on minimizing the imbalance of workloads among different servers. Representative routing algorithms are heuristics like Round Robin, Random Routing, IP Hash method , the Least Connection, and their weighted variants [6]. These algorithms are based on either some degree of randomness or some hardware indicators of the servers. None of them considers the behaviors of users, i.e., the user connection duration. This explains why in real cases the idle servers usually retain a few connections when the user concurrency largely decreases. Therefore, though they have greatly contributed to enhancing the capacity (concurrent users) and the stability of the cloud services by balancing workloads, they may perform poor on routing the user requests in regarding to further reducing idle servers without killing connections, as requested in the financial services.

To address this issue, this paper aims to automatically learn the optimal policy for routing incoming user request over the available servers so that the load imbalance and the idle time of all servers can be simultaneously minimized. For this purpose, when routing an incoming user request, we not only re-organize the traditional indicators of the servers along time axis but also explicitly exploit the historical connection duration of that user as new features. Specifically, it is assumed that the connection duration of each user on different days follows similar distributions. Thus the connection duration of an incoming user can be statistically predicted. This assumption is reasonable because most of the users of financial services are from financial companies who usually have regular work schedules. Different degrees of the predictability of users are simulated in the experiments to positively support this assumption.

Due to the online decision-making nature of routing incoming requests, the whole routing process is modeled as a Reinforcement Learning (RL) problem with two objectives, i.e., the load imbalance and the idle time of all available servers. Then, a parameter-sharing neural network based scalable routing policy is designed for two reasons: 1) learning the non-linear relationship between the statistically estimated connection duration and the states of the servers; 2) dealing with the elastic scenarios where the number of servers may change due to the autoscaling or servers' failure. Considering that it is difficult to set proper weights for aggregating the two objectives, an evolutionary multi-objective algorithm based training framework is proposed to optimize the parameters of the neural network based policy. Last, an action mask operator is designed to help stably train the policy. As a result, a Multi-objective Evolutionary Reinforcement Learning based Load Balancer (MERL-LB) is proposed.

Extensive simulations show that MERL-LB can significantly outperform the compared algorithms on the emerging task of reducing idleness without disconnecting user connections. The diverse Pareto optima produced by the evolutionary multi-objective training framework offer various options for the users. Among the diverse policies, the idleness objective can be reduced by over 130% of the traditional methods while the load balancing objective is still slightly improved. Meanwhile, it has been shown that the evolutionary multi-objective training framework facilitates a much faster convergence rate over policy gradients [29]. The proposed scalable routing policy has also been successfully verified against varied number of servers and request loads, with the number of users ranging from 600 to 7500. At last, an emerging sawtooth pattern is observed from the process of MERL-LB based policies, which has been studied by comparing to the patterns of traditional methods in detail to explain its rationality and may shed light on designing novel heuristics in the future. In a Nut Shell, MERL-LB empirically shows to be a powerful solution to the emerging problem of reducing idleness and load imbalance in financial services.

The remainder of this work is as follows. Section 2 reviews the related works. Section 3 describes the problem scenarios. Section 4 first models this problem as an RL problem and then presents the scalable neural network based policy and its evolutionary multi-objective algorithms based training framework. The simulations are conducted in section 5 to verify the effectiveness of the proposed MERL-LB. The conclusions of this work are drawn in the last section.

2 Related works

The load balancer works between the users' applications and the servers of the backend system. In cloud data services, the users' requests come up in an online manner. The load balancer is expected to route each incoming user's request to one of the available servers so that any single server in the data center will not be overloaded. And once the request is routed successfully, a connection is constructed between the user's application and the routed server to bi-directionally transmit data.

Industrial routing algorithms mostly make their decisions with certain degrees of randomness or based on the states of the servers to keep load balance. Typical static routing algorithms include Random Routing, Round Robin, Weighted Round Robin, and IP Hash [7]. Round Robin and Weighted Round Robin distribute requests according to a specific probabilistic distribution among servers. IP Hash maps the IP address to the corresponding server using a hash algorithm. Static routing algorithms are generally simple to implement but lack the ability to adapt their probabilistic distributions to the changing characteristics of the network traffic. The other type of dynamic load balancing algorithms considers the servers' states as input. Examples are Least Connection [6] and the Throttled method [8]. Among them, the Least Connection dynamically assigns tasks based on the already served number of connections of each server. The Throttled method consists of an index table of available virtual machines and their states. Requests are allocated to the virtual machine that is available and has sufficient resources. However, those methods are designed for the single objective of load balancing and cannot address the problem of reducing idleness without disconnecting user connections.

There are also some intelligent load balancers. Meta-heuristics based load balancers [9] are general derived from the behavior of natural evolution and iteratively search the optimal solution in the policy space. Unfortunately, those methods can hardly be applied to the underlying online real-time routing scenarios, as their iterative optimization process usually takes a relatively long time to find an optimal routing solution. RL-based load balancers are also warmly studied for the purpose of real-time routing. The witnessed works mainly focus on the scenarios of Software Defined Networks [10] and Internet of Things networks [11]. However, as the elasticity is not the major concern of those networks, those works do not encounter the same difficulty as in the financial scenarios of keeping connections alive while reducing idleness.

For cloud computing scenarios, the general load balance problems have been more intensively studied in the context of job shop scheduling [12–14] in the scheduler module in the backend. Honestly, the job scheduling problems and the connection routing problems generally follow the same vector bin-packing mathematical model [15, 16], where a set of vector items are expected to be distributed among multiple bins so that their loads can be balanced. Unfortunately, we discuss as follows that those job scheduling methods cannot readily address the underlying problem of this work. The two major differences between them emerge from practice.

(1) A computing job is basically a sequence of predefined computing steps where any intermediate data can be easily saved as checkpoints with established techniques. In this regard, killing a job on one server and restarting it on another server can lose no data with extra techniques like Live Migration [17]. However, a user connection does not know what data will be delivered through it at a specific time, which means a disconnection will lose data that can hardly be recovered. In a nutshell, those job scheduling methods do not explicitly keep connections alive while reducing idleness.

(2) Though some online job scheduling works also consider objectives similar to reducing idleness, like minimizing slowdown [18], makespan [19] or job completion time [20], they need the running duration of the jobs explicitly in advance. While in the connection routing problem, the online duration of any user connection is unknown when the request comes up.

To summarize, to our best knowledge, existing related works are not dedicated to solving the underlying bi-objective problem, i.e., load balancing and reducing idleness without actively disconnecting users. Thus, directly applying them may not be satisfactory.

3 Problem description

In typical financial data services, e.g., the stock market data services, the user connections usually follow a quite regular mode. First, the number of user connections increases very rapidly at the market opening in the morning, and then decreases gradually when the o'clock is approaching to the market closing in the afternoon. Second, the duration of different user connections may vary significantly. Consequently, as shown in Fig.1, the users usually disconnect from the data services at different times, leaving the servers running with a very low utilization rate but cannot be shutdown due to few alive connections on them, e.g., the duration from t_2 to t_3. This economic waste resulting from the server idleness can be significantly scaled up as the financial data services usually run on cloud computing infrastructures with large-scale servers.

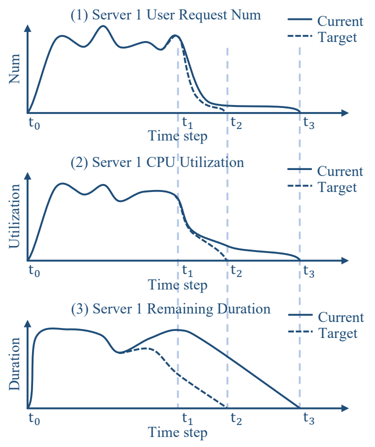


Figure 1 A typical phenomenon of financial services that there exist very few connections from t_2 to t_3, which are targeted to be reduced in this work.

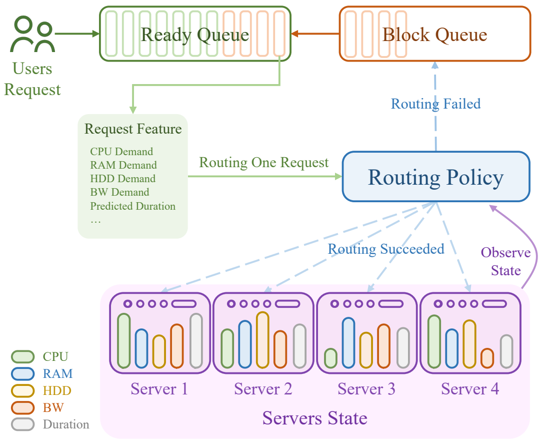


Figure 2 The routing process of load balancer in financial services.

Fortunately, the duration of each user connection in different days often varies slightly as most users are staff from financial industries who often have regular schedules [21]. This simply suggests that the disconnection time of the same user on different days can be predictive to some extent and maybe statistically modeled following a probabilistic distribution. With this special feature in the financial services, this work describes the problem of routing user connections to reduce idleness as follows.

As shown in Fig.2, each incoming user request has five types of features, including the demand of CPU, memory (RAM), storage (HDD), bandwidth (BW) and the historical distribution of the connection duration of this user. The first four features are hardware requirements indicating how many resources the system needs to serve that connection and its related data services. The last feature emerges from the financial services and is, for the first time, considered in the load balancer. Each time a new request arrives, it first enters the ready queue and waits to be routed to one of the available servers by the routing policy. Once the user request is successfully routed to a server, a user connection between the user and the server can be constructed, and data services can be provided to the user through the connection. Otherwise, those requests that cannot be routed due to insufficient hardware resources will be tentatively placed in the block queue until the next round of routing. The routing policy works by optimally distributing the users' requests to minimize the load imbalance of 4 types of resource utilization among servers while simultaneously reducing the idle time of the servers.

In this paper, the above mentioned two objective functions are defined as follows. Suppose the load balancer continues serving for a period of timesteps. For the objective of load balancing, it is defined as the standard deviation of the utilization among 4 resource types:In this paper, the above mentioned two objective functions are defined as follows. Suppose the load balancer continues serving for a period of T timesteps. For the objective of load balancing, it is defined as the standard deviation of the utilization among 4 resource types:

$$F_{\text{balance}} = \frac{1}{4 \cdot T} \sum_{t=0}^{T} \sum_{r=1}^{4} \sqrt{\frac{\sum_{i=1}^{N} (x_{ri}^{t} - \mu_r^t)^2}{N}}, \tag{1}$$

where x_{ri}^t indicates the utilization of r-th resource type of the i-th server at the t-th timestep, and μ_r^t denotes the average of the r-th type resource utilization among all N servers at the t-th timestep. For the objective of reducing idle time, instead of using the commonly-seen makespan of all the servers, we calculate the average remaining duration of the servers over all timesteps. The reason is that we want to achieve idle reduction at any time, not just at the last moment.

$$F_{\text{idle}} = \frac{1}{T \cdot N} \sum_{t=0}^{T} \sum_{i=1}^{N} d_i^t, \tag{2}$$

where d_i^t is maximal remaining duration among all connections of the i-th server at the t-th timestep. Note that, the remaining duration of any connection cannot be known exactly before its disconnection. On the other hand, as discussed previously, according to the observation that each user's connection duration on different official days changes slightly due to the regular schedule, we can predict the duration of each user at the current t-th timestep as the average of the user's historical duration. This paper also empirically assesses the different impacts on the load balancer caused by different degree of predictability. Also note that, in the whole problem description, the system is not allowed to actively disconnect the user connections. Only passive disconnection from the user's application is permitted.

4 Method

In this section, the RL-based problem formulation is first described, including the representation of actions, the RL states, and the rewards. Then, a neural network is designed to learn the non-linear relationship between the statistically estimated connection duration and the available servers. The architecture of this neural network is also designed for elastic scenarios with changing numbers of servers. After that, an evolutionary multi-objective algorithms based training framework is introduced for this scalable policy, in terms of both minimal load imbalance and server idleness. The action mask operator for helping stably train the policy is detailed lastly.

4.1 Reinforcement learning based formulation

In this RL model, each of the user's requests in the ready queue is represented together with the utilization of all servers as the RL states. Based on the states, the policy outputs the action of routing the request to one of the servers to construct a user connection.

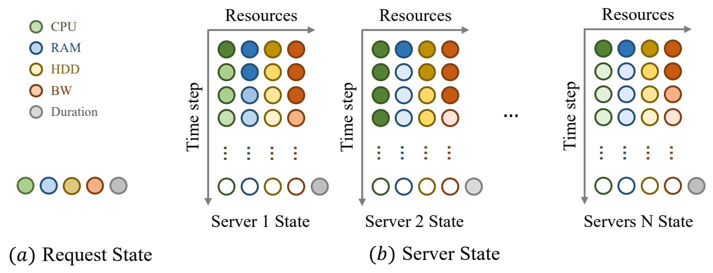


Figure 3 This work considers two types of states for RL: (a) the 5-dimensional states of each user request, (b) the 41N-dimensional server states include N servers with 4 types of resources looking forward 10 timesteps into the future, and 1 maximal remaining duration.

As shown in Fig.3, the RL states are a conjunction of both the states of one user's request and the states of N servers. The state of each request is represented as a 5-dimensional vector, including the demands of 4 types of hardware resources and 1 scalar value of the predicted duration, i.e., the average of the historical duration of that user. The state of each server is represented as a $(4h+1)$-dimensional vector, i.e., the utilization of 4 types of resources along h looking forward timesteps and 1 scalar value of the maximal remaining duration among all connections in the future h-th timestep. Here the remaining duration of each connection is calculated as its predicted duration minus h timesteps, and the utilization of each resource in some future timestep is calculated as the summation overof all alive connections at that moment. By using this feature along h looking forward timesteps, the future states of the servers are involved to help the policy learn the optimal distribution of the requests for reducing idleness in the future. In the empirical studies, we simply set h to 10. As a result, the total length of the RL states vector is $5 + 41N$.

The RL model has $N+1$ actions, i.e., routing the request to one of the N servers or the blocking queue if no proper server is available. For the reward function, as we employ the evolutionary multi-objective algorithm to train the policy, it is unnecessary to derive the intermediate rewards of each routing action. Instead, the two objectives in Eqs.(1)-(2) are used as the accumulative rewards for each policy after the whole simulation process [22].

4.2 The scalable policy

To learn the optimal distribution of connections over servers, the neural network based policy is preferred due to its strong learning ability. The input of the neural network will be the above described $(5+41N)$-dimensional RL states, and the output of the neural network will be the $(N+1)$-dimensional RL actions. Therefore, the sizes of the input and output of the policy change with the number of available servers. Additionally, due to the elasticity of cloud computing, the number of available servers does usually change over time for a given data service. Besides, it is also important to generalize the trained policy to different data services that may be supported by different numbers of servers. For these purposes, the employed policy should be able to scale with the number of servers automatically. To this end, this paper proposes a parameter-sharing based neural network architecture for the underlying routing tasks.

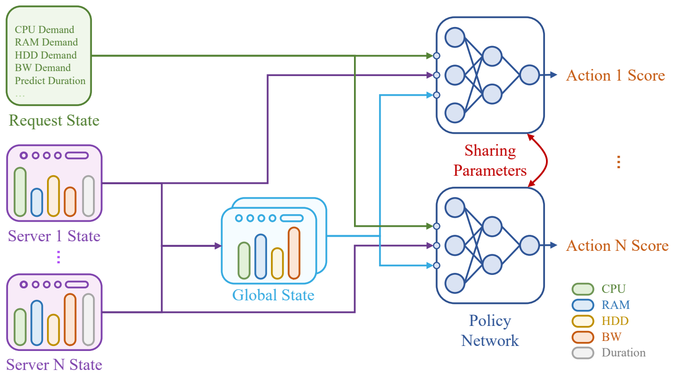


Figure 4 Inference process of the scalable policy network.

As shown in Fig.4, the policy network achieves the scalability with the global state management and the sharing parameters among N copies, each of which corresponds to one available server. The input of this policy network is divided into three components: the request state, the individual server state, and the global state. The global state refers to the mean and the standard deviation among all individual server states, either of which is basically a 40-dimensional vector with 4 resources looking forward for 10 timesteps and 1 remaining duration. There are N fully connected networks jointly for the decision making. All networks share the same weight parameters and can receive the shared global state of servers as well as the 5-dimensional request state. In addition, each i-th network will individually receive the 41-dimensional states of the i-th server. For the decision making, each i-th network outputs a score. A higher score indicates better load balance and predicted idle time reduction. At last, the action with the highest score will be finally selected and the corresponding server will be chosen to construct the connection. By this architecture, the whole policy is able to scale with the number of servers and can intuitively be generalized to different data services.

4.3 Evolutionary multi-objective training framework

As discussed above, the underlying load balancer has two objectives (see Eqs.(1)-(2)), i.e., the long term rewards. In traditional RL policy training, it is common to aggregate these two rewards into a single one by weights sum. Unfortunately, due to the different scales of those two objectives, it is quite difficult to set proper weights, and the two objectives might not easily be satisfied.

In this section, we propose to employ the evolutionary multi-objective algorithms [23] to optimize the weights of the policy directly for three reasons. First, evolutionary multi-objective algorithms have been the mainstream methods for multi-objective optimization problems and have successfully shown their power in many real-world applications [24]. Second, it does not need to set the aggregation weights manually but directly compare solutions with the domination relation. Third, it suffices to output a set of optimal solutions called non-dominated Pareto set, which is equivalent to the optima with different weights sum aggregation [25], thus offering the users the flexibility to select the policy online. Here we employ the well-established NSGA-II [26], i.e., almost the most successful evolutionary multi-objective algorithm developed by Deb et al. in 2002, to construct the evolutionary multi-objective training framework for the policy. For more details on NSGA-II, please refer to [26].

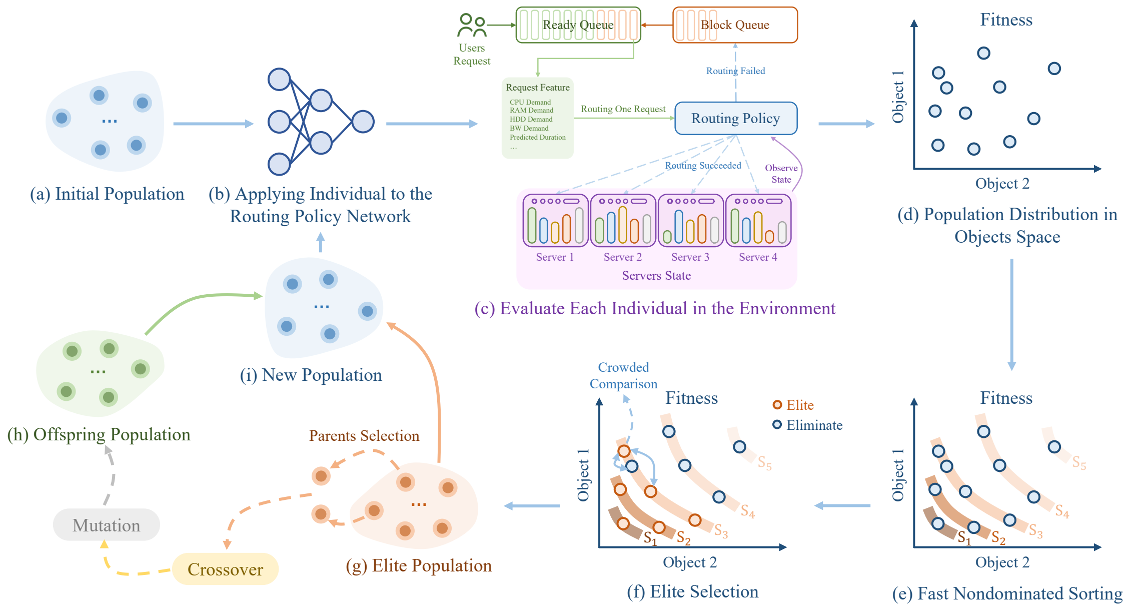


Figure 5 Overview of the Evolutionary Multi-objective Training Framework.

The proposed NSGA-II based training framework shown in Fig.5 is an iterative process. At the beginning of the training (step (a)), a population of the routing policy network is randomly initialized. Second, we evaluate each individual policy in the simulation environment (steps (b)-(c)). Here, the objective functions in Eqs.(1)-(2) are directly used for evaluation, as evolutionary algorithms usually measure the accumulated performance of a candidate policy over the whole simulation period [27]. It is important to note that the settings of the simulation environment, e.g., the requests sequence and the servers' initial states, are identical for the evaluation of all individual policies. After the evaluation, we obtain the bi-objective fitness values for all individuals in the current population, which distributes in a 2-dimensional objective space (step (d)). In this space, the closer to the origin, the better the individual will be.

Then, the individuals in the current population are sorted by the fast non-dominated sorting of NSGA-II, and L non-dominated solution sets $S_1, S_2, \dots, S_L$ are obtained (step (e)). Here, a non-dominated solution set means that an individual within the solution set cannot dominate any other individuals within the same solution set, but is dominated by the sets prior to it, e.g., S_1 dominates S_2, $\dots$, S_L. More specifically, an individual A dominates an individual B means that A is not inferior to B in all objectives and is superior to B in at least one objective. The elite individuals are selected from the current population based on the precedence of the non-dominated solution sets and the crowding comparison (step (f)), where the crowding comparison algorithm is used to avoid selecting similar individuals and thus ensure the diversity of the population. Finally, based on the elite population, the offspring population is generated with typical operators of evolutionary algorithms (step (h)) and forms. It forms the new population for the next iteration together with the elite population (steps (g)-(i)). And then the iteration continues until some stop-criteria is met, and the individuals in S_1 are output as the trained policies.

In the above framework, each policy is represented as a vector of weight parameters of the network described in section 4.2. The offspring generation step is to generate two new vectors based on two parent vectors. As shown in Fig.6, the two parent vectors crossover at a uniformly randomly chosen point to produce two tentative vectors. Then, based on the two tentative vectors, two offspring vectors are generated by the commonly used Gaussian mutation operator. Specifically, for each i-th weight parameter of the policy network, with a probability of γ, it will be mutated by sampling from the

Gaussian distribution $N(\theta_i, \beta)$, where θ_i denotes the i-th weight parameter of a tentative vector, and β denotes the magnitude of the mutation.

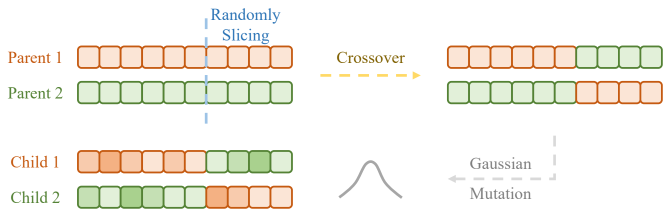


Figure 6 The process of generating offspring.

4.4 Actions mask operator

Note that, while simulating the individual policies within the step (c) of Fig.5, the policies will not be updated unless the simulation ends. This might be a major difference between evolutionary RL and traditional policy gradient based RL. Though it has been revealed that this evolutionary RL training paradigm has merits for those scarce reward problems [28], it indeed causes an issue in the load balancer. That is, the policy may be trapped in the situation of continuously routing the incoming requests to the same server, making that server fully utilized quickly. As a result, the subsequent requests cannot be connected to that server and thus have to be put into the block queue. Once the block queue overflows, the simulation will be stopped, and the evaluated objective fitness of that policy will be very poor.

As the initial policies are randomly generated, they are highly likely to face this issue. And the training may easily fail. To stabilize the training process, a simple action mask operator is proposed. That is, after N sharing networks in a policy outputting their scores, those servers whothat cannot satisfy the hardware demand of the current request will be mask and the score of the corresponding sharing networks will be set to 0. After that, the server with the highest score will be selected. By doing so, the above issue has been addressed and the action mask operator can be viewed as a specific kind of diversity enhancement.

5 Experiments

The experimental studies are designed to answer the following research questions.

(1) How does MERL-LB perform on the task of reducing idleness without disconnecting user connections, in comparisons with traditional methods?

(2) How does the proposed evolutionary multi-objective framework facilitate the training of RL-based routing policy, in terms of the convergence rate and the diversified user options?

(3) How is the stability of MERL-LB against different loads?

(4) How is the scalability of the proposed parameter-sharing routing policy against different numbers of user requests and servers?

(5) How does MERL-LB tolerate to the different variances of the user connection time?

5.1 Simulation environment

The experiment simulates a commonly encountered scenario of financial data services. The simulation starts at the opening of the stock market and ends after all users disconnect from the servers naturally, which will be later than the close of the stock market. For the major experiments, we simulate around 1500 user requests coming up gradually to 10 servers. The coming up duration simulates between the opening and close time of the stock market, which is set to 2 hours. Later, for the stability and scalability tests, the number of user requests varied up to 7500, and the number of available servers increases from 10 to 50.

All simulations are conducted on a Discrete Event Simulation based virtual data center implemented with Python[1)]. The configurations of the virtual data center are shown in Table 1. The number of servers in the data center is set to 10; Each server has four resources (CPU, RAM, HDD, and BW), and the capacity of each resource is set to 500 units; At the beginning of the simulation, all the resource utilization

1) https://github.com/zlaom/MERL-LB

is initialized to zero. The queue sizes of the ready queue and block queue in the data center are fixed to 200 connections, respectively. The event of the simulation process is triggered five times every minute. That is, the minimum simulation interval is set to 12 seconds. The predicted range for the future states of the data center is 120 minutes, among which $h = 10$ timestamps are sampled to represent the prediction of the system's future state.

Table 1 Configurations of the simulated virtual data center.

Parameters	Value
time_step	12 (seconds)
server_num	10
cpu_capacity	500 (units)
ram_capacity	500 (units)
hdd_capacity	500 (units)
bw_capacity	500 (units)
ready_queue_size	200
block_queue_size	200
predicted range	120 (minutes)
future_sample	10

Table 2 Simulation data generation parameter setting.

Parameters	Value
data_time	120 (minutes)
time_step	12 (seconds)
mean_req_num	3
min_res_req	0 (units)
max_res_req	10 (units)
min_user_duration	1 (minutes)
max_user_duration	120 (minutes)

5.2 Compared algorithms

Four classic rule-based heuristic routing methods and two RL based algorithms were used for comparisons. As discussed in section 2, three industrial algorithms, like Random routing, Round Robin, and the Least Connection are selected as they merely consider load balancing. Another algorithm named Least Duration Gap is designed based on Least Connection to only consider the reduction of idleness, ignoring the load balancing. The core idea of the Least Duration Gap is to greedily route the user requests with similar predicted remaining duration to the same server with the following rule:

$$a = \arg\min_{i} \left(\left| d_s^i - d_r \right| \right), \tag{3}$$

whered_s^i indicates the maximum predicted remaining duration of the i-th server, while d_r is the predicted duration of the request.

For the two RL based methods, they are designed to show the effectiveness of the proposed evolutionary multi-objective training framework. For this purpose, the Proximal Policy Optimization (PPO) [29] and the Independent-input Policy Gradient (IPG) [30] are employed to train the proposed parameter-sharing policy. The former is a well-established RL method that has been successfully applied in many fields. The latter is a variant of the original IPG method for job scheduling problems that are similar to the traditional load balancing problem. Both of them follow the policy gradient training framework. PPO calculates the gradient based on the temporal-difference error by an incomplete sequence of interactions, while IPG calculates the gradient with a complete sequence of interactions. The immediate rewards of a routing action at the specific t-th timestep can be simply derived from Eqs.(1)-(2) by eliminating the summation over t, as shown in Eqs.(4)-(5). And the two rewards are linearly aggregated with fixed weights to form the training signal, as shown in Eqs.(6).

$$\text{reward}_{\text{balance}}^t = \frac{1}{4} \sum_{r=1}^{4} \sqrt{\frac{\sum_{i=1}^{N} \left(x_{ri}^t - \mu_r^t \right)^2}{N}}, \tag{4}$$

$$\text{reward}_{\text{idle}}^t = \frac{1}{N} \sum_{i=1}^{N} d_i^t, \tag{5}$$

$$\text{reward}^t = w_i \cdot \text{reward}_{\text{balance}}^t + w_2 \cdot \text{reward}_{\text{idle}}^t. \tag{6}$$

5.3 Implementation details

The architecture of the parameter-sharing neural network is introduced as follows. The network is basically a multi-layer perceptron with a 126-dimensional input layers, a 32-dimensional hidden layers, and a 1-dimensional output layer. The Relu [31] is used as the activation function of the hidden layer. The 126-dimensional input refers to the 5-dimensional resource state of each incoming request, the 41-dimensional individual server state of the corresponding server, and the 80-dimensional global state with half for the average and half for the standard deviation of all individual servers on the 4 types of hardware indicators along 10 steps forward. The 1-dimensional output is the score of routing the incoming request to the corresponding server. The network with the highest output score among all N networks is found, and the corresponding server is chosen to be routed. All three RL-based methods share this network through the experiments.

The major hyperparameters of the PPO-LB and IPG-LB are described as follows. For the major simulation settings, both PPO-LB and IPG-LB share the same parameters. The maximum number of training simulations is set to 750000 for both algorithms, which is kept the same for MERL-LB. The learning rate of the network is set to 0.001, the discount factor γ is 0.99, and the aggregation weights of the load balancing reward and the idle time reward are 0.5, respectively, for both algorithms. The network of PPO-LB is updated every 256 steps of interaction. Each interaction sequence is reused for 5 epochs. The clipping threshold ϵ of PPO-LB is set to 0.2. The weight α_1 of the loss of the value network in the loss function is 0.5, and the weight α_2 of the entropy term coefficient is 0.01. The batch size during training is set to 512 for IPG-LB, and each data instance randomly samples 10 interaction sequences using the same policy. In general, those parameters are all suggested by the original paper of PPO [29] and IPG [30]. The 4 rule-based heuristics also do not involve hyperparameters.

For the major hyperparameters of the proposed MERL-LB, we set the population size to 50, the number of elites selected from the population per iteration to 25, and the number of offspring generated by elites to 25. The probability of mutation on each variable is 0.25, and the magnitude of mutation is 0.05. The time budget of the simulation is the same as PPO-LB and IPG-LB for fairness.

Table 3 Performance of different algorithms for two objectives on a validation set of 10 servers with a 75% load.

Methods	$F_{\text{balance}} \downarrow$	$F_{\text{idle}} \downarrow$
Random	30.15±2.55	106.26±0.83
Round Robin	22.15±1.96	107.12±0.85
Least Connection	**16.27**±1.04	106.98±0.95
Least Duration Gap	66.89±6.12	**69.91**±1.56
IPG-LB	17.05±1.12	**79.75**±1.41
PPO-LB	**16.23**±1.14	80.67±1.34
MERL-LB-1	**15.70**±1.17	88.68±1.56
MERL-LB 4	16.88±1.06	76.75±1.25
MERL-LB-6	23.42±1.50	72.44±1.26
MERL-LB-11	66.58±4.93	61.63±1.26
MERL-LB-25	178.77±4.88	**46.48**±1.73

5.4 Overall comparisons

Table 3 reports the overall comparison results between MERL-LB and the compared algorithms, and is divided into three parts. The top part shows the performance of the heuristic-based algorithms, the middle part is about the two policy gradient based reinforcement learning load balancers , and the bottom part shows representative solutions generated by the proposed MERL-LB algorithm, where the suffix indicates different individuals in the last population.

First of all, the three RL-based methods generally perform better than the heuristic-based methods, especially on the load balancing objective. Furthermore, MERL-LB has been able to find the best solution for either objective and also offers different options for various balances between the two objectives.

It can be seen that MERL-LB-1 focuses more on the load balancing objective and performs the best among all algorithms. Compared with the Least Connection method, which merely targets such an objective and performs the best among the compared heuristics, MERL-LB-1 reduces the idle time by 17.11% while achieving better load balancing performance. MERL-LB-25 focuses more on the idle time objective and can reduce up to 33.51% over the Least Duration Gap method, who is specifically designed for reducing idleness.

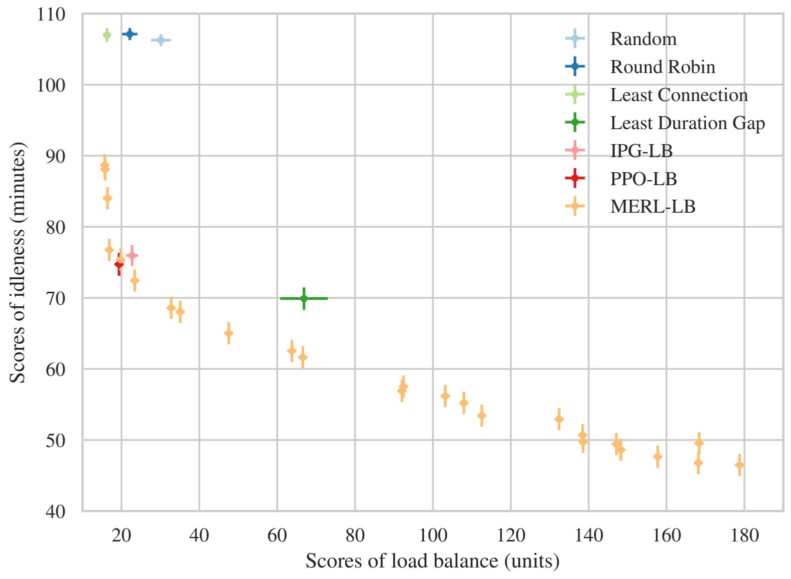


Figure 7 Performance of different algorithms on two objectives. The cross-like error bars indicate the standard deviation on both objectives.

The other 3 solutions of MERL-LB successfully show how evolutionary multi-objective optimization can offer users different options. MERL-LB-4 not only achieves a better load balancing objective than IPG-LB but reduces the idle time by 3.76%. MERL-LB-6 achieves similar idleness to the Least Duration Gap method and can improve the load balancing objective by 64.99%. MERL-LB-11 performs similarly to the Least Duration Gap in terms of the load balance while successfully reducing the idleness by 11.87%. To summarize, the Pareto optima output by MERL-LB can effectively balance the two objectives automatically.

Fig.7 shows the distribution of the optimal policies generated by different algorithms in the objective space. The figure clearly shows that compared with PPO-LB and IPG-LB, MERL-LB can generate a set of diverse policies that are non-dominated to each other on both objectives. That means any one of the policies is a good choice for a specific need. By looking at the error bars of the policies in both directions (i.e., both objectives) in the figure shows that the policies generated by MERL-LB are mostly stable, especially compared with the Least Duration Gap method.

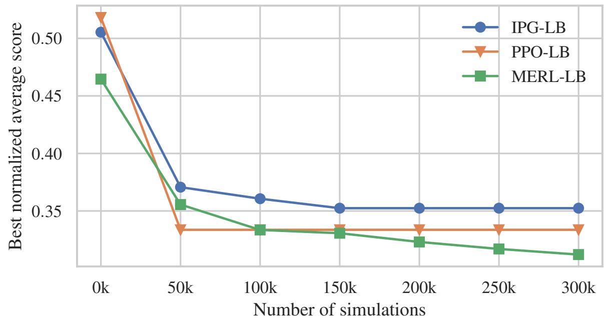


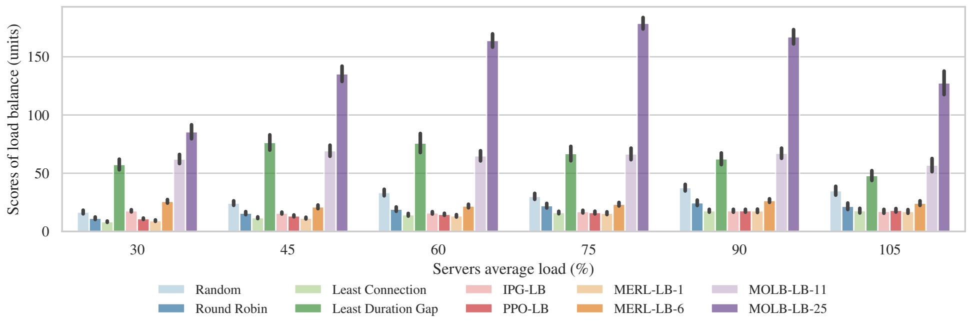
Figure 8 The convergence curves of RL-based load balancers on two objectives in the training process.

To further demonstrate the effectiveness of the proposed evolutionary multi-objective training framework, Fig.8 depicts how the three RL-based methods converge during the first 300000 simulations in the training process. Each curve is composed of the scores over the training simulation. For each score, it is an average of two min-max normalized objective values. In short, all three methods have converged, and MERL-LB achieves the fastest convergence rate.

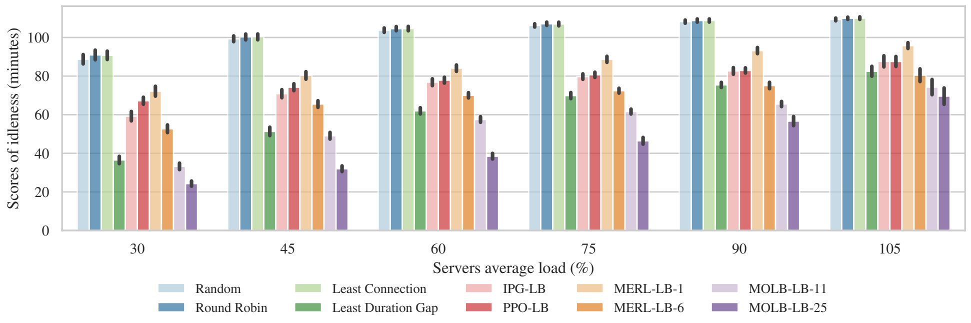
5.5 Impacts on different numbers of user requests

Fig.9 shows how different numbers of user requests may impact the performance of the algorithms on the two objectives. Through this group of experiments, the number of available servers is fixed to 10. The number of user requests is simulated from 600 to 2100 by increasing the mean value of the employed Poisson distribution. With a fixed 2 hours of the simulated opening of the market, not only does the total number of user requests increase but the concurrently incoming user requests at each time interval are enlarged, which leads to the increased loads of servers. As a result, this group of experiments assesses the algorithms for different numbers of user requests and different loads of servers in the same way.

For the load balancing objective, the performance of Random, Round Robin, the Least Connection, PPO-LB, and MERL-LB-6 gradually deteriorate as the load increases. This is mainly because higher resource utilization rates may lead to a greater load imbalance among servers, as the standard deviation among servers can be larger. With the increase of loads, the load balancing performance of LDG, MERL-LB-11 and MERL-LB-25 all first deteriorated and then gradually improved. This is mainly because those algorithms all pay more attention to reducing idleness. When the load is at a low level, it leads to a greater imbalance among servers. However, when the load increases, even servers with lower loads will be assigned more requests, which will gradually reduce the differences in resource utilization rates among servers.



(a) The impacts of different loads on the load balance objective.



(b) The impacts of different loads on the idleness objective.

Figure 9 Performance of algorithms on two objectives under different loads, i.e., different numbers of user requests.

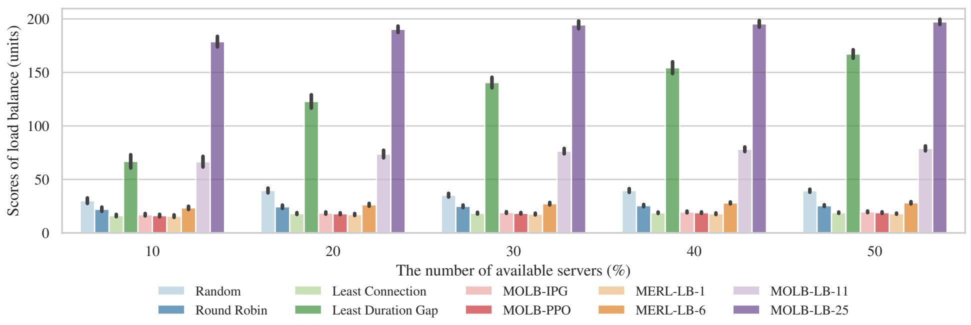
For the idleness objective, as the load increases, the idle time of all algorithms increases accordingly. This is mainly because an increase in the load means an increase in the number of long connection requests, which would need more servers to serve the connections, while the number of servers is fixed at 10.

In general, MERL-LB-1 performs similarly to the Least Connection method at all loads and is able to reduce the idle time by 10-20%. MERL-LB-6 achieves a similar idleness objective value to LDG at 75%-

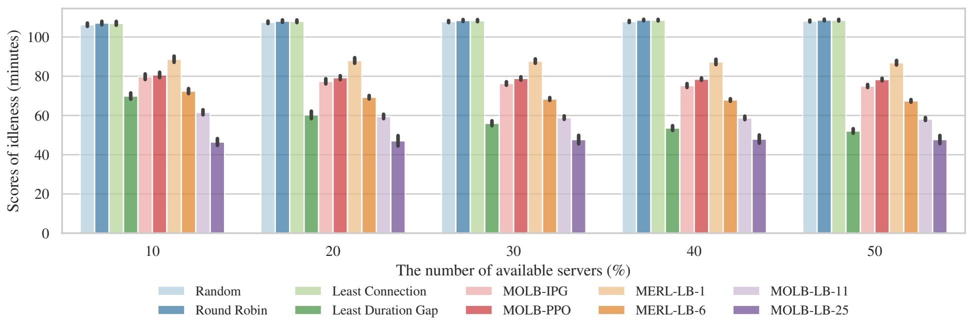
105% loads while it improves the load balancing objective by about 50%-60%. Compared to PPO-LB and IPG-LB, MERL-LB can generate diverse policies and consistently outperform the two at different loads. Therefore, MERL-LB performs more robustly overall when the number of requests or loads varies.

5.6 Impacts on different numbers of available servers

Fig.10 shows the performance of the algorithms on two objectives under different numbers of servers, while the average load is fixed at 75%. Thus, by varying the number of servers from 10 to 50, the number of user requests also changes proportionally, up to around 7500 requests in total for the case of 50 servers. For the load balancing objective, it can be observed that as the number of servers increases, the load balance of each algorithm does not deteriorate too much, except the Least Duration Gap method, who is the only algorithm that never considers the load balance objective. Similarly, for the idleness objective, the performance of the Least Duration Gap method improves most significantly as the number of servers increased, though it still cannot outperform MERL-LB-25 in all tested cases. The other algorithms perform rather stably, regardless of the changed number of servers. Overall, this suggests that the proposed scalable policy network can be flexibly applied to scenarios with different numbers of servers and can maintain stable routing capabilities on the two objectives.



(a) The impacts of different numbers of user requests on the load balance objective.



(b) The impacts of different numbers of user requests on the idleness objective.

Figure 10 Performance of algorithms on two objectives with different numbers of servers.

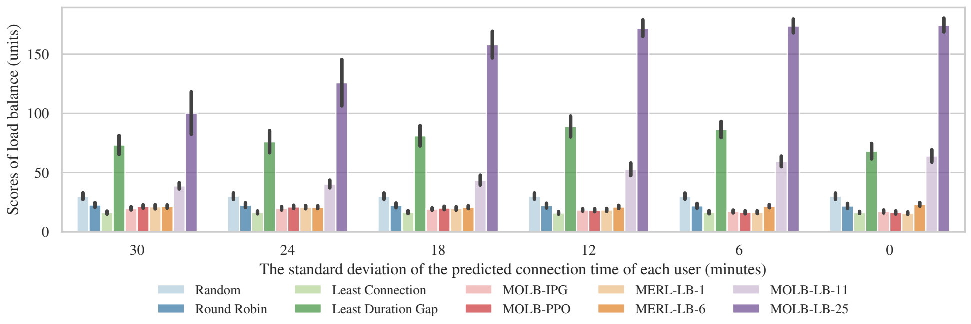
5.7 Impacts on the predictability of user duration

This work assumes that we can predict the duration of a user by statistically analyzing their connection time over a recent period (for example, the average value of the user's recent connection duration). In real cases, the actual duration of users may vary over time, and is highly unlikely identical to the predicted value. Thus, this group of experiments aims to assess how different degrees of the predictability may impact the performance of the algorithms, reflecting how effective MERL-LB can behave in real-world noisy environment.

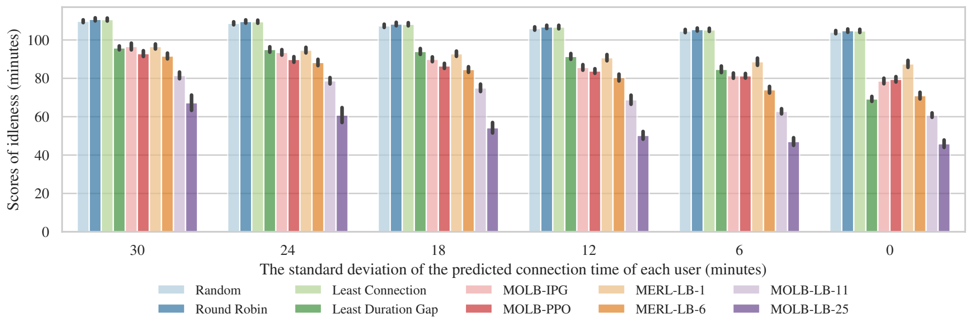
For this purpose, we generate new testing data sequences based on the original 50 testing instances by adding a random noise to each user's connection time. More specifically, the random noise follows a Gaussian distribution $N(\mu, \sigma)$, where μ is the duration of the original data, and σ represents the standard deviation of noise, resulting in different predictability of the user connection time. The random noise is truncated at a 3σ level.

Fig.11 shows the impacts of different user duration predictability on the algorithms for two objectives respectively. In general, for the three heuristics that do not consider the idleness objective, their performance does not improve when the predictability increases. For the Least Duration Gap method, which focuses on the idleness objective but does not rely on the prediction of user connections, its performance on the idleness objective improves near 20 minutes. For the other RL-based policies that share the same scalable routing network, the prediction of the user connections has been involved in their training processes. As can be seen, the various predictability does not deteriorate their performance unexpectedly. That is, as the predictability increases, their performance on the idleness objective becomes better similar to the Least Duration Gap method. This shows that with up to a standard deviation of 30 minutes error on the prediction of the user connection time (maximally 120 minutes), the proposed scalable routing network works quite stable.

On the other hand, the policies that tend to focus more on the idleness objective deteriorate on the load balance objective as the predictability increases. The reason is that as the predictability of the user connection increases, the randomness of the data as well as the routing decreases. And the randomness usually forms a source of load balancing, as suggested by the Random method.



(a) The impacts of different user predictability on the load balance objective.



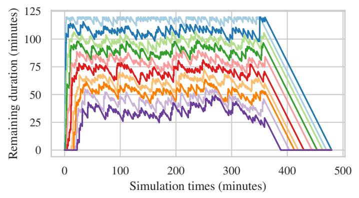
(b) The impacts of different user predictability on the idleness objective.

Figure 11 Performance of algorithms on two objectives under different user predictability.

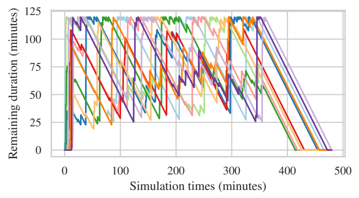
5.8 Understanding the trade-off between two objectives

To understand how the load balancing objective and the idleness objective are balanced during the learning of MERL-LB, we show how the two objectives change over time for the Least Duration Gap method, the Least Connection method, and MERL-LB-6. For the Least Duration Gap method, only the idleness objective is considered, while for the Least Connection method, only the load balancing objective is considered. MERL-LB-6 is selected here as it balances the two objectives well. We depict the curves of the idle time and the CPU resource utilization in Fig.12. For clarity, the simulated opening period is enlarged to a full range of 6 hours to make the observation easier. As shown in Fig.12 (a), (b), (c), each curve in the figure represents the change of the remaining duration of each server per minute (known from the ground truth of the testing data). In Fig.12 (d), (e), (f), each curve in the figure represents the change of CPU resource utilization of each server per minute.

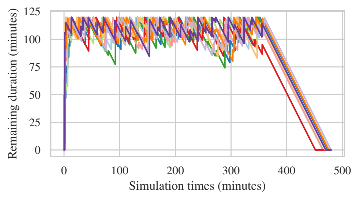
During the opening period, the number of users continues to come up, so the CPU utilization of the 10 servers rises up and fluctuates, and becomes stable after the number of concurrent alive users is stabilized.



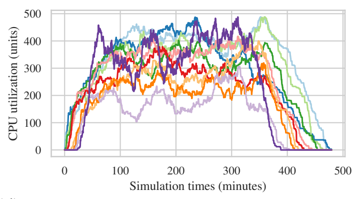
(a) The remaining duration of each server in Least Duration Gap.



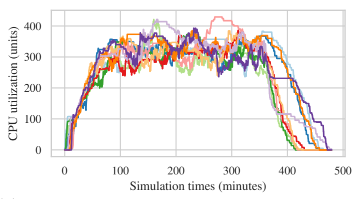
(b) The remaining duration of each server in MERL-LB-6.



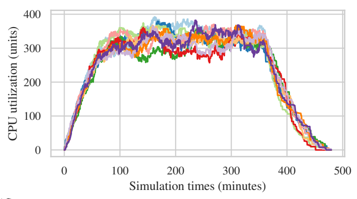
(c) The remaining duration of each server in Least Connection.



(d) The CPU utilization of each server in Least Duration Gap.



(e) The CPU utilization of each server in MERL-LB-6.



(f) The CPU utilization of each server in Least Connection.

Figure 12 Comparison of Least Duration Gap, MERL-LB-6, and Least Connection characteristics on 10 servers.

Then, after the close of the market, no new user request comes up, and the number of connections as well as the CPU utilization, continues to decrease. The closer the utilization rate among servers per minute, the better the load balance is. In Fig.12 (e), each server's CPU resource utilization of MERL-LB-6 fluctuates about half of that in Fig.12 (d), so MERL-LB-6 has better load balancing performance than the Least Duration Gap. In Fig.12 (f), the Least Connection's CPU utilization fluctuates even smaller than MERL-LB-6, so its load balancing performance is better correspondingly.

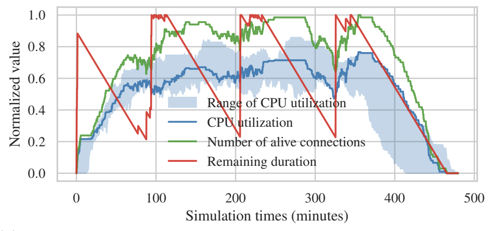
An interesting phenomenon can be observed by comparing Fig.12 (a), (b), (c) that the Least Duration Gap and MERL-LB-6 minimize the idleness very differently. In Fig.12 (a), the curve of the remaining duration of each server is completely separated and even shows a layered distribution. In this regard, the Least Duration Gap actually minimizes the idleness by clustering each incoming user request with all the alive connections based on their predicted remaining durations. On the other hand, as shown in Fig.12 (b), the curves of MERL-LB-6 overlap with each other and show a sawtooth pattern. This pattern may trade off the load balance better and contribute to better idleness as all curves drop to 0 in the shortest expectation time. In either above-mentioned case, the spectrum of the curves is much larger than that of the Least Connection, indicating that both the Least Duration Gap and MERL-LB-6 have a better trade-off on both objectives.

5.9 Understanding the sawtooth pattern of MERL-LB

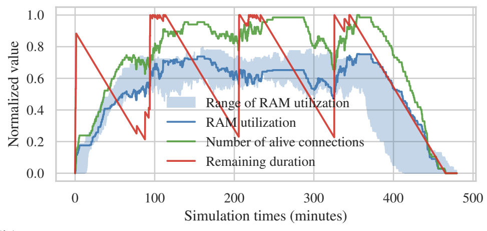
To further understand the above sawtooth pattern of MERL-LB, the utilization rates of 4 types resources of MERL-LB-6 over time are visualized in Fig.13. In each figure, 4 curves are depicted, i.e., the corresponding resource utilization of a server, the range of the corresponding resource utilization among all servers, the number of alive connections, and the ground truth remaining duration of this server.

It can be observed that the number of alive connections generally increases though keeps fluctuating. This indicates that new user requests are routed to the server continuously and some existing connections become disconnected coincidently. With the increase of the number of connections, the remaining duration does not go up accordingly. In fact, its sawtooth pattern suggests that the requests routed to this server are mostly with remaining durations smaller than that of this server. Thus, the remaining duration of this server gradually drops as time goes by.

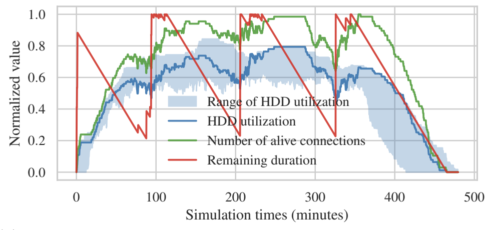
If we look at the place where the remaining duration of the server goes up significantly, it is usually found that the utilization of one resource type drops to the bottom of the corresponding range. If the utilization curve goes below its range, it means that the load balance in this resource type is enlarged. And it is necessary to route a request with a larger remaining duration to this server so that the resource utilization can be enlarged for a long time. On the other hand, once the utilization of one resource type goes to the top of the range, the number of alive connections will not increase. This is the opposite strategy learned by the policy to minimize the load balance by forcing the resource utilization not to exceed the range.



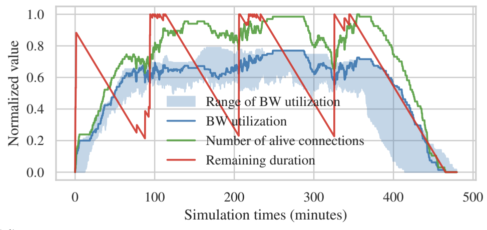
(a) The ground truth remaining duration changes with CPU utilization over time.



(b) The ground truth remaining duration changes with RAM utilization over time.



(c) The ground truth remaining duration changes with HDD utilization over time.



(d) The ground truth remaining duration changes with BW utilization over time.

Figure 13 Analysis of the sawtooth pattern of MERL-LB-6 on a server.

6 Conclusions

This paper first introduced a new issue of reducing idleness in the financial servers where the user connections were only allowed to be naturally disconnected by users. This paper targeted the issue as a constrained online routing problem, and identified the limitations of existing load balancers. as not considering user connection duration information. To address this issue, this paper proposes to consider the user connection duration information and model the problem as a bi-objective reinforcement learning problem, i.e., the load balancing objective and the idleness objective. With this modeling, a parameter-sharing routing policy is designed for varied numbers of servers, and an evolutionary multi-objective training framework is proposed based on NSGA-II to train the policy.

Extensive experimental studies have been conducted to reveal the detailed advantages of the proposed method. It is found that the proposed method generally beat the compared algorithms, including the traditional heuristics and advanced RL-based ones. The evolutionary multi-objective training framework not only facilitates policy training with a much faster convergence rate but also offer a set of diverse non-dominated policies for users' options. To solidate the experiments, we tested the algorithms with different numbers of user requests and different numbers of servers. It was observed that the proposed method performed quite stable due to the proposed scalable routing policy. At last, the proposed method generated a sawtooth pattern of decision-making strategy, which has been analyzed to be beneficial to balancing the two objectives.

In the near future, we are excited to apply this policy to real-world financial cloud systems. In this regard, it is important to study more robust evolutionary training method for MERL-LB in realistic noisy environment [32] with the collection of real user behavior data. Moreover, novel heuristics inspired by the effective sawtooth pattern can be studied with sound theoretic guarantees.

Acknowledgements This work was supported in part by National Natural Science Foundation of China (Grants 62272210 and 62250710682), in part by the Program for Guangdong Introducing Innovative and Entrepreneurial Teams (Grant 2017ZT07X386), in part by the CCF-Tencent Open Fund (Grant RAGR20220110), and in part by the Market Data Cloud Joint laboratory between Shenzhen Securities Information Co., Ltd. and Southern University of Science and Technology (Grant XB11ZC20210189).

References

1 Al-Dhuraibi Y, Paraiso F, Djarallah N, et al. Elasticity in cloud computing: State of the art and research challenges. IEEE Transactions on Services Computing, 2018, 11(2): 430-447.
2 Qu C, Calheiros R N, Buyya R. Auto-scaling web applications in clouds: A taxonomy and survey. ACM Computing Surveys, 2018, 51(4): 33.
3 Stoll H R. Electronic trading in stock markets. Journal of Economic Perspectives, 2006, 20(1): 153-174.
4 Li F. Cloud-native database systems at alibaba: Opportunities and challenges. Proceedings of the VLDB Endowment, 2019, 12(12): 2263-2272.

5 Easley D, De Prado M L, O'Hara M. The microstructure of the flash crash. Journal of Portfolio Management, 2011, 37(2): 118-128.

6 Shafiq D A, Jhanjhi N Z, Abdullah A. Load balancing techniques in cloud computing environment: A review. Journal of King Saud University-Computer and Information Sciences, 2022, 34(7): 3910-3933.

7 Sajjan R S, Yashwantrao B R. Load balancing and its algorithms in cloud computing: A survey. International Journal of Computer Sciences and Engineering, 2017, 5(1): 95-100.

8 Johora F T, Ahmed I, Shajal M A I, et al. A load balancing strategy for reducing data loss risk on cloud using remodified throttled algorithm. International Journal of Electrical and Computer Engineering, 2022, 12(3): 3217.

9 Sim K M, Sun W H. Ant colony optimization for routing and load-balancing: Survey and new directions. IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans, 2003, 33(5): 560-572.

10 Gures E, Shayea I, Ergen M, et al. Machine learning based load balancing algorithms in future heterogeneous networks: A survey. IEEE Access, 2022. 10: 37689-37717.

11 Farag H, Stefanovič Č. Congestion-aware routing in dynamic IoT networks: A reinforcement learning approach. IEEE Global Communications Conference, 2021: 1-6.

12 Afzal S, Kavitha G. Load balancing in cloud computing-A hierarchical taxonomical classification. Journal of Cloud Computing, 2019, 8(1): 22.

13 Carrión C. Kubernetes scheduling: Taxonomy, ongoing issues and challenges. ACM Computing Surveys, 2022, 55(7): 1-37.

14 Kashani M H, Mahdipour E. Load balancing algorithms in fog computing: A systematic review. IEEE Transactions on Services Computing, 2023, 16(2): 1505-1521.

15 Trivella A, Pisinger D. The load-balanced multi-dimensional bin-packing problem. Computers & Operations Research, 2016, 74: 152-164.

16 Kumar P, Kumar R. Issues and challenges of load balancing techniques in cloud computing: A survey. ACM Computing Surveys, 2019, 51(6).

17 Basu D, Wang X, Hong Y, et al. Learn-as-you-go with megh: Efficient live migration of virtual machines. IEEE Transactions on Parallel and Distributed Systems, 2019, 30(8): 1786-1801.

18 Zhu J, Yang R, Sun X, et al. Qos-aware co-scheduling for distributed long-running applications on shared clusters. IEEE Transactions on Parallel and Distributed Systems, 2022, 33(12): 4818-4834.

19 Zhang S, Guo Y, Guo Z, et al. SMAF: A secure and makespan-aware framework for executing serverless workflows. Science China Information Sciences, 2023, 66(3): 139105.

20 Wang G G, Gao D, Pedrycz W. Solving multiobjective fuzzy job-shop scheduling problem by a hybrid adaptive differential evolution algorithm. IEEE Transactions on Industrial Informatics, 2022, 18(12): 8519-8528.

21 Kandel S, Paepcke A, Hellerstein J M, et al. Enterprise data analysis and visualization: An interview study. IEEE Transactions on Visualization and Computer Graphics, 2012, 18(12): 2917-2926.

22 Salimans T, Ho J, Chen X, et al. Evolution strategies as a scalable alternative to reinforcement learning. arXiv preprint arXiv:1703.03864, 2017.

23 Qian C, Liu D X, Feng C, et al. Multi-objective evolutionary algorithms are generally good: Maximizing monotone submodular functions over sequences. Theoretical Computer Science, 2023, 943: 241-266.

24 Falcón-Cardona J G, Coello C A C. Indicator-based multi-objective evolutionary algorithms: A comprehensive survey. ACM Computing Surveys, 2020, 53(2): 35.

25 Chen L, Xin B, Chen J. Interactive multi-objective evolutionary algorithm based on decomposition and compression. Science China Information Sciences, 2021, 64: 1-16.

26 Deb K, Pratap A, Agarwal S, et al. A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation, 2002, 6(2): 182-197.

27 Wang Y, Xue K, Qian C. Evolutionary diversity optimization with clustering-based selection for reinforcement learning. Proceedings of International Conference on Learning Representations, 2022.

28 Kaushik R, Chatzilygeroudis K, Mouret J B. Multi-objective model-based policy search for data-efficient learning with sparse rewards. Proceedings of Conference on Robot Learning, 2018, 87: 839-855.

29 Schulman J, Wolski F, Dhariwal P, et al. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347, 2017.

30 Mao H, Venkatakrishnan S B, Schwarzkopf M, et al. Variance reduction for reinforcement learning in input-driven environments. Proceedings of International Conference on Learning Representations, 2019.

31 Arora R, Basu A, Mianjy P, et al. Understanding deep neural networks with rectified linear units. Proceedings of International Conference on Learning Representations, 2018.

32 Bian C, Qian C, Yu Y, et al. On the robustness of median sampling in noisy evolutionary optimization. Science China Information Sciences, 2021, 64: 1-13.