Bulletin of Networking, Computing, Systems, and Software – www.bncss.org, ISSN 2186-5140
Volume 10, Number 1, pages 20–24, January 2021

Optimal Resource Allocation for Machine Learning Tasks in Distributed Computing Environments

1st Shoya Kyan
Graduate School of Science and Engineering,
University of the Ryukyus
Okinawa, Japan
k198585@ie.u-ryukyu.ac.jp

2nd Morikazu Nakamura
Computer Science and Intelligent Systems
University of the Ryukyus
Okinawa, Japan
morikazu@ie.u-ryukyu.ac.jp

***Abstract*—This paper considers resource allocation problems in distributed computing environments for machine learning tasks based on mathematical programming and greedy algorithms. We implement a distributed computing platform based on Docker, Kubernetes, and Rancher, mainly for machine learning applications. The simulation results show how the prediction quality of the computation time of machine learning tasks affects scheduling. We also verify our approach by evaluating real machine learning tasks for predicting the backbone structure of proteins.**

***Index Terms*—Resource Allocation, Distributed Environments, Mathematical Programming Problems, Petri nets**

I. Introduction

As IoT machine learning is being applied to many aspects of our daily lives, the demand for efficient computational resources is increasing. Major cloud services and many other cloud environments support enormous computation demands, but it is not always possible to allocate resources efficiently. Although it is possible to utilize a scheduling method based on meta-heuristics in container technology, it is still in its infancy [1].

Many researchers have treated this problem in the literature, such as container scheduling that takes energy costs into account [2], priority queue based algorithms [3], multi-objective optimization scheduling [4], task scheduling for optimizing cost value based on completion time [5], container scheduling for CNC systems [6], and so on.

The resource allocation problem is difficult to solve within a reasonable computation time except for some exceptional cases, such as processors with identical performance and all computation tasks of uniform size. However, recent advances in mathematical programming algorithms have shown that we can solve even the practical size of such problems within a realistic computational time if we formulate them in integer linear programming. Our previous study developed an algorithm to generate integer linear programming problems for scheduling and resource allocation problems based on the Petri Net model. The method makes it possible to use the mathematical programming solver easily. Machine learning processes are a good practical example since their task flows are relatively typical, series of sequential tasks of preprocessing, learning, and prediction.

In this paper, we propose a static scheduling method for processing many machine learning processes in a cloud environment. Our scheduling method can process multiple sequential processes with numerous computational resources efficiently.

In this study, we experiment with three scheduling policies, such as random allocation (RA) and time-ordered greedy allocation (Greedy), and the mathematical programming approach (MIP) with a simulator and a real distributed environment.

II. Preliminaries

A. Middleware for Computing Systems

Docker [7] is open-source software for the virtualization of containers, where we can create and set up many computing servers in distributed environments.

Kubernetes [8] is an orchestration tool for open-source software containers. Conventional container management requires manual monitoring of container resources. However, Kubernetes makes resource management automated. The smallest unit of an application running on a Kubernetes cluster is called a pod. Also, there is a job that creates one or more pods and specifies the number of completed pods.

Rancher [9] is also open-source software to centrally manage Kubernetes clusters, both cloud and on-premises, from a single dashboard.

B. Petri Nets

A Petri net is a directed bipartite graph represented by $PN = (P, T, Pre, Post)$. Place $P = \{p_1, p_2, p_3, ..., p_n\}$ and transition $T = \{t_1, t_2, t_3, ..., t_n\}$ are connected by a weighted directed arc. $Pre(p,t), Post(p,t)$ represents the weight of the arc connecting the place p to the transition t to the place p. Arcs without a weight mean weight 1. The places entering to transition t is called the input place set of t, noted by ${}^\bullet t$, and the places exiting from t is called the output place set of t, noted by $t^\bullet$, respectively. Non-negative integer tokens are placed on places. The distribution of the tokens on the place is called marking, we express it by vector $M^\top = (M(p_1), M(p_2), M(p_3), ..., M(p_n), M(p_n))$, where $M(p_i)$ means the number of tokens placed in place p_i. Therefore, the token distributions in the net model represent

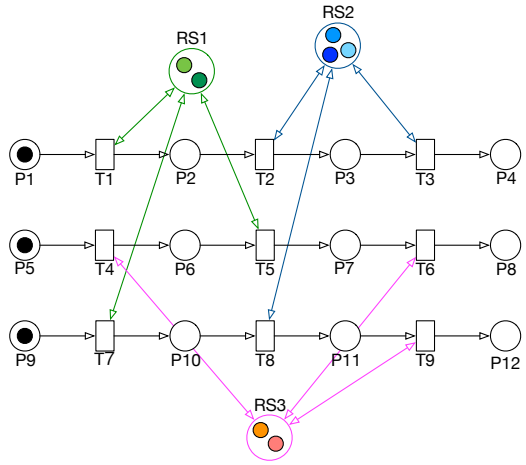

Fig. 1. Petri Net Example of Resource Allocation

the state of the system. M_0 represents the initial state of the system.

Transition t is enabled when $M(p) \geq Pre(p,t)$, for all $p \in^\bullet t$. Enabled transitions can fire. The firing of t removes $Pre(p,t)$ tokens from each place in the input place $^\bullet t$ and locate tokens of $Post(p,t)$ in each output place in $t^\bullet$. The firing of a transition represents the occurrence of an event. The marking before and after the firing indicates the state before and after the event, respectively. Therefore, a series of marking changes by firings of transitions represents the system behavior. Colored Petri nets, an extension of Petri nets, can treat values with tokens and time. Colored Petri nets enable more flexible modeling by adding information to the tokens.

III. Petri Net Model for Resource Allocation

The resource allocation problem is a type of scheduling problem. It is one of the combinatorial optimization problems that minimize the total amount of resources used while generating the overall schedule by assigning resources to tasks. Usually, the length of the overall schedule (delivery date) is given as a constraint or to be minimized.

The following are the conditions for resource allocation problems.

1) Each task requires the requested amount of resources
2) There may be more than one resource of each type. And the same resource can handle different capabilities.
3) Each resource is always available (no failure)
4) The scheduler does not allocate resources to another task until the processing of the task complete.
5) The cost and capacity of the resource are given in advance.

Figure 1 shows three sequential systems of four tasks, each of which assigns appropriate resources from the resource place required by each task. Each resource place represents a pool of resources, and the colored tokens indicate computing resources with a variety of performance (e.g., processing speed, memory size).

IV. Experimental Evaluation

This section evaluates the performance of three resource assignment policies, random assignment (RA), greedy assignment (longer task first) (Greedy), and mathematical programming (MIP) with simulation and a real environment.

A. Simulation

In this subsection, we describe our simulator experiments.

1) Simulation Overview: The simulator has three main functions: generating data, deciding allocation according to the scheduling policy, and visualizing the task schedule from the allocation results.

Each task data consists of three information: task ID, predicted processing time, and actual processing time. The processing time is generated randomly using a uniform distribution in an arbitrary preset range. The actual processing time is also generated from the predicted processing time by the box-muller method, assuming that the error from the prediction follows the normal distribution.

We characterize computing nodes by two information: ID and its processing performance (PP). Processing performance is a positive number, where its value 1 means the node can execute the task in the processing time given as the actual processing time. PP=0.5 means that the node can complete the task within twice the processing time, and PP=2 in $\frac{1}{2}$ of the given actual processing time.

Once the resource set and the task sets are given, the scheduler assigns a resource node to each task based on the scheduling policy. In this experiment, we employed the three scheduling policies, RA, Greedy, and MIP. We automatically generated the mixed integer programming problems from the Petri net model shown in Figure 1.

The quality of the schedule based on Greedy and MIP strongly depends on the quality of the prediction. We assume that the difference between the predicted processing time and the actual one can be represented as a random variable of a normal distribution with the mean 0 and the standard deviation σ. In this experiment, we generate the predicted time for 30 tasks with uniform distribution of range 1 to 1200 and vary the standard deviation σ from 0 to 500.

Our scheduler assigned a computing resource to each task based on the three scheduling policies, RA, Greedy, and MIP.

We used the same processing performance of the same node for all the measurements through the experiment, as shown in Table I.

TABLE I
Processing Performance per Computing Node

Node Name	Processing Performance
node0	1.6351
node1	2.9753
node2	0.8506

2) Experimental Results: Figure 2 shows the average of the makespan by varying the standard deviation. The x-axis shows the standard deviation σ of the random variable representing

the difference from the predicted processing time and the y-axis the makespan.

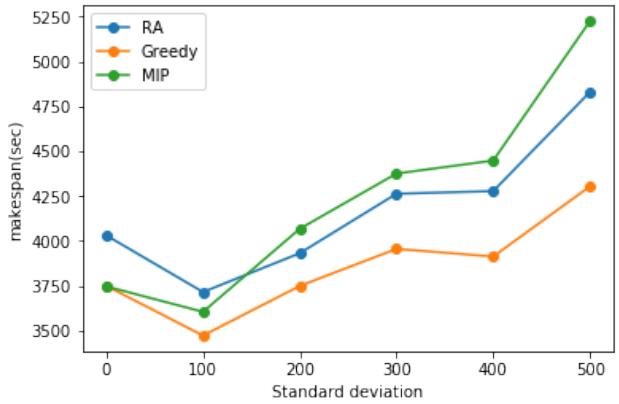

Fig. 2. Mean Value of Makespan vs. Standard Deviation

To show the load imbalance among computing resources, Figure 3 depicts the difference between the earliest node and the latest completion one. The horizontal axis is the standard deviation of the difference between the predicted and the actual processing time. The vertical axis is the difference between the earliest and the latest nodes.

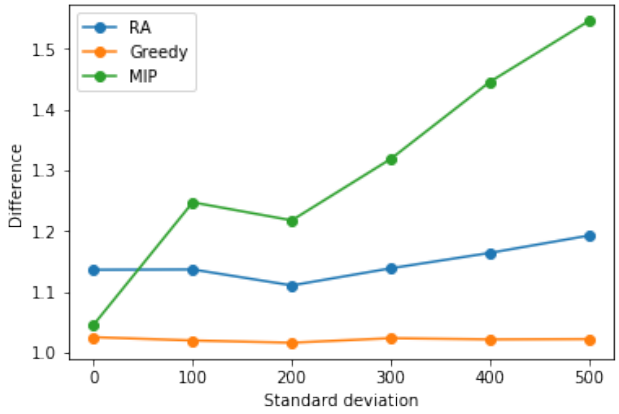

Fig. 3. Load Imbalance vs. Standard Deviation of Prediction Errors

3) Discussion: Figure 2 shows that MIP has the shortest makespan when there is no difference between the predicted and measured values (standard deviation of 0). However, from a standard deviation of more than 200, the MIP-based scheduling becomes worse than the others. We confirm that the MIP-based scheduling is not effective when the quality of prediction is low.

From Figure 3, RA and Greedy do not change much for the prediction quality. Since both of RA and Greedy are dynamic scheduling, the idle time in nodes flexibly is reduced. At the same time, MIP scheduler is a static assignment. Therefore, the scheduling quality is affected by the prediction quality.

B. Experiments in Real Environment

We also performed experiments in a real environment.

1) Developed Distributed Environment: We configured the cluster using Rancher from the perspective that it is possible to build a highly scalable parallel and distributed environment. The cluster consists of four nodes: two nodes using CentOS and the remaining two Ubuntu. Table II shows the node information for the cluster.

TABLE II
CLUSTER CONFIGURATION

Node name	Role	CPU cores	memory
ubuntu4	Rancher Server	6 cores	64GB
anago	etcd, Control Plane, Worker	6 cores	32GB
centos	Worker	6 cores	64GB
ubuntu	Worker	6 cores	64GB

We configured Figure 4 as a distributed environment. This distributed environment performs tasks in three main steps. In the first step, each pod on each node accesses the queue keyed by the node name where the pod itself exists to Redis on the same cluster. In the second step, we get a CSV corresponding to the task number obtained in the first step via NFS. In the third step, each pod is trained using the CSV obtained in the second step. Each pod stores the results in MySQL on the same cluster after the processing. Each pod runs these three steps as many times as we set during deployment.

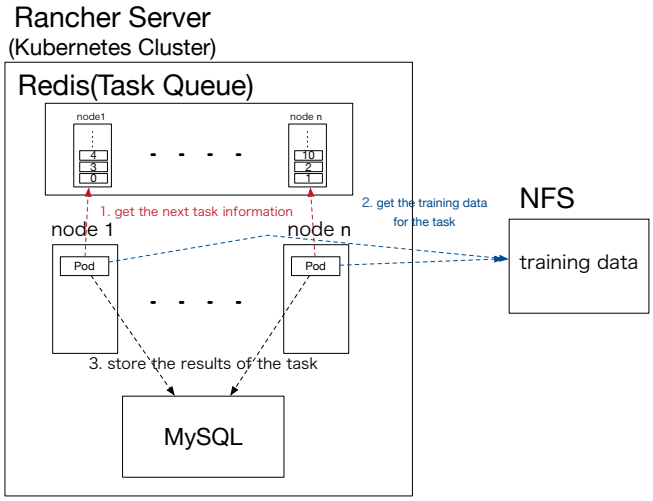

Fig. 4. Cluster Configuration

2) Automatic Generation of Mixed Integer Programming: Figure 5 shows the colored Petri-net model of the resource assignment problem. The place named RESOURCE includes seven computing resources $\{anago, centos, centos2, centos3, ubuntu, ubuntu2, ubuntu3\}$, while the place TASK three machine learning processes.

Step 1:Export the Petri Net model of a figure 5 into XML format file, and convert it into a mixed-integer linear

programming problem in lp file format using our original integer programming tool PN2MIP[7].

Step 2: Solve the mathematical program in the lp file using a solver.

Step 3: Extract necessary information for resource allocation from the output sol file and converts it to JSON format.

Step 4: Generate a yml file to decide which node to allocate a pod to based on the JSON file.

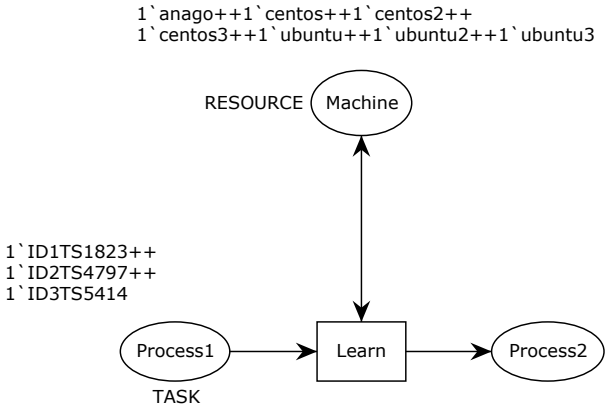

Fig. 5. Petri Net Model for Machine Learning Process

3) Prediction: We employ a simple regression algorithm to predict learning time, linear regression of variables with the maximum degree three. The feature vector is composed of the size of the training data. We generated the feature vector sets by varying the set size from 1,000 to 10,000 with interval 1,000 and measured processing time with two specific learning algorithms, the Lasso regression, and the decision tree regression, on our computing nodes shown in Table labeltab:cluster-info.

For the prediction process, we constructed the function to predict the processing time when we input the task size and the computing resource.

In this experiment, we performed machine learning tasks to estimate the dihedral angles of proteins. In this task, we collected short amino acid sequences with calculated dihedral angles for training data and employed two learning algorithms, the Lasso regression, and the decision tree regression. In the experiment, we measured and compared the total processing time of the schedules generated based on RA, Greedy, and MIP.

4) Experimental Results: We firstly measured ten times the computation time for each scheduling policy, RA, Greedy, and MIP in the Lasso regression and the decision tree regression for 30 tasks. Table III and Table IV show the makespan needed for training by the Lasso and the decision tree, respectively. The results for the Lasso show the order of the length of makespan, RA $>$ MIP $>$ Greedy. In the decision tree ones, the MIP is the worst, followed by RA and Greedy.

5) Discussion: As for the Lasso regression, we found that RA is the slowest. Figure 6 is a Gantt chart for RA. Each

TABLE III
AVERAGE MAKESPAN PER SCHEDULING POLICY [SEC] (LASSO)

	RA	Greedy	MIP
Average execution time	1572.2	1480.2	1549.3

TABLE IV
AVERAGE MAKESPAN PER SCHEDULING POLICY [SEC](DECISION TREE)

	schedule1	schedule2	schedule3
Average run time	7781.2	8587.6	9166.4

computing node is colored differently. The colored rectangles are when tasks are executed, and the blank space between them is the idle time. Looking at Figure 6, we can see that a task with a large processing time is allocated at the end of the ubuntu node overall execution time is longer, and RA is worse than the others.

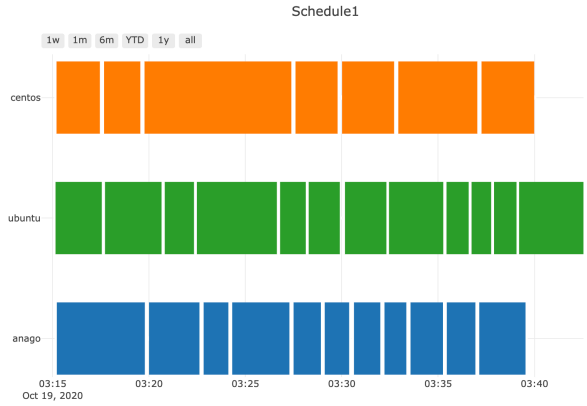

Fig. 6. Grantt Chart of Rasso Regression Tasks based on RA Policy

When we derived the makespan from the predicted time, the MIP was the fastest. However, the MIP was not the fastest in terms of the actual measured processing time since the prediction accuracy was poor. The coefficient of determination between the predicted and measured values was -171.17. Therefore, the fact leads to the worse quality of the MIP-based scheduling.

Concerning the decision tree regression, the MIP was the slowest. The coefficient of determination between the predicted and measured values for all nodes is relatively good at 0.58. Table V summarizes the coefficients of determination of the predicted and measured values for each individual node.

Table V shows that the centos node has a bad coefficient of determination compared to the other computing nodes. The coefficient of determination is worse for certain nodes; that is, the scheduler assigned many data outside the regression equation to a specific node. The fact also makes us consider the typical computer specification and computing resources situation when we collect training data.

TABLE V
COEFFICIENT OF DETERMINATION OF PREDICTIONS AND MEASUREMENTS PER NODE

Node name	coefficient of determination
anago	0.6284921
centos	-0.020542
ubuntu	0.774317

V. CONCLUDING REMARKS

This paper presented some evaluation results of our resource allocation scheme for machine learning tasks. Simulation experiments showed that the MIP-based resource scheduling does not seem efficient when the processing time prediction is low quality, while the greedy algorithm performs very well. Even the greedy algorithms seem to worsen if we mix a wide variety of learning algorithms since the prediction quality becomes lower.

We currently improve the prediction algorithm of learning time, where the algorithm can treat several prediction models and computing resources.

REFERENCES

[1] C. Kaewkasi and K. Chuenmuneewong, "Improvement of container scheduling for Docker using Ant Colony Optimization," 2017 9th International Conference on Knowledge and Smart Technology (KST), Chonburi, 2017, pp. 254-259.

[2] M. Sureshkumar and P. Rajesh, "Optimizing the docker container usage based on load scheduling," 2017 2nd International Conference on Computing and Communications Technologies (ICCCT), Chennai, 2017, pp. 165-168.

[3] Madhumathi RamasamyMathivanan BalakrishnanChithrakumar Thangaraj, "Priority Queue Scheduling Approach for Resource Allocation in Containerized Clouds," Inventive Computation Technologies, ICICIT 2019. Lecture Notes in Networks and Systems, vol 98, Springer, Cham, 2019.

[4] Liu, B., Li, P., Lin, W. et al., "A new container scheduling algorithm based on multi-objective optimization," Soft Comput 22, 7741-7752 (2018).

[5] P. Dziurzanski and L. S. Indrusiak, "Value-Based Allocation of Docker Containers," 2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP), Cambridge, 2018, pp. 358-362.

[6] H. Jin et al., "Architecture Modelling and Task Scheduling of an Integrated Parallel CNC System in Docker Containers Based on Colored Petri Nets," in IEEE Access, vol. 7, pp. 47535-47549, 2019.

[7] Docker, https://www.docker.com

[8] Kubernetes, https://kubernetes.io

[9] Rancher, https://rancher.com

[10] Morikazu Nakamura, Takeshi Tengan, Takeo Yoshida, "A Petri Net Approach to Generate Integer Linear Programming Problems," IEICE Trans. on Fundamentals, vol. E102.A, no. 2, pp.389-398, 2019.