A Comparison of Machine Learning Algorithms for Automatic Cloud Resource Scaling on a Multi-Tenant Platform

Chun-Hsiang Lee1,8, Zhengda He1,8, Zhaofeng Li3, Xu Lu4, Jian Wang1, Chao Wu2,*
1 Zhejiang Post and Telecom Engineering Construction Co Ltd, 170 Kaixuan Road, Hangzhou, Zhejiang province, China.
2 Zhejiang University, Zhejiang, province, China.
*Email: chao.wu@zju.edu.cn
#These authors contributed equally to this study and share first authorship

Abstract. In an online multi-tenant machine learning platform, the system manager would dynamically load the computing resource according to the tenant’s demand. With cloud computing services, the platform can rent or release computing resources dynamically to fulfill tenant usage which minimizes resource consumption and ensure scalability. Currently, many cloud-based services providers are using the rule-based, threshold auto-scaling mechanism. However, the rule-based method is not efficient, as the nature of availability and cost-reducing violate each other in this method, especially with the sudden increase or variation of the demand. In this paper, we compare several machine-learning-based predictive algorithms to build models based on the information of the system used to predict future usage demands. Decisions made based on this prediction save over 80% cloud resource consumption compared to the rule-based method.

1. Introduction
In an online multi-tenant machine learning environment, the system manager would dynamically distribute the computing resource to tenants as in-demand. However, by increasing user demand, maintaining the resources by hand is not effective or efficient. Moreover, since our consumption of computing resources is linearly based on the number of online end users, and our end-user comprises of real people in general, we face the people activity issue. The reason is the existence of obvious regular time pattern exists in people’s activity. In our case, people are usually active in their spare or study time and expose no active record during their sleep time making our resource consumption curve periodic. Since computing resource is costly, it is not wise to maintain an adequate quantity of resource handling the peak amount of demand from tenant all the time.

To address the above problems, we propose an automatic scaling mechanism to dynamically manage the flexible resource to satisfy multiple tenants’ demands. The system has its own set of resources and satisfies general demands; however, it relies on the dynamically managed cloud resource on peak demand point. Our goal is first, making sure the system is available for all tenants’ demand and second, saving resource as much as possible by intelligently managing the dynamic resources.
In this paper, we present our various experiments and attempt to achieve the above goals. To illustrate the usefulness of our work, we made a comparison based on the difference between machine time of the decision made by our mechanism and the real need of machine time which was calculated based on real-time load.

In this section, we analyze and present the related work in this regard.

Several types of research have been made regarding auto-scaling techniques. In 2013, Jiang et al. proposed a virtual machine (VM) level auto-scaling scheme increasing, decreasing, or making no operation on a VM in terms of their prediction of future resource demands [1]. However, the prediction model is oriented by linear regression which is proven to be a week technique in time series prediction compared to newer techniques [2].

Kanagala and Sekaran in 2013 proposed a threshold-based auto-scaling system, with the threshold dynamically determined by a double exponential smoothing method [3]. However, this time-series-analysis-based method could not effectively remember the peak from long-term memories.

Gandhi et al. in 2014 proposed a reactive auto-scaling mechanism based on monitoring the system resource and performance requirement from the tenants [4]. However, this method passively makes reactions based on system input, while not considering the sudden burst increase in demand.

Nikravesh et al. in 2015 investigated the accuracy of the two-time series prediction model, SVM and NN model on different cloud service workload patterns [5]. They found that SVM and NN have an advantage in various workload patterns and proposed a self-adapting auto-scaling prediction architecture based on NN and SVM. However, they focused on enhancing the demand prediction accuracy not the effect of accuracy improvement on the machine level.

Shahin in 2016 represented an auto-scaling algorithm utilizing the LSTM-RNN network to recognize the upcoming Slashdot, which is the unpredictable sudden increase of workload [6]. The algorithm outperformed other systems.

In this paper, the experiment is made on a multi-tenant system for machine learning users [7], oriented by Kubernetes making freely allocating computing resources possible. The system distributes these computing resources through docker containers in Kubernetes pods to each of our tenants, i.e. our machine learning end-user. The pod here indicates the smallest isolation unit inside the Kubernetes container [8]. Each machine in the cluster could be known as a node, and the system can dynamically manage its cluster of resources by deleting or adding the nodes as the basics of the scaling mechanism. The system could order a new machine from cloud server providers, initiate and add it to the cluster as a node when required.

In our proposed mechanism, the core idea is to build a predictive model for predicting future demands, then taking actions based on the model predictions, thus, several time-series prediction methods are visited.

Time series analysis (Hamilton, 1994) [9] is a heavily studied subject in the last several decades. Its application was broadly applied to the related industrial domains such as store sales prediction and stock price trend analysis. It is caused by learning the pattern from observed historical time series data, in the form of a mathematical formula, and applying the learned function in the future time steps.

The recurrent neural network refers to the kinds of deep learning network structure maintaining extra hidden state and delivering to the next cell. The delivery of hidden states makes the networks consisting of this kind of cell structure memorizing the series input data. Hochreiter & Schmidhuber introduced variants of RNN known as Long Short Term Memory (LSTM) network structure, which added an extra hidden state enabling the network cell to forget delivered states, thus, the gradient explosion was avoided and the network was made to memorize the long series [10].

Convolution neural network is ubiquitous in fields such as image classification, however, it shows no notable advantages in series prediction until the proposal of Bai et al. (2018) [11]. They claimed that the convolution structure-based Temporal Convolution Network (TCN) utilizing causal convolution with residual connection blocks [12], could outperform the RNN families in common long-term series tasks.
The Xgboost [13] is an implementation of the Gradient Boosting Decision Tree algorithm oriented by the decision tree and boosting the ensemble method. Xgboost is famous and considered as one of the most powerful machine learning algorithms since it is appropriate for various types of tasks. There are several winning cases of the data science community competitions in recent years. As part of the decision tree family, Xgboost is not only specialized at finding the relationship between variables, but also provides the ability to explain its decision-making process.

2. Experiment Design

2.1. Problem Specification and Analysis

First, we make clear specifications about our problem, regarding the business requirements and system limitations.

2.1.1. Metric Definition. To measure the number of saved or used resources, we set a business level loss metric as the sum of the machine time for each machine initiated in the period of our test dataset. This metric measures the overall resource consumption made by each strategy.

2.1.2. Data Overview. To tackle this problem, essential data were collected mainly from resource monitor and partially from the event tracker for user activities. As shown in Figure 1 and Table 1, they provide some insight against the curve of the pod number, which is our target. The data were collected for every fixed time interval determined through our analysis on the change rate of data within the time interval of 5 minutes.

![Figure 1](image.png)

Figure 1. The time series target and variables.
Table 1. The business meaning of variables.

| Variable            | Business meaning                                                                 |
|---------------------|----------------------------------------------------------------------------------|
| Pod_num             | Number of pods in the system                                                     |
| active_user_num     | Number of users who used a machine learning environment within 5 minutes         |
| online_user_num     | Number of users with activity recorded by event tracker                          |
| anonymous_user_num  | Number of users with activity recorded by event tracker but did not login         |
| time_since_clean    | Time since we clean inactive user from the system                                |

We collected 14160 records in total, with a 5-minute interval, which is approximately 50 days in the real world. We split the train and test dataset in the 12000th record, take the head of 12000 records as the training dataset, and other 2160 as the test dataset.

2.1.3. System Specification. In this paper, the computing resource in the system is measured through Kubernetes pods. With the capacity of 64 pods in our initial cluster, adding a new node to the cluster gives the cluster 16 extra pods capacity. Adding new resources to the system requires a part of the time. First, we made the cloud providers in order, waiting for the machine to set ready, then installed necessary software components for adding to our cluster. This whole process takes approximately 30 minutes in the real world. Deleting flexible resources from the Kubernetes cluster also needs time uncertainly. The time required to delete a node mainly depends on the time of eliminating the last active pod from the node, which is not concerned with our problem here. To simplify the experiment comparison procedure and remove the irrelevant regards, we set the time of 2 hours for closing a node followed by sending the close signal by our mechanism.

2.1.4. Mechanism Comparison. In this paper, we will visit several methods and models all following the same comparison method.

We made a comparison as follows: In the selected test dataset, at every time step, we run the algorithm, recorded its outputs, and made the simulations. In the simulation, a new node was initiated after 6 time steps the open signal was sent by our mechanism. A node will be shut down after 24-time steps by sending the close signal through our mechanism.

Ultimately, we calculated the business metric described above, based on the simulated data.

2.2. Rule-Based Method
Our goal is to determine the time for initiating a new node or shutting down the extra nodes, hence, we could meet all the tenants' requests while reducing the waste the most. We proposed the first solution based on a simple idea, i.e. initiating a new node whenever the state of the system satisfied some rule.

Inspired by Gandhi et al. [4], the rule is generally based on a supervised item of the system, i.e. the free resource amount. Our rule for initiating a new node is based on the number of free pods left in the system. The rule is a straight line in the plot, which was triggered once the target curve crossed the line.

The specific number of pods left in the rule is determined by another business situation, which is the sudden burst speed of demand from tenants. The fastest growth of pods was observed during a specific interval, from the historical data. To ensure not blocking the requests, we should make sure that the number of pods will not exceed our system limitations before the successful initiation of a new node. Thus, we should assume the increasing speed equal to the burst speed observed in the past, and the number of remained pods is calculated by the burst speed multiplied by the time required for initiating a new node. As previously mentioned, the time needed to initiate a new node is about 30
minutes, while the extreme case of required increasing speed is around 20 pods in 30 minutes, through analysis of our historical data.

The decision-making rule is described by the following Algorithm 1 in detail:

**Algorithm 1** decision making rule

1. $A_{capacity} \leftarrow$ sum of previous capacity
2. $B_{capacity} \leftarrow$ capacity of every single node
3. $Threshold \leftarrow$ The Threshold determined by rule or model
4. $X \leftarrow$ Number of node initiated
5. $NC \leftarrow$ Number of Current pod
6. if $A_{capacity} - Threshold + X * B_{capacity} \leq NC$ then $X = X+1$
7. if $A_{capacity} - Threshold + X * B_{capacity} \leq NC - B_{capacity}$ then $X = X-1$
8. if $A_{capacity} - Threshold + X * B_{capacity} > NC$ then NO Change

To summarize, we should set the rule based on the equation to achieve the availability of 100%. Once the condition is satisfied, we make actions accordingly.

### 2.3 Machine Learning Models

The rule-based system is simple, useful, automatic, and reliable. It does dynamically manage flexible computing resources, but not intelligent and economic. However, to meet its security requirement, it must initiate new nodes in advance at a very early time. The newly initiated resource is mainly not needed since the extreme case of burst increase does not usually happen. By predicting the number of resultant pods in a short time, our response could be more effective.

We found that the prediction system could be optimized by identifying whether the burst increase of demand would happen at some specific time step. We also realized that there is a relationship between the considered target for prediction with some other variables and our business, hence, our model can be used to enhance our prediction ability.

With the above considerations, we tried to build a machine learning model framework. As shown in Figure 2, the prediction model would use some data as the input, to predict the number of pod number after 30 minutes. A key parameter in this problem is the number of history data time steps utilized in the training, which could affect the useful number of train and test data, thus, it should be carefully considered.

We considered this type of prediction as regression since the target was to predict the number. Among several regression metrics, we decided to use Root Mean Square Error (RMSE) as our model metric, since our goal is to make every prediction close to the target as much as possible. The RMSE rather than Mean Absolute Error is selected for the bigger penalty on the great deviation.

First, we focused on the deep learning model, since it has a flexible structure to adjust the number of input and outputs, which is appropriate for time series tasks. The first consideration is LSTM from RNN family for its well-known ubiquitous in recent years, also for the most state-of-the-art model in series prediction is in RNN family. We then considered TCN from another big class of the neural network since its sudden growth in recent years outperforms the LSTM in several series prediction benchmarks. For both deep learning models, in each time step, we took the related inputs and its several previous time steps, to predict the next successive 6 time steps.

Our final machine learning model was based on Xgboost for its explainable model structure. Unlike the deep learning networks, the Xgboost model could only predict one target at a time. Thus, compared to the deep learning models, we trained 6 Xgboost model for every parameter set with the same input data and parameter, with 6 successive targets as the deep learning models.
3. Evaluation & Discussions

3.1. Evaluation
Comparisons were made among the rule-based method and the real need of the system in the business loss metric while using no management method. For the rule-based method, the system manages the resource according to the real-time system loads notably reducing the quantity of waste. However, the machine learning model made great improvements over the rule-based method, by accurately predicting the real need based on the input variables. We plotted the best model’s performance and compared it to the rule-based method.

![Figure 2. The model prediction and comparison process.](image2)

From the Figure 3, it is observed that the predicted node curve is fairly close to the actual need, with only several further time steps. For the availability aspect, it covers the real need curve satisfying the tenant demand in our test dataset.

![Figure 3. The machine initiated via model predictions vs rules.](image3)
Table 2. The model and business loss for a different method.

| methods  | Model loss | Business loss | Diff with actual | Improvement overrule |
|----------|------------|---------------|------------------|---------------------|
| actual   | 82         | 0             |                  |                     |
| rule     | 474        | 392           |                  |                     |
| Xgb-1-step| 0.11391    | 124           | 42               | 89.3%               |
| Xgb-5-step| 0.10994    | 125           | 43               | 89.0%               |
| Xgb-60-step| 0.10911   | 125           | 43               | 89.0%               |
| LSTM-5-step| 0.1261     | 172           | 90               | 77.0%               |
| LSTM-60-step| 0.1184    | 196           | 114              | 70.9%               |
| LSTM-288-step| 0.1164    | 122           | 40               | 89.8%               |
| TCN-5-step| 0.1258     | 253           | 171              | 56.4%               |
| TCN-60-step| 0.1241     | 164           | 82               | 79.1%               |
| TCN-288-step| 0.1365     | 175           | 93               | 76.2%               |

As shown in Table 2, we made several experiments. As mentioned before, the model loss refers to the average regression error RMSE from our 6 time step predictions, and the business loss refers to the sum of machine time for each extra initiated nodes. Since each time step stands for 5 minutes, we decided to present the model results utilizing 5,60,288 steps of historical data to train the model, corresponding to 25 minutes, 3 hours, and 1 day in the real world. Based on the results, we found that the Xgboost model beats the deep learning family model, the model level metric, and the business level metric. Moreover, we found a case where the loss in model level is very close, however, the gap increases in the business loss.

We also found interesting results as follows: while tuning the LSTM network, the model maintains its best RMSE loss with only pod number as input and the loss would increase by adding a new feature to the model. While in the business loss metric, adding variables to the model would increment the performance, which indicates the existence of deviation between our metrics.

3.2. Discussions

Based on the experiment results, we found that the rule-based method possesses several advantages. Firstly, the method is simple, which means its simplicity to implement without errors, it is especially appropriate for the newly started system that is difficult to collect data and build models. Then, it may not be efficient although it utilized the flexible resource, initiated a new node when needed, and shut down the nodes by reducing the demand.

The machine learning model made full improvement over the rule-based method. The curve from Xgboost is almost equal to the curve of real need, implying that it is highly effective to predict the target.

The disagreement between the business loss metric and the model loss metric indicates that there are contradictions in our model building process. The possible reason is that the RMSE mainly measures the average loss over, while the business loss metric mostly focuses on whether the model could correctly predict sudden burst increase of pod number by approaching the current system limitation for the pod number.

4. Conclusion

In this work, we made several experiments and the result indicates that the rule-based mechanism can be considered when the system requires a simple and reliable mechanism. The rule-based system is simple and reliable and requires only minimal parameter calculation to ensure high availability. The Xgboost model and deep learning model taking history time step data as input and outputs future time step prediction could accurately inform the system of upcoming demand pressure. The tree-based
Xgboost model is superior in both level metrics compared to the neural networks, however, all machine learning model makes a great improvement over the rule-based method. Moreover, with the Xgboost model, we could determine the number of other input required variables for the model development.

Some limitations were also revealed: we found that our business metric (the summation of time flexible nodes initiated), is closely related to a key characteristic we measured. The model could accurately capture and follow the burst increasing part of the time series, which is different from our model-level metric RMSE. Furthermore, the amount of collected data is limited so that in our evaluation test set no extreme fast increase occurs making it unclear whether our model could predict such a burst increase.

References
[1] Jiang J, Lu J, Zhang G, Long G 2013 Optimal cloud resource auto-scaling for web applications, in: Cluster, Cloud and Grid Computing (CCGrid) 13th IEEE/ACM International Symposium on Cluster (IEEE, 2013) pp 58-65
[2] Singh, S, Jaishi, H. P, Tiwari, R. P and Tiwari, R. C 2017 Time series analysis of soil radon data using multiple linear regression and artificial neural network in seismic precursory studies Pure Appl. Geophys. 174(7) 2793-2802
[3] Kanagala K and Sekaran C 2013 An approach for dynamic scaling of resources in enterprise cloud in 2013 IEEE 5th International Conference on Cloud Computing Technology and Science (CloudCom) (December 2013) 2 345–8
[4] Gandhi A, Dube P, Karve A, Kochut A and Zhang L 2014 Adaptive, model-driven autoscaling for cloud applications in 11th International Conference on Autonomic Computing (ICAC 14) (Philadelphia, PA: USENIX Association, June, 2014) pp. 57–64
[5] Nikravesh A, Ajila S and Lung C 2015 Towards an autonomic autoscaling prediction system for cloud resource provisioning. In Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (May 2015) pp 35-45
[6] Shahin A 2016 Automatic Cloud Resource Scaling Algorithm based on Long Short-Term Memory Recurrent Neural Network International Journal of Advanced Computer Science and Applications (IJACSA) 7(2) 279-285
[7] Li C, Li Z, Lu X, Chen T, Wu C 2020 Multi-tenant Machine Learning Platform Based on Kubernetes In International Conference on Computing and Artificial Intelligence (IJCAI)
[8] Kubernetes Pods https://kubernetes.io/docs/concepts/workloads/pods/
[9] Hamilton J D 1994 Time Series Analysis (Princeton University Press)
[10] Hochreiter S and Schmidhuber J 1997 Long short-term memory Neural Computation 9 8
[11] Bai S, Kolter J and Koltun V 2018 Convolutional Sequence Modeling Revisited. In Proceedings of the International Conference on Learning Representations (Vancouver, BC, Canada, 30 April–3 May 2018)
[12] He K, Zhang X, Ren S, and Sun J 2016 Deep residual learning for image recognition. In Computer Vision and Pattern Recognition (CVPR)
[13] Chen T and Guer strin C 2016 Xgboost: A scalable tree boosting system. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (ACM) pp 785–794