A Cognitive Serverless Framework for the Cloud-Edge Continuum

D4.4 COGNIT Serverless Platform - Scientific Report - d

Version 1.0

30 April 2025

Abstract

COGNIT is an AI-enabled Adaptive Serverless Framework for the Cognitive Cloud-Edge Continuum that enables the seamless, transparent, and trustworthy integration of data processing resources from providers and on-premises data centres. The continuum and their automatic and intelligent adaptation to optimise where and how data is processed according to application requirements, changes in application demands and behaviour, and the operation of the infrastructure in terms of the main environmental sustainability metrics. This document describes the research and development carried out in WP4 "AI-enabled Distributed Serverless Platform and Workload Orchestration" during the Fourth Research & Innovation Cycle (M22-M27) according to the new COGNIT architecture (see details in D2.4), providing details on the status of a number of key components of the COGNIT Framework (i.e., Cloud-Edge Manager and AI-Enabled Orchestrator) as well as reporting the work related to supporting Energy Efficiency Optimization in the Multi-Provider Cloud-Edge Continuum.

This project is funded by the European Union's Horizon Europe research and innovation programme under Grant Agreement 101092711 – SovereignEdge.Cognit

Deliverable Metadata

Project Title:	A Cognitive Serverless Framework for the Cloud-Edge Continuum
Project Acronym:	SovereignEdge.Cognit
Call:	HORIZON-CL4-2022-DATA-01-02
Grant Agreement:	101092711
WP number and Title:	WP4. AI-enabled Distributed Serverless Platform and Workload Orchestration
Nature:	R: Report
Dissemination Level:	PU: Public
Version:	1.0
Contractual Date of Delivery:	31/03/2025
Actual Date of Delivery:	30/04/2025
Lead Author:	Monowar Bhuyan (UMU) & Paul Townend (UMU)
Authors:	Yashwant Singh Patel (UMU), Adil Bin Bhutto (UMU), Rohail Gulbaz (UMU), Malik Bouhou (CETIC), Aritz Brosa (Ikerlan), David Eklund (RISE), Javad Forough (UMU), Idoia de la Iglesia (Ikerlan), Iván Valdés (Ikerlan), Fátima Fernández (Ikerlan), Carlos Lopez (ACISA), Tomasz Korniluk (Phoenix), Johan Kristiansson (RISE), Antonio Lalaguna (ACISA), Marco Mancini (OpenNebula), Philippe Massonet (CETIC), Nikolaos Matskanis (CETIC), Mikel Irazola (Ikerlan), Álvaro Puente (Ikerlan), Juan José Ruiz (ACISA), Kaja Swat (Phoenix), Thomas Ohlson Timoudas (RISE), Iván Valdés (Ikerlan), Cristina Cruces (Ikerlan), Martxel Lasa (Ikerlan), Victor Palma (OpenNebula), Michal Opala (OpenNebula), Jakub Walczak (OpenNebula), Carlos Moral (OpenNebula), Aitor Garciancia (Ikerlan), Francesco Renzi (Nature 4.0), Riccardo Valentini (Nature 4.0)
Status:	Submitted

Document History

Version	Issue Date	Status[1]	Content and changes
0.1	22/04/2025	Draft	Initial Draft
0.2	24/04/2025	Peer-Reviewed	Reviewed Draft
1.0	30/04/2025	Submitted	Final Version

Peer Review History

Version	Peer Review Date	Reviewed By
0.2	24/04/2025	Idoia de la Iglesia (Ikerlan)
0.2	24/04/2025	Antonio Álvarez (OpenNebula)

Summary of Changes from Previous Versions

First Version of Deliverable D4.4

[1] A deliverable can be in one of these stages: Draft, Peer-Reviewed, Submitted, and Approved.

Executive Summary

This is the fourth “COGNIT Serverless Platform - Scientific Report” that has been produced in WP4 “AI-enabled Distributed Serverless Platform and Workload Orchestration”. It describes in detail the progress of the software requirements that have been active during the Fourth Research & Innovation Cycle (M22-M27) in connection with these main components of the COGNIT Framework:

Cloud-Edge Manager

- **SR4.6** Plan Executor:

  Plan Executor is responsible for the execution of plans produced by the AI-Enabled Orchestrator related to the placement and migration of Serverless Runtimes within an Edge Cluster.

AI-Enabled Orchestrator

- **SR5.1** Building Learning Models:

  Provide AI/ML models based on collected metrics from the Cloud-Edge Manager monitoring service related to Edge Clusters, Serverless Runtimes, and infrastructure usage.

- **SR5.2** Smart management of Cloud-Edge resources:

  AI-Enabled Orchestrator is responsible for managing and optimizing the lifecycle of Edge Clusters and serverless runtimes within Edge Clusters according to the application requirements, infrastructure and virtual resource usage, and energy-aware policies.

This deliverable has been released at the end of the Fourth Research & Innovation Cycle (M27). The final version, D4.5, to be released in M33, will be a standalone document including the results of the five research cycles.

Table of Contents

Abbreviations and Acronyms

ADWIN	Adaptive windowing
AE	Auto Encoder
AI	Artificial Intelligence
AI-O	AI-Enabled Orchestrator
API	Application Programming Interface
ARUR	Average Resource Utilization Ratio
CLI	Command Line Interface
CMA	Carbon-aware Model Agent
CPCA	Common Principal Component Analysis
CSP	Cloud Service Provider
DB	Database
DCs	Data Centres
DL	Deep Learning
DTW	Dynamic Time Warping
EC	Energy Consumption
EP	Energy Provider
EVPN	Ethernet VPN
FaaS	Function as a Service
FFNN	Feed-Forward Neural Network
GC	Global Controller
GPU	Graphics Processing Unit
GRU	Gated Recurrent Unit
HDBSCAN	Hierarchical Density-Based Spatial Clustering of Applications with Noise
HRUR	Host Resource Utilisation Ratio
HTTP	Hypertext Transfer Protocol
HVMC	Host-VM Combination
IDEC	Improved Deep Embedded Clustering
ILP	Integer Linear Programming
IP	Internet Protocol
IPAM	IP Address Management
JSON	Javascript Object Notation
KVM	Kernel Virtual Machine
LC	Local Controller
LSTM	Long Short-Term Memory

MAPE-K	Monitoring, Analysis, Planning, Execution, and Knowledge
MI	Million Instructions
MIPS	Million Instructions Per Second
ML	Machine Learning
MOGA	Multi-Objective Genetic Algorithm
MSE	Mean Squared Error
MTS	Multivariate Time Series
NSGA-II	Non-dominated Sorting Genetic Algorithm II
OS	Operating System
PDN	Power Distribution Network
PH	Page-Hinkley
PSRs	Power Source Regions
QoS	Quality of Service
RAE	Relative Absolute Error
RAPL	Running Average Power Limit
REST	Representational State Transfer
RMSE	Root Mean Squared Error
RNN	Recurrent Neural Network
ROCKET	Random Convolutional Kernel Transform
RUR	Resource Utilization Ratio
SGD	Stochastic Gradient Descent
SLA	Service Level Agreement
SLO	Service Level Objective
SoC	State of Charge
SPEA2	Strength Pareto Evolutionary Algorithm 2
SRTs	Serverless Runtimes
SVD	Single Value Decomposition
TCN	Temporal Convolutional Network
VM	Virtual Machine

1. Cloud-Edge Manager

The Cloud-Edge Manager is responsible for autonomous management of distributed cloud-edge continuum resources according to the application demand and availability of resources. This development cycle has made progress related to the SR4.6 software requirement (i.e. Plan Executor) that allows the execution of plans produced by the AI-Enabled Orchestrator related to the placement and migration of Serverless Runtimes within an Edge Cluster. Details of each software requirement are reported in Deliverable D2.5.

1.1 [SR4.6] Plan Executor

Description

The Cloud Edge Manager has been enhanced by integrating new components that have been developed in OpenNebula in the context of the European Project IPCEI-CIS (Important Project of Common European Interest on Next Generation Cloud Infrastructure and Services), which has co-funded such developments. In particular, we have integrated in the COGNIT Framework the new scheduler architecture of OpenNebula, including the Plan Executor and the Scheduler Manager. The Plan Executor allows the execution of plans by performing actions (e.g. deploy, migrate, poweroff, etc…) related to the Serverless Runtimes within an Edge Cluster. The plans are produced by the AI-Enabled Orchestrator that has been integrated within the Cloud-Edge Manager according to the architecture reported in Figure 1.1.

The Scheduler Manager interacts with the AI-Enabled Orchestrator to produce placement and optimization plans. The AI-Enabled Orchestrator optimization algorithms have been integrated by implementing new drivers that are used by the Scheduler Manager for placement and workload balancing of Serverless Runtime within an Edge Cluster.

The Plan Executor executes the plans produced by the Scheduler manager using the AI-Enabled Orchestrator drivers.

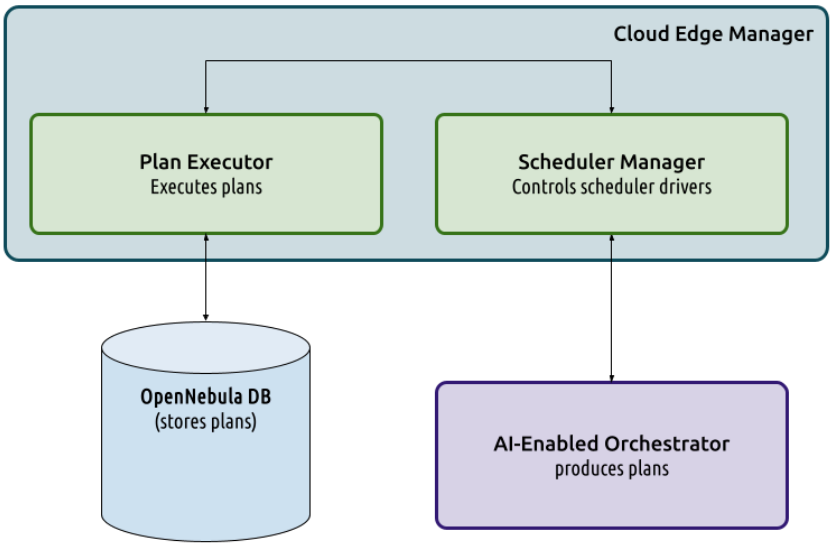


Figure 1.1: Scheduler Architecture

The plans are produced by the AI-Enabled Orchestrator according to the following XML schema:

Table 1.1: XML schema for plans produced by the AI-Enabled Orchestrator

XML PATH	Description
ID	ID of the cluster the plan applies to.
ACTION/	
VM_ID	ID of the Serverless Runtime the action applies to.
OPERATION	Serverless Runtime operation (deploy, migrate, poweroff).
HOST_ID	Host ID where for deploy and migrate operations.

Example of a plan produced by the AI-Enabled Orchestrator is reported below:

```
<PLAN>
  <ID>2</ID>
  <ACTION>
    <VM_ID>23</VM_ID>
    <OPERATION>deploy</OPERATION>
    <HOST_ID>12</HOST_ID>
  </ACTION>
  <ACTION>
    <VM_ID>24</VM_ID>
    <OPERATION>migrate</OPERATION>
    <HOST_ID>15</HOST_ID>
  </ACTION>
```

```
  <ACTION>
    <VM_ID>25</VM_ID>
    <OPERATION>poweroff</OPERATION>
  </ACTION>
</PLAN>
```

2. AI-Enabled Orchestrator

The AI-Enabled Orchestrator (AI-O) enables multiple features for smart management of distributed cloud-edge resources, including:

- Determining optimal number of serverless runtimes,
- Generating initial placement plan, and
- Optimizing resource allocation in the cloud-edge continuum.

More detailed description about the AI-Enabled Orchestrator is available in earlier Deliverable report D4.3. This development cycle makes progress on the learning models to categorize different resource utilization metrics, model retraining and improving AIOps pipeline for training, validation and model repository integration. Additionally, ongoing efforts focus on developing multi-objective evolutionary algorithms and advancing energy-aware continuum systems modelling, with validation conducted in simulated environments. These advancements contribute to optimizing resource management and sustainability in the cloud-edge continuum.

2.1 [SR5.1] Building learning models

In this development cycle, the AI-O has been implemented, with a specific focus on developing AI/ML models for unsupervised workload classification and model retraining.

2.1.1 AI/ML Models

Recent efforts (particularly using real workload traces such as Google Cluster [19], Microsoft Azure [32], PlanetLab [33], Bitbrains trace [34], and the Alibaba cluster [35]) have advanced workload modeling significantly - however, metrics generated by monitoring tools typically do not provide labeled data. To address this, unsupervised learning methods are useful, enabling models to learn directly from the data without requiring prior labels.

As part of AI/ML model development, The AI-O can leverage two types of clustering models for classifying the workloads: 1) classical K-means clustering, and 2) Deep K-means clustering.

1) Classical K-means Clustering: The classical K-means algorithm operates directly on the normalized input features extracted from the workloads. The clustering process begins with loading and pre-processing datasets from various clusters. Key resource features such as GPU utilization, memory utilization, and energy consumption are extracted and standardized to ensure uniform scaling. K-means iteratively partitions the feature space into K clusters by minimizing the squared Euclidean distance between data points and cluster centroids.

2) Deep K-means Clustering: To improve clustering accuracy, we incorporate deep feature learning using an autoencoder. The Deep K-means approach optimizes both reconstruction

loss and clustering loss simultaneously to learn meaningful latent representations. This enhances the separation of workloads in a lower-dimensional latent space.

2.1.2 Data

These models have been developed, tuned, validated and integrated with the AIOps pipeline in COGNIT using synthetic, simulated, and real-world datasets.

1) Synthetic dataset: To generate the synthetic dataset, a probabilistic cumulative modeling is adopted; which is based on probabilistic models which capture cumulative properties of real-world data distributions. Compared to the real-world dataset, the synthetic dataset introduces more significant variations in resource utilization patterns (e.g., GPU usage, memory usage, CPU usage, and energy usage) and carbon emissions (g·CO2eq/kWh).

2) Simulated dataset: Discrete-event simulators [3] are tools designed to simulate complex systems where events occur at specific points in time, triggered by predefined conditions, such as the scheduling of FaaS requests or the deployment of virtual machines. The proposed framework is evaluated using *ContinuumSim* [4], an open-source discrete-event simulator developed collaboratively by RISE and Umeå University in the COGNIT project. ContinuumSim is specifically designed to facilitate the implementation and experimentation of Continuum schedulers, e.g., carbon-aware schedulers. Renewable energy sources are modeled using real-world data to accurately capture fluctuations in green energy availability, leveraging carbon intensity data obtained from Electricity Maps [5]. Real-world workloads were obtained from the MIT Supercloud dataset [6], providing realistic usage patterns. By replaying the workloads in a certain order, it becomes possible to evaluate different scheduling algorithms under varying workload demands and carbon intensity scenarios. Finally, we introduce randomized cloud costs, ranging from €2 to €4 per GPU/h, across different clusters. Clusters powered by greener energy sources were assigned slightly higher costs, thus creating a deliberate trade-off between reducing carbon intensity and operational expenses. Figure 2.1 provides an overview of the simulator.

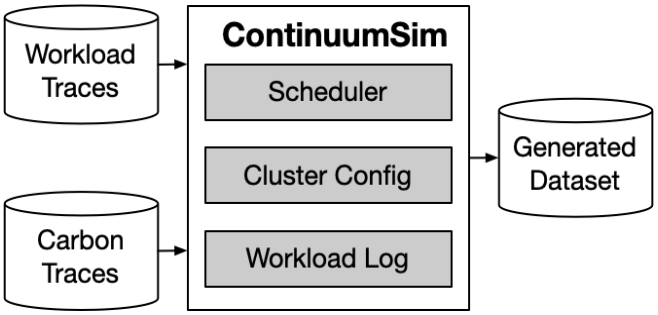


Figure 2.1: Simulator overview.

3) Real-world FaaS dataset: For the function arrival rate prediction, we adopt Globus Compute [7], a function-as-a-service (FaaS) dataset, which is a federated platform enabling users to deploy endpoints across edge devices to HPC clusters and invoke Python

functions via a cloud-hosted service. The dataset includes 2.1 million tasks from 252 users executed on 580 endpoints, and comprises 277,000 registered functions. It reveals patterns seen in other FaaS datasets that includes user workloads, distributed computing endpoints, and investigation of registered function bodies.

2.1.3 Results and Analyses

In this part, we evaluate the performance of two clustering approaches- Classical K-means and Deep K-means. The clustering results are visualized in 2D and 3D spaces to demonstrate the separation achieved by each approach.

1) Classical K-means Clustering

2D Clustering (Figure 2.2): With K = 2, the classical k-means clustering algorithm effectively separates workloads into two distinct clusters based on GPU and memory utilization. These clusters highlight processes with low vs. high resource demands, offering beneficial information for workload prioritization and resource scheduling.

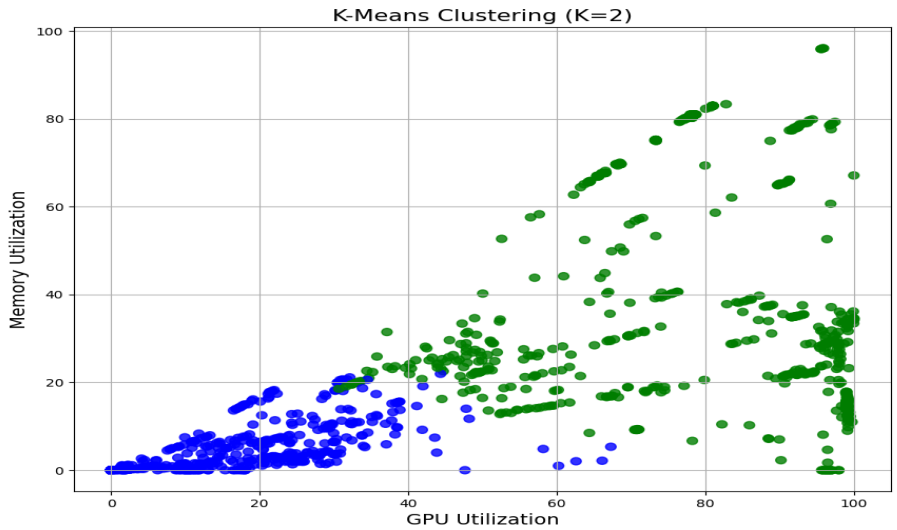


Figure 2.2: K-Means 2D clustering.

3D Clustering (Figure 2.3): When extended to K = 3 in 3D space, the clustering includes energy consumption as an additional dimension. This reveals finer distinctions, identifying clusters dominated by low, moderate, and high energy-consuming workloads, which are crucial for energy-aware workload management.

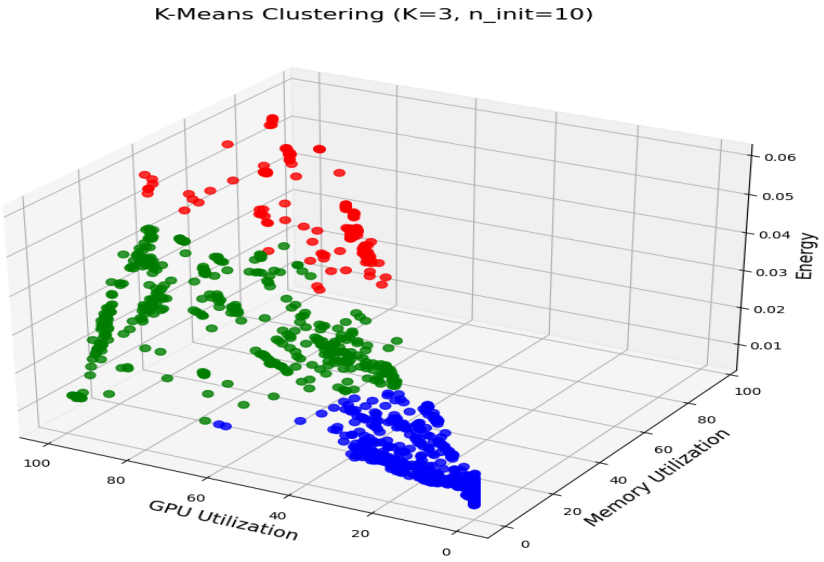


Figure 2.3: K-Means 3D clustering

2) Deep K-means Clustering:

Latent Space Learning: An autoencoder compresses input features into a latent space of dimension d = 3 while retaining the essential structure of the data. A clustering loss ensures that latent representations align closely with the learned cluster centers during training, providing improved feature separation.

2D Clustering (Figure 2.4): With K = 2, Deep K-means identifies distinct clusters in the latent space. Compared to classical K-means, the learned latent features exhibit clearer separations, demonstrating the benefits of deep feature extraction.

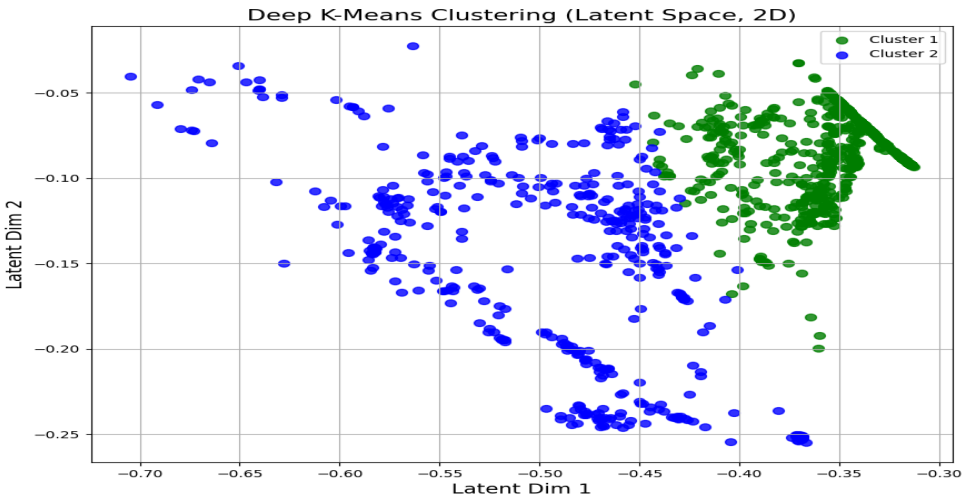


Figure 2.4: Deep K-Means 2D clustering.

3D Clustering (Figure 2.5): For K = 3, Deep K-means achieves superior separation of workloads into three clusters in the latent space. The clusters are well-defined, with reduced overlap between processes, enabling more accurate workload profiling and optimization.

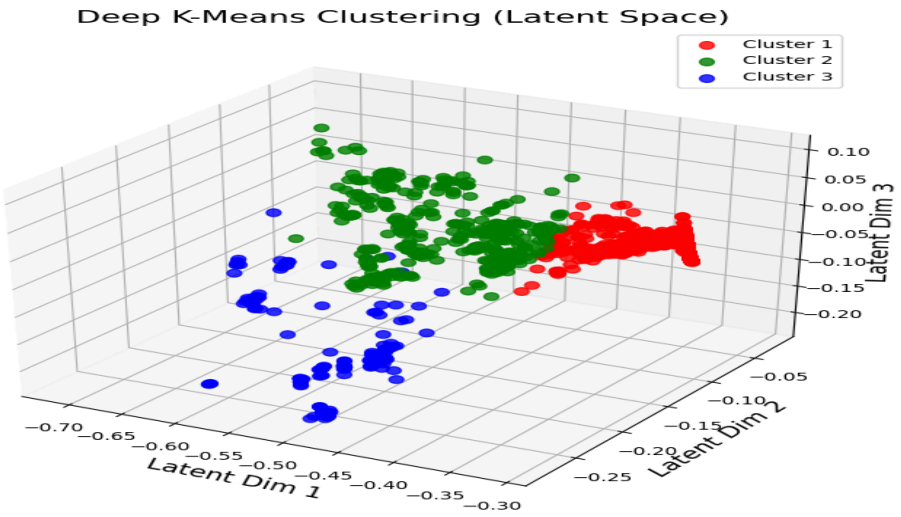


Figure 2.5: Deep K-Means 3D clustering

3) Analysis of Results: Figures 2.2, 2.3, 2.4, and 2.5 compare the results of classical K-means and Deep K-means clustering. As illustrated, Deep K-means clustering significantly enhances the separation of workloads by leveraging deep feature learning. The latent representations reduce noise and redundancy in the input features, leading to tighter and more distinguishable clusters. By grouping workloads with similar resource demands, the system can prioritize critical processes and minimize contention. Moreover, the comparison highlights the advantages of Deep K-means over classical K-means in terms of cluster separation and interpretability.

2.1.4 Retraining method for ML-based predictive behavioral forecasting

Autonomous resource management is essential for large-scale cloud data centres , where Machine Learning (ML) enables intelligent decision-making. However, shifts in data patterns within operational streams pose significant challenges to sustaining model accuracy and system efficiency. Through COGNIT, we propose an auto-adaptive ML approach to mitigate the impact of data drift in cloud systems, published in [11]. In this work, a knowledge base of distinct time series batches and corresponding ML models is constructed and clustered using the Hierarchical Density-Based Spatial Clustering of Applications with Noise (HDBSCAN) algorithm. When model performance degrades, the system uses Dynamic Time Warping (DTW) to retrieve matching hyper-parameters from the knowledge base and apply them to the deployed model, optimizing inference accuracy on new data streams.

a) Time series matching-based auto-adaptive ML model

Figure 2.6 provides an overview of the Auto-adaptive ML approach based on time series matching, featuring its two main components: the Knowledge Base and the Time Series Matching Mechanism. Essentially, the approach involves building a knowledge base containing various time series data batches, each associated with a ML model trained on that specific batch. When the deployed ML model experiences

performance degradation, the system uses the matching mechanism to find the recent time series batch that best matches one in the knowledge base, thereby invoking the most appropriate ML model.

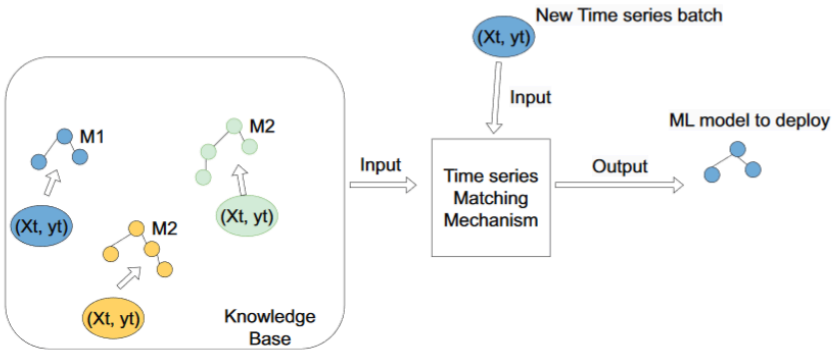


Figure 2.6: The overview of Auto-adaptive ML approach based on time series matching.

a1. Building Knowledge Base

This is an offline process that can be performed periodically to ensure it remains up to date. Figure 2.7 depicts the flow of this process. The first step involves collecting time series batches from the historical dataset, ensuring each batch has distinct characteristics and patterns. To achieve this, we apply data drift detection to segment the data at drift points, using a combination of two popular methods: Adaptive Window (ADWIN) [12] and Page-Hinkley (PH) [13].

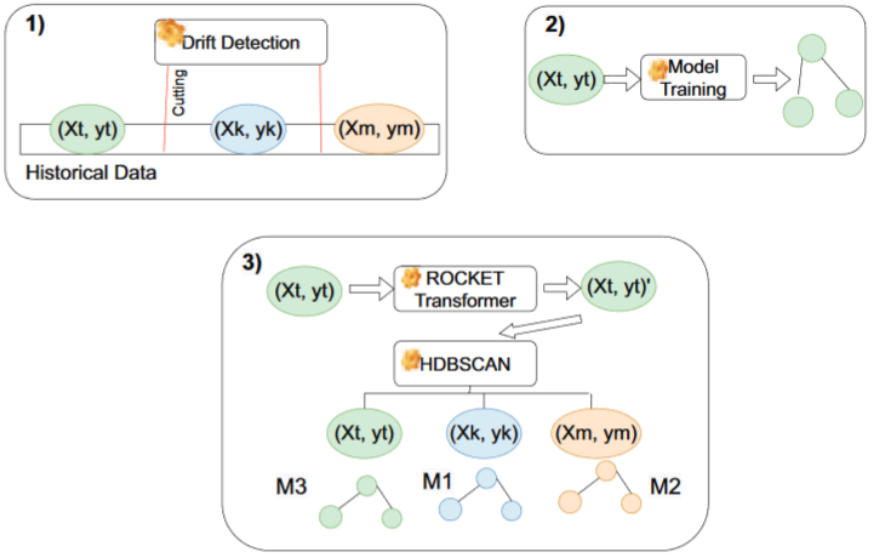


Figure 2.7: Step by step process of building the knowledge base.

After obtaining the time series batches, the next step is to train different ML models on each batch and save the corresponding hyperparameters for future use. All time series batches in the knowledge base form a search space for future time series matching. However, without efficient clustering or classification, the matching process can be highly time-consuming, as it must sequentially evaluate each batch – leading to a brute-force search with a complexity of O(N), where N is

the number of time series batches in the knowledge base. To narrow the search space, we segment the knowledge base using HDBSCAN algorithm [25]. Assuming an average cluster size of N/k, where k is number clusters, the complexity is reduced to O(k+N/k).

The advantage of HDBSCAN compared to other clustering methods (e.g., partitioning clustering methods, density-based clustering methods) is that it creates a hierarchy of clusters based on the density of data points, allowing the algorithm to identify clusters of varying shapes and densities without needing to specify the number of clusters in advance. This is particularly valuable when building (and updating later) the knowledge base, as the evolving and unpredictable nature of cloud operational data makes it difficult to determine cluster sizes in advance.

To ensure efficient HDBSCAN clustering and better identification of coherent clusters, it is crucial to extract significant features such as trends, periodicity, and noise from the time series batches. For this, we use the Random Convolutional Kernel Transform (ROCKET) [14], a highly efficient and fast method for transforming time series data, which minimizes computational overhead.

a2. Time series Matching Mechanism

To measure the similarity between time series batches, we utilize the advantages of DTW. Unlike other methods that are highly sensitive to noise and temporal distortions (e.g., Euclidean distance), DTW can handle time shifts and varying lengths, making it well-suited for evolving time series data in the knowledge base.

Giving two time series $X = (x_1, x_2, ..., x_n)$, and $Y = (y_1, y_2, ..., y_m)$, with lengths n and m respectively, the DTW algorithm calculate the DT W (X, Y) through the following procedure:

- **Initialize:**
  $\mathrm{D}(1,1) = d(x_1,y_1)$ where $d(x_j,y_j) = (x_i - y_j)^2$
- **Fill the Cost Matrix:**
  For row i =1:$\mathrm{D}(1,\mathrm{j}) = d(x_1,y_j)+\mathrm{D}(1, \mathrm{j}\text{-}1)$ with j = 2, 3, . . . , m.
  For column j =1:$\mathrm{D}(\mathrm{i},1) = d(x_i,y_1)+\mathrm{D}(\mathrm{i}\text{-}1, 1)$ with i = 2, 3, . . . , n.
- **Recurrence Relation:**
  For the remaining elements, compute $\mathrm{D}(\mathrm{i},\mathrm{j}) = d(x_i,y_j)+\min\{\mathrm{D}(\mathrm{i}\text{-}1, \mathrm{j}),\mathrm{D}(\mathrm{i}, \mathrm{j}\text{-}1), \mathrm{D}(\mathrm{i}\text{-}1, \mathrm{j}\text{-}1)\}$ with i = 2, 3, . . . , n and j = 2, 3, . . . , m.
- **Calculate DTW Distance:**
  $\mathrm{DTW}(\mathrm{X}, \mathrm{Y}) = \mathrm{D}(\mathrm{n},\mathrm{m})$.

To this end, the procedure for automatic adaptation of the deployed ML model is presented in Algorithm 1 (Figure 2.8). The deployed model's performance is continuously monitored to ensure that the average prediction accuracy exceeds a

specified threshold (e.g., *accu_thresh* = 80%). If the model's performance falls below this threshold over a predefined period (line 4), it triggers the need for model adaptation.

Algorithm 1 Auto-Adaptive ML Model with Time Series Clustering

1: **Input:** dist_thresh, accu_thresh
2: **while** True **do**
3: Evaluate $P(t)$
4: **if** $\frac{1}{M-N}\sum_{t=N}^{M} P(t) < accu\_thresh$ **then**
5: $Cluster\_rep \leftarrow$ Best-matching cluster for T_{new}
6: **if** DTW$(Cluster_rep, T_{\text{new}}) <$ dist_thresh **then**
7: best_match $\leftarrow$ Closest series in *Cluster_rep* via DTW
8: **end if**
9: **if** best_match $\neq$ None **then**
10: **break**
11: **else**
12: Update knowledge base and train with T_{new}
13: **end if**
14: **end if**
15: **end while**
16: **Output:** Model corresponding to *best_match*

Figure 2.8: Algorithm for automatic adaptation of the deployed ML model

The update process begins by matching the current inference time series batch, T_{new}, containing data from the previous update to the present, with the cluster representatives stored in the knowledge base (line 5). If a representative with a DTW distance below the threshold *dist_thresh* is found (line 6), the search proceeds within the corresponding cluster (line 7).

If a best match is identified, the algorithm retrieves the hyper-parameters corresponding to the best match batch and deploys the updated ML model for future inferences (line 9-10, line 16). If no match is found, the system continues using the current ML model while incorporating T_{new} into the knowledge base and training a new model (line 11–13).

Here, the value of *dist_thresh* is derived from the average intra-cluster distance and the average inter-cluster distance, both of which are calculated offline during the process of building the knowledge base. More specifically, for a cluster C_k with n_k members, the $intra\_dist_k$ is calculated as:

$$intra\_dist_k = \frac{1}{n_k(n_k-1)/2}\sum_{i=1}^{n_k}\sum_{j=i+1}^{n_k} DTW(X_i, X_j)$$

where X_i and X_{ij} are time series within cluster C_k; and the inter-cluster distance, $intra\_dist_{k,l}$, is calculated as:

$$intra\_dist_{k,l} = \frac{1}{n_k n_l} \sum_{i=1}^{n_k} \sum_{j=1}^{n_l} DTW(X_i, Y_j)$$

where $X_i \in C_k$ and $Y_j \in C_l$.

The distance threshold, dist_thresh, is then defined as:

$$dist\_thresh = \alpha \times \frac{1}{|C|} \sum_{C_k \in C} intra\_dist_k + \beta \times \frac{1}{|C|(|C|-1)} \sum_{C_k, C_l \in C, k \neq l} intra\_dist_{k,l}$$

where C is the set of all clusters; α, β are weights that can be adjusted based on the specific context of preference. For simplicity and to maintain neutrality between intra-cluster and inter-cluster distances, we set the weights equally in the experiment below, with $\alpha = \beta = 0.5$.

b) ML-Based Prediction

b1. Workload prediction

In this experiment, we use real operational data traces from Wikipedia [15], [16] to simulate a real cloud system and its streaming operational data. We then develop a workload prediction model using the well-known time series prediction technique Long Short-Term Memory (LSTM) [17] to forecast the incoming workload.

b1.1 Wikipedia trace.

Wikipedia trace represents 10% of HTTP requests to Wikipedia projects from 19/09/2007 to 31/12/2013, initiated by front-end proxy caches and aggregated hourly with a unique ID, timestamp, and request count. The Arabic Wikipedia trace is used for the experiment due to its various data drifts [18]. The first 20,000 data points from the trace are used for the initial training of the prediction model and serve as the historical operational data for building the knowledge base. The initially trained ML model is deployed to predict the incoming workload starting from data point 20,001, while the proposed auto-adaptive ML method is applied simultaneously to maintain the robustness of the deployed prediction model. Therefore, the remaining Wikipedia trace from data point 20,001 is used to evaluate both the accuracy of the ML model and the efficiency of the proposed auto-adaptive ML method.

b1.2 LSTM-based workload prediction.

The model consists of two LSTM layers with 128 and 32 units, respectively, processing input sequences of shape (10, 1). A dropout layer with a rate of 0.3 is added to prevent overfitting. The final dense layer with 1 unit outputs the predicted value for the next time step. At each time slot t, the model predicts the number of requests at t + 1.

b2. CPU utilization prediction

In this experiment, we build a resource utilization prediction model using real trace data from the Google cluster. The model employs SGDRegressor algorithm to predict incoming CPU utilization for a machine in the Google data centre.

b2.1 Google cluster trace

The Google cluster trace [19] includes runtime traces for 12,500 physical machines (PMs) and over 650,000 jobs, with resource utilization (CPU and memory) recorded every 10 seconds, starting from 01/05/2011. For our experiment, we use the processed data from [20], which aggregates CPU and memory utilization in 5-minute intervals per machine over a 24-hour period. The dataset, extracted from the first 10 days, filters CPU and memory utilization between 5% and 90%. We reserve the first 2 day's data points to train the initial CPU utilization prediction model and to build the knowledge base for this experiment. The remaining data (more than 2,000 data points over 8 days) is used to evaluate the performance of the ML model and the efficiency of the auto-adaptive ML method.

b2.2 SGDRegressor-based CPU utilization prediction

The SGDRegressor model [21] is a linear regression model optimized using the Stochastic Gradient Descent (SGD) algorithm. At each iteration, the algorithm selects a small, randomly chosen subset of data to calculate the gradient of the loss function, then updates the model's parameters. This makes SGD particularly efficient for large datasets and online learning, where the model can be continuously updated with new data, making it ideal for large-scale applications like CPU utilization prediction [22].

SGDRegressor supports various loss functions and penalties for fitting linear regression models. To implement the CPU utilization prediction, we employ the squared epsilon insensitive loss function [23], which is robust to small deviations and emphasizes significant errors. We configure the model to take input in the shape (10, 1), meaning it looks back 10 time steps to predict CPU utilization for the next time step, t + 1.

c) Baseline Approaches for Adapting ML Models

c1. Incremental retraining

This is a widely used approach in the literature to address the degradation of deployed ML models, based on the premise that the most recent data best reflects the characteristics of the operational data stream. It involves periodically retraining ML models as new data becomes available. To implement this approach, we retrain the deployed ML model as follows: After every 1-day interval, when a new data batch becomes available, the model is retrained using the dataset that incorporates the most recent data batch. Intuitively, if the incoming operational data follows patterns similar to the recently observed data, this retraining approach is expected to improve the performance of the ML model.

c2. Drift detection based retraining

To reduce the retraining frequency, many studies use data drift detection methods to detect the changes in the operational data streams, from which to trigger the retraining to update accordingly [27], [28]. We implement a similar baseline retraining strategy, employing the well-known data drift detection method, ADaptive WINdowing (ADWIN) [12], which has proven effective in minimizing both false positive and false negative rates, particularly in one-dimensional data [24]. A challenge with concept drift-based retraining methods is determining the appropriate data for retraining. In the Wikipedia workload prediction experiment, the retraining dataset consists of 672 data points following the drift point, as observed in [27]. Conversely, in the Google CPU utilization prediction experiment, the retraining dataset is constructed by including data from the previous retraining point up to the current drift point detected by the data drift mechanism, along with an additional recent data batch covering a 1-day interval after the drift point

d) Evaluation Metrics

d1. ML resilience level

We evaluate the efficacy of the proposed method by analyzing its impact on maintaining the high performance of the ML model throughout the experiment. This involves computing the prediction error of the ML model at each time point, indicated by the relative absolute error (RAE), calculated as RAE = $\frac{|y\_predict - y\_real|}{y\_real}$, and observing the evolution of its performance over time.

d2. Overhead cost

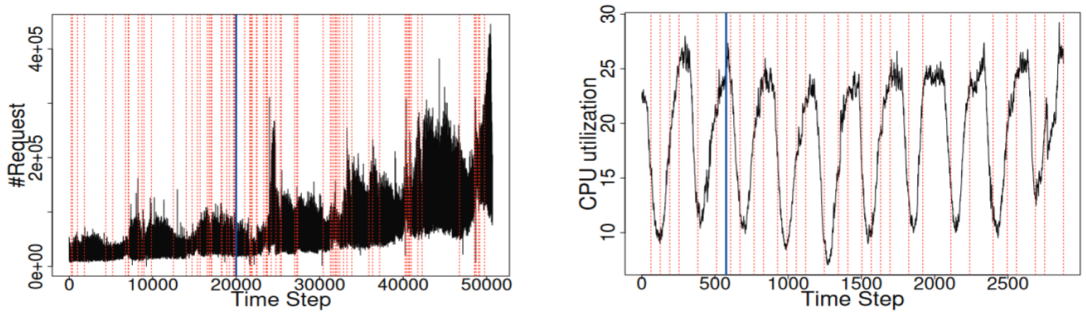
Given that the retraining process incurs significant costs (e.g., training resources and data management), we abstract these overhead costs into the frequency of retraining invocations and evaluate the efficiency of the retraining framework accordingly.

d3. Scalability

In large-scale cloud systems, which typically generate millions of operational data points per time unit, the decision time for updating or retraining the deployed ML model is critical to ensuring the timely deployment of the new model. Therefore, we measure the time complexity of the proposed approach, providing detailed insights into each step of the process.

e) Results and discussion

Figure 2.9 shows the evolution of the operational data in the two experiments: the red lines highlight the drift points detected by the ADWIN method [12], the blue line indicates the starting point at which the ML model adaptation approaches are applied. In the Wikipedia dataset, a total of 48 drifts occur after the blue line. In the Google trace dataset, 25 drifts occur after the blue line.



a) Wikipedia b) Google trace

Figure 2.9: The two datasets in the two experiments: a) Wikipedia, and b) Google trace.

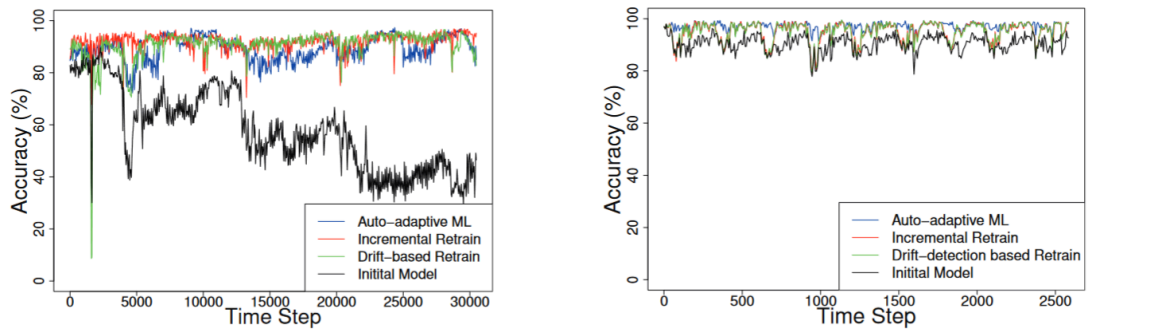
The red dashed lines indicate positions where drift is detected by the ADWINmethod; the blue line indicates positions where the retraining method begins to operate. Further, Table 2.1 presents the two knowledge bases from the experiments, detailing the total number of time series batches, the number of cluster labels created by the HDBSCAN algorithm, and the initially estimated *dist_thresh*. The knowledge bases are dynamically updated throughout the experiments, with adjustments to cluster labels and recalculations of *dist_thresh* as new models are introduced.

Experiment	#TS Batches	#Clusters	dist_thresh
Workload Prediction	42	5	93
CPU Utilization Prediction	6	3	14

Table 2.1: Initial configuration of knowledge bases used in the experiments. Values dynamically evolve as new models are introduced during the experiment.

e1. Reliability of ML Models

Figure 2.10 (a) and (b) show the performance of the ML models during the two experiments with different adaptation approaches. The performance of the ML models without updates (represented by the black curve) is also plotted in this figure. Without adaptation, the initial ML model clearly becomes obsolete, resulting in a decline in accuracy as the experiments progress.



a) Workload prediction (Wikipedia) b) CPU utilization prediction (Google trace)

Figure 2.10: The ML model performance in the two experiments.

Table 2.2 summarizes the average accuracy of the ML models and the total retraining events across the two experiments when using different adaptation approaches. In both experiments, the proposed Auto-adaptive ML approach demonstrates a competitive level of reliability for the two ML models, while incurring significantly lower overhead costs compared to the other approaches. In the Wikipedia experiment, the average accuracy achieved by the workload prediction using the Auto-adaptive ML approach is 89.4%, compared to 90.2% with the Drift detection-based retraining, and 91.9% with the Incremental retraining. Notably, while the difference in reliability between the Auto-adaptive ML and the two retraining approaches is small, the Auto-adaptive ML achieves this with significantly fewer retrainings. The Drift detection-based retraining method initiates retraining based on detected drifts, resulting in 48 retraining events. In contrast, the Incremental retraining method triggers 1,250 retrainings, as it updates the mode at one-day intervals (for the entire experiment with 30,000 data points). By the end of the experiment with the Auto-adaptive ML approach, only 37 additional models were added to the knowledge base, meaning that the Auto-adaptive ML invoked just 37 retraining processes – only when it could not find an appropriate model in the existing knowledge base. Consequently, the Auto-adaptive ML approach reduces the overhead cost for retraining by approximately 22.9% compared to the Drift detection-based retraining approach, and by approximately 97% compared to the Incremental retraining approach.

Method	**Wikipedia**		**Google Trace**	
	Accuracy (%)	**#Retrainings**	**Accuracy (%)**	**#Retrainings**
Incremental	91.9 (3.9)	1250	95.1 (2.2)	9
Drift Detection	90.2 (7.4)	48	95.0 (2.1)	27
Auto-adaptive ML	89.4 (5.2)	37	97.1 (1.8)	1

Table 2.2: Average accuracy and overhead cost (retrainings) for Wikipedia and Google Trace predictions. Numbers in parentheses represent standard deviation.

In the Google trace experiment, we observe that the initial model performs quite well throughout the experiment, even without updates, achieving an average accuracy of 80.1%. This is understandable, as the dataset only shows a recurring drift with a repeating pattern (Figure 2.9(b)), which the ML model effectively learned during the initial training. Specifically, the Auto-adaptive ML approach achieves a slightly better reliability level for the CPU utilization prediction model compared to other retraining approaches. The average accuracy of the ML model achieved with the Auto-adaptive ML approach is 97.1% compared to 95.1% with Incremental retraining and approximately 95% with Drift detection-based retraining. The experimental results from this relatively stable data stream further demonstrate that frequent retraining does not necessarily lead to the highest reliability for the ML model. The Drift detection-based retraining approach invoked a total of 27 retraining processes, while the Incremental retraining approach invoked 9 retraining processes. In contrast, the Auto-adaptive ML approach required only 1 retraining event. Consequently, in this experiment, the Auto-adaptive ML reduces the overhead cost for retraining by approximately 96.3% compared to the Drift-based retraining approach, and by approximately 88.9% compared to the Incremental retraining approach.

e1. The Auto-adaptive ML Approach's Scalability

The two main processes in the Auto-Adaptive ML approach that require time are the time series matching process (implemented DTW) and the clustering process (implemented HDBSCAN). The time series matching is an online process and thus has a more significant impact on the decision time of the proposed approach. In contrast, the clustering process is performed offline to update the knowledge base. To comprehensively evaluate the scalability of the proposed method, we will discuss the time complexity of both processes in detail.

e1.1 Building Knowledge base- HDBSCAN clustering

Generally, with N number of data points, HDBSCAN has an average-case complexity of O(N log N)[25]. For large datasets with thousands of data points, the practical performance of HDBSCAN can be influenced by factors such as data point density. To evaluate the time complexity of HDBSCAN with large datasets, we conduct a benchmark by performing HDBSCAN clustering on a sequence of datasets ranging from 1,000 to 20,000 data points, with each setting of the number of data points being tested 10 times. Figure 2.11 shows the average time (error bars show the standard deviation) taken for clustering with HDBSCAN across different sizes of input data points. Although the algorithm's time complexity is not linear with respect to the size of the input, the increase in time is not significantly large. For input sizes up to 20,000, the algorithm completes clustering in under 5 seconds. It is worth noting that, for a specific domain, the input size of the knowledge base is often not large and can be truncated by removing obsolete time series batches if they are infrequently matched with the inference batches.

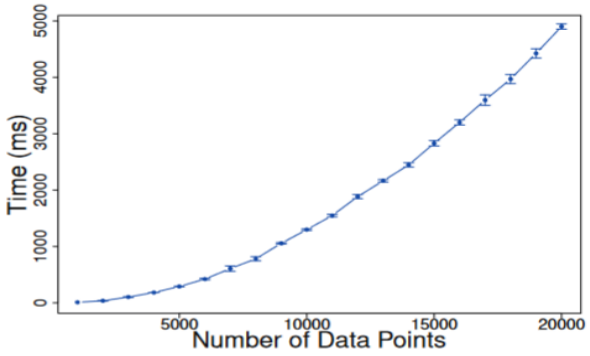


Figure 2.11: Time taken for clustering with HDBSCAN algorithm across different sizes of input.

e1.2 Time series matching – Dynamic Time Warping

Generally, the time complexity of the DTW algorithm is O(N×M) where N and M are the lengths of the two time series being compared [26]. This quadratic complexity makes DTW computationally intensive when dealing with long time series. To evaluate the time complexity of time series batch matching, we conducted a benchmark using DTW with different settings for the length of the time series input, ranging from 1,000 to 10,000 data points. Each setting was tested 10 times. Figure 2.12 presents the benchmarking results. It is clear that the time taken for DTW increases with the length of the time series, rising from an average of 24 ms for a length of 1,000 to approximately 2,9 seconds for a length of 10,000. However, the matching time with this algorithm is acceptable for real-time systems where the data streaming granularity is on the order of minutes, such as the streaming operational data of cloud data centres as observed in the datasets used in the experiment.

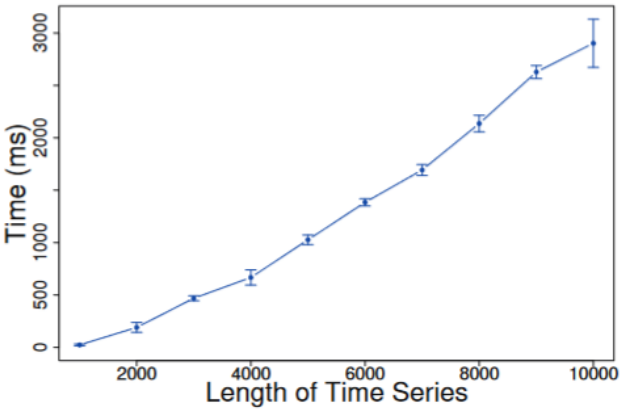


Figure 2.12: Time taken for DTW time series matching with different length of time series.

2.2 [SR5.2] Smart Management of Cloud-Edge Resources

Management of Cloud-Edge resources is crucial in the AI-enabled orchestrator across the Computing Continuum. This is achieved by leveraging the predictive capabilities of AI/ML models, which analyze workload patterns in relation to dynamic resource consumption, including CPU, memory, and network bandwidth, as well as energy consumption. The systems are modeled using two complementary approaches. The first one involves the development of advanced algorithms for resource optimization, which are then integrated with the COGNIT testbed for real-world validation and deployment (subsection 2.2.1). The second approach focuses on understanding the scalability and adaptability of these systems across the continuum by employing modeling and simulation techniques. These simulations help to evaluate system performance under varying workloads and deployment scenarios, with the findings detailed in subsection 2.2.2.

2.2.1 Cloud-Edge resource optimization algorithms

Optimizing cloud-edge resources is essential to achieve resource management across the continuum, ensuring seamless operation amid dynamic workloads and evolving infrastructure policies. In this development cycle, we focus on developing the ILP model and multi-objective evolutionary algorithms that incorporate workload characteristics, interference effects, green energy availability, and resource usage cost to achieve environmentally sustainable resource placement within the Continuum.

a) Metrics

Metrics for application and infrastructure are collected by Cloud-Edge manager, which are utilised to develop optimization algorithms and taking into consideration some optimization objectives such as resource interference cost, carbon emission, and monetary cost.

b) Algorithms

b1.1 Optimal number of SRTs:

To determine how many SRTs are needed, the algorithm considers the current and incoming workload. It applies a reactive SRT estimation approach by analyzing real-time load metrics to assess the number of SRTs required to manage the current demand effectively. The procedure for determining the optimal number of SRTs is presented in the algorithm (Figure 2.13).

Algorithm #1: Optimal SRTs

Input: Incoming requests

Output: Optimal SRTs

Step 1: Determine SRTs needed for new incoming requests
Calculate SRTs needed for incoming request rate

Step 2: Account for backlog
Calculate SRTs needed for waiting functions

Step 3: Combine immediate needs
Total immediate SRT needs = incoming requests + waiting functions

Step 4: Consider current resources
Count idle SRTs and busy SRTs that will free up soon

Step 5: Make final calculation
IF we need more SRTs than will be available soon (ideal + soon to be freed):
Return total needed + busy SRTs that won't free up soon
ELSE:
Return total needed + all currently busy SRTs

Figure 2.13: Algorithm for determining the optimal number of SRTs

b1.2 Initial placement mapping of SRTs to hosts:
The initial placement mapping of SRTs to hosts begins by profiling each SRT to understand its resource requirements and calculate a priority score. Then the SRTs are sorted by priority, focusing on the most resource-intensive first to optimize the overall allocation. Next phase is to proceed to map each SRT in order, checking placement constraints and selecting a host that meets resource availability, adheres to safety thresholds, and minimizes resource waste. Finally, the SRTs are allocated to the best-fit host or mark it as unallocated if no suitable option is found. The algorithm (Figure 2.14) for determining the initial placement of SRTs to hosts is discussed as follows:

Algorithm #2: SRTs placement

Input: SRTs and available hosts

Output: Allocation solution

```
// Profile and prioritize
  FOR each SRT:
    Determine resource characteristics
    Calculate priority score

  Sort SRTs by priority (largest/most intensive first)

  // Placement phase
  FOR each SRT in prioritized order:
    Verify placement constraints

    Find best matching host that:
      - Has sufficient available resources
      - Maintains safety thresholds
      - Minimizes resource waste

    IF suitable host found:
      Allocate SRT to host
      Update resource tracking
    ELSE:
      Mark SRT as unallocated

  Return allocation solution
```

Figure 2.14: Initial placement mapping of SRTs to hosts

b1.3 ILP For Cloud-Edge Resource Optimization

The ILP formulation primarily focuses on the integration of energy consumption, migrations, and resource interference considerations into the integer linear programming model. The input variables, decision variables, constraints, and objective functions are as follows:

Input variables:

- Hosts $H = \{h_1, h_2, ..., h_m\}$, each of which has the following input variables:
  - H_j^m: the amount of memory available on the host h_j, $\forall j = 1, 2, ..., m$,
  - H_j^c: the number of CPU cores available on the host h_j, $\forall j = 1, 2, ..., m$.

- c_j : base energy consumption of host h_j
- b_j : energy per unit CPU usage for host h_j
- $E_{j,max}$: is the maximum possible energy consumption of the host h_j when 100% CPU is utilized.

- The set of all virtual machines (VMs) to be mapped to the physical infrastructure is $V = \{v_1, v_2, ..., v_n\}$, each of which has the following input variables:
  - V_i^m: the amount of memory requested for VM v_i, $\forall i = 1, 2, ..., n$,
  - V_i^c: the number of CPU cores requested VM v_i, $\forall i = 1, 2, ..., n$,
  - V_i^u: the CPU usage VM v_i, $\forall i = 1, 2, ..., n$,
  - $\hat{X}_{ij}$: the current allocation of VMs to hosts, which is equal to 1 if the VM v_i is currently allocated to the host h_j and 0 otherwise.

Decision variables:

- X_{ij}: the allocation of VMs to hosts for the planning period, which is equal to 1 if the VM v_i *will be* allocated to the host h_j during the planning period and 0 otherwise, $\forall i = 1, 2, ..., n, \forall j = 1, 2, ..., m$.
- Y_j: 1 if the host h_j will be used during the planning period and 0 otherwise, $\forall j = 1, 2, ..., m$.

Constraints:

- The memory demanded by all VMs allocated to a host cannot exceed the available memory of that host:

$$\sum_{i=1}^{n} X_{ij} V_i^m \leq Y_j H_j^m, \forall j = 1, 2, ..., m$$

- The CPU demanded by all VMs allocated to a host cannot exceed the available CPU of that host:

$$\sum_{i=1}^{n} X_{ij} V_i^c \leq Y_j H_j^c, \forall j = 1, 2, ..., m$$

- A VM cannot be allocated to a host that is off (implicit):

$$X_{ij} \leq Y_j, \forall i = 1, 2, ..., n, \forall j = 1, 2, ..., m$$

- Each VM must be assigned to exactly one host:

$$\sum_{j=1}^{m} X_{ij} = 1, \forall i = 1, 2, ..., n$$

Objective Function 1: Minimizing Energy Consumption:

The simplest way to express the energy consumption (EC) is:

$$EC = \sum_{j=1}^{m}\left(\sum_{i=1}^{n} E_{ij} + c_j Y_j\right)$$

where:

- E_{ij} is the contribution of the VM v_i to the total EC of the host h_j, $\forall i = 1, 2, ..., n$, $\forall j = 1, 2, ..., m$,
- c_j is the optional term that expresses EC of the host h_j non-related to VMs, $\forall j = 1, 2, ..., m$,
- $\sum_{i=1}^{n} E_{ij} + c_j Y_j$ is the total EC of the host h_j, $\forall j = 1, 2, ..., m$.

The simplest way to express EC of the VM v_i on the host h_j, E_{ij} is:

$$E_{ij} = b_j V_i^u X_{ij}$$

where b_j is the coefficient that represents the average energy consumption on the host h_j per the unit CPU usage, if this host-VM combination (HVMC) is used ($X_{ij} = 1$).

EC defined this way can be used as:

- An objective function, e.g. minimize $EC = \sum_{j=1}^{m}\left(\sum_{i=1}^{n} E_{ij} + c_j Y_j\right)$,
- A constraint, e.g. $EC \leq E_{max}$, where E_{max} is a predefined value.

Objective Function 2: Minimizing Migrations:

The simplest way to define the number of migrations as:

$$\sum_{i=1}^{n}\sum_{j=1, j\neq k}^{m} X_{ij}$$

Where the VM v_i is currently on the host h_k, i.e., $\hat{X}_{ik} = 1$.

Migrations defined this way can be used as:

- An objective function, e.g. minimize $MIG = \sum_{i=1}^{n}\sum_{j=1, j\neq k}^{m} X_{ij}$,
- A constraint, e.g. $MIG \leq M_{mig}$, where M_{mig} is a predefined value representing the allowed number of VM migrations.

Objective Function 3: Minimizing Interferences:

The simplest way to express the Resource Utilization Ratio (RUR) of host h_j is:

$$\frac{Energy\ consumption\ of\ the\ host\ h_j\ if\ VM\ v_i\ is\ mapped\ to\ the\ the\ host\ h_j}{Maximum\ \ possible\ energy\ consumption\ of\ the\ host\ h_j}$$

$$RUR_j = \frac{\sum_{i=1}^{n} E_{ij}}{E_{j,max}}$$

Where:

- E_{ij} is the energy contribution of VM v_i on the host h_j
- $\sum_{i=1}^{n} E_{ij}$ is the expected energy consumption of the host h_j if VM v_i is mapped to the host h_j in addition to other VMs.
- $E_{j,max}$ is the maximum possible Energy Consumption of the host h_j when 100% CPU is utilized. The time period is the same for all hosts H.
- Objective function is to maximize Average Resource Utilization Ratio (ARUR),e.g., $ARUR = (\sum_{j=1}^{m} RUR_j)/m$
- Another objective function can be somehow trying to pick the least ratio RUR_j to place VM.

b1.4 Multi-Objective Cloud-Edge Resource Optimization: Formulation

We consider a discretized workload model dividing time T into consecutive equal time slots, represented as T = {t|t ∈ [0, T], (t+1) - t = Δt seconds}. The system operates across a set of geographically distributed regions R = $\{r^j|j\in[1,n]\}$ and set of n heterogeneous computing clusters represented as H = $\{h_i^j|i\in[1,n], j\in[1,m]\}$ located across m different regions. Each computing cluster h_i^j owns d types of resources such as CPU, GPU, memory, disk storage, and network bandwidth, etc. for which $R_{i,j}^d$ depicts the availability of resource d in h_i^j. The current utilization of resource d at computing cluster i located at j^{th} region is defined as $CU_{i,j}^d$. Let $\epsilon_t^{i,j}$ denote the CO2 emissions (e.g., kg CO2 per unit) for cluster *i* located at j^{th} region. The monetary cost of using cluster *i* at *t* time slot is $p_t^{i,j}$. The availability of a computing cluster is expressed as Z = $\{z_t^{i,j} | i\in[1, n], j\in[1,m], t\in T, z_t^{i,j}\in\{0, 1\}\}$, which means $z_t^{i,j}$ = 0, whenever the computing cluster is unavailable during *t*. Next, we define the incoming function invocations as F = $\{f_t^{i,j,k}|i\in[1,n], j\in[1,m], k\in[1,l], t \in T\}$, Each invocation request contains four attributes, i.e., $f^{i,j,k}$ = <$f_{ID}^{i,j,k}$, $f_{at}^{i,j,k}$, $f_{ft}^{i,j,k}$, $f_d^{i,j,k}$>, where $f_{ID}^{i,j,k}$

denotes the function invocation ID, $f_{at}^{i,j,k}$ denotes arrival time, $f_{ft}^{i,j,k}$ represents the finish time, and $f_{d}^{i,j,k}$ denotes the resource requirement.

The multi-objective cloud-edge resource optimization is employed to balance three key objectives: minimising resource interference, carbon emission, and monetary cost across Cloud-Edge clusters. Mathematically, we formulate the optimization objectives as follows:

1. Minimize interference effect (f_1): The objective of interference-aware scheduling is to minimize contention of hardware resources on the computing clusters. In principle, picking the cluster where adding the Serverless function's resource usage would minimise the distance D_{hi}^{j} between the clusters' resource utilisation. We model D_{hi}^{j} as $(CU_{i,j}^{d} + f_{d}^{i,j,k} - \overline{CU}_{i,j}^{d})$ where, $\overline{CU}_{i,j}^{d}$ denotes the average utilisation of resource *d* across all computing clusters. The objective is to minimize the interference effect.

$$min \sum_{t} \sum_{i=1}^{n} \sum_{j=1}^{m} (max\ (D_{hi}^{j}))$$

2. Minimize emission of carbon footprints (f_2): The carbon footprint of Continuum clusters is computed by measuring the carbon intensity '$\epsilon_{t}^{i,j}$' of the power grids powering each node; this is expressed in grams of CO2 equivalent per kilowatt-hour (gCO2eq/kWh), representing the average weighted carbon emissions of the mix of sources utilized to generate energy at any given time. The objective is to minimize carbon footprint emissions.

$$min \sum_{t} \sum_{i=1}^{n} \sum_{j=1}^{m} (\epsilon_{t}^{i,j})$$

3. Minimize the total monetary cost (f_3): This objective function deals with the minimization of the overall monetary cost '$p_{t}^{i,j}$' of using cluster *i* resources at *t* time slot; this is expressed using randomized cloud costs across different clusters. Clusters powered by greener energy sources were assigned slightly higher costs, thus creating a deliberate trade-off between reducing carbon intensity and operational expenses. The objective is to minimize the total monetary cost.

$$min \sum_{t} \sum_{i=1}^{n} \sum_{j=1}^{m} (p_{t}^{i,j})$$

4. Resource constraints: The optimal scheduling of functions on the available computing clusters shall ensure that the placement process respects the availability of resources. The assignments of functions is defined as
   $A = \{\sigma_{t}^{i,j,k} \mid i \in [1,n], j \in [1,m], k \in [1,l], t \in T, \sigma \geq 0\}$.
   The availability constraint is formulated as follows:

$$\sum_{k=1}^{l} (\sigma_{t}^{i,j,k} \times f_{d}^{i,j,k}) \leq (z_{t}^{i,j} \times R_{i,j}^{d}),\ t \in T, \forall\ h_{i}^{j} | i \in H$$

b1.5 Multi-Objective Cloud-Edge Resource Optimization: Algorithms

We compare the performance of three multi-objective optimization algorithms- Non-dominated Sorting Genetic Algorithm II (NSGA-II), Multi-Objective Genetic Algorithm (MOGA), and Strength Pareto Evolutionary Algorithm 2 (SPEA2) to balance three key objectives: minimising resource interference, carbon emission, and monetary cost across Cloud-Edge clusters. The working of these algorithms are as follows:

1) Multi-Objective Optimization with NSGA-II:

The NSGA-II [1, 2] is a multi-objective evolutionary algorithm used to address conflicting objectives in complex optimization problems. As discussed in the deliverable D4.3, NSGA-II operates by iteratively refining a set of candidate solutions through a process of selection, crossover, and mutation to achieve Pareto-optimal solutions.

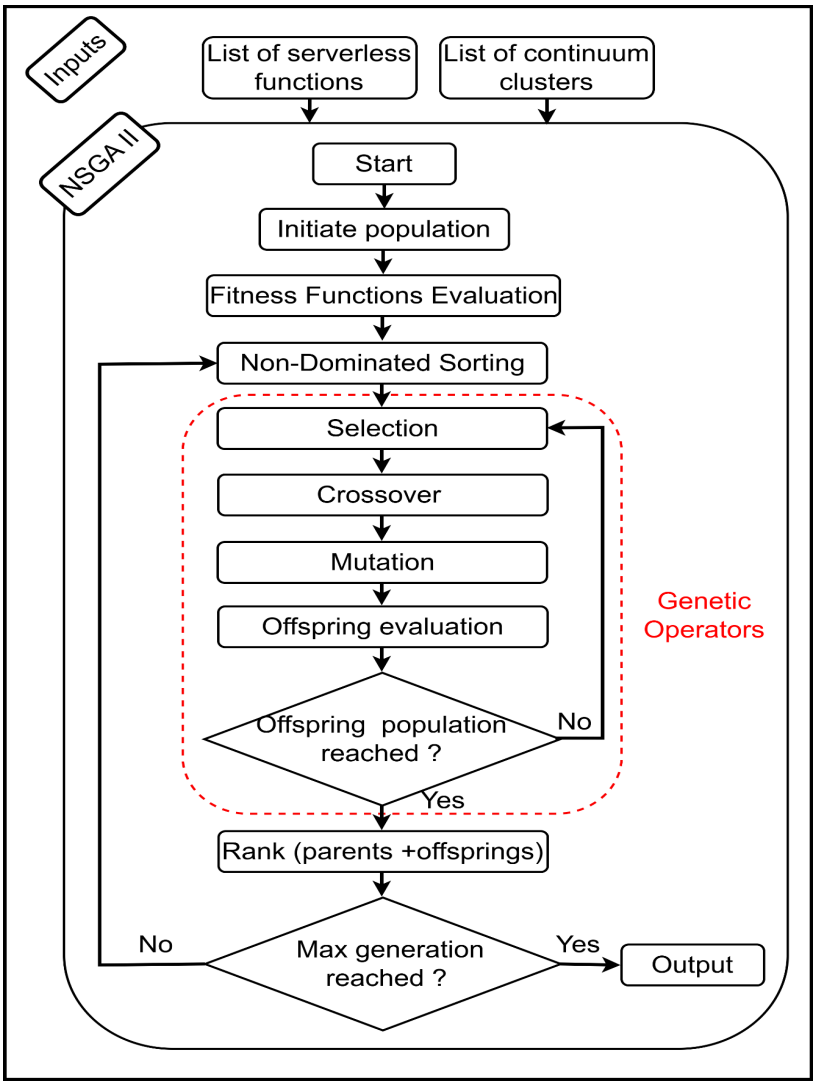


Figure 2.15: NSGA-II algorithm flowchart

The workflow of the NSGA-II algorithm [1, 2, 8] is presented in Figure 2.15. The algorithm takes the list of serverless functions and list of continuum clusters for placement. In the

encoding mechanism, each solution is encoded as a vector of integers. The length of the vector is considered as a chromosome of size *l*; which is a number of functions for placement. The content of each cell of the vector is a gene value in the chromosome; which can take a number between 1 and n that denotes the cluster assigned to that function.

The main phases of this algorithm are as follows: (i) Initiate population: The algorithm begins by initializing a random parent population; where a random number between 1 and n is assigned to each cell of the vector with the use of given capacity constraints; (ii) Nondominated sorting: After evaluating the individuals at each iteration, they are ranked based on both their fitness and crowding distance. Fitness reflects an individual's ability to survive and reproduce in the next generation, whereas crowding distance quantifies how close an individual is to its neighbors in the objective space. The crowding distance is computed as the summation of normalized distances between neighboring individuals across all objectives; (iii) Genetic operators: Using the ranking of individuals, the top-ranked candidates are selected as parents and genetic operators (i.e., selection, crossover, mutation) are applied to generate offspring. Subsequently, we evaluate the objective functions for each individual in the offspring population; (iv) Recombination and Selection: This phase combines the parents and offspring, and performs nondominated sorting to assess the fitness for each individual. Finally, the top-ranked individuals are picked to serve as parents for the next generation. The rank i.e., fitness of each individual is set in such a way that those in the first pareto front are entirely nondominant, while individuals in the second pareto front are dominated only by those in the first front. Priority is given to individuals from the first pareto front. If the desired population size has not been reached, individuals from the second pareto front will be considered, the process will continue until the required number of individuals is met to proceed with the next iteration.

2) Multi-Objective Optimization with MOGA:
The MOGA contains problem-specific operators (like evaluation, mutation, and crossover), and parameters such as population size, number of generations, and probabilities for crossover (cxpb) and mutation (mutpb). The algorithm begins by initializing and evaluating a population of individuals. In each generation, it calculates a scalar fitness for each individual as the weighted sum of multiple objective values (defaulting to equal weights). This scalar fitness is used for sorting the population to guide the selection process. The genetic variation process is applied using crossover and mutation, and the resulting offspring are evaluated. The next generation is selected by combining the current population and offspring, then selecting the best individuals. After all generations are completed, the Pareto front is extracted from the final population using nondominated sorting and returned.

3) Multi-Objective Optimization with SPEA2:
The SPEA2 [9] is a well known multi-objective evolutionary algorithm proposed by Zitzler et al. The evolutionary algorithm is based on the concept of Pareto domination for fitness evaluation and selection, and incorporates a niche strategy and an external archiving

mechanism for elite retention. The detailed workflow of SPEA2 is presented in Figure 2.16.

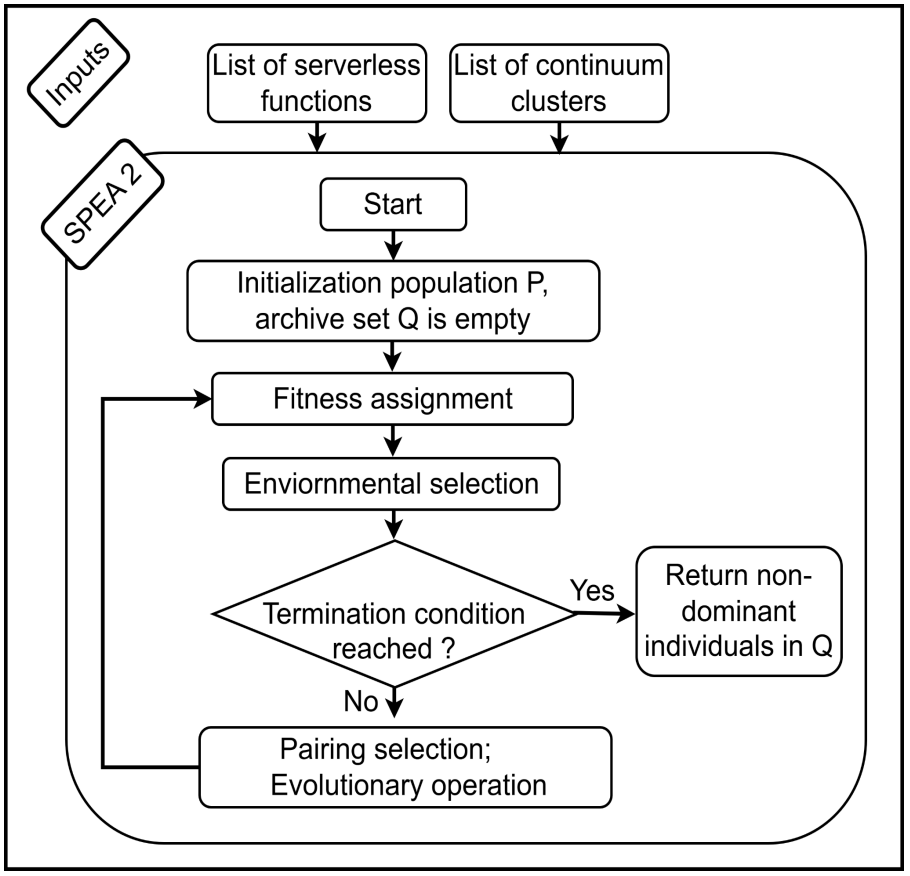


Figure 2.16: SPEA2 algorithm flowchart

The SPEA2 offers several advantages over other multi-objective evolutionary algorithms, its genetic evolution process exhibits significant randomness. This randomness helps expand the search space, preventing the algorithm from getting into local optima. However, it also leads to insufficient local search capabilities. As the algorithm approaches regions near the Pareto optimal solution, its optimization efficiency tends to drop significantly, and in some cases, the algorithm fails to locate the true Pareto optimal solution [10].

b1.6 Results and Analyses

In this part, we analyze the performance of three multi-objective optimization approaches across multiple scenarios and datasets. The evaluation focuses on three primary objectives: interference minimization, carbon emissions reduction, and cost-effectiveness. The analysis leverages results derived from real-world and synthetic datasets, as presented in Tables 2.4 and 2.5, along with Figures 2.17 to 2.24 over different control parameter values shown in Table 2.3. We discuss the trade-offs and insights observed for NSGA-II, MOGA, and SPEA2.

Control Parameter	Value(s)
Population Size	500
Stopping Criterion (Generations)	10
Crossover Probability	0.8, 0.9
Mutation Probability	0.05, 0.1

Table 2.3: Control parameters for optimization

Multi-Objective Optimization with varying parameters on synthetic dataset:

The performance metrics on the synthetic dataset are summarized in Table 2.4. The Pareto fronts for varying parameters are shown in Figures 2.17 to 2.20. Compared to the real-world dataset, the synthetic dataset introduces more significant variations in interference and emissions. Using control parameters such as a population size of 500, 10 generations, a crossover probability (cxpb) of 0.8, and a mutation probability (mutpb) of 0.05, MOGA demonstrated superior performance in terms of interference score, such as 800.24 and execution time of 16.10 seconds, whereas NSGA-II achieves better results in carbon emissions and cost. For instance, NSGA-II achieves carbon emissions of 5.88, cost of 5.49, and an execution time of 21.89 seconds. SPEA2 reports a higher interference score (800.24), increased carbon emissions (7.43), and greater monetary costs (7.20).

Algorithm	cxpb	mutpb	Interference Score	Carbon Emissions (gCO2eq/kWh)	Cost (€)	Execution Time (s)
NSGA-II	0.8	0.05	971.24	**5.88**	**5.49**	21.89
MOGA	0.8	0.05	**800.24**	6.35	5.99	**16.10**
SPEA2	0.8	0.05	1259.92	7.43	7.20	105.92
NSGA-II	0.8	0.1	989.97	**5.94**	**5.56**	22.18
MOGA	0.8	0.1	**829.24**	6.36	6.30	**17.36**
SPEA2	0.8	0.1	1292.5	7.36	7.19	104.68

Table 2.4: Comparison of NSGA-II, MOGA, and SPEA2 on synthetic dataset for population size: 500 and generations: 10

However, increasing the mutation probability (mutpb) from 0.05 to 0.1 did not significantly affect the overall performance while SPEA2 struggles to compete due to its higher execution times. Overall, SPEA2 demonstrates the poorest performance, with an interference score of 1292.5, carbon emissions of 7.36, a cost of 7.19, and an execution

time of 104.68 seconds. In terms of interference score and execution time, MOGA outperforms both SPEA2 and NSGA-II, achieving an interference score of 829.24 and an execution time of 17.36 seconds. On the other hand, NSGA-II delivers better performance than MOGA and SPEA2 in terms of carbon emissions and cost, with carbon emissions of 5.94 and a cost of 5.56, albeit with a slightly higher execution time of 22.18 seconds.

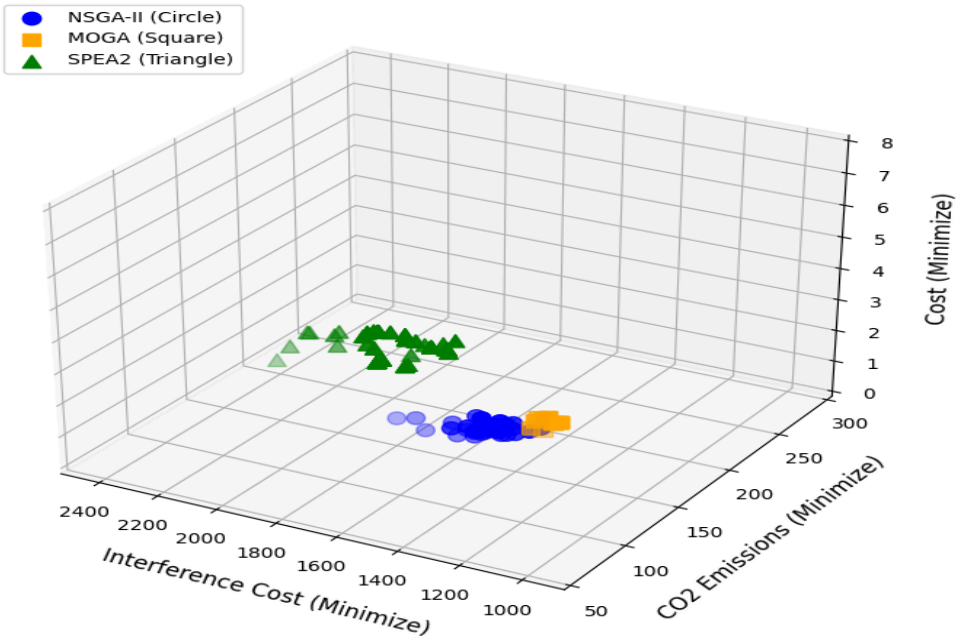


Figure 2.17: Population size = 500, Gen=10, cxpb=0.8, mutpb=0.05.

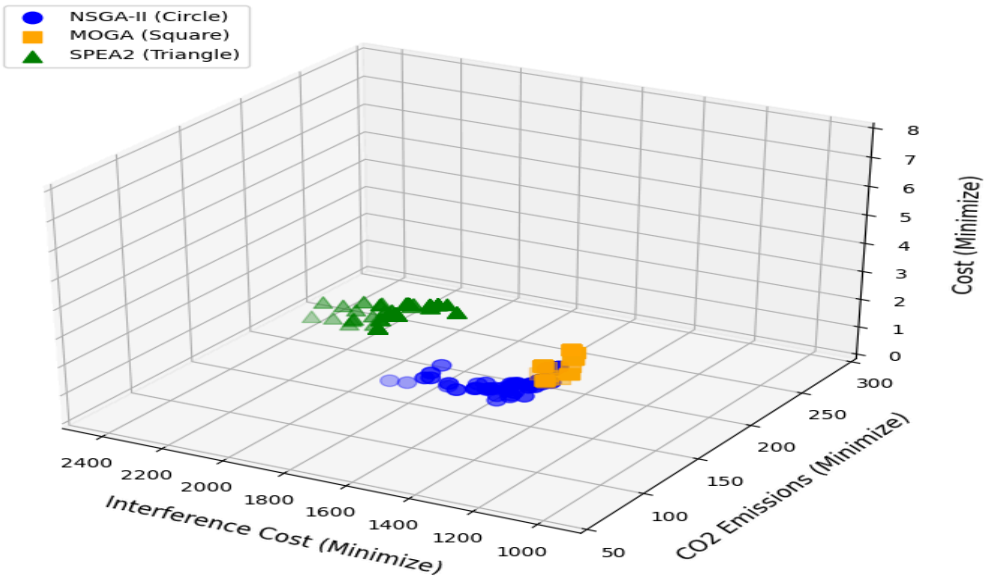


Figure 2.18: Population size = 500, Gen=10, cxpb=0.8, mutpb=0.1.

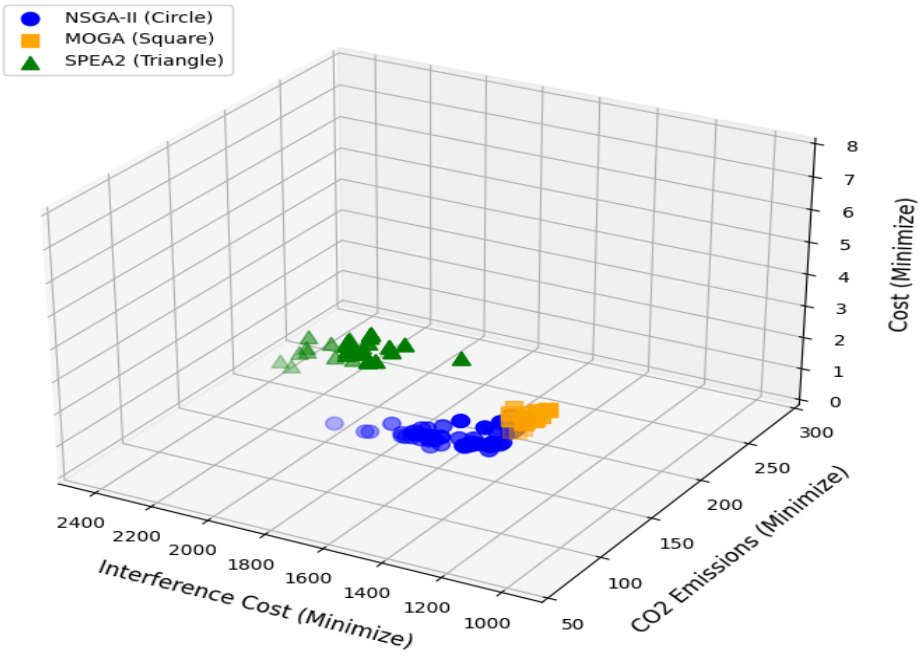


Figure 2.19: Population size = 500, Gen=10, cxpb=0.9, mutpb=0.05.

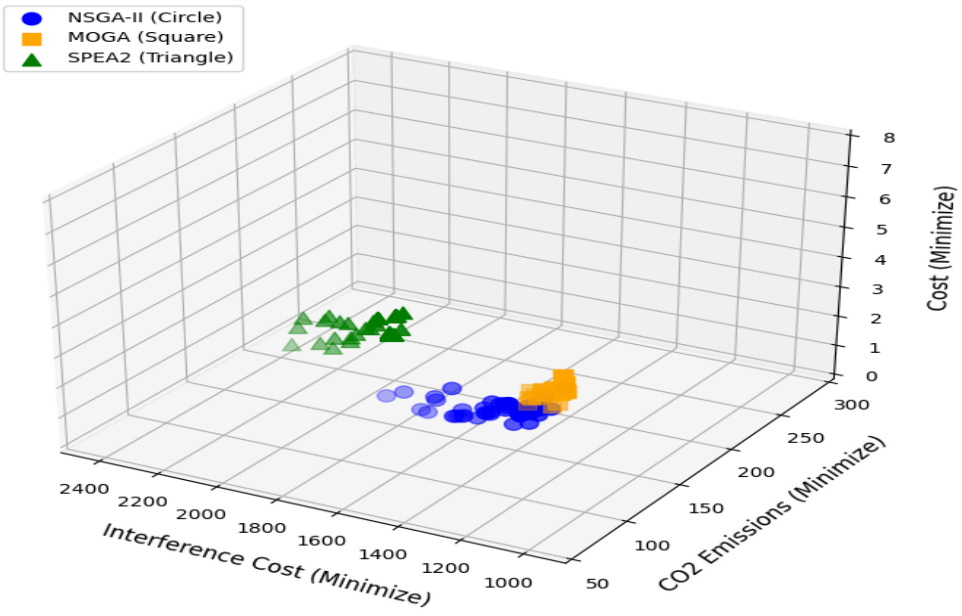


Figure 2.20: Population size = 500, Gen=10, cxpb=0.9, mutpb=0.1.

Multi-Objective Optimization with varying parameters on simulated dataset

The performance metrics on the simulated dataset are reported in Table 2.5. The pareto fronts for varying parameters are shown in Figures 2.21 to 2.24. Using control parameters such as a population size of 500, 10 generations, a crossover probability (cxpb) of 0.8, and a mutation probability (mutpb) of 0.05, NSGA-II achieve strong trade-offs across interference, cost, carbon emissions, and execution time. NSGA-II consistently achieves the lowest interference across multiple configurations, for example, reporting a score of 19.32 with cxpb=0.8 and mutpb=0.05. NSGA-II also outperforms in terms of carbon emissions, such as 17.64, resource usage cost 6.05, and execution time of 21.14 seconds. In contrast, SPEA2 reports significantly higher values across all metrics, with an interference score of 68.25, carbon emissions of 208.44, monetary cost of 11.97, and an execution time of 106.08 seconds.

Increasing the mutation probability (mutpb) from 0.05 to 0.1 had minimal impact on overall performance. However, SPEA2 continued to lag behind, primarily due to its higher execution times. For instance, SPEA2 demonstrates the poorest performance, reported an interference score of 62.82, carbon emissions of 203.76, a cost of 11.88, and an execution time of 101.57 seconds.

On the other hand, NSGA-II maintained superior results, achieving interference as low as 20.41, carbon emissions of 14.86, and cost of 6.45 for cxpb=0.8, mutpb=0.1. NSGA-II demonstrates superior execution efficiency, with times as low as 22.77 seconds. NSGA-II consistently appears as one of the dominant algorithms, demonstrating robust exploration of the solution space and offering a diverse set of Pareto-optimal solutions; which is an asset for effective decision-making.

Algorithm	cxpb	mutpb	Interference Score	Carbon Emissions (gCO2eq/kWh)	Cost (€)	Execution Time (s)
NSGA-II	0.8	0.05	**19.32**	**17.64**	**6.05**	**21.14**
SPEA2	0.8	0.05	68.25	208.44	11.97	106.08
NSGA-II	0.8	0.1	**20.41**	**14.86**	**6.45**	**22.77**
SPEA2	0.8	0.1	62.82	203.76	11.88	101.57

Table 2.5: Comparison of NSGA-II and SPEA2 on simulated dataset for population size: 500 and generations: 10

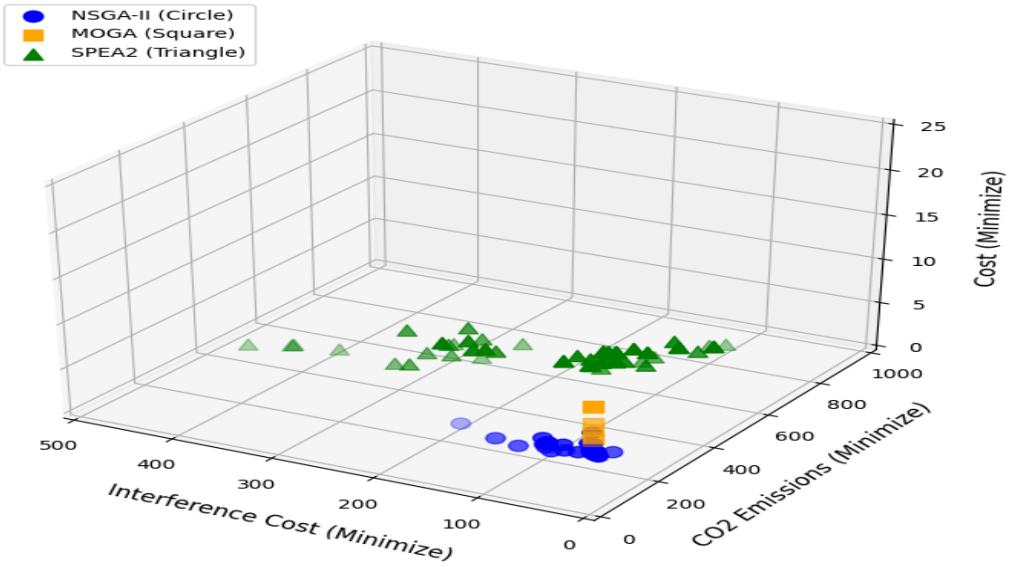


Figure 2.21: Population size = 500, Gen=10, cxpb=0.8, mutpb=0.05.

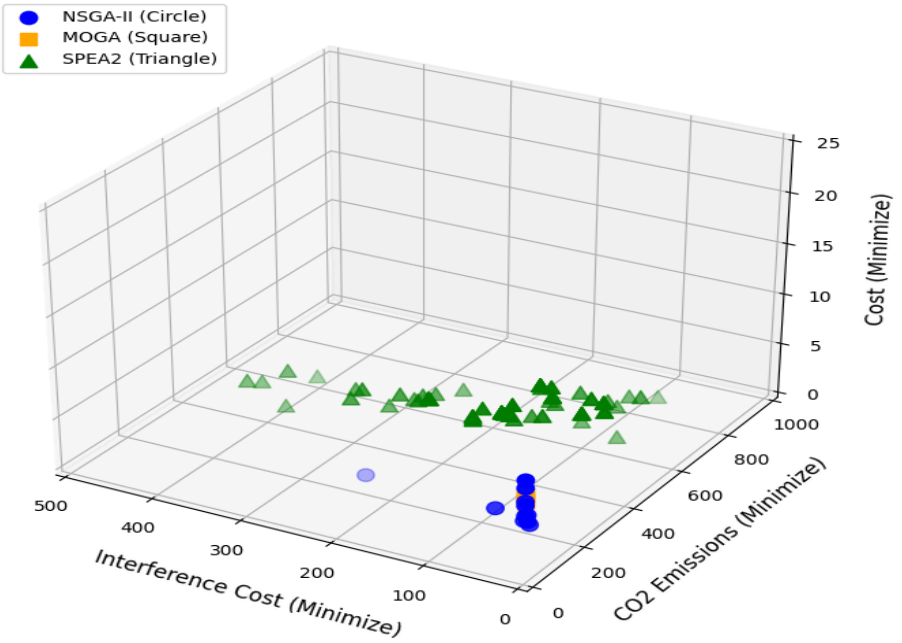


Figure 2.22: Population size = 500, Gen=10, cxpb=0.8, mutpb=0.1.

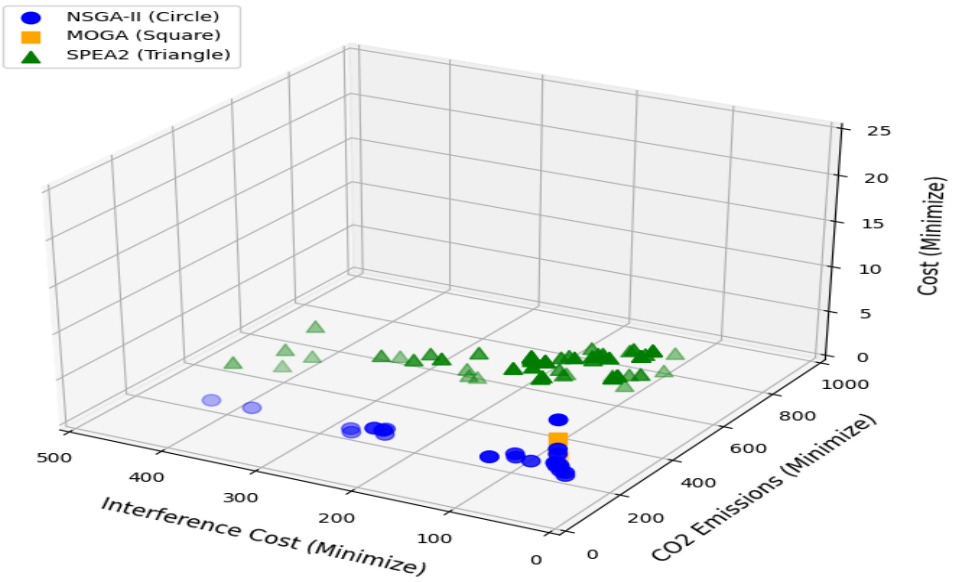


Figure 2.23: Population size = 500, Gen=10, cxpb=0.9, mutpb=0.05.

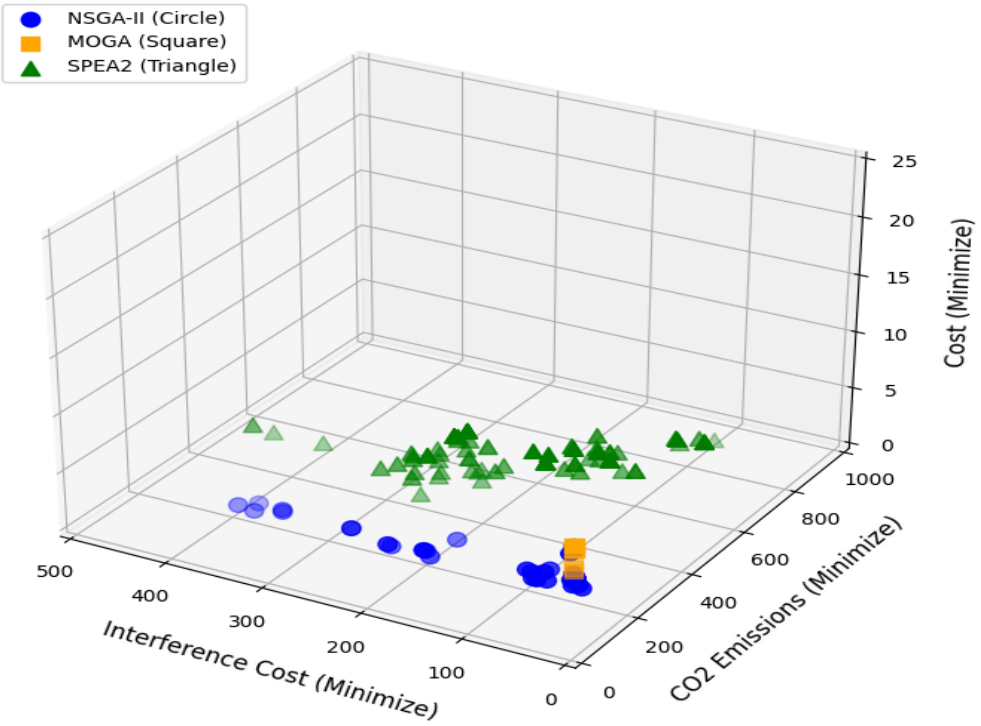


Figure 2.24: Population size = 500, Gen=10, cxpb=0.9, mutpb=0.1.

2.2.2 System Modelling and Simulations

a) Modelling for energy-aware continuum systems

Continuum infrastructures are expected to consume significant amounts of energy; data centres alone are projected to consume 10% of global energy by 2030 [29], whilst demand and complexity is expected to further increase to support over 50 billion IoT devices. Added to this, a reliance on ML-based training and inference is causing data centres to draw massive electrical power, with approximately 40-50% of a data centre's operational costs attributed to electricity bills[2].

Given the cost of Continuum operations, there is a clear need to reduce energy consumption – typically through efficient orchestration mechanisms. Efficiency can be defined in multiple aspects - reducing data transmission costs, maximizing resource utilisation, processing data closer to the user to reduce latency-induced overheads, etc. A particularly promising approach is in *distributing compute load based on Carbon Intensity (CI) and green energy availability in a particular region* to maximise green energy utilisation.

To achieve this, we are investigating how to reason across not only Continuum resources, but the underlying electrical infrastructure on which they consume. Specifically, we are investigating h*ow Continuum orchestration can interact with Smart Grid technology*. The Smart Grid provides decentralized power generation, enhancing reliability, real-time monitoring, and consumer empowerment, etc.

To design novel methods for improving energy efficiency and sustainability in such infrastructures, it is important to accurately represent the Continuum, and formal models must be created that have integrated the Continuum with energy considerations (e.g. energy providers, the Smart Grid, etc.). Furthermore, model-based solutions are required to address energy concerns rigorously for autonomous resource management in the continuum environment. However, there remains a notable gap in the literature concerning energy models specifically tailored for the Edge-Cloud Continuum instead of traditional clouds [30]. Moreover, different components of energy (such as energy providers and import/export of power) addressed separately across existing literature need to be integrated into the Edge-Cloud Continuum for comprehensive energy modelling and simulation of models. To address this, as part of D4.4 we:

- Present - for the first time - a formal model that integrates Smart Grid and Cloud-Fog-Edge Continuum components, their interactions, and their characteristics.
- Present mathematical models that can be utilised to transform the model into a simulation by introducing container-level granularity, idle and busy power, and carbon intensity of the Smart Grid.
- Develop a model-based simulation, showing the transformation of the model into a simulation and validating the significance of the model.

[2] https://encoradvisors.com/data-center-cost/ (accessed on 14-April-2025)

a1. Integrated Continuum/Smart Grid model

A model should capture the key elements of the integrated Continuum and Energy system, whereas, in addition, our formal model rigorously defines the components interacting with each other and their characteristics to reduce carbon emissions. Formal models aiming to represent problems appropriately and provide guidelines are usually validated through rationales and simulations.

The evolved Continuum involves horizontally and vertically federated data centre layers to address the concern of latency, but apart from latency, many other factors play their role in Continuum management, and therefore, an appropriate model of the Continuum should cover the guidelines addressed in this section. The Continuum system has a heterogeneity of platforms, including tools and resources available through Mist, Edge, Fog and Cloud. The Figure 2.25 presents a scenario of six data centres represented as D = {DC_1, DC_2, DC_3, DC_4, DC_5, DC_6}, and these data centres become part of four types of regions, which are Cloud Service Provider (CSP), Workload, Energy Provider (EP) and Geographical regions.

Figure 2.25 shows that there can be more than one CSP; some examples can be AWS, Azure, Google, Red Hat, Alibaba, IBM, etc. Furthermore, service consumers can also have their own on-premises data centre (which might not be very powerful). The processing of users' tasks can begin anywhere in the Continuum, starting from on-device to data centres such as on-premises, far edge, near edge, and cloud. Different service providers through various Service Level Agreements (SLAs) with other providers and users formulate a loosely coupled collaboration (also known as federation), helpful in flexible decision-making to facilitate operations and ultimately assist their own corporate goals. Along with the introduction of heterogeneous platforms in the Continuum system, all participating CSPs, i.e. the multiple providers, bring more flexibility in massive scalability when needed.

In addition, the integration of multiple layers and providers also increases the complexity of automatically configuring tools and handling the issues through a self-healing process, which is the need of time. Figure 2.25 shows the CSP regions C = {CSP_1, CSP_2}, where CSP_1 = {DC_1, DC_2, DC_3, DC_4, DC_5}, CSP_2 = {DC_5, DC_6} and DC_5 is shared between two CSPs. The data centres can have shared servers also available at the time of need for their collaborators. If one CSP’s data centres are overloaded or are not able to meet some QoS (Quality of Service) like latency, energy, etc., then a shared data centre option can be utilised. The federation among CSPs can be developed in two different ways to implement decentralised control:

1) CSPs can maintain a public log visible to collaborators to share information about resource availability or percentage of utilisation, e.g. 20% of the resources of a data centre are available at a certain time, which can be used by the collaborators. This log can be seen as a partial view of the global infrastructure, while individual CSPs have a full view of their infrastructure. In this first way of federation among CSPs, each CSP is fully autonomous in making decisions about job placement anywhere in the data centres owned by it. The Local Controller (LC) of CSP has a property of sovereignty in terms of self-management to handle jobs received from the collaborators.

2) CSPs can also have some global view of the concurred infrastructures, so an LC of a CSP can also make job offloading decisions itself, rather than handing over the job to the LC of the collaborator. Therefore, in this type of federation, LC is a kind of Global Controller (GC) with more control over all available resources within a CSP region.

Figure 2.25 also shows multiple workload or application regions. There are a total of 4 applications A = {A1, A2, A3, A4}, shown in the figure. Multiple collaborating CSPs have parallel workloads, facilitated by multiple data centres, represented as A1 = {D1}, A2 = {D1, D2, D3, D5}, A3 = {D2, D3, D4, D5}, and A4 = {D5, D6}. Considering the CSP regions, we can further realise that CSP1 has a dedicated workload A1; apart from A1, all applications are using the shared data centre D5 owned by one of these CSPs. For a realistic Continuum representation, a model for the Continuum must consider the workload regions having conflicting QoS, like response time, not giving enough room to another workload in the same data centre. The Continuum system has a characteristic of infrastructural dynamicity because resources can be in mobility (e.g. information processing by a moving autonomous vehicle), and application users can fluctuate. Therefore, effective management is required to connect or divert users to the most suitable node and scale the resources by turning them on/off, etc.

From Figure 2.25 we can also see that there can be multiple Power Source Regions (PSRs) or Power Distribution Network (PDN) routes represented as PSR = {PSR1, PSR2, PSR3, PSR4, PSR5, PSR6, PSR7, PSR8}, where PSR1 = {DC3}, PSR2 = {DC1, DC3}, PSR3 = {DC1, DC2, DC3}, PSR4 = {DC2}, PSR5 = {DC2, DC5}, PSR6 = {DC5, DC6}, PSR7 = {DC4}, PSR8 = {DC4, DC6}. The diagram shows that DC3 can be powered by {PSR1, PSR2, PSR3}. A PDN can have many sources of energy, and a provider can have multiple PDNs. This makes the overall energy-related decision-making more complex due to the versatility of providers, sources, intermittent production behaviours, agreements, load distribution, balancing, and energy factors conflicting with other QoS. If at any moment one EP does not have enough green energy left, the workload or job can be placed or migrated to a data centre currently powered by a green source, be it from green PSR, on-site, or another EP. Apart from the availability of green energy, cost and CI that can be the reason behind picking a certain source/provider or migrating the job somewhere.

From Figure 2.25, we can see that the data centres of a Continuum can be dispersed across various geographical boundaries. Boundaries may represent cities, states, countries, Edge layer regions, Fog layer regions, Cloud layer regions, data-regulation boundaries, security-defined regions, etc. In the figure there are two geographical regions G = {G1, G2}, where G1 = {DC1, DC2, DC3, DC5} and G2 = {DC4, DC6}. Suppose that if the carbon intensity of G2 > G1 is high at a certain time, then a job of A4, which is part of both regions, can be placed in G1. Overlooking such regional considerations can compromise the model decisions involving a complex continuum.

In Figure 2.26, an integrated Smart Grid model is presented. For simplicity, it is divided into three layers: power, logic and distribution. The power layer consists of different power sources, including green, renewable (not all renewable sources are entirely green), brown and nuclear power (low carbon). A major property of renewable energy sources is the intermittence of production, rising uncertainty, i.e. the pattern of production is very

inconsistent, and it varies significantly based on region, weather conditions, month and time of day. The power layer highlights that the users or consumers of the power can also be a source of power for the Smart Grid. Any time new sources can be added or removed dynamically. Users can have solar panels, which they can either use directly as a power source - feeding any onsite surplus to the Smart Grid, or the harvested power can be directly fed to the Smart Grid with adjustments made in the bills. Apart from the green sources contributed by the users, there are dedicated green and brown sources as part of PDN or EPs. Although energy is produced as much as required to balance the frequency, batteries on the user side or in the Smart Grid are used to store the surplus energy from green and brown sources.

The logic layer represents the decision-making phase of the Smart Grid. There can be multiple PDNs, e.g. a national grid having power infrastructure spread across multiple cities, where each city's infrastructure can be called a PDN. Furthermore, there can be multiple power providers in the form of microgrids, the national grid and agreements of import with other EPs. Communication among multiple PDNs can be addressed by the Meta-PDN. PDN knows green-only, brown-only and power-mix sources. Green-only sources can come from green onsite surplus or dedicated green sources. Brown can only come from dedicated brown sources. Power-mix comes from dedicated brown, surplus battery storage and imported power from other PDNs and regions. The same surplus storage contributing to the power-mix can be used to export power to other regions and PDNs.

The third layer is the distribution layer, representing a Continuum consisting of several data centres. Some have multiple power sources, including the on-site power of the data centre. According to the requirement, source availability, and power distribution agreements (if any) with the user, power sources are utilised, where PDN is responsible for dynamically deciding the source to power a particular region.

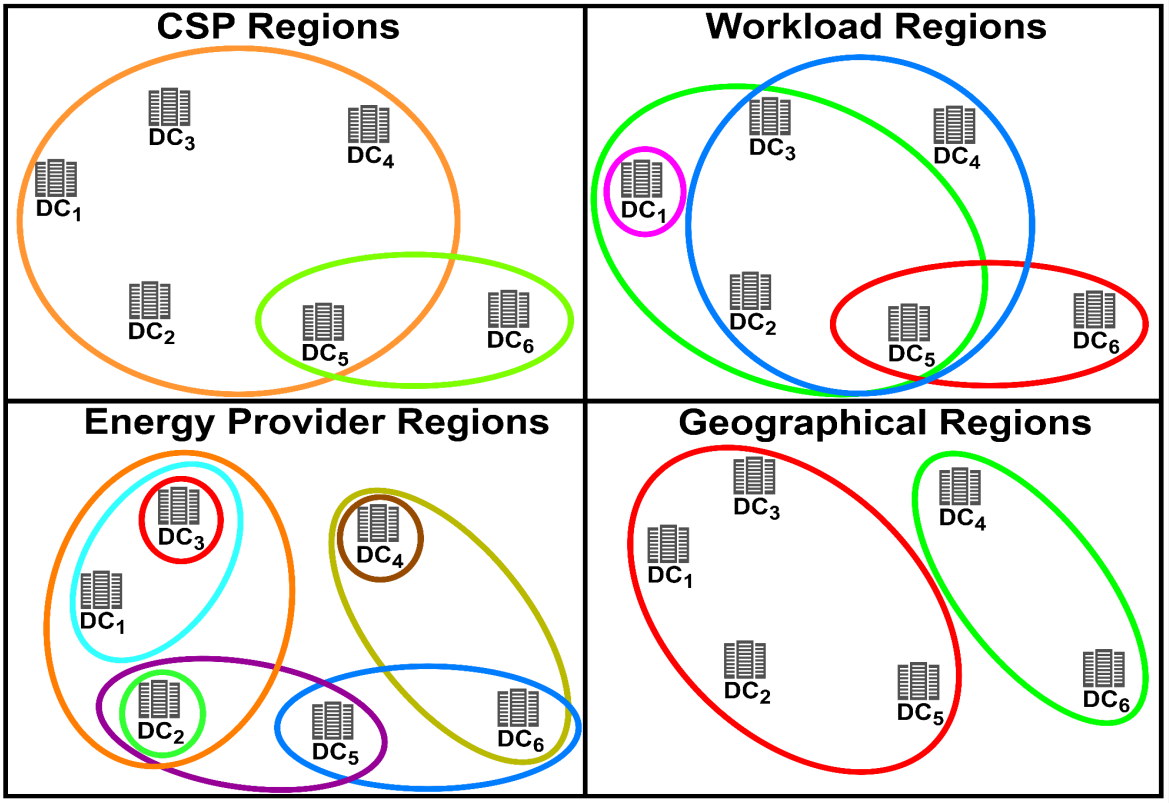


Figure 2.25: Regional Representation

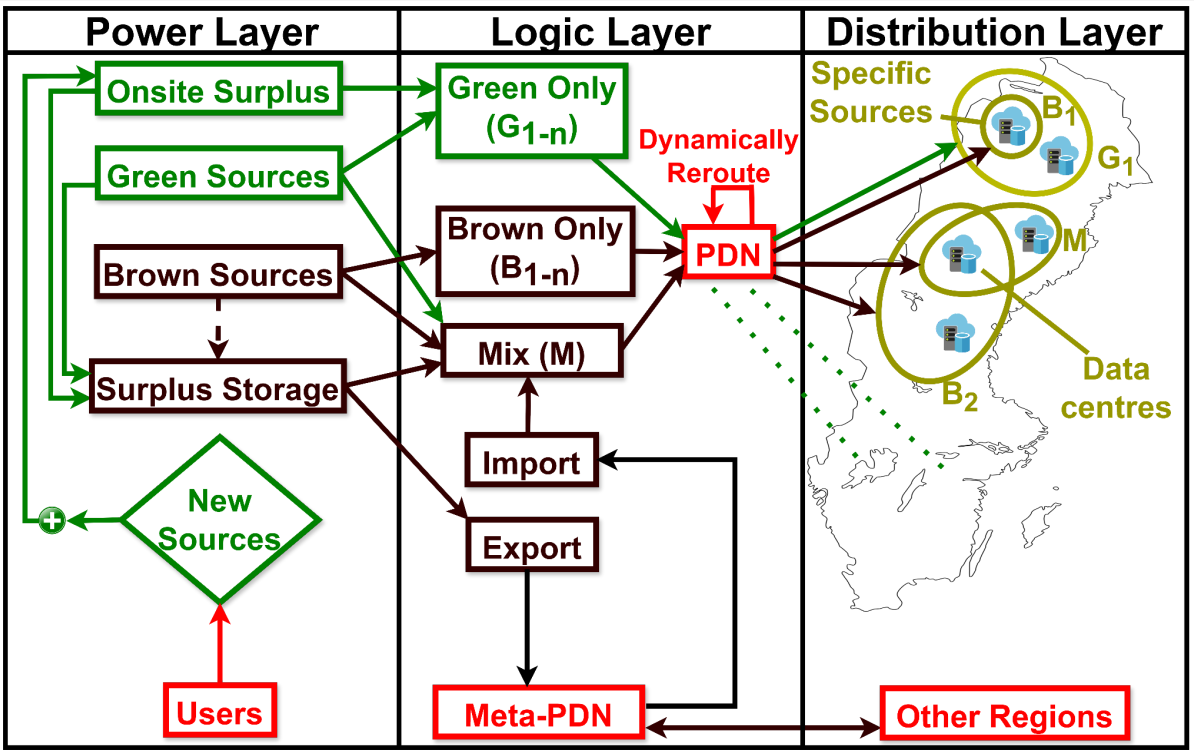


Figure 2.26: Smart Grid Representation

a2. Mathematical modelling

The evolved nature of the Continuum requires novel mathematical models to drive the simulation and show the components' interaction of the Smart Grid and Edge-Cloud Continuum. In the proposed mathematical models, for the first time to the best of our knowledge, the container-level granularity, idle power, busy power, and CI are collectively introduced as contributors to energy and other metrics.

MS_C indicates the makespan of container 'i', which is the maximum or overall time taken by a set of jobs k.

$$MS\_C = Max\,(T_k),\ \forall\, Jobs\ k \in Containers\ i$$

The makespan of server 'j', denoted as $MS,$ is the maximum makespan of containers that have run on it for the time under consideration.

$$MS == Max\,(MS\_C_i),\ \forall\, Containers\ i \in Server\ j$$

The share ratio of container 'i', denoted as $SR\_C,$ defines the limit of processing power. In the simulation limit ratio can be calculated by dividing the MIPS of a container by the MIPS of the server.

$$SR\_C = \frac{MIPS_i}{MIPS_j},\ Container\ i \in Server\ j$$

The energy consumption by container 'i', represented as $EC,$ is the subset of busy power $P_{busy_j} \times SR\_C_i$ consumed for MS_C_i time.

$$EC = MS\_C_i \times P_{busy_j} \times SR\_C_i,\ Container\ i \in Server\ j$$

The energy consumption by a server, denoted as $E_s,$ is based on idle and busy power of the server. The P_{idle_j} power is always consumed, whereas the busy power is consumed in addition to the idle power when some workload is assigned to a server. The summation of total idle power and energy consumed by all containers is the energy consumption by the server.

$$E_s = MS_j \times P_{idle_j} + \sum_{i=1}^{n} \left(EC_i\right),\ \forall\, Containers\ i \in Server\ j$$

The energy consumption by the DC is based on all servers 'j' that are part of that DC.

$$E_{DC} = \sum_{j=1}^{n} E_{s_j},\ \forall\, Server\ j \in DC$$

The energy consumption by edge, fog, cloud or geographical region depends on the summation of energy consumption by all DCs that belong to the region.

$$E_{E/F/C/G} = \sum_{k=1}^{n} E_{DC_k},\ \forall\, DC\ k \in E/F/C/G$$

The Server's Energy Full Busy portion, denoted as E_{SFB}, is the maximum possible power drawn from the server regardless of the workload.

$$E_{SFB} = MS_j \times P_{busy_j}$$

The energy consumption of the Server for which containers of the workload are responsible, denoted as E_{SW} depends on the energy consumed by all containers belonging to the workload running on the server 'j'. Server 'j' can be multiple, running the containers of the workload 'w'.

$$E_{SW} = \sum_{i=1}^{n} \left(EC_i\right),\ \forall\ Containers\ i \in Workload\ w,\ Server\ j \in S_w,\ where\ w\ is\ running\ on\ j$$

Share Ratio of Workload, i.e. the percentage of energy consumed by containers of the workload out of the Full Possible Energy of Server is given by SR_W.

$$SR\_W = \frac{E_{SW}}{E_{SFB}}$$

A workload is not responsible for all the idle power consumed by the set of Servers 'Sw' running the workload. Therefore, we propose to assign a subset of idle consumption of servers in addition to busy power consumption by containers i.e. E_{SW}. We attribute an idle portion of the same ratio as of busy share, achieved through $MS_j \times P_{idle_j} \times SR\_W$ for each server.

$$EW = \sum_{j=1}^{n} \left(\left(MS_j \times P_{idle_j} \times SR\_W\right) + E_{SW}\right),\ Server\ j \in S_w,\ where\ w\ is\ running\ on\ j$$

Energy consumption by a CSP does not depend on DCs because a DC may be running workloads owned by another CSP. Therefore, energy consumption is determined by the workloads belonging to the CSP.

$$E\_CSP = \sum_{w=1}^{n} EW_w,\ \forall\ Workload\ w \in CSP$$

Carbon Emission of Container 'i', denoted by $CECont$, is the sum of the products of kWh EC_i in interval 'k' and gCO2-eq/kWh CI in every interval 'k' of regions set 'R' where the containers are deployed.

$$CECont = \sum_{k=1}^{n} EC_{i_k} \times CI_{k_R}$$

Carbon Emission of Container 'i' average, denoted by $CECont_A$, takes the average of CI in regions 'R' during the intervals set 'K'. The CI average is helpful when CI readings do not exactly match with the container's life cycle 'K'.

$$CECont_A = EC_i \times CI_{K.Avg.R}$$

Carbon Emission Idle, denoted as $CEIdle$ is the product of the sum of all the idle power of servers 'j' in region 'r', each interval time 'k', and CI during interval 'k' for which separate CI readings are available.

$$CEIdle = \sum_{r=1}^{l} \sum_{k=1}^{m} \left(\sum_{j=1}^{n} P_{idle_j} \right) \times k \times CI_{k_r}$$

The total Carbon Emission, denoted as CE, is the sum of Carbon emitted by containers and total idle carbon emissions.

$$CE = \left(\sum_{i=1}^{n} CECont_i \right) + CEIdle$$

The Host Resource Utilisation Ratio (HRUR) is the energy consumed by the host out of the total possible consumption.

$$HRUR = \frac{\sum_{i=1}^{n} (EC_i)}{E_{SFB_j}}, \forall Containers\ i \in Server\ j$$

Our proposed Average Resource Utilisation Ratio is not based on time estimates; rather, it is an average HRUR.

$$ARUR = \frac{\sum_{j=1}^{n} HRUR_j}{n}$$

a3. Initial Results and Analysis

A subset of Azure Functions real traces analysed in [31] for July 2019 is utilised in the simulation. The traces give a pattern of function invocations concerning the time of the day, duration and ending time of functions belonging to different workloads or applications running on the Azure Functions platform. We have considered two workloads, W1 and W2, represented with the hashes Workload_73427…(truncated) and Workload_85479…(truncated), respectively, in the original dataset. These workloads have 10 and 9 functions, respectively. The time duration of function invocations, or in other words, jobs arrival time considered is 0s to 16740.00579s. CPriorityN has been tested in 6 different settings, namely CPriorityN, CPriorityLessDCN, CPriorityLessHostN, CPriorityAustraliaN, CPriorityUSAN, and CPriorityAustraliaM.

CPriorityUSAN, and CPriorityAustraliaM. CPriorityN has 3 EPs, one from the USA and two from Australia. In CPriorityN CI is of night 'N' time, where the night-time is 1-Jan-2024 00:00 – 04:39 hours. CPriorityLessDCN has 2 DCs less than CPriorityN, and the rest of the settings are the same. Similarly, CPriorityLessHostN has 2 fewer hosts than CPriorityN. CPriorityAustraliaN and CPriorityAustraliaM only consider Australian EPs, i.e. EP3 and EP4, but CPriorityAustraliaM has considered morning 'M time CI values, i.e. 1-Jan-2024 06:00 – 10:39 hours.

The following five scenarios are considered for the initial validation with these 6 settings, while all other settings are the same:

- Different number of EPs (CPriorityN vs CPriorityAustraliaN).
- Different EPs but same number of EPs (CPriorityUSAN vs CPriorityAustraliaN).
- Different number of host machines (CPriorityN vs CPriorityLessHostN).
- Different number of DCs (CPriorityN vs CPriorityLessDCN).
- Different time zones (CPriorityAustraliaN vs CPriorityAustraliaM).

The CPriorityN algorithm is as follows:

Algorithm: CPriorityN

Input: *R*~(set of regions), DC_r ~(DCs in region r), *H*~(set of hosts), *j*~(incoming job)

Output: Job *j* assigned to FaaS container

Step 1. $Wait\ for\ a\ job\ j\ to\ arrive$

Step 2. $\mathbf{r}^* \leftarrow \min_{\mathbf{r}\in\mathcal{R}} \mathbf{CI(r)}$

Step 3. $RA(d) = \frac{totalAvailableMips(d)\times 100}{totalMips(d)}, \quad \forall d \in DC_{r^*}$

Step 4. $\boldsymbol{d^*} \leftarrow \max_{d\in DC_{r^*}} RA(d)$

Step 5. $h^* \leftarrow NextHost_{RR}(d^*)$

Step 6. $If\ \exists c \in h^*, such\ that\ S_j = service(c)\ and\ Load(c) < 2$

Step 7. $Then\ c \leftarrow j$

Step 8. $Else\ c' \leftarrow NewContainer(S_j)\ for\ S_j\ and\ c' \leftarrow j$

Step 9. $Return\ to\ Step\ 1$

Figure 2.27: Carbon-aware Scheduling of FaaS

These five scenarios demonstrate that Smart Grid factors like (EPs count, same count but different EPs, different time zones showing intermittence), highlighted in the model, can have an impact on carbon footprints (CF) of the Continuum, for the same Continuum settings. Similarly, the Continuum factors like (number of DCs, number of hosts) can also affect CF of the Continuum, for the same Smart Grid settings.

The Figure 2.28 shows the number of FaaS function invocations (jobs) handled by Geographical Regions (GRs), Energy Provider Regions (EPRs), Non-federated Regions (NFRs) - if the subset of the workload of CSP1 gets mapped to the DCs owned by CSP1 then this scenario is of a non-federated region, and Federated Regions (FRs) - if the subset

of the workload of CSP1 gets mapped to the DCs owned by CSP2 then this scenario is of a federated region. Looking at these five scenarios, it is evident from the jobs placement in different settings of the Smart Grid and Cloud-Edge Continuum, that Smart Grid settings impact the Continuum's carbon emissions, and similarly, a minor change in the Continuum settings also impacts the carbon emissions associated with the Smart Grid.

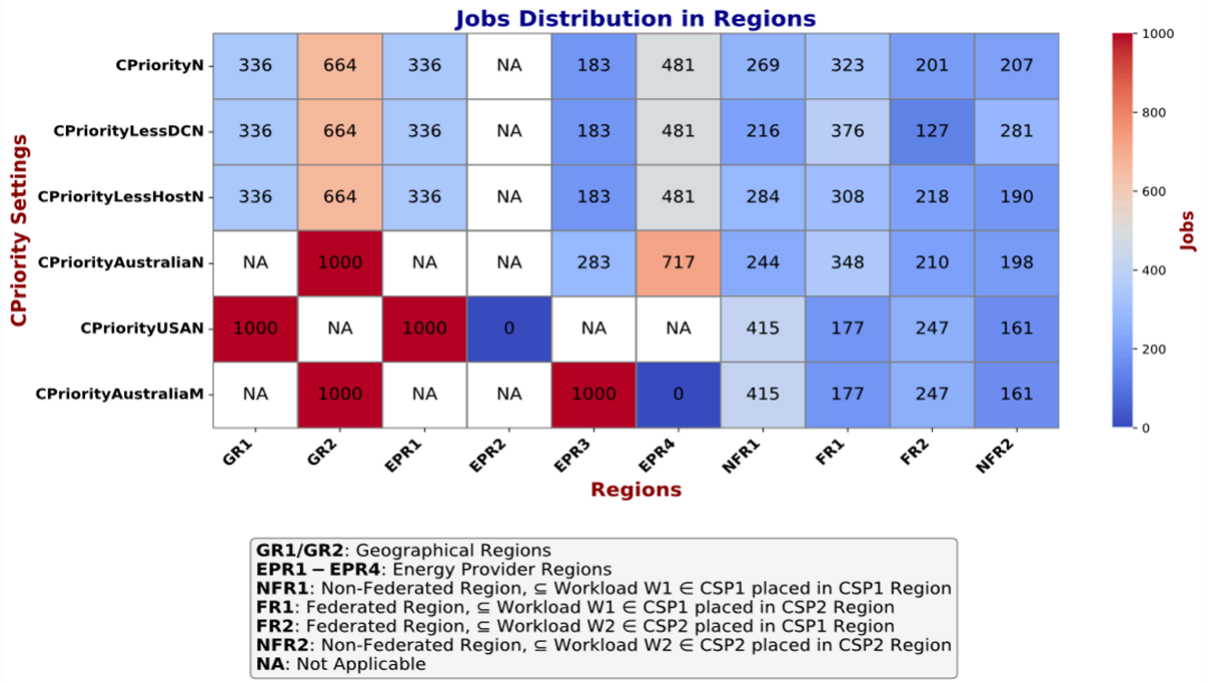


Figure 2.28: Jobs Distribution in Regions

Figure 2.29 shows the carbon emissions for six different settings of the CPriorityN algorithm. When CPriorityN is configured with fewer DCs and hosts (i.e. minor differences in Continuum settings but the same Smart Grids), carbon emissions are reduced. The comparison of CPriorityN with CPriorityAustraliaN (both have different number of EPs) shows that for the same set of serverless function invocations, the Smart Grid settings associated with the number of EPs influence the carbon emissions. This trend is observed because the CI of each Smart Grid is different and the algorithms take advantage of multiple EPs to place functions' invocations to the least CI region. The CPriorityUSAN and CPriorityAustraliaN have the same count but different EPs. A similar variation of carbon emissions is observed for these settings because from Figure 2.28 we can see that one EP region of the USA gets exactly zero jobs. The final scenario considers the same function invocations at two different times i.e. Morning vs. Night hours of Australia on the same day. The CI values for the same Smart Grid shows significant variation across different time zones, as Smart Grid energy production and demand fluctuate due to factors such as weather, time of day, and regions etc.

Therefore, it is necessary to consider the integration and also the guidelines presented in the formal model because the integrated model has significant potential for reducing carbon footprints.

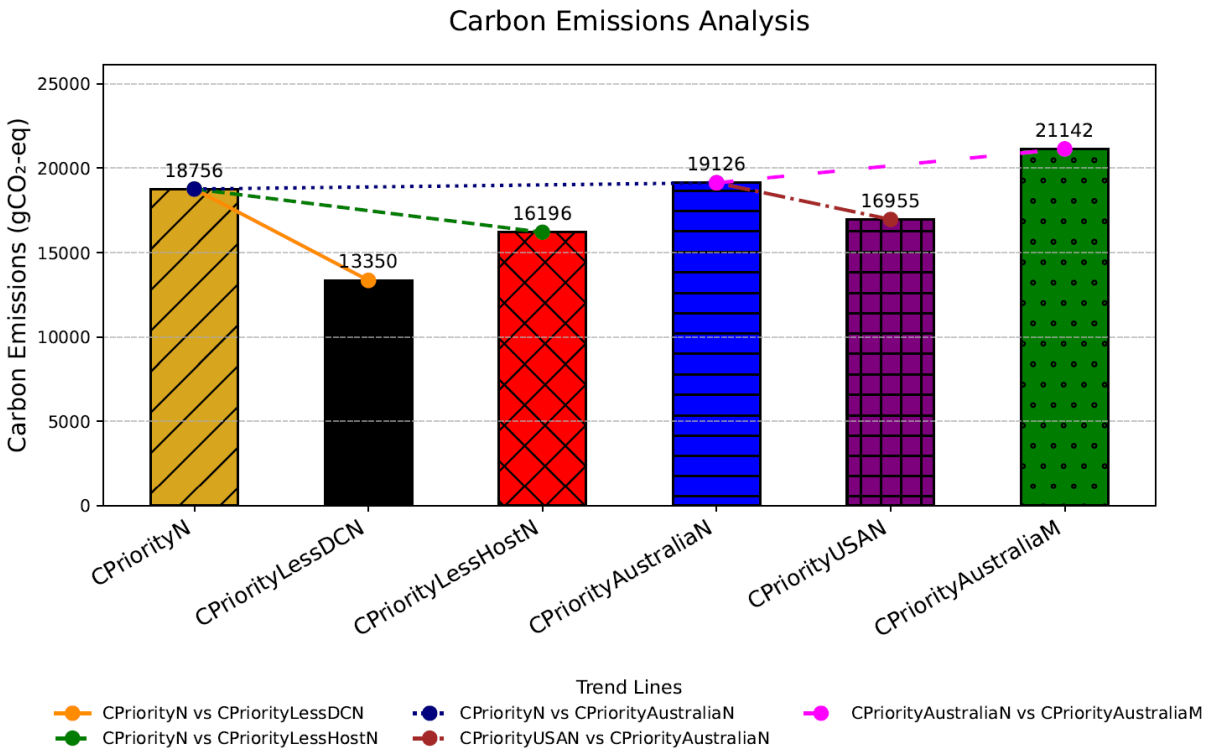


Figure 2.29: Carbon Emissions Analysis

b) Energy-aware scheduling

As part of the research activities, we wanted to investigate the potential of using different strategies for carbon- and energy-aware scheduling. More specifically, we wanted to:

- Understand the impact on carbon emissions of different workload placement and scheduling strategies.
- Investigate the potential of using time-shifting strategies to lower carbon emissions.
- Investigate the potential of developing relatively simple decision-making algorithms that can make placement and scheduling decisions in close to real-time.
- Create a simulation environment to investigate and compare the performance of such different strategies.

As part of these investigations we developed a spatiotemporal carbon-aware scheduling algorithm for workload placement across heterogeneous platforms, a discrete-event simulator capable of replaying realistic workloads from the MIT SuperCloud dataset, and performed a comprehensive empirical evaluation. The following text and experiments are reproduced with some modifications from a conference paper that will be presented at the 8th IEEE Conference on Industrial Cyber-Physical Systems (ICPS) in May 2025.

b1. Mathematical model

Given a set of compute clusters C, each cluster $c \in C$ is characterized by a CO2 intensity function $i_c(t)$, which provides the predicted carbon intensity of using the cluster at a given time t, measured in grams of CO2 per kilowatt-hour (gCO2/kWh).

Letting P be a set of processes to be executed, we characterise each individual process $p \in P$ by a power consumption function $e_p(t)$, measured in watts (W), which returns the power required to execute the process at time t. Furthermore, each process p has a deadline function $d(p)$, specifying the remaining time before it must start. The set of processes running on cluster c at time t is denoted $P_c(t)$, while $t_{submitted}(p)$, $t_{start}(p)$ and $t_{end}(p)$ represent the submission, start and end times of the process p, respectively.

The carbon emissions of an individual process can then be calculated as follows:

$$E_p = \int_{t_{start(p)}}^{t_{end(p)}} e_p(t) \cdot i_c(t)\, dt$$

The problem of minimizing the cumulative carbon emissions E for all processes and clusters can be formulated as the following multi-objective minimization problem:

$$min\ E, \text{where } E = \sum_{p \in P} E_p,$$

Minimizing over all possible assignments of processes to clusters subject to these constraints:

$$t_{start}(p) \leq t_{submitted}(p) + d(p),\ \forall p \in P,$$

$$\sum_{p \in P_c(t)} r(p) \leq R(c), \forall t, \forall c \in C.$$

Here, $r(p)$ represents the resource requirements of a process p, $R(c)$ represents the total resource capacity of a cluster c. The goal is to develop a scheduling algorithm that minimizes E, while ensuring all processes meet their respective deadlines and also respecting resource availability.

b2. Scheduling algorithms and strategies

Several different strategies could be used to reduce carbon emissions, depending on the type of workload. For high-energy workloads, one would like to do the following:

> **Strategy 1:** Run the most energy-intensive workloads on servers powered by the lowest CO_2 intensity energy sources.
>
> **Strategy 2:** Delay (time-shift) workloads to periods when CO_2 intensity is lower to reduce carbon emissions.

For low-energy workloads, one would instead like to use the following strategies:

> **Strategy 3:** When servers powered by low CO_2 intensity sources are scarce, prioritise running low-energy workloads on servers with higher CO_2 intensity.
>
> **Strategy 4:** When low CO_2 intensity periods are fully booked and a higher-energy workload is scheduled, reschedule lower-priority workloads to higher CO_2 intensity periods to make room for the higher-energy workload.

It is much more complex to handle mixed-energy workloads, those that vary between periods of high and low power utilisation. To find a strategy that balances execution time of the decision-making algorithm with performance, we decided on the following:

> **Strategy 5:** Use the mean power consumption as a representative value for scheduling decisions.

Based on these observations, we propose two different algorithms for our initial investigations, leveraging all these strategies to optimize carbon efficiency:

- *Greedy cluster selection (Algorithm 1 in Figure 2.30)*, which immediately assigns incoming processes to the cluster with the lowest CO2 intensity that also has available capacity to execute it.
- *Time shifting (Algorithm 2 in Figure 2.31)*, which leverages temporal variations in CO2 intensity, by delaying execution of processes until the forecasted CO2 intensity of the clusters reaches its lowest level, without exceeding the deadline for the process. This approach requires predicting the CO2 intensity of clusters over time and reserving time slots for running processes given their remaining deadlines before they to be started.

Algorithm 1 Greedy Cluster Selection

function SCHEDULE(p, C)
  $C_{\text{available}} \leftarrow \{c \in C \mid \text{HASFREERESOURCES}(c, p)\}$
  if $C_{\text{available}} \neq \emptyset$ **then**
    Assign p to $c_{\text{selected}} \leftarrow \arg\min_{c \in C_{\text{available}}} i_c(t_{\text{current}})$
  end if
end function

function HASFREERESOURCES(c, p)
  return $r(p) + \sum_{\widetilde{p} \in P_c(t_{\text{current}})} r(\widetilde{p}) \leq R(c)$
end function

Figure 2.30: Greedy cluster selection algorithm

Algorithm 2 Time Shifting

function SCHEDULE(p, $\boldsymbol{A}$)
  $e_{\text{best}} \leftarrow \infty$, $c_{\text{best}} \leftarrow$ None, $t_{\text{best}} \leftarrow$ None
  for $c \in C$ **do**
    for $t \in [t_{\text{current}}, t_{\text{current}} + d(p)]$ **do**
      if HASFREERESOURCES($c, t, p, \boldsymbol{A}$) **then**
        $e \leftarrow$ EMISSION(p, c, t)
        if $e < e_{\text{best}}$ **then**
          $e_{\text{best}} \leftarrow e$, $c_{\text{best}} \leftarrow c$, $t_{\text{best}} \leftarrow t$
        end if
      end if
    end for
  end for
  if $c_{\text{best}} \neq$ None **then**
    p.planned_time $\leftarrow t_{\text{best}}$
    p.planned_cluster $\leftarrow c_{\text{best}}$
    # Update Process Allocation matrix
    for $t \in [t_{\text{best}}, t_{\text{best}} + p.\text{duration})$ **do**
      $\boldsymbol{A}[c_{\text{best}}][t] \leftarrow \boldsymbol{A}[c_{\text{best}}][t] + r(p)$
    end for
  end if
end function

function HASFREERESOURCES($c, t_{\text{start}}, p, \boldsymbol{A}$)
  for $t \in [t_{\text{start}}, t_{\text{start}} + p.\text{duration})$ **do**
    if $\boldsymbol{A}[c][t] + r(p) > R(c)$ **then**
      return false
    end if
  end for
  return true
end function

function EMISSION(p, c, t_{start})
  emission $\leftarrow 0$
  for $t \in [t_{\text{start}}, t_{\text{start}} + p.\text{duration})$ **do**
    emission $\leftarrow$ emission $+ e_p(t) \cdot i_c(t)$
  end for
  return emission
end function

Figure 2.31: Time shifting algorithm

Algorithm 2 in Figure 2.31 outlines a method for assigning processes to clusters and determining an optimal start time by searching the Process Allocation Space. It calculates the estimated emissions for a process at different time steps to identify the scheduling option with the lowest emission. The *HasFreeResources* function checks if a cluster has sufficient resources available at a specified time in the future. The function *Emission* estimates the carbon emissions of running a process at a given time. The *Schedule* function finds consecutive time slots between the current time and the process's deadline that minimize the carbon emission for running the process.

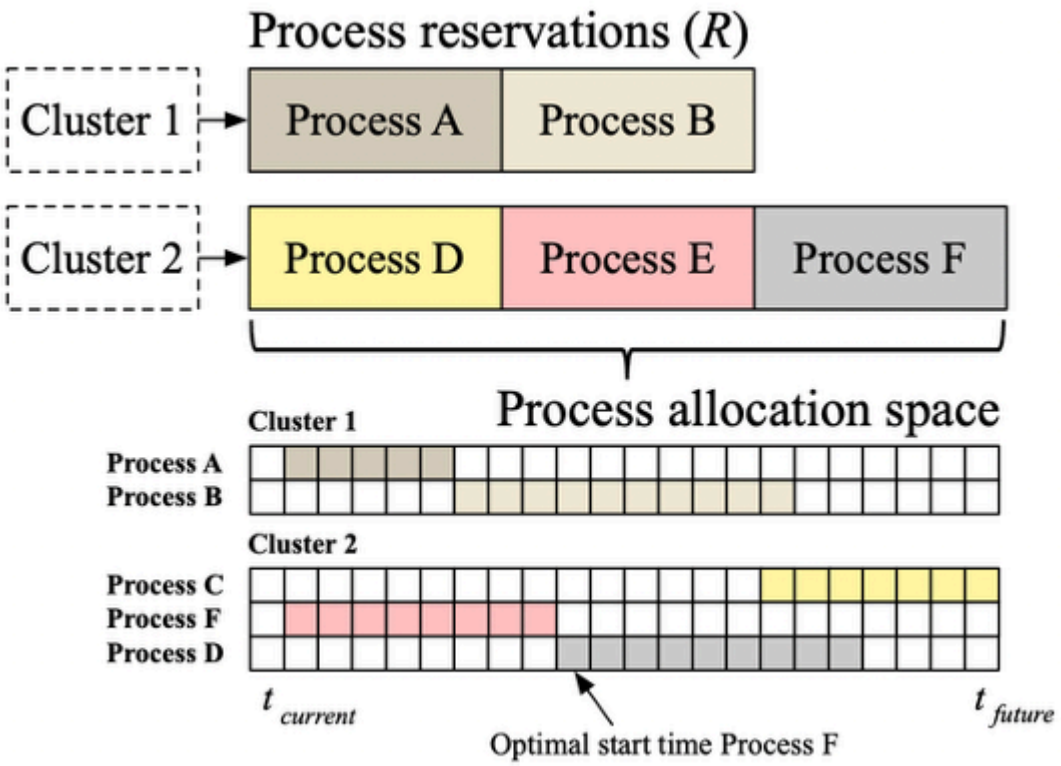


Figure 2.32: Data structures used by the time-shifting algorithm

The time shifting algorithm could be implemented using a hash table to store lists of reserved processes for each cluster. Figure 2.32 illustrates the data structures used by the proposed time-shifting algorithm. Each process is modeled as a data structure containing its predicted power usage, estimated execution time, planned start time, and assigned cluster. It is assumed that users specify a walltime, which defines the maximum duration a process can wait before it must run. From these lists, it is possible to derive a matrix $A\colon C \times T \rightarrow Z$ representing the *Process Allocation Space*, where C is the set of clusters and T is the set of discrete time steps, which can be used to reserve and determine available resources at any future time. The entry

$$A(c,\ t) \ = \sum_{p \in P_c(t)} r(p)$$

represents the amount of resources scheduled to be used on cluster c at the time instance t.

b3. Experimental setup and evaluation

We wanted to compare the time-shifting strategy with a greedy placement algorithm, that simply assigns an incoming workload to the cluster with the lowest CO2 intensity with available processing capacity.

Simulation environment: A discrete event simulator was developed to model workload execution and evaluate the performance of different placement and scheduling strategies. The simulator takes a dataset of time series of different workloads and a time series of CO2 intensity as input and then randomly samples and replays the workloads across a specified set of clusters. Each cluster is associated with a CO2 intensity log file and a predefined number of available GPUs, with GPUs serving as the only resource type for simplicity. The simulator can be configured to use different scheduling algorithms and outputs a dataset and statistical metrics generated from replaying the traces.

Workload dataset: The workload traces were extracted from the MIT SuperCloud dataset [6], which contains a large number of time series of workloads that include power consumption. Note that the dataset does not contain the total energy consumption, but only the energy consumption of the GPUs collected using the *nvidia-smi* command. CO2 intensity data for various clusters was obtained from Electricity Maps[3], which includes 24-hour predictions of CO2 intensity. The simulated clusters used CO2 intensities from European regions, Figure 2.33 shows a few examples.

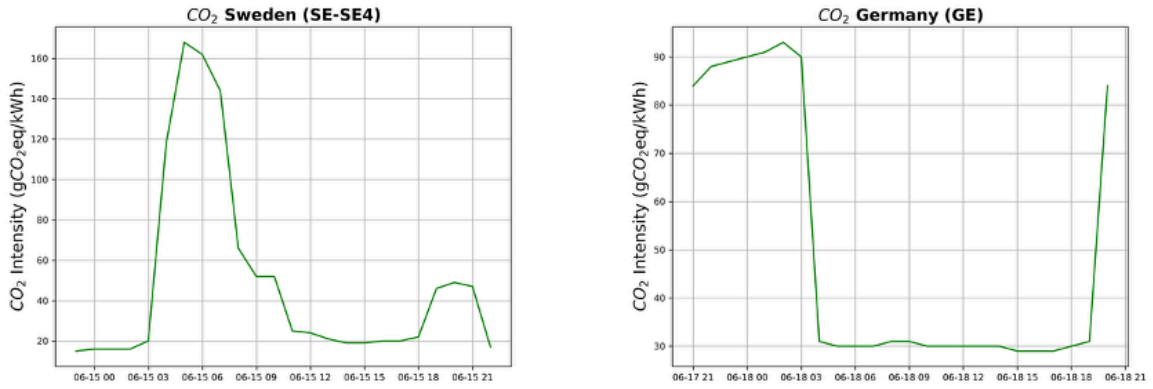


Figure 2.33: Example CO2 intensities for two European regions.

For the experiments, a dataset of 10,000 workloads was randomly selected from the MIT SuperCloud dataset. Some statistics for the dataset is shown in Figure 2.34, and a few example time series are shown in Figures 2.35 and 2.36. Several log files were generated to record the start times of each workload. To control the rate at which workloads were replayed, a random waiting time between workloads was sampled from an exponential distribution. Both the CO2 intensity and workload traces were re-sampled at a resolution of 1 second, meaning that the Process Allocation Spaces also had a 1-second resolution.

[3] https://www.electricitymaps.com

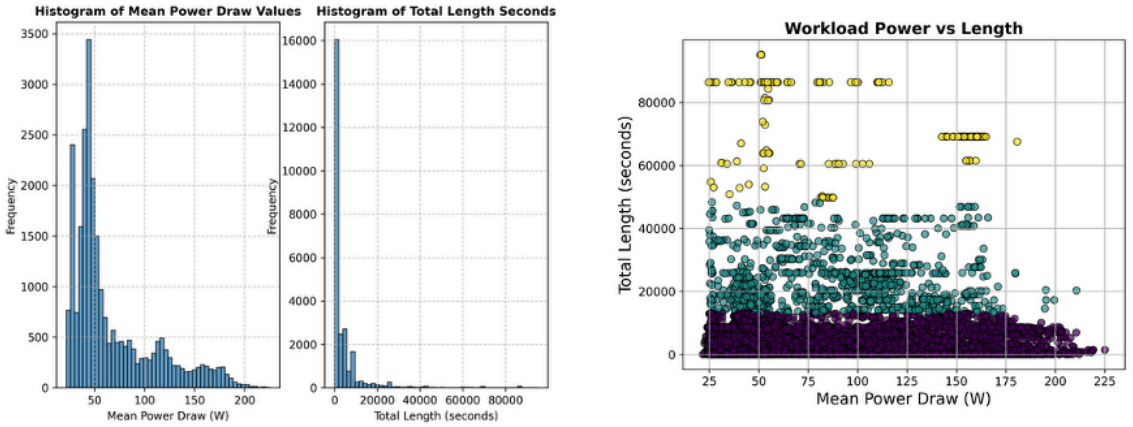


Figure 2.34: Mean power draw and total length of the different workloads in the dataset.

Figure 2.35: Example power consumptions of “high-energy workloads” that use close to the maximum power.

Figure 2.36: Example power consumptions of “mixed-energy workloads” with power usage varying between high and low throughout its execution.

Experiment 1 - Time-shifting: This experiment evaluated how effective the time-shifting algorithm was in reducing carbon emissions under no resource constraints. The simulator was configured to use a single cluster equipped with a large number of GPUs, ensuring that the cluster never ran out of resources. The CO2 intensity dataset associated with the cluster was obtained from southern Sweden (SE-SE4). The results presented in Figure 2.37 show that the time-shifting algorithm outperformed the Greedy algorithm, reducing carbon emissions by 61 %.

Figure 2.38 illustrates the CO2 intensity over time and the carbon emissions of the time-shifting algorithm when process deadlines were set to 24 hours. As can be seen, the time-shifting algorithm effectively reduced emissions by leveraging periods of low CO2 intensity, which aligns well with the expected behavior. However, when the process deadlines were reduced to 12 hours and 6 hours, the effectiveness of the time-shifting algorithm decreased, as shown in Figure 2.37.

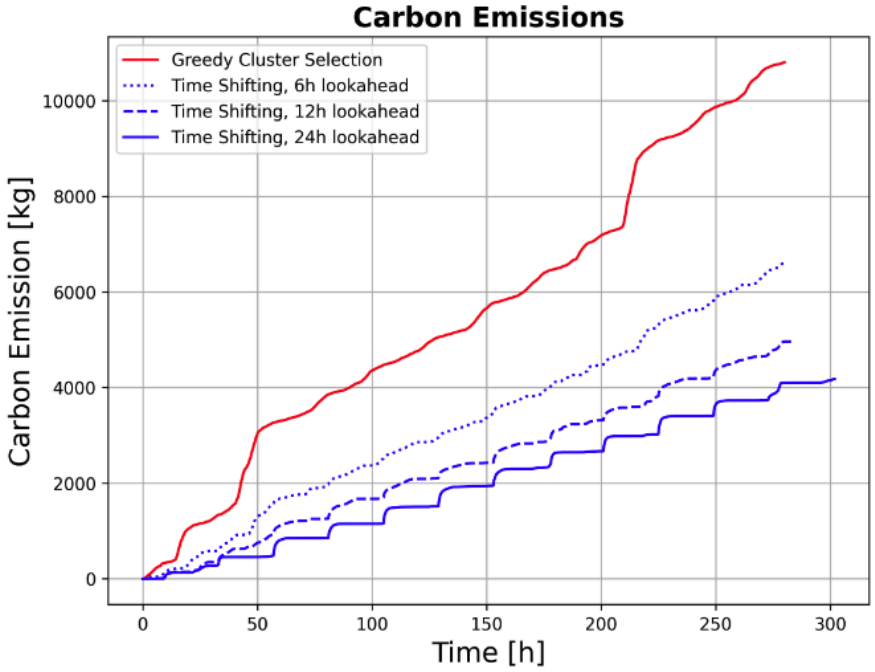


Figure 2.37: Results of experiment 1 that shows the cumulative carbon emissions for different deadlines (6, 12, 24 hours).

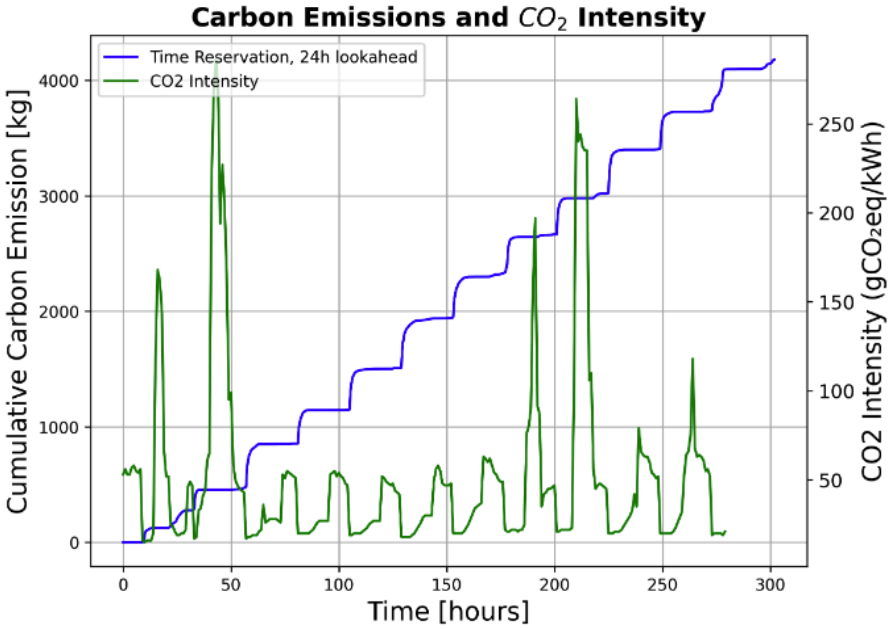


Figure 2.38: Results of experiment 1 showing time reservations and CO2 intensity over time.

Experiment 2 - High Utilization: This experiment aimed to evaluate the performance of the scheduling algorithms under conditions where the low CO2 intensity clusters become fully utilized. The simulator was configured with two clusters. The first cluster, referred to as the green cluster, consisted of 32 GPUs located in northern Sweden (SE-SE1) and characterized by very low CO2 intensity. The second cluster, referred to as the brown cluster, consisted of 127 GPUs, and was located in Poland, with significantly higher CO2 intensity. This setup intentionally created an imbalance between green and brown compute resources and thus provided a controlled environment to assess how effectively the algorithms utilized renewable energy.

The results, presented in Figure 2.39, indicate that the time-shifting algorithm performed significantly worse than the simple Greedy algorithm. The reasons for this performance difference will be discussed in detail in the next section.

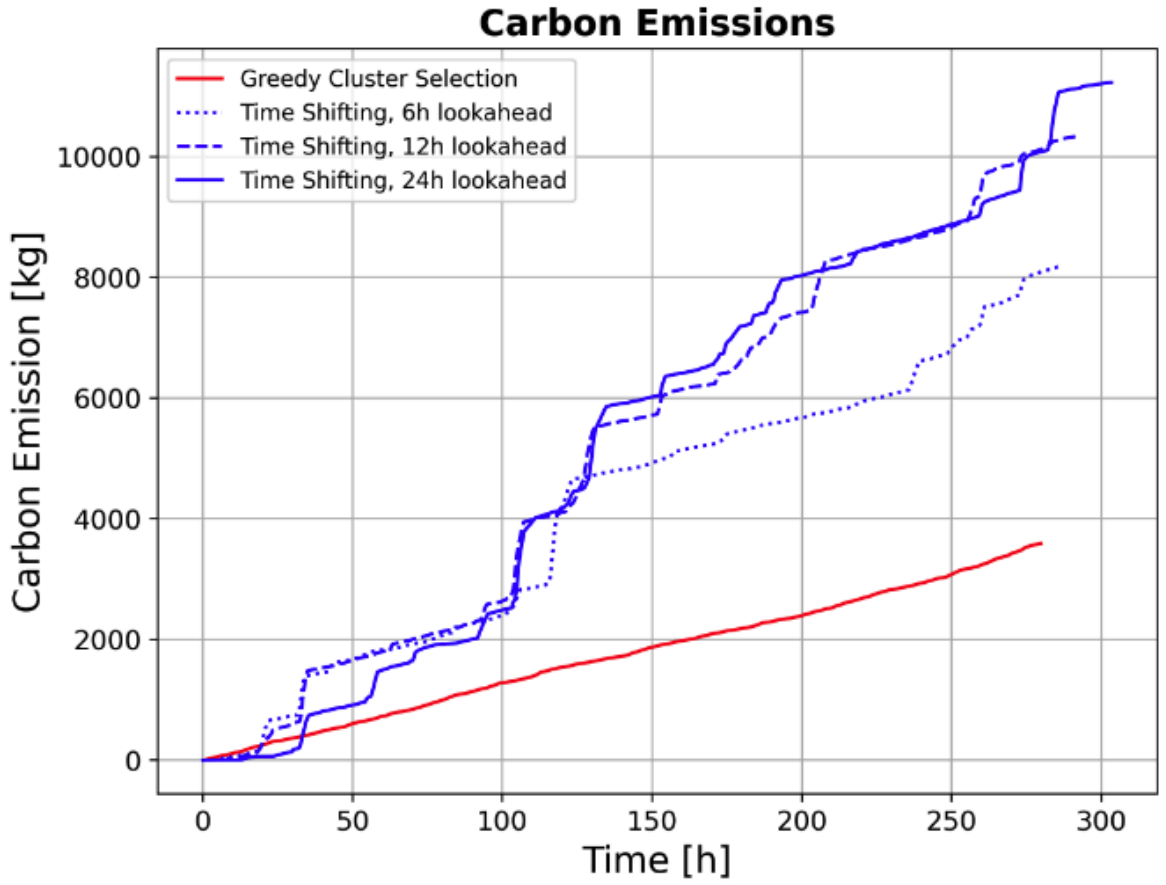


Figure 2.39: Results of experiment 2 (high utilisation scenario) that shows the cumulative carbon emissions for different deadlines (6, 12, 24 hours). Notice that the time shifting algorithm fails to perform as expected in this scenario.

b5. Conclusions and future work

Our findings show substantial reductions in carbon emissions by prioritizing renewable energy sources and the high potential of time-shifting workloads to periods of lower carbon intensity. However, when clusters operate under high utilization, our findings also

show that time-shifting strategies could inadvertently result in significantly higher emissions. In such scenarios, even simpler greedy algorithms could be more effective.

A limitation of the proposed time-shifting algorithm is its tendency to concentrate processes on clusters at certain start times. Although this approach reduced emissions in Experiment 1, it led to unbalanced resource utilization in Experiment 2. As more processes were reserved for future execution and available time slots became scarce, the algorithm's flexibility significantly decreased, forcing processes to be executed on the brown cluster, thus increasing carbon emissions. The Greedy algorithm instead performed better by consistently prioritizing the green cluster.

This may be due to the randomized dataset and its tendency to naturally balance long- and short-lived processes, leading to a more even distribution. A simple improvement to the time-shifting algorithm is to assign new processes to the latest available time slot once cluster resources become exhausted, allowing it to behave more like the Greedy approach under high load.

To conclude, the work has proposed a method to reduce emissions by aligning workloads with periods of low carbon intensity. This can also indirectly contribute to lower operational costs due to the correlation between carbon intensity and electricity prices. However, time-shifting may not be suitable for all workloads, particularly real-time applications.

The time-shifting algorithm could also be adapted to scenarios with processes that must execute in close to real-time, using CO2 and workload forecasts to provision resources from cloud/edge clusters with low CO2 intensity.

Further research is needed to investigate these questions in more detail.

3. Conclusions and future work

This development cycle focused on advancing learning models and model retraining methods (a) To enhance the performance of unsupervised workload classification that enables system efficiency while allocating resources across the Cloud-Edge continuum, the deep K-means enables deep feature learning using an autoencoder; which optimizes the reconstruction loss and clustering loss simultaneously to learn meaningful latent representations and improves the separation of workloads in a lower-dimensional latent space. (b) To achieve optimal model accuracy, an auto-adaptive strategy is implemented that mitigate the data-drift problems and minimizes retraining cost. When system performance declines, the system retrieves a matching time-series batch from the knowledge base and loads the hyperparameters from the associated ML model, adjusting the deployed model to optimize inference accuracy on new data streams.

Three multi-objective optimization algorithms, such as NSGA-II, MOGA, and SPEA2 are designed and implemented, which were formulated as resource optimization as multiobjective problems by defined objectives, such as minimizing energy usage, reducing interference, and cost. These algorithms are validated with emulated environments based MIT supercloud and EU's energy usage patterns. Moreover, we model ILP as multi-objective problems for optimizing similar objectives defined before, while allocating and load balancing resources across the continuum. To optimize energy efficiency and sustainability of COGNIT, a formal model is designed, and the integration of the Smart Grid and Cloud Continuum is initially validated through model-based simulation. To transform the model into a simulation, mathematical models leveraging container-level granularity, idle and busy power, and carbon intensity of the Smart Grid are presented. These efforts contributed significantly to achieving more efficient, energy-aware cloud-edge infrastructures to realize overall system performance, and enabling more effective, intelligent orchestration strategies within the COGNIT framework.

In the next cycle, the *Cloud-Edge Manager* will be extended to provide the capabilities to dynamically create new edge locations automatically by implementing a Provider Catalogue and exposing an API for Edge Cluster Provisioning. On the monitoring side, metrics related to the latency with respect to the Device Client will be stored in order to be used by the AI-Enabled Orchestrator to anticipate the creation of Serverless Runtime and/or Edge Cluster; furthermore, methods for intrusion and anomaly detection of Edge Clusters and Serverless Runtimes will be implemented. Finally, the Plan Executor will be extended to execute plans produced by the AI-Enabled Orchestrator to perform operations at the infrastructure level (i.e., creating, scaling and deleting Edge Clusters).

The development and integration of *AI-Enabled Orchestrator* will be extended to NSGA-III and approximate gradient evolutionary algorithms to balance the conflicting objectives for resource optimization while maintaining expected performance and increasing green energy usage. The AI-Enabled Orchestrator will utilise the extended metrics to predict the number of serverless runtimes and new edge clusters that will be required for optimal and cost-effective resource usage. However, the next cycle will explore scalability and carry out multidimensional experiments by deploying and managing dynamic workloads across the

Cloud-Edge Continuum systems, which will illustrate the scalability, energy-efficiency, sustainability, and ensuring the expected performance within simulation environments.

References

[1] Srinivas, N., Deb, K. (1994). Multi Objective optimization using nondominated sorting in genetic algorithms. Evol. Comput. 2(3), 221–248.

[2] Deb, K., Pratap, A., Agarwal, S., Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: Nsga-ii. IEEE Trans. Evol. Comput. 6, 182–197.

[3] Zeigler, B. P., Muzy, A., & Kofman, E. (2018), Theory of Modeling and Simulation: Discrete Event & Iterative System Computational Foundations, 3rd ed. USA: Academic Press, Inc.

[4] ContinuumSim GitHub, https://github.com/SovereignEdgeEU-COGNIT/ContinuumSim (accessed: 17/12/2024)

[5] Electricity Maps, https://www.electricitymaps.com (accessed: 3/12/2024)

[6] Samsi, S. et al., The MIT SuperCloud Dataset, CoRR, vol. abs/2108.02037, 2021. [Online]. Available: https://arxiv.org/abs/2108.02037

[7] Bauer, A. et al. (2024). The globus compute dataset: An open function-as-a-service dataset from the edge to the cloud. Future Generation Computer Systems 153, 558-574.

[8] Chang, Y., Bouzarkouna, Z. & Devegowda, D. (2015). Multi-objective optimization for rapid and robust optimal oilfield development under geological uncertainty. Computational Geosciences 19, 933-950.

[9] Zitzler, E., Laumanns, M., & Thiele, L. (2001). SPEA2: Improving the strength Pareto evolutionary algorithm. TIK report, 103.

[10] Liu, X., & Zhang, D. (2019). An improved SPEA2 algorithm with local search for multi-objective investment decision-making. Applied Sciences, 9(8), 1675.

[11] Nguyen, C., Bhuyan, M., & Elmroth, E. (2024). Enhancing Machine Learning Performance in Dynamic Cloud Environments with Auto-Adaptive Models, in Proc. IEEE International Conference on Cloud Computing Technology and Science (CloudCom), Abu Dhabi, United Arab Emirates, 184-191.

[12] Bifet, A., & Gavalda, R. (2007). Learning from time-changing data with adaptive windowing," in Proc. SIAM international conference on data mining, in Proc. SIAM international conference on data mining, 443–448.

[13] Kadwe, Y., & Suryawanshi, V. (2015). A review on concept drift. Iosr J. Comput. Eng, 17(1), 20-26.

[14] Dempster, A., Petitjean, F., & Webb, G. I. (2020). ROCKET: exceptionally fast and accurate time series classification using random convolutional kernels. Data Mining and Knowledge Discovery, 34(5), 1454-1495.

[15] Eldin, A. A. et al. (2014). How will your workload look like in 6 years? analyzing wikimedia's workload, in Proc. IEEE international conference on cloud engineering (pp. 349-354). IEEE.

[16] Urdaneta, G., Pierre, G., & Van Steen, M. (2009). Wikipedia workload analysis for decentralized hosting. Computer Networks, 53(11), 1830-1845.

[17] Graves, A., & Graves, A. (2012). Long short-term memory. Supervised sequence labelling with recurrent neural networks, 37-45.

[18] Ali-Eldin, A., Seleznjev, O., Sjöstedt-de Luna, S., Tordsson, J., & Elmroth, E. (2014). Measuring cloud workload burstiness, in Proc. IEEE/ACM 7th International Conference on Utility and Cloud Computing (pp. 566-572). IEEE.

[19] Reiss, C., Wilkes, J., & Hellerstein, J. L. (2011). Google cluster-usage traces: format+ schema. Google Inc., White Paper, 1, 1-14.

[20] Hieu, N. T., Di Francesco, M., & Ylä-Jääski, A. (2017). Virtual machine consolidation with multiple usage prediction for energy-efficient cloud data centers. IEEE Transactions on Services Computing, 13(1), 186-199.

[21] Tsuruoka, Y., Tsujii, J. I., & Ananiadou, S. (2009). Stochastic gradient descent training for l1-regularized log-linear models with cumulative penalty, in Proc. Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP (pp. 477-485).

[22] Zhang, T. (2004). Solving large scale linear prediction problems using stochastic gradient descent algorithms, in Proc. twenty-first international conference on Machine learning (p. 116).

[23] Karal, O. (2023). Robust and optimal epsilon-insensitive Kernel-based regression for general noise models. Engineering Applications of Artificial Intelligence, 120, 105841.

[24] Gonçalves Jr, P. M., de Carvalho Santos, S. G., Barros, R. S., & Vieira, D. C. (2014). A comparative study on concept drift detectors. Expert Systems with Applications, 41(18), 8144-8156.

[25] Campello, R. J., Moulavi, D., Zimek, A., & Sander, J. (2015). Hierarchical density estimates for data clustering, visualization, and outlier detection. ACM Transactions on Knowledge Discovery from Data (TKDD), 10(1), 1-51.

[26] Salvador, S., & Chan, P. (2007). Toward accurate dynamic time warping in linear time and space. Intelligent data analysis, 11(5), 561-580.

[27] Kidane, L., Townend, P., Metsch, T., & Elmroth, E. (2022). When and how to retrain machine learning-based cloud management systems, in Proc. IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW) (pp. 688-698). IEEE.

[28] Shayesteh, B., Fu, C., Ebrahimzadeh, A., & Glitho, R. (2021). Auto-adaptive fault prediction system for edge cloud environments in the presence of concept drift, in Proc. International Conference on Cloud Engineering (IC2E) (pp. 217-223), IEEE.

[29] Masanet, E., Shehabi, A., Lei, N., Smith, S., & Koomey, J. (2020). Recalibrating global data center energy-use estimates. Science, 367(6481), 984-986.

[30] Patel, Y. S., Townend, P., & Östberg, P. O. (2023). Formal Models for the Energy-Aware Cloud-Edge Computing Continuum: Analysis and Challenges, in Proc. International Conference on Service-Oriented System Engineering (SOSE) (pp. 48-59). IEEE.

[31] Shahrad, M. et al. (2020). Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider, in Proc. USENIX annual technical conference (USENIX ATC 20) (pp. 205-218).

[32] Azure public dataset, https://github.com/Azure/AzurePublicDataset

[33] Planetlab trace, https://github.com/beloglazov/planetlab-workload-traces

[34] Shen, S., Van Beek, V., & Iosup, A. (2015). Statistical characterization of business-critical workloads hosted in cloud datacenters, in Proc. 15th IEEE/ACM international symposium on cluster, cloud and grid computing (pp. 465-474). IEEE.

[35] Alibaba trace, https://github.com/alibaba/clusterdata