Minimize Cold Start Delay in Serverless Computing using Workload prediction

End Review Presentation

Anish Agarwal -197110, Vedant Gandhi -197187, D.Sai Teja -197123
Under the supervision of
Dr. Rashmi Ranjan Rout

National Institute of Technology Warangal, India
({agarwa_961937,gandhi_961973,daggu_961911}@student.nitw.ac.in)

May 12, 2023

Table of Contents

Introduction

- Serverless computing has revolutionized the world of cloud-based and event-driven applications with the introduction of Function-as-a-Service (FaaS) as the latest cloud computing Model.
- It brings benefits such as ease of development, saving resources, and reducing product launch time for enterprises and developers.
- One of the features of serverless computing is ability to scale the containers to zero, which results in a problem called cold start problem. The challenging part is to reduce the cold start latency without the consumption of extra resources.

Background

- Serverless Computing does not mean that there are no servers, it merely means that the servers are there, but the user does not have the hassle to manage those.
- Serverless Computing comes with many advantages. Significant of them are no back-end operational overheads means the CSP does all back-end-related stuff such as server management, auto-scaling, load-balancing and others so that the users focus only on developing the business logic and pay-per-use scheme, which means the execution cost of any function charged on the millisecond scale.
- However, despite having much prosperity, Serverless Computing has several drawbacks that deteriorate its performance at an extensive level; the Cold-start problem is one of the major of [1].

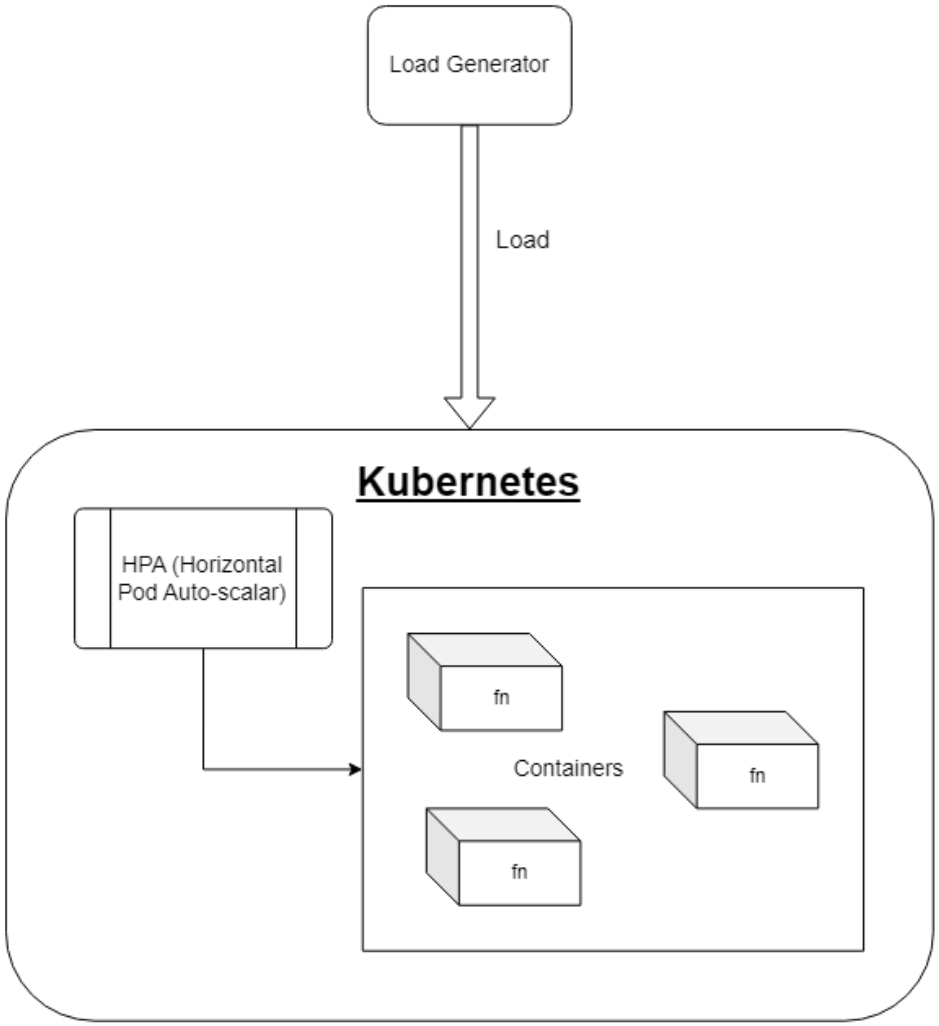
Related Work

- Tools such as CloudWatch, Thundra.io, Dashbird. io, and Lambda Warmer were used to set rules to periodically invoke functions and warm-up them (for example, every 5 minutes).
- [2] uses the Q-learning algorithm to determine the number of function instances dynamically. The number of available function instances and the discretize CPU consumption of each function, are considered as environmental states, and the action is scaled up or down of function instances.
- Researchers in [3] explain the phenomenon of cold starts with respect to the Knative serverless platform and suggest a pod migration technique to reduce the cold start of the function containers.

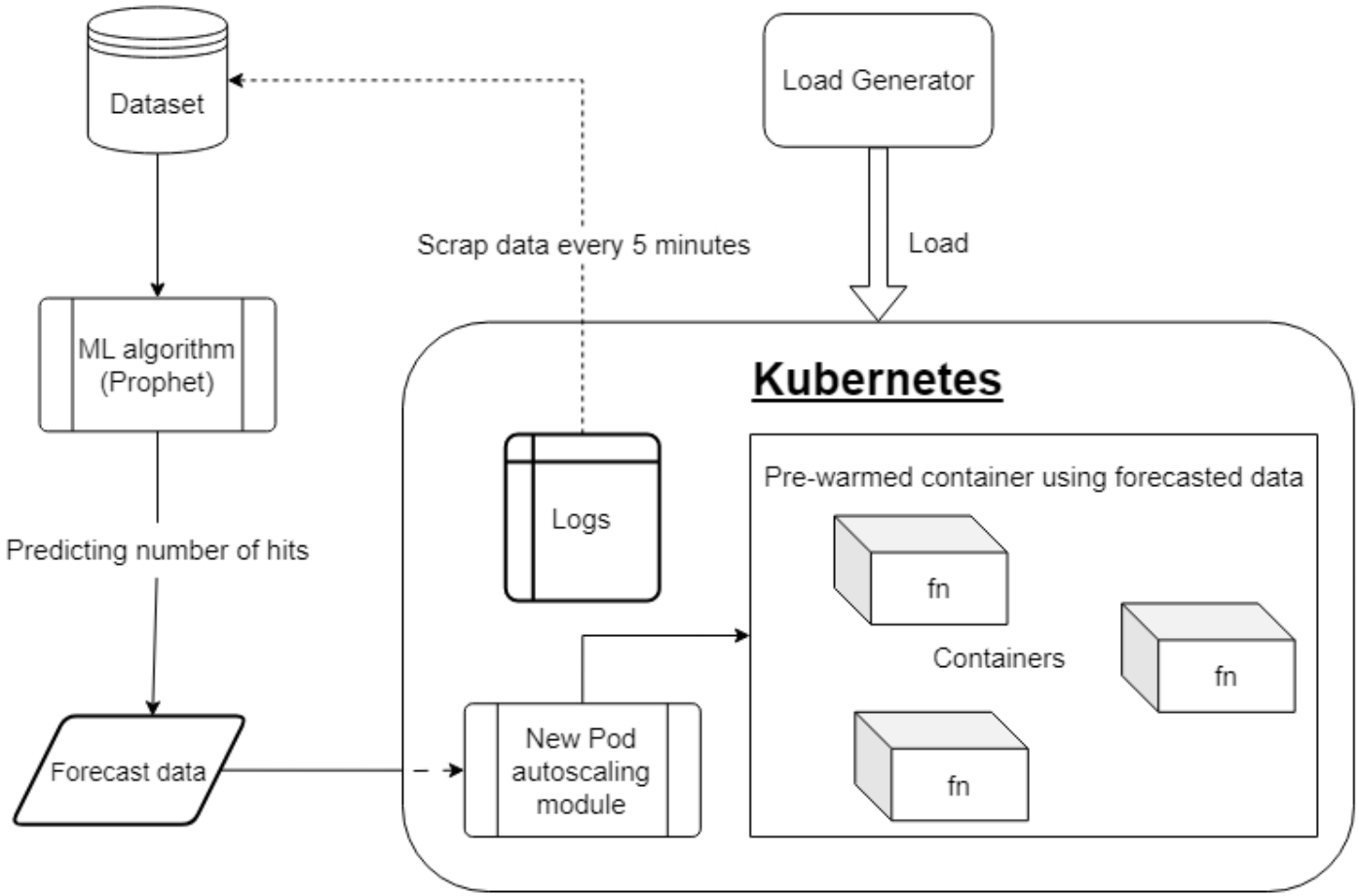
Proposed Approach

- Implement an ML model that predicts the number of future function invocations based on the past data.
- A learning-based time series forecasting algorithm in Kubernetes autoscaler that, based on predicted number of future function invocations, allocates computing resources accordingly, making it possible to increase the containers before high function invocation traffic.
- Comparative analysis of our proposed model against HPA, that is inbuilt in Kubernetes for the synthetic function workload pattern.

Traditional Kubernetes Approach



Proposed Approach contd.



Proposed Algorithm.

Algorithm 1 Scaling algorithm

Initialize: $reqPerPod =$ int(sys.argv[1]), $holidays[] =$ int(sys.argv[2]), $first\_inv = False$
for pred in prediction **do**
  if current_date is a holiday **then**
    $pred+ = holidays[current\_date]$
  end if
  if $first\_inv == True$ **then**
    $prev\_pods = \lceil \frac{pred}{reqPerPod} \rceil$
    $current\_pods = \lceil \frac{pred}{reqPerPod} \rceil$
  else
    $current\_pods = \lceil \frac{pred}{reqPerPod} \rceil$
  end if
  try
    Scale pods with $\max(current\_pods, prev\_pods)$
    $prev\_pods = \lceil \frac{pred}{reqPerPod} \rceil$
  end try
  if exception caught **then**
    print("error")
  end if
  $first\_inv = False$
end for

Experimental Results I

- **Platform:**
  For the simulation, we have used the python programming language. The python programming language consists of a set of libraries. It is used to simulate the network traffic pattern. For the experimental purpose, we have used the Google Colab notebook IDE.
- **Dataset:**
  In this work, a prediction model is used which predicts the average load on hourly basis of a distributed server of 1 year for an interval of 24 hours.[4] The dataset contains two attributes: timestamp and number of hits respectively.
  For the experimental purpose, we have divided the data set in two parts. The first part of divided data has been used to train the system and the second part of data is used for testing the prediction. The testing part is used to test the accuracy of the predicted data.

Experimental Results II

- We implemented various ML models i.e. Prophet model, SARIMA model and LSTM model.

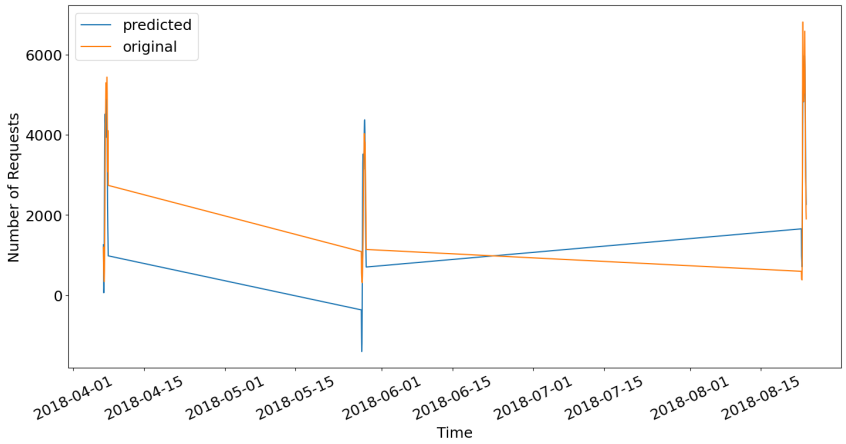
Comparison of results for ML models.

MODEL	MAE	MAPE
PROPHET(HOLIDAY)	525.61	26.45
PROPHET	650.82	35.67
LSTM	585.63	31.46
SARIMA	590.74	32.26

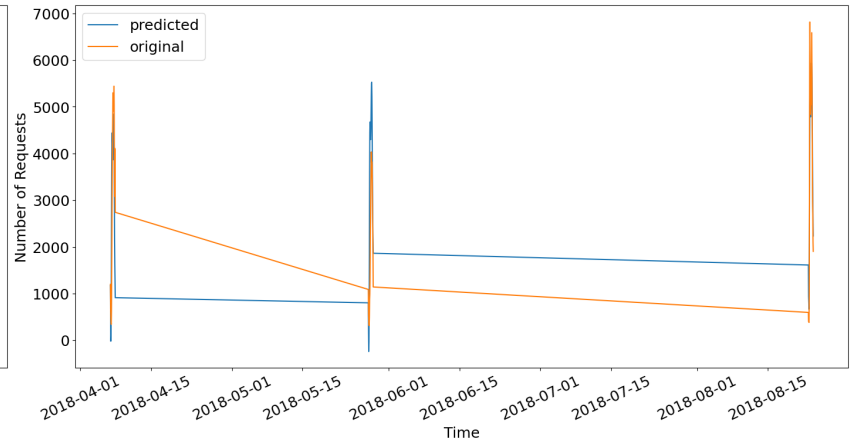
Experimental Results III

Comparison after using holiday feature.

Prophet with holiday



Prophet without holiday



Holiday weights obtained from Prophet model

	ds	holiday	weight
0	2012-11-22	Thanksgiving Day	-1576.558931
1	2012-12-11	Veterans Day	-128.870983
2	2012-12-25	Christmas Day	-1323.619976
3	2013-01-01	New Years Day	-1088.680924
4	2013-02-18	Washingtons Birthday	-432.528907

Experimental Results IV

- We tested the autoscalers with different volumes of load and ran them for a period of time to get the average latency incurred using them.

Custom Autoscaler

HPA

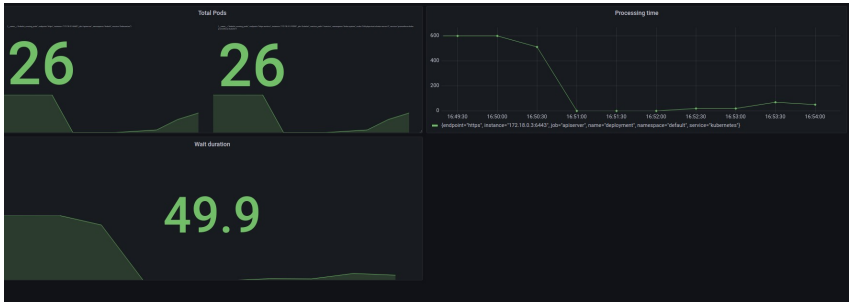


Figure 1: Comparison for light load (1000-2000 users).

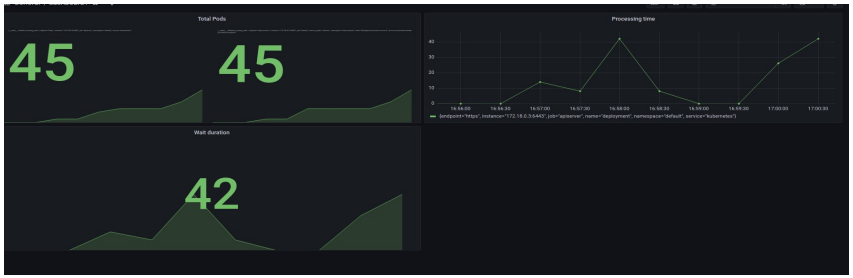
Experimental Results V

Custom Autoscaler

HPA

Figure 2: Comparison for medium load (4000-5000 users).

Custom Autoscaler



HPA

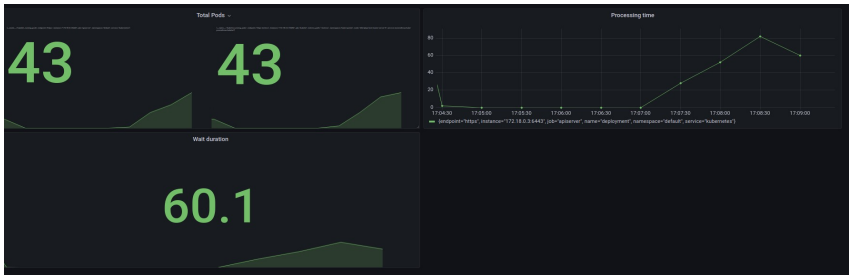


Figure 3: Comparison for heavy load (8000-10000 users).

Experimental Results VI

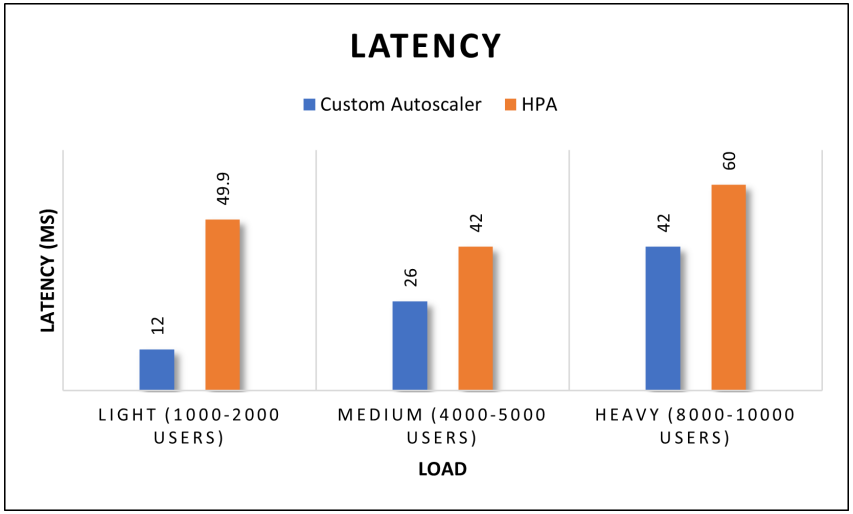


Figure 4: Comparison of latency.

Experimental Results VII

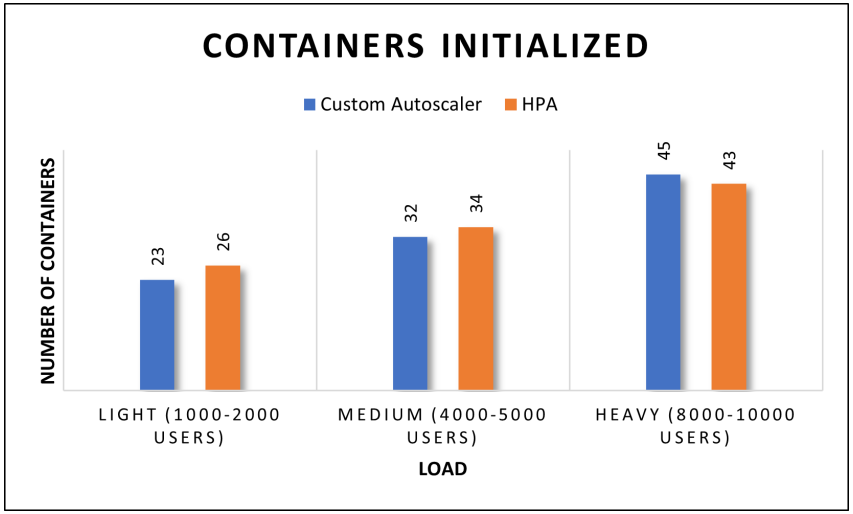


Figure 5: Comparison of Containers initialized.

Experimental Results VIII

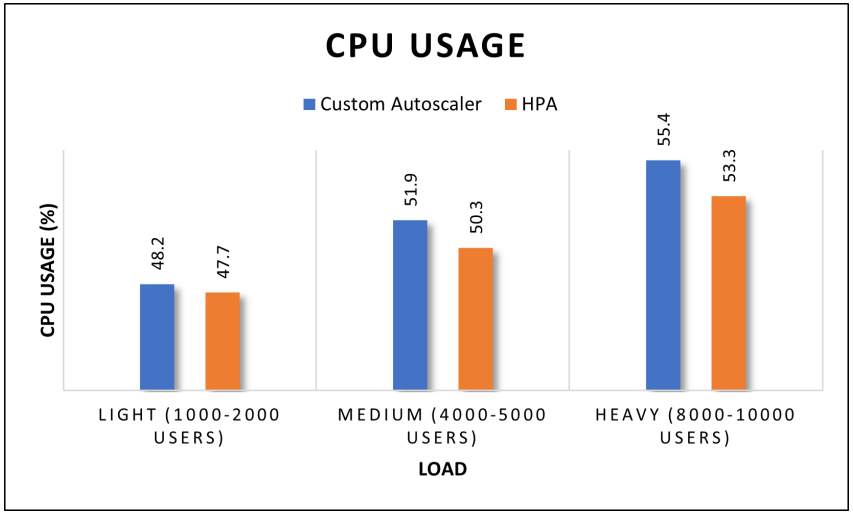
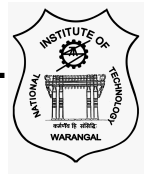


Figure 6: Comparison of CPU usage percentage.

Conclusion and Future Work

- In this project, Prophet model [5] has been used to predict the future workload through the given data set. Results show the prediction accuracy of the model (i.e. minimum prediction error) is better than any other ML model. MAPE (Mean Absolute Percentage Error) and MAE (Mean Absolute Error) have been used to obtain the accuracy of the ML models.
- Prophet model takes into consideration an extra feature called Holidays for predicting the future function invocation traffic.
- Using this holiday matrix we have further fine tune our predictions to account for the irregular fluctuations in load that occur during holidays.
- Using this predicted load information we decided for the required number of containers and keep that number of containers in a pre-warmed state so that we can utilise them when the load arrives thereby solving our problem of cold start delay.
- Lastly, we have compare the results for resources used, latency with Horizontal Pods Autoscaler (HPA) of Kubernetes[6] and found that our custom autoscaler has performed better in terms of reducing the latency.



References I

[1] Yongkang Li et al. “Serverless Computing: State-of-the-Art, Challenges and Opportunities”. In: **IEEE Transactions on Services Computing** (2022), pp. 1–1. DOI: 10.1109/TSC.2022.3166553.

[2] Siddharth Agarwal, Maria A. Rodriguez, and Rajkumar Buyya. “A Reinforcement Learning Approach to Reduce Serverless Function Cold Start Frequency”. In: **2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)**. 2021, pp. 797–803. DOI: 10.1109/CCGrid51090.2021.00097.

[3] Ping-Min Lin and Alex Glikson. “Mitigating cold starts in serverless platforms: A pool-based approach”. In: **arXiv preprint arXiv:1903.12221** (2019).

[4] Available online. **Dataset**. URL: https://drive.google.com/drive/folders/1I-kOyqknS2MVwoIi3fnX7w__PzyN9Wi8?usp=share_link.

References II

[5] Available online. **Prophet ML model**. URL: https://facebook.github.io/prophet/.

[6] Available online. **Kubernetes**. URL: https://kubernetes.io/docs/home/.

Thank You !