2023 Sixth International Symposium on Computer, Consumer and Control (IS3C) | 979-8-3503-0195-3/23/$31.00 ©2023 IEEE | DOI: 10.1109/IS3C57901.2023.00024

Designing an Improved ML Task Scheduling Mechanism on Kubernetes

Hung-Ming Chen*, Shih-Ying Chen, Sheng-Hsien Hsueh and Sheng-Kai Wang
National Taichung University of Science and Technology
No. 129, Section 3, Sanmin Rd, North Dist. Taichung City, Taiwan, ROC.
E-mail: hmchen@nutc.edu.tw*

***Abstract*—As the fields related to machine learning (ML)/deep learning (DL) continue to mature, the MLOps machine learning automation process is also gradually emerging. Then, many open-source MLOps frameworks based on Kubernetes have begun to be proposed. Currently, most Kubernetes-based MLOps frameworks aim to establish a common and easy-to-use ML pipeline environment for users to use based on ML containerized tasks. However, Kubernetes' default container scheduler only considers the resource conditions of individual containerized tasks, rather than considering the scheduling of the entire containerized ML task composition. Such a situation may lead to the system resources not being utilized properly. Therefore, this study designs an improved ML task mechanism based on the Kubernetes-based platform to replace the Kubernetes default scheduler. The scheduling strategy in Kubernetes can be modified to better suit the needs of the machine learning development environment.**

Keywords— MLOps, Task Scheduling, Kubernetes, Machine learning, Deep learning

I. Introduction

With the large and extensive applications of the Internet and the Internet of Things (IoT), the amount of data in the industry has increased. The concept of big data has attracted more and more attention. Through the big data analysis, valuable information from these big data enables enterprises to use these information, improve service quality, help improve problems, or make decisions to create more values. In order to accelerate these large amounts of data into effective analysis, applications in many fields have begun to use machine learning (ML) and Deep Learning(DL) techniques [1].

ML / DL are two main types of Artificial Intelligence (AI) that uses algorithms to learn from large amounts of collected data, identifying patterns and correlations within the data for classification or prediction model training to generate inference models. When new data is obtained, predictions or classifications can be inferenced by the trained models, which can be utilized to develop a variety of applications such as recommendation engines, weather forecasting, facial recognition, and speech processing [2].

With the continuous development of many ML/DL related projects, the integration of DevOps processes into ML/DL development has been proposed to provide a standard training process. The ML/DL development process that incorporates DevOps is called MLOps (Machine Learning Operations) [3]. MLOps is designed to make ML/DL products or services continuously integrated in the development phase, to achieve continuous deployment in the operation phase, and to move towards automation on above phases.

At present, there are many open source MLOps frameworks such as MLFlow [4], Metaflow [5], OSCAR [6], etc. The MLOps frameworks mentioned above are all managed by Kubernetes [7] as a container orchestration platform for ML containerized tasks. Currently, when Kubernetes performs scheduling, it only provides a generalized container computing resources scheduling mechanism. The goal of the aforementioned MLOps frameworks based on Kubernetes is mainly focused on building a universal and easy-to-use ML pipeline environment for users to use. A ML task is defined as a group of subtasks for ML/DL computations in the MLOps pipeline. Those frameworks have not optimized the container scheduling system in the Kubernetes for ML tasks. Therefore, this study proposes an improved ML task scheduling system based on a Genetic Algorithm to improve the container scheduling mechanism in Kubernetes-based MLOps frameworks.

II. Background

A. Kubernetes

Kubernetes is an open-source container orchestration platform developed by Google and hosted by CNCF foundation currently [7]. It is a container resource management and scheduling platform that provides auto-scaling, self-healing, rolling updates, and rollback capabilities. Kubernetes is mainly composed of Master node and Worker nodes [8], as shown in Fig. 1. The Master node runs four main components: API Server, Controller Manager, Etcd, and Scheduler. The Worker nodes is composed of Kubelet, Kube-Proxy, Container runtime, and Pod, which is the container-based ML/DL subtask, for example.

The Kubernetes scheduler is a component on the Kubernetes Master node responsible for scheduling Pods. Its main function is to listen for user requests to create Pods, then binding them to appropriate Worker nodes in the cluster based on placement algorithms and related policies of the scheduler. The Kubernetes scheduler monitors the resource utilization of all Worker nodes in the Kubernetes cluster and adopts appropriate scheduling strategies.

The execution process of Kubernetes scheduler is shown in Fig. 2. After the administrator requests the API server to create a Pod, the Kubernetes scheduler will monitor whether there is a request to create a new Pod. If it receives a request to create a new Pod, it will traverse all Worker nodes and filter to the Worker nodes with available resources e.g. vCPUs or Memory. In addition, the scheduler executes the placement algorithm to score the Worker nodes then select the best Worker Node for the Pod according to the score, it notifies the API server to bind the Pod to the appropriate Worker node after the selection is completed. Finally, repeat the above process until all Pods are scheduled.

The default Kubernetes scheduler schedules containers is relatively simple. It only schedules according to the order in which Pods are created. However, in an actual environment, a complete task often includes multiple related subtasks. How these subtasks are assigned to different Worker nodes may have a great impact on performance. In other words, the

979-8-3503-0195-3/23/$31.00 ©2023 IEEE
DOI 10.1109/IS3C57901.2023.00024

default Kubernetes scheduler only considers the scheduling of each subtask aka Pod according to resource conditions, which does not consider the overall task.

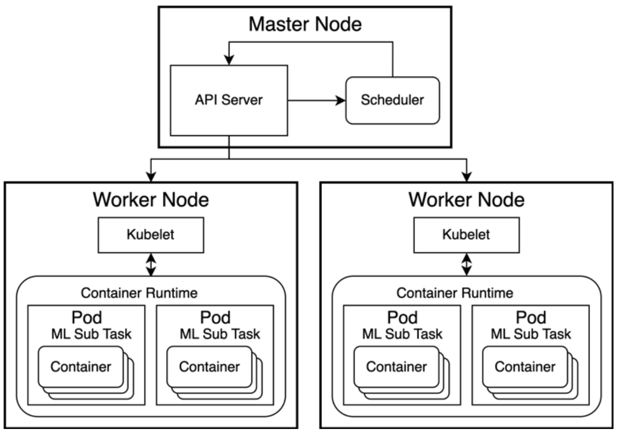

Figure 1. Kubernetes architecture diagram [7]

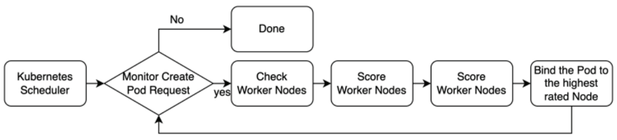

Figure 2. The execution process of the Kubernetes scheduler

B. Multi-objective Optimization Problem

The scheduling optimization problem is to schedule tasks to the appropriate work nodes according to a better order. When a problem has multiple objective functions that require the best solution at the same time, the problem can be called a multi-objective problem. There may be conflicting situations in the objective functions, so some trade-off criteria are needed to meet the requirements, among the multi-objective knapsack placement problem has been proved to be an NP-hard problem [9]. In the research proposed by scholars such as H. M. Fard et al. [10], in addition to the general scheduling problem, the scheduling of tasks in Kubernetes is similar to the multi-objective knapsack placement problem, the multi-objective knapsack placement problem cannot be solved in polynomial time. In the research proposed by J. Huang et al. [11], it is proposed to use reinforcement learning (RL) to rank the Pods in Kubernetes. Scheduling optimization, and the research proposed by M Imdoukh et al. [12], based on Genetic Algorithm (GA) to optimize the scheduler in Kubernetes.

C. Genetic algorithm, GA

The genetic algorithm (GA) is an algorithm derived from natural phenomena such as elimination and evolution [13]. Its principle is to simulate the concept of cell gene crossover. In a living organism, a cell is a basic unit, and the chromosome in the cell carries information of the genetic. Chromosomes are composed of DNA, proteins, and genes are DNA fragments that encode different proteins, representing different biological characteristics. During the process of evolution, organisms undergo cell division, gene exchange, chromosome recombination, and mutations, which occurs due to errors in cell division and gene exchange.

The genetic algorithm flow chart is shown in Fig. 3. In the first step, the objective problem is encoded into chromosomes. In the second step, several gene combinations are randomly generated through the encoding process, the chromosomes are initialized form an initial chromosome set. The third step is to calculate the genetic fitness of each chromosome score. The fourth step is to judge whether the termination condition is satisfied, if the termination condition is satisfied, the better solution of the gene (local optimal solution) will be outputted, otherwise the fifth step will be performed to choose the gene mating point randomly. The sixth step mats the gene for exchange. The seventh step sets the mutation probability, choosing the chromosome for internal gene mutation, this is similar to the way of gene mating point exchange. The eighth step generates a new chromosome, and returns to the third step to repeat until the end condition is met to generate the best solution.

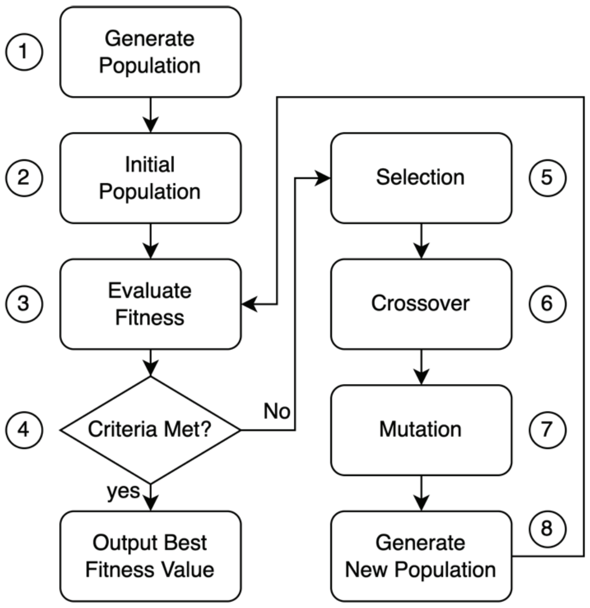

Figure 3. Genetic Algorithm Process

III. THE PROPOSED DESIGN

In order to implement the scheduler of the ML task scheduling optimization strategy, this study regards the scheduling problem of ML task as a multi-objective knapsack placement problem. According to [9], the proposed study defines this problem as NP-hard problem. A genetic algorithm (GA) is designed to achieve ML task scheduling optimization. This study will elaborate on the process of using the GA to achieve scheduling optimization in detail.

A. ML Task Generate Population

In this study, a genetic coding model was designed for the problem of ML/DL task scheduling optimization, as shown in Fig. 4， A chromosome represents an individual, it consists of a Worker node and an evaluation score. A chromosome contains multiple DNA fragments, with each DNA fragment representing a Worker node. Within the DNA fragment, there are multiple ML sub-tasks.

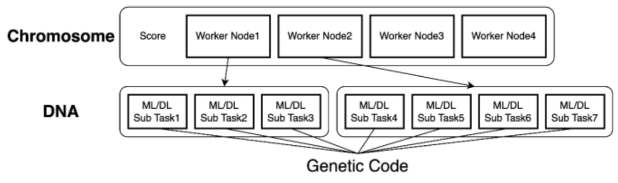

Figure 4. The process of coding chromosome

B. ML Task Genetic Mating(Crossover)

In this section, the detailed process of genetic algorithm crossover and mutation is explained. The ML task gene crossover module will randomly select two individuals from the population to perform chromosome gene crossover, with two chromosomes as an example. The first step is to randomly select the gene mating point, as shown in Fig. 5. The second step is to carry out genetic mating, as shown in Fig. 6. The third step is gene correction, because the new chromosomes produced after mating may have unreasonable conditions, as shown in Fig. 7. Chromosome A has two Sub Task 7 and Sub-Task 8, while Sub-Task1 and Sub-Task2 disappear. Chromosome B has two Sub Task1 and Sub-Task2, while Sub-Task7 and Sub-Task8 disappear. Therefore, it is necessary to modify the chromosomes after mating. The process of gene correction consists of two steps. Firstly, remove the duplicated Subtask genes that were not used in mating. Secondly, randomly insert the lost Sub Task genes into the gene. The flowchart of the correction process is shown in Fig. 8. The fourth step, gene mutation, the main purpose of gene mutation is to discover a new search area, so that there are more possibilities for finding the best solution. ML task gene mutation will randomly select a population chromosome gene from the chromosome gene set to perform the mutation algorithm. The chromosome mutation process is shown in Fig. 9. Randomly choosing a population chromosome A, randomly select two genes Worker node 2 and Worker node 3 located on chromosome A, exchange the Subtasks in the two genes, the chromosome mutation process ends. The decision whether to mutate depends on the parameter mutation rate α, α is between 0 and 1, and the parameter chromosome mutation rate n is also between 0 and 1. In the genetic algorithm, each chromosome has a mutation rate, when the mutation rate of the chromosome is lower than the set mutation rate, the mutation will be carried out, so when n is less than α, the gene mutation process will start.

The above is the algorithm operation process of ML task scheduling based on genetic algorithm optimization proposed by this study. We will continue to iterate the above 1~4 steps until the end condition is met (ex. the number of iterations is reached). Ultimately, outputs a chromosome with the highest fitness score as a result of scheduling optimization.

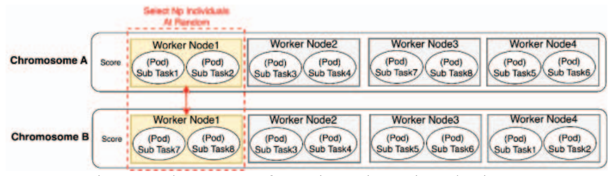

Figure 5. The process of genetic mating point selection

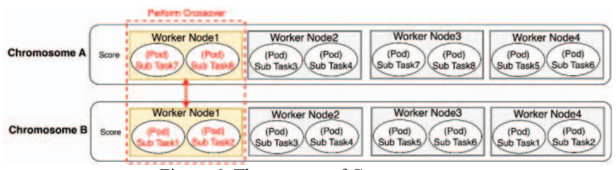

Figure 6. The process of Crossover

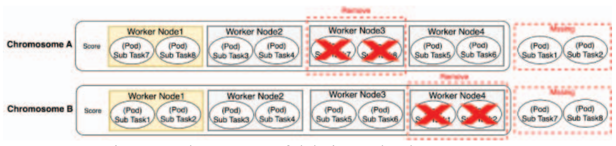

Figure 7. The process of deleting redundant genes

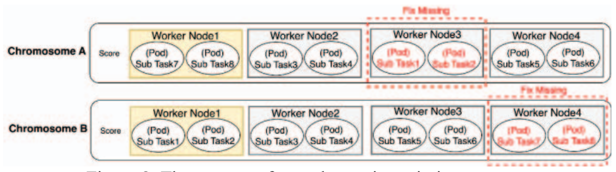

Figure 8. The process of complementing missing gene

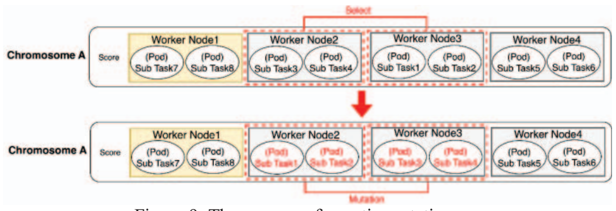

Figure 9. The process of genetic mutation

C. Score Fitness

Finally, there is the scoring formula we designed to calculate Fitness. The scoring formula represents the utilization rate of node resources with scheduled subtasks, the weight of various resources can be adjusted according to needs, as shown in Eq. (1). Where $\alpha^{\{ML/DL\,Task\}}_{\{subtask\},\{node\}}$ represents whether the subtask in the ML task is scheduled in the Worker node. $R^{\{subtask\}}_{u}$ represents the computing resources required by subtask . $R^{\{node\}}_{u}$ represents the computing resources owned by Worker node. u_{θ}represents the weight of computing resources, u represents vCPU or Memory. $R^{\{subtask\}}_{image}$ represents the container image of the scheduled subtask. $R^{\{node\}}_{image}$represents the container image existing on the Worker node. Then, using the final value of fitness function in Eq.(1) as the score to evaluate the placement status of each node in the cluster, the higher the final score, the closer to ML task scheduling optimization solution.

$$Fitness = \theta_u \left[\sum_{\{node\}} \left[\frac{\left(\sum_{\{ML/DL\,Task\}} \sum_{\{subtask\}} \alpha^{\{ML/DL\,Task\}}_{\{subtask\},\{node\}} \times R^{\{subtask\}}_{u} \right)}{R^{node}_{u}} \right] \right] + \sum_{\{ML/DL\,Task\}} \sum_{\{subtask\}} \alpha^{\{ML/DL\,Task\}}_{\{subtask\},\{node\}} \times \left(R^{\{subtask\}}_{image} \times R^{\{node\}}_{image} + T^{\{subtask\}} \right) \qquad (1)$$

IV. EXPERIMENTAL RESULT

An ML task, according to the definition of ML pipeline, includes different types of subtasks such as data preprocessing, data feature extraction, model development, model training, etc. The experiment will simulate random generation of ML tasks with 3 different requirements, each ML task contains 6 subtasks. Each ML task has different computing resource requirements. Hence, this experiment will set up subtasks with high, medium and low computing resource requirements respectively. The high computing resource requirements are 8 vCPUs and 8GB Memory. The medium computing resource requirements are 6 vCPUs and 6GB Memory. The low computing resource requirements are 4 vCPUs and 4GB Memory. Next, 100 ML tasks randomly generated are composed with three types of ML tasks. Each ML task will also be divided into three types with different proportions of subtasks. The first type of ML task is high resource type composed of 50% subtasks with high computing resource requirements, 25% subtasks with medium computing resource requirements and 25% subtask with low computing resource requirements. According to above division, the second type is medium resource type of ML task composed of 25%, 50%, 25%, respectively. The third type is low resource type of ML task composed of 25%, 25%, 50%, respectively.

In this experiment, the ML task Scheduler is deployed in the system, and the ML task optimization scheduling training is carried out through the genetic algorithm. The comparison of the execution time shown in Table 1 by using the random forest prediction for machine learning process as the pipeline in this study. The proposed ML task scheduler and the Kubernetes default scheduler test on three different types of ML tasks combinations. From Table I, it can be seen that among the ML tasks with three different resource types, the total execution time of the proposed ML task scheduler is reduced by about 5.77% in low resource type, by 4.71% in medium resource type, and by 7.03% in high resource type. In summary, the average speedup ratio is 5.83%.

TABLE I. THE COMPARISON OF THE EXECUTION TIME WITH THE PROPOSED THE KUBERNETES DEFAULT SCHEDULER AND THE ML TASK SCHEDULER

	Kubernetes default Scheduler	**ML Task Scheduler (Proposed)**	**Speed Up Ratio**
Low type of the ML Tasks	44,168 secs	41,620 secs	5.77%
Medium type of the ML Tasks	52,643 secs	50,163 secs	4.71%
High type of the ML Tasks	59,863 secs	55,657 secs	7.03%

V. CONCLUSION

In the ML task scheduler proposed in this study, the genetic algorithm is used as the decision-making algorithm to implement the task scheduling optimization mechanism of the ML tasks orchestrating on Kubernetes in order to reduce the total execution time of the overall ML Tasks. By comparing with the default Kubernetes scheduler, the experimental results show that the proposed ML task scheduler have 5.83% average speedup ratio.

REFERENCES

[1] Goodfellow, Y. Bengio, and A. Courville, Deep learning. MIT press, 2016.

[2] T. Wang, Y. Chen, M. Qiao, and H. Snoussi, "A fast and robust convolutional neural network-based defect detection model in product quality control," The International Journal of Advanced Manufacturing Technology, vol. 94, no. 9-12, pp. 3465-3471, 2018.

[3] Sasu Mäkinen, Henrik Skogström, Eero Laaksonen, Tommi Mikkonen,"Who Needs MLOps: What Data Scientists Seek to Accomplish and How Can MLOps Help?", 2021 IEEE/ACM 1st Workshop on AI Engineering - Software Engineering for AI (WAIN) pp.30-31, 2021.

[4] Matei Zaharia, Andrew Chen, Aaron Davidson, "Accelerating the Machine Learning Lifecycle with MLflow", Bulletin of the IEEE Computer Society Technical Committee on Data Engineering pp.39-45, 2018.

[5] Cédric Arisdakessian, Sean B. Cleveland, Mahdi Belcaid, "MetaFlow|mics: Scalable and Reproducible Nextflow Pipelines for the Analysis of Microbiome Marker Data", PEARC '20: Practice and Experience in Advanced Research Computing pp.120-124, July 2020.

[6] "OSCAR", "https://github.com/grycap/oscar"(accessed 2022).

[7] K. Hightower, B. Burns, and J. Beda, Kubernetes: up and running: dive into the future of infrastructure. " O'Reilly Media, Inc.", 2017.

[8] Giuseppe Muntoni, Jacopo Soldani, Antonio Brogi, "Mining the Architecture of Microservice-Based Applications from their Kubernetes Deployment", Advances in Service-Oriented and Cloud Computing, pp. 103-115, 2020.

[9] T. Erlebach, H. Kellerer, and U. Pferschy, "Approximating multi-objective knapsack problems," in Workshop on Algorithms and Data Structures, 2001: Springer, pp. 210-221.

[10] H. M. Fard, R. Prodan and F. Wolf, "Dynamic Multi-objective Scheduling of Microservices in the Cloud," 2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC), pp. 386-393, 2020.

[11] J. Huang, C. Xiao and W. Wu, "RLSK: A Job Scheduler for Federated Kubernetes Clusters based on Reinforcement Learning," 2020 IEEE International Conference on Cloud Engineering (IC2E), pp. 116-123, 2020,.

[12] Harichane, Ishak, Sid Ahmed Makhlouf, and Ghalem Belalem. "A proposal of kubernetes scheduler using machine-learning on cpu/gpu cluster." Intelligent Algorithms in Software Engineering: Proceedings of the 9th Computer Science On-line Conference 2020, Volume 1 9. Springer International Publishing, 2020

[13] V. Barichard and J.-K. Hao, "Genetic tabu search for the multi-objective knapsack problem," *Tsinghua Science and Technology,* vol. 8, no. 1, pp. 8-13, 2003.