Distributed Resource Orchestration at the Edge Based on Consensus

Gabriele Morabito[1,*], Annamaria Ficara[1], Antonio Celesti[1] and Maria Fazio[1]

[1]University of Messina, Messina, Italy

Abstract

This paper presents a consensus-based distributed solution for orchestrating microservices at the Edge. Our approach aims to distribute workloads among Edge nodes while ensuring trust and, at the same time, robust decision-making in a distributed log system. The proposed framework enhances the benefits of distributed systems for functionalities traditionally centralized, enabling trust through the use of consensus. We present the details of our solution and then we evaluate its performance.

Keywords

Orchestration, Microservices, Edge Computing, Consensus, Raft

1. Introduction

Edge Computing is a key technology to push the processing of huge amount of data close to the data sources, spreading computation and storage functionalities across wide information systems. By managing data locally, Edge Computing reduces latency and bandwidth consumption, improving reliability against network issues and efficiency in (near)real-time services. Its decentralized approach also adds an extra layer of security in terms of data privacy, as data can be processed and stored locally, reducing the risk of data breaches during transmission. However, Edge Computing presents some specific challenges related to the limited resources of Edge devices and security issues due to its inherent distributed infrastructure. Unlike the Cloud, where computing nodes are packed into one or more Data Centers under the control of the provider, at the Edge, the proliferation of heterogeneous devices creates a larger attack surface, making it more difficult to protect against threats. In addition, Edge devices are often deployed in remote locations, making them susceptible to physical tampering and theft. Such challenges affect the implementation of management services, such as maintenance, monitoring, scalability, and connectivity. In this paper, we specifically investigate strategies to implement distributed orchestration of microservices at the Edge taking into consideration the above peculiar issues. Our ambition is to ensure that nodes coordinate their activities in a trusted environment, where the orchestration decisions are achieved by a group of nodes, even in the face of failures or uncertainties. To this aim, we make use of a consensus algorithm to create a foundation of trust by establishing a shared understanding of the system state and ensuring the integrity of data. This paper introduces a consensus-based distributed framework for orchestrating microservices at the Edge, where all the nodes share a unified view of the system's state. The Edge nodes form a trusted system able to deploy microservices according to reliability, consistency and security principles. In particular, we leverage the Raft consensus algorithm to distribute workloads among Edge nodes while ensuring robust decision-making in a distributed log system. In this paper, a 'trusted environment' refers to a system where decisions are made based on consensus among nodes, ensuring data integrity and security through distributed logging and Raft-based coordination.

BigHPC2024: Special Track on Big Data and High-Performance Computing, co-located with the 3rd Italian Conference on Big Data and Data Science, ITADATA2024, September 17 – 19, 2024, Pisa, Italy.

*Corresponding author.

gamorabito@unime.it (G. Morabito); aficara@unime.it (A. Ficara); acelesti@unime.it (A. Celesti); mfazio@unime.it (M. Fazio)

0009-0006-2144-8746 (G. Morabito); 0000-0001-9517-4131 (A. Ficara); 0000-0001-9003-6194 (A. Celesti); 0000-0003-3574-1848 (M. Fazio)

© 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073

CEUR-WS.org/Vol-3785/paper112.pdf

The scientific contributions of this paper are:

- Adaptation of Raft consensus for Edge orchestration.
- Distributed logging strategy for fault tolerance.
- Performance analysis of distributed orchestration based on consensus.

The rest of the paper is organized as follows: Section 2 describes solutions in the literature on trusted and distributed orchestration. In Section 3, we present key concepts of consensus. Then, in Section 4 we describe our proposed solution. Section 5 resumes the implementation and experiments that we carried out, also discussing performance of our solution. In Section 6 we draw our conclusions and present possible future works.

2. State of the art

In recent years, the topic of trust in distributed systems and microservice orchestration has received considerable attention from the scientific community. Numerous studies have investigated innovative solutions to ensure the security, integrity, and trustworthiness of operations [1], especially in decentralized Edge Cloud environments [2]. In particular, Fog computing, which acts as a conduit between the Cloud and the Edge resources, has shown significant potential in improving trust in distributed systems. Researchers proposed a Fog orchestration system for the Edge [3] that employs advanced encryption techniques to ensure that only necessary devices participate in monitoring, reducing latency and energy consumption.

In parallel, Blockchain technology has emerged as a promising solution to address the challenges of trust in distributed systems. [4] proposes an architecture based on permissioned distributed ledgers, creating a trusted environment for Cloud service providers and mobile network operators. This approach enables secure and transparent management of the lifecycle of network services in a Multi-Cloud environment. Other solutions [5, 6] describes the use of smart contracts and sharding techniques to improve blockchain efficiency by supporting automated orchestration of microservices in IoT environments. Moreover, a study [7] highlights how a Blockchain-based architecture can create a trusted environment for the various stakeholders involved, including Cloud service providers, mobile network operators and regulators. In addition, some studies have combined Blockchain with other technologies to further enhance the trust and efficiency of distributed systems. For example, the adoption of self-sovereign identities and verifiable credentials for decentralized discovery and orchestration of microservices has improved the security and reliability of operations [8]. Other approaches [9, 10] combine Blockchain with containerized orchestration platforms, such as Kubernetes, to ensure the integrity and transparency of orchestration decisions in Edge networks. Furthermore, recent research [11] has explored the integration of Blockchain with deep reinforcement learning (DRL) techniques to minimize orchestration costs and improve quality of service by dynamically adapting to network conditions.

Despite the advantages, implementing blockchain in Edge environments presents several challenges. Latency and resource consumption are major concerns, as Edge devices often have limited computational and memory capacity. For example, one study [12] highlights the need to efficiently orchestrate resources without compromising performance, suggesting the use of semantic patterns to manage data and software resources in the Edge layer. In summary, the state of the art in trust enhancement for distributed systems and microservice orchestration demonstrates significant strides through the integration of technologies such as Fog computing, Blockchain, and advanced cryptographic techniques. Despite these advancements, many challenges remain unresolved. While Fog computing and Blockchain offer promising solutions, they come with inherent limitations, such as latency and resource consumption issues, which are particularly pronounced in Edge environments. The incorporation of additional technologies, such as self-sovereign identities and deep reinforcement learning, has introduced new ways to address these issues, yet these solutions are not without their own drawbacks.

3. Consensus

A consensus algorithm allows a group of nodes working towards a common goal to reach an agreement that ensures the reliability and consistency of the nodes' activities. In Edge computing, a consensus algorithm ensures agreement among distributed Edge nodes on how to carry on an efficient and reliable management of computational, storage, and networking resources at the edge. In our solution, we used Raft as consensus algorithm. Renowned for its simplicity and efficiency, Raft has found widespread adoption in distributed systems demanding strong consistency and fault-tolerance, such as distributed databases, key-value stores, blockchain systems, and cloud storage platforms. Raft is based on a leader-follower paradigm. A single node assumes the role of leader, responsible for receiving and processing client requests. This leader takes decisions and asks for approval from the remaining nodes, known as followers. To track system progression and differentiate between competing leadership claims, a concept known as "term" is employed. Each election of a new leader initiates a new term.

The Raft algorithm unfolds in 3 main phases:

- **Leader Election**: each participant node can candidate itself as leader, while the other nodes vote for its election. Leader election is carried on whenever the current leader becomes unavailable. Nodes independently initiate timers waiting for votes, and the node with the highest number of votes is crowned the new leader. To detect if a leader becomes unavailable, Raft uses *heartbeats* and *terms*: heartbeats are messages sent periodically by the leader to the followers to signal its activity; a term is the period of time between two election phases. When a follower does not receive the heartbeat in a term, it considers the leader failed and starts a new election, thus initiating a new term. If the leader receives notice of a new term, it downgrades to a follower and contributes to future elections.
- **Voting**: Raft operates using the principle of majority agreement. Before committing an operation, the leader asks the followers to finalize a decision and waits for acknowledgements from most of them. When any vote is taken, a quorum of 50%+1 is necessary for consensus to be reached, regardless of whether the reply to the consensus request is positive or negative.
- **Logging**: after receiving the consensus on the decision, the leader appends new log entries to its local log and disseminates them to followers. Followers validate the incoming log entries against their existing logs, incorporating consistent entries while requesting missing segments in case of discrepancies.

In the following section we explain how the basic functionalities of Raft have been adapted to implement a new orchestration service.

4. Consensus in Resource Orchestration at the Edge

The high-level reference architecture of the proposed framework is shown in Figure 1. It is composed of three key components: 1) a network of Edge nodes, 2) a distributed orchestration layer and 3) a distributed log layer. Each Edge node can execute microservices as containerized units managed through a dedicated Container Engine (e.g., Docker, RKT or LXD). Moreover, each node is equipped with a Resource Monitoring Agent to track the consumption of local resources such as CPU, memory and network utilization, and the activities of microservices. Some (or even all) Edge nodes (the green ones in Figure 1) can receive requests for microservice deployment. The efficient deployment of microservices is managed by the distributed orchestration layer. Based on the data collected by the Resource Monitoring Agents, it manages the workload across the distributed computing resources. The distributed log layer implements the functionality to keep track of the requests sent to the network on which consensus was reached and for which an orchestration activity is performed. More details on the architecture components are provided in the following.

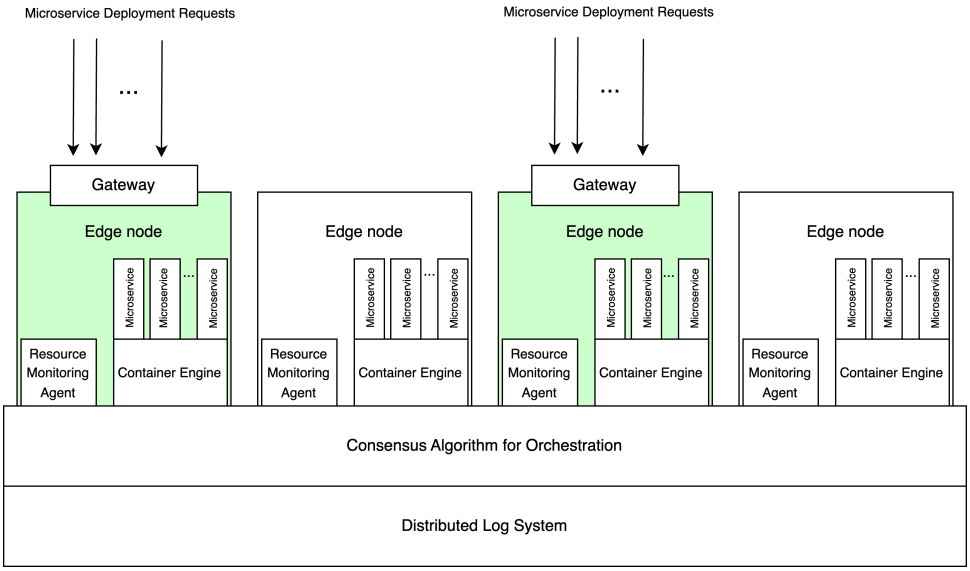


Figure 1: High Level Architecture

4.1. Orchestration

In general, orchestration activities implement two types of service:

- **The deployment of a microservice**: it consists of the provision of a service that is not already being executed, for which an explicit request from an external user is expected.
- **The migration of a microservice**: it is a process that only affects services already instantiated on a node of the network; it requires that a certain service be assigned different resources from those already allocated, resulting in the service running on another node. This must be done without altering the execution state of the service itself and minimising the interruption time of the service. This orchestration mechanism is useful for load balancing, improving latency and throughput of the system.

Despite the fact that both these goals are equally important, in this work we focused only on service deployment. In fact, a migration can be modelled as a deployment whose request comes from the Edge nodes. So, migration involves the same orchestration policies and solutions of deployment. When a gateway receives a request, the deployment routine is executed. Traditional orchestration solutions, tied for the Cloud, tend to be centralized, which is not ideal for the distributed nature of Edge Computing. Coordinating and managing a network of numerous, geographically dispersed Edge devices involves ensuring consistent software updates, handling device failures, and maintaining overall system performance while addressing the constraints of limited resources and varying connectivity conditions. Effective orchestration in this context demands sophisticated tools and strategies to ensure seamless integration and operation of all Edge components. The current centralized orchestration frameworks are ill-suited to handle these dynamic and distributed environments effectively. The need for a more efficient orchestration mechanism is evident. One promising approach is to enable Edge devices to manage orchestration themselves in a distributed manner. This paradigm shift not only minimizes latency by reducing data transmission distances but also enhances the system's robustness and fault tolerance. However, a distributed system introduces the problem of trust among the Edge devices, as they must rely on each other to maintain system integrity and performance.

In our work, the orchestration system works following 4 steps:

1. The node receiving the request activates the Election phase of the consensus algorithm, announcing its candidature as leader.

2. During the consensus phase, the state of the nodes in the network is asked. In this context, the state of a node is its workload, represented as a numeric quantity. Depending on the use cases, it may coincide numerically with the average CPU usage in a given time interval, or the amount of central memory used, or even the weighted average of these two quantities.
3. Once all the states are received, the leader makes the choice of the node on which to do the deployment based on a workload ranking, choosing the node with the lowest load.
4. If consensus is reached, each node, including the leader, verifies that it is the chosen one; the latter takes over the client request, sent by the leader in question, and deploys the requested service.

In our solution, the orchestration layer is managed by the Raft consensus algorithm. The Raft algorithm allows to maintain a trusted environment by ensuring consensus among nodes. It prevents single points of failure by rotating the leader role, thus distributing trust across multiple nodes. This guarantees that decisions, such as microservice deployment, are validated and agreed upon by a majority of nodes, enhancing system robustness. The Raft algorithm was suitably modified to handle incoming external requests for orchestrating microservices. In detail, a fourth phase (*Pause*) was added during which there are no request to be handled and no node is the leader. When a new request arrives, the node that receives it candidates itself as leader to handle it. This algorithm is crucial to guarantee the consistency and availability of the system, ensuring that all nodes operate in a concerted and synchronised manner. Underlying the architecture is a distributed logging system, which is used to maintain a log of the operations and events that occur in the system in a distributed manner. This distributed log is critical to the resilience of the system, allowing for the recovery and replication of data in the event of failures, thus ensuring data consistency and business continuity.

4.2. Logging

Distributed logging is the functionality that keeps track of the requests sent to the network on which consensus was reached and for which an orchestration activity is performed. There are two possible approaches for enabling distributed log: 1) to store the log in an external system, accessible by all the Edge nodes, such as a networked database; 2) to store the log in a distributed data collection system on the Edge nodes themselves, for example, in a distributed file system. The first approach involves a log storage system external to the orchestration one, which may also involve different nodes or a remote collection system (e.g. on the Cloud). However, this makes the proposed solution dependent on external entities, which may lead to problems with the effectiveness of the entire solution. Therefore, it was decided to opt for the second approach, in which the Edge nodes are responsible for all functionalities of the proposed system. This makes the solution more robust in the event of problems connecting the Edge network to the Internet or remote systems. Log writing is an event that occurs at the conclusion of the consensus phase, once it has been reached, prior to deployment, in order to keep track of the choices made and the deployment command sent. The actual instantiation of the service and/or any problems arising during the deployment itself are not tracked by the system as they are activities outside the scope of orchestration. While the proposed solution does not currently handle faults arising during deployment (e.g., node failures), future work could involve integrating advanced monitoring and self-healing mechanisms to address this gap. Fault tolerance could be enhanced by allowing nodes to dynamically recover from failures during the deployment process. The different functionalities of our proposal and their integration are described in the sequence diagram in Figure 2.

5. Implementation and validation

We have implemented the Edge network using 6 virtual machines configured with minimal resources, so to replicate the performance characteristics of Raspberry Pi 4 devices, and the virtual machines were connected to the same private subnetwork. Edge nodes have installed Ubuntu 22.04 LTS as Operating

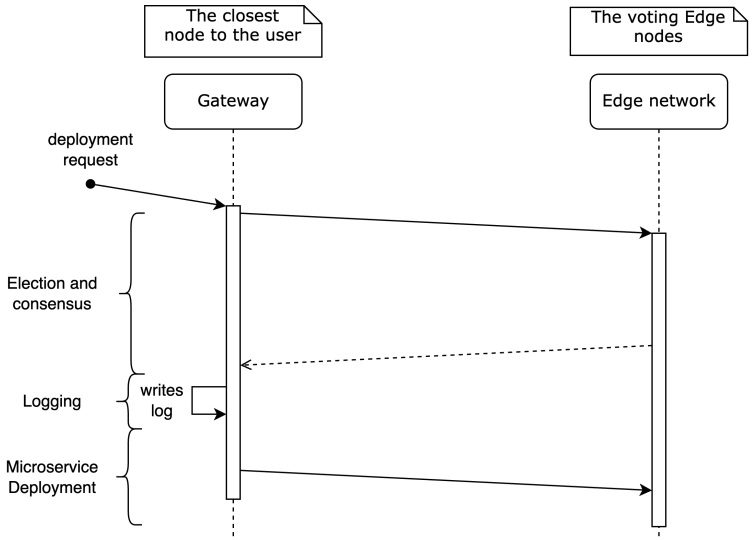


Figure 2: Sequence diagram of the three phases of the designed system

System and Docker[1] as the Container Engine. The microservices were deployed as Docker containers. GlusterFS[2] was adopted as distributed file system for logging. The Orchestration algorithm has been implemented using the Go programming language. The code repository is available on GitHub[3].

5.1. Performance analysis

We conducted some temporal tests, which involved timing the execution or communication of various parts of the algorithm. Figure 3, resumes the entire software execution, expliciting the significant intervals on which the tests were carried out. The tests were executed by varying the number of Edge

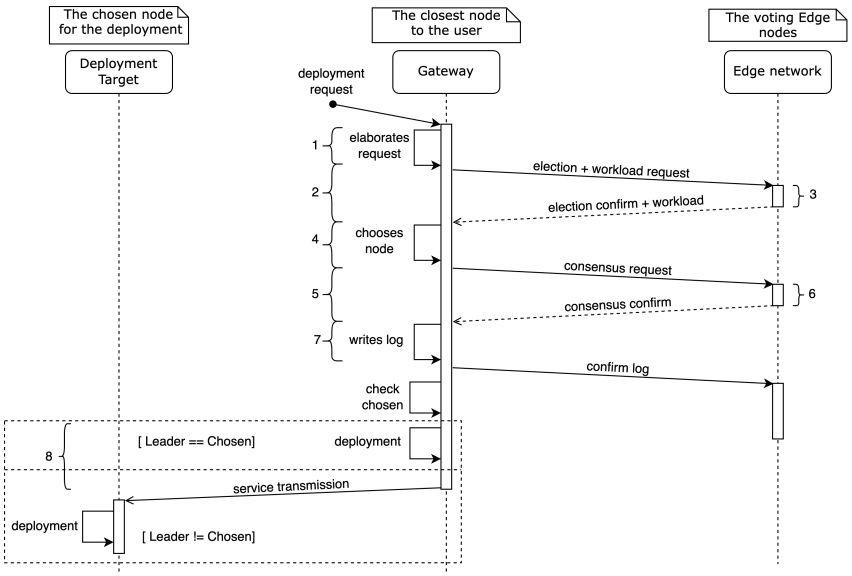


Figure 3: Diagram of the entire software execution

nodes acting as gateway (2, 4 and 6) and by varying the rate of the requests (2 req/s, 4 req/s, 6 req/s). The following execution times were computed:

[1]https://www.docker.com/
[2]https://www.gluster.org/
[3]https://github.com/fcrlab-unime/distributed-orchestration-consensus.git

- **RequestElab**. The processing time for the received request, which includes the time needed to parse it and separate any multiple requests.
- **ElectionNetworkMean**. The transmission time in the network for election-related messages. This value is averaged from the transmission times between the leader and all voters in the network, after subtracting the election request processing time at the node being considered from each individual value.
- **VoteElectionMean**. The processing time for the election request at the voting node.
- **ChoosingPhase**. The selection time among the voters to determine the node on which the deployment will be performed.
- **VoteConsNetMean**. The transmission time in the network for consensus-related messages. This value is averaged from the transmission times between the leader and all voters in the network, after subtracting the vote processing time at the node being considered from each individual value.
- **VoteConsElabMean**. The processing time for the vote at the voting node.
- **WriteLog**. The time to write the log to the file system shared among the nodes.
- **TransfertReq**. The transmission time in the network for deployment-related messages.

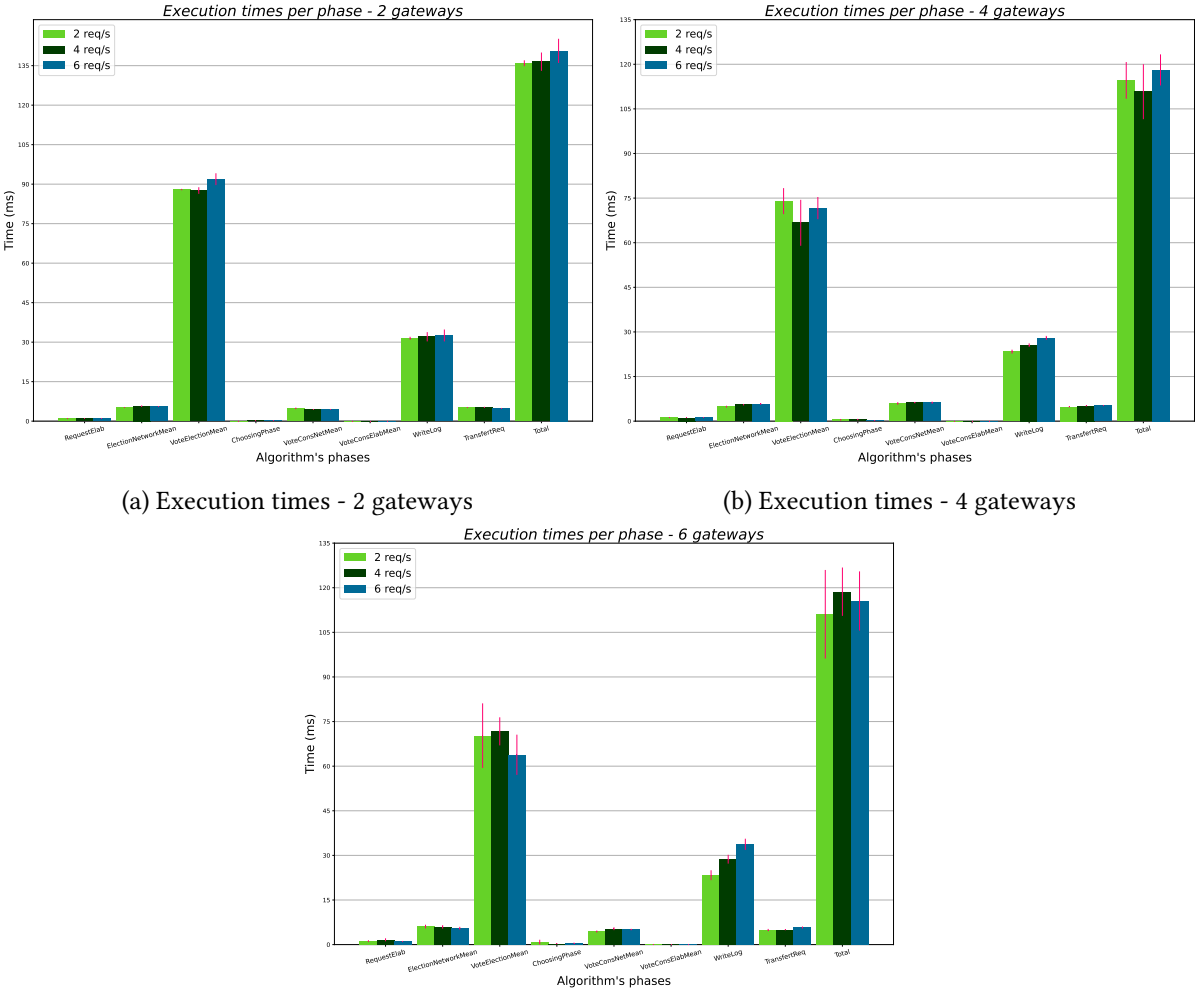
Figure 4 presents a summary of the results. Specifically, it highlights that the execution times for ChoosingPhase and VoteConsElabMean are negligible compared to the other phases. In contrast, the phases that consume the most time are VoteElectionMean and WriteLog. The overall system performance improves as the number of gateway nodes increases: more nodes distribute the load more effectively. The number of requests per second, instead, does not significantly impact performance, as shown by the confidence intervals in Figure 4.

6. Conclusion

In this paper, we proposed a consensus-based distributed solution for orchestrating microservices at the Edge. The proposed solution allows to distribute workloads among Edge nodes, mantaining successful decision-making activities in a distributed log system. The system is trusted, since consensus implies that a majority of nodes must agree on the orchestration-related decisions. The results from the experimental evaluations indicate that this approach to orchestration is promising, as it allows for the benefits of distributed systems to be obtained for functionalities that are traditionally developed in a centralized manner. The decision to persistently maintain a file containing essential information about decision-making allows infrastructure administrators to observe inconsistencies and identify critical issues. Additionally, the choice to enable any node to become the leader increases the algorithm's level of distribution, thereby reducing the likelihood of having a single point of failure. Future work will address additional functionalities, such as microservice migration, horizontal scaling and enhanced fault tolerance, to further improve robustness and applicability in dynamic environments.

Acknowledgments

This work has been partially supported by the European Union (NextGeneration EU), through the MUR-PNRR project SAMOTHRACE (ECS00000022), and the Italian Ministry of Health, Piano Operativo Salute (POS) trajectory 2 "eHealth, diagnostica avanzata, medical device e mini invasività" through the project "Rete eHealth: AI e strumenti ICT Innovativi orientati alla Diagnostica Digitale (RAIDD)"(CUP J43C22000380001). We would like to thank Claudio Anchesi for his valuable work in the implementation of the presented solution.



(a) Execution times - 2 gateways

(b) Execution times - 4 gateways

(c) Execution times - 6 gateways

Figure 4: Execution times per phase

References

[1] G. P. Fernandez, A. Brito, Secure container orchestration in the cloud: policies and implementation, in: Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing, SAC '19, Association for Computing Machinery, New York, NY, USA, 2019, p. 138–145.

[2] V. T. Le, Trusted orchestrator architecture in mobile edge cloud computing, in: C. Zirpins, I. Paraskakis, V. Andrikopoulos, N. Kratzke, C. Pahl, N. El Ioini, A. S. Andreou, G. Feuerlicht, W. Lamersdorf, G. Ortiz, W.-J. Van den Heuvel, J. Soldani, M. Villari, G. Casale, P. Plebani (Eds.), Advances in Service-Oriented and Cloud Computing, Springer International Publishing, Cham, 2021, pp. 121–132.

[3] A. Viejo, D. Sánchez, Secure monitoring in iot-based services via fog orchestration, Future Generation Computer Systems 107 (2020) 443–457.

[4] E. Zeydan, J. Baranda, J. Mangues-Bafalluy, Y. Turk, S. B. Ozturk, Blockchain-based service orchestration for 5g vertical industries in multicloud environment, IEEE Transactions on Network and Service Management 19 (2022) 4888–4904.

[5] Y. Qi, S. Shao, S. Wu, X. Qiu, S. Guo, S. Guo, A distributed intelligent service trusted provision approach for iot, IEEE Internet of Things Journal 10 (2023) 22341–22355.

[6] C. Pahl, N. E. Ioini, S. Helmer, B. Lee, An architecture pattern for trusted orchestration in iot edge clouds, in: 2018 Third International Conference on Fog and Mobile Edge Computing (FMEC), 2018, pp. 63–70.

[7] E. Zeydan, J. Baranda, J. Mangues-Bafalluy, Y. Turk, Blockchain for network service orchestration: Trust and adoption in multi-domain environments, IEEE Communications Standards Magazine 7 (2023) 16–22.
[8] I. Barclay, C. Simpkin, G. Bent, T. La Porta, D. Millar, A. Preece, I. Taylor, D. Verma, Trustable service discovery for highly dynamic decentralized workflows, Future Generation Computer Systems 134 (2022) 236–246.
[9] N. E. Ioini, C. Pahl, Trustworthy orchestration of container based edge computing using permissioned blockchain, in: 2018 Fifth International Conference on Internet of Things: Systems, Management and Security, 2018, pp. 147–154.
[10] S. Ren, C. Lee, Z. Latif, Blockchain-based trusted container orchestration for edge computing, in: 2022 13th International Conference on Information and Communication Technology Convergence (ICTC), 2022, pp. 88–92.
[11] S. Guo, Y. Dai, S. Xu, X. Qiu, F. Qi, Trusted cloud-edge network resource management: Drl-driven service function chain orchestration for iot, IEEE Internet of Things Journal 7 (2020) 6010–6022.
[12] C. Pahl, N. El Ioini, S. Helmer, B. Lee, A semantic pattern for trusted orchestration in iot edge clouds, Internet Technology Letters 2 (2019) e95.