Performance Analysis of Metric-Based Scaling in Kubernetes Environments: A Comparative Study of CPU Utilization and Custom Metric Approaches

Jin-Cheol Jung[1]
[1]Dept. of Software Convergence, Kyung-Hee University

bik1111@khu.ac.kr

Abstract

This study compares CPU-based and custom metric-based scaling methods in Kubernetes, showing that custom metrics tailored to application needs can enhance scalability and efficiency. Findings reveal that, at certain scaling thresholds under dynamic network traffic, custom metrics reduce average latency by 85% to 87% compared to CPU-based scaling.

I. INTRODUCTION

Effective resource management is vital in cloud environments to optimize performance and costs. Kubernetes' Horizontal Pod Autoscaler (HPA) scales applications based on metrics like CPU usage, but it has limitations with diverse application needs and complex workloads [1]. These limitations are especially noticeable when specific metrics, such as request latency, are crucial for performance. Custom metric-based scaling, using metrics like network traffic or requests per second, enables more precise resource allocation suited to application demands, such as real-time streaming. This study compares CPU-based and custom metric-based scaling in Kubernetes, showing that custom metrics can greatly enhance performance and efficiency, especially in latency-sensitive applications with dynamic workloads. The goal is to demonstrate how custom metrics provide an optimized solution for complex cloud environments.

II. MAIN DISCUSSION

In this section, we conduct a comparative analysis of Kubernetes scaling strategies using CPU utilization metrics and custom metrics such as Average Network Transmit/Receive Bytes, applying various scaling thresholds for each metric. This analysis evaluates each approach, using metrics such as 99% tail-latency, and CPU usage as performance indicators.

II. I. EXPERIMENTAL ENVIRONMENT

The experiment is conducted using two 8-core CPU servers connected via a 10 Gb Ethernet network, running Ubuntu 20.04 with the Linux kernel 6.0 and Kubernetes 1.31.0. Container communication is facilitated by the Cilium [2] CNI plugin.

The cluster comprises two nodes, each with a distinct role in monitoring CPU utilization during the autoscaling process. The Master Node generates load using the HTTP load testing tool, Vegeta [3] (HTTP load testing tool), while simultaneously measuring 99% tail latency. In contrast, the Worker Node handles incoming requests, performs pod autoscaling, and monitors CPU usage.

Mpstat [4] (Performance monitoring benchmark) is employed as a benchmarking tool to measure CPU utilization, using the sum of System Time, Soft IRQ, and User Time metrics as the basis for assessing CPU resource utilization during autoscaling.

For custom metric-based autoscaling, a Prometheus Adapter [5] is deployed on the Worker Node, serving as an intermediary between Prometheus and Kubernetes HPA. This adapter converts metrics collected from Prometheus into a format compatible with HPA, enabling it to make scaling decisions based on custom metrics.

In this study, network usage is evaluated by dividing the total transmitted and received network bytes on the Worker Node by the current number of containers. The irate function is applied to calculate the per-second rate of change based on the two most recent data points in the time series. This function automatically adjusts for non-monotonic changes, such as counter resets, allowing for accurate measurement of per-container network usage at each autoscaling interval.

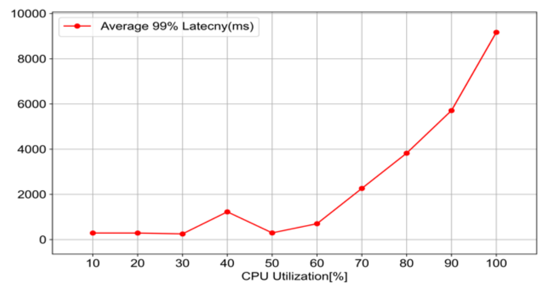
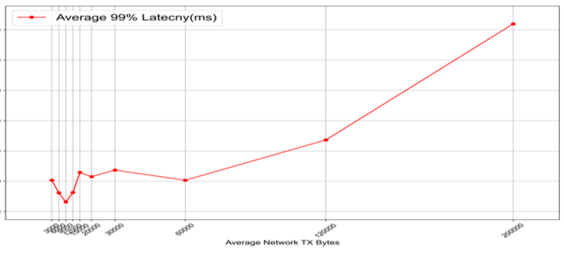
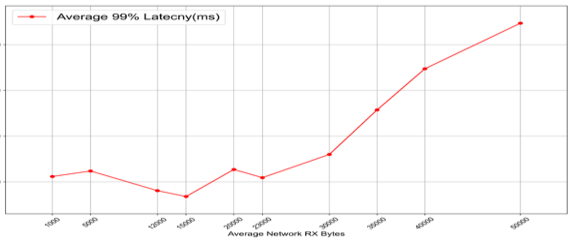


Figure 1. Average 99% Tail-Latency (ms)



(a) Network TX Bytes Metric



(b) Network RX Bytes Metric

Figure 2. Average 99% tail-latency (ms)

Nginx is deployed as the performance benchmark, with HTTP requests increasing from 100 to 104.9 requests per second. This increment triggers the HPA to scale Pods on the Worker Node based on predefined thresholds. To assess scaling performance, CPU utilization thresholds range from 10% to 100% in increments, while Average Network Transmit and Receive Bytes thresholds are set in various increments up to 200,000 bytes for transmit and 50,000 bytes for receive, respectively.

II. II. RESULTS

II. II. I. Average 99% tail-latency

Figure 1 shows the 99th percentile tail latency for CPU utilization, while Figure 2 displays it for Network TX/RX Bytes as custom metrics. These figures present average latency values calculated for each scaling threshold as request rates rise from 100 to 104.9. For performance comparison, we calculate mean latency for each threshold and then assess overall performance by determining the mean, median, and maximum latency values across all thresholds for each metric.

1. **Mean**: Network TX and Network RX metrics demonstrated approximately 7.9 times and 7.5 times faster performance compared to CPU utilization
2. **Median**: Network TX and Network RX metrics showed approximately 4.3 times and 4.1 times faster performance than CPU utilization
3. **Maximum**: Network TX and Network RX metrics exhibited approximately 13.68 times and 16.84 times faster performance relative to CPU utilization

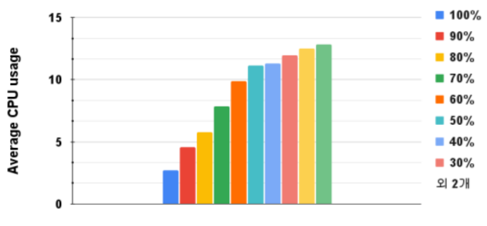
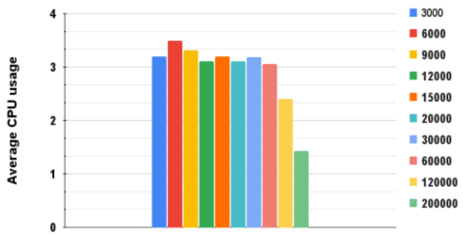
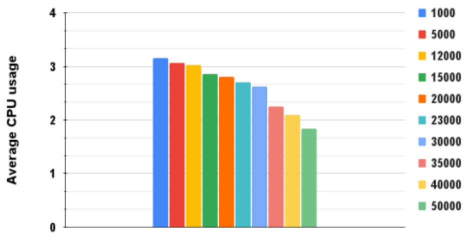


Figure 3. Average CPU Usage



(a) Network TX Bytes Metric



(b) Network RX Bytes Metric

Figure 4. Average CPU usage

II. II. II. Average CPU usage

Figures 3 and 4 present the average CPU usage required to handle request responses at each threshold for each metric. The analysis yields the following results:

1. **Mean:** Network TX and Network RX metrics demonstrate approximately 3.07 times and 3.42 times more efficient CPU usage compared to CPU utilization metrics, respectively
2. **Median**: Network TX and Network RX metrics show around 3.33 times and 3.81 times greater efficiency in CPU usage than CPU utilization metrics
3. **Maximum**: Network TX and Network RX metrics exhibit roughly 3.72 times and 4.08 times higher efficiency in CPU usage compared to CPU utilization metrics

III. CONCLUSION

This study presents a comparative analysis of scaling metrics based on CPU utilization and network traffic in Kubernetes environments, emphasizing the need for efficient scaling using tailored metrics that align with application characteristics. The results indicate that scaling based on custom metrics derived from network traffic provides stable performance and efficient CPU utilization even during traffic surges. Future research will focus on developing more sophisticated scaling metrics and algorithms, considering various workloads and cluster configurations, to enable real-time prediction and resource optimization.

REFERENCES

[1] Anjaly Parayil et al. "Towards Cloud Efficiency with Large-scale Workload Characterization", arXiv:2405.07250v1, 2024

[2] Cilium, https://cilium.io/.

[3] Vegeta, https://github.com/tsenart/vegeta

[4] Mpstat, https://linux.die.net/man/1/mpstat

[5] Prometheus adapter, https://github.com/kubernetes-sigs/prometheus-adapter