ECE750-T37 Assignment3 Report G13

1 Planning implementation

The implementation of the planner is straightforward as it relies solely on a rule-based approach. The underlying logic is quite simple: among all the available plans, we select the one with the highest utility, and then pass this optimal plan to the executor.

2 Executing implementation

To execute the task of modifying the configurations of our deployments and services, we've opted to modify the YAML file associated with the desired changes. We developed a shell script named "config.sh" to facilitate these changes. For instance, when adjusting the CPU and memory settings, we can invoke the script with the following command:

$$./config.sh \quad cpu = 500 \quad memory = 512 \quad replica = 2 \quad acmeair\text{-}bookingservice$$

When executed, this command parses the CPU, memory, and replica values that need to be applied and utilizes the "sed" command to modify the corresponding YAML file with the new configuration settings. Subsequently, the shell script runs the command "oc apply -f $file" to enact these changes.

3 Completing the adaptation loop

Besides **Planner and Executor**, the complete adaptation loop comprises the following integral components:

3.1 Driver

The driver serves as the primary program responsible for orchestrating the entire adaptation loop. It initializes the monitor, analyzer, planner, executor, and defines the core workflow of the adaptation loop.

3.2 Monitor

Given specified metrics, the monitor sends requests to a remote server and downloads data into the dataset folder. The monitor fetches the latest 5 minutes of data, in alignment with the adaptation loop interval. This ensures real-time monitoring and timely adjustments when necessary.

3.3 Analyzer

After data retrieval, the analyzer calculates metrics such as average CPU usage, memory usage, response time, and request numbers for selected microservices. The analyzer assesses these metrics against predefined quality requirements, and if any quality requirement is not met, the analyzer initiates an adaptation process. For each scenario, the analyzer generates different adaptation options. For example, if the response time exceeds 2e7 microseconds, two adaptation plans are proposed: **Plan 1:** increase memory by 256, and **Plan 2:** increase the pod number by 1. Before implementing these options, feasibility is assessed, taking into account limited available resources (e.g. $memory \leq 1024, replica \leq 3$). Following feasibility testing, the analyzer then predicts future CPU, memory, and response time using precomputed data combinations. These precomputed data encompasses 81 groups of combinations. Table 1 outlines a truncated part of the experiment. Lastly, the utility is computed using the predicted data, and the configuration data along with the calculated utility are encapsulated in one object, which is sent to the planner for further action.

3.4 Post-Execution

After one cycle of MAPE-K loop, the program enters a 5-minute sleep phase, awaiting the next adaptation cycle. This comprehensive adaptation loop ensures that the system continually adapts to changing conditions and maintains the desired quality requirements.

Table 1: Testing Customer Services under High Workload with Different Pod Configurations

Microservice	Memory	Replica	Workload	CPU Usage(%)	Memory Usage(%)	Response Time(us)
customerservice	512MB	1	high	54	77	5.82e7
customerservice	512MB	2	high	30	67	1.65e7
customerservice	512MB	3	high	23	56	1.02e7

4 Evaluation of adaptation strategies

We evaluated our adaptation strategies using a control variable approach. Firstly, we used JMeter in a Docker environment to generate varying workloads for the AcmeAir server. The workloads gradually increased from 0 to 50 threads. Next, we captured the microservice data under default settings (1 pod, 512MB RAM), establishing a baseline for low, medium, and high workloads. We then deployed the self-adaptive system and tested its performance under the same workload conditions. This methodology allowed us to assess the effects of our adaptation strategies and draw conclusions from the observed data. As depicted in Figure 1, the response time of the three microservices exhibited an upward trend with increasing load, reaching a maximum of 6 seconds. However, after deploying the self-adaptive system, the memory automatically increased, resulting in a significant reduction in server response time to less than 1 second. Likewise, as illustrated in Figure 2, when the server was subjected to a high workload, the memory configuration of the microservice was increased from 512MB to 768MB. This adjustment effectively decreased the percentage of memory usage, improving the overall system performance.

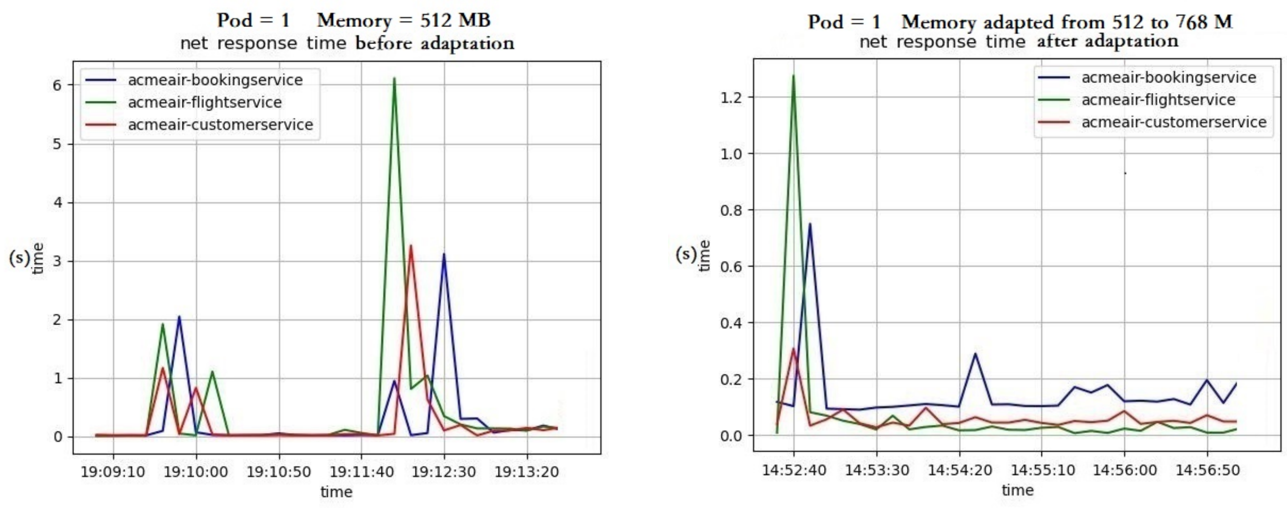


Figure 1: Response time before and after adaptation

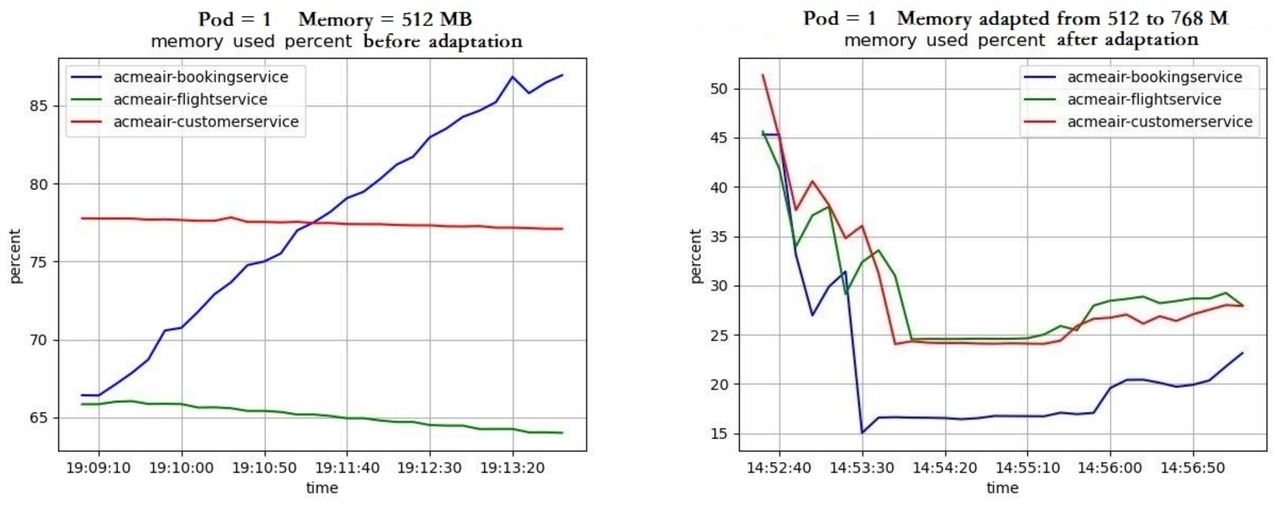


Figure 2: Memory used percentage before and after adaptation

5 Key Insights for Successful Adaptation System Development

We've gained valuable insights from the design and implementation of this self-adaptive system. Firstly, meticulous designing the MAPE-K loop is crucial to ensure smooth deployment and execution. We learned the importance of identifying uncertainties, defining clear quality requirements, and creating comprehensive components for each phase of the MAPE-K framework. Secondly, continuous evaluation is vital to assess the system's performance and adaptability. Regularly monitoring key metrics and evaluating the effectiveness of self-adaptation rules helped us make informed decisions for system improvement. Lastly, leveraging tools like OpenShift and Docker streamlined the deployment process, facilitating scalability and efficient resource management. Overall, this experience highlighted the significance of careful designing, continual evaluation, and leveraging appropriate tools for successful system deployment.

6 Steps to Run the Driver Program

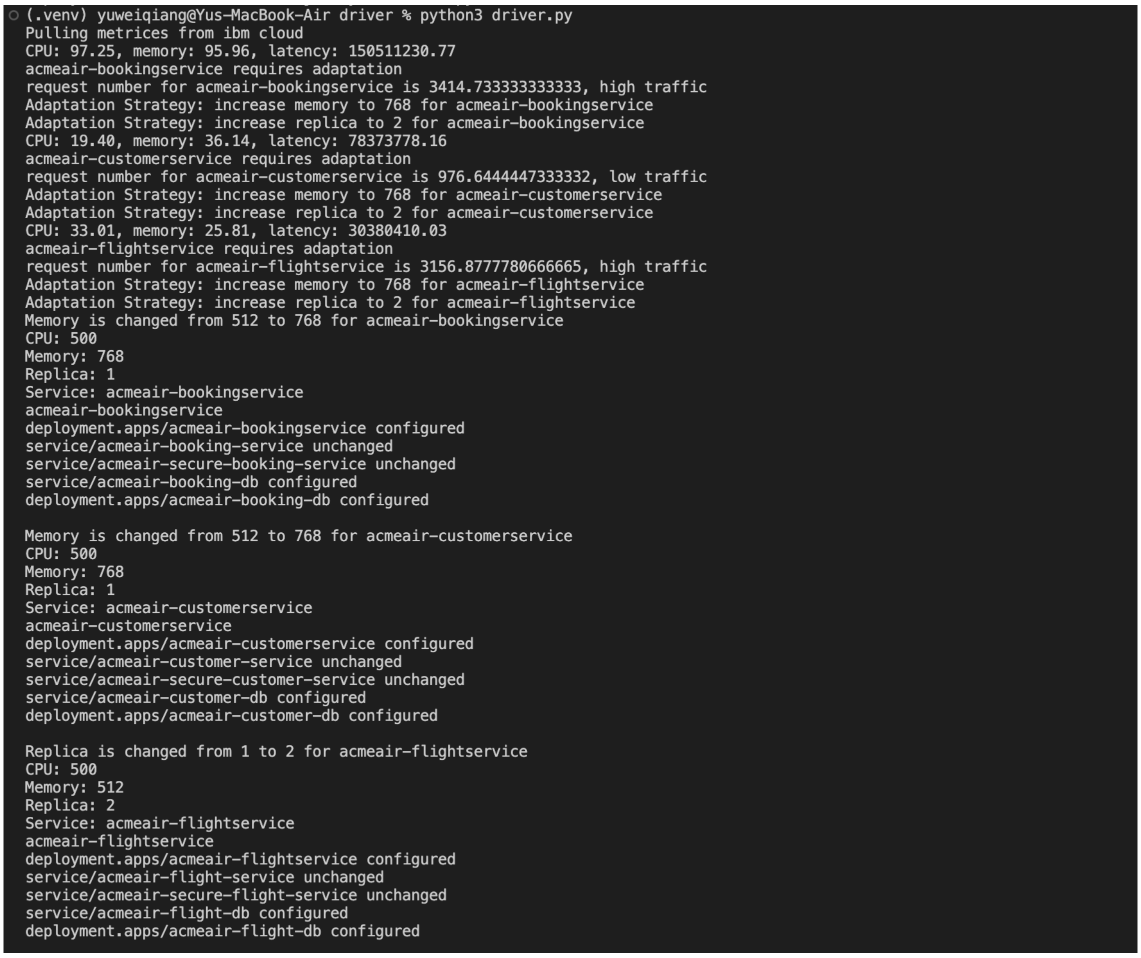
Group 13's MAPE Loop Driver Program on Mac & Linux
STEP 1: Create Virtual Environment and Install Packages

```
$ cd driver
$ python3 -m venv myenv
$ source myenv/bin/activate
$ pip3 install numpy pandas sdcclient
$ sudo chmod 777 config.sh
```

STEP 2: Executing the Driver Program

```
$ oc login ...
$ oc project "acmeair-g13"
$ python3 driver.py
```

```
Please be aware that shell scripts may exhibit different behavior on various operating systems.
Ensure that you use the appropriate ‘config.sh’ file when running the program. If you are using
a Mac machine, the default ‘config.sh’ is suitable. However, if you are using a Ubuntu machine,
please replace the existing ‘config.sh’ with ‘config1.sh’."
```



```
○ (.venv) yuweiqiang@Yus-MacBook-Air driver % python3 driver.py
  Pulling metrices from ibm cloud
  CPU: 97.25, memory: 95.96, latency: 150511230.77
  acmeair-bookingservice requires adaptation
  request number for acmeair-bookingservice is 3414.733333333333, high traffic
  Adaptation Strategy: increase memory to 768 for acmeair-bookingservice
  Adaptation Strategy: increase replica to 2 for acmeair-bookingservice
  CPU: 19.40, memory: 36.14, latency: 78373778.16
  acmeair-customerservice requires adaptation
  request number for acmeair-customerservice is 976.6444447333332, low traffic
  Adaptation Strategy: increase memory to 768 for acmeair-customerservice
  Adaptation Strategy: increase replica to 2 for acmeair-customerservice
  CPU: 33.01, memory: 25.81, latency: 30380410.03
  acmeair-flightservice requires adaptation
  request number for acmeair-flightservice is 3156.8777780666665, high traffic
  Adaptation Strategy: increase memory to 768 for acmeair-flightservice
  Adaptation Strategy: increase replica to 2 for acmeair-flightservice
  Memory is changed from 512 to 768 for acmeair-bookingservice
  CPU: 500
  Memory: 768
  Replica: 1
  Service: acmeair-bookingservice
  acmeair-bookingservice
  deployment.apps/acmeair-bookingservice configured
  service/acmeair-booking-service unchanged
  service/acmeair-secure-booking-service unchanged
  service/acmeair-booking-db configured
  deployment.apps/acmeair-booking-db configured

  Memory is changed from 512 to 768 for acmeair-customerservice
  CPU: 500
  Memory: 768
  Replica: 1
  Service: acmeair-customerservice
  acmeair-customerservice
  deployment.apps/acmeair-customerservice configured
  service/acmeair-customer-service unchanged
  service/acmeair-secure-customer-service unchanged
  service/acmeair-customer-db configured
  deployment.apps/acmeair-customer-db configured

  Replica is changed from 1 to 2 for acmeair-flightservice
  CPU: 500
  Memory: 512
  Replica: 2
  Service: acmeair-flightservice
  acmeair-flightservice
  deployment.apps/acmeair-flightservice configured
  service/acmeair-flight-service unchanged
  service/acmeair-secure-flight-service unchanged
  service/acmeair-flight-db configured
  deployment.apps/acmeair-flight-db configured
```

Figure 3: Sample Output Produced by the Driver Program