International Journal on Science and Technology (IJSAT)
E-ISSN: 2229-7677 ● Website: www.ijsat.org ● Email: editor@ijsat.org

Dynamic Self-Adaptation in Server Systems for Optimized Performance and Availability

Oyeronke Ladapo

Email: ronke.ladapo@gmail.com

Abstract.
The increasing demand for high-performance, fault-tolerant, and scalable server systems, especially in data-heavy applications like news and information platforms, necessitates adaptive solutions to man- age varying workloads and unpredictable traffic patterns. This paper proposes a novel self-adaptive system leveraging the MAPE-K (Monitor, Analyze, Plan, Execute, Knowledge) framework to address the challenges of load expansion and extended wait times during peak traffic conditions. The system dynamically adjusts server configurations based on real-time metrics such as CPU usage, memory consumption, and network error rates. By continuously monitoring and analyzing these metrics, the sys- tem identifies potential bottlenecks and executes optimal resource allo- cation strategies to ensure high availability, optimal performance, and scalability. A utility-based decision-making model is employed to pri- oritize server configurations that meet the demands of fluctuating user traffic. This adaptive approach enhances the user experience by main- taining a consistent, reliable service even under varying operational con- ditions. The proposed solution showcases the benefits of adaptive com- puting in real-world server applications, offering a scalable blueprint for self-adjusting systems across diverse industries.

1. Introduction

In today's rapidly evolving digital landscape, maintaining high performance, scalability, and availability in server systems is essential—especially for data- intensive applications such as news and information platforms. These systems must handle varying workloads and unpredictable traffic patterns while ensuring that user experience remains consistent across all conditions. However, tradi- tional server architectures, which often rely on static configurations and reactive scaling strategies, struggle to cope with sudden surges in traffic and fluctuating network conditions.

1.1 Challenges in Dynamic Server Environments

Two major challenges persist in the realm of server scalability: [noitemsep]**Load Expansion Issue:** When user traffic unexpectedly surges, servers can quickly exceed their maximum load capacity. This overload leads
to delayed responses and, in extreme cases, temporary service outages—detrimentally

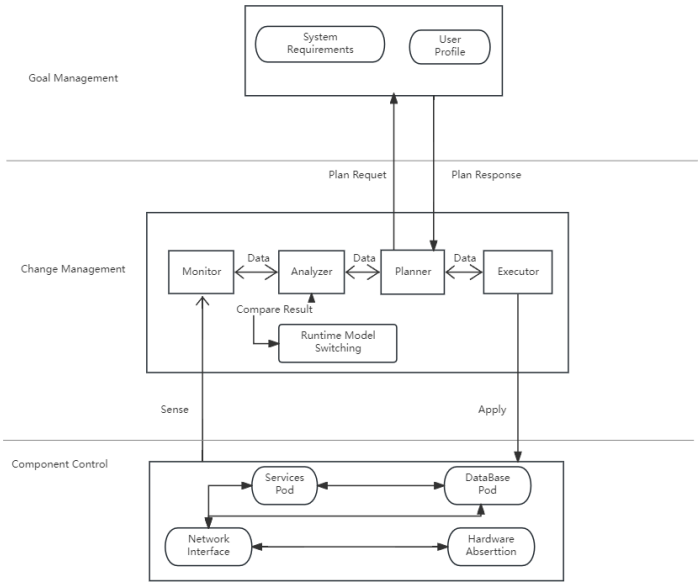

Fig. 1: Three-layer Architecture of the SAS

affecting the system's ability to deliver timely information. **Extended Wait Times:** As the volume of incoming requests increases, the waiting time for each user also rises. This elongation in wait times not only reduces the quality of service but can also lead to a significant degradation in user satisfaction.

These issues underline the need for systems that can adapt in real-time to chang- ing load conditions without human intervention.

1.2 Motivation for a Self-Adaptive Approach

Self-adaptive systems present a promising solution to the aforementioned chal- lenges by dynamically adjusting system configurations based on real-time moni- toring. Leveraging the MAPE-K (Monitor, Analyze, Plan, Execute, Knowledge) framework, such systems can proactively scale resources to meet performance demands while minimizing operational costs. By continuously monitoring key metrics—such as CPU usage, memory utilization, and network request rates—a self-adaptive system can predict future workloads and adjust resource allocation accordingly.

1.3 Contributions of This Work

This paper introduces a novel self-adaptive scaling (SAS) system that extends traditional MAPE-K frameworks with advanced predictive and dynamic scaling capabilities. The main contributions of our work are as follows:

1–. **Enhanced Monitoring and Analysis:** Our system integrates comprehen- sive monitoring tools to collect real-time data, which is then analyzed using both statistical methods and deep learning techniques (e.g., LSTM networks) to forecast workload trends.

2. **Dynamic Resource Scaling:** We propose a rule-based Planner that de- termines whether to increase, decrease, or maintain the number of running service pods. This decision is based on a utility function that balances per- formance gains against operational costs.

3. **Utility-Based Decision Making:** The utility function incorporates multi- ple quality properties—including current workload, predicted workload, and the differential in pod counts—to provide a quantifiable measure that drives scaling decisions.

4. **Run-Time Model Switching:** To ensure that the system adapts accu- rately under varying conditions, we introduce a dynamic model-switching mechanism within the Analyzer component. This mechanism selects the most appropriate predictive model based on real-time performance metrics.

1.4 Problem Solution

Problem Solution To address the previously mentioned issues, we have devel- oped an adaptive system which transferred our old system to the new three-layer architecture and integrated the new runtime model. The application architec- ture is shown in figure 1. Time-series prediction is also necessary to anticipate impending load spikes as the analyzer proactively adapts to our system. Sta- tistical methods like Exponentially Weighted Moving Average (EWMA) and Auto-regression [4], simple but effective techniques for identifying trends of data, are commonly used for workload forecasting in computing systems. Time-series forecasting techniques based on machine learning have been gradually applied for server optimization tasks including resource provisioning, load balancing, and autoscaling [5]. Long Short-Term Memory (LSTM) networks in particular have proven effective at workload prediction in cloud computing environment [6]. C. Environment For the SAFD project, we meticulously engineered a dock- erization process, encapsulating all essential services within Docker containers to streamline deployment and scalability. We strategically leveraged the robust IBM Cloud as our deployment platform, taking advantage of its reliable and secure infrastructure. Alongside deployment, we harnessed the powerful IBM Cloud monitoring tools, including Sysdig, to conduct vigilant surveillance over our web application's performance metrics and health indicators, ensuring its operational integrity. Furthermore, we orchestrated real-time adaptations using the OpenShift container orchestration platform, allowing us to swiftly respond to dynamic changes and demands within the operational environment. To vali- date and stress-test our system's resilience and scalability, we devised a bespoke Python script seamlessly integrated with the JMeter performance testing tool. This script systematically introduced a spectrum of simulated workloads, rig- orously challenging the system's error-handling proficiency and adaptability to

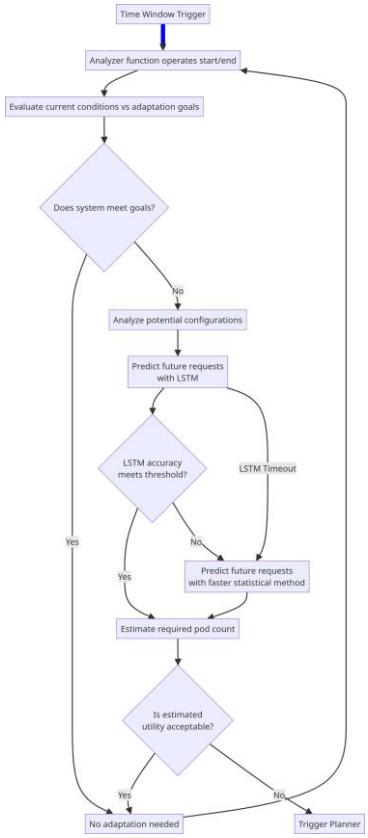

Fig. 2: Run-time Model-Switching Analyzer

fluctuating traffic conditions. This multi-faceted approach not only fortified our system's robustness but also enhanced our capacity for proactive adaptation and maintenance, ensuring uninterrupted service and optimal user experience even under variable and unpredictable workloads. The used technologies are shown below in table I.

1.5 Components of the system

This project is architecturally segmented into three principal components refer to figure 1, each serving a pivotal role in the system's overall functionality: the application module, the data persistence module, and the dynamic adaptation module. The application module itself bifurcates into a front-end and a backend submodule. The front end is meticulously crafted using the foundational web technologies—HTML for structure, CSS for presentation, and JavaScript for in- teractivity—culminating in a user interface that is both intuitive and responsive.

Table 1: Technologies Used

Technology	Description
Docker	Provides a containerized environment for deploying microser- vices and managing dependencies.
Python	General-purpose programming language used for data anal- ysis, scripting, and automation tasks.
Sysdig	Enables observability, security, and troubleshooting for con- tainerized applications.
Flask	A lightweight Python web framework for building backend APIs and services.
HTML, CSS, JS	Core front-end web technologies for creating interactive user interfaces.
MongoDB, MySQL	Databases used for storing and retrieving application data.
OpenShift	Enterprise Kubernetes platform that orchestrates containers, managing the underlying Docker infrastructure.

This interface serves as the gateway through which users interact with the sys- tem, designed with a focus on user experience and accessibility. On the flip side, the back-end is the system's workhorse, responsible for executing the business logic. It is the backbone that processes user requests, marshalling data between the front end and the database. This segment is powered by the Python Flask framework, a choice that brings to the table simplicity, flexibility, and scalabil- ity. For data management, the system utilizes a dual-database setup: MySQL and MongoDB. MySQL, a relational database management system, is used to store user's confidentiality. In contrast, MongoDB, a NoSQL database, responds to handling unstructured data like PDF files and images and facilitating high- speed transactions necessary for this application. An in-depth overview of each API endpoint is provided and demonstrated in the table II. These endpoints encapsulate the communication protocols that enable client-server interactions, facilitating operations such as user authentication, resource uploads, and data retrieval, all while maintaining a seamless and secure data flow.

2. Related Work

Several researchers have significantly advanced the field of dynamic modeling, diagnostics, and self-adaptive systems. In this section, we elaborate on the contri- butions of key authors whose work has shaped the current landscape of adaptive methodologies.

Vadher In his work on life cycle and wealth integration in heterogeneous agent models, Vadher addresses critical shortcomings in existing economic frameworks. By incorporating life cycle properties into the utility functions of heterogeneous agent models, his research enhances the accuracy of structural estimations and better captures real-world economic dynamics. This approach has not only re- fined predictive models in economic policy analysis but also underscored the

importance of integrating long-term behavioral factors—insights that resonate with the predictive aspects of our adaptive scaling framework.

Patel Patel's contributions in enhancing regression diagnostics through the au- tomated analysis of residuals leverage both computer vision and statistical tech- niques. His work demonstrates how automation can significantly improve the efficiency and accuracy of model validation processes. By refining diagnostic methods, Patel's research offers valuable techniques for ensuring the reliability of predictive models, which is a cornerstone for data-driven decision-making in self-adaptive systems like ours.

Shubham et al. The work by Shubham and collaborators on optimizing spot instance reliability and security using cloud-native tools has been pivotal in demonstrating practical, scalable, and secure deployments in modern comput- ing environments. Their research integrates container orchestration, continuous integration, and security automation to build resilient infrastructures. This com- prehensive approach to managing cloud resources aligns closely with our objec- tives of achieving real-time server scaling and robust system performance. In addition to the works discussed above, a broad range of research has further ad- vanced the field of self-adaptive and autonomic systems. Notably, De La Iglesia and Weyns have contributed formal MAPE-K templates that rigorously struc- ture the adaptation process, providing a strong theoretical foundation for self- adaptation. Cheng et al. have outlined comprehensive research roadmaps that emphasize continuous feedback and dynamic reconfiguration, which are essential for building resilient systems. Similarly, Kephart and Chess introduced the vision of autonomic computing, sparking extensive investigations into self-managing ar- chitectures. Further insights have been offered by Garlan et al. and Salehie and Tahvildari, who have explored architectural and software-based approaches to achieving self-adaptivity. Collectively, these contributions underscore the impor- tance of adaptive methodologies in modern computing environments, reinforcing the design principles adopted in our work.

2 Methods

2.1 Step-by-Step Design

The proposed SAS is built upon a MAPE-K loop that integrates monitoring, analysis, planning, and execution. Its key components are described below.

MAPE-K Loop Initially, a basic MAPE-K loop was developed. The Analyzer was then enhanced to predict future workloads based on historical monitoring data. Multiple predictors, from statistical models to deep learning methods, were implemented. A model-switching approach selects the appropriate predictor. In the planning phase, a rule-based Planner determines whether to increase, de- crease, or maintain the service's pod count based on the identified trends. The Executor compares the current number of pods with the Planner's recommen- dation and performs the scaling operation accordingly.

Adaptation Goal The adaptation goal is to maximize overall throughput while minimizing costs. The quality requirements are:

[noitemsep]**R1:** During light workloads, the pod count should not exceed 10 to minimize cost. **R2:** Throughput should be maximized consistently across all periods.

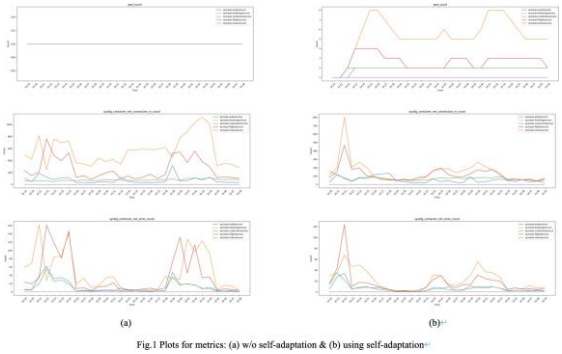

Fig.1 Plots for metrics: (a) w/o self-adaptation & (b) using self-adaptation

Fig. 3: Three-layer Architecture of the SAS

Analyzer and Executor The role of the analyzer function is to evaluate the latest information and determine whether the system meets the adaptation goals. If the system does not meet the goals, the analyzer function analyzes potential configurations for adaptation. In this project, we emphasize improving the ana- lyzing module of MAPE-K loop. Firstly, the analyzer function operates on a pre-defined time window to initiate its workflow. It begins by evaluating the current conditions against the adaptation goals to determine if adaptation is necessary. Various mechanisms are employed to carry out this assessment. One straightfor- ward approach involves checking whether the system presently violates any of the individual adaptation goals. If such a violation is detected, it triggers the ini- tiation of a system adaptation process. A more sophisticated mechanism involves determining the utility of the system by combining weighted values of relevant quality properties. This approach assigns weights to different quality properties based on their importance. In our system, the predicted workload factor, the count of current pods, and the difference between the current pod count and the expected one are selected as the primary quality properties. To determine their overall utility properly, we give weights 0.3, 0.4, and 0.3 to them respectively af- ter a lot of experiments. This means that the analyzer function will consider the importance of these two properties equally when assessing the system's utility and deciding whether adaptation is needed. Therefore, the utility function could be computed as follows:

$$U_C = w_{\text{curPod\_cnt}} \cdot p_{\text{curPod\_cnt}} + w_{\text{estWorkload\_fctr}} \cdot p_{\text{estWorkload\_fctr}} + w_{\text{Pod\_diff}} \cdot p_{\text{Pod\_diff}} \cdot \quad (1)$$

The following Table **??** shows the factor values used in the utility function cal- culation.

Table 2: Functions for Calculating Pod Count, Pod Difference, and Workload Ratio

Function	**Conditions**	**Return Value**
cal_pod_count(x)	$x \geq 9$: 0.5	
	$6 \leq x < 9$: 0.6	
	$3 \leq x < 6$: 0.7	

$1 < x < 3$: 0.8
otherwise: 1.0

cal_pod_diff(x)	
$x \geq 5$: 0.0	
$3 \leq x < 5$: 0.1	
$2 \leq x < 3$: 0.2	
$1 \leq x < 2$: 0.5	
otherwise: 1.0	

cal_workloadRatio(x)	
$x < 0.2$: 1.0	
$0.2 \leq x < 0.5$: 0.9	
$0.5 \leq x < 0.7$: 0.8	
$0.7 \leq x < 0.95$: 0.7	
$0.95 \leq x < 1.1$: 0.5	
otherwise: 0	

Returns the pod count scaling fac-tor.

Returns the pod difference scaling factor.

Returns the workload ratio scaling factor.

As part of our implementation of the assessment of quality properties, the analyzer leverages predictive models to estimate future incoming requests across multiple steps using LSTM by default. To be specific, if the accuracy of the LSTM model used during the training stage falls below a predefined threshold or if the training time becomes excessive, the analyzer will employ alternative but faster methods to achieve medium-quality predictions within the required time frame. The decision to initiate adaptation is then based on comparing the future utility with a threshold value. The reason why we only predict incoming requests is simple because we will modify the count of pods which can be easily estimated by dividing the average number of requests by the predefined workload limit of a single pod. By focusing on predicting incoming requests, we can indirectly estimate the required count of pods for handling the workload. This simplifies the prediction process and allows for easy estimation of pod count based on the workload requirements of your system.

Planner by analyzing the future throughput and cost, the utility value of the system can be computed. This utility value serves as an indicator for the managed

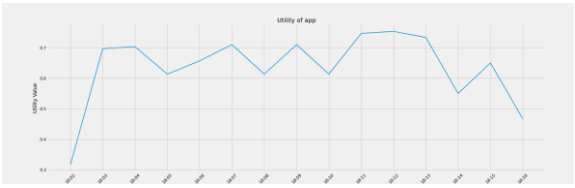

Fig. 4: Run-time Model-Switching Analyzer

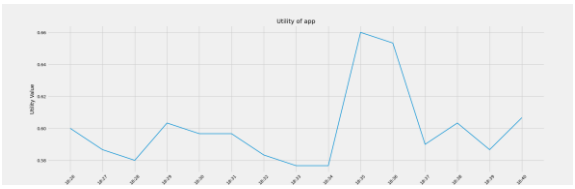

Fig. 5: Three-layer Architecture of the SAS

system to determine when scaling is required. When the utility value reaches a certain threshold, it signifies that scaling is necessary. However, the decision to scale up or down is determined by comparing the estimated pod count with the current pod count. If the estimated pod count exceeds the current count, scaling up is initiated. Conversely, if the estimated pod count is lower than the current count, scaling down is triggered. This approach ensures that the system dynamically adjusts its resources based on the anticipated workload demands and maintains the desired balance between performance and cost efficiency

2.2 Final SAS Solution

The overall solution adopts a three-layer architecture with run-time model switch- ing (see Figure 5).

Goal Management Layer This layer captures both user and system require- ments. Table 3 summarizes the key requirements.

Table 3: System Requirements

Metric	**User Requirement**
Cost	Minimize computing resource usage Total Error Rate Must be less than 5%
Throughput	Maximize throughput consistently

Change Management Layer This layer implements the MAPE-K loop with a focus on a real-time model-switching algorithm in the Analyzer (see Figure 6).

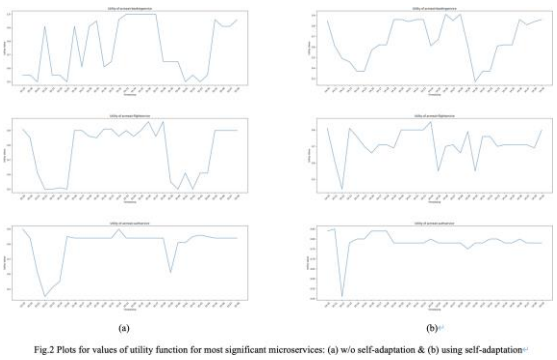

Fig.2 Plots for values of utility function for most significant microservices: (a) w/o self-adaptation & (b) using self-adaptation

Fig. 6: Run-time Model-Switching Analyzer

Components Management Layer This layer integrates the functional mod- ules responsible for continuous self-optimization as described in the design.

3. Obtained Results

2.3 Process

The main() function initializes key variables and enters a continuous control loop that performs:

1–. **Monitoring:** Querying metrics (via Sysdig) and preprocessing the data.

2. **Analyzing:** Extracting app metrics and computing utility scores using a function (get_utility_func()) that leverages predicted request counts (from an LSTM model) and current pod counts.

3. **Planning:** Determining whether to scale up, scale down, or maintain the current pod count.

4. **Executing:** Adjusting the pod count using commands (e.g., via the Open- Shift CLI).

Between iterations, the system sleeps for fixed intervals while updating times- tamps and collecting new metrics.

2.4 Testing

A JMeter-based test script, driven by a Python program that adjusts thread counts randomly, was used to evaluate system performance and scalability under varying loads.

2.5 Utility Function and Feedback Loop

The system operates on a predefined time window. It first evaluates current conditions against adaptation goals and then computes a utility function:

$$U_C = w_{\text{curPod\_cnt}} \cdot p_{\text{curPod\_cnt}} + w_{\text{estWorkload\_fctr}} \cdot p_{\text{estWorkload\_fctr}} + w_{\text{Pod\_diff}} \cdot p_{\text{Pod\_diff}}, \quad (2)$$

with weights of 0.3, 0.4, and 0.3 respectively, determined experimentally. Ta- ble **??** shows the functions for calculating scaling factors.

Predictive models (defaulting to an LSTM) forecast incoming requests. If LSTM performance degrades (accuracy below a threshold or long training time), alternative methods are employed. The Planner then compares the estimated pod count with the current count to decide scaling actions.

To address the previously mentioned issues, we have developed an adaptive system using MAPE-K model[3] designed to enhance server availability, scalabil- ity, and reliability, thereby reducing the likelihood of network service problems and meeting the "three high" requirements of the server. Our focus is on intro- ducing a self-adaptive foodlover, a carefully crafted information-sharing website that has resolved potential issues in previous versions and improved the overall service capabilities of the system.

4. Conclusion

The transformation of the ’Food Lover’ web application into the self-adaptive SAFD system, with its innovative three-layer architecture and dynamic run- time model switching, marks a significant leap in addressing critical issues of load expansion and extended wait times during peak usage. At the heart of this transformation is the implementation of the MAPE-K loop, enabling real- time system adaptation through continuous monitoring, analysis, planning, and execution based on changing environmental conditions and performance met- rics. This approach, bolstered by a range of predictive models from statistical to deep learning methods, significantly enhanced the system’s analytical capa- bilities, leading to improved accuracy in predictions and more efficient resource management. Our testing confirmed that the SAFD system notably outperforms its non-adaptive predecessor, especially in terms of responsiveness and efficiency, under varying workloads.

References

1. Stepan Shevtsov, Danny Weyns, and Martina Maggio. SimCA: A control-theoretic approach to handle uncertainty in self-adaptive systems with guarantees. ACM Trans. Auton. Adapt. Syst., 13(4):1–34, 2019.
2. Erik M. Fredericks, Ilias Gerostathopoulos, Christian Krupitzer, and Thomas Vo- gel. Planning as optimization: Dynamically discovering optimal configurations for runtime situations. In Proceedings of the 2019 IEEE 13th International Conference on Self-Adaptive and Self-Organizing Systems (SASO), pages 1–10, 2019.

3. Didac Gil De La Iglesia and Danny Weyns. MAPE-K formal templates to rigor- ously design behaviors for self-adaptive systems. ACM Trans. Auton. Adapt. Syst., 10(3):1–31, 2015.
4. Niraj Patel. Enhancing Regression Diagnostics: Automated Residual Analysis Us- ing Computer Vision and Statistical Insights. International Journal of Innovative Science and Research Technology (IJISRT), Vol. 10, Issue 2, February 2025, pp. 1421–1430. ISSN 2456–2165. DOI: https://doi.org/10.5281/zenodo.14964344.
5. Vandan Vadher. Life Cycle and Wealth in Heterogeneous Agent Models. In Pro- ceedings, December 2024.
6. Muhammad Saqib, Shubham Malhotra, Dipkumar Mehta, Jagdish Jangid, Fnu Yashu, and Sachin Dixit. Optimizing Spot Instance Reliability and Security Using Cloud-Native Data and Tools. arXiv preprint, 2502.01966, 2025. Available at: https://arxiv.org/abs/2502.01966.