International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 09 Issue: 09 | Sept - 2025 SJIF Rating: 8.586 ISSN: 2582-3930

Cost-Aware Resource Allocation and Execution Scheduler (Cares)

Prof. Sandarsh Gowda M M [1], Hemanth Raj N [2]

[1] Assistant Professor, Department of MCA, Bangalore Institute of Technology, Karnataka, India

[2] Student, Department of MCA, Bangalore Institute of Technology, Karnataka, India

ABSTRACT

This paper presents CARES (Cost-Aware Resource Allocation and Execution Scheduler), a distributed computing platform engineered to optimize function execution across a dynamic cluster of nodes. CARES introduces a cost-aware scheduling algorithm that leverages real-time CPU and memory metrics to achieve efficient load balancing and reduce operational costs. Built in Go, the system integrates Docker for containerized execution, gRPC for inter-node communication, and a modern Terminal User Interface (TUI) for real-time monitoring. CARES demonstrates advanced concepts in distributed systems, including service discovery, dynamic resource allocation, and fault tolerance, providing both a practical orchestration solution and an educational tool for understanding the complexities of distributed architectures.

I. INTRODUCTION

The rapid shift toward cloud-native architectures—driven by microservices, serverless computing, and containerization—has fundamentally transformed how distributed systems are designed and deployed. These paradigms enable organizations to achieve elastic scalability, rapid deployment, and operational resilience, but they also pose significant challenges in resource management, cost efficiency, and workload scheduling. In highly dynamic environments, where workloads fluctuate and resources are heterogeneous, achieving optimal utilization while minimizing costs becomes a non-trivial task.

Traditional orchestrators such as Kubernetes, Docker Swarm, and Apache Mesos provide powerful abstractions for container orchestration and cluster management. However, these systems often introduce operational complexity, steep learning curves, and opaque scheduling decisions, making them less suitable for lightweight research and experimental deployments. In particular, their scheduling mechanisms are either static (e.g., round-robin, random allocation) or optimized for generalized workloads, which may not adapt well to real-time fluctuations in resource usage.

To address these limitations, we propose CARES (Cost-Aware Resource Allocation and Execution Scheduler), a distributed orchestration framework that emphasizes transparency, cost-awareness, and real-time decision-making. CARES integrates fine-grained resource monitoring with a custom scheduling algorithm that considers both CPU and memory utilization metrics to determine the most cost-effective node for function execution. Unlike traditional black-box orchestrators, CARES provides an interactive Terminal User Interface (TUI) that enables developers and system administrators to visualize cluster health, resource consumption trends, and scheduling outcomes in real-time.

Moreover, CARES is built using Go for high concurrency and lightweight execution, gRPC for efficient inter-node communication, and Docker Engine for container-native execution of user-defined functions. This combination of technologies enables CARES to deliver a minimal yet extensible platform that is particularly well-suited for academic research, small-scale deployments, and environments where visibility, adaptability, and cost efficiency are critical. By bridging the gap between complex large-scale orchestrators and lightweight experimental frameworks, CARES demonstrates how distributed scheduling can be made both intuitive and effective.

II. RELATED WORK

Research in distributed scheduling and container orchestration has been shaped by systems like Google's Borg [1], Apache Mesos [2], Kubernetes [1], Docker Swarm [3], and Nomad [4]. While Borg pioneered large-scale centralized scheduling, Mesos introduced two-level scheduling. Kubernetes became the industry

standard but often operates as a black box. CARES differentiates itself by focusing on transparency and real-time cost-aware decisions. Prior surveys [7] highlight the limitations of static load-balancing strategies like round robin, motivating CARES's dynamic approach.

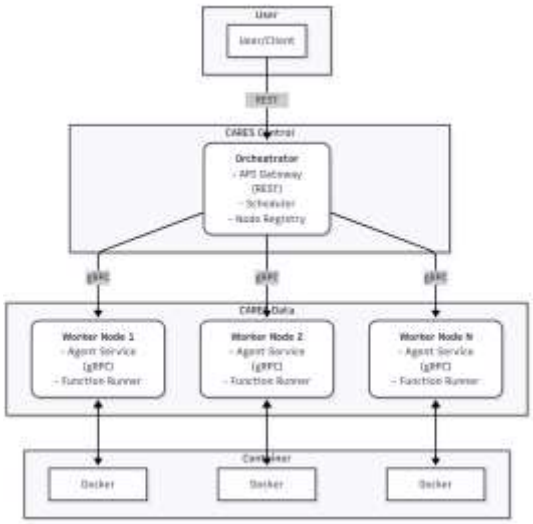
III. PROBLEM STATEMENT

Distributed systems face the NP-hard challenge of allocating heterogeneous resources efficiently across clusters. Traditional schedulers fail to account for real-time fluctuations, leading to resource fragmentation, bottlenecks, and inflated costs. CARES aims to resolve these issues by implementing a cost-aware scheduler that dynamically assigns tasks to optimal worker nodes using CPU and memory usage as core metrics.

IV. PROPOSED SYSTEM

CARES follows a master-worker architecture where the Orchestrator maintains a live node registry, executes scheduling logic, and exposes REST APIs. Worker nodes act as execution agents, reporting resource usage via gRPC and running containerized functions. A unique feature of CARES is its interactive TUI built with Bubble Tea, enabling real-time visualization of CPU/memory graphs, node health, and logs. CARES ensures:

- Cost-Aware Scheduling
- Real-Time Monitoring
- Fault Tolerance
- Container-Native Execution
- Transparent User Interfaces



V. METHODOLOGY AND IMPLEMENTATION

The CARES implementation leverages Go (v1.24.5) as its primary language, chosen for its strong concurrency model through goroutines and channels, which ensures high throughput and low latency in distributed environments. gRPC with Protocol Buffers is employed for inter-node communication, enabling bidirectional streaming between the orchestrator and worker nodes for metrics updates, heartbeats, and function execution requests. For execution, the system integrates seamlessly with the Docker Engine, which provides isolation, reproducibility, and rapid deployment of containerized functions.

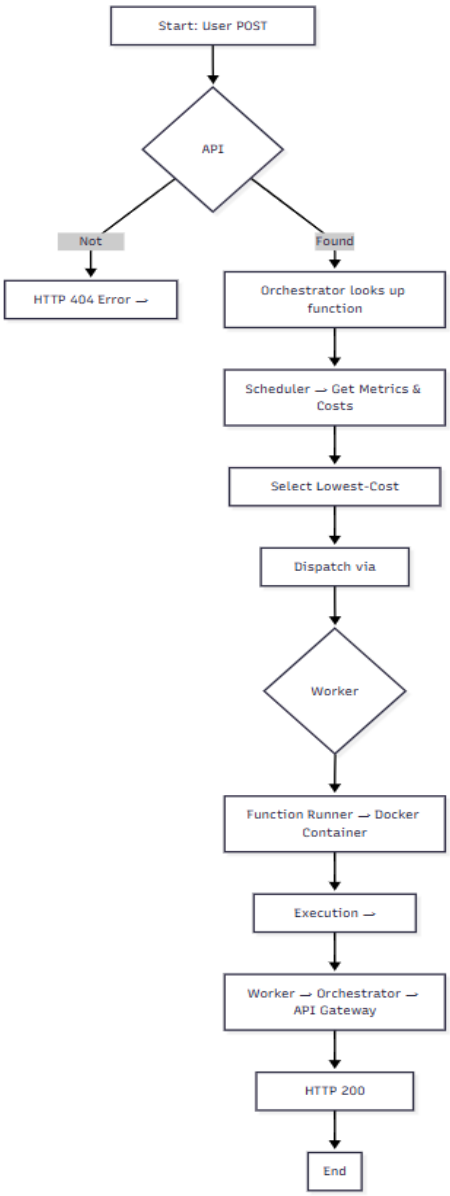
At the heart of the system lies the Scheduler, which evaluates real-time metrics collected from workers to determine the optimal node for executing a given function. The orchestrator maintains two registries: a Node Registry, which tracks active worker nodes and their health status, and a Function Registry, which stores metadata about uploaded functions. When a new execution request is received, the scheduler computes a weighted cost function:

$$\text{Cost}=(W_{cpu}\times\text{CPUUsage})+(W_{mem}\times\text{MemUsage})$$

Where W_{cpu} and W_{mem} are configurable weights that balance between CPU and memory utilization. This ensures that workload placement accounts for both computational and memory constraints, avoiding hotspots and underutilization.

Worker nodes run lightweight Agent Services that continuously collect resource metrics using the gopsutil library. These agents report CPU, memory, and system load information to the orchestrator at regular intervals, enabling the system to react quickly to changing resource conditions. Functions are executed inside Docker containers, with the Function Runner module managing lifecycle operations including image pull, container start, log capture, and cleanup.

The Terminal User Interface (TUI), built with Bubble Tea and styled using Lipgloss, provides real-time visualization of system performance. Features include dynamic ASCII graphs for CPU/memory usage trends, color-coded logs, interactive menus, and cluster topology displays. The TUI adapts to terminal resizing and supports responsive layouts, ensuring usability across environments. Together, these components enable CARES to deliver a lightweight yet powerful orchestration platform with strong transparency and cost-aware decision-making.



VI. RESULTS AND EVALUATION

CARES has been validated in a Docker-based multi-container environment. The cluster successfully executed distributed workloads with real-time monitoring. Results demonstrate low-latency scheduling, graceful handling of node failures, and transparent monitoring through the TUI. Key outcomes include:

- Efficient workload distribution based on real-time cost metrics
- Real-time visualization of CPU and memory usage
- Fault-tolerant cluster healing
- Low scheduling latency (<100 ms)

VII. CONCLUSION AND FUTURE WORK

CARES demonstrates a practical and educational approach to distributed orchestration. It integrates cost-aware scheduling, real-time monitoring, and containerized execution into a lightweight platform. The project proves the feasibility of balancing workloads intelligently without the overhead of larger orchestration frameworks. Future work includes integration with Prometheus for advanced monitoring, TLS for secure communication, and high-availability orchestration via replicated masters.

REFERENCES

[1] Verma, A., et al. “Large-scale cluster management at Google with Borg.” EuroSys, 2015.

[2] Hindman, B., et al. “Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center.” NSDI, 2011.

[3] Merkel, D. “Docker: Lightweight Linux Containers for Consistent Development and Deployment.” Linux Journal, 2014.

[4] HashiCorp. Nomad: A Flexible Orchestration System, 2015.

[5] Ongaro, D., and Ousterhout, J. “In Search of an Understandable Consensus Algorithm (Raft).” USENIX ATC, 2014.

[6] Google. gRPC: A High Performance, Open Source Universal RPC Framework.

[7] Sharma, S., et al. “A Survey of Load Balancing Algorithms in Distributed Systems.” IJCA, 2016.

[8] Bubble Tea Framework. Charm.sh. Available at: https://github.com/charmbracelet/bubbletea

[9] Gopsutil Library. Available at: https://github.com/shirou/gopsutil