International Journal of Advanced Research in Education (IJARE)
Volume 5, Issue 1, January-December 2022, pp. 9-21, Article ID: IJARE_05_01_002
Available online at https://iaeme.com/Home/issue/IJARE?Volume=5&Issue=1
Impact Factor (2022): 11.45 (Based on Google Scholar Citation)
ISSN Print: 2395-5236 and ISSN Online: 2395-5244
DOI: https://doi.org/10.34218/IJARE_05_01_002

INTEGRATING AI-DRIVEN AUTOSCALING MECHANISMS IN KUBERNETES-BASED MICROSERVICES ARCHITECTURES

Venkatramana Reddy Panyala
Production Engineer, Yahoo, USA

ABSTRACT

The increasing popularity of cloud-native software has led to the need for improved efficiency in managing resources in containerized environments. The fact that Kubernetes is a de-facto solution when it comes to orchestrating workloads of containers offers inherent autoscaling mechanisms in the shape of Horizontal Pod Autoscaler (HPA) and Vertical Pod Autoscaler (VPA). However, such solutions do not cope well with the dynamics and burstiness of modern micro service-oriented architectures. This paper discusses how a brand-new solution combining Kubernetes-based autoscaling with smart prediction based on algorithms developed by Artificial Intelligence and Machine Learning can be applied. In particular, we will dwell on the combination of Long Short-Term Memory (LSTM), Autoregressive Integrated Moving Average (ARIMA) models, and Reinforcement Learning (RL) to predict the next workload peaks. The given approach will be implemented and tested in a sample microservices environment that consists of auth, order, and payment services. We shall consider the technical design of the approach, its challenges and potential integration possibilities. Early findings indicate that the proposed architecture performs better as compared to the traditional reactive autoscalers and enables the implementation of higher SLO compliance levels and reduction of the risks of latency peaks. We conclude by outlining further research directions.

Keywords: Kubernetes, Autoscaling, Microservices, Machine Learning, LSTM, Reinforcement Learning

Cite this Article: Venkatramana Reddy Panyala. (2022). Integrating AI-Driven Autoscaling Mechanisms in Kubernetes-Based Microservices Architectures. *International Journal of Advanced Research in Education (IJARE)*, 5(1), 9-21.

https://iaeme.com/MasterAdmin/Journal_uploads/IJARE/VOLUME_5_ISSUE_1/IJARE_05_01_002.pdf

1. Introduction

The transition to the pattern of microservices architecture over monoliths has entailed a much-needed change in the way software applications are developed, managed, and deployed. The microservices model consists of decomposing monolithic software systems into smaller units that can be deployed independently, and that can achieve a specific business operation. [1] This pattern brings about flexibility, resilience, and heterogeneity. Conversely, it becomes more challenging to manage such systems with regard to resource allocation and high performance.

Kubernetes is the most popular platform to deploy and manage containerized microservices. The declarative characters of Kubernetes configuration models, and its self-healing feature, make it applicable to various workloads, including stateless web apps and stateful data pipelines[2]. Kubernetes provides several other components; the autoscaling module is one of them and plays a vital role in the cloud-native operations.

Reactive native Kubernetes auto-scaling controllers such as the Horizontal and Vertical Pod Autoscalers, rely on monitoring the existing resource consumption and adjusting the number of replicas or resource limits when some predefined limits are exceeded[3]. Though rather effective in regular traffic, the approaches possess a number of disadvantages which become evident in cases of very dynamic environment. In fact, the time it takes the system to identify an anomaly and assign the needed resources, which is typically between 30 and 120 seconds, creates intervals during which the system is not performing according to its SLOs.

The use of Artificial Intelligence and Machine Learning techniques presents itself as a very promising approach in addressing the shortcomings of the above-mentioned methods. [4][5]First, machine learning-based algorithms are likely to provide adjustments before such a need arises, thereby removing the allocation gap, because they track the patterns in the past data and make predictions on future workloads. Moreover, with the help of Reinforcement Learning systems, it is possible to work out effective scaling policies with the constant interaction with the environment.[6]

The research value of this paper to the research field of intelligent cloud resource management is:

- A single architecture that integrates artificial intelligence-driven predictive models with Kubernetes intrinsically scaled mechanisms.
- An evaluation of the LSTM workload prediction technique and how it can be tailored to microservice traffic.
- The theory of reinforcement learning algorithms to optimize policies with multiple objectives was explored.
- Implementation consideration like metrics pipeline, retraining and backup plans.

A difficulty encountered with integrating AI methods into the container orchestration setting. The remainder of this piece is organized in the following way. In Section 2, the literature review for autoscaling and AI-powered resource management is discussed. Section 3 gives an overview of the background of Kubernetes auto-scaling approaches. Section 4 describes the proposed AI-based autoscaling solution. Section 5 introduces the architecture of the system and its integration. The algorithmic aspects are discussed in detail in section 6. Section 7 contains

information on the strategies of deployment and real-life problems. In Section 8, the difficulties and constraints are discussed.

2. Related Work

Machine learning with cloud resource management has received some significant attention in the academic community in the past 10 years. The first instance of processing cloud resources with machine learning was demonstrated by Mao et al. [7] who trained neural networks to do it. They based their experiments on offline workload traces and used feed-forward neural networks, which were able to forecast 15-minute ahead load with a reasonable amount of precision.

Following the emergence of Kubernetes, academics started looking into its autoscaling capabilities as potential areas for intelligent controllers. Rzadca et al. [8] in their work analyzed large workloads in the real world and those operated at Google but found that policies based on fixed thresholds did not work well in scaling out. Consequently, they concluded the need to have workload-based scheduling policies.

KubeSphere DevOps T. K. Community (2022) is a platform built on Kubernetes, intended to be used by DevOps-oriented teams. The platform incorporates continuous integration, delivery and monitoring in one environment. It increases automation, scalability, and efficiency of workflow. The paper notes the ease of use of Kubernetes in contemporary DevOps.[9]

W. Xu (2022) has published a scalability test report on KubeEdge showing that it supports 100,000 edge nodes. The report measures system performance as latency, stability, and resource consumption. It validates the practicable nature of massive deployment of edge computing. The paper also points to the difficulties in the control of distributed edge systems. [10]

C. Carrión (2022) examined Kubernetes scheduling by offering an in-depth taxonomy of the current methods. Resource fragmentation and inefficient workload distribution are some of the challenges discussed in the paper. It points out shortcomings of existing time scheduling systems. According to the author, more adaptive and intelligent scheduling strategies are required. [11]

S. N. A. Jawaddi, M. H. Johari, and A. Ismail (2022) conducted a review of microservices autoscaling in the framework of formal verification methods. The paper examines the question of validation of scaling decisions as correct and reliable. It finds loopholes in achieving consistency and performance assurances. The authors suggest that verification techniques should be incorporated into autoscaling systems. [12]

C. Carrión (2022) also highlighted the problems of the Kubernetes scheduling, supporting such issues as the inefficient resource allocation and the absence of predictive capacity. The paper restates the significance of enhancing scheduling algorithms. It also emphasizes the possibility of improvement in performance with the use of AI-based methods. [13]

3. Background: Kubernetes Autoscaling:

Kubernetes provides a multi-layered resource management concept, based on the concept of the declarative desired state. The operator determines how he would like his workloads to be

configured by means of objects in the Kubernetes API such as Deployments, StatefulSets, and DaemonSets. Kubernetes will make sure that the desired state corresponds with the current state by spinning up or down resources when needed.

3.1 Horizontal pod autoscaler (HPA):

The Horizontal Pod Autoscaler increases or decreases the number of replicas of the workload based on some metrics. The simplest way in which the HPA works is that it monitors CPU usage on average on all pods in a Deployment and scales the number of replicas based on the average utilization of CPUs. The scaling mechanism is proportional controller; i.e. the scale is computed by the formula: "ceiling(current number of replicas x current utilization)/target utilization).

Kubernetes 1.18 had the Custom Metrics API that enabled HPA to utilize any metrics of external instruments such as Prometheus to scale the replicas. This opened the way to application metrics like request queue length, error rates, and p99 latency as a trigger to scalability mechanisms.

3.2 Vertical Pod Autoscaler (VPA)?

Vertical Pod Autoscaler addresses the resource sizing issue by giving an advice on the number of requests of CPU and memory resources that should be made per pod. The VPA monitors the pattern of resource utilization during the period and adjusts resource constraints accordingly, thereby reducing not only the cost of over-provisioning resources, but also the number of out-of-memory events. The primary disadvantage of this mechanism is that VPA is only able to adjust resource limits upon starting a pod.

3.3 Cluster Autoscaler

Cluster Autoscaler operates on the node level and scales out/in the number of nodes in the underlying virtual machines cluster based on the quantity of scheduled pods. If pods cannot get scheduled due to lack of resources, additional nodes will be added. Alternatively, nodes can be scaled down if they are not used for a certain period of time.The Cluster Autoscaler is integrated with the HPA in which the former ensures the availability of adequate nodes to execute the amount of pods decided by the latter.[14]

Provisioning of nodes is a time-consuming process. Therefore, it would be beneficial to anticipate demand for nodes and scale up nodes even before pods require scheduling since it might take some time to add additional nodes to the Kubernetes cluster.

3.4 Native Autoscaling limitations.

Despite its maturity, native Kubernetes auto-scaling system has a number of challenges requiring AI-based enhancements. To start with, the fact that control loops are reactive means that they cannot anticipate jumps in demand, hence, temporary SLO violations as new pods are being generated. Secondly, threshold values must be manually set, adjusted to each workload, and are not capable of responding to changes in traffic patterns, and can actually cause unwanted oscillations about the thresholds. Third, HPA is not aware of interdependencies among services; so the addition of pods to one service might not resolve the issue but instead merely transfer it to a different service. Finally, since there is no cost consideration in native auto-scalers, pod allocations are made irrespective of their cost.

4. Suggested AI-based Autoscaling Framework.:

The proposed architecture is a further development based on the existing native Kubernetes autoscaler and proposes an AI layer to manage the workloads forecasting, anomalies detection, and the optimization of policies. It is designed as a loosely coupled extension that can communicate with Kubernetes by connecting to its standard API without an intrusive way.

The mechanism as identified in Figure 1 is a closed loop control system. The metrics are collected in real-time on pods, nodes and application endpoints. These metrics are processed by the prediction engine of the AI so that they can generate forecasts of load which can be adjusted to a particular time frame. The scaling policies are decided by such forecasts and then submitted to the Kubernetes API in order to start or stop pods.

4.1 Metrics Collection and Feature Engineering:

The metrics pipeline brings together data availed by different sources. The Kubernetes Metrics Server and node exporter capture infrastructure metrics such as CPU, memory, network and disk I/O. Prometheus collects service metrics such as request rate, error rate and latency percentiles by instrumenting services. The metrics of the business such as the number of sessions and queue are published using application-specific endpoints and registered using the Custom Metrics API. Another significant aspect of metric preparation prior to consumption by AI models is feature engineering. Raw measures are noisy, non-stationary over time, and contain gaps. Pre-processing pipeline: This uses windowing, z-score scaling, and the creation of lagged features to produce a well-structured input format, which can be consumed by sequence models. Trends, seasonality and residuals are decomposed into seasonal decomposition to enable the model to focus on periodicity and residues are regarded as noise.[15]

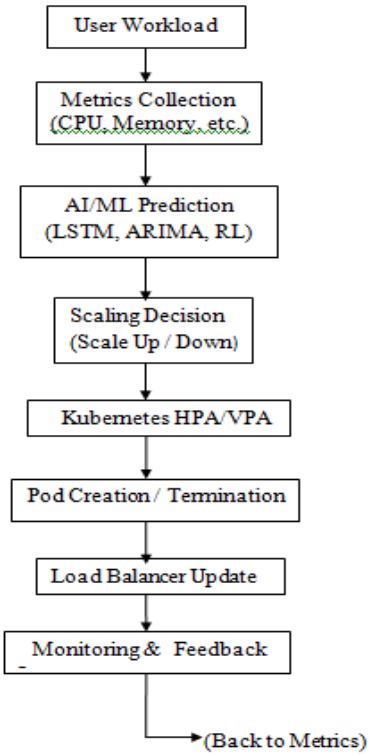

Figure 1: AI-Driven Autoscaling Architecture Flow in Kubernetes

4.2 Workload Forecasting Module :

The central part of the AI layer is the workload prediction module. This module maintains a number of models that are trained with the help of previous workload data, and models are selected in accordance with their recent results. The primary prediction model is an LSTM model which is trained on multi-variate time-series data such as request rate, CPU usage, and response time. Additional secondary models are the ARIMA model of stationary data and the gradient-boosted tree models of data with categorical covariates such as day of week and time of day.

4.3 Scaling Policy Module:

The scaling policy component converts the forecast results into scaling operations. This component takes the output of forecast horizon produced by the workload forecasting component and computes the target replica number to handle the anticipated workload while satisfying SLOs. This policy takes into account the provisioning delay by triggering scaling operation ahead of the predicted workloads by determining the lead time using the provisioning rate of the targeted cluster.

It has yet another reinforcement learning policy component which is an abstraction policy of a higher level than the forecasting policy. [16]The reinforcement learning agent monitors a state feature vector which comprises of the recent measures, historical trend features, and recent scaling policies. The agent makes decisions based on a set of actions, which are finite and may include up scaling, down scaling and doing nothing. The reward function is based on the objectives of the operator, such as the SLO violations and costs.

5. System Architecture and Integration:

The architecture is based on four levels with well defined functions and interfaces. Figure 2 shows the microservices architecture used for deploying the system where the autoscaling mechanism works. The system architecture in the perspective of layer is shown in figure 3.

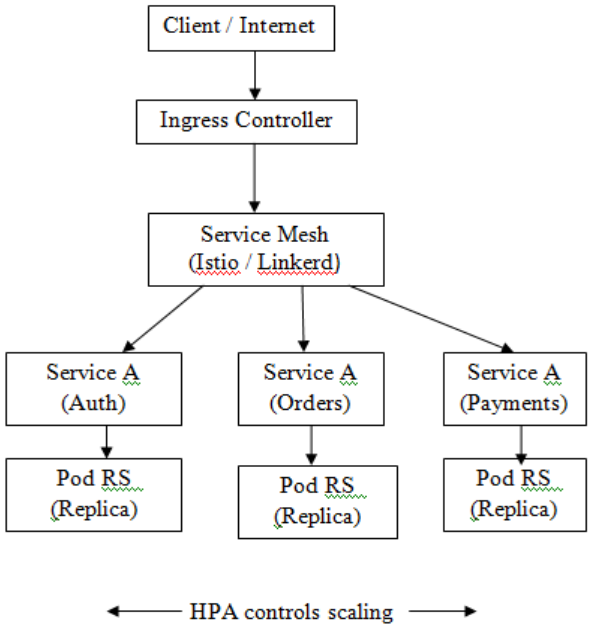

Figure 2: Kubernetes Microservices Deployment Topology

The architecture depicted by Figure 2 represents a typical representation of an e-commerce microservices application. The application consists of three different components: an authentication service, an order service, and a payment service.[17] Each of the services is deployed as a Kubernetes Deployment consisting of a Kubernetes Service and HPA object. Ingress Controller is used to regulate the access to the outside of the system, and the service mesh provides mutual TLS, traffic management, and distributed tracing.

5.1 Infrastructure Layer:

The real or virtual compute resources which are utilized to schedule the Kubernetes nodes are the infrastructure layer. In the case of cloud deployments, a cloud provider virtual machine autoscaling group manages the infrastructure layer, and is operated with the Kubernetes Cluster Autoscaler. Information about the capacity of the resources in the infrastructure layer is fed into the orchestration layer by using the node status API. Infrastructure layer capacity planning must consider the extra capacity overhead due to the AI engine components.

5.2 Orchestration Layer:

Orchestration Layer is the Kubernetes Control Plane that includes the elements of API server, scheduler, controller manager and the etcd data store. The Auto-scaling System of the AI interacts with the orchestration layer through Custom Metrics API of HPA and Scale sub-resource of the deployment resource. The External Metrics API can be used to make the AI auto-scaling system export metrics based on scaling decisions that the AI makes, as first class Kubernetes metrics.

5.3 AI Engine Layer:

The workload forecasting, anomaly detection, and policy optimization engines are contained in the layer that contains the AI engine as described in Section 4. The AI engine is deployed like other applications as a set of microservices within the Kubernetes cluster. The AI engine is exposed through gRPC calls when making live prediction requests and a RESTful interface to configuration and monitoring of the model. The persistent state of every model is kept in a distributed key value store and backed up to object storage. 5.4 Application Layer The application layer comprises the business logic-based microservices which are managed by the auto scaling system. These services have instrumentation for publishing their own metrics via a Prometheus library, thus allowing for application-level scaling triggers. The dependency graph is a dependency graph of these services that is handled by the AI engine.

6. Algorithmic Components:

This section will give a thorough analysis of the algorithmic tools of the AI-based autoscaling framework, which includes LSTM-based forecasting, ARIMA modeling, and policy optimization of Reinforcement Learning.

6.1 LSTM-Based Workload Forecasting:

Long Short-Term Memory (LSTM) models are applicable to workloads prediction because it is a type of model capable of modeling long-term dependencies by means of memory gates. The equations of the LSTM model determine the flow of information through the forget gate, input gate, and the output gate to determine whether to preserve historical context or not.

In the workloads prediction case, the LSTM model takes a sliding window of T time-steps of multidimensional features and provides a prediction of the next H time-steps. The training procedure includes reducing the mean squared loss with a validation set, and applying an early stopping criterion to prevent overfitting. There is dropout regularization between the LSTM layers.

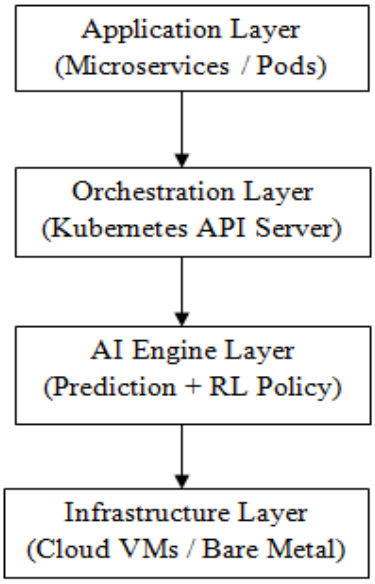

Figure 3: System Integration Layers

An important aspect in designing a LSTM model is choosing the forecast horizon H. A short horizon ($H \leq 5$ min) would provide correct forecasts but would not provide sufficient lead time to pod provisioning. On the other hand, cumulative errors in prediction are high with a long horizon ($H > 30$ min) and may lead to misallocation of the resource. An intermediate value of H, ranging from 10 to 15 minutes, seems reasonable.

Figure 4 illustrates the entire process of the decision making process beginning with the collection of metrics to the actual decision of making the scaling decision. The key element in the workflow that is critical in terms of maintaining forecasting accuracy in the presence of workload dynamics variations is the feedback loop between the scaling process and the model updating pipeline.

6.2 Stationary Workloads ARIMA Model:

For workloads that show stationary statistics, an ARIMA model gives a more lightweight solution compared to LSTM models. Depending on the order of the time series of workloads (p, d, q), an ARIMA model divides the time series into three factors: auto regression, differencing and moving average. Selection of the ARIMA order including application of autocorrelation function (ACF), partial autocorrelation function (PACF) tests and information criteria in the comparison of the various ARIMA models is automatically done by the model selection pipeline of the framework. A SARIMA variant of the ARIMA model suits the workloads which have periodic trend like scheduled jobs to report or other similar activities. The computational requirements for training and evaluating ARIMA models are much less than those of an LSTM model.

6.3 Policy Optimization of Reinforcement Learning:

The process of RL policy optimization begins with the auto-scaling in the form of a Markov Decision Process. The current metrics vector, the past scaling action information and time of the day information form the state space. The actions correspond to a finite number of possible scalings, that include increasing the number of replicas by k, maintaining the current number of replicas, or decreasing the number of replicas by k. The reward functionality is a linear form of negative SLO violation indicators and cost terms, which are tuned to the preferences of the operator. Proximal Policy Optimization is used as policy optimization to ensure the stability of the learning since the importance ratio and value function bootstrap are clipped. Offline training of the policy on an offline simulation of an environment, based on real traces of workloads, and then adapting the policy to a real environment periodically, can be based on a limited exploration budget.

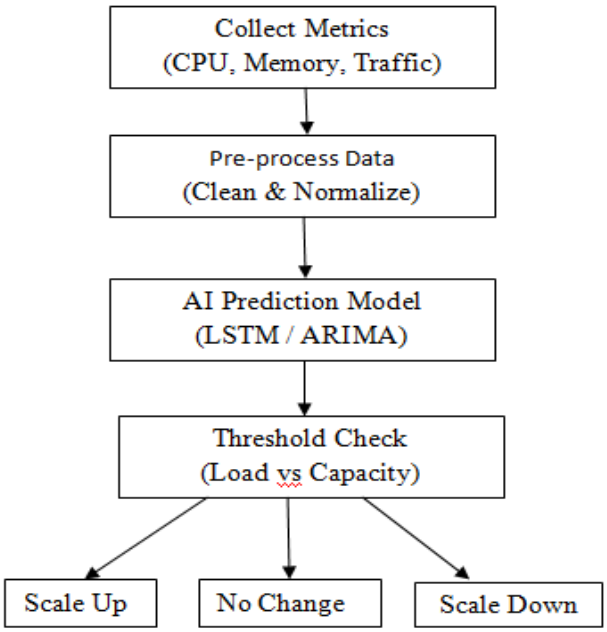

Figure 4: Predictive Autoscaling Decision Loop — From Metrics to Scaling Action

6.4 Anomaly Detection:

The anomaly detecting system will monitor the input signals as to whether they are within the expected range of values according to the predictive model. There can be point anomalies and contextual anomalies, which may be caused by applications problems and traffic attacks, among other errors. These problems need instant response in the form of scaling, which must be executed separately of the predictive pipeline. The system identifies the anomalies in two ways; point anomalies by using statistical control charts and contextual anomalies by using auto encoders.

7. Implementation plans and real world considerations:

Implementing an AI-based autoscaling system in production systems needs to be cautious of operational issues outside of the algorithmic elements. In this section, model lifecycle management, fallback mechanisms, observability, and multi-tenancy are discussed.

7.1 Model Lifecycle Management:

The deployed ML models should have periodic maintenance done to sustain their accuracy as the workload characteristics change. This architecture has been designed in such a way that there is an established training pipeline which periodically trains the models based on fresh data samples, assesses any new candidate models using the holdout data samples, and pushes the models into production which pass the accuracy threshold. Shadow deployment is done on the new models created so that before the model is deployed into production, one can be certain that the models are fine and they are functioning as expected.

7.2 Fallback and Safety Mechanisms:

Just like in the case of auto-scaling, it is a crucial aspect of the production environment. Hence, graceful failure of the AI layer in the event of failures should be ensured by hierarchy of fallbacks whereby in case the LSTM predictive services fail, the ARIMA services should be invoked; in case the AI engines are unavailable, then the native Kubernetes HPA with conservative threshold values are to be invoked. Having this type of structure, AI augmentation will not deteriorate worst-case behavior, but instead boost the best-case performance. Safety is ensured by the use of policy guardian process that ensures the checking of whether all scaling actions are limited by rules that are subject to configuration. Scaling requests should always be checked against minimum/maximum replicas, rate-of-change values and budgets before being submitted to Kubernetes.

7.3 Observability and Monitoring:

It is essential to have observability to control an AI-driven autoscaling system. Observability is supported on the platform by a wide variety of metrics, which are packaged as Prometheus. Such metrics include MAE, RMSE of predictions, scaling operations frequency, percentage of service level objective (SLO) violations, and cost efficiency. Framework performance can be represented using grafana dashboards.

Distributed tracing is also integrated with Jaeger to augment the AI engine with tracing and inference latency metrics, enabling operators to monitor the inference engine. This metric becomes crucial due to its importance in the context of autoscaling operation.

8. Challenges and Limitations:

8.1 Cold Start Problem

A service which is new to itself and having new types of workloads does not have enough past data which can help in making an effective model. It is referred to as a cold start problem and this issue is even more severe when it comes to microservices applications that develop rapidly and their respective services restart with entirely different behavior.

8.2 Model Interpretability

LSTM and reinforcement learning are also complicated and hence hard to debug and identify causes of issues. In cases where the AI system recommends an unintuitive scaling up or down of resources, operators do not have enough information about why this particular action was suggested. Explainable methods such as SHAP values are only partially effective on

LSTMs and can not be used with reinforcement learning. It is a difficult challenge to create explainable systems for autoscaling.

8.3 Computational Overhead

The AI engine has additional resource requirements that must be taken into account in addition to the application load. The inference of LSTM model is relatively efficient as online training and simulation of reinforcement learning may consume significant CPU and memory. The benefits of the optimized scaling decisions may be offset by the added cost of the AI system in resource-limited cluster configurations.

8.4 Distribution Shift and Non-Stationarity

The change in load profiles over time is due to modifications in applications, seasonality, and user behaviors. Previous datasets are prone to becoming outdated and provide inaccurate predictions when distributions vary, since machine-learning models are based on previous datasets. Though this issue is solved by the continuous learning solution, there is still a gap between the change of distributions and the re-estimation of the model. One possible method to reduce this delay is the use of an approach of constant online learning, which is however not stable.

9. Future Research Directions

AI and container orchestration are a new but quickly emerging area. The limitations found in the previous section present several promising avenues of research.

9.1 Federated Learning of Cross-Cluster Models:

Federated learning algorithms, which train the models using aggregated data, can be used by the multi-cluster organizations situated in different cloud environments and different geographical locations. Federated learning algorithms make it easy to train models with distributed data without centralizing the data. Federated learning algorithms have the potential for being customized to suit the non-IID nature of multi-cluster workloads.

9.2 Dependency-Aware Scaling using Graph Neural Networks.

The microservices architecture is typified by complex dependency relationships among various microservices, which affect their scalability ability. In the event that a dependent microservice is scaled to meet increasing demand, then this may shift the bottleneck up the dependency pattern.

9.3 Sustainability-Driven Autoscaling

The greenhouse gas emission and the power consumption of data centers have become a major concern among the cloud service providers and enterprises. In the future, carbon-intensity signals can be incorporated into the set of criteria used by auto-scaling systems to optimize the scaling process, and workloads can be scheduled to run at times when free power is in abundance.

10. Conclusion

The study introduces an all-in-one approach of using AI-based autoscaling and microservices on a Kubernetes setup. It employs prediction policies such as LSTM, ARIMA,

and Reinforcement Learning to optimize scaling policies and minimize the provisioning delays due to bursty workloads, thus minimizing SLO violations. It is loosely-coupled, clearly-defined, and safely degrades, and uses standard Kubernetes APIs, without locking in with a vendor. Nevertheless, there are still some challenges, such as cold start problems, model interpretability, computational complexity, and pod-node autoscaler coordination. The following areas have research opportunities. With the progress in cloud-native systems, resource management will require intelligence. This effort is a step in the direction of infrastructures that can autonomously make decisions and AI-based orchestration is likely to become a common Kubernetes feature.

References

[1] Burns, B., Grant, B., Oppenheimer, D., Brewer, E., & Wilkes, J. (2016). Borg, Omega, and Kubernetes. ACM Queue, 14(1), 70–93.

[2] Hutter, F., Kotthoff, L., & Vanschoren, J. (Eds.). (2019). Automated Machine Learning: Methods, Systems, Challenges. Springer.

[3] Peng, B., Chen, Y., Schuster, M., Pierson, H., & Ding, S. (2021). Elastic scaling in cloud-native environments using predictive control. IEEE Transactions on Cloud Computing, 9(2), 544–558.

[4] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347.

[5] Li, C., Zheng, C., & Shen, H. (2020). Workload prediction for cloud computing using recurrent neural networks. Future Generation Computer Systems, 113, 285–296.

[6] Casalicchio, E., & Silvestri, L. (2013). Mechanisms for SLA provisioning in cloud-based service providers. Computer Networks, 57(3), 795–810.

[7] Mao, H., Alizadeh, M., Menache, I., & Kandula, S. (2016). Resource management with deep reinforcement learning. In Proceedings of the 15th ACM Workshop on Hot Topics in Networks (pp. 50–56). ACM.

[8] Rzadca, K., Findeisen, P., Swiderski, J., Zych, P., Bronson, P., Ostrowski, D., ... & Wilkes, J. (2020). Autopilot: workload autoscaling at Google. In Proceedings of the 15th European Conference on Computer Systems (EuroSys). ACM.

[9] T. K. Community, “Kubesphere devops: A powerful ci/cd platform built on top of kubernetes for devops-oriented teams,” https://kubesphere.io/devops/, 2022.

[10] W. Xu, “Test report on kubeedge’s support for 100,000 edge nodes,” https://kubeedge.io/en/blog/scalability-test-report/, 2022.

[11] C. Carrión, “Kubernetes scheduling: Taxonomy, ongoing issues and challenges,” ACM Computing Surveys (CSUR), 2022.

[12] S. N. A. Jawaddi, M. H. Johari, and A. Ismail, “A review of microservices autoscaling with formal verification perspective,” Software: Practice and Experience, 2022.

[13] C. Carrión, "Kubernetes scheduling: Taxonomy, ongoing issues and challenges," ACM Computing Surveys (CSUR), 2022.

[14] Mao, H., Schwarzkopf, M., Venkatakrishnan, S. B., Meng, Z., & Alizadeh, M. (2019). Learning scheduling algorithms for data processing clusters. In Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM). ACM.

[15] Thinakaran, P., Gunasekaran, J. R., Sharma, B., Kandemir, M. T., & Das, C. R. (2019). Kube-knots: Resource harvesting through dynamic container orchestration in GPU-based datacenters. In Proceedings of the IEEE International Conference on Cluster Computing. IEEE.

[16] Wang, C., Urgaonkar, B., Gupta, A., Kesidis, G., & Lim, Q. (2016). Effective capacity modelling in cloud computing: A fine-grained workload prediction approach. In Proceedings of the 12th IEEE International Conference on e-Science. IEEE.

[17] Ahmad, I., & Ranka, S. (Eds.). (2020). Handbook of Energy-Aware and Green Computing. CRC Press.

[18] Shahrad, M., Fonseca, R., Goiri, I., Chaudhry, G., Batum, P., Cooke, J., ... & Bianchini, R. (2020). Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider. In Proceedings of the USENIX Annual Technical Conference (ATC). USENIX.

[19] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. In Advances in Neural Information Processing Systems (NeurIPS). MIT Press.