Cloud Resource Allocation and Power Management with Deep Reinforcement Learning

Hongyi Guo
516030910306
guohongyi@sjtu.edu.cn

Shenglong Ye
516030910294
yeshenglong1997@sjtu.edu.cn

1 Introduction

Our Project aims to optimize the task distribution in clusters and the cluster management with Deep Reinforcement Learning (DRL). There are tasks that last for different time and require varied resources in the clusters of big companies such as Google, Amazon, Alibaba, etc. Better allocation of these tasks brings higher service quality and reduces power consumption. Therefore, our group implemented a hierarchical task scheduler for clusters based on the framework of [5]. The hierarchical framework comprises a global tier for VM resource allocation to the servers and a local tier for distributed power management of local servers. And the goal is to achieve the best trade-off between tasks' latency and power/energy consumption in a server cluster. We implemented this complex framework, then analyzed some issues of it and made modification to address them. We also proposed our brand new architecture. Our codes are on github[1].

2 Related Work

Our problem is related to the Mobile Edge Computing, which aims to design an offloading and caching policies. Computation offloading is essential for many prior studies in both mobile cloud computing[4][2] and mobile edge computing[7][11]. They studies what/when/how to offload users' workload from their devices to the edge servers or the cloud. However, they assumed that any computing task can be assigned to an edge server whatever its computing ability, which is the limited resources of the edge server. Cache service also contributes to MEC, and [9]some work has done some investigation in this field. While it considered the storage limitation, it didn't take the computation limitation into account. Based on these research, [13] has proposed an online algorithm called OREO, which jointly optimizes dynamic service caching and task offloading to address service heterogeneity, unknown system dynamics, spatial demand coupling and decentralized coordination. What's more, [10]. and [14] considered there is only single offload in the system, but it can not apply to multiple offloader situations as in practical MCC systems such as mobile devices.

Also, Datacenter-Scale Automatic Traffic Optimization is similar with our problem. It's about three categories of mechanisms: load balancing, congestion control, and flow scheduling. About load balance, it has employed RL in [1], but RL was difficult to implement at line rate in modern datacenters with >10 GbE links[6]. About congestion control, machine learning has been used to optimize parameter[12]. The parameters are fixed given a set of traffic distributions, and there is no adaptation of parameters at run-time.About flow scheduling, unsupervised clustering algorithm has been used to identify flow information, but its decision is still made with fixed parameters[10]. [3] has developed a two-level DRL system to collect network information, learn from past decisions, and perform actions to achieve operator-defined goals. As a result, [15] has proposed an online distributed task offloading (DTO) algorithm for practical MCC systems where each mobile user can dynamically make offloading decisions to nearby mobile devices in order to process computation tasks in a collaborative manner.

[1]Source code address: https://github.com/gohsyi/cluster_optimization

Virtual machine placement is a similar problem.Its goal is to reduce the deployment cost, maximize energy saving and improve user experience, given constraints on hardware configuration and load balancing.However, we cannot simply apply their work to the system because our model is more complex and volatile.

3 Model

3.1 Local Manager

In this section,we decide the sleeping time of each local sever by its own local manager. There will be 2 parts in the process.The first part is a LSTM model which is to predict the interval time of the jobs.The second part is a RL model to decide the sleeping time based on the number of uncompleted jobs and the prediction in the first part.

3.1.1 LSTM

In this part, we try to predict the time of the next job based on the previous interval time with a LSTM model.

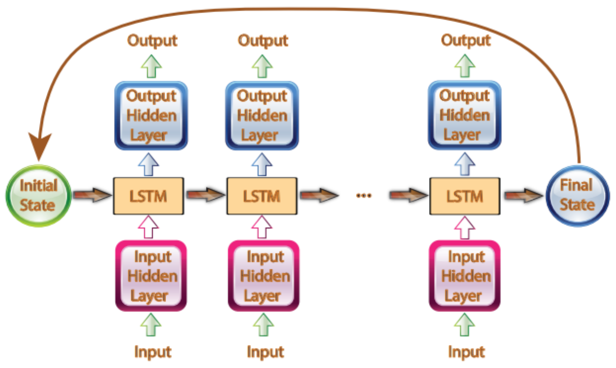


Figure 1: lstm

The figure1 shows the structure of the LSTM we use. The structure has three layers: input layer, LSTM cell layer and output layer. We try to predict the next job inter-arrival time based on previous 35 interval times. The size of the input and output layer are both 1 and we set 30 hidden units in each LSTM cell, and all LSTM cell share the same weights.In our LSTM-based prediction model, we discretize the output inter-arrival time prediction by setting n predefined categories as n states of the RL model.

In the training process, we initialized the the weights for the input layer and output layer as a normal distribution with a mean value of 0 and standard deviation of 1.For the LSTM cell, we initialize it as all 0.The network is updated by adopting Adam optimization

The output of the LSTM model will be the parameter of the state of the RL model.

3.1.2 RL Model

In this part, we try to decide the sleeping time of the local sever based on a model-free RL model.

There are some key time when we should make the decision. Let t_i donate the the of job j arriving. Suppose that we are at $t_i < t < t_{i+1}$. We can make the decision based on the state.

The state of the model at time t is $(M(t), T)$,where $M(t)$ is the machine power,which can be sleep,idle,active etc,T is the prediction of the interval time in the first part.

The moment we make decision is in the following three situation:1) The machine enters the idle state =and no waiting job in the buffer queue.2) The machine is in the idle state and a new job arrives.3) The machine is in the sleep state and a new job arrives.When we meet the second and third situation,we have no choice but to finish the job.The action we can choose is only one, so we do not need to perform the update.In the first situation, we should use the RL model to decide the best sleeping time to maximize the reward and perform the update.The action is the sleeping time we choose.

The reward of the state is $r(t) = -wP(t) - (1-w)JQ(t)$.where $P(t)$ is the power consumption of the server we have mentioned above and $JQ(t)$ is the number of the jobs in the service queue.w is a weighting factor.Based on the previous research,the average number of buffered service jobs is proportional to the average latency for each job.Since the job the local server should do is fixed, the latency is proportional to the number of job in the service queue.The relative weight w between the average power consumption and latency can be adjusted to obtain the power-latency trade-off.

We can show our RL model in the following algorithm

```
1  for each time epoch $t_i$ do
2      $p \leftarrow$ a random number in $[0, 1]$
3      if $p \leq \epsilon$ then
4          $a_i \leftarrow A$
5      else
6          $a_i \leftarrow argmax_a Q(s(t_i), a)$
7      Apply the sleeping time $a_i$
8      If job arrives during the timeout period, turn active to process the job until the job queue is
         empty. Otherwise turn sleep until the next job arrives.
9      Go to the next time epoch $t_{i+1}$
10     Calculate $r \leftarrow$ reward during $[t_i, t_{i+1})$
11     Updating $Q$ based on the reward $r$
```

3.2 Global Manager

The global tier is in charge of resource allocation. Our framework adopts a continuous-time and event-driven decision framework in which each decision epoch coincides with the arrival time of a new VM (job) request. In this way the action at each decision epoch is simply the target server for VM allocation, which ensures that the available actions are enumerable at each epoch. The continuous-time A2C is chosen as the underlying RL technique in the DRL framework. Next, we give the setting of the state space, action space and the reward function. We denote the set of machines by M.

State space

The state space is the union of the server cluster state at job j's arrival time and the job j's state s_j.

$$s^{t_j} = [s_C^{t_j}, s_j] = [u_1 1^{t_j}, \cdots, u_{1|D|}^{t_j}, \cdots, u_{|M||D|}^{t_j}, u_{j1}, \cdots u_{|D|}, d_j]$$

where u_{mp}^t denotes the utilization level of resource type p in server m at time t, and u_{jp} denotes the utilization requirement of resource type p of job j.

Action space

The action is just the index of server for job allocation. The action space for a cluster with servers M is defined as

$$\mathcal{A} = \{a | a \in \{1, 2, \cdots, |M|\}\}$$

Reward

The overall profit of the server cluster equals to the total revenue of processing all the incoming jobs minus the total energy cost and the reliability penalty. The income achieved by processing a job decreases with the increase in job latency, including both waiting time in the queue and processing time in the server.

$$r(t) = -w_1 \cdot Total\_power(t) - w_2 \cdot Number\_VMs(t) - w_3 \cdot Reli\_Obj(t)$$

But in practice, we find that use a rule-based global tier is more efficient than a RL-based global tier. The rule-based model is described as following: when a new task comes, we dispatch it to the server with the lowest CPU usage. Now we prove that this surrogate method can improve the performance by giving the following reasons.

1. When the global tier adopts Reinforcement Learning, there are at least three deep learning models in the framework: global RL model, local RL model and local workload predictor. Note that they are not independent. The predictor's results are part of the local model's input. The local model's decision (sleep or not) will influence the global model's rewards. While the global model's decision (which machine will the current task be dispatched to) affects the inputs of the predictor. These three models tangle together and it's extremely hard to train one model well while some other model performs poorly. So it's best to let them three converge near the same time, but it's hardly going to happen in practice.

2. Note that the two RL models, global RL model and local RL model, cannot guarantee convergence although there is convergence guarantee of the Q-learning methods. This is because the conditions of Q-learning's convergence includes that the environment has to be stationary, which means the environment will give the same reward r_t and state s_{t+1} when the agent makes the same action a_t at state s_t. But due to the inter-influence between the three deep learning models, this stationary condition cannot be guaranteed, and so the convergence of the two RL models.

3. Last but not least, when we use the RL-based global model, the observation is $s^{t_j} = [s_c^{t_j}, s_j] = [u_{11}^{t_j}, \cdots, u_{1|D|}^{t_j}, \cdots, u_{|M||D|}^{t_j}, u_{j1}, \cdots u_{j|D|}, d_j]$, where the $|M| \times |D|$ dimensions of this $(|M| \times |D| + 1)$-dimension vector is the state of machines and have nothing to do with the task itself. But global tier needs to dispatch the task to a specific machine conditional on this observation vector. Think about tasks coming continuously, the machine hardly change the state, the input vectors will not differ much. The actions made by the global tier has a great chance to be the same in a time period. So it's common that tasks are all dispatched to only one machines, causing huge latency.

3.3 Improvement

In this part, we want to propose a brand new architecture to achieve better trade-off between job latency and power usage. According to the analysis above, we notice the key problem of the hierarchical architecture is that deep learning models are not independent and will affect each other. So we designed a special communication architecture to work for both local and global tier and so to help address the problem.

Note that in our proposed architecture, we still adopt a continuous-time and event-driven decision framework. So we still use t to denote the current job j's arrive time. But we will omit this mark in the following part.

In our proposed architecture, there is a Bidirectional-LSTM network. We treat server sequence as a time series. To be more specific, in step τ $(< |M|)$ of this series, RNN cell processes server M_τ, the input to the cell is the vector containing server M_τ's state and the job j's state:

$$x_\tau = [u_{\tau 1}, u_{\tau 2}, \cdots, u_{\tau |D|}, u_{j1}, \cdots, u_{j|D|}, d_j]$$

And we will use each cell's state to compute the probability we dispatch job j to this server via a fully connected layer, and use the state to compute the time period we want to put the server into sleep (or 0, meaning that we keep the server on).

Then we give the explanation of using this quite strange architecture. Actually, this is inspired by [8], where they use Bi-directional LSTM as (1) a communication channel and (2) a memory saver. The bi-direction architecture eliminates the Markov property of RNN and made both forward and backward direction symmetric.

Unfortunately, we don't have enough time to tune this model well. So we will omit the experiments of this model.

4 Experiment

We will introduce the experiments we carried in this part. The dataset we choose is alibaba_clusterdata_v2018.

We compare the performance of stochastic model, round robin model, greedy(rule-based) model, hierarchical model and local model (hierarchical model with rule-based global tier).

1. The stochastic model dispatches jobs randomly and the server will not sleep.

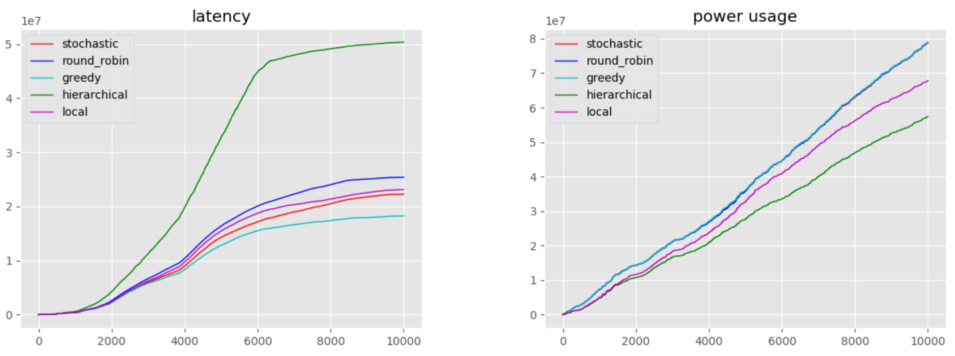
2. The round robin model dispatches jobs in a round-robin way (in turns) and the server will not sleep.

3. The greedy model (rule-based model) dispatches job to the current server with the lowest CPU usage and the server will not sleep.

4. The hierarchical model is the original model we implemented and has been introduced in Section 3.

5. The local model is the hierarchical model but with global tier replaced by greedy model.

Their performances are as in Figure 2. Our local model can save power almost as much as the hierarchical model but the latency is must less than it. The hierarchical model that our reference paper proposed has a very large latency and is not realistic in practice. So our modified model (means local model) achieves best trade-off between job latency and power usage.



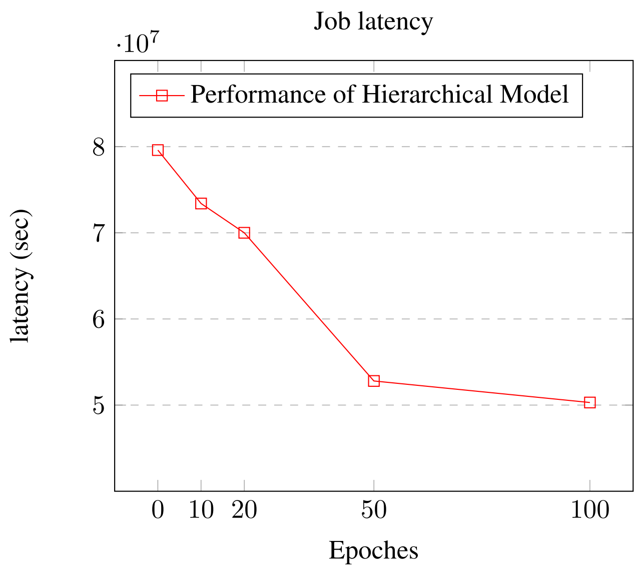
(a) Latency

(b) Power usage

Figure 2: The performances of different models. The graph on the left is the accumulated job latency curve. The x-axis represents the number of jobs and y-axis represents latency (units: sec). The graph on the right represents the total power usage of the clusters. The x-axis represents the number of jobs and y-axis represents power usage (units: kW·h)

.

To show that our reinforcement learning model is actually learning something. We compare the performance of different training epoches as following.



5 Conclusion

Division of the labour:

Name	Survey	Code Framework	Local-Tier Code	Global-Tier Code	Experiments
Hongyi Guo		✓	✓	✓	✓
Shenglong Ye	✓		✓		

Table 1: Division of Labour

Our efforts are as follows:

1. We surveyed nearly twenty papers related to cluster optimization. Then we chose to optimize the trade off between job latency and clusters' power usage and selected a hierarchical model as our reference.
2. We studied different Reinforcement Learning models and chose A2C as both global tier model and local tier model.
3. We analyzed the deficiency of the original hierarchical model, and made some modification to the structure and get some improvement, shown in our experiments.
4. We also proposed a brand new architecture in Section 3.3 to address the issues in hierarchical architecture.

References

[1] Justin A. Boyan and Michael L. Littman. Packet routing in dynamically changing networks: A reinforcement learning approach. In J. D. Cowan, G. Tesauro, and J. Alspector, editors, *Advances in Neural Information Processing Systems 6*, pages 671–678. Morgan-Kaufmann, 1994.

[2] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, 25(6):599–616, 2009.

[3] Li Chen, Justinas Lingys, Kai Chen, and Feng Liu. Auto: scaling deep reinforcement learning for datacenter-scale automatic traffic optimization. pages 191–205, 08 2018.

[4] Niroshinie Fernando, Seng W. Loke, and Wenny Rahayu. Mobile cloud computing: A survey. *Future Generation Computer Systems*, 29(1):84–106, 2013.

[5] Ning Liu, Zhe Li, Jielong Xu, Zhiyuan Xu, Sheng Lin, Qinru Qiu, Jian Tang, and Yanzhi Wang. A Hierarchical Framework of Cloud Resource Allocation and Power Management Using Deep Reinforcement Learning. *Proceedings - International Conference on Distributed Computing Systems*, pages 372–382, 2017.

[6] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. Neural Adaptive Video Streaming with Pensieve. pages 197–210, 2017.

[7] Yuyi Mao, Changsheng You, Jun Zhang, Kaibin Huang, and Khaled B. Letaief. A Survey on Mobile Edge Computing: The Communication Perspective. *IEEE Communications Surveys and Tutorials*, 19(4):2322–2358, 2017.

[8] Peng Peng, Ying Wen, Yaodong Yang, Quan Yuan, Zhenkun Tang, Haitao Long, and Jun Wang. Multiagent Bidirectionally-Coordinated Nets: Emergence of Human-level Coordination in Learning to Play StarCraft Combat Games. 2017.

[9] K. Shashi Prabh and Tarek F. Abdelzaher. Energy-conserving data cache placement in sensor networks. *ACM Transactions on Sensor Networks*, 1(2):178–203, 2006.

[10] Haleh Shahzad and Ted H. Szymanski. A dynamic programming offloading algorithm for mobile cloud computing. *Canadian Conference on Electrical and Computer Engineering*, 2016-Octob(October 2017), 2016.

[11] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. Edge Computing: Vision and Challenges. *IEEE Internet of Things Journal*, 3(5):637–646, 2016.

[12] Keith Winstein and Hari Balakrishnan. Remy_TCP ex machina. *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM - SIGCOMM '13*, 2013.

[13] Jie Xu, Lixing Chen, and Pan Zhou. Joint Service Caching and Task Offloading for Mobile Edge Computing in Dense Networks. *Proceedings - IEEE INFOCOM*, 2018-April:207–215, 2018.

[14] Yang Zhang, Dusit Niyato, and Ping Wang. Offloading in Mobile Cloudlet Systems with Intermittent Connectivity. *IEEE Transactions on Mobile Computing*, 14(12):2516–2529, 2015.

[15] Chongyu Zhou and Chen-Khong Tham. Deadline-aware peer-to-peer task offloading in stochastic mobile cloud computing systems. In *2018 15th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*, pages 1–9. IEEE, 2018.