POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Executive Summary of the Thesis

From the Edge to the Cloud, A New Approach To Hybrid Computing

Master of Science in Computer science and Engineering

Author: Leonardo Barilani, Giampietro Fabrizio Bonaccorsi

Advisor: Prof. Alessandro Margara

Co-advisor: Gianpaolo Cugola

Academic year: 2022-2023

1. Introduction

The rapidly evolving landscape of technology and data-driven industries has given rise to the transformative paradigm of **Edge Computing**. This approach brings computation and data storage closer to the location where it is needed, enabling faster processing and reduced latency compared to traditional **Cloud Computing**.

1.1. Definition and Core Principles of Edge Computing

Edge Computing involves processing data at or near the edge of the network, such as routers, gateways, or cloudlets, rather than relying on a centralized cloud infrastructure. This proximity to data sources allows for real-time processing and reduced reliance on data transfer.
The explosive growth of the **Internet of Things** (**IoT**) and the increasing need for low latency and real-time decision-making drive the adoption of Edge Computing. With the rise of autonomous vehicles, smart cities, and industrial automation, processing data at the network edge becomes crucial for efficient data analysis and decision-making.
The proliferation of **IoT** devices has led to a surge in data generation, posing challenges to traditional data centers in terms of bandwidth and computational capacity. Edge Computing presents a practical solution by leveraging the computational power of edge devices and minimizing data transfer to distant cloud servers.
While Edge Computing holds immense promise, it also presents challenges such as managing distributed computing infrastructure, ensuring data security and privacy, and dealing with heterogeneous edge devices. However, these challenges drive innovation and investment in edge-specific hardware, software, and approaches.

1.2. Scope of this Thesis

This thesis aims to investigate the development and management of edge resources in Edge Computing. By addressing the identified challenges, innovative solutions and enhancements will be proposed, contributing to the advancement of the field.

1.3. Research Methodology

The research methodology followed a systematic approach, including a literature review, analysis of available frameworks, design of a novel solution, and evaluation through prototype imple-

mentation and simulations. The goal was to address those challenges by developing a stateful **Function-as-a-Service (FaaS)** framework for Edge Computing and measure key metrics such as latency and bandwidth utilization.

2. State of the Art and Scientific Literature Review

Through a comprehensive review of recent research papers, focusing on those presented at the **IEEE International Conference on Fog and Edge Computing** (**ICFEC**), we have identified key points that shape our proposed framework.

One of the primary focuses of our work is to enable "serverless" computing on the edge. **Serverless computing** has emerged as an abstraction that frees developers from infrastructure management concerns, allowing them to focus solely on business logic implementation. The Serverless model works effectively on the **Cloud**, where homogeneous and readily available resources facilitate immediate scalability and cost-efficiency. Nastic et al. in their article "A Serverless Real-Time Data Analytics Platform for Edge Computing" [6] shed light on the limitations of current approaches for data analytics on the edge, which often require developers to employ ad hoc solutions tailored to the available infrastructure.

Enabling serverless computing on the edge is most effectively achieved through a "Function as a Service" (**FaaS**) platform. While FaaS excels at executing stateless functions, supporting stateful constructs and distributed transactions across multiple functions presents a challenge. Accessing remote state introduces latency and network traffic, limiting the effectiveness of FaaS in Edge Computing. Therefore, the implementation of a stateful FaaS framework becomes necessary and serves as the main focus of our thesis.

2.1. Main Focuses of Our Research

Our in-depth analysis of scientific literature has revealed several other key problems and limitations that require attention:

- **Inefficient resource utilization**: Edge devices often have limited computational power, memory, and reliability, necessitating resource optimization techniques to ensure efficient utilization. Additionally, intelligent data location and migration strategies are essential to maximize resource efficiency [4].
- **Lack of data and computation locality**: Current frameworks do not adequately consider the proximity of data and computation, leading to suboptimal latency and bandwidth utilization.[7]
- **Challenges in distributed transactions**: Coordinating distributed transactions across Serverless Stateful functions poses difficulties in managing state, ensuring consistency, and coordinating transactions among functions.[1]
- **Metadata exchange and session tracking**: Efficient resource discovery and service provisioning in Edge Computing can be improved through peer-to-peer metadata exchange and decentralized session tracking mechanisms [5].
- **Hierarchical representation and resource management**: Organizing edge nodes in a hierarchical structure enables effective management of edge resources, reduces communication latency, and improves the overall performance of Edge Computing systems.[3]
- **Seamless migration of edge services**: Existing solutions for edge service migration lack flexibility and fail to consider optimal resource allocation and computational offloading across edge nodes.[2]

3. Existing Solutions

This section examines the current landscape of serverless computing frameworks, evaluating their stateful support capabilities and identifying their limitations. We also explore popular open-source FaaS platforms and discuss their relevance to our research.

3.1. Hosted Serverless Edge Platforms

Serverless computing frameworks offered by industry leaders, such as **AWS Lambda@Edge** and **Akamai EdgeWorkers**, provide edge computing capabilities but primarily focus on stateless and caching use cases. While they offer some stateful support through key-value databases, their capabilities are limited compared to other solutions. Cloudflare Workers, on the other

hand, provide robust stateful support through **Cloudflare Durable Objects**, making them ideal for applications requiring frequent read operations and managing stateful sessions. We drew inspiration from the offloading capabilities of Durable Objects, which primarily focus on relocating data within the CloudFlare network. Our framework leverages the concepts used by Durable Objects to manage stateful sessions, ensure transaction guarantees, and facilitate data migration, adapting them to the deployment in an Edge environment.

3.2. FaaS Platforms

In our exploration of FaaS platforms, we studied Apache OpenWhisk, Fission, OpenFaas, and other platforms. Apache OpenWhisk offers extensive language support but introduces significant overhead due to its architecture, making it unsuitable for edge deployments. Fission provides low-latency and auto-scaling capabilities but lacks native support for stateful functions. OpenFaas offers scalability and resource-constrained device support, making it a viable choice for edge computing, but also lacks built-in support for stateful functions. Other platforms, while mentioned briefly, are not extensively discussed due to limited availability and support.

3.3. Limitations and Missing Features

Existing serverless frameworks for edge computing lack comprehensive stateful support for use cases involving frequent write operations. Moreover, the absence of lightweight FaaS frameworks capable of handling stateful functions poses a significant challenge.
Our research aims to address these limitations by modifying OpenFaaS to deploy stateful functions on the edge.

4. Design of the Solution

We have developed a comprehensive framework for stateful function development and deployment. Our framework enables the seamless creation and deployment of stateful functions across a hierarchical network infrastructure, ranging from edge devices to the cloud. This architecture facilitates automatic session offloading to higher-level nodes or onloading to lower-level nodes based on resource availability.

4.1. Framework Features

Our framework offers the following features:
- **Writing stateful functions with different characteristics**: Developers can create blocking functions with read/write permissions on sessions and non-blocking functions with read-only permissions on sessions.
- **Defining a network of nodes**: Developers can specify a network of nodes to deploy stateful functions.
- **Session management**: Sessions hold key-value and key-list data.
- **Automatic offloading/onloading**: Sessions are automatically offloaded or onloaded based on resource availability.
- **Custom offloading/onloading**: Developers can implement their own offloading/onloading strategies using our offload/onload APIs.
To implement these features, we have addressed various aspects such as handling incoming connections, session management, triggering offload/onload, ensuring data consistency during migration and parallel accesses, and implementing garbage collection.

4.2. Data Consistency

Data consistency is essential for stateful functions. We address this by implementing a **client-centric consistency model** and **ensuring atomic writes** during function execution.
Clients generate unique identifiers (UUIDv4) for each request to **guarantee at-most-once execution** and **prevent duplicates**. The server-side mechanism checks if a request with the same UUID has been completed before, preventing duplicate execution.
During function execution, read and write operations are performed on a local cache, improving response times. **Write operations are atomically executed** on the session's storage **at the end of execution**. This ensures that all writes are completed or none of them, preventing partial writes that could corrupt the state of the session.
We have integrated a locking mechanism into our framework to prevent potential inconsistencies from concurrent requests. When a request is received, a lock is created and associated with the SESSION-ID. At the end of function execution, the lock is released, allowing other func-

tions to access the session.
To prevent deadlocks caused by interrupted function execution, we introduced a timeout feature for locks. If a lock is not released within the specified timeout period, it is automatically released.
To ensure the correct lock is released, we associate a random value with each lock when it is acquired. Functions must provide the randomly generated value to unlock the session. If the value matches, the session is updated, and the lock is released.

4.3. Garbage Collection

Our framework includes a garbage collection mechanism to free unused memory on resource-constrained devices. We implement implicit collection by setting an expiration timeout for sessions. If a session remains inactive beyond the timeout duration, it is considered expired and can be eliminated. A garbage collector component is invoked periodically to check and delete expired sessions.

4.4. Offloading and Onloading

The core functionality of our framework is the automatic offloading and onloading of sessions across the network infrastructure. This mechanism maximizes resource utilization and ensures efficient use of available resources while maintaining transparency for developers and clients.
The objectives of this feature are:

- Move sessions up the hierarchy (offload) or down the hierarchy (onload) based on resource availability.
- Guarantee data consistency during migration.
- Transparent redirection for clients.

To achieve these objectives, the framework executes the following operations for offload and onload:

1. Nodes express the need for offload or the availability for onload, and negotiate the migration of a session.
2. Both nodes create a local lock on the session to render it unavailable.
3. The session is migrated between nodes.
4. Locks are released, and requests are redirected to the appropriate node.

4.4.1 Offloading

Offloading involves moving one or more sessions from a lower-level node to an upper-level node in the infrastructure. This process frees up resources on lower-level nodes and leverages the computational power of higher-level nodes, potentially improving function execution performance. While offloading may introduce slight latency due to increased distance from the client, the benefits outweigh this trade-off.
Our framework focuses on moving sessions and redirecting clients to the appropriate node. Horizontal scaling of function replicas within each node is delegated to the underlying infrastructure (e.g., Kubernetes). To minimize cold starts after offloading to upper-level nodes, a minimum number of replicas (typically one) can be set on those nodes. Alternatively, to free up resources on underutilized upper nodes, the minimum replicas can be scaled down to zero, resulting in longer cold start times.
To ensure session integrity during migration, we use a locking mechanism as explained before. The lock is applied to the session in both the lower and upper nodes to prevent access during offloading and protect the session in case of a previous onload operation.
If an upper-level node is overwhelmed and rejects an offloading request, the request can be redirected to a higher-level node in the hierarchy. This approach avoids additional migrations and maintains scalability and resource management.

5. Experimental Evaluation

In this section, we present the experimental evaluation of our framework, focusing on its key goals: scalability, interoperability, and orchestration management. We present benchmark results and real use cases to demonstrate the achievements of our framework in the field of Edge Computing.

5.1. Functional Testing and Evaluation

In order to simulate different scenarios and evaluate the performance of our framework, various tests were conducted using different types of functions. Here is a breakdown of the function types used and their purposes in the testing:

1. **Empty Function**: This function was used to benchmark the latencies and costs introduced by our framework without being impacted by the computational cost of the function itself. It simply returns an HTTP 200 response.
2. **Non-blocking Function (CDN Download)**: This function was used to test the performance of handling multiple parallel accesses to the same node within the same session. The function's purpose is to allow users to download a requested file and stream it if it's a video file. This test helps evaluate the efficiency of our framework in handling concurrent read-only requests.
3. **Blocking Function**:
   (a) **CDN Upload**: This function was used to test the performance of blocking functions, which have both read and write permissions and lock the session during their execution. The purpose of the CDN Upload function is to save the content of the body received into a Redis instance and associate that content with the body contained in the Session header. This test allows us to assess the performance of our framework in handling write-intensive tasks and concurrent requests that require session locking.
   (b) **Speech to Text**: This function serves as an example of a computationally expensive function. It translates speech to text and stores the result in the history associated with the Session header. This test demonstrates the performance of our framework in executing complex computational tasks and managing the associated session data.

5.2. Average Offloaded Response Time

We conducted two experiments to assess the impact of offloading from the client's perspective. In both experiments, we tested the continuous interaction between a client and a node. During the message exchange, an offload was triggered, resulting in the migration of the client's associated sessions to another node.
The two experiments differed in terms of the environment. In the first simulation, there were no other clients connected to the node, resulting in optimal response times. Additionally, the edge node exhibited faster responses compared to the cloud, leveraging its advantage of low latency.
The second experiment showcases the fluctuation of latency in a scenario with an increasing number of concurrent client accesses to the same node. Similar to the previous experiment, a client continuously sends identical requests to the node. However, in this experiment, the number of additional clients connecting concurrently to the node gradually increases from 0 to 30, added sequentially between the 40th and 140th seconds.
Graph 1 shows that the average response time is affected by this increased load, up until the point where the offload is triggered. After this, the session that the client is accessing is moved to an upper node, with superior computational capabilities. This results in a lower, more stable response time, despite the latency being higher compared to the edge node. The graph effectively demonstrates the effectiveness of our framework in maintaining a low response time and adaptively responding to changing environmental conditions.

6. Performances in real use cases

While much of the existing literature on offloading functions to the edge focuses on methodologies, innovative approaches, and benchmarks to evaluate the feasibility of these methods, not many real-world use cases have been implemented or tested. This is primarily because the key objective of a FaaS framework on the edge is not to create new use cases or solve new problems. Instead, the goal is to extend the vast array of existing use cases currently deployed on the cloud, using existing FaaS frameworks, on the edge.
For this reason, our testing prioritizes benchmarking real scenarios with diverse settings, including different hardware capabilities and varying numbers of concurrent client access. By repeating the same tests across these varied scenarios, we aim to demonstrate the resilience and robustness of our framework in addressing the potential challenges an edge node may encounter. Rather than implementing a multitude

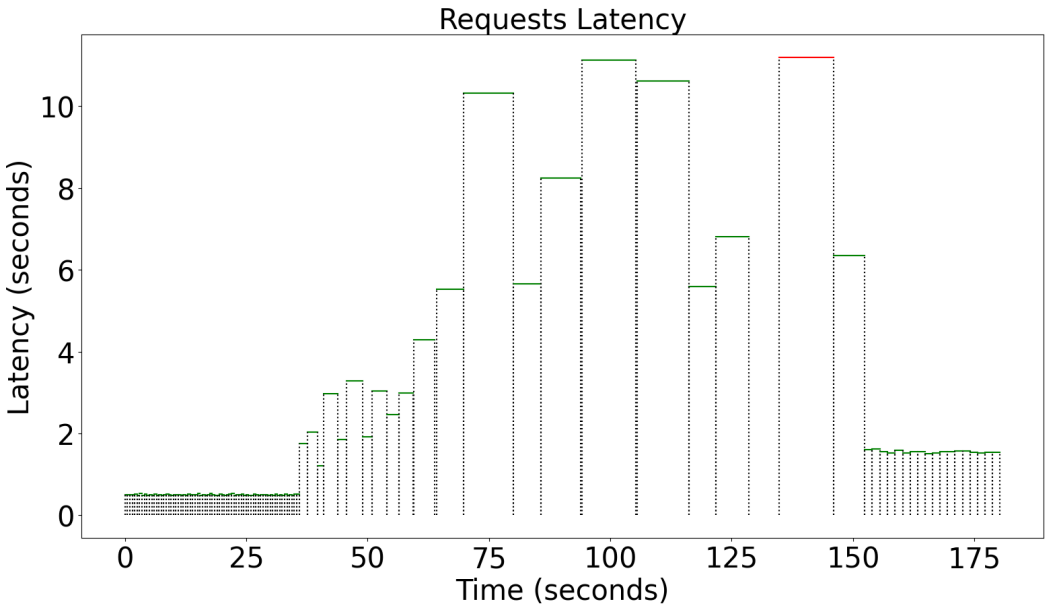

Figure 1: Experiment conducted to show the impact of an increasing number of clients

of different functions or use cases—which would not provide significant insight into the capabilities of our framework—we have chosen to concentrate on implementing a select few use cases and testing them rigorously.

6.1. Average Response Time for Concurrent Accesses

Our framework is designed to mitigate the constraints inherent in edge computing. The average response time naturally increases with the increase in parallel clients until it reaches a threshold. This threshold is contingent on the specific characteristics of the executed function and the technical specifications of the node under consideration. Once the node's resources are exhausted, the average response time tends to infinity, resulting in unserved or indefinitely delayed client requests. In such scenarios, cloud solutions can take advantage of their virtually limitless resources and autoscaling capabilities to dynamically reallocate resources for the task, ensuring continued service with minimal latency. We aim to demonstrate that our innovative approach to edge computing can effectively manage a real use case. By replacing horizontal scaling in the cloud with vertical offloading on the edge, we aim to optimize network resource utilization and maintain low client response times. To conduct this test, we used a local node and a

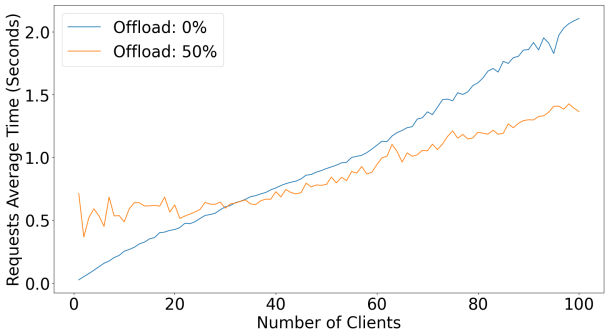

Figure 2: Experiments conducted to evaluate the average response time by increasing clients for the CDN Download function

cloud instance. The cloud instance maintained a fixed amount of 5 replicas for the function. The edge instance had a single fixed replica for the function, and it was limited to the use of a single CPU. We conducted multiple simulations with an increasing number of clients. Each client made 10 requests per simulation, and we averaged the response time per request obtained by each client across the 10 requests.
In graph 2, the blue line represents the average response time on a single node, without any offloading, while increasing the number of parallel clients connected to it. The orange lines depict a simulation wherein half of the sessions are consistently offloaded to an upper node. As

evident, by distributing connections evenly between two nodes (by offloading half of the sessions to an upper node), we can serve a larger number of clients while maintaining a lower response time. Initially, the response time in case of an offload is higher due to increased latency. Hence, offloading sessions when there are fewer connected clients doesn't make sense, as the edge is perfectly capable of managing all the clients independently while maintaining low latencies. The intersection point marks the stage where offloading begins to be beneficial. As such, our framework only initiates offloading when computational limits are reached.

7. Conclusion

In this master thesis, we have proposed and developed a framework for location-aware and stateful serverless computing on the edge. The framework aims to address the challenges and limitations of edge computing by leveraging existing frameworks such as Kubernetes and OpenFaaS, while introducing new functionalities specific to edge environments.

Throughout the thesis, we have defined the objectives of the framework, which include scalability, interoperability, and consistency. We have achieved these goals by implementing a centralized approach to deploying and managing the infrastructure, ensuring compatibility within a multi-vendor environment, and creating an environment where applications can be developed once and deployed anywhere.

The experimental evaluation conducted on the framework has demonstrated its effectiveness, efficiency, and usability. We have conducted benchmarks to evaluate performance metrics such as redirect time, offloaded response time, and offload time. The results of these benchmarks have validated the capabilities of the framework in handling offloading, managing session data, and optimizing network resource utilization. Additionally, real use cases have been implemented and tested, showcasing the framework's ability to handle various scenarios and maintain low response times even with increasing client accesses.

The achievements of this research provide a significant contribution to the field of edge computing. By combining the advantages of edge and cloud computing, our framework offers an optimized solution for deploying serverless functions in edge environments. It allows for efficient resource utilization, reduces bandwidth usage, and ensures low-latency execution of functions.

References

[1] Martijn de Heus, Kyriakos Psarakis, Marios Fragkoulis, and Asterios Katsifodimos. Distributed transactions on serverless stateful functions. In *Proceedings of the 15th ACM International Conference on Distributed and Event-Based Systems*, DEBS '21, page 31–42, New York, NY, USA, 2021. Association for Computing Machinery.

[2] Paulo Souza Junior, Daniele Miorandi, and Guillaume Pierre. Good shepherds care for their cattle: Seamless pod migration in geo-distributed kubernetes. In *2022 IEEE 6th International Conference on Fog and Edge Computing (ICFEC)*, pages 26–33, 2022.

[3] Vasileios Karagiannis and Stefan Schulte. Comparison of alternative architectures in fog computing. In *2020 IEEE 4th International Conference on Fog and Edge Computing (ICFEC)*, pages 19–28, 2020.

[4] Ivan Lujic, Vincenzo De Maio, and Ivona Brandic. Efficient edge storage management based on near real-time forecasts. In *2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC)*, pages 21–30, 2017.

[5] Ilir Murturi, Cosmin Avasalcai, Christos Tsigkanos, and Schahram Dustdar. Edge-to-edge resource discovery using metadata replication. In *2019 IEEE 3rd International Conference on Fog and Edge Computing (ICFEC)*, pages 1–6, 2019.

[6] Stefan Nastic, Thomas Rausch, Ognjen Scekic, Schahram Dustdar, Marjan Gusev, Bojana Koteska, Magdalena Kostoska, Boro Jakimovski, Sashko Ristov, and Radu Prodan. A serverless real-time data analytics platform for edge computing. *IEEE Internet Computing*, 21:64–71, 01 2017.

[7] Carlo Puliafito, Claudio Cicconetti, Marco Conti, Enzo Mingozzi, and Andrea Passarella. Stateful function as a service at the edge. *Computer*, 55(9):54–64, 2022.