POLITECNICO DI TORINO

Master's Degree
in Computer Engineering

Master's thesis

Edge-to-Cloud Multi-Cluster Orchestration for Smart Grid Monitoring Services

Supervisors
prof. Fulvio Risso
ing. Stefano Galantino

Candidate
Riccardo Medina

Academic Year 2023-2024

Summary

In recent times, the introduction of edge/fog computing platforms, which operate on the principle of processing data in locations other than the central node (directly at the production node or at intermediate nodes), has enabled the development of monitoring systems and data consumption with greater speed and precision.

Within the context of the energy network, these platforms can be particularly suitable for the rapid (or automated) deployment of applications and services.

In fact, the geographically distributed compute infrastructure in combination with the transparency of virtualized (or cloud native) platforms opens up unprecedented flexibility for application deployment.

Furthermore, they facilitate the implementation of new functionalities, for example to ensure the infrastructure's operation in case of disconnection from the backbone network.

This is achieved by temporarily relocating essential software services locally and subsequently realigning the data with the remote "main" instance once the connection is restored (island mode operation).

Based on the feasibility of local solutions using the Kubernetes platform in a distributed environment, this thesis presents a potential model to extend their edge/fog computing features to the entire electrical network, employing the innovative open-source project Liqo as the main technology for the management of the distributed cluster architecture.

In developing this model, significant emphasis was placed on ensuring a high degree of resilience.

This was accomplished through the careful selection of a topology that not only supports the seamless integration of existing distributed database systems within a multi-cluster environment, without requiring any additional modifications, but it also provides the flexibility to distribute workload across any node, unrestricted by fixed architectural constraints but allowing the addition of logical constraints depending on the desired hierarchical architecture.

After examining the topology, the study progressed to creating a possible implementation in the domain of the electrical networks, analysing its behaviour in the event of faults and assessing its scalability.

Comparing the obtained results with the initial solutions, the architecture adopted in this thesis demonstrates its capability to integrate and extend the local solution's features to the entire network, without significant increases in latency despite the greater complexity. It also introduces new functionalities such as island mode operation, enabling separate management of the two network parts during disconnection and automatic resources reconciliation once the link is restored.

Acknowledgements

I would like to dedicate this space to the people who, with their support, have helped me in this wonderful journey of deepening the knowledge acquired during my university years.

Led by my supervisor Risso, who, through his courses during my master's years, encouraged me to give my best and made me passionate about this course of study, even though it also brought some negative aspects, such as being forced to devise new low-tone curses in front of yet another bug during evening debugging sessions.

Owed to my co-supervisor Galantino, always present in times of need and consistently available to organize last-minute meetings despite his already busy schedule. I often wonder how he didn't uninstall Slack after yet another message from me asking desperately for help.

Vital to my journey are my parents, who have always supported me in achieving my projects. From the very beginning, they believed in my potential and invested in my education, providing both moral and financial support. They were my pillars of strength during moments of discouragement, offering unwavering encouragement and advice that kept me focused on my goals. They celebrated my achievements as if they were their own, sharing in my joy and pride at every milestone. It is primarily thanks to their love, guidance, and sacrifices that I have reached this point. I could not have asked for better parents, and I am profoundly grateful for their constant presence and support throughout this journey.

Endlessly grateful to my little sister "Babbusè", with whom, despite our squabbles, I have shared so much, from the interminable and dreadful lunchtime TV series to our walks to stay fit.

Never forgetting my grandparents, who, in addition to making me experience the hardships of farm life (harvesting potatoes under the sun is a nightmare), managed to instill in me the values of humility intertwined with pride. I only regret not being able to celebrate with all of them.

I thank my housemates, with whom I have lived for years, experiencing adventures of all kinds and who found the strength to put up with me, despite trying to undermine my confidence in my cooking abilities. I especially thank Mina "Dove è Mikkel, no DOVE è Mikkel", who, besides instilling in me some musical culture, improved my discussion skills; Mich "Occhio alla testa", for the countless moral debates and numerous duo matches in FIFA; Pietro "Baba Piaga", who, besides leading me down the bad path of LoL, taught me how to cook much better.

Grateful to my friends Antonino "Scaffa The Legend", Attilio "CT Fioradoni Sbarra Tatanga", Davide "Hacker CBCR", Arianna "The 96 girl", and the other companions from my hometown for always being there and putting up with me until now.

Generously thanking all the people at Lab 9, as they created a peaceful workplace welcoming anyone who enters, even for someone as unsocial as me, and the people from RSE, always available and professional even during my terrible presentations.

Appreciating their understanding, I conclude with a hope for forgiveness regarding my preference for only my family at the ceremony. Additionally, I give an honorable mention to the song 'No rules!', which greatly aided in relieving stress over the past few months.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In recent years, advancements in technology have significantly increased the capacity for data collection across all sectors. However, these improvements have also resurrected longstanding issues associated with managing large volumes of data, such as inefficiencies in transportation networks and data processing centers.

The adoption of edge/fog computing paradigms has addressed these challenges by decentralizing data processing. Edge computing involves processing data directly at the source, while fog computing processes data at intermediate nodes within the network. This approach enhances the speed and accuracy of data monitoring and consumption systems by reducing the burden on transportation and central processing nodes.

Moreover, these paradigms facilitate the implementation of new functionalities, such as enabling infrastructure to operate independently (island-mode) when disconnected from the main network. They also increase flexibility by allowing data flows to be rerouted to alternative destinations in response to failures.

Since data is partially processed at distributed nodes, there is less reliance on highly specialized or memory-intensive destination centers, thereby supporting the creation of multiple destination points.

1.1 Energy section

In Italy, the national electrical grid can be divided into four main areas, as highlighted in Figure 1.1.

- **Production:** This area encompasses all energy production facilities, historically dominated by large power plants such as fossil fuel and hydroelectric plants, as well as imported energy. Nowadays, smaller-scale and more variable production outputs from renewable energy sources have been introduced.

- **Transmission:** This area includes the infrastructure responsible for long-distance transmission of produced energy, using high-voltage alternating current. Its primary function, known as "dispatching"[1], is to balance consumption levels with supply levels since energy cannot be efficiently stored. Managed in Italy by Terna[2], this

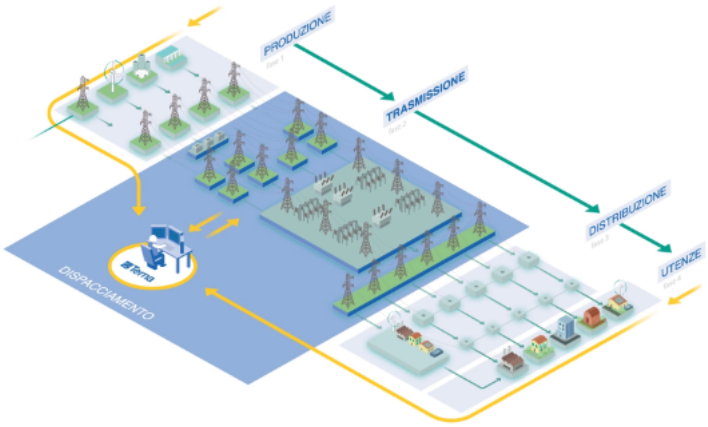

Figure 1.1. General Overview of the Electric Energy Grid Macro Areas[1].

infrastructure is highly automated to handle power plant failures or service interruptions.

- **Distribution:** This area comprises the infrastructure that transports electrical energy to end-users, passing through primary substations (transforming high-voltage electricity to medium voltage) and secondary substations (from medium voltage to low voltage). It is divided into zones managed by independent distribution companies responsible for maintenance. The Smart Grid model, developed over the past decade, aims to automate this infrastructure similarly to the transmission area to enhance energy control and usage.

- **Consumers:** This area involves delivering electricity to the final customer, determining economic costs, and the characteristics of the supplied energy. These aspects are managed by various sales companies that negotiate agreements with distributors and end-users.

Each area plays a crucial role in the overall operation and management of Italy's electricity network, ensuring efficient and reliable energy supply across the country.

1.2 Thesis objectives

The Smart Grid model has revolutionize the traditional distribution network by establishing an intelligent information network upon the entire traditional distribution architecture. This new network could integrate edge/fog computing principles and could support new

[1] https://www.terna.it/it/sistema-elettrico/ruolo-terna/come-funziona-sistema-elettrico

functionalities such as enabling temporary indipendent management of network segments (island mode operation) or securely managing different energy sources through centralized automation.

Current approaches propose using the Kubernetes platform with multi-master clusters to manage individual stations. While these solutions introduce edge/fog computing and local centralized control, they face challenges in scalability, security, and cannot support critical features like temporary indipendent management in case of disconnession (island-mode operationality) if they are translated to manage the entire network. Building manually a control logic between the various cluster could be a solution, but would be very expensive to create, cumbersome to manage and time-consuming to update.

Starting from the current local Kubernetes solutions, this thesis aims to develop a model that extends edge/fog computing capabilities across the entire power grid using a centralized automatic control logic, thanks to the innovative open-source project Liqo. It will be utilized to manage the distributed cluster architecture, enhancing resilience by enabling the system to withstand failures of entire clusters while enabling new features as the insland mode operation.

1.3 Overview

This thesis endeavors to create a scalable model for the entire network, building upon successful local solutions.
Chapter 2 introduces Kubernetes, an open-source platform, with a focus on its lightweight version, k3s, as the foundational technology.
Chapter 3 delves into the local solution based on Kubernetes, detailing its components and addressing scalability and functionality challenges.
Chapter 4 introduces the Liqo project, pivotal for extending the local model across the entire network, elucidating its core concepts. It examines how this technology interacts with existing distributed database systems designed for single-cluster environments, emphasizing the adjustments needed for seamless integration.
From Chapter 5 onward, the thesis delves deeper into its core discussions. Chapter 5 initiates by outlining the rationale behind selecting the partial mesh star physical topology, exploring various hierarchical configurations within this framework and analizing those configuration in term of scalability and resilience.
Chapter 6 then presents two viable implementations of this model, tailored to the structure of energy distribution grids.
Chapter 7 critically evaluates the first implementation chosen for its robust resilience, contrasting its performance against the initial local solution.
Finally, Chapter 8 provides a reflective analysis of the findings and proposes future research directions.

Chapter 2

Kubernetes

In this chapter, we will briefly describe Kubernetes technology, which has been used as the foundation for local solutions studied in recent years. This thesis integrates Kubernetes with Liqo technology.

As stated in its official documentation [3], Kubernetes is a portable, extensible, open source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation. Kubernetes emerged as a platform designed to automate the management of containerized applications, ensuring periodic checks to maintain alignment between the actual operational state and the defined ideal state through a declarative language.

A decade since its release as an open-source project, Kubernetes stands as one of the most extensively utilized platforms worldwide. This project focuses on k3s, a lightweight variant of Kubernetes tailored for operation in resource-constrained environments.

2.1 Basic concepts

The foundational principles underlying the architecture of Kubernetes are articulated as follows:

1. Implementation-agnostic APIs: Each Kubernetes object can be implemented differently depending on the version being used, yet the interface used to manage these objects remains consistent across all versions.

2. Completely declarative specification: Kubernetes facilitates the use of a declarative language instead of the traditional imperative approach, simplifying application management by specifying the desired state directly rather than detailing how to achieve that state from various starting points.

3. Control loop-oriented approach: Kubernetes employs components known as controllers that cyclically monitor whether the current state aligns with the desired state. If discrepancies are identified, these controllers initiate actions to minimize the gap between the states. This behaviour is shown in Figure 2.1.

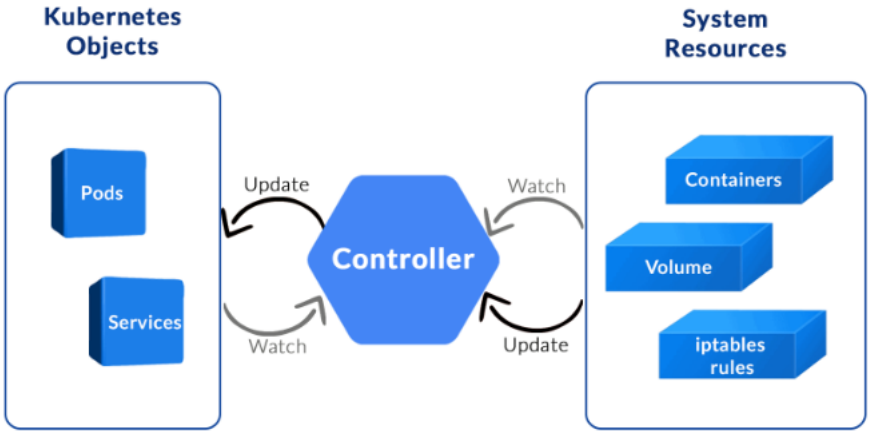

Figure 2.1. Control loop mechanism[1].

These principles are the cornerstone of Kubernetes, facilitating efficient management and orchestration of containerized applications in a variety of computing environments.

Every object within Kubernetes is meticulously crafted to adhere to foundational principles, starting with its smallest operational unit: the object Pod. A Pod may consist of one or more containers that will run the application and its configuration is defined in its respective YAML file. This file may reference other configuration files using key-value pairs, such as Secrets or ConfigMaps, useful in case these configurations are repeated multiple times.

Containers within pods typically share a common network and can communicate with each other by default. However, Kubernetes provides flexibility to configure additional sharing capabilities, including interactions with the underlying host infrastructure, depending on specific deployment requirements. In addition to application containers, a od can include init containers that execute during the Pod's startup phase to set up the environment.

Kubernetes also supports the injection of ephemeral containers for debugging purposes while a Pod is running. In typical usage scenarios, direct interaction with Pods is uncommon, as they are ephemeral entities within Kubernetes. Instead, management typically focuses on higher-level objects such as controllers. The emphasis is on the collective state of these replicas rather than individual Pods.

Pods are typically instantiated and taked care through the implementation of various Kubernetes controllers. The Deployment controller, commonly used for stateless applications, describes the desired application state while managing scalability through the ReplicaSet.

[1] https://k21academy.com/docker-kubernetes/kubernetes-operator/

For stateful applications, the StatefulSet controller is normally utilized, managing the Pod-to-volume binding and ensuring properties such as unique network IDs that the stateful application required to function properly. A DaemonSet places a replica of a Pod on each node that matches its nodeSelector options (or on every node in the cluster if nodeSelector is not set).

As a controller, this behavior supports dynamic changes: if a node is added to the network, the DaemonSet starts a Pod replica on that node; if a node is removed, the corresponding pod is not rescheduled. Typically, DaemonSets are employed for specific tasks such as log collection and node monitoring.

While controllers oversee the lifecycle of Pods, Pod discovery is entrusted to Services. These Kubernetes objects target all pods matching their selector criteria, facilitating exposure both within and outside the cluster. ClusterIP services expose pods solely within the cluster, whereas NodePort or LoadBalancer services extend pod accessibility externally.

Pods are instantiated on physical or virtual machines known as nodes, which serve either as master or worker nodes based on their role. A master node not only executes various Kubernetes components, as previously discussed, but also hosts the cluster's control plane such as the scheduler, controller manager, and API server. Conversely, a worker node is dedicated solely to executing the workloads of Kubernetes objects.

The cluster, comprising these nodes, can be structured as a single-master or multi-master configuration. In a multi-master setup, control plane components are replicated across all master nodes, and decisions are made via a consensus mechanism facilitated by the etcd quorum process, which utilizes the Raft algorithm [4].

This setup necessitates an odd number of master nodes to prevent split-brain [5] scenarios and reduce decision-making delays.

These structural and operational principles form the backbone of Kubernetes architecture, facilitating scalable and efficient container orchestration in diverse computing environments.

2.2 K3s

K3s is a lightweight variant of Kubernetes tailored for operation in resource-constrained environments, as described in the official documentation[6]:

- Edge
- Homelab
- Internet of Things (IoT)
- Continuous Integration (CI)
- Development
- Single board computers (ARM)
- Air-gapped environments

- Embedded K8s
- And as the official page says, situations where a PhD in K8s clusterology is infeasible

K3s efficiently utilizes approximately half the memory resources compared to Kubernetes (K8s) by implementing several optimizations. These include eliminating legacy libraries, opting for lightweight alternatives such as SQLite instead of the standard etcd for database management, and containerd instead of Docker as the container runtime.

Additionally, internal mechanisms have been adjusted to reduce memory consumption.

K3s gets its name from the fact that it uses about half the memory, hence it was jokingly named K3s as it consists of 5 letters(K+3+s), which is half of the 10 letters in Kubernetes(K+8+s).

This was demonstrated by two researchers in 2021, Sebastian Böhm and Guido Wirtz [7]. In their tests, they compared the standard Kubernetes technologies, K3s, and microk8s, using 4 virtual machines with the following parameters:

- **CPU:** 2 vCPUs
- **RAM:** 4 GB memory
- **Disk memory:** SSD with a capacity of 50GB
- **OS:** Ubuntu 20.04

The results indicated that CPU and memory resource usage were quite similar across the two version of Kubernetes, but the k3s version demonstrated significantly lower disk space utilization compared to the standard Kubernetes version, approximately half as much, as illustrated in the Figure 2.2.

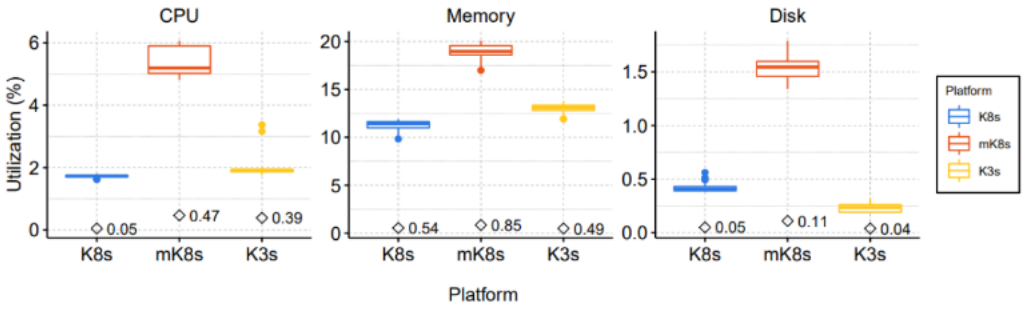

Figure 2.2. Resource utilization for K8s, MicroK8s, and K3s.

The installation process is streamlined with a simple script that weighs less than 100 MB. Despite its compact size, this script effectively manages many of the complexities typically associated with Kubernetes environments, such as automatically configuring TLS certificates, configuring worker nodes inside an existing cluster or configuring new master nodes.

This revision clarifies the optimizations made in K3s to reduce memory usage and highlights the streamlined installation process while maintaining readability and technical accuracy.

Chapter 3

State of the art

In this chapter is presented the current implementation to support edge and fog computing paradigms within the energy production and monitoring network, based on the Smart Grid model (translating hardware components into an IT network) and the use of the Kubernetes platform. The limitations that the implementation studied in this thesis aims to eliminate will also be highlighted.

3.1 Smart Grid components

Currently, the Smart Grid model of the energy monitoring network, illustrated in Figure 3.1, consists of several integral components that work together to ensure efficient energy management and distribution. At the core of this model is the Area Control Center, which serves as the central hub for control and decision-making mechanisms, then there are the production and distribution stations, which are divided into primary and secondary stations.

To manage the network, three main applications are used: Phasor Measurement Units (PMU) for measurements, Phasor Data Concentrator (PDC) from the openPDC project for aggregating data from various PMUs, and Grid State Estimation (GSE) for monitoring the network based on the data provided by the previous applications.

3.1.1 Area Control Center

A computing node within the context of the Smart Grid that serves as an Operational Distribution Center, housing the control and management logic for the entire network. This node is primarily responsible for overseeing the high-level PDC, where data streams from other high-level PDCs situated at primary stations are aggregated. Additionally, it manages the GSE application, which utilizes data from the aforementioned PDC to control the network.

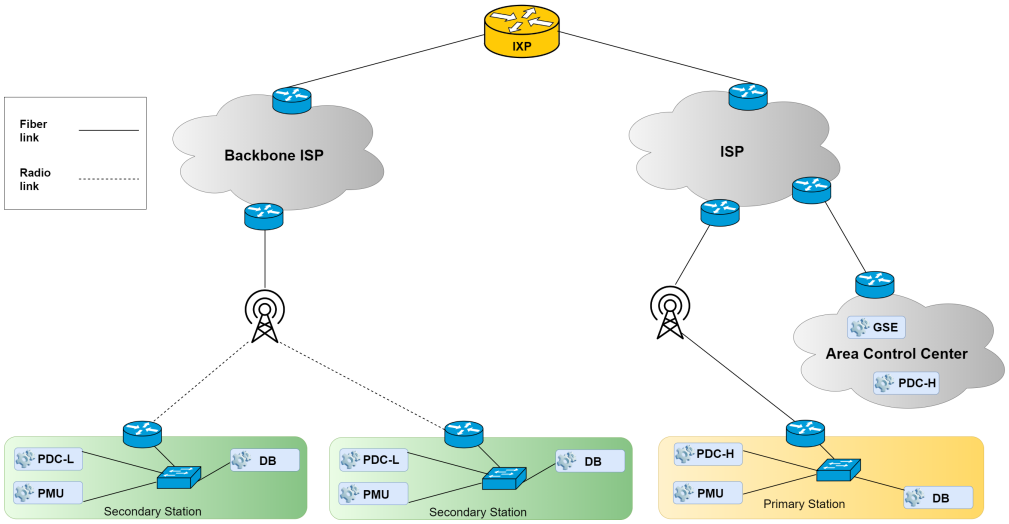

Figure 3.1. Smart Grid abstract informatic model

3.1.2 Station

In the context of the Smart Grid, it is a computing node that represents an energy production or distribution station. Primary stations typically utilize a high-level PDC to aggregate data streams from their subnet of secondary stations. Additionally, they may manage a certain number of PMUs (Phasor Measurement Units). Secondary stations primarily handle PMUs or aggregate data streams generated by them through low-level PDCs.

3.1.3 Phasor Measurement Units

PMUs provide measurements of fundamental electrical quantities, such as voltage and current, in the form of phasors, including information on the amplitude and phase of the measured quantities.

These measurements, synchronized via GPS and sampled at a frequency of 50 samples per second, enable precise monitoring of rapid changes in the electrical system caused by the dynamism of distributed energy resources, and do not need to be saved at this level.

PMUs offer a detailed perspective of system dynamics, overcoming the limitations of more traditional Remote Terminal Units (RTUs), which have an update period of several seconds and are not synchronized. The use of PMUs is expected to enhance the observability and reliability of the distribution system, but due to being expensive they could be put only on subset of the stations.

3.1.4 Phasor Data Concentrator

A phasor data concentrator (PDC) is designed to receive streaming synchrophasor data from phasor measurement units (PMUs) installed on power transmission lines and align these data using GPS timestamps (i.e., it "concentrates" the data based on time).

The output of a PDC is a time-synchronized data set that is forwarded to one or more software applications. These data need to be saved differently depending on the importance of the aggregator. Data from lower-level PDCs should be stored for only 1-2 days for failure analysis, whereas data from higher-level PDCs should be stored for at least 1 month for post-incident analysis.

openPDC is a flexible platform for high-speed time-series data processing, both in real-time and historically. It does not have significant computational power requirements, so it can be installed anywhere within the synchrophasor infrastructure, including on fanless computers located in substations.

3.1.5 Grid State Estimation

State estimation is a technique that allows for the reconstruction of network states, such as nodal voltages, based on available measurements and the electrical network model. Unlike traditional meters, PMU measurements, which include the phase relative to an absolute reference, simplify the state estimation problem by making it a linear system and significantly reducing the computational load.

The objectives of state estimation include the recognition and reduction of measurement errors, the identification of topology errors, the estimation of unmeasured network quantities, and the determination of network parameters through redundant measurements.

3.2 Multi-master station architecture

Recent research has progressed towards managing individual locations such as stations within the Smart Grid using a multi-master architecture [8][9]. This approach leverages enhanced resilience, allowing the system to withstand the failure of a master node. However, it comes with the trade-off of requiring additional resources for replicated control-plane components.

Within the cluster, application configurations (such as those for a PDC if the cluster represents a station) are stored in a high-availability distributed database system. This setup facilitates rapid and automatic redeployment to another node within the cluster in case of a failure, without needing to reconfigure the application parameters. Consequently, the clusters support autonomous local recovery from both application and node failures.

Each cluster serves as a point of edge (for managing secondary stations) or fog (for managing primary stations) computing within the Smart Grid, depending on its location. However, the overall control architecture is manually established between each pair of clusters, each of which operates as a fully independent entity.

This manual establishment of control logic between clusters exhibits limited resilience due to potential errors in configuration and vulnerability in handling failures as shown in

Table 3.1, while also presenting challenges in scalability.

Component Failure	Severity	Cause
Single PMU	Low	Generally the number of other PMUs guarantees the observability threshold.
Multiple PMUs	Low-High	It depends on whether the number of other PMUs guarantees the observability threshold.
Single PDC-l	Moderate-High	If the fault affects only some nodes the observability is stopped only for the time of the rescheduling. Otherwise, that part of the network is no longer be observable.
Multiple PDCs-l	Moderate-High	If the fault affects only some nodes the observability is stopped only for the time of the rescheduling. Otherwise, that part of the network is no longer be observable.
Single PDC-h	High	If the fault affects only some nodes the observability is stopped only for the time of the rescheduling. Otherwise, the entire subnetwork is no longer be observable.

Table 3.1. Component failures overview.

3.2.1 Local extended solution limitations

While this approach of a multi-cluster kubernetes is effective for managing a single station, it proves suboptimal when applied to the entire electrical control system.

This is due to both the complexity involved in managing a large number of nodes (with stations alone numbering in the tens of thousands, whereas Kubernetes officially supports up to 5000 nodes [10]) and the fundamental inability to function in isolation.

For example, it's best to consider the possible failures that can occur within the architecture, as shown in Figure 3.2. Despite there being no unrecoverable failures, since it functions as a single large Kubernetes cluster where deployments are not concentrated in one location and therefore continue to operate, any part that becomes disconnected from the network will no longer be controllable.

This is because the master nodes on the isolated network loses the necessary consensus to initiate new workloads (new pods to manage the isolated entities) and can only partially manage existing workloads (because it can't reschedule workloads if it fails).

The only additional failure scenario that the translated architecture (from local environment to the entire network) can address without losing the ability to control, other than the failure inside a station, is when an entire secondary station without source data but with the aggregator applicative becomes isolated from the network, because it simply transfer the PDC application to another healty node. However, this advantage does not

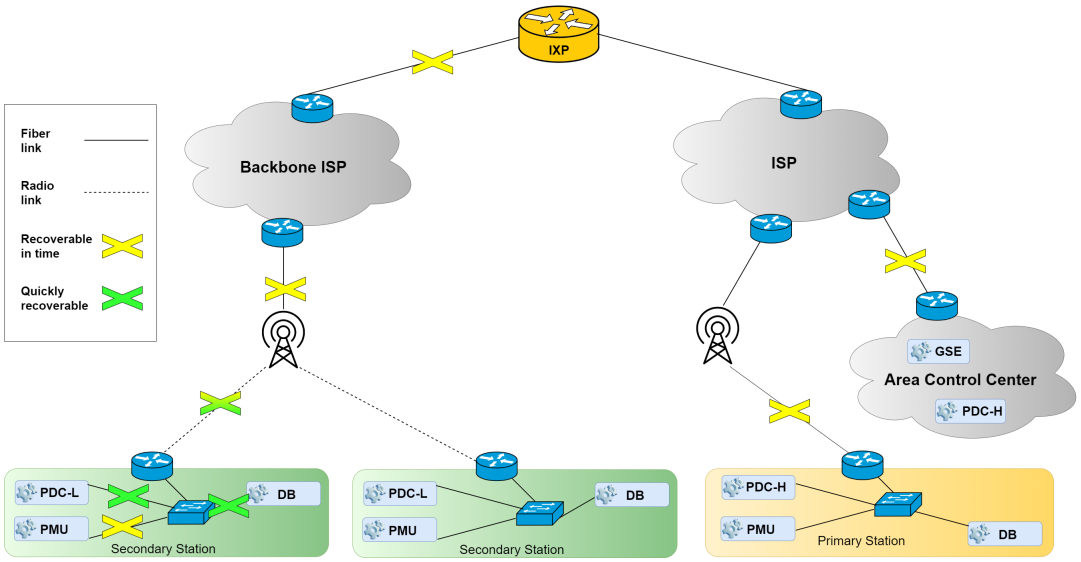

Figure 3.2. Possible failures in multi-level logical domains implementation

outweigh the drawbacks in terms of complexity,lack of scalability, and resource demands inherent in the overall architecture.

An additional limitation concerns the number of master nodes that must necessarily remain healthy. Considering a cluster at its maximum size (which has already been shown to be insufficient to cover an entire network), the highest resilience is achieved with 7 master nodes. In this case, the architecture continues to function as long as 4 out of the 7 master nodes remain active. These limitations can be effectively addressed by adopting Liqo technology.

Chapter 4

Liqo

Due to the rapid adoption of containers as the development environment for applications, there is now a well-established trend towards using orchestration platforms to automate the lifecycle management of containerized applications. Among the various implementations of these platforms, Kubernetes has gained predominant traction, to the extent that multinational corporations with dedicated cloud departments (such as Google, Amazon, Microsoft, Alibaba...) have developed proprietary solutions based on it.

Recently, a trend similar to the one observed with container adoption has emerged, in which there is a growing need for a system that can automate relationships between various clusters managed by these platforms, whether in the cloud or on-premise.

In this chapter will be summarized the Liqo project, designed to address this necessity, described by its creators [11] as "an open-source project that enables dynamic and seamless Kubernetes multi-cluster topologies, supporting heterogeneous on-premise, cloud, and edge infrastructures."

4.1 Basic concepts

The Liqo technology facilitates the creation of a unified virtual network across diverse clusters, enabling the execution of application pods on remote clusters as if they were local. This system is founded on four primary characteristics: network fabric, peering, offloading, storage fabric.

4.1.1 Network fabric

The network fabric is the subsystem of Liqo that seamlessly extends the Kubernetes networking model across multiple independent clusters, enabling pods on different clusters to communicate smoothly even when address NAT is applied.

The control plane of this subsystem resides in the network manager, instantiated as a pod responsible for managing network parameters. It handles tasks both during cluster peering and inter-cluster communications, as example featuring an interface used by the reflection logic for IP address translation.

Interconnecting two clusters involves deploying a secure VPN tunnel using WireGuard, typically established at the end of the peering process based on negotiated parameters. This functionality is implemented by the Liqo gateway component, operating within the cluster as a privileged pod. It also manages routing tables and configures necessary NAT rules to resolve address conflicts.

Although initialized within the cluster's network, this pod utilizes a separate network namespace and policy routing to avoid conflicts with Kubernetes' existing Container Network Interface (CNI) plugins. Traffic from local nodes/pods directed to a remote cluster is routed through an overlay network, based on VXLAN, managed by a DaemonSet component. This component is responsible for routing entries and ensures proper handling of traffic across the VPN tunnels.

4.1.2 Peering

Standard peering is the process that establishes a unidirectional link between two different Kubernetes clusters, enabling the sharing of resources and services. Through this connection, the consumer cluster can initiate processes using resources provided by the provider cluster, but not vice versa.

In this context, the consumer cluster initiates an outgoing peering towards the provider, which reciprocates with an incoming peering from the consumer. This linkage is not exclusive, supporting possible bidirectionality and the scenario where a cluster can act as a consumer for some peerings and as a provider for others.

The peering process unfolds through the following steps:

1. Authentication: Each cluster uses a pre-shared token to verify its identity, which has some permissions for Liqo-related resources and negotiations.

2. Parameter Negotiation: The two clusters exchange sets of parameters necessary for finalizing the peering, including network information such as their respective CIDRs or as the amount of resources shared by the provider.

   Some of these parameters can be modified directly or using dedicated plugins, for example is possible adjusting the available resources that the provider cluster shows to the consumer cluster.

3. Creation of the Virtual Node: Within the consumer cluster, a virtual node is created to represent the resources shared by the provider cluster. Processes instantiated using the provider cluster's resources appear to be located within this virtual node, maintaining transparency in the offloading process and adhering to standard Kubernetes practices without requiring API modifications.

4. Configuration of the Network Fabric: The two clusters configure their respective network fabrics and establish a secure VPN tunnel using the previously negotiated parameters (address remapping, endpoints, etc.).

Each connection can be differentiated based on how Liqo's control plane traffic is managed: whether it passes through the VPN tunnel alongside pod traffic (in-band control plane peering) or uses traditional communication channels (out-of-band control plane peering).

In the former case, it is required to expose only the Liqo VPN endpoint to the pod of the remote cluster. However, this setup requires control over both clusters to negotiate network parameters through Liqo CTL tool [12], resulting in a static peering configuration that requires manual intervention for updates.

In the latter case, while to the remote pods must be expose not only the Liqo VPN endpoint but also the Kubernetes API and Liqo authentication service endpoints (as shown in the Figure 4.1), it offers the flexibility to connect clusters across different domains using a pre-shared token and enables dynamic peering, so that an automatic renegotiation of parameters occurs in response to configuration changes.

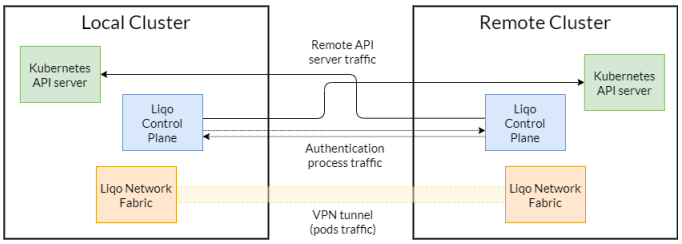

Figure 4.1. Out-of-band control plane peering.

4.1.3 Offloading

Offloading is the method enabling transparent extension of the local cluster into a remote cluster, allowing Kubernetes scheduler to seamlessly schedule workloads in the remote cluster when it's deemed optimal. The virtual node is managed by an extended version of the Virtual Kubelet project, which replaces the traditional kubelet if the node isn't physical.

In the context of Liqo, it interacts with the Kubernetes API servers of both clusters to manage artifact propagation (pods, services, config maps) and reconcile state in case of changes on the negotiated configurations. It also performs configurable periodic health checks to assess reachability of the remote API server, marking the node as unready in case of repeated failures and triggering standard Kubernetes evacuation strategies. An instance of the virtual kubelet is created for each remote cluster to ensure isolation and segregation of authentication tokens.

The offloading process comprises three stages:

1. Namespace Extension: The local cluster's namespace is extended into the remote cluster by creating a gemini counterpart namespace, which will host both offloaded pods and resources required for pod reflection.

2. Resource Reflection: Selected arctifacts from the control plane are reflected in remote clusters to ensure the operational functionality of the pods. Supported resources for reflection include service exposure (Ingress, Services, EndpointSlices), persistent storage (PersistentVolumeClaims, PersistentVolumes), and configuration data (ConfigMaps, Secrets).

3. Pod Offloading: After the scheduler schedules a pod on the virtual node, the corresponding virtual kubelet creates a mirrored pod object in the remote cluster. This object is managed by the custom resource ShadowPod [13], serving as a representation of the pod to maintain service functionality even if connectivity with the remote cluster is lost.

Both stateless and stateful pods are supported, with the latter utilizing either the storage fabric or relying on an external volume provider.

4.1.4 Storage fabric

The storage fabric is the Liqo subsystem responsible for managing the creation of remote volumes for stateful applications. Its operation revolves around delaying the binding of a volume to a pod until it has been scheduled to a node, ensuring volumes are always created where their associated pod is scheduled.

Subsequent scheduling adheres to a data gravity principle, transparently rescheduling the pod to the node where the physical volume resides. These behaviors are implemented through Liqo's virtual storage class, utilizing reflection mechanisms when pods are scheduled on virtual nodes to create the mirrored PVCs in remote clusters. Alternatively, it relies on the real storage class when pods are scheduled on local physical nodes.

4.2 Distributed DB interaction

At present, most of the distributed database systems doesn't support general multi-cluster architecture, primarily due to their reliance on internal headless services for direct pod-to-pod communication. These services return the IP address of the corresponding pod directly when queried, using their DNS entry, unlike regular services that route requests via kube-proxy.

Liqo employs an address remapping mechanism to facilitate seamless communication between clusters; however, this approach results in incorrect IP resolutions for pods scheduled on remote clusters when queried by headless services. To enable the use of these architectures, Liqo developers currently recommend[14] leveraging the peering process, which exposes the address ranges of the two clusters in two different ways:

1. Connecting a cluster to all others via peering while forcing a pod of the distributed system onto it: This approach ensures that the service in that cluster is aware of all real address ranges, allowing replication through the forced pod, which becomes a critical point.

2. Creating a full mesh of peering between various clusters: This ensures that each headless service knows the addresses of all others, and this is the solution adopted in this research.

Some distributed database architectures, such as those implemented by the Percona XtraDB Cluster Operator, may introduce additional complexity. After receiving the

translated IP of a remote pod, they may encounter difficulties establishing a connection, primarily because their cluster logic operates with standard Kubernetes component independently of Liqo. This requires the implementation of distinct CIDRs across clusters, ensuring that traffic is routed through Liqo components to establish connections correctly.

Chapter 5

General topology Partial Mesh Star

In this chapter will be described the process that led to the selection of the partial mesh star architecture, a model capable of applying the logical paradigms of edge/fog computing to a multi-cluster architecture while ensuring the possibility of deploying high-availability systems. Initially, the structural choices will be discussed, based on both the multi-cluster environment and the requirements of the adopted technologies (Percona, Liqo, etc.).

Potential use cases will then be described, demonstrating the flexibility of the logical hierarchical architectures that can be implemented. Finally, we will evaluate the characteristics and various limitations that this model entails.

5.1 Cyber-Physical Architecture

The first step was to consider how to abstract a logical model from the initial real-world situation. The electric power control and monitoring network, as shown in the Figure 5.1, can be schematized using both hierarchical tree topology graphs and peer-to-peer topology graphs.

Peer-to-peer should be discarded because, although the nodes representing the stations can be both data providers and receivers, the node representing the Area Control Center needs to exercise centralized control over all other nodes. Additionally, not all nodes have sufficient processing capabilities, especially if they represent secondary stations.

Among the various tree topology models, the star model is the only one that can be physically implemented using the standard version of Liqo. Indeed, it does not allow the offloading of an already offloaded namespace to prevent critical situations such as circular offloading. This means that all multi-level hierarchical topologies cannot be physically implemented without making customized changes to the technology's code. Moreover, distributed HA database systems tend to need to be in a single namespace, and multi-namespace solutions via operators do not support multi-cluster technologies as they cannot know the namespaces of other clusters.

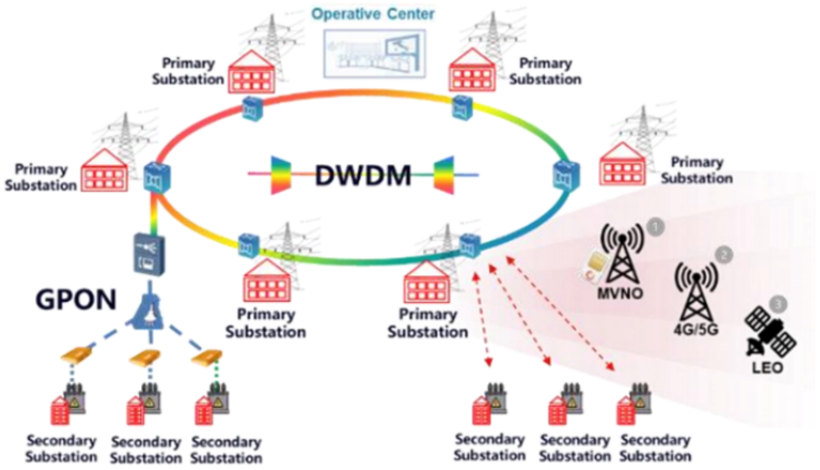

Figure 5.1. Target Telecommunications Architecture Foreseen in the 2023 Development Plan for e-distribution.

The partial mesh version of the star model, which allows direct connections between leaves, is necessary for the correct transparent multi-cluster functioning of distributed database systems that rely on headless services. Each cluster that uses the same database system will need, in addition to having the offloaded namespace of the database, a direct connection with all other clusters, thus creating a partial mesh topology (partial because the connections do not necessarily have to be bidirectional).

5.2 Logical hierarchy using labels and affinity

A simple partial-mesh star topology offers only a two-level physical separation: a central node and the leaves. It lacks the necessary flexibility to manage the various real-world scenarios encountered in a monitoring and distribution network. Therefore, it is necessary to introduce a strategy to construct a complex logical topology on the existing physical model. This strategy is based on the use of Kubernetes' native label and affinity mechanisms.

Each cluster will be identified by a group of labels that specify its position in the desired logical topology and can be used by the scheduler to distribute the workload according to the intended logic. The node affinity mechanism can be used both to distinguish between different clusters and to differentiate the various nodes within a cluster, as it allows specifying different labels as targets. This way, one can define the label that identifies the cluster as well as the labels of the individual nodes within the cluster.

Pod affinity, on the other hand, is used to enforce coexistence conditions between pods on the same node. These mechanisms also offer a degree of flexibility, as they provide both "required" and "preferred" options. The "preferred" option allows the scheduler to

prioritize the specified target for pod placement while still considering alternative targets if the preferred one is unavailable.

The following subsections will discuss some basic logical topologies, from which one can start to build their desired configuration.

5.2.1 Independent Groups

The leaf clusters of the partial-mesh star model are segmented into independent groups by assigning each node within the cluster a label that uniquely identifies its respective group. This method establishes distinct logical areas, as peering between clusters is only necessary within the same group (and only in case of using a distributed database system), to which separated workloads can be allocated. To enhance the delineation of these divisions, the root node could assign a separate namespace to each group, thereby also increasing security between them.

These groups are not mutually exclusive regarding the ownership of a node, provided there is no logical contradiction among the identifying labels. Consequently, a cluster may simultaneously belong to multiple groups, as demonstrated by leaf C in Figure 5.2.

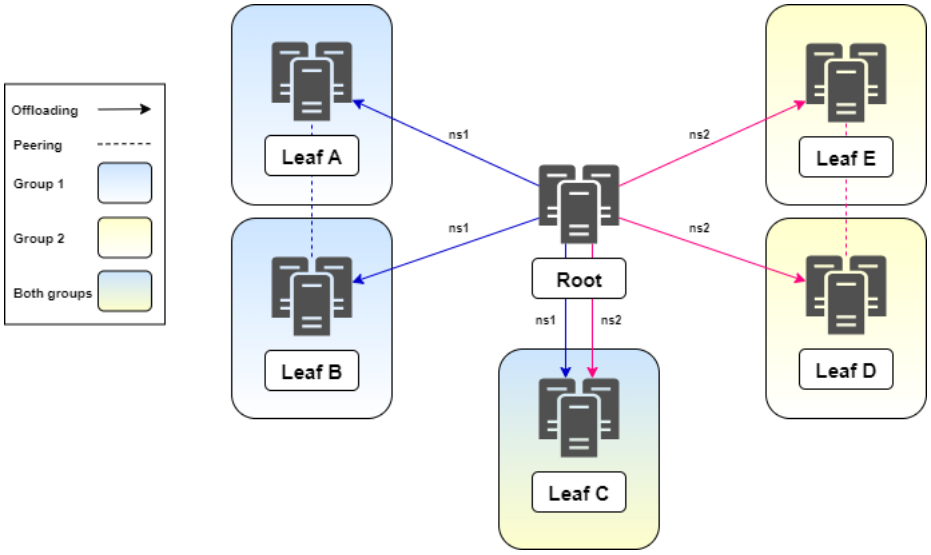

Figure 5.2. Independent Groups Scheme.

5.2.2 Dependent Groups

The leaf clusters of the partial-mesh star model are divided following a logical hierarchy by assigning a label indicating their position within the hierarchy. This approach creates dependent logical areas, as peering is necessary even between clusters belonging to different groups if a distributed database system is used. Figure 5.3 illustrates the worst-case

scenario, where the database domain encompasses all leaf clusters. This topology supports new behaviors, such as allowing not only the selection of which groups to schedule workloads within the same domain.

This mechanism allows for the creation of multiple logical hierarchies within the same physical network, each with its own set of labels. Within a given logical hierarchical structure, a cluster can belong to only one group. However, when considering multiple structures, a cluster can belong to different groups.

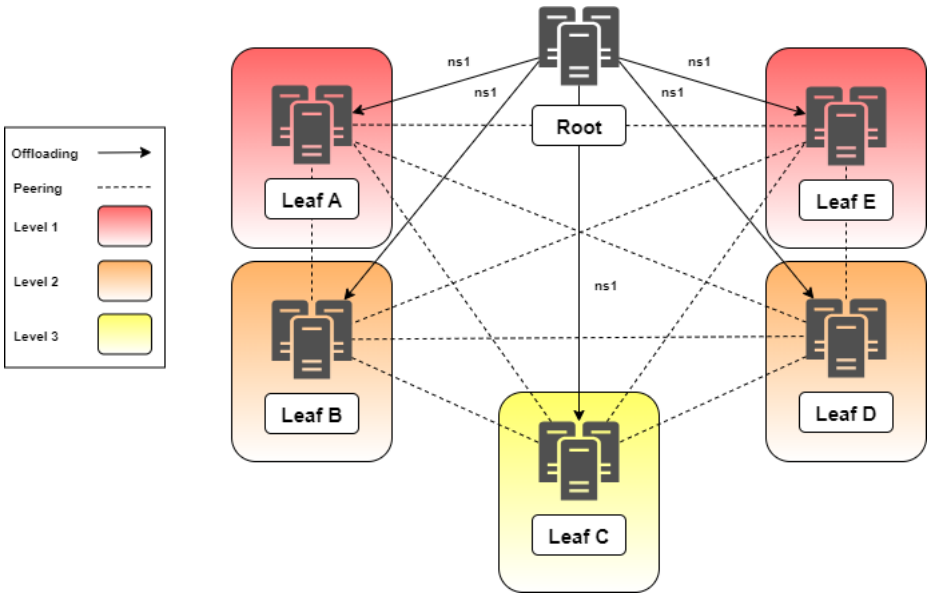

Figure 5.3. Dependent Groups Scheme.

5.3 Partial Mesh Star Analysis

As previously illustrated, the partial mesh star topology allows for the use of systems not originally designed for multi-cluster environments, such as distributed HA databases. However, this feature results in a quadratic increase in the number of peerings required between clusters within the database domain. This is because it requires at least a unidirectional peering full mesh. The number of links can be determined using the formula (5.1):

$$\text{Link} = \frac{N(N-1)}{2} \tag{5.1}$$

where N is the number of clusters.

The increase in the number of peerings only affects the time required to set up the entire architecture during its creation, as the consumption of additional resources is negligible, as noted in the article "Computing without Borders: The Way Towards Liquid Computing" [15].

The label mechanism offers substantial flexibility in selecting the logical architecture to overlay on the physical infrastructure. However, a drawback is the linear increase in setup time as the number of clusters expands. This characteristic renders the partial mesh star topology ideal for systems with relatively stable physical topologies, facilitating quick adjustments in logical configurations. While significant logical topological changes are supported, they require a corresponding setup time.

It should also be noted that this topology enhances the overall system resilience. Since clusters operate as independent entities, the architecture can support any number of disconnections as long as the central node remains unaffected. Therefore, resilience no longer depends on the number of disconnected clusters (as is the case in a Kubernetes cluster, where at least half plus one of the master nodes must remain healthy) but will instead depend on the constraints of the various applications installed within the architecture.

Chapter 6

Possible implementations

This chapter will discuss the potential implementations of the partial mesh star topology within the context of a computer network dedicated to energy monitoring. The network primarily consists of the Area Control Center, primary stations, and secondary stations, each managed by its own Kubernetes cluster.

6.1 Logical domains

This implementation is depicted in Figure 6.1, where the Area Control Center occupies the central position in the star topology, establishing unidirectional peering with offloading to every other entity in the topology, whether it is a primary station or a secondary station.

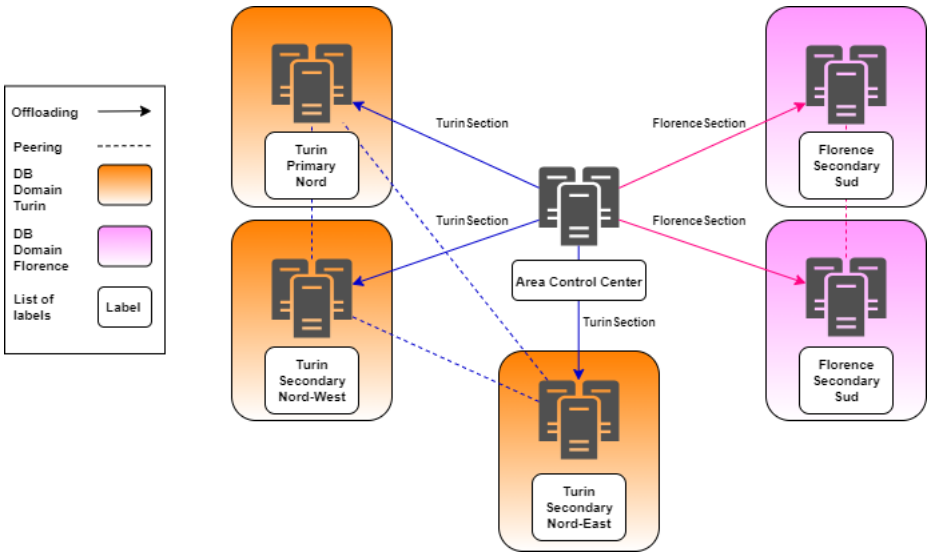

Figure 6.1. Logical domains scheme.

This allows the Area Control Center to manage all application deployments without the need to delegate them to other nodes. The remaining clusters are divided into groups, typically consisting of a primary station and its associated secondary stations. These groups represent a logical domain of applications with their own high-availability distributed database system and, therefore, do not have interconnections among them. Within a group, the clusters form a full mesh of unidirectional peerings for the database system's operation, and they share the same offloaded namespace from the Area Control Center.

6.1.1 Logical domains analysis

This architecture allows for the highest degree of resilience, as considering every possible failure in the control and power distribution network infrastructure, represented in Figure 6.2, the only fault that is not automatically recoverable is the disconnection of the Area Control Center.

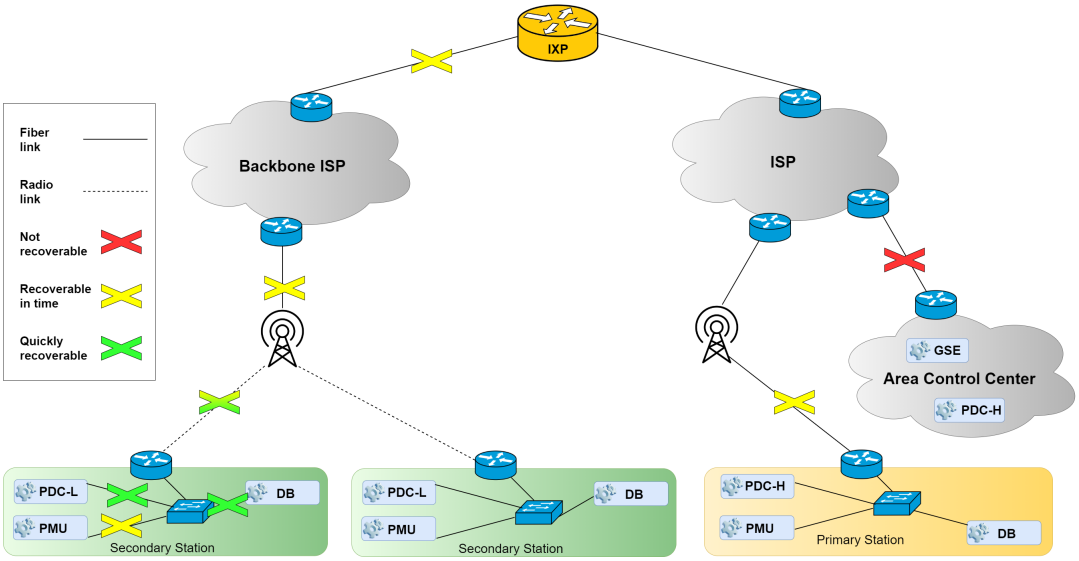

Figure 6.2. Possible failures in logical domains implementation.

The cluster representing the Area Control Center is a critical point as it hosts the central logic of the network, but the effects of a failure or disconnection of this node are negligible when compared to environmental constraints:

1. In the event of a physical failure of the central node, deployments would be lost, making the reconciliation process with the entire network impossible. However, this is negligible because without the central node's logic, the network would not be observable by default.

2. In the event of a complete disconnection from the network, active workloads would continue to function, but the reconciliation process for stateful applications (as the database system) will not occur since, by the point of view of the deployments, there would not be enough surviving replicas to maintain the system. Yet, this is also negligible as it falls under the same scenario as before.

In contrast, all other failures are recoverable from the perspective of the Area Control Center. Failures within a cluster are generally recoverable in a short time as the applications are automatically and quickly ricreated into a healthy node (which can belong to either the local cluster or a remote cluster) in case of disconnection or internal pod failure.

Disconnections of nodes, entire clusters, or parts of the network containing data production sources (PMUs) are automatically recoverable only with the restoration of the connection itself, as the PMU is physical hardware tied to its node and cannot be moved to others.

The aforementioned concerns the perspective of the Area Control Center, but as described in the previous chapter, the disconnected part of the network continues to function perfectly, and thanks to Liqo technology, additional applications can be instantiated to support the temporary independence of the network.

The continuation of operations can be observed in Figure 6.3, which illustrates the data stream about frequency values seen by an instance of PDC lower and its directly superior PDC higher, shortly before and shortly after the disconnection of the cluster hosting the PDC lower and its data sources, which occurred at 44,633 s. PDC lower continues to receive data from the sources, operating in an isolated environment, while PDC higher stops receiving the data stream from the isolated source.

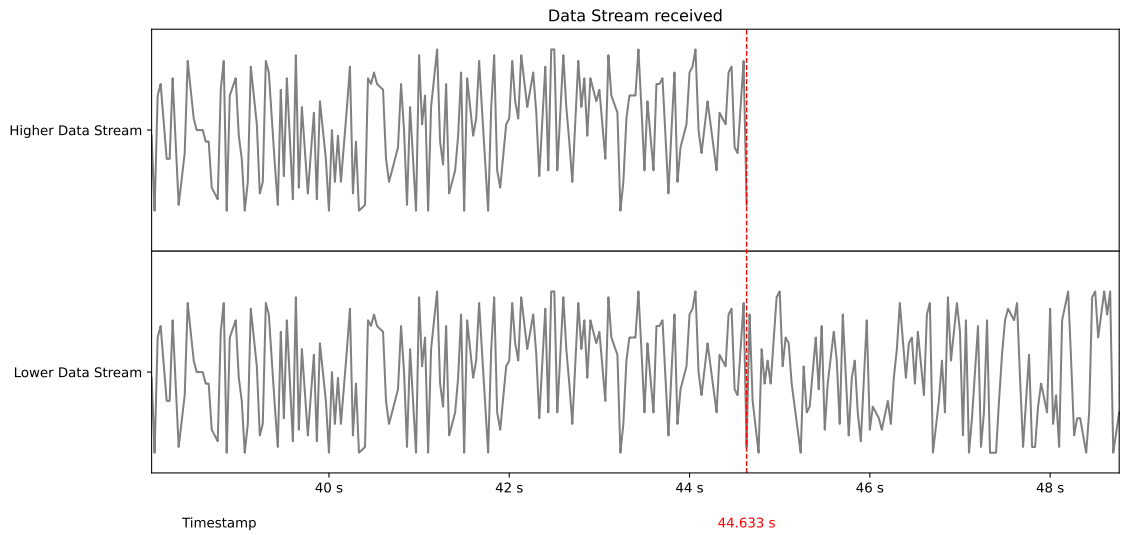

Figure 6.3. Data Stream comparison in case of failure.

Comparing the risks arising from the failure of one component with those that an

extended local solution encompassed, as shown in Table 6.1, it can be noted that many have decreased, as there is no longer the possibility of the network remaining unobservable. The risks related to PMU failures have remained the same, as it cannot be rescheduled to another node.

Component Failure	Severity	Cause
Single PMU	Low-> Low	Generally the number of other PMUs guarantees the observability threshold.
Multiple PMUs	Low-High-> Low-High	It depends on whether the number of other PMUs guarantees the observability threshold.
Single PDC-l	Moderate-High-> Low-Moderate	The observability of the network is impaired until the PDC is rescheduled onto another node.
Multiple PDCs-l	Moderate-High-> Low-Moderate	The observability of the network is impaired until the PDCs are rescheduled onto another nodes.
Single PDC-h	High-> Moderate	The observability of the network is impaired until the PDC is rescheduled onto another node.

Table 6.1. Component failures on Logical Domains topology overview.

The limitations of this architecture pertain to scalability, as each cluster requires its own distinct CIDR for the transparent operation of high-availability distributed database systems. Additionally, each peering creates a virtual representative node in the central cluster, limiting the number of possible clusters to 5000, according to the official Kubernetes documentation.

6.2 Multi-level logical domains

The implementation described in this section leverages a star topology twice, once with a partial mesh version and once with a full version, as shown in Figure 6.4. This follows the division of stations into primary and secondary, although it could be adapted to n subdivisions.

The first topology used is a partial mesh star topology used to connect the Area Control Center (central cluster) with all primary stations (leaf clusters). The central cluster handles the deployment of high-level applications along with their respective distributed

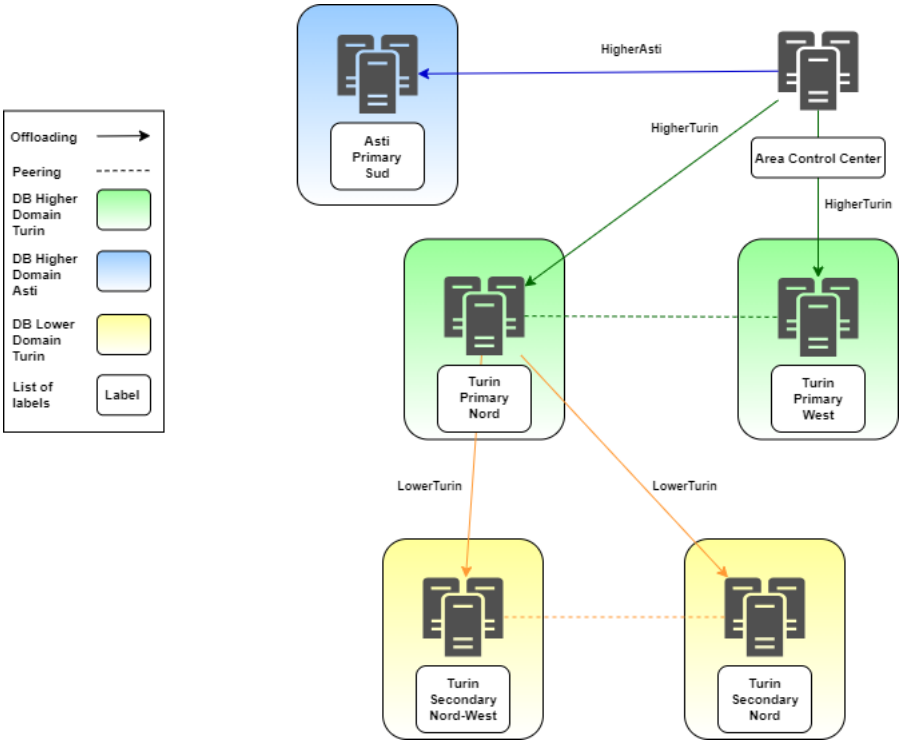

Figure 6.4. 2 Level logical domains.

database systems, offloading the corresponding namespace through peering to the respective group of primary stations.

The groups of primary stations are composed of a main primary station, where workload is preferably directed (using labels), while the others in the group primarily serve as backups in case of failure of the main station. This means that primary stations can be in multiple groups, one where they are primary and others where they function as backups, leveraging the topology seen in Chapter 5 section 5.2.1.

The main primary station of each group also serves as the central cluster in the second star topology, connecting not only to the backup primary stations but also to all secondary stations under its jurisdiction. In our implementation, this will be a full mesh star topology, but a partial mesh could also be used if the secondary stations do not share the same distributed database system.

In this second topology, the main primary station handles the deployment of low-level applications along with their respective high-availability distributed databases, consequently offloading the namespace to its secondary stations. The data stream for monitoring, which passes through two different namespaces (from the low-level to the high-level), relies on external exposure services such as load balancers and ingress, enabling access to the high-level application whether it resides in the primary station or, due to a failure, in one of the backup primary stations.

6.2.1 Multi-level Logical domains analysis

This architecture enhances scalability limits compared to the previous implementation by reducing the number of peerings managed by the Area Control Center, as shown in Figure 6.5, and by requiring distinct CIDRs only within the secondary topologies associated with a primary station, allowing for reuse across different types.

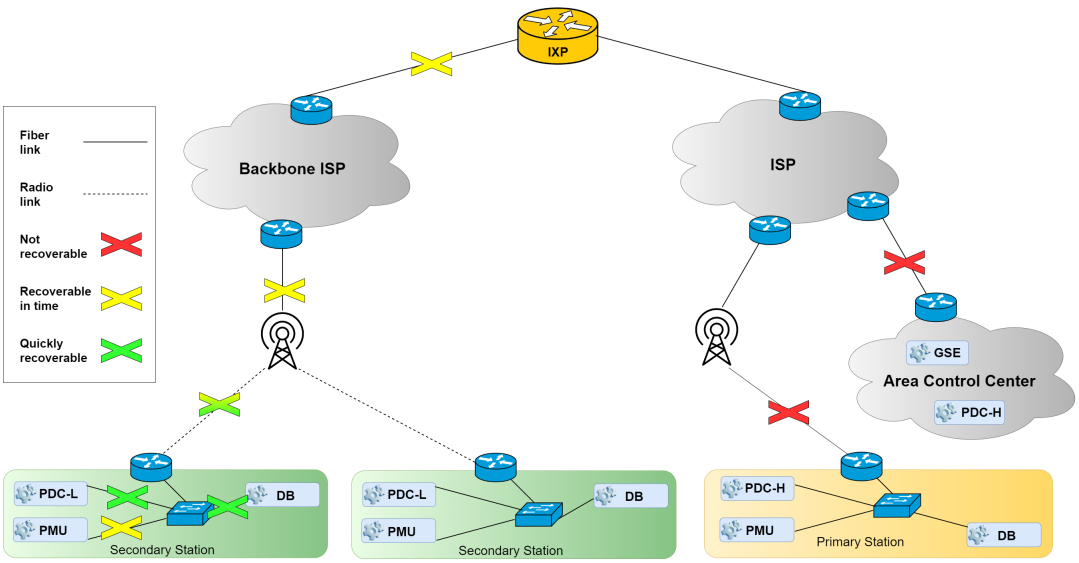

Figure 6.5. Possible failures in multi-level logical domains implementation.

However, this benefit is balanced by a decrease in overall resilience, because in addition to the critical point represented by the Area Control Center, all primary stations that manage a subnetwork of secondary stations also become critical points. This is because a failure or disconnection of the primary station results in the loss of deployment for low-level applications, which is not a feature supported by Liqo technology, thus necessitating a system reset upon reconnection.

Comparing the risks arising from the failure of one component with those that an extended local solution encompassed, as shown in Table 6.2, it can be noted that many have decreased, as there is no longer the possibility of the network remaining unobservable. The risks related to PMU failures have remained the same, as it cannot be rescheduled to another node.

This thesis focuses on achieving the highest degree of resilience; therefore, the following chapter will focus on testing the first implementation, as there is no risk of having to redeploy the software components in a subnetwork of secondary substations in the event of a failure of the entire associated primary station cluster.

It is important to note that these two implementations are not mutually exclusive; they can be implemented simultaneously within the same physical network, in cases where different parts of the network require varying degrees of resilience.

Component Failure	Severity	Cause
Single PMU	Low-> Low	Generally the number of other PMUs guarantees the observability threshold.
Multiple PMUs	Low-High-> Low-High	It depends on whether the number of other PMUs guarantees the observability threshold.
Single PDC-l	Moderate-High-> Low-Moderate	The observability of the network is impaired until the PDC is rescheduled onto another node.
Multiple PDCs-l	Moderate-High-> Low-Moderate	The observability of the network is impaired until the PDCs are rescheduled onto another nodes.
Single PDC-h	High-> Moderate	The observability of the network is impaired until the PDC is rescheduled onto another node.

Table 6.2. Component failures on Multi-level Logical Domains topology overview.

Chapter 7

Domain peering evaluation

In this chapter, we present the analyses conducted on the logical domain grouping implementation, chosen for its higher resilience compared to the multi-level implementation.

7.1 Test Environment

The test environment was created by leveraging the functionalities of Crownlabs, an open-source platform associated with the Politecnico di Torino, which was developed during the years of the Coronavirus spread.

7.1.1 Crownlabs

Crownlabs is an open-source project created to provide students with access to laboratory systems and services during the challenging times of the coronavirus pandemic, which imposed severe travel restrictions. In fact, the name derives from the virus itself and its initial purpose, as "Crown" translates to "Corona" in Italian and labs measn laboratories.

The authors of this project were a group of volunteers primarily composed of MSc students who, within just a few weeks of very hard work, as described on the project website [16], managed to deliver a functioning version, to address the university places closures mandated by the Italian government.

Nowadays, Crownlabs continues to be supported by students, and its functionalities have expanded: it not only allows the remote use of laboratory machines through a web browser, enabling both personal exercises and group work, but also leverages the Politecnico di Torino's data center to instantiate and use virtual machines transparently within an internal network. It is precisely this latter functionality that has been utilized, as the Kubernetes clusters used were composed of these virtual machines.

7.1.2 Nodes configuration

Each virtual machine representing a node in a cluster possesses the following characteristics, chosen to simulate low computational capacity typical of devices found in energy monitoring and distribution stations.

- **Operating system:** Ubuntu server 20.04 LTS.
- **CPU:** 4 core.
- **RAM:** 8 GB.
- **Disk memory:** 25 GB.

7.1.3 Software configuration

Di seguito vengono specificate le versioni delle piattaforme utilizzate:

- **K3s:** v1.24.17+k3s1.
- **Liqoctl:** v0.10.2.
- **Liqo:** v0.10.2.
- **PDC:** v2.4.
- **Database:** Percona XtraDB operator v1.11.0.
- **Database connector:** MYSQL connector v8.2.

It should be noted that certain parameters of the k3s kubelet and manager controller have been adjusted to decrease the cluster response time in case of failure, as detailed in the Table 7.1, and these values can be considered safe for networks of any size. In contrast, the virtual kubelet instantiated by Liqo has not undergone any changes.

Option	Value	Description
node-status-update-frequency	10s -> 5s	Specifies how often kubelet posts node status
node-monitor-grace-period	40s -> 20s	Specifies the amount of time in seconds that the Kubernetes Controller Manager waits for an update from a kubelet before marking the node unhealthy. Must be N times more than kubelet's nodeStatusUpdateFrequency, where N means number of retries allowed for kubelet to post node status
pod-eviction-timeout	300s -> 5s	This parameter specifies how long Kubernetes waits before evicting pod from a node marked as "NotReady"
node-monitor-period	5s -> 5s	The period for syncing NodeStatus in cloud-node-lifecycle-controller.

Table 7.1. Kubelet and Controller Manager list of parameter safe changes

These parameters can be reduced again to the values shown in Table 7.2. However, it is important to note that they are highly dependent on the size of the network.

Option	Value
node-status-update-frequency	10s -> 4s
node-monitor-grace-period	40s -> 12s
pod-eviction-timeout	300s -> 4s
node-monitor-period	5s -> 4s

Table 7.2. Kubelet and Controller Manager list of parameter min changes

7.1.4 Cluster configuration

Due to the limit of 5 virtual machines, the system was organized into 5 Kubernetes clusters, each comprising a single node. As shown in Figure 7.1, the topology is a fully-meshed star topology where the root cluster occupies the central position, hosting all deployments of the PMU, PDC, and database applications. Through Liqo peering, it offloads the test namespace to the leaf clusters.

The leaf clusters are connected by unidirectional peering for the transparent operation of the distributed Percona database system, and they will be the only locations where the pods of the aforementioned applications can be scheduled.

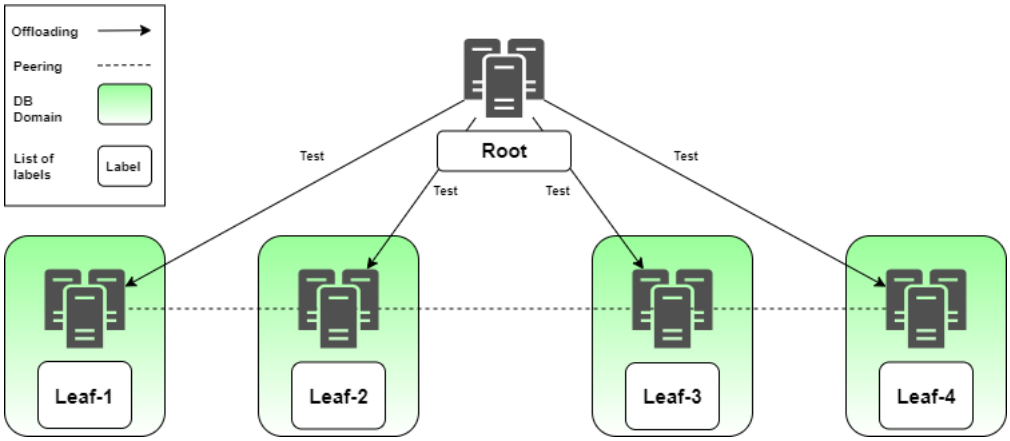

Figure 7.1. Configuration test environment

7.2 Latency

In this section, the latency increase due to the overhead generated by Liqo technology will be evaluated using 4 virtual machines. One machine is consistently used as the secondary member (Root), while the others represent the primary member (Leaves). The maximum tolerated data communication latency for state estimation applications is approximately 1000 ms.

Typically, data, as depicted in Figure 7.2, traverses about 4 clusters. The first two clusters usually consist of secondary stations, the third is the primary station overseeing the subnetwork of the two secondary stations, and the fourth is the Area Control Center containing the applications utilizing the data.

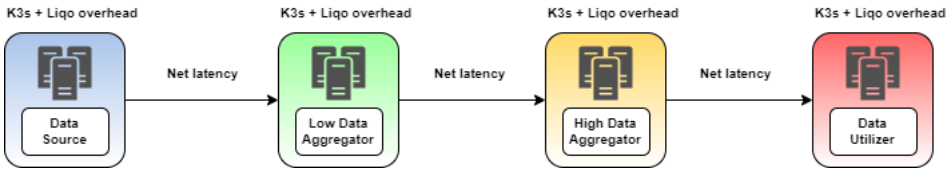

Figure 7.2. Data flow from source to destination

Therefore, the overall latency derives from the sum of Kubernetes + Liqo overhead for each individual cluster, plus the latencies of the networks connecting these clusters. The networks connecting these clusters typically consist of optical fiber networks for links between primary stations and Area Control Centers, and either fiber optic or radio links for the subnetwork of secondary stations.

According to the 2023 Development Plan of e-distribution [17], these networks include fiber optic cables illuminated with DWDM technology for connecting primaries to the control center, GPON fiber architectures, or 5G/LTE/4G radio links for connections between secondary and primary stations.

Let's consider the New AT/MT transformation station "Livigno" (SO) as presented in the 2023 development plan of e-distribuzione [18], as it represents an extreme case regarding the extension of the managed territory, approximately 400 km^2 in a rural environment. To estimate the network latency, we consider a fairly central location of the station within the area, hence the maximum distance between it and the secondary substations can be hypothesized to be about 30 km in the worst case. Over this distance, we consider a worst-case scenario of a 4G LTE network based on radio links, with cell towers approximately 5 km apart.

The network latency is calculated by adding the propagation speed over 30 km and the signal processing time for each cell tower. Assuming a signal processing time of about 20 ms and a propagation speed of 0.1 ms, the total latency would be $20 * 6 + 0.1 = 120.1$ ms. For the distance between the primary station and its respective Area Control Center, a time of 1 ms can be assumed, as this technology operates at around tens of microseconds over 140 km. Therefore, the network latency considering one of the worst-case scenarios (use of radio links in a vast territory) can be approximated to about 122 ms, leaving a theoretical limit of 878 ms for the overhead of the technologies.

7.2.1 Latency test

Given that data in the architecture is transmitted using TCP protocols, which utilize acknowledgments (ACKs), latency is measured as the round-trip time of a packet. The tests were conducted using the ping command, with approximately 1000 iterations per test, individually executed from the Leaf machines to the Root machine.

Firstly, to establish a baseline for the tests, the network latency was calculated by averaging the mean of three ping values obtained from the virtual machines towards the root machine, as shown in the Table 7.3. In this and the following tables, the arithmetic average will be calculated because every measures ah the same importance, and the avg standard deviation will be calculated using the formula (7.1) because tha data are uncorrelated:

$$\sigma_{\text{mean}} = \sqrt{\frac{\sum_{i=1}^{n} \sigma_i^2}{n}} \tag{7.1}$$

where:

- σ_i is the standard deviation of the value i.
- n is the total number of values.

Virtual machine	**Latency**	σ
Leaf-1	0.689 ms	0.447 ms
Leaf-2	0.760 ms	0.731 ms
Leaf-3	0.715 ms	0.468 ms
Avg	0.721 ms	0.564 ms

Table 7.3. Network average latency

After establishing the network latency, we proceed to calculate the latency between a pod located on different Leaf nodes and a pod in the Root node, where the Leaf nodes and the Root node belong to the same cluster, shown in Table 7.4, or belong to different clusters peered with Liqo, shown in Table 7.5.

Cluster Node	**Latency**	σ
Leaf-1	0.735 ms	0.294 ms
Leaf-2	0.992 ms	0.568 ms
Leaf-3	0.936 ms	0.483 ms
Avg	0.888 ms	0.463 ms

Table 7.4. Latency between pod on different nodes, but on the same cluster

Remote Node	**Latency**	σ
Leaf-1	1.269 ms	0.724 ms
Leaf-2	1.400 ms	0.829 ms
Leaf-3	1.368 ms	0.903 ms
Avg	1.346 ms	0.822 ms

Table 7.5. Latency between pod on different clusters, peered with Liqo

The increase due to Liqo is measured by subtracting the average latency from Table 7.4, which is the sum of network latency + Kubernetes overhead, from the average latency shown in Table 7.5, which is the sum of network latency + Kubernetes overhead + Liqo overhead. This calculation yields the result shown in Table 7.6.

The result shows that the overhead added by Liqo is negligible compared to the total tolerated latency of 1000 ms, as well as compared to one of the worst-case scenarios such as the Livigno case.

Average Liqo Latency	σ
0.458 ms	0.679 ms

Table 7.6. Average Liqo Latency

The latency between pods on different Kubernetes clusters without multi-cluster technologies was not tested, as the goal is to demonstrate the latency increase using Liqo across different clusters compared to using a single Kubernetes cluster connecting all nodes.

7.3 K3s reaction time

In the upcoming test, two clusters of virtual machines connected via unidirectional Liqo peering were utilized: the consumer cluster and the provider cluster. The objective was to show the consumer cluster's response time in the event of disconnection of the virtual node representing the provider cluster, for any reason.

The test involved two scripts. The first script disabled the network interface on the virtual machine running Liqo in the provider cluster and recorded the timestamp. The second script executed a loop on the consumer cluster, running 'kubectl get node' every 0.4 seconds, and appending the output with a timestamp. (A shorter interval wasn't feasible due to the command execution time.)

The results are depicted in Graph 7.3.

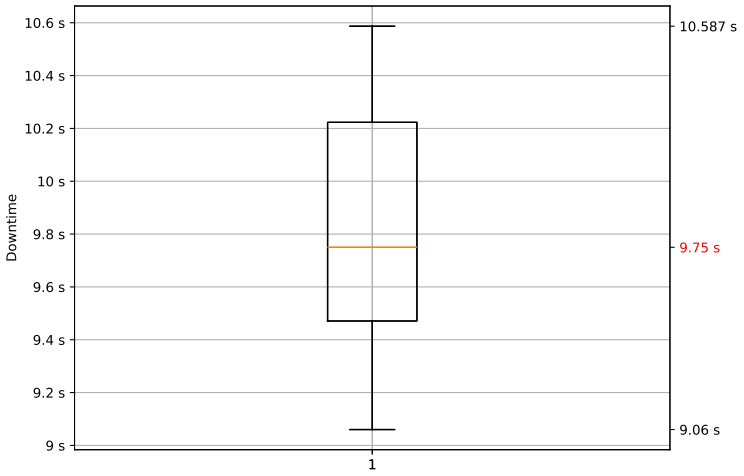

Figure 7.3. Reaction to set virtual node as Not Ready in case of remote cluster disconnession.

Comparing these results with findings from previous thesis [19] about the local solution using only Kubernetes, which utilized virtual machines running on different physical machines with slightly different parameter settings, it is evident that the order of magnitude remains consistent. This indicates that the introduction of Liqo technology does not add noteworthy delays compared to the typical delays of a straightforward Kubernetes architecture.

Moreover, using the same machines but connecting them through a single Kubernetes server and utilizing identical parameter settings, it was observed that the cluster response time in the event of a node failure is slightly higher by a few seconds compared to the time required to detect a node failure representing a Liqo cluster, as depicted in Figure 7.4. This difference stems from the Virtual Kubelet's unique management and health check optimization implemented by Liqo, which supersedes traditional kubelet functions for the virtual node management.

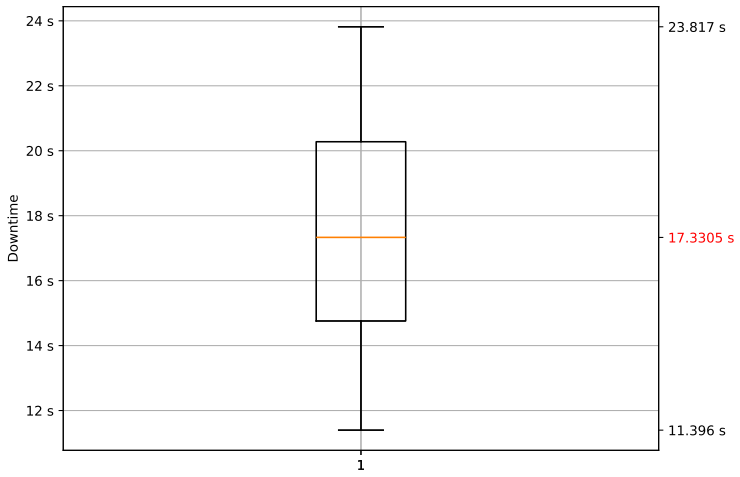

Figure 7.4. Reaction to set virtual node as Not Ready in case of local node disconnession.

7.4 Stream reaction time

The following tests demonstrate the downtime of a data stream from a PMU in the following scenarios:

1. Internal failure of a PDC pod, resulting in the rescheduling of the application.
2. Fault/disconnection of the cluster hosting a PDC pod, resulting in the application being rescheduled to another cluster.

Downtime is calculated from the timestamp of the last data frame of the old stream to the timestamp of the first data frame of the new stream, encompassing the time required for rescheduling the PDC pod, retrieving configurations from the database system, and reconnecting to the data stream.

These tests examine a data stream originating from a PMU that traverses through two PDC pod, one considered low-level and the other considered high-level, before reaching its intended application. The use of the data frame timestamp is crucial due to the PMU's real-time production of data frames at 33-millisecond intervals, ensuring precise downtime calculations and analysis.

7.4.1 Pod failure

The failure scenario of the PDC pod was simulated by customizing the liveness probe mechanism, intentionally triggering a failure check after the pod had been running for 60 seconds. The pod was located in a remote cluster managed by Liqo.

The test results, displayed in Graph 7.5, illustrate the median duration required for the lower-level PDC application to resume normal operation. This duration encompasses the time from detecting the PDC pod failure to its subsequent recovery, including the processes of restarting the pod, retrieving configurations from the system database, and re-establishing connection to the data stream.

Comparing these results with findings from previous thesis [19] about the local solution using only Kubernetes, where the pod recovery time ranged between 17-25 seconds, it is noted that here too, despite slight differences of a few seconds primarily due to environmental variables such as machine power, the order of magnitude remains consistent.

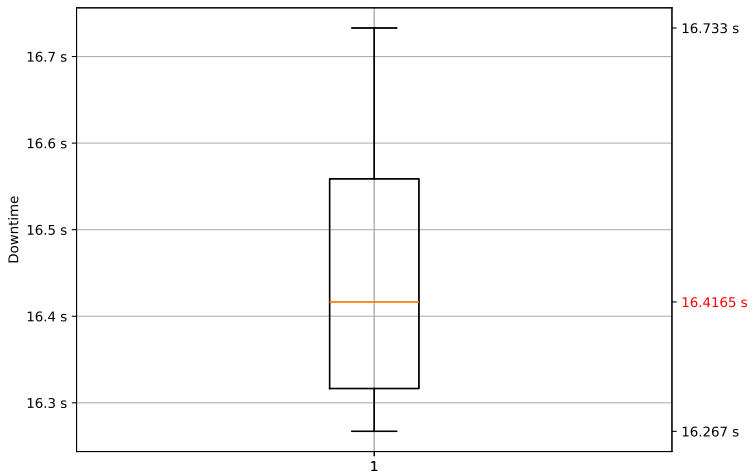

Figure 7.5. Box plot regarding stream downtime from last old data to the first new data in case of pod failure.

7.4.2 Cluster failure

The failure scenario was simulated by deliberately disabling the network interface of the lower-level PDC pod within the cluster. The deployment configuration of the lower-level PDC includes specific affinities to ensure that in the event of rescheduling, it can only be placed on another leaf cluster. As depicted in Figure 7.1, these leaf clusters are also managed by Liqo.

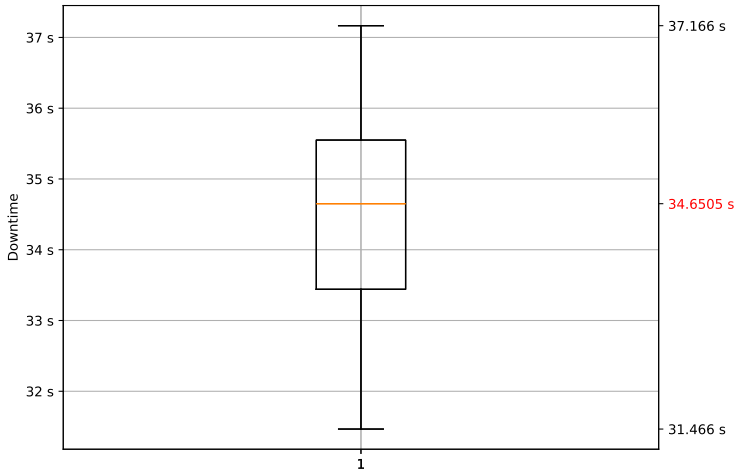

Figure 7.6. Box plot regarding stream downtime from last old data to the first new data in case of cluster failure.

The test results, displayed in Graph 7.6, illustrate the median duration required for the lower-level PDC application to resume normal operation. This duration includes the time required for the cluster to detect that the virtual node hosting the PDC is unreachable (9.75 seconds as shown in Graph 7.3), the waiting time before it can be rescheduled to another node (5 seconds as indicated in table 7.1), and the time necessary for the pod to restart (16.41 seconds as depicted in Graph 7.5).

Comparing the results with the values shown in Figure 7.7, illustrating findings from the previous thesis [19] focusing on the local solution using only Kubernetes, it is observed that the order of magnitude remains consistent. This reaffirms alongside previous tests that the introduction of Liqo technology does not introduce significant changes in terms of architectural overhead. In fact, in some cases, such as the optimization of the Kubelet, it enables better performance compared to the traditional Kubernetes architecture.

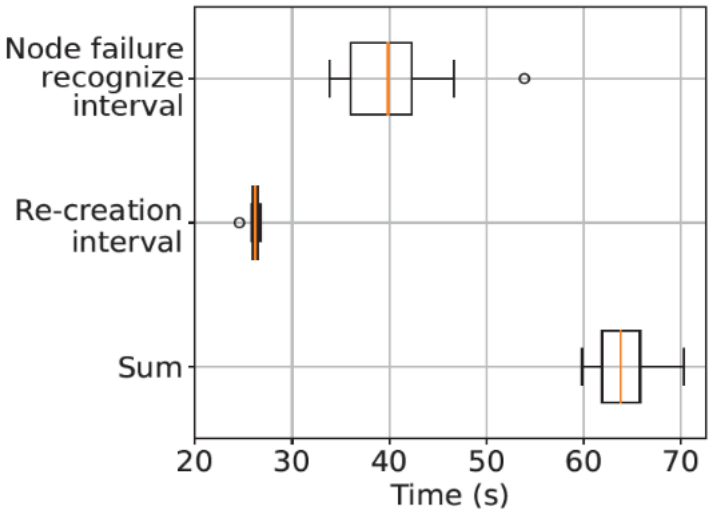

Figure 7.7. Time required to recover services on a disconnected node, on a traditional kubernetes cluster.

7.5 Overall evaluation

In this section, we summarize the findings from the various tests conducted to evaluate the performance, reliability, and resilience of the logical domain grouping implementation using Crownlabs and Liqo technology. The following points encapsulate the overall evaluation:

- **Test Environment and Configuration:** The test environment, meticulously configured using Crownlabs and Kubernetes clusters, accurately simulated real-world scenarios. The use of low-capacity virtual machines reflects typical setups in energy monitoring and distribution stations, ensuring relevance and applicability of the results.

  The fully-meshed star topology with a central root cluster efficiently managed the workload distribution and ensured robustness in case of node or cluster failures. The unidirectional peering for distributed database operations, as implemented through Liqo, facilitated seamless resource sharing and enhanced system reliability.

- **Latency Analysis:** The latency tests revealed that the additional overhead introduced by Liqo technology is minimal. The measured average Liqo latency of 0.458 ms is well within acceptable limits for state estimation applications, which tolerate up to 1000 ms. This demonstrates that Liqo can be effectively utilized in distributed systems without significantly impacting communication performance.

- **K3s Node Reaction Time:** The reaction time tests indicated that the introduction of Liqo technology did not introduce substantial delays. In fact, the Liqo-managed

virtual kubelet demonstrated optimized health checks and node management, resulting in slightly faster response times in certain failure scenarios compared to traditional Kubernetes setups.

- **Stream Reaction Time:** The tests simulating pod and cluster failures demonstrated that the downtime for resuming data streams was comparable to, if not slightly better than (in the case of cluster failure), traditional Kubernetes-only environments. This, along with new features such as the temporary independent management of a part of the network (with the continuation of the data stream inside) and the optimization of control capacity by a central entity to manage the rescheduling of the aggregator on other clusters, illustrates the significant advantages of introducing Liqo technology.
- **Reliability and Resilience:** Overall, the tests conducted on the logical domain grouping architecture have demonstrated its high resilience against both internal and external cluster failures. Throughout the testing process, there was not a single instance where the system failed to automatically converge to a new stable state after a fault was introduced. Moreover, in scenarios where a cluster became disconnected, the system reliably reintegrated the cluster into the architecture automatically once the connection was restored. This seamless reintegration highlights the robustness and efficiency of the architecture in maintaining operational continuity and stability, underscoring the significant advantages of adopting this approach.
- **Scalability and Flexibility:** The modular setup of clusters and the dynamic nature of Liqo peering allow for a high degree of adaptability, enabling the inclusion or exclusion of clusters without compromising the entire architecture. This is crucial for evolving system requirements and expanding infrastructure without significant overhauls. Scalability remains anchored to the official limits present in the Kubernetes documentation, although it has increased considerably since the set of nodes within a cluster is perceived as a single virtual node.
- **Comparison with Traditional Kubernetes:** Throughout the evaluations, it was evident that the logical domain grouping implementation leveraging Liqo provided performance metrics on par with those of traditional Kubernetes architectures. Additionally, the introduction of Liqo offers several benefits, such as isolated operation in case of disconnections, enhanced central control over the entire network, and the ability to manage cluster failures that a Kubernetes-only architecture cannot easily implement.

In conclusion, the analyses confirm that the chosen implementation of logical domain grouping offers a robust, scalable, and efficient solution for managing distributed systems. The minimal overhead introduced by Liqo, coupled with its advanced features, makes it a valuable addition to Kubernetes-based infrastructures, ensuring high resilience and optimal performance.

Chapter 8

Conclusion and future work

The introduction of Liqo technology, used to implement the paradigms of Edge and Fog computing within the Smart Grid model, has not only increased scalability by condensing multiple nodes into a single virtual node but has also enabled the introduction of new functionalities that were previously difficult to implement. For example, in case of disconnection from the central network, the two parts of the network can operate autonomously, with the capability to deploy new applications until reconnection with the central network (island-mode operation).

The chosen topology in this thesis, humorously called the "winning" topology, for real implementation is the partial mesh star topology. Compared to the other possible architectures, this topology efficiently satisfies all constraints, all constraints, stemming from the transparent operation of non-multi-cluster native applications, such as distributed database systems, and from design constraints like seeking the lowest possible power consumption and high resilience. These results were described in the previous chapter, comparing them with the baseline solution values and demonstrating their similarity, without significant latency increases despite increased complexity.

Obviously, this solution does not represent a panacea for all issues; for instance, it is still quite limited in terms of scalability. There are various potential research direction, such as:

- Hierarchical physical topology: Explore the possibility of introducing a hierarchical physical layer, as exemplified in the second implementation in Chapter 6 or by modifying the Liqo code to support the offloading of a namespace that has already been offloaded, while maintaining the high level of reliability demonstrated in this thesis.
- Updating the software: This study utilized software versions from previous research to compare results and showcase the efficiencies enabled by Liqo technology. openPDC v2.4 currently only supports Kubernetes until v1.24 and Percona until v1.11.0. Upgrading this software would enable the use of recent functionalities, such as the use of the Kubernetes spread operator in the newer versions of Percona, reducing complexity when creating logical topologies with labels and affinity.
- Security: Future research could focus on investigating the security implications of

deploying such decentralized systems and the potential damages that various cyber attacks can cause. Furthermore, collaboration with industry partners could facilitate the transition from theoretical research to practical, real-world applications.

In conclusion, this thesis has demonstrated that integrating Liqo technology into Smart Grid models significantly enhances scalability and functionality comparing to the previous solutions. While challenges remain, the groundwork laid here provides a solid foundation for future advancements. The continued evolution and optimization of these technologies promise to drive significant improvements in the efficiency and resilience of critical infrastructures.

Bibliography

[1] Wikipedia. Dispacciamento, 2024. URL https://it.wikipedia.org/wiki/Dispacciamento. (citation on page 9).

[2] Terna. Terna, 2024. URL https://www.terna.it/it. (citation on page 9).

[3] Kubernetes documentation. Kubernetes overview, 2024. URL https://kubernetes.io/docs/concepts/overview/. (citation on page 13).

[4] RAFT documentation. Raft algorithm, 2024. URL https://raft.github.io/. (citation on page 15).

[5] Sidero Labs. Split brain scenario, 2024. URL https://www.siderolabs.com/blog/why-should-a-kubernetes-control-plane-be-three-nodes/. (citation on page 15).

[6] K3s documentation. Best environment for k3s, 2024. URL https://docs.k3s.io/. (citation on page 15).

[7] Sebastian Böhm and Guido Wirtz. Profiling lightweight container platforms: Microk8s and k3s in comparison to kubernetes., 2021. (citation on page 16).

[8] Andrea Cazzaniga Fabrizio Garrone Roberta Terruggia Riccardo Lazzari Stefano Galantino, Fulvio Risso. An edge-based architecture for phasor measurements in smart grids. *2022 AEIT International Annual Conference (AEIT)*, pages 1–6, 2022. (citation on page 21).

[9] Sebastiano La Terra. Analysis of the resilience of monitoring services in smart grid. 2022. URL https://webthesis.biblio.polito.it/24583/. (citation on page 21).

[10] K3s documentation. Large cluster consideration, 2024. URL https://kubernetes.io/docs/setup/best-practices/cluster-large/. (citation on page 22).

[11] Liqo documentation. Liqo definition, 2024. URL https://docs.liqo.io/en/v0.11.0-rc.3/. (citation on page 25).

[12] Liqo documentation. Liqo ctl tool, 2024. URL https://docs.liqo.io/en/v0.11.0-rc.3/installation/liqoctl.html. (citation on page 27).

[13] Giuseppe Alicino. Prototyping a cloud resource broker. 2021. URL https://webthesis.biblio.polito.it/21145/. (citation on page 28).

[14] Riccardo Medina. Issue distributed database, 2024. URL https://github.com/liqotech/liqo/issues/2386. (citation on page 28).

[15] Marco Iorio, Fulvio Risso, Alex Palesandro, Leonardo Camiciotti, and Antonio Manzalini. Computing without borders: The way towards liquid computing. *IEEE Transactions on Cloud Computing*, 11(3):2820–2838, 2022. (citation on page 34).

[16] Crownlabs authors. Crownlabs history, 2024. URL https://crownlabs.polito.it/about/. (citation on page 45).

[17] e-distribuzione. Piano di sviluppo 2023, page 101, 2023. URL https://www.e-distribuzione.it/content/dam/e-distribuzione/documenti/piano-di-sviluppo/Piano_di_sviluppo_2023_ARERA.pdf. (citation on page 48).

[18] e-distribuzione. Piano di sviluppo 2023, page 51, 2023. URL https://www.e-distribuzione.it/content/dam/e-distribuzione/documenti/piano-di-sviluppo/Piano_di_sviluppo_2023_ARERA.pdf. (citation on page 48).

[19] Sebastiano La Terra. Analysis of the resilience of monitoring services in smart grid. pages 45–47, 2022. URL https://webthesis.biblio.polito.it/24583/. (citation on page 53).