Realizing the MAPE-K Cycle: A Comprehensive Evaluation of System Adaptation to Microservices

Yibo Zhang, Jiaxin Ge
University of Waterloo
Email: y549zhan@uwaterloo.ca, j3ge@uwaterloo.ca

***Index Terms*—Self-adaptive systems, MAPE-K loop, resource allocation, planning, execution, dynamic adaptation, system optimization, quality of service (QoS).**

I. Introduction

THE aim of this coursework is to extend prior work in the building of a self-adaptive system through the integration of the Planning and Execution components into the MAPE-K loop[1]. Our effort now shifts toward how one can effectively plan resource allocation strategies and execute adaptations triggered by bottlenecks in performance, based on real-time monitoring and analysis of the system performance. This will contribute to increasing the overall responsiveness and adaptiveness of the system, while keeping it in conformance with the predefined quality requirements on managing traffic fluctuation.

II. Methodology

A. Monitoring Component

This component shall be responsible for collecting real-time system metrics and performance data. In this respect, this will constitute the very foundation of all adaptive actions, as it provides an accurate and timely picture of what the present status of the system is.

Key Aspects of the Monitoring Component

- **Data Collection**: This would fetch critical metrics such as CPU utilization, memory usage, latency, transactions per second, and garbage collection times through APIs like Sysdig. It will routinely fetch the data, say every 5 minutes, to show the health status of the system.
- **Storing Metrics**: The job of the monitoring system is to ensure that the fetched data is stored in some structured format, like JSON files, which can be retrieved and analyzed by other components with ease.
- **Historical Data**: Along with the current data, it also stores the history of the performance of the system, which enables facilities like trend analysis and comparison with the past states.

It always keeps the system updated about its current working status through its performance data, which the Monitoring component does at a high level of detail for making proactive and reactive scaling decisions.

B. Analysis Component

The Analysis component analyzes the raw monitored data generated by the system so as to decide on the actual status of the system. This simply involves checking whether the system is operating within threshold boundaries while meeting the required conditions or whether an adjustment needs to be done.

Key Aspects of the Analysis Component

- **Threshold Evaluation**: The analyzer checks if crucial metrics such as CPU/memory utilization exceed predefined thresholds. Whenever the metrics are out of the acceptable bounds, that triggers the need for an adaptation.
- **Utility Scoring**: Given weights on metrics such as CPU, memory, latency, and TPS, the analyzer sets utility scores associated with the performance of a system. The system automatically maximizes overall utility by optimizing these factors.
- **Adaptation Decision**: The analyzer, taking as input utility scores and threshold breaches, decides how the system should scale up or down to cope with the workload; for instance, CPU, memory, or replicas.

Analysis is a crucial part in this process to appraise the health status of a system and reach a preliminary decision on the scaling that should be performed or resources adjusted to sustain performance.

C. Planning Component

Planning follows from the Analysis stage by developing adaptation strategies that achieve near-optimal performance of the system. The resource allocation plans with detailed adjustments of resources due to changing demands will also be designed.

Key Aspects of the Planning Component

- **Strategy Selection**: It will consider a set of scaling strategies based on their utility scores and choose one that would yield maximum resource efficiency while meeting the performance objectives.
- **Resource Allocation Plans**: This lists plans of identifying the changes to be effected in the system, such as upgrading CPU or memory, increasing or decreasing the number of replicas.
- **System Constraints**: A planner ensures the proposed adaptation respects system constraints and does not introduce new bottlenecks or instabilities.

It ensures system changes are made for long-term performance goals without over-reacting on short-term fluctuations.

TABLE I: Dynamic Resource Scaling Strategies

Condition	Threshold/Metric	Action	Resource Adjustment
CPU Adjustment			
High CPU Utilization	`cpu_util > cpu_upper_threshold`	Increase CPU	`current_config["cpu"] += 100`
Low CPU Utilization	`cpu_util < cpu_lower_threshold and current_config["cpu"] > min_cpu`	Decrease CPU	`current_config["cpu"] -= 100`
Memory Adjustment			
High Memory Utilization	`mem_util > mem_upper_threshold`	Increase Memory	`current_config["memory"] += 256`
Low Memory Utilization	`mem_util < mem_lower_threshold and current_config["memory"] > min_memory`	Decrease Memory	`current_config["memory"] -= 256`
Replica Adjustment			
High CPU and Memory Utilization	`cpu_util > cpu_upper_threshold and mem_util > mem_upper_threshold and current_config["replica"] < max_replicas`	Increase Replicas	`current_config["replica"] += 1`
Low CPU and Memory Utilization	`cpu_util < cpu_lower_threshold and mem_util < mem_lower_threshold and current_config["replica"] > min_replicas`	Decrease Replicas	`current_config["replica"] -= 1`

D. Execution Component

The Execution component shall perform the implementation of resource allocation plans defined by the Planning component. Thus, it has to translate the plans into actual changes in the infrastructure of the system.

Key Aspects of the Execution Component

- **Resource Adjustment**: The executor shall execute the actions involving scale up or down of the replicas, CPU and memory allocations, and garbage collection tuning.
- **Feedback Loop**: The system will track the effects of changes after their execution in real time to ascertain whether adaptations have the intended performance impact.
- **Error Handling**: In case of failure of execution, the system may either revert the changes or apply fallback strategies that guarantee stability.

III. Results

To evaluate the effectiveness of the integrated MAPE-K loop, we conducted several experiments simulating varying workloads. The primary objective was to assess how well the Planning and Execution components enhance system responsiveness compared to the Monitoring and Analysis alone.

We established low, medium, and high workload scenarios, similar to the previous report. Each scenario triggered different adaptations based on the feedback from the Monitoring and Analysis components.

In this section, we will delve into the system's detailed evaluation, focusing on a comparison of performance metrics and demonstrating various adaptation strategies under different conditions, changing CPU allocation, memory usage, and the number of pods. We measure using metrics that were developed in our first assignment, with important performance metrics such as latency, throughput, CPU utilization, and memory consumption, to make the analysis complete. From these metrics, we evaluate the efficiency of the adaptation strategies in place to deal with three kinds of workload scenarios: low, medium, and high loads. More importantly, we demonstrate how the integrated MAPE-K loop dynamically adjusts, based on real-time feedback, to optimize performance of the system; this is important in pointing out the need for continuous monitoring and fine-tuning of the parameters in a system to achieve optimum performance against changing operational conditions.

The results from Figure 3, when there is no load, show how system adaptations have affected various important services. It does so by comparing their performance before and after adjustments were made via the MAPE-K loop. Services like acmeair-mainservice and acmeair-authservice, prior to adaptation, showed very poor performance regarding resource utilization and high latencies. These services needed adaptations; the reductions in CPU allocation from 500 to 400 units were necessary for balancing resource consumption with the minimum load.

After adaptation, vast improvements could be visibly noticed: drastically fallen latencies, an acmeair-mainservice dropping to about 6 million milliseconds, while CPU utilization was optimized. Although some of the services, like acmeair-flightservice, did not necessarily need adaptation as their performance was already quite stable before the adaptation, overall, resource usage smoothed out across the system. The adapted system was more efficient when there was no load; its CPU usage was generally higher, yet more appropriate to the low demand of the system. This further facilitated smoother operation with fewer resource bottlenecks, hence justifying self-optimizing capability for the MAPE-K loop under low workloads.

Fig. 1: Comparison of services running performance before and after adaptation without load.

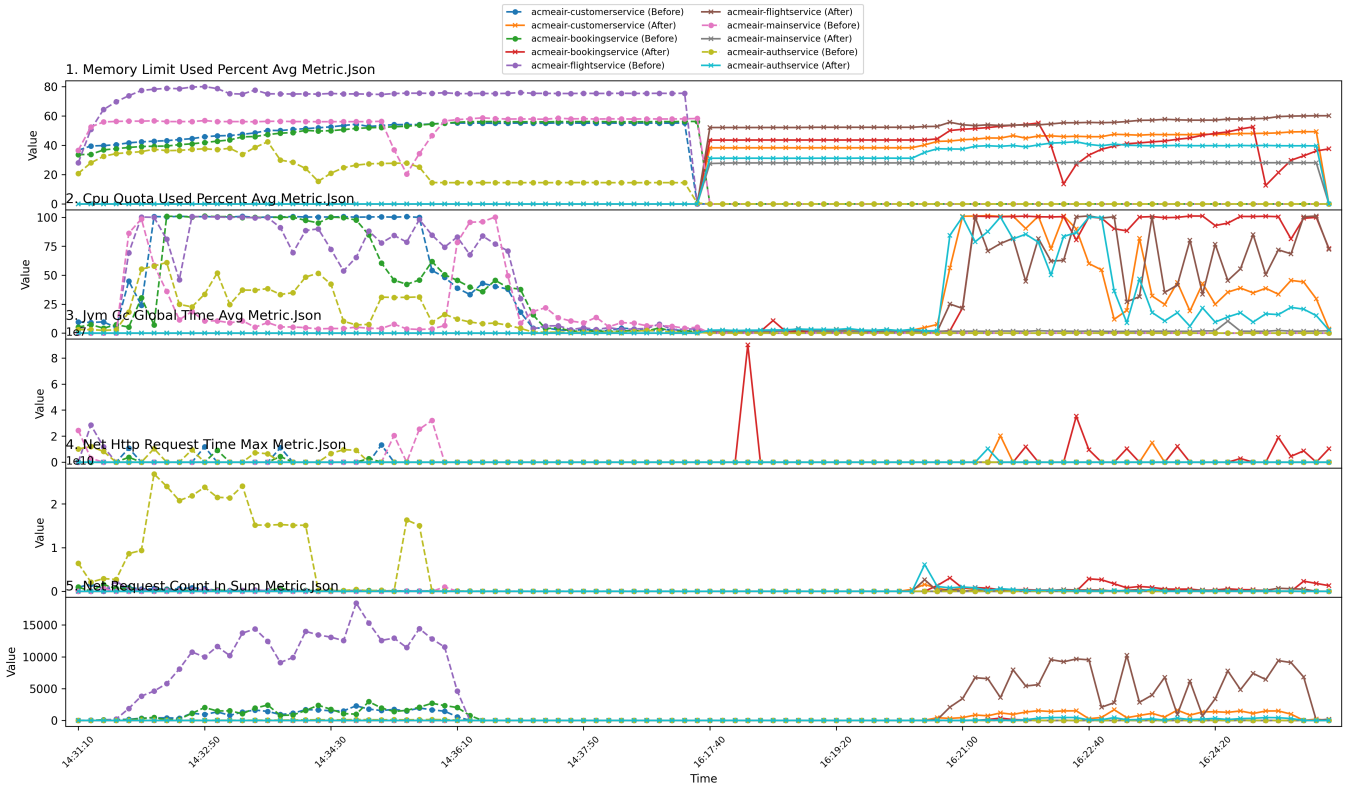

Fig. 2: Comparison of services running performance before and after adaptation with high load.

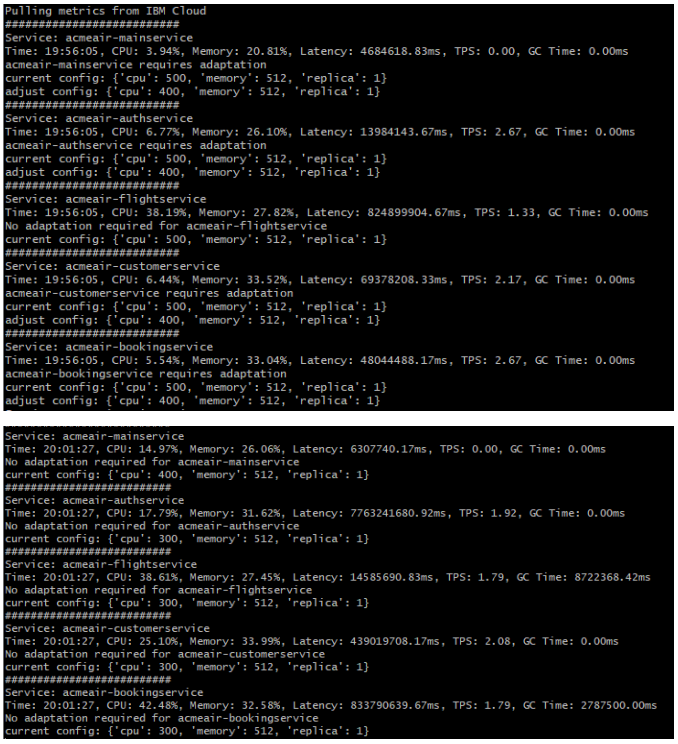

Fig. 3: Command output of Acme Air system before and after adaptation without load.

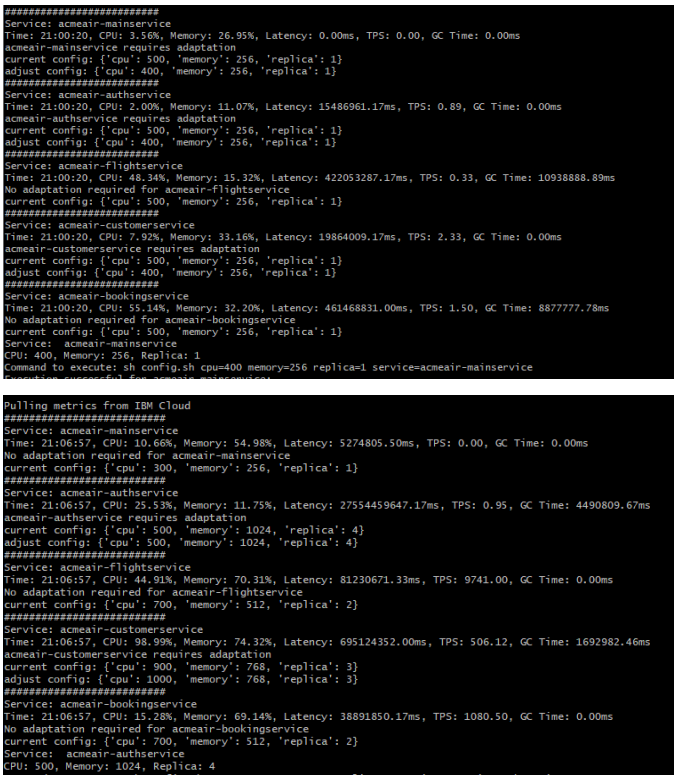

Fig. 4: Command output of Acme Air system before and after adaptation load.

Depending on the load in Figure 4, the system had obvious improvements in performance after adaptation. For example, before adaptation, the service acmeair-mainservice and acmeair-authservice had a low CPU usage of 3.56% and 2.00%, respectively, with very high latency. In relation to this, after adaptation, acmeair-mainservice increased in CPU utilization to 10.66%, entailing memory adjustments and replica changes with instances of drastic reduction of latency due to proper readjustment of the resources.

On the other hand, acmeair-authservice showed increased CPU and memory consumption with variations in replica numbers but remained at high latency; hence, adaptations were beneficial but not sufficient for this service. These types of results show that it is necessary to constantly monitor and refine the adaptation strategy in pursuit of optimal performance for different load scenarios.

Figures 1-2 illustrate the performance metrics before and after the integration of the Planner and Executor components, showcasing improvements in latency and throughput during varying load conditions.

IV. Conclusion

The integration of the Planning and Execution components into the MAPE-K loop significantly enhances the system's adaptability and responsiveness to varying workloads. By effectively translating performance analysis into actionable strategies, the system can dynamically allocate resources in real-time, thereby improving quality of service and maintaining stability under fluctuating traffic conditions[2].

This complete MAPE-K cycle provides a solution not only to the short-term performance challenges but also allows continuous learning and adaptation based on historical data. Additional work could be conducted for further refinement of planning algorithms and execution methods to optimize self-adaptation capabilities of the system.

References

[1] M. Lee, "Optimizing Quality of Service in Cloud Environments," *Journal of Computer Networks and Communications*, vol. 2021, Article ID 7391503, 2021.

[2] R. Erdei and L. Toka, "Minimizing Resource Allocation for Cloud-Native Microservices," *Journal of Network and Systems Management*, vol. 31, no. 1, pp. 35, 2023. Available: https://doi.org/10.1007/s10922-023-09726-3