Renewable Energy, Environment, and Power Quality Journal (REE&PQJ)
ISSN:3020-531X
Volume 26, July 2026

Where to Run the EMS Planner: Practical Cloud–Edge Placement for Two-Level Hierarchical EMS in Microgrids

G. Fernández[1], R. Estudillo[1], I. Sanz[1], J.F. Sanz-Osorio[2] and L. Elorza-Uriarte[3]

[1] Department of Electrical Systems, CIRCE Technology Centre,
50018 Zaragoza, Spain; gfernandez@fcirce.es, restudillo@fcirtce.es, isanz@fcirce.es

[2] Instituto Universitario de Investigación Mixto ENERGAIA
(Universidad de Zaragoza—Fundación CIRCE), 50018, Zaragoza, Spain; jfsanz@unizar.es

[3] Energy Storage System unit, CIDETEC (BRTA),
20014 Donostia-San Sebastián, Spain; lorenaelorza@cidetec.es

Abstract. The increasing penetration of distributed energy resources in microgrids is driving the adoption of Energy Management Systems (EMS) capable of coordinating generation, storage and flexible loads while meeting operational constraints. Two-level hierarchical EMS architectures, typically combining an upper-layer planner (rolling-horizon scheduling) with a lower-layer real-time controller, are a practical approach to cope with forecast uncertainty and fast dynamics. However, an open implementation question remains: where should the upper-layer EMS planner be executed—on the cloud, on an edge device, or in a hybrid configuration—given constraints on computation time, connectivity, and latency.

This paper proposes a lightweight cloud–edge placement tool to support this decision. The method uses a small set of measurable inputs (optimization time budget, problem size indicators, update period/granularity, and communication reliability/latency) to recommend a deployment option and to flag configurations that risk infeasibility or timeouts. The approach is validated on a representative two-level microgrid EMS workflow, comparing cloud and edge executions under different computational and communication conditions. Results show that the proposed tool correctly anticipates when edge execution becomes impractical due to solver time limits and when cloud execution may be penalized by communication constraints, providing actionable guidance for robust EMS deployment.

Key words. Microgrids, hierarchical energy management systems, cloud–edge computing, model predictive control, flexibility.

1. Introduction

The rapid growth of distributed energy resources (DER) such as photovoltaic generation, battery energy storage systems, electric vehicles and flexible loads is accelerating the transition from conventional power systems to smarter and more digitalized grids [1, 2]. This transition enables new opportunities as higher renewable hosting capacity, improved self-consumption and more efficient operation but also introduces relevant challenges, including increased variability, local constraints and a higher need for coordination across heterogeneous assets [3,4].

In this context, flexibility, known as the ability to modify generation and consumption patterns in response to external signals (prices, grid requests, congestion or renewable availability) has become a key enabling solution [5,6]. Flexibility can be provided internally optimizing behind-the-meter operation to reduce costs or improve self-consumption and externally offering services to the grid or aggregators [7]. Achieving this in a scalable and reliable way requires automated decision-making, which is typically delivered through an Energy Management System (EMS).

An EMS can be formulated as an optimization-based controller that schedules and dispatches microgrid resources under technical and economic constraints. Model Predictive Control (MPC) and rolling-horizon optimization are widely adopted approaches, as they can incorporate forecasts and update decisions as new information becomes available [8]. However, practical deployments often require two-level hierarchical EMS/MPC architectures, where an upper-layer planner computes optimal schedules over a horizon, and a lower-layer controller adapts setpoints locally to account for fast dynamics and forecast errors [9].

A key implementation question remains open: where should the upper-layer EMS planner be executed—on the cloud, on an edge device, or using a hybrid approach? This decision is not trivial because smaller simulation granularity and frequent re-optimization can improve objective performance and robustness to uncertainty [10, 11, 12], but they increase computational burden and may violate real-time deadlines, especially on constrained edge hardware [13]. In addition, remote execution introduces communication latency and reliability constraints. To address this gap, this work proposes a

practical methodology to support cloud–edge placement of the upper EMS layer, primarily evaluating trade-offs in terms of execution time feasibility (deadlines/timeout risk) and objective-function value (operational performance) under rolling-horizon operation of EMS/MPCs.

The rest of this paper is organized as follows. Section 2 outlines the proposed cloud–edge placement methodology and the associated tool, including the main workflow, inputs and decision logic. Section 3 summarizes the case study set-up and the preliminary results used to illustrate the approach. Finally, Section 4 concludes the paper and discusses next steps.

2. Methodology: Cloud–Edge Placement Tool

In this section, a concise description of the proposed cloud–edge placement approach is provided. First, the overall methodology is outlined through a high-level block diagram, and then the key inputs, assumptions and KPIs used to characterize both the EMS planning problem and the main execution constraints are summarized.

A Methodology overview

The proposed methodology provides a practical workflow to assess where the upper-layer EMS planner should be executed (cloud, edge or hybrid) under rolling-horizon operation. The workflow is organized as an iterative loop in which, at each control update, the current system state (e.g., stored energy levels and operational status) and the most recent forecasts are used to formulate and solve the planning problem over a finite horizon. The resulting schedule is then applied for the next time step (or control window), the system state is updated using measurements, and the process repeats under the rolling horizon approach.

The rolling-horizon workflow presented in this paper, see Fig.1, follows a structured sequence of initialization, iterative optimization and post-processing stages, enabling the evaluation of EMS performance under different cloud–edge deployment conditions.

1) Model and simulation setup:

The first step consists of loading the microgrid model, which includes the full mathematical representation of the system. This model integrates:

- Physical constraints (power balances, device limits, efficiencies).
- Economic parameters (energy prices, operational costs).
- Asset-specific models (BESS, electric vehicles, hydrogen systems and loads).

In addition, the initial state of the system is defined, including:

- State of charge (SoC) of energy storage systems (BESS, EVs).
- Hydrogen storage levels.
- Availability of controllable devices.

The simulation parameters are then configured, including:

- Optimization horizon H (e.g., 24–72 hours)
- Temporal granularity Δt (e.g., 10 min, 15 min, 1 h),
- Total simulation length (e.g., one month of forecasts or historical data),
- Solver and execution constraints (time limits, platform characteristics).

2) Initialization of simulation variables:

Before starting the rolling-horizon loop, the time index t is initialized at the beginning of the dataset and the initial system state is set according to the defined SoC and storage conditions. These values act as the initial condition for the first optimization problem.

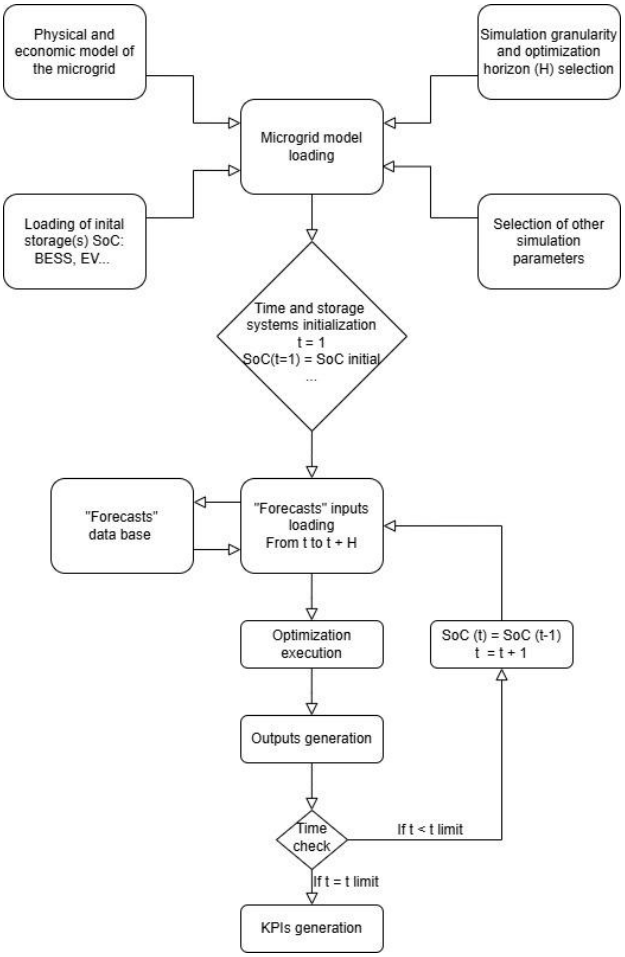

Fig. 1. Rolling-horizon workflow of the two-level hierarchical EMS, showing data inputs, optimization execution, state update and KPI calculation (applicable to cloud/edge deployments).

3) Rolling-horizon iterative loop:

The EMS is then executed in a receding-horizon way, iterating over the full simulation dataset. At each iteration:

- Data extraction: A time window of forecast data is extracted from the database, starting at the current time step t and spanning the full optimization horizon H. This includes demand, renewable generation, prices and other relevant inputs.
- Optimization problem solving: The EMS optimization problem is formulated using the extracted data and the current system state. The

solver computes an optimal schedule for all controllable assets over the horizon.

- Set-point application (receding horizon): Only the first control action (corresponding to the current time step t) is applied to the system. The remaining planned trajectory is discarded, as it will be recalculated in the next iteration.
- State update: The system state is updated based on the applied control actions. In particular:
    - The SoC of BESS and EVs is updated according to charging/discharging decisions.
    - Hydrogen storage levels are updated.
    - Any other dynamic state variables are propagated.
- Time advancement: The time index is incremented by one step and the process is repeated:

$$t \leftarrow t+\Delta t$$

This loop continues until the end of the available forecast or historical dataset is reached. In this way, the EMS simulates continuous real-time operation over long periods (e.g., weeks or months), while continuously correcting decisions based on updated system states.

4) KPI computation and result aggregation:

After completing the rolling-horizon simulation, performance indicators are computed over the full simulation period. These include:

- Objective function statistics (minimum, maximum, average values).
- Execution time metrics (minimum, maximum, average solver runtime).

These KPIs provide a quantitative basis to compare different configurations (granularity, horizon, cloud vs edge execution) and to assess the trade-off between operational performance and computational feasibility.

This algorithmic structure enables a realistic emulation of EMS operation in practice, where decisions are continuously updated based on new information and system evolution. Furthermore, by maintaining the same workflow across different execution platforms, the methodology ensures a consistent and fair comparison of cloud, edge, and hybrid deployment options. The methodology is designed to be microgrid-agnostic (it can represent different portfolios of DER and flexible loads through appropriate models), platform-agnostic (cloud, edge, or mixed deployments), and solver-agnostic (any mathematical programming solver or optimization engine can be used, provided it returns feasible schedules within the required time budget). This abstraction allows the same workflow to be applied across different microgrid configurations and computational environments, enabling a consistent comparison of operational performance and execution feasibility when evaluating cloud–edge placement options.

B Inputs and assumptions

The proposed methodology assumes that the rolling-horizon planner is configured and executed using standardized input data files, and that each execution produces both operational setpoints and relevant execution metadata.

A deterministic optimization framework is considered, consistent with the reference two-stage EMS approach. In this context, all uncertainties (e.g., demand, renewable generation, or prices) are represented through forecasts that are updated at each iteration of the rolling-horizon process.

The execution environment (cloud, edge, or hybrid) is not embedded in the optimization model itself, but is instead treated as an external constraint. In practice, this constraint is mainly defined by the available computational resources and execution time budget, and—when remote execution is considered—by communication latency and reliability.

This separation between the optimization model and the execution environment allows the same EMS formulation to be evaluated under different deployment scenarios, enabling a consistent comparison of feasibility and performance across cloud and edge implementations.

This methodology has been programmed and tested in PYTHON environment, using the following main libraries:

- PYTHON, version 3.9;
- PANDAS, version 2.3.3;
- NUMPY, version 2.0.2;
- ORTOOLS, version 9.14.6206, and SCIP as backend solver.

C Key Performance Indicators (KPIs) used

The performance of the proposed methodology is evaluated using a reduced set of Key Performance Indicators (KPIs) that capture both operational performance and computational feasibility:

1) Objective function KPIs (operational performance):

These indicators evaluate the quality of the EMS decisions over the simulation period:

- Maximum value [€].
- Minimum value [€].
- Average value [€].

2) Execution-time KPIs (computational feasibility):

These indicators evaluate whether the EMS can be executed within real-time constraints:

- Maximum execution time [s].
- Minimum execution time [s].
- Average execution time [s].

Among these indicators, the maximum execution time is particularly relevant, as it determines whether the EMS can meet the time constraints imposed by the selected temporal granularity. If the maximum execution time approaches or exceeds the control interval, the

configuration is considered at risk of infeasibility due to potential timeouts.

These KPIs enable a direct comparison between different configurations (granularity, horizon and execution platform), highlighting the trade-off between improved operational performance and increased computational requirements.

3. Case study and results

This section summarizes the reference case study used to illustrate the proposed methodology and reports the key numerical outcomes through a compact set of tables with emphasis on execution-time feasibility and objective-function impact.

A Case study Set Up

The case study follows the generic microgrid configuration and two-stage EMS workflow described in the reference paper "Greedy-VoI Time-Mesh Design for Rolling-Horizon EMS: Optimizing Block-Variable Granularity and Horizon Under Compute Budgets" [14] and adopts the same modelling assumptions and rolling-horizon planning framework used for the tests reported in that work.

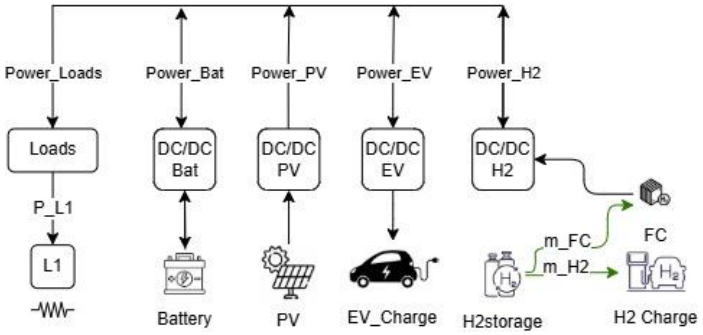

Fig. 2. Microgrid model used in the optimization process to test the proposed algorithm [14].

B Results

The first numerical results are summarized in Tables I to V, which report the objective-function statistics and the corresponding execution-time indicators for different temporal granularities and execution platforms (cloud and edge).

Table I. - Granularity and platform effect on the objective function and execution times - cloud execution and 10 min granularity

Objective Function Comparison	
Max [€]	−25.49
Min [€]	−109.37
Average [€]	−72.07
Execution Time Comparison	
Max execution time [s]	301
Min execution time [s]	2
Average execution time [s]	15

Table II. - Granularity and platform effect on the objective function and execution times - cloud execution and 15 min granularity

Objective Function Comparison	
Max [€]	−25.55
Min [€]	−105.34
Average [€]	−72.12
Execution Time Comparison	
Max execution time [s]	149
Min execution time [s]	1
Average execution time [s]	8

Table III. - Granularity and platform effect on the objective function and execution times - cloud execution and 1 hour granularity

Objective Function Comparison	
Max [€]	−22.3
Min [€]	−84.7
Average [€]	−49.39
Execution Time Comparison	
Max execution time [s]	6
Min execution time [s]	<1
Average execution time [s]	1

Table IV. - Granularity and platform effect on the objective function and execution times - edge execution (500 MB Limit) 10 min granularity

Objective Function Comparison	
Max [€]	−25.43
Min [€]	−108
Average [€]	−72.1
Execution Time Comparison	
Max execution time [s]	300
Min execution time [s]	<1
Average execution time [s]	87

Table V. - Granularity and platform effect on the objective function and execution times - edge execution (500 MB Limit) 15 min granularity

Objective Function Comparison	
Max [€]	−25.38
Min [€]	−105.99
Average [€]	−72.1
Execution Time Comparison	
Max execution time [s]	300
Min execution time [s]	<1
Average execution time [s]	35

Overall, finer temporal resolutions (10–15 min) achieve a clearly better objective-function performance than the coarse 1 h discretization, with very similar objective values between cloud and edge for the same granularity. However, this improvement comes at a computational cost: execution times increase sharply when moving from 1 h (max ≈ 6 s) to 15 min (max ≈ 149 s in cloud, and up to the 300 s cap in edge) and to 10 min (worst-case ≈ 301 s, i.e., effectively reaching the time limit). These results highlight the core trade-off addressed by the placement methodology: small granularities can improve operational performance, but they may violate practical execution budgets—especially on constrained edge deployments—while cloud execution reduces this risk at 15 min but still faces worst-case peaks at 10 min.

These trends directly inform where the EMS planner should run. If fine granularities are needed to improve

performance, edge execution becomes risky because worst-case runtimes can approach the rolling-horizon deadline, leaving little slack and increasing timeout probability; cloud execution is then preferable to preserve feasibility margins. If coarser granularities already deliver acceptable performance, edge execution may be sufficient and reduces reliance on communications. In short, the platform choice should follow the trade-off between objective-value gains and worst-case runtime/deadline risk.

In addition to the solver execution time, it is important to consider the full sequence of operations involved in a real EMS deployment. These include data acquisition from physical assets, communication between edge and cloud layers, forecast generation, and post-processing of optimization results, all of which contribute to the total control-cycle time.

In a cloud-based execution, the workflow typically involves a distributed architecture. A local edge device (e.g., gateway or industrial controller) continuously monitors physical assets (BESS, PV, EV chargers, etc.), collects measurements and preprocesses the data. This information is then transmitted to the cloud, where it is stored and combined with external data sources (e.g., weather forecasts or market prices retrieved via APIs such as AEMET or OMIE). Based on this aggregated dataset, forecasting algorithms are executed, followed by the EMS optimization process. The resulting setpoints are then post-processed and sent back to the local gateway, where they are applied to the physical system.

In contrast, in an edge-based execution, most of these operations are performed locally. The edge device directly monitors and processes asset data, retrieves external information through APIs and stores the relevant data in a local database. Forecasts are computed locally (typically using simplified or pre-trained models) and the EMS optimization is executed on the same device. The resulting setpoints are then immediately applied to the system without requiring remote communication. However, all these tasks must share the same limited computational resources and are often executed alongside other background processes (e.g., communication services, data logging or device control), which can further constrain the available execution time.

In practical terms, these components typically correspond to:

- External data acquisition (APIs): ~1–3 s
- Local data acquisition (gateways/SCADA): ~0.1–0.5 s
- Forecast computation: typically ~0.5–5 s, depending on model complexity and execution platform
- Communication latency (cloud): ~0.1–1 s per direction
- Optimization (as shown in Tables I–V): ~1–300 s depending on granularity

It should be noted that forecast computation time is highly dependent on the deployment architecture. Cloud environments can execute more complex models faster due to higher computational resources, while edge implementations may achieve lower latency when using simpler or pre-computed models. Therefore, the proposed range reflects typical practical values rather than a fixed assignment to cloud or edge execution.

These estimations show that, although optimization time remains the dominant factor, the additional overhead associated with data handling, forecasting, and communication can significantly reduce the available time margin. This effect is particularly critical in edge deployments, where all processes compete for limited computational resources and worst-case optimization times already approach the control interval. In cloud deployments, higher computational capacity alleviates this constraint but introduces additional latency and dependency on communication reliability.

4. Conclusions

The ongoing growth of DER portfolios and flexibility-driven operation is pushing EMS deployments from “algorithm design” to real-world engineering, where computation deadlines, platform constraints, and communication reliability become decisive. In this context, a practical cloud–edge placement methodology is needed to ensure that upper-layer rolling-horizon planners remain both operationally and deployable under realistic execution budgets. The proposed approach is designed to be robust and broadly applicable—agnostic to the specific microgrid configuration, execution platform, and solver—providing actionable guidance for selecting cloud, edge, or hybrid execution depending on the trade-off between performance gains and worst-case runtime risk.

The test results confirm that the deployment of the upper-layer EMS planner is strongly constrained by real-time execution requirements rather than only by optimization performance. Finer temporal granularities (10–15 min) improve the objective-function value but significantly increase execution times, often approaching practical limits in edge environments. When considering the full EMS workflow, including data acquisition, forecasting, and communication, the available time margin is further reduced, increasing the risk of infeasibility. As a result, edge execution becomes less reliable for fine granularities, while cloud execution provides higher robustness at the cost of communication dependency.

Future work will extend the validation to a wider set of microgrid models, asset portfolios and operating scenarios (including different forecast qualities, flexibility requests, and communication conditions).

Acknowledgement

This paper shows the results of a thorough research and work carried out in the REEFLEX project. REEFLEX

has received funding from the European Union's Horizon Europe Research and Innovation program under Grant Agreement No. 101096192.

References

[1] Drgona, J.; Arroyo, J.; Figueroa, I.C.; Blum, D.; Arendt, K.; Kim, D.; Ollé, E.P.; Oravec, J.;Wetter, M.; Vrabie, D.L.; et al. All you need to know about model predictive control for buildings. Annu. Rev. Control 2020, 50, 190–232.
[2] Fang, X.; Misra, S.; Xue, G.; Yang, D. Smart Grid—The New and Improved Power Grid: A Survey. IEEE Commun. Surv. Tutor. 2012, 14, 944–980.
[3] Zhou, E.; Hale, E.; Stephen, G.; Radhakrishnan, N. Reducing Grid Costs while Abating Emissions: Opportunities for Flexible Building Loads; National Renewable Energy Laboratory (NREL): Golden, CO, USA, 2023; NREL/TP-6A40-84828.
[4] Shen, F.; Huang, S.; Wu, Q.; Repo, S.; Xu, Y.; Østergaard, J. Comprehensive Congestion Management for Distribution Networks based on Dynamic Tariff, Reconfiguration and Re-profiling Product. IEEE Trans. Smart Grid 2019, 10, 4795–4805.
[5] Strezoski, L. Distributed energy resource management systems—DERMS: State of the art and how to move forward. In Wiley Interdisciplinary Reviews: Energy and Environment; John Wiley and Sons Ltd.: Hoboken, NJ, USA, 2023; Volume 12.
[6] ENTSO-E. System Flexibility Needs for the Energy Transition: Executive Summary; ENTSO-E—European Network of Transmission System Operators for Electricity: Brussels, Belgium, 2024.
[7] Striani, S.; Unterluggauer, T.; Andersen, P.B.; Marinelli, M. Flexibility potential quantification of electric vehicle charging clusters. Sustain. Energy Grids Netw. 2024, 40, 101547.
[8] Hu, J.; Shan, Y.; Guerrero, J.M.; Ioinovici, A.; Chan, K.W.; Rodriguez, J. Model predictive control of microgrids—An overview. Renew. Sustain. Energy Rev. 2021, 136, 110422.
[9] Fernández, G.; Sanz Osorio, J.F.; Rocca, R.; Luengo-Baranguan, L.; Torres, M. Practical Considerations for the Development of Two-Stage Deterministic EMS (Cloud–Edge) to Mitigate Forecast Error Impact on the Objective Function. Appl. Sci. 2026, 16, 1844.
[10] Mayne, D.Q.; Rawlings, J.B.; Rao, C.V.; Scokaert, P.O.M. Constrained model predictive control: Stability and optimality. Autom.2000, 36, 789–814.
[11] Qin, S.J.; Badgwell, T.A. A survey of industrial model predictive control technology. Control. Eng. Pract. 2003, 11, 733–764.
[12] Hoffmann, M.; Kotzur, L.; Stolten, D.; Robinius, M. A Review on Time Series Aggregation Methods for Energy System Models. Energies 2020
[13] Hu, J.; Shan, Y.; Guerrero, J.M.; Ioinovici, A.; Chan, K.W.; Rodriguez, J. Model predictive control of microgrids—An overview. Renew. Sustain. Energy Rev. **2021**, 136, 110422.
[14] Fernández, G.; Sanz Osorio, J.F.; Alarcón, A.; Torres, M.; Calavia, A. Greedy-VoI Time-Mesh Design for Rolling-Horizon EMS: Optimizing Block-Variable Granularity and Horizon Under Compute Budgets. Smart Cities 2026, 9, 30.