DRL-Cloud: Deep Reinforcement Learning-Based Resource Provisioning and Task Scheduling for Cloud Service Providers

Junquan Yu (junquany@usc.edu)
Rui Guo (ruiguo@usc.edu)
Zhendong Ju (zhendonj@usc.edu)

Abstract:
In phase I, we mainly focused on completing the Round-Robin Method using C++ and understanding how reinforcement learning works by exploring Maze problem. In the first part, we successfully simulated the environment and task scheduling performance, and get the result of the cost based on RRM. For the second part, we do some modification like adding a trap or changing some input parameters to the reference code to deepen our comprehension.

1.Introduction:

Cloud computing is the delivery of on-demand computing services from applications to storage and processing power, typically over the internet and on a pay-as-you-go basis. Rather than owning their own computing infrastructure or data centers, companies can rent access to anything from applications to storage from a cloud service provider. One of the main issues that people care about in cloud computing is how to schedule tasks efficiently to the server, which directly relates to cost. In order to find the optimal solution of task scheduling, people have tried a lot of strategies. Here, we try to apply reinforcement learning to task scheduling and resource provision, expecting to make the system more efficient. Deep Reinforcement Learning (DRL) is a type of machine learning technique that enables an agent to learn in an interactive environment by trial and error using feedback from its own actions and experiences. In our project, we mainly utilized Q-learning technique; Q-learning is a technique that evaluates which action to take based on an action-value function that determines the value of being in a certain state and taking a certain action at that state.

2. Problem Description:

Cloud computing has become the standard paradigm in almost every industry due to booming growing size of data. Cloud Services Providers (CSPs) that owns data centers provides hardware resources and software resources to customers. Profit-driven CSPs charge users for service access, but also, they need to reduce energy consumption and electrical cost as much as possible. This is the reason they need to keep finding ways to minimize energy cost. Resource Provisioning and Task Scheduling are two essential parts in reducing energy. We adopt Deep Reinforcement Learning (DRL), an idea in Machine Learning field, to try to tackle these problems.

3. Challenge:

We face a bunch of challenges in this project.
At the very beginning, there are few models in the paper we need to realize. For example, the Energy Consumption Model and Realistic Pricing Model are the two models we need to implement. Fortunately, the data Google provides has already been classified into specific tasks that we no longer need to implement User Workload Model and Cloud Platform Model. And Energy Consumption Model and Realistic Pricing Model are quite simple to implement.
Secondly is Round Robin Method, Theoretically, Round Robin is a brute force method that it simply goes through all servers in sequence to check if the incoming task can be allocated in to a server, otherwise, reject it. Difficulty of Round Robin Method in our case is as followed: Normally, servers farm is processing incoming jobs and tasks in real-time. But our implementation is to process recorded tasks and jobs that already have timestamp and other information like CPU requested usage, etc. Meaning we need to simulate how time passes by using recorded data.
Secondly, we need to we are asked use C++ mandatorily. As far as we know, python is good at analyzing huge bunch of data, and it is extremely powerful in machine learning algorithm, such as what we will use in the future, DRL. In this case, we need to figure out which part we need C++ to implement.

Next step, all three members have zero idea about machine learning and just started learning how to use python. We need to get familiar with python and at the same time try to familiarize DRL implementation using python.
Finally, according to our initial idea, we first extract data from like .txt, .csv files using C++ and put this huge amount of data into our DQN which is implemented in python. This is also a big challenge, because the data we used to train DQN is of several Gigabytes.
DQN is short for Deep Q-learning Network. This neural network can adjust parameters and choose an action to achieve the biggest Q value. We now are still learning one famous implementation of DQN, how it is used for an explorer to find the treasure. We have 70% figured out the implementation of this instance, like how neural layer is built and how data is input and how to choose action based to previous action and corresponding reward it achieved.

4. Project Timeline:

A. Progress Report:

a) Round Robin

In order to solve the problem mentioned above. We place the tasks in time order following our scheme or so-called algorithm.

b) DQN

We now can use DQN to play a explorer game. Next step is to combine the DQN with idea in the paper.

5. Analysis:

A. solution

(a) Round Robin

In order to solve the problem I mentioned above. We place the tasks in time order following our scheme or so-called algorithm. Round Robin is a naive method used to schedule tasks. The basic idea is to assign tasks to server based on circular order. In our program, the core principle is the requested CPU of current running tasks shouldn't exceed 70% of the total CPU of the server, 100% of total RAM, meanwhile, meet the requested DDL. We let the task go from certain server and check whether it is suitable to assigned to the current server, if not, we continue do this. However, if we cannot find a suitable server after a round, we then push the task into the queue and handle it later. In this algorithm, the priority of the task is based on time. For a server, there are basically three types of situation when the task comes. The first situation is that adding new task won't exceed 70% total CPU that means the task can be executed at once. The second situation is that adding new task will make CPU used between 70% - 100% and RAM used below 100%. In this case, we consider putting it into a queue which means that it will be executed later. To choose which queue, we should also follow the core principle. If there is no queue meet the core principle, we go to the next server to do this again. The third situation is that adding the task
would make used CPU or used RAM exceed 100%. In this case, we simple go to the next server. Besides, we also implement the price model to calculate the cost of all task.

(b) DQN

Before using DQN to achieve our goal, we need to learn how DQN works.
So, we found an online tutorial about how to play treasure-searching game using DQN idea. The explorer starts at origin, try to find the treasure hiding somewhere and at the same time try not to fall into hell. If he falls into one of the hells, game over and he needs to restart at origin.
The game works well, and I think we can modify it to our cloud computing project. We now have the structure, later we need to figure out in cloud computing how environment updates and what parameters we need to input into our Deep Q-learning Network.

B. Pseudo-code:

```
Create vector V to store the current state of each server;
// states represents how many tasks are currently running
While (task comes) //read the data one by one
{
   Get information of current task;
   //do RRM
   pop some tasks from V according to the coming task;
   For (int i; I < total number of server; i++ )
   {
        start to match from the next server according to the last match;
        If(case 1) // diagram 4.1
        {
             calculate price;
             add the state of the coming task to V;
             record the current server;
             return price;
        }
        Else if(case 2)
        {
             eliminate the corner case;
             check whether the task can be placed to the queue of current running task;
             If    (it can)// diagram 4.2.1
             {
                  calculate the price;
                  Pop the corresponding running task in V;
                  add the task to the corresponding place in V;
                  record the current server;
                  return price;
             }
             Else: // diagram 4.2.2
             {
                  go to next server;
             }
        }
        Else// diagram 4.3
        {
             go to next server;
        }
   }
   task go to queue;
   Return 0;
}
```

C. Diagram

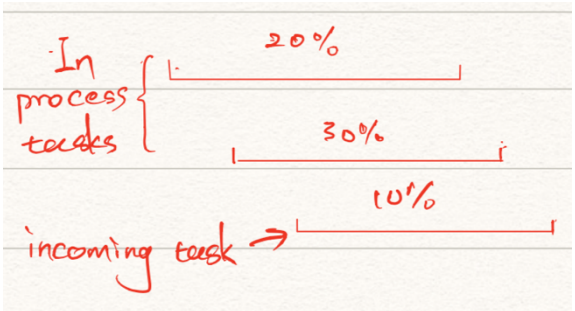


Diagram 4.1

Plus the incoming task, if the total CPU usage of tasks in process is less than 70%, and RAM usage is less than 100%, and also meet the requirement of deadline, the incoming task can be processed immediately.

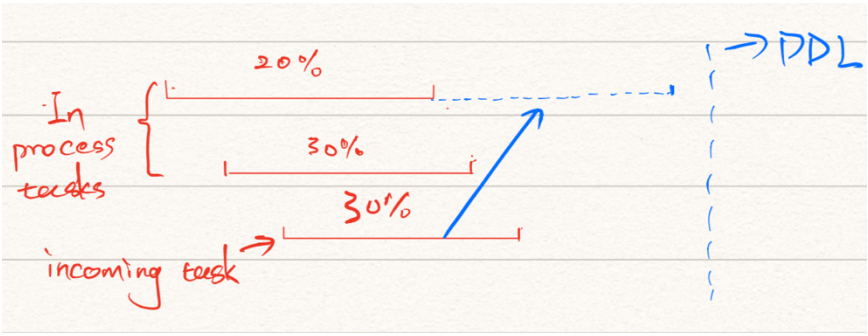


Diagram 4.2.1

Plus the incoming task, if the total CPU usage of tasks in process is greater than 70%, then this task should be pre-placed after some task that is in process to see if it meets the requirement of CPU usage, RAM usage is less than 100%, and also meet the requirement of deadline, the incoming task can placed after that sepicif task.

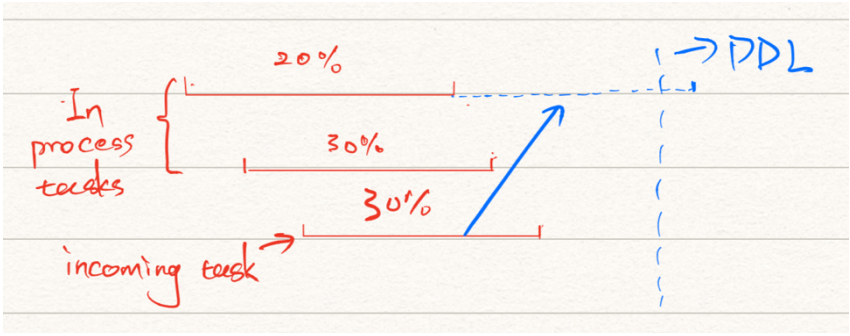
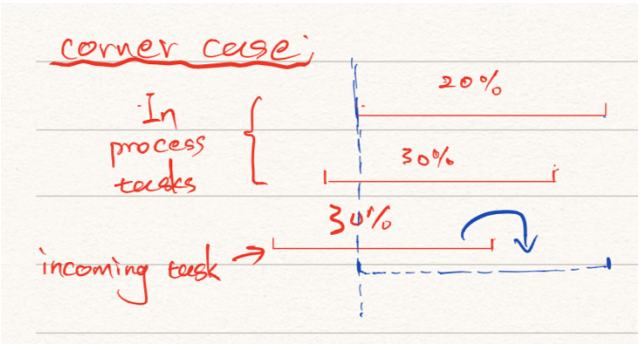


Diagram 4.2.2

The difference of this with diagram 4.2 is that if after placing the incoming task and see this placement does not meet the requirement of deadline. This task will be put into next server.



Corner Case

If the start time of the coming task is earlier than the latest start time of the running time, then it is a corner case. We simply move the start time of coming task equal to the latest start time of the running task.

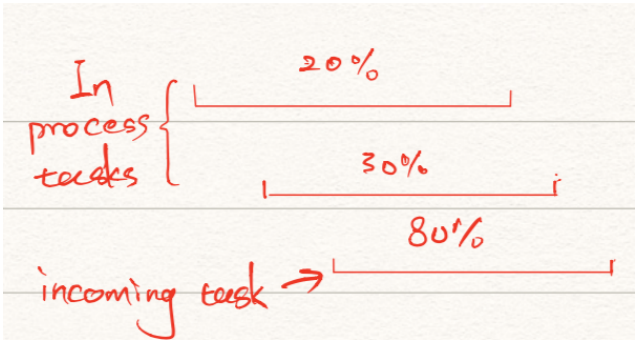


Diagram 4.3

If the CPU usage of incoming task itself is greater than 70%, then this server cannot process this task and this task goes into the next server to see if requirement satisfied.

D. Mathematical model:

a. Energy Consumption Model:

There are two parts in energy consumption model. One is static power, the other is dynamic power.

1. Static Power:
   When CPU usage is greater than 0, no matter the percentage, it will consume certain energy.
2. Dynamic Power:
   When CPU usage is greater than 0 and less than a certain threshold (normally 70%), energy consumption is in a linear range. When CPU usage is greater than 70%, the energy consumption will grow nonlinearly.

This part is directly related to CPU usage, in our "RRM.cpp" code file, it will continuously pass three arguments to this "models.cpp" file, three arguments are start_time, end_time and CPU usage separately. With these three parameters I can compute the power of this specific time duration. after finishing computing the power, we can pass this argument to price model.

b. Realistic Price Model:

[0.12,0.156,0.165,0.117,0.194,0.192,0.318,0.266,0.326,0.293,0.388,0.359,
0.447,0.478,0.513,0.491,0.457,0.506,0.640,0.544,0.592,0.486,0.499,0.292]

We are given the above number array. Each of that represents the price of power within certain hour range. Using the value "energy consumption model" passes, we can then compute price.

Finally, we are done here in "models.cpp" for this particular passed-by parameter.

6. Implementation:

We applied the vector in C++ STL as the container to store the information of the tasks and the servers we used in our code, that's because every element in this two-dimensional vector for the tasks not only indicates the index of the server handling this task, but also shows the index of this task and we need to push/pop the element out/in when we proceed to calculate the price. The vector can provide the flexibility we need in this scenario with no doubt. For the framework of our code, obeying the C++ object-oriented programming principle, we designed the public interface as the only way to access the data and separated the "declaration" portion of the class from the "implementation" portion. If the number of severs is denoted by m and the number of tasks is denoted by n, the space complexity of this program is $O(m+n)$ and time complexity of it is $O(mn^2)$.

7. Reference Summary:

1.Applying Deep Reinforcement Learning (DRL), Resource Provision and Task Scheduling to cloud computing system can reduce the energy consumption under situation that there are large-scale data centers and large amounts of user requests. The Technique used here is exactly Q - learning.

2. The implementation of deep Q-learning algorithm worked successfully with small environments with limited number of dynamic programming states. The result and analysis from the project have shown that deep Q-learning algorithm can be used to generalize the working of autonomous systems in the real world by learning the environment.

3. DRL-based forwarding strategy is an intelligent forwarding strategy based on Deep Reinforcement Learning considering data content and network state. DRL-based forwarding strategy has a good performance in reducing RTT and can improve throughput significantly.
Novelties of Work: We avoid using three-dimensional vector to store all states of servers by only recording the current state of servers and calculating price and energy cost in real time, so that the space complexity can be reduced.