Reinforcement Learning using Q-Learning for SWIM Adaptation

Aishwarya Naik

Department of Engineering, Computer Science and Mathematics
University of L'Aquila
L'Aquila, Italy
aishwarya.naik@student.univaq.it

***Abstract*— A self-adaptive example that mimics a web application is called Simulator for Web Infrastructure and Management (SWIM). SWIM can be utilized as target system with an adaptation manager developed as a simulation module or an external adaptation manager communicating with it via its TCP-based interface. Web applications that are operating on actual web servers are frequently used as target systems in research on self-adaptive systems. There are three issues with this strategy. We provide SWIM, an example that replicates a web application, in order to address these problems. Primarily, it is difficult and/or expensive to deploy these technologies. Furthermore, because of uncontrollable elements, run-time conditions can't really be precisely duplicated to evaluate various adaptation strategies. Lastly, conducting experiments takes time. The major purpose of this initiative is to implement an external adaptation manager for SWIM to improve its performance using Reinforcement Learning Methods.**

***Keywords*— SWIM Adaptation Manager, Reinforcement Learning, Q-Learning, Docker, gym library**

I. Implementation Overview

Self-Adaptive systems are typically set up to use many web servers in order for the self-adaptation manager can alter the number of active servers in response to changing application traffic. The need for a distributed system of real or virtual servers to distribute the web servers makes it more difficult to implement these experimental solutions. Another issue with executing the software onto real web servers would be that due to a number of unpredictable factors, such as the servers' processing speed, background operations, and network delays, the run-time parameters cannot be precisely recreated to evaluate various self-adaptation methodologies. The time commitment involved in conducting experiments is still another issue.

Here it is offered that the Simulator for Web Infrastructure and Management (SWIM), an example that replicates a web application like Znn.com or RUBiS, in order to solve these difficulties [1]. No matter how many of servers needed for an experiment, SWIM mimics the handling of requests across several servers while only requiring a single process to execute on a single machine. There are two ways the adaption manager and SWIM may communicate with one another. SWIM can be used as a target system in the first mode, communicating including an external adaptation manager through its TCP-based interface. This makes it simple to integrate the adaption manager with SWIM regardless of what language it was created in.

A web server tier, which processes client requests using computers, and a database tier make up this type of program. The web tier supports numerous servers through the deployment of a load balance. The load balance divides up the website requests among the web servers. The web server handling the request contacts the database layer when a client sends a request using a browser to obtain the data required to produce the page with updates. We want the system to be capable to self-adapt in order to comply with this changing environment because the proposal of arrival rate, which influences the workload upon that system, alters over time.

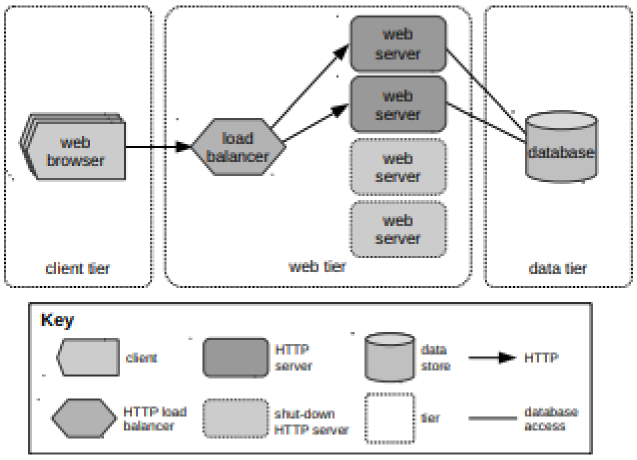


Fig. 1 Simulated Web Application Architecture. Source [1]

By playing back request traces that have been recorded in files, system traffic is mimicked. A single floating-point value that indicates how long it will take (in seconds) before submitting the next request appears on one line per request in a trace file. The WorldCup '98 trace repository and the ClarkNet traces, which have been originally captured from actual websites, are also included in SWIM.

A self-adaptive system can adjust its behaviour in response to changes in its surroundings. As a result, a self-adaptive system needs to constantly watch for context changes and respond appropriately.

II. Experiment and Evaluation

1) *SWIM with external adapter:*

There are three phases involved in conducting an experiment with SWIM. The simulation must first be set up, with the duration, input trace, and service time for handling requests among other parameters. Second, the tests must be carried out, and how they are carried out depends primarily on whether the adaptation manager is an external software or a simulation module. The obtained results are then examined.

The revenue increases when more optional content is served and the response time requirement is met, while operating costs are generally proportionate to the number of servers deployed. The adaption manager must now manage a trade-off as a result. SWIM offers the monitoring data required to take these elements into account when making an adaptation choice and to compare different adaptation strategies using utility functions that take these factors into account.

The following is the logic behind this adaption manager. If possible, a server is added if the recorded average response time exceeds the threshold; otherwise, the dimmer is turned down. If there are many active servers with spare capacity and the response time falls below the threshold, the dimmer is increased. If the dimmer has reached its maximum, a server is removed if that is possible. In this instance, the `MacroTactic` that the `evaluate()` method produces implements a complex pattern for tactics.

It contains the interaction with SWIM through its TCP-based interface.

A Swim Client class. The connection to SWIM is opened by a method in this class, and once it is established, the other methods use this connection to deliver orders to SWIM and wait for its answer. This adaptation manager must implement the entire self-adaptation loop, unlike the adaptation manager that was developed as a simulation model. The monitor is implemented in the first section of the loop's code utilizing the interface's probing commands to gather information about the system, with the model being the local variables used to store the results.

In order to run a simulation, you must choose which run numbers to use and which named settings from the configuration file to utilize. There is a script called run.sh available in each simulation directory, and it takes the following arguments.

```
./run.sh config [run-number(s)|all [ini-file]]
```

A range of run numbers or a list of comma-separated run numbers without spaces can be used as the optional input to specify which run to conduct (e.g., 0-4). The third input is optional, and it can be used to indicate a different.ini file (swim.ini and swim sa.ini are the defaults for swim and swim_sa, respectively).

2) *Docker:*

A technology known as containerization organizes system libraries, dependencies, and applications into a container-like structure. Applications that have been developed and arranged can be deployed and executed in a container. Docker, a platform, ensures that the application functions in all environments. Additionally, it automates the deployment of apps into containers. The container environment in which the programs are run and virtualized is supplemented by Docker with an additional layer of deployment engine. Docker contributes to the creation of a rapid and light environment that can run code effectively. Docker's four key components are its containers, clients/servers, images, and engine [15].

Docker Containers: Docker images produce Docker Containers. Every kit needed for the application is to be stored by the container in order to operate it in a constrained manner. The service requirements for the application or program can be used to construct the container images. Let's say the docker file should append an application that uses the Ubuntu operating system and the Nginx server. A container with an Ubuntu OS image that includes the Nginx server is generated and launched using the command "docker run."

The benchmark is intended to assess how quickly Docker container applications start up on physical and virtual machines. The time required to begin an app in a Docker container is known as the start-up time. We neglected the network latency, disk I/O, and CPU performance during our experimentation. The httpd2.4 standard image with the simplest hosting of web pages is used to launch a Docker container. Using a standard httpd image serves the objective of upholding a uniform benchmark environment.

Component performance is evaluated using low-level benchmarks, such as CPU clock and memory cycles. High-level benchmarks are used to assess how well the operating system and hardware drivers perform. Several benchmarking tools, including Lbench, N bench, and UNIX bench, are used to determine the container's efficiency. Lbench tests a system's capacity to transport data between the processor, cache, memory, and disk while evaluating performance congestion in a variety of system applications. The CPU, GPU, and RAM are also tested using the N bench, which evaluates the results in terms of cache, memory, integers, and floating points. In each chapter of Unix Bench, a distinct component of operating system functionality is highlighted, such as process spawning, inter-process communication (IPC), and file system.

According to the findings, virtual machines are slower than bare metal, which impacts the speed at which Docker containers start. In bare metal, Docker containers run about 50% better than virtual machines. The architecture of the virtual machine is to blame for this decline in performance. By operating on imitated hardware, the virtual machine adds more layers than bare metal. When dealing with fewer Docker containers, the virtual machine does not exhibit a lack of performance; nevertheless, if the number is performance declines as they get worse. The Docker containers must be installed on baremetal computers in order to fully benefit from their features.

The authors in a study, provide a summary of the CPU performance, Memory throughput, disk I/O, and operation speed measurement of virtual machines and Docker containers. The implementation of Docker containers in the HPC Cluster was the authors' primary concern. The authors discuss various implementation strategies, including how to use LNPACK and

BLAS while selecting a container model. Authors of a study talk about lightweight virtualization techniques that address container and unikernel problems. The paper also describes the statistical analysis of the ANOVA test and a post-hoc comparison of the gathered data using the Tukey method. The author also goes over the various benchmarking programs that were used to compare containers with unikernel.

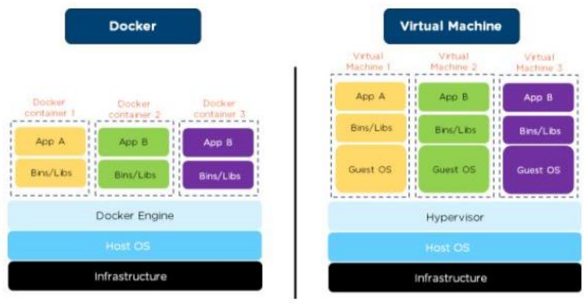


Fig. 2 Docker v/s Virtual Machine

3) *Reinforcement Learning, Q-Learning and gym Library:*

The science of leveraging experiences to make the best judgments is known as reinforcement learning. When broken down, the Reinforcement Learning procedure consists of the following easy steps:

- Monitoring the environment
- Making a decision and applying a strategy
- Acting appropriately
- Getting a reward or a punishment
- Using the lessons learned to improve our approach
- Iterate until you find the best course of action.

The incentives and/or punishments are chosen and their size in accordance with the fact that the agent is reward-motivated and will learn how to drive the cab through trial-and-error encounters in the environment. Here are some things to think about:

- Because this behaviour is highly desirable, the agent should be given a high positive reward for a successful drop-off.
- If an agent tries to drop off a passenger in the wrong place, they should be disciplined.
- For failing to reach the destination at the end of each time-step, the agent should receive a small negative incentive. We consider this to be "somewhat" bad because we would rather have our agent arrive late than have them rush to be there.

The agent in Reinforcement Learning meets a state and responds in accordance with that state.

The State Space is made up of any scenario that our cab could encounter. The state should have relevant data that the agent needs to take the appropriate action.

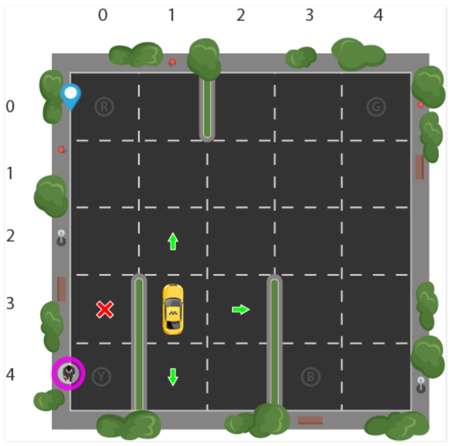
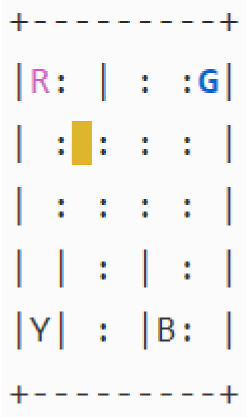


Fig. 3 Exemplar illustration of Reinforcement Learning

This setting is already set up in OpenAI Gym. Gym offers many gaming settings that we can integrate with our code and test an agent in. The library manages the API that gives our agent access to all the data it needs, including scores, available actions, and current status. For our agent, we only need to concentrate on the algorithm part.

All of the information discussed above was taken from the Gym environment called Taxi-V2, which is what we'll be using. All of the behaviours, goals, and rewards are the same.



```
+---------+
|R: | : :G|
| : : : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
```

Fig. 4 OpenAI Gym

An initial Reward table called "P" is also established so when Taxi environment is created. It can be conceived as a *state x action* matrix, with the quantity of states represented by rows and the number of actions by columns.

Note a couple things:

- The operations (south, north, east, west, pickup, and drop-off) that the taxi can carry out in the illustration at our current location are denoted by the numbers 0 through 5.
- Probability has always been 1.0 in this environment.
- The state we would be in following the action listed at this index of the dict is known as the next-state.
- In this particular state, all movement acts receive a -1 reward while pickup/drop-off actions have a -10 reward. If the taxi has a passenger and is at the correct location, we would see a reward of $20 at the drop-off location (5)

- When a passenger has been successfully dropped off in the intended place, the word done is used to signal that.

Q-learning enables the agent to use the rewards from the environment to gradually learn the best course of action in a specific condition.

The "quality" of an action being taken from a specific state is indicated by a Q-value for that state-activity pair. A higher Q-value indicates a higher likelihood of receiving larger rewards.

The Q-values are initially set to an arbitrary value, and as the agent interacts with the environment and performs various behaviours that result in a variety of rewards, the Q-values are updated using the formula:

$$Q(state, action) \leftarrow (1-\alpha)Q(state, action) + \alpha\left(reward + \gamma \max_a Q(next\ state, all\ actions)\right)$$

Where:

- Similar to supervised learning environments, the learning rate, or (α), indicates how frequently our Q-values are changed. $0 < \alpha \leq 1$
- The discount factor, or (γ), which influences how much weight to attach to future benefits. In contrast, a discount factor of 0 forces our agent to just think about immediate reward, making it greedy. A high value for the discount factor (near to 1) represents the long-term effective award. $0 \leq \gamma \leq 1$

The Q-table is a matrix where each row (500) represents a state and each column represents an action. After being set to 0 initially, values are adjusted following training. Even though the Q-dimensions table's match those of the reward table, it serves an entirely different function.

Comparing Q-Learning with no Reinforcement agent we have the following assessments. We assess our representatives using the following metrics:

- Per episode average amount of penalties: The efficiency of our agent increases with the decrease in the number. This measure should ideally be 0 or extremely near to zero.
- Average amount of timesteps per trip: We want fewer timesteps overall because we want our agent to travel the shortest distance possible to get to his or her destination.
- Average benefits per action: The agent is acting morally if the payoff is larger. Because of this, choosing rewards is an essential component of reinforcement learning.

A. *Experimental Setup*

The platform is a Docker running Linux on server 13.39.20.51 at host 6901. It features the SWIM application processor. A Reinforcement Learning backend is created in Python using "numpy" to create multidimensional arrays and matrices. The learning rate alpha is set to 0.7 initially. This is done so to check the changes in the system as the adaptation module is stimulated only when the response time crosses 0.6. The response time is read from the SWIM machine. The entire number of available actions is how we describe the model efficiency. The initial action is set to 8 which instructs "remove server and increase dimmer by 0.1". When the model efficiency is greater than 80% and 90%, respectively, the learning rate is decreased to 0.001 and ultimately to 0.

The assessment was set up on Docker on which a SWIM container was inherited.

An open-source solution to the reproducibility problems in SE/WE research is the Docker container. Containers are like portable virtual machines that make it possible to create a computing environment with all the required dependencies (such libraries), settings, code, and data in a single package (called image) [11]. A Docker_file, a document that contains all infrastructure setup and commands, contains the procedures required to reach the state in such an image. Containers increase the sharing of scientific findings by addressing the drawbacks of prior approaches (such as open source) and making artifacts from SE/WE research immediately available to reviewers, inquisitive readers, and future researchers.

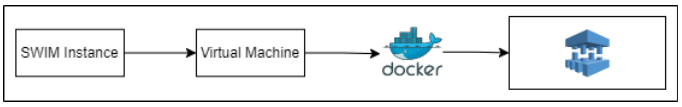
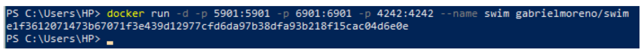
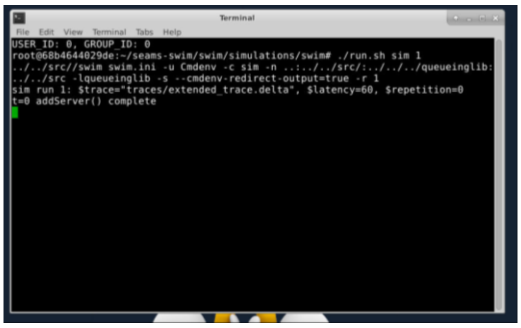


Fig. 5 Containerization of code



```
PS C:\Users\HP> docker run -d -p 5901:5901 -p 6901:6901 -p 4242:4242 --name swim gabrielmoreno/swim
e1f3612071473b67071f3e439d12977cfd6da97b38dfa93b218f15cac04d6e0e
PS C:\Users\HP> _
```

Fig. 6 SWIM container setup using Docker



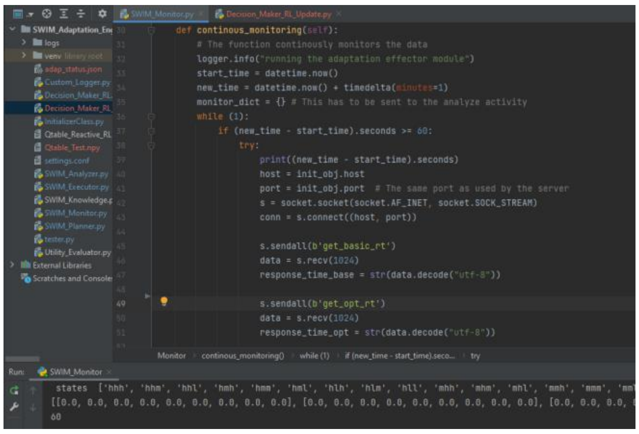
```
USER_ID: 0, GROUP_ID: 0
root@68b4644029de:~/seams-swim/swim/simulations/swim# ./run.sh sim 1
../../src//swim swim.ini -u Cmdenv -c sim -n ..:../../src/:../../../queueinglib:
../../src -lqueueinglib -s --cmdenv-redirect-output=true -r 1
sim run 1: $trace="traces/extended_trace.delta", $latency=60, $repetition=0
t=0 addServer() complete
```

Fig. 7 SWIM setup in Virtual Machine

1) *Adaptation Manager as an External Program:*

The TCP-based interface of SWIM is accessible to adaption managers.

Those are external programs. Since SWIM's protocol is straightforward and text-based, a TELNET client can even be used to communicate with SWIM. By default, SWIM runs on port 6901, 5901 & 4242. When a command is not detected or is executed incorrectly, SWIM responds with a text line beginning with the prefix "`error:`" and an error message. If an execution request, such as "`set_dimmer 0.4`", is given and is successful, SWIM responds "`OK`" SWIM responds with a protocol answer (a numeric or a floating-point number) in a full line if the code is a probing command.



```python
def continous_monitoring(self):
    # The function continously monitors the data
    logger.info("running the adaptation effector module")
    start_time = datetime.now()
    new_time = datetime.now() + timedelta(minutes=1)
    monitor_dict = {} # This has to be sent to the analyze activity
    while (1):
        if (new_time - start_time).seconds >= 60:
            try:
                print((new_time - start_time).seconds)
                host = init_obj.host
                port = init_obj.port  # The same port as used by the server
                s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
                conn = s.connect((host, port))

                s.sendall(b'get_basic_rt')
                data = s.recv(1024)
                response_time_base = str(data.decode("utf-8"))

                s.sendall(b'get_opt_rt')
                data = s.recv(1024)
                response_time_opt = str(data.decode("utf-8"))
```

```
 states  ['hhh', 'hhm', 'hhl', 'hmh', 'hmm', 'hml', 'hlh', 'hlm', 'hll', 'mhh', 'mhm', 'mhl', 'mmh', 'mmm', 'mm
[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 
60
```

Fig. 8 External Adaptation manager for SWIM

B. *Evaluation Candidates*

The control loop, which uses the MAPEK pattern, has elements for I observing the environmental components and the target system, (ii) analyse the results gathered, (iii) planning an adaptation, (iv) and (iv) putting the adaptation into practice in the communication system. A distributed Knowledge component is also used to store data that is needed by other components.

The most important evaluation candidates of this experiment are states, actions, gamma (which is a reference to future rewards), alpha (which is the learning rate), count (which keeps a track of number of adaptations).

1) Design of Reward Function:

The reward function is constructed for these reasons. The performance is a collaborated effect of *nine different actions*. Hence these cannot be evaluated individually. Further the best performance is achieved by multiple configurations. To avoid arbitrary differences, we the epsilon value is set to 0.1. This parameter induces randomness in the system, thus forcing it to try different actions [13].

The main purpose of using reinforcement learning to train the application is that it learns from the previous actions and reward to improvise itself. For the same reason, the action, state and reward for the upcoming iterations are calculated and stores in the qTable in the following way:

*qTable = (1-alpha) * qTable + alpha * (reward + gamma * max(qTable[next_state]))*

C. *Evaluation Metrics*

The adaption model is a type of reinforcement learning called Q-learning, which is dependent on system performance indicators discovered at the termination of each operation simulation.

The Evaluation metrics used in this implementation are response time, arrival rate, dimmer value, server count, number of requests, cumulative utility.

In the below shown results we can see that there is a significant change in utility, response time and number of requests being passed. The number of servers is set to 1 and the dimmer value is constantly between 0.75 and 1.00. The response time as read from the SWIM is uniformly fluctuating between 10.0 and 12.5.

D. *Results*

The configuration-defined average request rate per evaluation period is shown on the top chart. The server charts display the total number of servers (both booting and active servers) with a dashed line and the number of active servers with a solid line. When a server boots, the distinction between the two is visible. The magnitude of the dimmer is displayed on the chart in the centre at various points of time. It need not be this way; in this instance, the adaption manager employs distinct dimmer levels that display as plot points. The response time threshold is represented by a dashed line on the average response time graph.

The results of the experiment can be seen like below. In the Fig.3, the first image shows the graph for the inbuilt SWIM mechanism and its respective trace file. Further from the second picture we can see the changes in the graph based on the executed Reinforcement Learning mechanism and its trace file called “extended_trace.delta”.

From the following outcomes we can see how the graph changes with respect to constant dimmer values and stabilized response time.

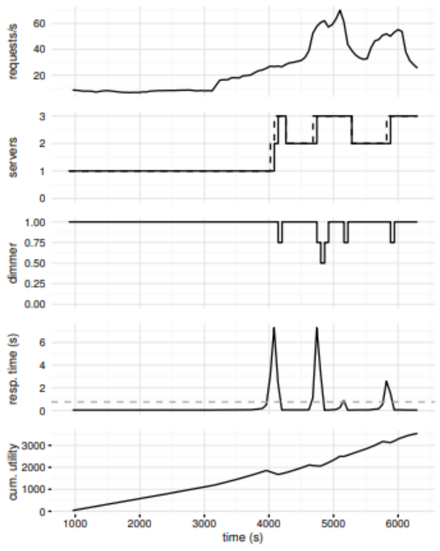


Fig. 9 Default Result for inbuilt adaptation

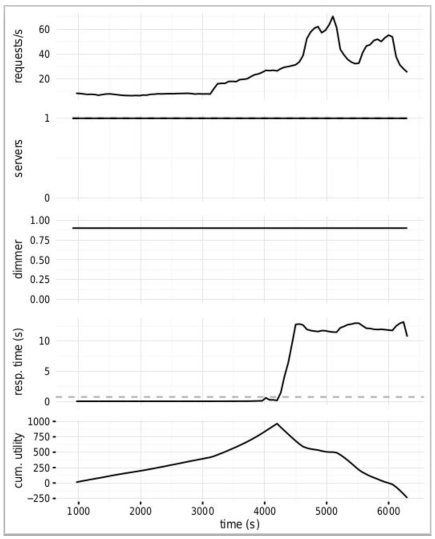
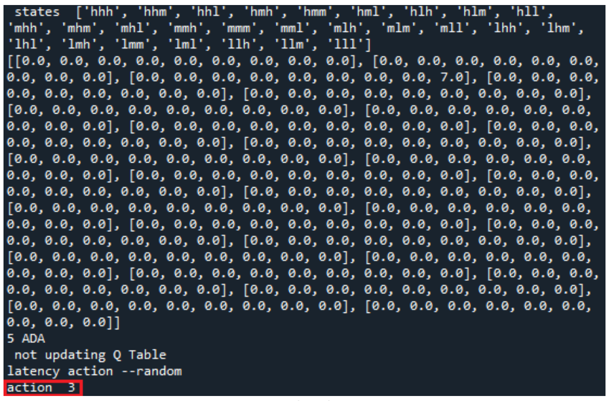


Fig. 10 Results for Zero values loaded



```
 states  ['hhh', 'hhm', 'hhl', 'hmh', 'hmm', 'hml', 'hlh', 'hlm', 'hll',
'mhh', 'mhm', 'mhl', 'mmh', 'mmm', 'mml', 'mlh', 'mlm', 'mll', 'lhh', 'lhm',
'lhl', 'lmh', 'lmm', 'lml', 'llh', 'llm', 'lll']
[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 7.0], [0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0]]
5 ADA
 not updating Q Table
latency action --random
action  3
```

Fig. 11 QTable for first adaptation

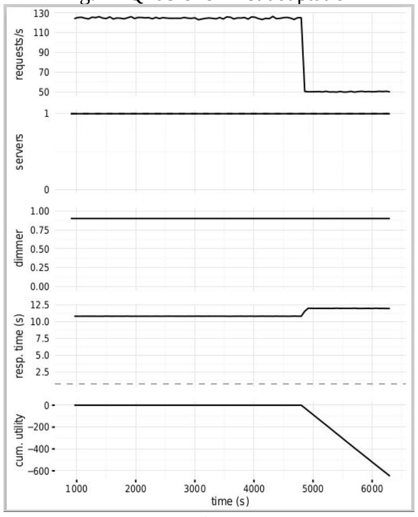
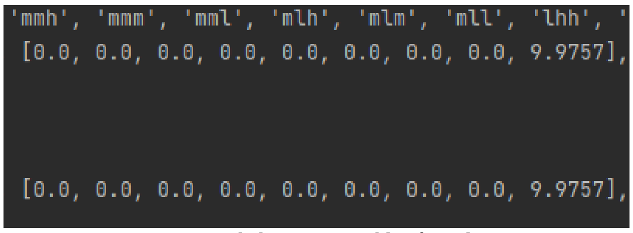


Fig. 12 Result after sixth iteration



```
'mmh', 'mmm', 'mml', 'mlh', 'mlm', 'mll', 'lhh', '
 [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 9.9757],


 [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 9.9757],
```

Fig. 13 Positional change in QTable after adaptation

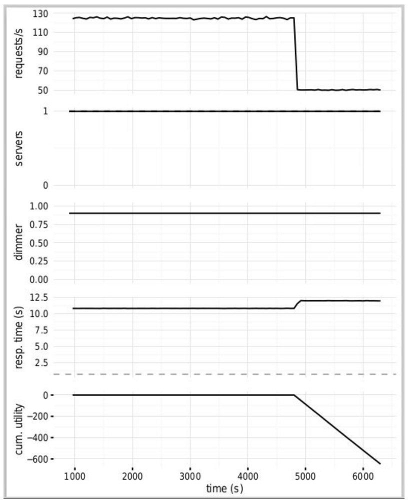


Fig. 14 Result after tenth iteration

III. Discussion

In this demonstration, a type of Reinforcement Learning called Q-Learning is used, which provides an integrated feedback loop, support for multiple languages, and a straightforward scattered and decentralized deployment. The capabilities and usability of the publicly accessible implementation of SWIM is demonstrated.

Prominent space - time reinforcement learning algorithm Q-learning frequently uses lookup tables to explicitly record state variables. Although it has been demonstrated that using this approach will lead to the best outcome, using a function-approximation system is frequently advantageous. The Bellman equation provides the Q-value for a pair of state-action relations.

*self.qTable[state][action] = reward + gamma * max(self.qTable[next_state])*

To some extent random exploration is allowed in two situations. First, if there are numerous potential adaptation actions, existing methods may demonstrate delayed learning because they examine adaptation actions at random. Second, they may examine new adaptive behaviours introduced during evolution fairly late since they are uninformed of system evolution.

IV. Conclusions

In this research, predictive algorithms are implemented for the improvisation of self-adaptive software in terms of run-time performance as well as development complexity. It is demonstrated that the suggested Q-Learning approach works effectively for workloads that are a combination of certain and uncertain as well as fully unknown. Finally, the suggested method produces results that are comparatively improvised versions of the results of the inbuilt mechanisms.

REFERENCES

[1] Moreno, G.A., Schmerl, B. and Garlan, D., 2018, May. Swim: an exemplar for evaluation and comparison of self-adaptation approaches for web applications. In 2018 IEEE/ACM 13th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS) (pp. 137-143). IEEE.

[2] Moreno, G.A., Papadopoulos, A.V., Angelopoulos, K., Cámara, J. and Schmerl, B., 2017, May. Comparing model-based predictive approaches to self-adaptation: CobRA and PLA. In 2017 IEEE/ACM 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS) (pp. 42-53). IEEE.

[3] Cheng, B.H., Giese, H., Inverardi, P., Magee, J., de Lemos, R., Andersson, J., Becker, B., Bencomo, N., Brun, Y., Cukic, B. and Di Marzo Serugendo, G., 2008. 08031--Software engineering for self-adaptive systems: A research road map. In Dagstuhl Seminar Proceedings. Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

[4] Gerostathopoulos, I., Raibulet, C. and Alberts, E., 2022, March. Assessing Self-Adaptation Strategies Using Cost-Benefit Analysis. In 2022 IEEE 19th International Conference on Software Architecture Companion (ICSA-C) (pp. 92-95). IEEE.

[5] Incerto, E. and Tribastone, M., 2019, March. Model-based performance self-adaptation: A tutorial. In Companion of the 2019 ACM/SPEC International Conference on Performance Engineering (pp. 49-52).

[6] Cheng, S.W., Garlan, D. and Schmerl, B., 2009, May. Evaluating the effectiveness of the rainbow self-adaptive system. In 2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems (pp. 132-141). IEEE.

[7] Di Menna, F., Muccini, H. and Vaidhyanathan, K., 2022, April. FEAST: a framework for evaluating implementation architectures of self-adaptive IoT systems. In Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing (pp. 1440-1447).

[8] Vaidhyanathan, K., 2021. Data-Driven Self-Adaptive Architecting Using Machine Learning.

[9] Cámara, J., Correia, P., De Lemos, R., Garlan, D., Gomes, P., Schmerl, B. and Ventura, R., 2013, May. Evolving an adaptive industrial software system to use architecture-based self-adaptation. In 2013 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS) (pp. 13-22). IEEE.

[10] Mesquita, E.D.M., Sampaio, R.C., Ayala, H.V.H. and Llanos, C.H., 2020. Recent meta-heuristics improved by self-adaptation applied to nonlinear model-based predictive control. IEEE Access, 8, pp.118841-118852.

[11] Mockus, A., Anda, B. and Sjøberg, D.I., 2010, May. Experiences from replicating a case study to investigate reproducibility of software development. In Proceedings of the 1st International Workshop on Replication in Empirical Software Engineering Research.

[12] Cito, J., Ferme, V., Gall, H.C. (2016). Using Docker Containers to Improve Reproducibility in Software and Web Engineering Research. In: Bozzon, A., Cudre-Maroux, P., Pautasso, C. (eds) Web Engineering. ICWE 2016. Lecture Notes in Computer Science (), vol 9671. Springer, Cham.https://doi.org/10.1007/978-3-319-38791-8\58

[13] U. Gupta, S. K. Mandal, M. Mao, C. Chakrabarti and U. Y. Ogras, "A Deep Q-Learning Approach for Dynamic Management of Heterogeneous Processors," in IEEE Computer Architecture Letters, vol. 18, no. 1, pp. 14-17, 1 Jan.-June 2019, doi: 10.1109/LCA.2019.2892151.

[14] M. Pfannemüller, M. Breitbach, C. Krupitzer, C. Becker and A. Schürr, "Enhancing a Communication System with Adaptive Behavior using REACT," 2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C), 2020, pp. 228-229, doi: 10.1109/ACSOS-C51401.2020.00062.

[15] Potdar, A.M., Narayan, D.G., Kengond, S. and Mulla, M.M., 2020. Performance evaluation of docker container and virtual machine. *Procedia Computer Science*, *171*, pp.1419-1428.