POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Executive Summary of the Thesis

# Ermes: enabling Stateful Serverless at the Edge through a locality-aware Migration Policy

Laurea Magistrale in Computer Science and Engineering e High Performance Computing Engineering - Ingegneria Informatica ed Ingegneria del Calcolo ad Alte Prestazioni

**Authors:** Matteo Briscini, Matteo Cenzato

**Advisor:** Prof. Alessandro Margara

**Co-advisors:** Dario d'Abate, Arianna Dragoni

**Academic year:** 2024-2025

## 1. Introduction

While *Serverless Computing* [1] and *Edge Computing* [2] offer significant operational advantages, their unification within a *Computing Continuum* is hindered by the stateless nature of traditional FaaS and the resource constraints of edge devices. Current architectures often incur prohibitive latency due to a lack of locality-aware scheduling and the necessity of offloading state to remote storage.

To address these challenges, we propose **Ermes**, a distributed, state-aware FaaS platform designed to bridge the gap between local data generation and distributed execution. Our architecture is built upon four primary pillars: (i) **integrated state management**, (ii) **locality-aware scheduling**, (iii) **transparent state migration and replication**, and (iv) **lightweight isolation**. By internalizing state management and utilizing lightweight WebAssembly-based execution, Ermes enables low-latency, data-local processing across the continuum.

Our contribution centers on a dual-policy approach to the data placement problem. We first formulate a centralized *Integer Linear Programming (ILP)* model to establish a theoretical performance baseline for optimal function and state distribution. To ensure scalability in large-scale networks, we then propose a *decentralized migration policy* inspired by mechanical potential energy. This mechanism allows nodes to autonomously relocate state and logic toward high-demand areas, enabling the system to self-organize without a global orchestrator.

We implemented the Ermes platform and validated its performance through an extensive experimental campaign. Our results demonstrate that the decentralized policy closely approximates the mathematical optimum while maintaining robustness under heavy workloads. Furthermore, we show that Ermes significantly reduces user-perceived latency by transparently managing state migration during client mobility, outperforming traditional cloud-centric FaaS models.

## 2. Ermes Framework

We design Ermes as a modular architecture that enforces a strict separation between the management layer and the distributed execution layer. The management layer comprises two distinct

components: the *Group Token Provider* (GTP) and the *Ermes Function Registry* (EFR). The GTP handles access control via the Role-Based Access Control (RBAC) mechanism [5], utilizing defined policies to issue and renew cryptographically signed tokens. The EFR is responsible for function storage and versioning, maintaining serverless function binaries along with their corresponding metadata.
The distributed execution layer consists of multiple *Ermes Nodes* organized into a hierarchical multi-tier topology. We designed this layer to support the dynamic reallocation of data collections, enabling the system to migrate data across the network to optimize load distribution and client proximity. This capability is crucial for maintaining high availability and performance in heterogeneous and evolving environments.

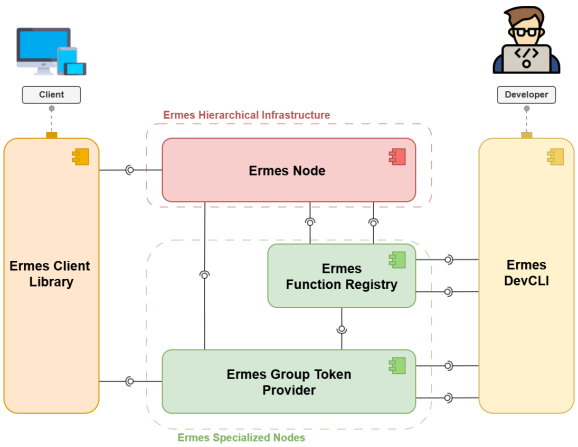


Figure 1: Ermes System Architecture.

## 2.1. Data Model

We define **function state** as the persistent data that survives across multiple invocations and is shared among users within the same group. This state relies on a backing storage system, which developers utilize to preserve application-level semantics. However, managing persistent state in a distributed environment introduces significant complexity, particularly regarding namespace collisions and data isolation. Furthermore, the system must efficiently distribute data across nodes, handle migrations in response to workload fluctuations, and ensure high availability.
To address these challenges, we partition the persistent state space into logical units termed **collections**. A collection aggregates related state elements and serves as the atomic unit for placement, migration, and replication across system nodes. To explicitly balance the trade-off between availability and consistency, we support two consistency models [6]. **Strong Eventual Consistency** guarantees that, in the absence of new writes, all replicas will eventually converge to the same state. **Sequential Consistency** enforces a global order on all write operations across replicas. We implement this model using a single-leader replication strategy, where all write requests are directed to a designated leader replica that asynchronously propagates updates to follower replicas.
To efficiently associate functions with their required resources, Ermes employs a metadata-based mechanism organized into two categories: *Function Metadata* and *Collection Metadata*. The former enables the resolution of dependencies between functions and data collections, ensuring accurate function-to-data mapping, while the latter governs the collections themselves, defining access control policies and supporting lookup operations. To manage these dependencies, we employ *Metadata Queries*. Since the specific data required for an execution may not be known until runtime, we introduce *Query Templates*, which allow data dependencies to be resolved dynamically upon function invocation.

## 2.2. Execution Model

We decompose the function lifecycle into three distinct logical phases. First, the *authentication and authorization* phase restricts function invocation and state access to verified entities. We delegate user authentication to an external Identity Provider (IdP), while we manage authorization internally. Second, the *Metadata Lookup* phase efficiently resolves resource dependencies via the Metadata Queries mechanism, leveraging caching to minimize latency. Finally, the *Scheduling* phase prioritizes data locality to minimize network latency associated with remote collection access. We initiate the scheduling process once the system identifies the required data collections, organizing it into three sub-phases. First, *Data Discovery* selects a suitable node for execution based on data proximity. This phase leverages the hierarchical network structure to determine data location, relying on the assumption that each node maintains knowledge of the data collections hosted within its descen-

dants' subtrees. Second, *Function Execution* involves the actual execution of the FaaS function on the selected node. Third, *Data Propagation and Consolidation* manages state consistency. We employ a write-back caching mechanism for write operations; during this phase, updates are consolidated in the local persistent storage and subsequently propagated upward and downward through the hierarchy to synchronize remote replicas.

## 3. Data Placement

The Ermes framework integrates function execution with data collection management, supporting stateful invocations across network nodes under diverse consistency models. To optimize system performance, we propose a *collection placement strategy* designed to minimize function execution time and end-to-end latency. We explore both centralized and decentralized solutions: the former utilizes an Integer Linear Programming (ILP) model, while the latter is based on a physical model of mechanical potential energy minimization.

### 3.1. Centralized Solution

We formalize the optimal data placement problem by extending the ILP formulation for function offloading proposed by Nardelli et al. [3]. Our approach adapts this model to the architectural requirements of Ermes, specifically targeting scheduling policies for sequential and strong eventual consistency to enable dynamic function dispatching. This formulation identifies a globally optimal placement strategy that minimizes a multi-objective cost function:

$$\min \mathcal{F} + \mathcal{C}$$

where $\mathcal{F}$ represents total function execution time, composed by the time to perform the request, executing the function on a node, and potentially performing remote calls to retrieve data, and $\mathcal{C}$ is a regularization term which discourages degenerate solutions that might utilize excessive system-wide resources.
Our analysis first introduces a baseline model without replication, requiring the co-location of functions and their associated data. This model defines the core system parameters, including latencies, collection dimensions, and execution times, and the primary decision variables for the final placement of collections and functions. Given the absence of replication, we enforce a uniqueness constraint on each collection.
We subsequently extend this formulation to incorporate replication and consistency-specific constraints. For Strong Eventual Consistency, we permit remote collection access and relax the uniqueness constraint by requiring that the collections are not completely destroyed. Conversely, for Sequential Consistency, the Ermes policy precludes remote execution and necessitates a dedicated decision variable to designate a unique collection leader for handling write operations.

### 3.2. Distributed Solution

The ILP formulation defines an optimal centralized solution. However, implementing such an approach requires capturing a consistent global view of the system state and aggregating this information at a central orchestrator. These processes are computationally intensive and incur prohibitive network overhead. Moreover, the ILP problem can become computationally expensive to solve at scale. These factors render the centralized solution impractical, motivating the need for a practical, decentralized approach where nodes make independent placement decisions based on limited, partial knowledge of the system state.

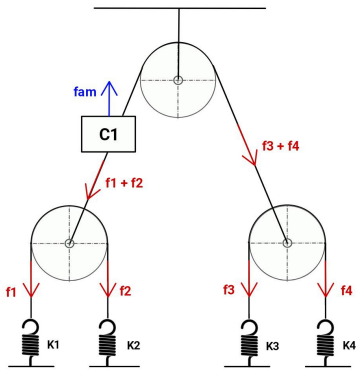


Figure 2: Physical representation of a 3-node subtree.

Our solution aims to store data in close proximity to the users accessing it. Strong Eventual Consistency inherently enables aggressive replication strategies, as it allows both read and write operations to be served by any available replica. Consequently, these properties allow

us to treat data placement as a **distributed caching problem**.
Conversely, for Sequential Consistency collections, our objective is to strategically place the leader and its followers to minimize the average user-perceived latency for both read and write operations. This requirement introduces a complex state space, rendering simple caching heuristics insufficient. To address this complexity, we formulate the data placement problem using a **physical model** inspired by [4], which is analogous to a system of ropes and pulleys as depicted in Figure 2. In this model, we represent the user demand for data proximity as an **elastic force**, while we model the network's objective to prevent node overload and preserve available space within node memory using **buoyant forces**. In a physical system, an object moves spontaneously if that movement causes a decrease in its **potential energy**. Equilibrium is reached when no available moves can further reduce $E_{total}$. Accordingly, the objective function is defined as:

$$\min(E^{W}_{elastic} + \sum_{n} E^{R}_{elastic_n} + \sum_{m} E_{hydrostatic_m})$$

This physical model provides an intuitive, continuous description of the competing forces that govern optimal data placement within a hierarchical network. From the objective function, we derive practical, distributed decision policies for sequentially consistent replica placement. These policies consist of two decision rules that each node applies using locally available information to determine whether to migrate a leader or a replica along an adjacent link. Specifically, **a node evaluates whether a proposed move yields a net reduction in the total potential energy of the system**. Consequently, delegating the leader or a replica is considered advantageous iff the energy benefit outweighs the associated energy cost.

# 4. Ermes Implementation

We implemented the core components of the Ermes framework in Go. Go's concurrency model, based on lightweight goroutines and channels, simplifies the implementation of scalable parallelism and synchronization. Furthermore, the Go standard library offers robust support for networking and I/O, which is essential for our communication-intensive components. For the execution of user-defined functions, our platform requires a runtime that supports multiple languages while guaranteeing security, efficiency, and strong isolation. To meet these goals, we selected *WebAssembly* (WASM) as the core technology for our function execution engine. WASM provides a lightweight, sandboxed, and platform-independent environment that aligns with the requirements of a geo-distributed serverless architecture. For persistent data storage, we selected *Redis*, a hybrid key-value store that enables low-latency operations and naturally supports horizontal scalability, efficient partitioning, and high availability.

## 4.1. Dynamic Extension of the Data Placement Protocol

Our distributed data placement strategy results in a dual-policy protocol: a streamlined "always-copy" policy for Eventually Consistent collections and a stricter policy for Sequentially Consistent collections that adheres to the energy minimization objective. While the physical model computes an instantaneous solution based on a static system representation, real-world operations require adaptation to temporal variations in access patterns and resource availability. To address this, we design a distributed protocol that executes periodically. Although we aggregate access requests in real-time as they traverse the network topology, we restrict the exchange of node state information to specific intervals to minimize network overhead. Each node operates according to a three-state cycle: *Idle*, *Gossip*, and *Decision*. A complete iteration of these states constitutes a single time window, or *epoch*. Within each window, we aggregate metrics such as request counts and cumulative latency. To base placement decisions on stable trends rather than short-term fluctuations, our protocol considers a history spanning multiple windows. We employ a configurable weighting function to decay the influence of older data, thereby prioritizing recent access patterns.

# 5. Evaluation

To evaluate our framework, we first analyze the scalability of the ILP model in a simulated environment to identify the computational constraints that necessitate a distributed approach.

Subsequently, we assess the Ermes framework's distributed data movement and replication policy, validating its behavior against the optimal solutions produced by the ILP model. Our evaluation focuses on key system requirements, primarily user-perceived latency and responsiveness under dynamic workloads.

## 5.1. Mathematical Model Scalability

We implemented the replication models in C++ using the *IBM CPLEX* Concert Technology to empirically assess their scalability, by framing them as multi-objective optimization models. Our analysis initially focused on the impact of variable reduction, demonstrating that a naive formulation results in quadratic memory scaling. This prevents the construction of large-scale problem instances and significantly increases solution time.

Further investigation into system variables, specifically the number of nodes, collections, and function invocations, reveals that solution times scale linearly with a small number of parameters, increasing in asymptotic complexity with bigger instances. Although the model theoretically scales linearly with the number of functions and collections, we observed significant coupling between these variables within the Sequential Consistency Model due to Ermes's strict function placement policies, leading to quadratic scaling with high function invocation density and a low number of collections in the system.

Under heavy workloads, in small-scale networks, solution times range from 10 to 80 seconds, and in larger networks the solving algorithm can exceed 10 minutes and almost completely exhaust the memory of a high-performant server machine (64 GB). In edge-cloud environments, such delays lead to state obsolescence, where the system state changes faster than the solver can reach an optimal solution. For larger networks, solution times exceeding 10 minutes further justify the adoption of our decentralized orchestration policy.

## 5.2. Framework Evaluation

To assess the Ermes framework and its distributed data placement solution under realistic conditions, we deployed a virtualized testbed based on Proxmox, configuring fixed Round-Trip Times (RTT) across network links to ensure experimental reproducibility. We structure our evaluation into two distinct phases to systematically analyze the performance implications of our design decisions: (i) **Use-case-driven experiments**, designed to evaluate the distributed migration policy and verify its convergence with the ILP solution; and (ii) **Ablation studies under stochastic access patterns**, aimed at quantifying the performance impact of specific architectural features

In the use-case-driven experiments, we ground our methodology in a representative operational scenario for the Ermes framework. We design a series of five deterministic scenarios. The primary objective is to assess the *convergence time*, and to analyze *transient behaviors* by contrasting user-perceived latency and message overhead during *Unbalanced Epochs* with the metrics observed during *Balanced Epochs*.

Our experiments demonstrate a strong correspondence between the equilibrium state reached by the implementation and the theoretical optimal configuration. Furthermore, we observe that the convergence time depends directly on the network distance (in hops) between a collection's initial location and its optimal placement. Regarding communication overhead, our results indicate that update propagation and offloaded function invocations constitute the primary sources of network traffic. In contrast, the gossip mechanism, essential for exchanging resource usage statistics and enforcing placement policies, accounts for a negligible fraction of the total message volume.

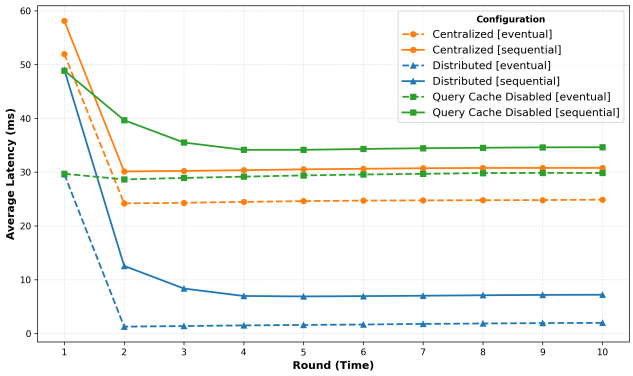


Figure 3: Ablation Results (1024 Clients, 100 Collections)

Subsequently, we conduct an ablation study to systematically assess system performance, scal-

ability, and robustness under stochastic operational conditions. We define these conditions by modulating the number of clients and collections to vary workload intensity. Initially, we establish a performance baseline using the complete Ermes framework. We then selectively disable two performance-critical architectural mechanisms to emulate the functional constraints typical of alternative state-of-the-art solutions: the distributed data storage on *Ermes Nodes* (replaced here by a centralized external storage to benchmark against a traditional stateless architecture) and the client-side cache for metadata query results.

The evaluation of the Ermes baseline identifies the consistency policy as a determining factor in system performance and convergence dynamics, with collections requiring weaker consistency guarantees exhibiting faster convergence times and lower steady-state latencies. Figure 3 illustrates the temporal evolution of the system across the three analyzed configurations, differentiating based on the consistency guarantees of the involved collections (with time quantized into 15-second rounds). These results empirically demonstrate the significant performance advantage of the Ermes framework compared to the ablated variants. Specifically, the Ermes baseline exhibits steady-state latency values in the range of 2–8 ms, whereas the ablated configurations show latencies between 20 ms and 35 ms. This finding underscores the synergistic relationship between client-side metadata caching and decentralized data placement, confirming that both components are essential to achieve optimal system performance. Notably, this performance gap persists even under increased memory pressure induced by a larger number of collections and clients, which leads to memory saturation on Edge nodes.

Finally, we evaluate the system's robustness against high user mobility, modeling this scenario as a periodic, synchronized round-robin migration of all clients across Edge nodes every 5 rounds. This configuration represents a worst-case operational scenario, demonstrating the capability of Ermes to recover from perturbations to the equilibrium state. The results presented in Figure 4 exhibit intervals of increased latency, corresponding to the performance degradation induced by the synchronized migration events. Notably, the local maxima observed during these perturbation phases remain significantly lower than the global maximum associated with the initial cold-start phase. Furthermore, consistent with the trade-off between consistency and availability, collections with weaker consistency guarantees exhibit faster recovery.

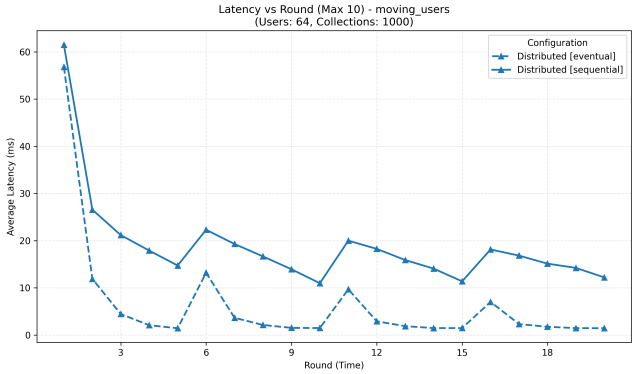


Figure 4: Client Mobility (64 Clients, 100 Collections)

## 6. Conclusions

In this thesis, we investigated the extension of stateful serverless computing to the network edge, a domain where latency, resource constraints, and data locality are primary concerns. The central contribution of this work is a distributed, physics-inspired data placement policy that dynamically manages both data migration and replication. We grounded this policy in a formal mathematical model, formulated as an ILP problem.

Our experimental evaluation empirically demonstrates that the distributed placement mechanisms successfully approximate the optimal solutions derived from the ILP model, when operating under realistic constraints and without centralized coordination. The results confirm that co-locating state with computation at the edge significantly reduces user-perceived latency.

## References

[1] Yongkang Li, Yanying Lin, Yang Wang, Kejiang Ye, and Chengzhong Xu. Serverless computing: State-of-the-art, challenges and opportunities. *IEEE Transactions on Services Computing*, 16(2):1522–1539, 2023.

[2] Redowan Mahmud, Kotagiri Ramamohanarao, and Rajkumar Buyya. Application

management in fog computing environments: A taxonomy, review and future directions. *ACM Comput. Surv.*, 53(4), July 2020.

[3] Matteo Nardelli and Gabriele Russo Russo. Function offloading and data migration for stateful serverless edge computing. In *Proceedings of the 15th ACM/SPEC International Conference on Performance Engineering*, ICPE '24, page 247–257, New York, NY, USA, 2024. Association for Computing Machinery.

[4] P. Pietzuch, J. Ledlie, J. Shneidman, M. Roussopoulos, M. Welsh, and M. Seltzer. Network-aware operator placement for stream-processing systems. In *22nd International Conference on Data Engineering (ICDE'06)*, pages 49–49, 2006.

[5] Ravi S. Sandhu. Role-based access control. Advances in Computers. Elsevier, 1998.

[6] Paolo Viotti and Marko Vukolić. Consistency in non-transactional distributed storage systems. *ACM Comput. Surv.*, 49(1), June 2016.