ComFaaS+: Optimizing Edge and Cloud Function-as-a-Service with Adaptive Resource Allocation using Machine Learning

Laween Al-Sulaivany\*

*Computer Science and Information Systems*

*Minnesota State University Moorhead*

Moorhead, MN

[laween.al-sulaivany@go.mnstate.edu](mailto:laween.al-sulaivany@go.mnstate.edu)

\*Corresponding author

Judah Nava\*

*Computer Science and Information Systems*

*Minnesota State University Moorhead*

Moorhead, MN

[judah.nava@go.mnstate.edu](mailto:judah.nava@go.mnstate.edu)

\*Corresponding author

Hanku Lee\*

*Computer Science and Information Systems*

*Minnesota State University Moorhead*

Moorhead, MN [hanku.lee@mnstate.edu](mailto:hanku.lee@mnstate.edu)

\*Corresponding author

Jaden Jinu Lee

*Computer Science and Information Systems*

*Minnesota State University Moorhead*

Moorhead, MN jinu.lee@go.mnstate.edu

*Abstract*—Function-as-a-Service (FaaS) continues to revolutionize cloud and edge computing by providing a scalable, event-driven execution model that abstracts hardware and infrastructure complexities. While cloud-based FaaS platforms offer flexibility and powerful computational resources, they introduce latency and bandwidth constraints, particularly for real-time and resource-sensitive applications. To address these limitations, the latest iteration of ComFaaS integrates an AI-driven resource allocation mechanism that optimizes execution between cloud and edge environments based on weighted factors such as latency, CPU availability, and memory usage. Additionally, ComFaaS also streamlines system setup, improves logging, supports multi-threading, and includes a web-based graphical interface. The AI API analyzes tasks, automatically estimates priority weights, and collaborates with ComFaaS to decide where each function will run. Benchmark evaluations demonstrate notable improvements in execution speed, automated workload distribution, and overall scalability. These advancements pave the way for future machine learning–driven optimizations tailored to dynamic edge-cloud scenarios.

Keywords—FaaS, Function-as-a-Service, Edge Computing, Cloud Computing, Resource Allocation, Machine Learning, Serverless Computing.

# Introduction

The rise of cloud and edge computing has transformed how modern applications process and execute computational workloads. Function-as-a-Service (FaaS) has emerged as a key player in this evolution, providing an event-driven execution model that abstracts infrastructure complexities, enabling developers to focus on application logic. Cloud computing has traditionally been the dominant platform for FaaS, offering scalability and centralized resource management through platforms like AWS Lambda and Google Cloud Functions. However, this centralization introduces challenges, particularly in latency-sensitive applications where real-time responses are critical.

Edge computing addresses these limitations by decentralizing computation, bringing processing power closer to data sources. This paradigm shift significantly reduces latency, minimizes bandwidth consumption, and enables real-time decision-making, making it highly suitable for applications in IoT, autonomous systems, and high-frequency data processing environments. While edge computing enhances responsiveness, balancing workload distribution between edge devices and cloud resources remains an ongoing challenge. ComFaaS was initially introduced as a framework to evaluate and compare the effectiveness of FaaS implementations across cloud and edge environments.

## Problem Statement

Previous iterations of ComFaaS relied on manual selection of execution environments, requiring users to specify whether a task should run on an edge device or a cloud server. This approach, while effective, lacked dynamic adaptability and resulted in inefficient resource utilization. Additionally, the absence of structured logging and automated scheduling hindered system scalability and performance monitoring. As applications grow increasingly complex, the need for an intelligent, automated resource allocation mechanism becomes evident.

## Contributions

To address these challenges, we introduce significant enhancements to ComFaaS, expanding its functionality and improving overall system efficiency. The key contributions of this work include:

* AI-Driven Resource Allocation: A novel mechanism couples a decision-making API with ComFaaS’s core scheduler. The AI API inspects each incoming task, infers relevant priority weights (latency, CPU, memory), and collaborates with an internal algorithm that finalizes where to run the task.
* Enhanced System Setup and Execution – A streamlined directory structure and unified setup scripts for both Windows and Linux environments, ensuring ease of deployment and execution.
* Structured Logging for Data-Driven Decision-Making – Improved logging capabilities with CSV-based data collection to facilitate performance tracking and future machine learning integrations.
* Expanded Testing Environment: Introduction of six additional edge servers to evaluate scalability and performance across a distributed network.
* Multi-Threading: Added support for multi-threading execution which allows us to simulate a larger range of real-world concurrency scenarios.

These advancements mark a significant evolution of ComFaaS, positioning it as a more robust and intelligent FaaS framework capable of adaptive execution across cloud and edge infrastructures.

The remainder of this paper is structured as follows: Section II explores related research and prior work on edge-cloud resource allocation. Section III details the architecture and system enhancements introduced in ComFaaS. Section IV presents experimental benchmarks and performance evaluations. Section V discusses the results and future directions for machine learning integration. Finally, Section VI concludes the paper with key findings and implications for future research..

# Related Work

In this section, we review notable advances in edge-cloud resource allocation, highlight the evolution of ComFaaS through prior publications, and examine the emerging role of machine learning in Function-as-a-Service (FaaS) optimization.

## Comparison to Previous ComFaaS Work

ComFaaS was originally conceived as a proof-of-concept framework for comparing the performance of FaaS deployments across cloud and edge infrastructures. Early work focused on latency analysis under varying network conditions and benchmarking different workloads (e.g., image processing and floating-point operations) to illustrate the advantages of offloading tasks closer to the data source [1], [2]. In subsequent iterations, ComFaaS expanded its scope by introducing parallel workloads (i.e., SPMD and message-passing libraries) to accommodate computationally intensive tasks, thereby deepening the exploration of edge-cloud synergy [3]. Despite demonstrating superior performance in latency-sensitive scenarios, earlier releases faced limitations such as:

### Manual Resource Selection: Users were required to specify if a function should run on the cloud or at the edge, limiting automation.

### Manual resource assignment could waste available CPU cores or lead to uneven distribution.

### Limited analytics infrastructure (sparse logging) offered minimal insights for advanced scheduling.

### Connections between nodes assumed minimal dynamism; large-scale or frequently changing networks were not rigorously tested.

### Restricted Language Support: Most prototypes were implemented in Java only, with no built-in modularity to accommodate new languages.

In contrast, the newest version of ComFaaS addresses these gaps by:

* Introducing automated resource allocation via an algorithm that weighs latency, CPU capacity, and free memory.
* Enhancing debugging and analytics through CSV-based logging and improved console outputs.
* Streamlining system setup with unified scripts for both Windows and Linux environments.

This evolution underscores a shift from a rudimentary prototype to a more comprehensive platform capable of dynamic scaling, automated decisions, and robust performance evaluation across diverse workloads.

## Edge vs. Cloud in FaaS

Contemporary research underscores edge computing as a critical solution for reducing the latency and bandwidth overhead often encountered in centralized cloud models [4], [5]. FaaS deployments at the edge can respond more quickly to localized events, making them ideal for time-sensitive applications such as smart cities, healthcare monitoring, and autonomous vehicles. Several studies propose advanced scheduling algorithms tailored for these distributed environments. For example, frameworks like PureFaaS, StateProp, and StateLocal illustrate how a network of edge nodes can communicate and offload tasks without overloading any single device [5], [6]. At the same time, peer-to-peer FaaS systems (P2PFaaS) have explored decentralized scheduling to mitigate latency by distributing the balancing logic across edge nodes [7].

However, these approaches often handle resource management primarily at the edge side, relying on heuristic or static rules that do not fully integrate with cloud-based resources. By contrast, ComFaaS combines edge-level autonomy with cloud-coordinated orchestration. Its new algorithm specifically takes into account real-time factors, such as network round-trip time and server load, enabling dynamic placement decisions that are neither purely edge-driven nor fully cloud-centric.

## Machine Learning in Resource Allocation

While traditional scheduling heuristics (e.g., shortest queue, round-robin) have been employed to route FaaS requests, machine learning (ML) is gaining traction for more adaptive workload allocation [8]–[10]. Reinforcement learning algorithms, for instance, dynamically learn from incoming task patterns and network states, incrementally refining how and where tasks are offloaded. Supervised models may also leverage historical data (e.g., CPU usage trends, data transfer rates) to predict load spikes and preemptively provision resources, thereby reducing congestion and delays.

Although ComFaaS primarily employs a weighted heuristic algorithm to determine resource allocation, it now integrates an AI-driven API that can further refine placement decisions. This hybrid strategy allows on-the-fly exploration of ML-based optimizations while retaining a robust baseline when ML models are insufficiently trained. The system’s logging mechanism, which stores event-driven metrics in CSV format, lays the groundwork for future ML models to train on extensive, real-world data and potentially replace the heuristic approach if performance gains can be demonstrated. ComFaaS’s new architecture incorporates an AI API that gathers runtime metrics, analyzes each task, and determines suitable resource weights before the final algorithm decides which node (edge or cloud) will host execution.

Together, these advancements in edge, cloud, and ML-based resource management provide the conceptual and technical underpinnings for ComFaaS’s latest iteration. In the following sections, we present the architectural improvements made to ComFaaS, outline the new automated scheduling algorithm in detail, and evaluate its effectiveness under diverse test scenarios.

# System Architecture and Ehancements

In this section, we describe the latest improvements made to ComFaaS, focusing on its reorganized directory structure, streamlined setup scripts, enhanced logging capabilities, a newly introduced web-based GUI, and performance-oriented modifications. These enhancements aim to create a more robust and easily maintainable system capable of managing larger workloads across both cloud and edge infrastructures.

## ComFaaS System Enhancements

### Directory Structure and Organization

### Previous ComFaaS implementations stored source code and configuration files in a single, monolithic repository, making maintenance cumbersome. In this release, we have reorganized the directories into a cleaner hierarchy, for example:

* backend/ – Houses the core Java classes for communication, API handling, and scheduling logic.
* scripts/ – Includes both Windows Batch and Linux Bash files for setup, compilation, and deployment.
* config/ – Holds environment properties, server IPs, and resource settings (e.g., CPU/memory constraints).
* frontend/ – Contains all frontend and backend files related to the new web-based Graphical User Interface.

This modular approach improves maintainability, easing the integration of new features. Developers can quickly locate relevant files and independently extend specific modules without disturbing the rest of the system.

### Automated Setup and Execution

To address the complexities of setting up ComFaaS on different operating systems, we provide unified scripts for Windows and Linux:

* setup**\_**environments.bat / setup**\_**environments.sh – Installs prerequisites (e.g., Python, MPI libraries), configures virtual environments, and validates network connectivity.
* compile.bat / compile.sh – Compiles all core classes and any user-defined FaaS functions.

By running these scripts in sequence, users can deploy ComFaaS reliably with minimal manual intervention. This ensures consistency across various environments, reduces the chance of dependency errors, and accelerates the onboarding process for new collaborators.

### Improved Console Output and CSV-Based Logging

ComFaaS now employs color-coded console messages (e.g., green for successful events, yellow for warnings, red for errors) to provide immediate visual feedback during operation. In parallel, the system logs execution details to CSV files for structured data analysis. Each log entry records:

* Timestamp – Marks when the function execution started and finished.
* Resource Usage – CPU load, memory usage, and network latency.
* Outcome – Success, warning, or error state.

This logging strategy is particularly beneficial for:

* Diagnostics – Pinpointing performance bottlenecks or configuration issues.
* Machine Learning – Serving as training data for the AI-based scheduling approach (described in Section IV-B).

These improvements ensure that ComFaaS produces comprehensive, high-quality datasets, paving the way for iterative optimization.

## Web-Based GUI

ComFaaS includes a basic web interface for real-time status checks, although its current capabilities remain limited. Users can:

* Upload and run tasks through the GUI interface
* Download the output
* Monitor basic job logs.

Future work will likely enhance this GUI’s role in visual diagnostics, system tuning, and advanced debugging, but the present interface only provides foundational functionality.

## Performance Enhancements

### Code Optimization for Execution Speed

Refactoring efforts targeted file I/O and redundant computations within core scheduling classes. Repetitive tasks, such as downloading the same FaaS binaries multiple times, are now minimized via caching strategies on each node. Early internal tests show:

* 10–15% (test between the old comfaas and the new comfaas) faster function startup due to caching and streamlined initialization.
* Reduced communication overhead by bundling resource checks into single API calls.

Such optimizations make the system more responsive under heavy loads, a crucial improvement for real-time or mission-critical applications.

### Scalability Improvements

ComFaaS now supports six additional servers in the testing environment, each capable of hosting multiple simultaneous FaaS invocations. We conducted load tests simulating bursty IoT sensor data, large media processing requests, and high-volume concurrency. Results indicate that ComFaaS can handle up to 2,000 function invocations per minute with only minor latency increases (Section V-C details the test setup and results). This expansion in server capacity provides a broader landscape for validating both performance and fault tolerance under real-world conditions.

# Algorithm for Intelligent Edge-Cloud Resource Allocation

A key innovation in this ComFaaS release is the automated scheduling algorithm, which alleviates users from manually deciding whether a function should execute on the cloud or a specific edge node. By dynamically evaluating real-time system metrics, ComFaaS aims to achieve optimal distribution of workloads across heterogeneous environments.

## Design of the Algorithm

### AI API Preprocessing: Upon receiving a new job request, the AI API examines the job type (e.g., CPU-bound, memory-intensive, latency-sensitive) and uses historical logs to estimate the importance of factors such as latency, CPU, and memory.

### Benchmarking: A copy of the job file is sent to the cloud for a benchmark run, which outputs performance metrics.

### Heuristic Scoring: ComFaaS uses the AI-suggested weights along with the benchmark outputs to calculate a composite score for each node. Here, Li, Ci, and Mi represent the node’s latency, available CPU capacity, and free memory, respectively.

### Score Calculation: For node i, ComFaaS computes a composite score using the following formula:

### Network Latency (L): Round-trip time to the cloud, measured via pings or heartbeat signals.

### CPU Availability (C): Number of free or CPU cores.

### Memory Usage (M): Remaining RAM capacity.

### These metrics are weighted based on their relative importance; higher scores indicate a more favorable execution environment.

### Dynamic Data Collection: When a node disconnects or a new node connects, the cloud polls each edge node to update metrics. These data points are stored in a global table accessible by the scheduler.

### Scheduling Decision: When a client submits a FaaS job, the scheduler reviews all node scores and selects the node with the highest score for task execution. Status updates are logged in CSV files for performance tracking.

This scoring strategy continuously adapts to load changes, ensuring that new tasks gravitate toward nodes with faster response times or greater resource availability.

## Integration of AI API for Optimization

### Short-Term AI Integration

Beyond the heuristic method, ComFaaS integrates an AI-based inference API that preprocesses each incoming task by predicting priority weights. This API works in tandem with an internal benchmark function—which runs the program once to gather runtime weights—and both sets of weights are combined within the scheduling algorithm to decisively determine the optimal edge node for execution. This API ingests each node’s real-time metrics plus historical job completion data (from the CSV logs) to predict the best placement for incoming jobs. An AI-based scheduler can:

• Adjust predictions as usage patterns evolve.

• Detect hidden correlations (e.g., frequent memory spikes) that a simple heuristic might miss.

Preliminary experiments show 5–10% improvements in average job completion times for CPU-intensive tasks. The AI approach is a critical and mandatory component of ComFaaS, ensuring that the API and the benchmark function operate in unison to optimize resource allocation.

### Future Machine Learning Implementation

If the AI approach consistently outperforms the heuristic method, we intend to:

#### Collect Extensive Data: Gather logs from thousands of invocations under varied loads.

#### Train Specialized Models: Separate models for memory-bound tasks, CPU-heavy tasks, etc.

#### Implement Online Learning: Dynamically retrain or fine-tune models with each new log entry, enabling rapid adaptation to changing environments.

Conversely, if AI improvements are minimal or variable, we will investigate better hyperparameter tuning, additional input features (e.g., bandwidth usage patterns), or advanced reinforcement learning algorithms. The long-term goal is a fully autonomous system that continuously improves scheduling decisions as more data becomes available.

# Expertimental Evaluation

We conducted a series of tests to assess how ComFaaS’s new features—including code optimizations, expanded server support, and AI-driven scheduling—impact overall performance. Both synthetic benchmarks and real-world workloads were used to validate the platform’s capabilities.

## Test Setup

### Hardware Configuration:

* Cloud: One high-end server (HPE DL360s Xeon-s 4108 8 Core CPU, 16GB of RAM)
* Edge: Edge: Eight high-end server (HPE DL360s Xeon-s 4108 8 Core CPU, 16GB of RAM)
* Network: Latencies ranged from 15 ms to 100 ms, simulating geographically diverse edges.

### Benchmark Workloads:

* Prime Computation: CPU-heavy workload calculating primes up to 10 million.
* Video Transcoding: Converts 50 short video clips (1080p to 480p).
* IoT Sensor Simulation: Streams random sensor data at rates up to 5,000 messages/second, requiring quick processing to avoid backlog.

## Performance Metrics

### Manual vs. Automated Scheduling: We compared user-selected deployment against the new dynamic scheduler. Key metrics were task completion time, server utilization, and queue length under bursts of requests.

### Speed Improvements from Code Optimizations: We measured average function startup time and total execution time before and after introducing caching and concurrency refinements.

### Effectiveness of AI API: A subset of the benchmarks was scheduled by either the heuristic-based or AI-based methods, allowing us to compare average response times, resource usage variance, and error rates.

## Results and Analysis

### Execution Time and Throughput

Table I summarizes selected results for the Prime Computation workload:

| Table 1 | EXECUTION TIMES FOR PRIME COMPUTATION (MILLIONS OF PRIMES) | | |
| --- | --- | --- | --- |
| Method | Time on Edge (s) | Time on Cloud (s) |
|  | Manual (User-Selected) | 10.2 | 12.8 |
|  | Heuristic Scheduler | 8.5 | 11.9 |
|  | AI-Based Scheduler | 7.8 | 11.3 |

We observe that the AI-Based Scheduler consistently outperforms both Manual and Heuristic approaches, reducing average edge execution time by up to 23% relative to manual deployment. On the cloud side, gains are modest (around 12%), likely due to higher baseline capacity.

### Resource Utilization and Load Balancing

The automated scheduler demonstrated superior load balancing, effectively distributing tasks among underutilized nodes. During high-load tests, CPU usage remained between 60% and 80% on each edge server, whereas manual selection often led to 100% utilization on one node while others remained idle. This improved distribution correlates with shorter queue times and fewer dropped tasks.

### Impact of Code Optimizations

By localizing frequently reused libraries and caching large data files, ComFaaS reduced function startup times by around 15% (from 2.0 seconds to 1.7 seconds on average). Across a 1,000-invocation run, these incremental improvements aggregated into a significant time saving, illustrating how internal code refactoring can bolster throughput.

Overall, these experiments confirm the efficacy of the new intelligent resource allocation mechanism and the tangible benefits of incremental code and structural enhancements. The next section discusses how these insights guide ongoing work toward even more advanced ML-driven models.

# Discussion & Future Work

## Key Findings

Our results underscore the tangible benefits of ComFaaS’s latest upgrades:

### Enhanced Setup & Maintenance: Unified scripts, modular directories, and dynamic configuration files reduce user errors and facilitate rapid deployment.

### Automated Scheduling: Even basic heuristic scoring yields consistent throughput gains, while the optional AI-based approach offers further latency reductions.

### Resource Efficiency: Color-coded console outputs and CSV logs enable deeper performance analysis, supporting real-time decision-making and guiding future ML improvements.

## Limitations & Challenges

### Edge-Cloud Coordination Complexity: Network fluctuations or temporary node outages may mislead the scoring system, leading to suboptimal resource choices.

### Network Variability: Latency can spike unpredictably, skewing short-term data collection. Some nodes may be temporarily underrated or overrated.

### AI/ML Data Requirements: For AI to robustly replace heuristic methods, extensive, high-quality datasets are essential. Continuous retraining costs and potential “drift” must also be managed.

## Future Directions

### Reinforcement Learning: Implementing RL-based agents that adapt over time could dynamically refine scheduling strategies to maximize long-term efficiency.

### Hybrid Models: Combining heuristics with ML-based predictions might balance reliability and adaptiveness. For instance, heuristics handle routine tasks while ML intervenes for complex or ambiguous edge cases.

### Feedback Mechanisms: Ongoing, real-time comparisons between predicted vs. actual performance data could refine scheduling behavior continuously, allowing the system to self-correct in response to fluctuating workloads.

# Conclusion

This paper presents a major update to ComFaaS, a platform bridging cloud and edge computing for serverless Function-as-a-Service applications. By introducing automated resource allocation, restructured directories, unified setup scripts, a web-based GUI, and CSV-based logging, we significantly enhance the system’s usability, scalability, and runtime efficiency. Our experiments validate that not only do these upgrades streamline deployment, but the new AI-driven scheduling approach also outperforms conventional heuristics in certain workload types.

Looking ahead, we plan to evolve ComFaaS into a fully self-optimizing framework, incorporating reinforcement learning or continuous ML pipelines. These enhancements promise to make ComFaaS an even more compelling choice for diverse real-time, data-intensive scenarios that demand minimal latency and robust throughput.

1. [1] J. Nava, J. Lee, and H. Lee, “ComFaaS: Comparative Analysis of Edge Computing with Function-as-a-Service,” Proc. of the 2023 8th Intl. Conf. on Cloud Computing and Internet of Things (CCIOT), 2023.
2. [2] J. Nava, J. Lee, and H. Lee, “ComFaaS Distributed: Edge Computing with Function-as-a-Service in Parallel Cloud Environments,” Proc. of the IEEE Intl. Conf. on Parallel and Distributed Systems, 2024.
3. [3] J. Nava, J. Lee, and H. Lee, “ComFaaS: A Dynamic Approach to Edge and Cloud Computing with Function-as-a-Service,” Journal of Cloud Computing and Edge Innovation, vol. 9, no. 3, pp. 112–125, 2025.
4. [4] F. Abbas, G. Ahmad, S. Oteafy, and A. Vukovic, “Balancing Local vs. Remote State Allocation for Microservices in the Cloud–Edge Continuum,” IEEE Transactions on Cloud Computing, vol. 9, no. 2, pp. 194–207, 2021.
5. [5] A. Smith and M. Taylor, “FaaS Execution Models for Edge Applications: PureFaaS, StateProp, and StateLocal,” IEEE Internet of Things Journal, vol. 8, no. 7, pp. 5890–5902, 2021.
6. [6] B. Nichols and T. M. Nichols, “Distributed FaaS at the Edge: A Case for State-Aware Orchestration,” ACM Trans. on Internet Technology, vol. 19, no. 2, pp. 15–29, 2022.
7. [7] K. Prost, C. Rodriguez, and K. Karlsson, “P2PFaaS: A Framework for Peer-to-Peer Scheduling and Load Balancing in Fog and Edge Computing,” Future Generation Computer Systems, vol. 131, pp. 165–180, 2022.
8. [8] W. Zhang, J. Xiao, X. Dong, Y. Zhang, and L. He, “Machine Learning for Multi-Access Edge Computing: A Survey,” IEEE Access, vol. 8, pp. 135–155, 2020.
9. [9] A. Rathore, R. Mehta, and S. Chandra, “Adaptive Resource Allocation in Edge-Cloud Environments Using Deep Reinforcement Learning,” IEEE Internet of Things Journal, vol. 9, no. 24, pp. 26011–26024, 2022.
10. [10] Y. Zheng, Z. Liu, J. Ni, T. Zhang, and G. Qu, “Edge Intelligence in Future Internet of Things: A Comprehensive Survey,” IEEE Internet of Things Journal, vol. 10, no. 2, pp. 1209–1232, Jan. 2023.