Distributed Fog Platform for Weather Forecasting at the Edge

ALEXANDRE BRUNO GEHL BAPTISTA RODRIGUES DA COSTA, Instituto Superior Técnico, Portugal

With the growing development and adoption of Internet of Things (IoT) technologies, it becomes necessary to create the conditions for these systems to operate effectively and in a scalable manner. The amount of data constantly generated and transferred by such devices is extremely high, and many of these systems may support services that operate in real time and with a strong sense of urgency. It is therefore essential to find solutions and approaches that enable and optimize their operation, capable of handling these large data flows in terms of storage, processing, and transmission. Edge computing, a paradigm that shifts data processing closer to the user, addresses some of these problems by mainly reducing response latencies and also offering advantages in terms of privacy. If an IoT device is capable of performing all or part of the required processing locally, it immediately prevents all data from having to leave the local network. Nevertheless, even if these devices can independently perform the necessary tasks, a mechanism is still required to distribute such tasks and organize the computational effort involved. In this context, the fog computing paradigm emerges, proposing an intermediate layer located above the local edge nodes, capable of coordinating distributed processing in an organized manner. By bringing computational intelligence closer to the user, this architecture not only enables efficient segmentation and distribution of tasks but also reinforces the relevance of distributed computing in scenarios with critical performance requirements. This work discusses the current relevance and applicability of these paradigms, proposing and developing an architecture and platform designed to demonstrate their potential. The implemented platform focuses on the prediction of meteorological variables, using tools oriented toward big data and distributed processing, such as Apache Spark and Hadoop, to support the cluster.

Additional Key Words and Phrases: Edge Computing; Fog Architecture; Internet of Things; Smart Cities; Distributed Computing

ACM Reference Format:
Alexandre Bruno Gehl Baptista Rodrigues da Costa. 2025. Distributed Fog Platform for Weather Forecasting at the Edge. 1, 1 (November 2025), 10 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Cloud computing has established itself as one of the most popular and essential paradigms in recent technological evolution, mainly due to its scalability and the wide range of available services capable of meeting diverse needs. These services, whether physical or virtual resources, allow organizations and users to leverage such resources without the need to maintain them within their own infrastructures.

Despite the success and importance of these technologies and this computing model, the cloud approach does not seem to establish the best paradigm for dealing with massive amounts of data transferred in real time, since such transfers imply considerable delays, which may potentially scale depending on the context. Especially since the emergence of Internet of Things (IoT) technologies — which generate these enormous amounts of real-time data and often require real-time decisions — efforts have been made to bring processing and storage closer to the user [25]. Due to these limitations, cloud computing faces difficulties and is unable, by itself, to meet the growing demands imposed by IoT technologies [7], particularly in critical service domains such as healthcare systems, vehicular networks, environmental monitoring, or smart cities. Moreover, the increasing need for real-time processing capacity and latency minimization makes these architectures inefficient in handling scenarios that require near-instantaneous responses and decisions.

Edge computing, closer to the user, and fog computing, which provides essential resources at the boundary between the local edge network and the cloud, significantly contribute to addressing these challenges. The edge paradigm implies a model in which data processing occurs near the user or the data source, focusing on the decentralization of computational resources and allowing devices that traditionally would not serve such a purpose to now contribute computationally *in loco* to the system. The fog component, positioned between the edge and the cloud, facilitates resource transfer and optimization for data processing, intelligently distributing workloads while considering network conditions and the state and capabilities of the available devices. By addressing critical issues such as latency, scalability, and bandwidth [26], these paradigms not only increase the operational efficiency of a system within such environments but also enable the emergence of new applications and services that previously struggled with the mentioned challenges.

Building upon these conclusions, a platform has been developed, which we shall refer to as "Weather@Edge", combining both the edge and fog approaches. This platform is capable of performing weather forecasting by leveraging not only the resources of the user's own devices but also those of nearby devices that are capable of contributing. These devices are orchestrated by a superior fog layer, which organizes and distributes the cooperative effort. We will therefore explore how these architectures operate together in a controlled environment, in this case, focusing on the prediction of hourly meteorological variables (temperature, humidity, and precipitation) for a short-term interval.

2 Related Work

With the advances in IoT and machine learning technologies, there arises the need to explore different approaches that enable understanding the most efficient ways to handle them computationally and to combine their capabilities.

2.1 Smart Cities @ Edge

The previously mentioned continuous development of IoT technologies further strengthens and sparks interest in the concept of smart cities. The potential inherent in traditional technologies is immense, and the examples are numerous:

Author's Contact Information: Alexandre Bruno Gehl Baptista Rodrigues da Costa, Instituto Superior Técnico, Lisbon, Portugal, alexandre.bruno.costa@tecnico.ulisboa.pt.

ACM XXXX-XXXX/2025/11-ART
https://doi.org/10.1145/nnnnnnn.nnnnnnn

- Urban areas filled with networks of intelligent infrastructures that communicate with each other, collecting and sharing information, data, and patterns — everything that can provide value to the cities and their inhabitants [18].
- Traffic lights capable of recognizing traffic patterns or identifying vehicles through video streams, making autonomous decisions, and even storing the data obtained or the decisions made [20].
- Local data of various types (for example, temperature, humidity, or probability of precipitation at a given location) made publicly available for citizens/users to query locally, for instance, via API requests, without the need to rely on cloud infrastructures or central servers (already exploring the concept of information and communication technologies) [16].

These scenarios are just examples of how everyday technologies can "gain intelligence" and how a city can transform into a smart city — offering new and varied functionalities or improving existing services and technologies, maximizing the potential of the surrounding infrastructures. The gains in efficiency for complex real-time services, energy savings [4], reductions in network congestion, and overall benefits for citizens could be substantial. The objectives of smart cities align with improving citizens' quality of life and enhancing urban efficiency.

Nam and Pardo [19] highlight that technologies related to IoT, big data, and artificial intelligence are an important step that enables the interconnection of different urban services and optimizes the distribution of resources.

Obviously, such benefits also bring new challenges. The amount of data generated by these technologies is enormous, and much of it has sufficient relevance to justify and require storage. Furthermore, these are technologies and devices that communicate and provide services demanding time-critical and urgent responses and decisions. As Apat et al. [3] point out, the cloud computing model, despite offering a vast pool of physical and virtual resources, does not yield ideal latency results for IoT users due to the extensive flow of data traffic and potential network bandwidth limitations caused by excessive request concurrency at a given moment [3]. How, then, can these challenges be addressed?

By moving both data processing and storage as close as possible to the devices generating the data. This is where the "@ Edge" component, as referred to in the section title, comes into play. Studies show that edge computing is the ideal solution when the goals are to reduce latency and improve data privacy [2], thanks to the shorter path data must travel when processed locally.

2.2 Fog Architecture

Despite the benefits already discussed and those that edge computing can offer, it must also be considered that, in reality, this type of architecture can hinder a system's scalability and flexibility — precisely the opposite of what cloud-based architectures provide.

A hybrid solution, therefore, emerges, capable of bringing such 'elasticity' to the network edge and addressing the identified challenges [26]. This solution leverages both edge service components and cloud services: the *fog* architecture. The fog layer, which can be defined as the layer that intermediates between local nodes and the cloud, at a one-hop distance from the nodes, combines the strengths of both approaches, offering low latency while being efficient in resource and bandwidth management [2]. Edge devices generate and process data in real time, while the orchestration and coordination of nodes are handled by this layer. Typically, this type of architecture also provides offloading capabilities, distributing and assigning tasks based, for example, on computational capacity, battery level, or proximity between devices.

Bebortta et al. [5] designed a framework for optimizing big data processing in heterogeneous IoT networks that "enables data collection, feature selection, predictive models, and data visualization," and it essentially relies on a structure such as the one illustrated in Figure 1.

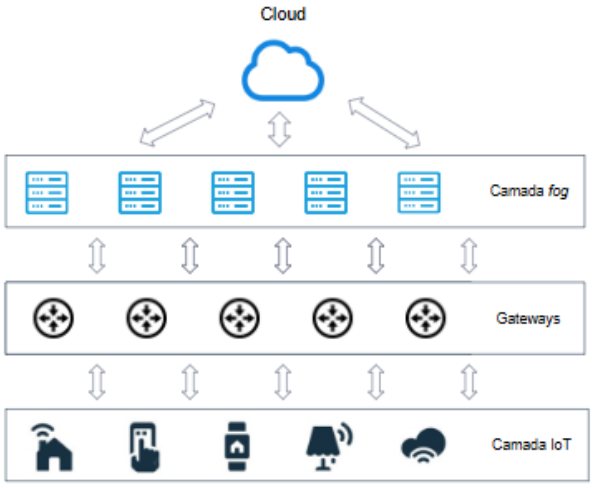

Fig. 1. Fog Architecture

In terms of security and privacy, the advantages can also be numerous: data that can now be processed locally no longer need to be transferred beyond the local network, thereby reducing the risks associated with network data traffic and potential cloud vulnerabilities. Regarding scalability, this distributed architecture facilitates the addition of nodes to a system in cases where increased processing or storage capacity is required, making it a very appealing model for large-scale sensor networks, such as in the context of smart cities, where the number of local devices and data production may increase constantly.

2.3 Distributed Computing

Distributed computing allows systems to divide a given task into smaller segments that can then be distributed across different devices capable of executing them. It therefore becomes a highly viable solution for contexts that demand constant or high computational effort, such as those found in the smart city environments we have been discussing [14].

By granting the architecture the ability to orchestrate the constituent nodes of the network through this intermediate layer, if

the tasks required for the system's operation are further partitioned and distributed among different nodes capable of contributing, then in addition to the proximity established between the system and the user, we also achieve full decentralization and, consequently, optimize its efficiency. Since there are always at least two elements in the system when an execution takes place — the user node and its corresponding fog-layer "orchestrator" LS — it means that there will always be parallelism and segmentation of computations. To support this division of computations, a framework such as Apache Hadoop can be employed — an open-source library widely used in such large-scale data processing contexts [23, 27], also extended for voluntary computing [9, 10]. Hadoop, through its distributed file system, HDFS (*Hadoop Distributed File System*), enables distributed storage of datasets among the network nodes [23]. Beyond its functionalities related to distributed data storage and management, Hadoop also offers distributed processing capabilities, particularly the well-known MapReduce model, which allows the decomposition of processing tasks into smaller sub-tasks that are executed in parallel by the network nodes. It is therefore important to distinguish the two fundamental stages of this programming model: *Map*, where data is processed and transformed, and *Reduce*, where the intermediate results are combined to produce the final output. The user defines a map function to be executed by the different nodes according to an input (key, value) (K, v), which will also generate an intermediate set of (K, v) pairs [11, 13], to be subsequently collected and aggregated.

Delving deeper, in the initial Map phase, the input dataset is partitioned into blocks, typically equal or proportional to the number of available nodes. Each node executes a map function that transforms the input data into (key, value) pairs, which will be used in a later aggregation phase. In the Reduce phase, the (K, v) pairs are grouped or aggregated, taking into account the pairs sent by each node. This model also presents some limitations, particularly regarding latency in iterative processes such as machine learning models or others that require multiple iterations, reads, and storage operations over the initial and intermediate data. Therefore, this methodology does not appear to be ideal for time series forecasting [1]. In addition to Hadoop, another fundamental and widely used tool for processing large volumes of real-time data (especially in iterative or streaming applications) is Apache Spark [24], an open-source distributed computing framework that provides an interface for executing applications in clusters, offering full parallelism. It overcomes some of the limitations of MapReduce and achieves better performance and faster results [24] through an in-memory processing model, making it suitable for applications that require low latency and high processing speeds, such as machine learning algorithms or real-time analytics. This in-memory processing approach reduces the need for disk access, allowing data to be stored in RAM during operations, thereby significantly decreasing execution times.

2.4 Weather Forecasting Systems

Time series play a crucial role in several practical domains, such as meteorology and economics [6]. The forecasting of such time series is primarily based on the analysis of historical data, for example, a dataset composed of meteorological observations, and, based on the observed patterns, aims to predict the continuation of that series over a future interval. In contrast, NWP (*Numerical Weather Prediction*) models, which depend heavily on computational power, rely on current atmospheric data, using them as input to produce an analysis and forecast future weather conditions [6]. Traditional statistical modeling systems assume that a time series follows certain patterns such as linearity, stationarity, or seasonality. A popular and traditional statistical modeling and time series forecasting algorithm is the Auto Regressive Integrated Moving Average model, or ARIMA [21]. It is an easy model to adopt because it does not require as much computational effort as Machine and Deep Learning models, such as Convolutional Neural Networks (CNN) or Recurrent Neural Networks (RNN). This can be particularly useful depending on the type of devices at the edge, enabling, for example, even the sensors responsible for measurements to have the capacity to execute such models [17]. The model is based on the assumption that *the time series is linear and follows a statistical distribution, such as a normal distribution* [6]. Even more appropriate for forecasting series identified as seasonal is the Seasonal Auto Regressive Integrated Moving Average, or SARIMA, which is formed by adding seasonal parameters to the input of the ARIMA model:

$$\text{ARIMA (p, d, q) (P, D, Q)}_m$$

Where (p, d, q) are the non-seasonal parameters, and (P, D, Q) are the seasonal ones, with m representing the periodicity of the identified seasonality. The definition of the parameters (p, d, q) and (P, D, Q) will be explored and demonstrated in the following section. Regarding the period of seasonality, if, for instance, we have monthly records or data (such as temperature or precipitation) and they display annual patterns, then $m = 12$.

3 Solution

Following reflection and analysis on this smart cities scenario at the edge, and aiming to identify solutions that contribute to the development of this concept, a possible approach was outlined for the development of a platform that enables the user—through the devices currently available to them and by leveraging their computational resources, as well as those of nearby devices capable of assisting—to perform weather forecasts for that location for the coming days. The computational work required by these models will be carried out on historical data related to the specific location; therefore, it is necessary to obtain a segment of essential data for the forecast (either through API requests or available libraries that allow such access — for example, Meteostat provides a Python library that allows access to a vast database of historical data for numerous locations). Regardless of how the fog layer performs this data collection, it may also periodically update a shared file system to which the edge layer nodes will have access, thus eliminating the need for data traffic between these layers.

3.1 Functional Architecture

We can distinguish two clearly distinct roles within our structure: the fog layer nodes, which act as masters/orchestrators, and the edge layer nodes, which in turn act as workers/slaves. It is therefore necessary to plan the responsibilities and tasks of each, as well as how their interaction will take place. We can outline the following possible 'responsibilities' associated with the fog nodes:

- Receive/obtain data
- Store and process data locally
- Orchestrate task offloading, for example, delegating parameterization or partial forecasts to the corresponding nodes
- Respond to requests from edge nodes for weather forecasts
- Act as a decision point for generating alerts (for example, when temperature or wind speed exceed certain thresholds)

In our case, this fog layer may consist of the Access Points (APs) available and associated with the respective device. Each AP performs periodic requests to keep the necessary data updated and stored close to the nodes. Whenever a new device (node) joins an AP's cluster, it must notify it. We can therefore define that all nodes within the cluster that are available to contribute computational resources must periodically send messages (or heartbeats) to their respective AP.

The local nodes of the edge layer, on the other hand, are responsible for periodically informing their availability (by broadcasting a heartbeat to the network) to the corresponding upper-layer node. Thus, the fog node can update its inventory of workers and distribute the required tasks among them, whether executing parameterization functions or running forecasting models. Regardless of the available worker nodes in the cluster, the master node will also operate simultaneously as a worker. In this way, it can perform the necessary periodic parameterization of the forecasting model and will be ready to execute that model quickly whenever requested by a user. This design choice stems from the long execution time of such parameterization functions, which would otherwise result in prolonged waits for simple forecasts of individual variables—something that contradicts the efficiency goals of this platform architecture.

3.2 Forecasting Model

Based on the research previously conducted, we began by implementing the ARIMA model on local nodes, followed by testing to evaluate its efficiency.

The first step was the definition of the parameter values to be used in this model: 'p, d, q'. A *dataset* of historical temperature records in Lisbon for the year 2023, obtained from the Meteostat database, was used. Starting with parameter 'p' (or *lag order*), that is, the order of the autoregressive model, Partial Correlation (PACF) tests were conducted to determine the number of significant lags until the correlation stabilized at 0. The results can be seen in Figure 2. In the resulting graph, approximately 25 significant lags can be identified; therefore, this will be our *lag order*.

Next, we analyzed parameter 'd', referring to the degree of differencing in the series. One of the ways to determine this parameter is by performing the Augmented Dickey-Fuller (ADF) test, in order to verify whether the series is stationary or not. After executing the test, the following results were obtained:

ADF Statistic: -3.4494898186886993
p-value: 0.00938310848841729

Since the *p-value* is lower than 0.05 and the ADF statistic is considerably negative, we can conclude that the series is non-stationary, meaning that differencing is not required. Thus, we will define the parameter as d = 0.

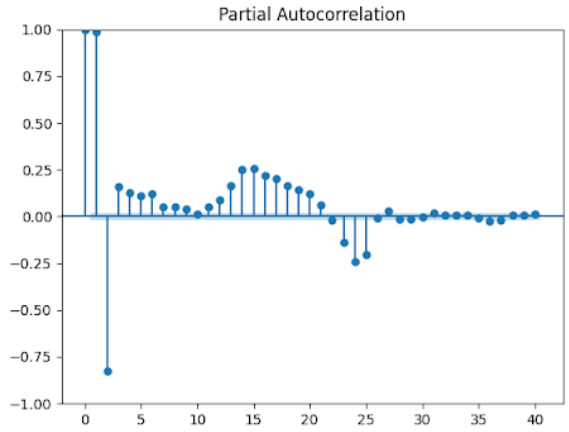

Fig. 2. Results of the PACF test

Proceeding to the third and final parameter required for the execution of an ARIMA model, 'q', or the order of the moving average model, an ACF (Autocorrelation Function) test was performed: By

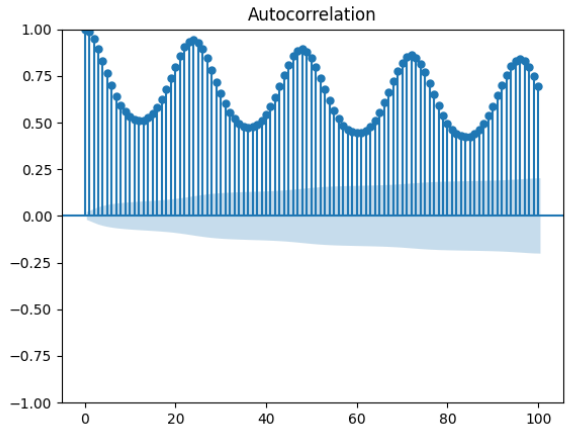

Fig. 3. Results of the ACF test

analyzing the resulting graph (Figure 3), which essentially consists of identifying the number of lags required until the oscillation decays to zero, we can observe that, even when extending the scope of the test up to a high number of lags (100), the ACF value oscillates between 1 and 0.5 without ever converging to zero. What does this result mean, and how can this obstacle be addressed? The fact that our ACF graph never converges to zero within a reasonable number of lags is a strong indicator of seasonality in our dataset. This means that a simple (non-seasonal) ARIMA model may not correctly capture the dynamics of the data. Considering this, we will investigate the seasonal ARIMA model, or SARIMA, comparing results between the two models, with parameters optimized for SARIMA through

Table 1. Errors of ARIMA vs SARIMA models

Model	MSE	RMSE	MAE
ARIMA	2.0258	1.4233	1.2107
SARIMA	1.8699	1.3674	1.1585

parameterization functions such as grid search, using the following possible intervals (assuming the seasonal parameter S = 24, due to the daily seasonal pattern of the series with a period of 24, reflecting hourly periodicity): paramGrid = {'p': [0, 1, 2], 'd': [0, 1], 'q': [0, 1, 2], 'P': [0, 1, 2], 'D': [0, 1], 'Q': [0, 1, 2]}. The ARIMA parameters (data, order=(25, 0, 0)) were selected earlier, and several evaluation metrics for forecasting accuracy—explained in more detail in Section 5—were used.

Our *grid search* function selected the following optimal parameters for the SARIMA model executed on the provided dataset: (order=(1, 1, 1), seasonalOrder=(1, 1, 1, 24)).

Comparing the results between the ARIMA and SARIMA models on the same dataset, as shown in table 1, and using the MSE metric (which measures the mean squared error—the difference between the forecast and the actual value), it can be concluded that the SARIMA model achieved slightly better performance. This conclusion is also supported by the RMSE values, expressed in the same units as the original data (that is, the root mean square error in temperature forecasting for the ARIMA model was approximately 1.42 °C, while for the SARIMA model it was around 1.37 °C—a slightly better fit to the data, although the difference is small). The mean absolute error (MAE) also shows a higher average error for the ARIMA model.

After evaluating the forecasting models and considering factors such as model simplicity—crucial to maximize the feasibility of execution on edge devices close to the data sources (e.g., temperature, humidity, or precipitation sensors)—it was concluded that the most suitable option for performing our weather forecasts is the SARIMA model.

4 Solution Development

To make this proposal functional, a set of open-source tools widely used in big data and distributed computing contexts was employed. Apache Hadoop file storage was used as the foundation for distributed data storage, through its Hadoop Distributed File System (HDFS), which allows simultaneous and fault-tolerant local access by multiple nodes. Apache Spark, on the other hand, was the technology chosen for in-memory processing of large volumes of data in a distributed manner, benefiting from its efficiency in executing parallel tasks and its flexibility to integrate machine learning and time series models. The following sections of this chapter describe, in a structured manner, the steps involved in creating the platform's execution environment, including the installation and configuration of each component, the definition of the cluster architecture, and the performance of functionality tests. The main technical challenges encountered during the process and the strategies adopted to overcome them are also discussed, contributing to the construction of a robust, modular solution ready to evolve according to new requirements and application contexts.

4.1 Cluster Configuration

Widely recognized components from the big data ecosystem were used, namely Apache Hadoop and Apache Spark. Hadoop, with its distributed file system HDFS, ensures data access and replication across multiple nodes. Spark, on the other hand, is responsible for the efficient processing of data, leveraging an in-memory execution model and native support for parallel tasks and predictive algorithms. Although Apache Hadoop includes its own distributed processing engine, also based on the MapReduce paradigm, it was decided to use Hadoop exclusively as a distributed storage system (HDFS), delegating data processing tasks to Apache Spark.

4.1.1 Apache Hadoop. To begin the creation and configuration of the cluster environment, the Apache Hadoop framework mentioned earlier was installed on an Ubuntu machine that would serve as the Master in later stages, with the goal of leveraging the capabilities of its distributed file system, HDFS. This system, as previously noted, enables shared access to datasets stored in parallel, thereby facilitating the coordinated operation of the network's worker nodes.

After creating the 'hadoop' user, a passwordless SSH public key was generated and stored as an authorized_key in the SSH directory, followed by testing SSH access to the localhost. These keys must also be distributed to the cluster's worker nodes.

Once the required software (JDK, Hadoop, and SSH) was installed on the machine and the environment variables were configured in the bashrc file, the next step was to configure the cluster's XML property files. The Hadoop cluster was then started, beginning with the NameNode and the DataNode.

To verify the correct creation of the cluster, access was made to the NameNode UI at http://localhost:9870, the DataNodes accessible through the same address, and the YARN ResourceManager located at http://localhost:8088.

4.1.2 Apache Spark. For the installation of Apache Spark (version 3.5.5), it was first necessary to ensure the installation of its main dependencies, namely Scala and Git. These components are essential, as Spark is developed in Scala, and Git facilitates access to project-related versions and repositories.

Once this step was completed, the installation of Apache Spark itself was carried out. After extracting and placing the Spark directory in the system, the necessary environment variables were configured, such as SPARK_HOME, along with updating the PATH variable, in order to allow the execution of Spark commands from any location within the system.

With the installation completed, the configuration of the Spark cluster was initiated using the scripts provided by the framework. The master node was started through the command `'start-master.sh'`, becoming accessible through the web interface at the master's IP address, where it is possible to monitor the cluster status, active nodes, and running tasks.

To enable HDFS access for the worker nodes, the SSH keys must be distributed among the cluster elements. In other words, once the

master node receives the first heartbeat from a new node in the cluster, it must execute the command `'ssh-copy-id new_node'`.

Next, two worker nodes associated with the master were launched using the command `'start-worker.sh spark://master:7077'`, where 'master' corresponds to the hostname or IP address of the master node. This connection allows the workers to register themselves and become part of the cluster, making their computational resources available for submitted tasks.

4.2 Application

To test the efficiency assumptions of our architecture, it is now necessary to run an application based on the execution of simple forecasting models within our cluster, in order to verify that, indeed, beyond the reduction of data traffic to external networks, there is also an increase in the efficiency of our distributed platform when compared to an execution that does not involve task and data distribution or parallel processing.

In a weather forecasting system that depends on and is based on a forecasting model—linear in this case—the first challenge to be faced is the definition of its optimal parameters, based on a given dataset, as previously discussed and analyzed. Since the SARIMA model proved to be the best option in this context, and knowing that the selection of optimal parameters may vary depending on the training dataset, it was decided that a periodic (e.g., daily) execution of a parameterization function for this model would be required. Given the complexity and computational load of this type of task, it immediately becomes an ideal 'target' for applying our distributed and parallel computing strategies, in order to optimize this step.

Therefore, a range of 'normal' values for the parameters in question was first defined — namely SARIMA (p, d, q) (P, D, Q, M) — which were analyzed individually in the Solution chapter. Having this parameter grid defined, how could we then segment this task into subtasks that could be distributed among the available nodes? The solution found was to distribute all possible parameter combinations among the edge nodes of the network using the `sparkContext.parallelize()` function.

Regarding our forecasting model, the chosen approach was the SARIMA model, as already justified earlier (Section 4, Solution Proposal). Nevertheless, the same question that arose during the parameterization component reappears here: how should we segment the execution of our model? The final answer was to divide the number of variables (or columns) requested by the user among the available nodes.

5 Testing and Results Discussion

This chapter first presents the evaluation methodologies of our system and analyzes the results obtained from executing the various components of the developed platform — particularly the parameterization operations, the weather forecasting functions, and the impact of cluster scalability on the overall system performance. The experiments carried out aimed primarily to assess the platform's behavior under different configurations, both in single-node and multi-node environments, analyzing how the increase in the number of nodes influences computational efficiency, latency, and the responsiveness of the architecture.

Additionally, forecasting functions were executed with different parameter combinations and workload distributions in order to identify potential bottlenecks in the system, as well as situations where a real performance gain is achieved through parallelism and node collaboration.

The master node, configured with 6 GB of RAM, was responsible for coordinating the cluster and executing the Spark driver. The virtual machines comprising the cluster, running Linux (Ubuntu 22.04) and each equipped with 2 vCPUs, were interconnected through a virtual network in bridge mode, allowing direct communication between them and simulating a local area network (LAN) environment.

Regarding the system's fault model, and considering the dynamic nature of the cluster, in which nodes may intermittently join or leave (churn), it is important to understand how the system reacts to such situations. At the data processing level, in the event of a node failure or sudden unavailability, the Spark environment—providing fault tolerance—automatically redistributes the pending tasks to the remaining available nodes, ensuring the continuity of execution. In terms of storage, HDFS guarantees the recovery and replication of data blocks in case of a DataNode failure, ensuring that the loss of a node does not compromise the integrity of the stored data.

5.1 Evaluation Metrics

In the platforms discussed so far, which include predictive models at the edge, it also becomes necessary to assess the quality of these models, as demonstrated in our proposed solution — not only to ensure that the most reliable result is delivered to the user, but also to verify that the model is optimized and functioning correctly. For this purpose, a given dataset can be divided into training and testing subsets, comparing the results produced by the model based on the training dataset with the actual results contained in the testing subset. How, then, can these results be evaluated beyond simple visual observations?

5.1.1 Forecasting Model Evaluation. Some of the most commonly used evaluation metrics for forecasting algorithms include the *Root Mean Square Error* (RMSE), the *Mean Absolute Percentage Error* (MAPE), and the coefficient of determination (R^2) [8]. The *Mean Square Error* (MSE) measures the average of the squared errors (the difference between the forecast and the actual value). The lower the MSE value, the better the model. The RMSE is simply the square root of the MSE and is more easily interpretable in application contexts, as it is expressed in the same units as the original data. Lower RMSE values indicate a more accurate model.

The *Mean Absolute Error* (MAE) measures the average of absolute errors — that is, the mean of the absolute distances between the forecast and the actual value. It is also expressed in the same units as the original data. Although this facilitates interpretation, it makes the metric dependent on the data scale, which can hinder comparisons between variables. The MAPE, on the other hand, expresses the error in relative terms, as a percentage [8].

The *R-squared* (R^2), or coefficient of determination, measures the proportion of variance of a dependent variable explained by the model. However, it is not entirely reliable for non-linear models and

time series. It ranges between 0 and 1, and the closer the value is to 1, the more accurate the forecast is considered to be.

5.1.2 Parallel Computing Component Evaluation. The aforementioned metrics allow us to interpret the efficiency of our forecasting models. What remains to be evaluated is another important component of our system: its efficiency in distributing and executing the necessary tasks across the existing nodes — that is, its ability to relieve load and make the platform more efficient through distribution orchestrated by the fog layer of APs/LSs. It is necessary to demonstrate that efficiency scales (execution times decrease) as a function of factors such as the increase in the number of nodes in the network, for example.

To achieve this, it is necessary to create a cluster with a variable number of nodes, something that can be accomplished using software such as Docker. Docker is an open-source platform that uses containers to isolate applications and execution environments, ensuring that they behave consistently and independently of the underlying infrastructure [23]. Each node is represented by a container configured to process the tasks assigned by the fog layer, which is responsible for orchestrating the clusters. Thus, Docker provides the necessary conditions to simulate clusters of independent nodes orchestrated by master nodes (in this case, the elements of the fog layer), each with individual and customizable characteristics and conditions. This setup allows testing of various practical scenarios, such as variations in the number of active nodes, differing network conditions and latencies, or variations in the computational load assigned to each node according to these parameters. In this way, we can assess the capability of our platform to handle different contexts and challenges.

The tests will initially consist of validating the basic functionality of our orchestration layer, that is, its ability to correctly distribute the computational load among the nodes existing in the cluster. By comparing the execution time between a single-node run — requiring one user to complete all tasks individually — and a multi-node execution (for instance, using two nodes and assuming that the combined computational capacity of those two nodes is greater than that of the previously tested single-node), we can immediately validate the platform's ability to reduce execution times and latency by distributing tasks among different contributing nodes, thereby leveraging a distributed computing architecture.

5.1.3 Stress Tests. Finally, it is also important to evaluate the system's behavior as the volume of data required for the tasks executed by the nodes scales up, in order to understand how the system handles increasing data loads. By keeping the number of available nodes low (for instance, a single node or just two), the processing load can be successively increased to assess how the system reacts until it reaches its operational limits. These results will also make it possible to detect potential *bottlenecks* and recurring patterns in our platform that can be used to refine and improve it.

5.2 Forecasting Model Parameterization

This section presents the results obtained during the parameterization phase of the forecasting model used in the platform. The main objective of this stage was to test different combinations of parameters and both single-node and distributed executions, in order to identify those parameter sets that best balance forecasting accuracy and computational performance within the proposed architecture, as well as to verify that the platform's efficiency objectives are met.

Table 2. Model parameterization times with grid 1

Nº of nodes	24 combinations grid	36 combinations grid
1	84.89 s	200.52 s
1	82.79 s	208.79 s
1	84.79 s	225.18 s
2	79.55 s	181.88 s
2	78.86 s	173.52 s
2	79.73 s	167.28 s

Executions were carried out in different configurations using the variable *temp* (temperature), varying not only the model's internal parameters but also the number of active nodes in the cluster, with the goal of evaluating the impact of task distribution on the system's overall performance. Initially, a variation of the `auto_sarima()` function was executed in single-node mode, followed by the same experiment using two nodes, with a `.csv` file of approximately 11.1 KB and a parameter grid of (0, 2) for 'q', 'P', and 'Q', and (0, 3) for 'p'. The results can be observed in Table 2. All these executions identified ARIMA: (2, 1, 1) Seasonal: (0, 1, 1, 24) as the best parameter set for the model in this context.

As can be seen, in addition to consistently identifying the same model as the most suitable, the average execution time of the function dropped from 84.16 seconds to 79.38 seconds, representing an improvement of approximately 5% (4.78 seconds). And what happens if we expand the grid's scope, increasing, for example, the range of parameter 'q' to (0, 3)? How will our cluster respond?

Despite the expanded grid with new combinations, the parameterization function continued to identify the same set of parameters as those that best fit our forecasting model, reinforcing the validity of the initially chosen grid. It is easy to observe considerable gains in terms of system efficiency. As expected, the greater the computational complexity and number of calculations to be performed, the more significant the reduction in execution times achieved through parallel computations over the data. With the first and smaller grid used (encompassing 24 possible combinations), we achieved a 5% improvement between centralized and parallel computing. When expanding to a larger grid (with 36 combinations, representing roughly 50% more computational load), the improvement increased to 20.8% — with an average execution time of 211.50 seconds using only one worker node, and 167.56 seconds with two nodes.

These results also reveal that, although increasing the number of nodes brings performance improvements, such gains are not linear nor guaranteed in all scenarios. In configurations with smaller data volumes or less demanding tasks, the coordination overhead between nodes may offset the benefits of distribution. On the other hand, in situations where the parameter complexity or grid size increases significantly, the distributed architecture demonstrates a clear advantage, reducing total execution time in a meaningful way.

This observation reinforces the importance of dynamic and well-considered offloading management, which must take into account the nature of the task, the capacity of the available nodes, and the costs associated with parallelization.

5.3 Forecast Model Execution

The tests related to the distributed computing efficiency component of our forecasting model began with the execution of the model configured at the maximum conditions available to the user, using only one contributing node, with the objective of forecasting five variables (temperature, humidity, wind speed, dew point, and atmospheric pressure) over a 5-day interval, with a training slice of 4 weeks (24h * 7 days * 4 weeks = 672 hourly records). The average time observed across the five executions was 22.74 seconds. The next step was to verify whether these times would be reduced by adding one worker node to our cluster. After the same five executions, an average time of 18.42 seconds was recorded, resulting in an 18.99% reduction in total execution time between a centralized and a distributed execution.

With this test completed and the expected results obtained, we decided to gradually reduce task complexity to possibly identify a point where parallel execution is no longer justified. The same five target variables were maintained, but the forecast interval was reduced to three days (72 future records). For 504 training records, the cluster produced somewhat inconsistent results, occasionally even worse than executions with larger training datasets (672 records), possibly due to internal mechanisms of the SARIMA model, which may sometimes favor larger arrays and not scale linearly in execution time relative to the increase in input data volume. Using the same data slice as in the 5-day forecasts, the improvements are evident. We therefore opted to use the most recent 700 records for 3-day forecasts.

Although parallelization brings performance gains in many scenarios, not all tasks—especially smaller or less computationally demanding ones—benefit from distributed execution. In some cases, the additional overhead can even degrade performance. In other words, the extra costs related to task distribution and result collection (data partitioning, task dispatch, waiting for workers, and gathering results) may outweigh the gains achieved through parallel task execution, effectively negating the expected efficiency improvements and worsening overall system performance.

Now, in more demanding scenarios—such as higher input data volume, longer forecasting horizons, unstable network conditions, or limited computational capacity—how does the platform respond? We simulated these challenging conditions by changing the number of training records used by our model: hourly forecasts were requested for six variables (columns) over the next 31 days (24 * 31 steps), using 8664 records (hourly observations from the past 12 months for a given location). Additionally, network delays of 300ms were introduced on all interfaces using the command `'tc qdisc add dev ethx root netem delay 300ms'`.

With one node, three execution attempts were made. All failed, reaching the maximum number of permitted worker failures. Next, one node was added to the system, and the same parameters were tested on two different datasets: London (*dataset* 1), with execution times of 156.33s and 205.20s; and Lisbon (*dataset* 2), with results of 329.33s and 196.10s. We thus reached a situation where the model could only be executed with the support of multiple devices, through distributed processing.

Subsequently, to further stress the platform and analyze its behavior, new columns [temp2, temp3, temp4, temp5, temp6] were added—simulated hourly temperature records—to dataset 1, thereby increasing its size and allowing forecasts for a larger number of variables [temp, dwpt, wdir, wspd, wpgt, pres, temp2, temp3, temp4, temp5, temp6], raising the count from six to eleven forecasted variables. The forecast horizon was maintained, still requesting hourly forecasts for the next 31 days (24 * 31 steps). For these tests, dual-core nodes were used, each limited to 1GB of memory allocated to the *executors*.

With two nodes, four execution attempts were made—all of which crashed during execution. A third node, identical to the others, was then added to the environment. With these three nodes, the forecasts were requested again, most of which completed successfully (one out of five executions failed), with an average execution time of 1042.76 seconds. This demonstrated that increasing the number of nodes in highly demanding tasks is highly beneficial for systems of this kind.

Finally, the complexity of the forecasts was increased further by extending the forecasting steps to 24 * 31 * 3, corresponding to the next three months. The node capacity was also slightly increased, allowing 2GB of RAM per node—effectively doubling the previous allocation—to enable the two-node system to complete the tasks as well. The average execution time was 1612.14 seconds for the 3-node setup and 1734.50 seconds for the 2-node setup. Although the difference is not very large (a bit under 10%), it demonstrates both improved efficiency and greater result stability with the increased number of nodes. These tests ultimately show that, even with workloads that are unrealistic for our target scenario, increasing the number of nodes improves the system's resilience, making it more resistant to failures and allowing it to complete tasks that it was previously unable to handle with fewer nodes. By increasing the capacity of these participating nodes, this capability is also observed with only two nodes, although the results are less efficient due to the lower overall processing capacity of the system.

6 Conclusion

This work aimed to explore the main challenges and solutions associated with the integration of distributed computing technologies at the edge, framed within the context of IoT technologies or potential smart city applications. The proposed platform, which materialized as a fog-based weather forecasting system operating on edge devices, represents a synthesis of all the conclusions drawn throughout the research, addressing common challenges while leveraging the most suitable technologies to handle them.

The review of related work demonstrated the potential of combining local processing paradigms with fog architectures to manage large-scale or real-time data stream computations, decentralizing processing and bringing operations closer to both users and data

sources. Furthermore, regarding the weather forecasting component specifically, the decision phase focused on simplifying computational algorithms to provide greater flexibility and scalability to the platform — particularly important given one of the main objectives was to bring computation closer to the user, thus requiring consideration of a high heterogeneity of devices. Despite these considerations, ensuring the quality and reliability of the forecasts remained essential. A testing methodology was also outlined to evaluate the system's behavior and performance.

The development phase transformed the theoretical concepts explored in previous chapters into a practical and functional implementation. By defining a clear functional architecture, it was possible to logically and efficiently structure the edge and fog layers, assigning distinct responsibilities to each type of node — from data collection and requests to storage, coordination, and distributed processing. The setup of the execution environment, using Apache Hadoop and Apache Spark, highlighted the importance of integrating distributed storage systems with in-memory processing engines, achieving a balance between robustness and performance. The use of Hadoop HDFS as the storage infrastructure, combined with the computational power of Spark, contributed to a lightweight yet capable platform for handling data volume and velocity. The developed code served as the foundation for experimental tests, allowing the simulation of weather forecast execution and parameterization under various configurations.

The discussion of results demonstrated, through controlled experiments, how model parameterization and the number of active nodes directly influence system performance. It was observed that efficient task distribution, although not always advantageous for low-complexity operations, provides clear benefits in more demanding scenarios.

This work therefore reinforces the importance and adaptability of fog architectures in edge networks that leverage distributed processing capabilities to handle contexts requiring the processing of large or continuous data streams (such as the massive IoT environments seen in smart city networks) or those demanding real-time responses, offering efficient resource management and maximizing the utilization of available computational resources.

Finally, it is worth noting that the results and architecture developed in this work can be adapted or extended to different application domains, especially in contexts where real-time response and local autonomy are critical. The platform could evolve, for instance, to incorporate more advanced forecasting techniques, support a greater number of variables, or integrate event-based decision mechanisms.

Thus, this work opens the door for future developments — either through deployment in real-world edge computing environments [12, 22] or through the expansion of its capabilities within the broader smart city and IoT ecosystems. As future work, it would be particularly interesting to extend the developed platform for testing in real environments with heterogeneous devices and variable network conditions, to validate its robustness under production scenarios. Also consider the introduction of federated learning mechanisms, which would enable cooperative forecasting while preserving user data privacy and reducing the need to transfer sensitive information, by smart data integration [15]. Additionally, integrating more advanced predictive models, scaling up data ingestion for massive volumes, and automating the offloading decision process through adaptive intelligence could further enhance the solution's ability to handle dynamic urban environments typical of smart city networks. As an additional perspective for future work, the integration of direct communication mechanisms between edge devices could also be considered, thereby eliminating the dependency on fixed network infrastructure. Technologies such as Wi-Fi Direct and Wi-Fi Aware (Neighbor Awareness Networking), for example, enable peer-to-peer communication and the discovery of nearby nodes. The use of these approaches would strengthen the platform's resilience and availability in scenarios of network connectivity failure. As an additional perspective for future development, the integration of direct communication mechanisms between edge devices could be considered, thereby eliminating the dependency on fixed network infrastructure. Technologies such as Wi-Fi Direct and Wi-Fi Aware (Neighbor Awareness Networking), for example, enable peer-to-peer communication and the discovery of nearby nodes. The use of these approaches would strengthen the platform's resilience and availability in scenarios of network connectivity failure.

Acknowledgments

To Professor Luís Veiga for his support and guidance, as well as to Instituto Superior Técnico, particularly the Taguspark campus.

References

[1] A. Ajina, Jaya Christiyan, Dheerej Bhat, and Kanishk Saxena. 2023. Prediction of weather forecasting using artificial neural networks. *Journal of Applied Research and Technology* 21 (04 2023), 205–211. https://doi.org/10.22201/icat.24486736e.2023.21.2.1698

[2] Francesco Cosimo Andriulo, Marco Fiore, Marina Mongiello, Emanuele Traversa, and Vera Zizzo. 2024. Edge Computing and Cloud Computing for Internet of Things: A Review. *Informatics* 11, 4 (2024). https://doi.org/10.3390/informatics11040071

[3] Hemant Kumar Apat, Rashmiranjan Nayak, and Bibhudatta Sahoo. 2023. A comprehensive review on Internet of Things application placement in Fog computing environment. *Internet of Things* 23 (2023), 100866. https://doi.org/10.1016/j.iot.2023.100866

[4] Enzo Baccarelli, Paola G. Vinueza Naranjo, Michele Scarpiniti, Mohammad Shojafar, and Jemal H. Abawajy. 2017. Fog of Everything: Energy-Efficient Networked Computing Architectures, Research Challenges, and a Case Study. *IEEE Access* 5 (2017), 9882–9910. https://doi.org/10.1109/ACCESS.2017.2702013

[5] Sujit Bebortta, Subhranshu Sekhar Tripathy, Umar Muhammad Modibbo, and Irfan Ali. 2023. An optimal fog-cloud offloading framework for big data optimization in heterogeneous IoT networks. *Decision Analytics Journal* 8 (2023), 100295. https://doi.org/10.1016/j.dajour.2023.100295

[6] Peng Chen, Aichen Niu, Duanyang Liu, Wei Jiang, and Bin Ma. 2018. Time Series Forecasting of Temperatures using SARIMA: An Example from Nanjing. *IOP Conference Series: Materials Science and Engineering* 394 (08 2018), 052024. https://doi.org/10.1088/1757-899X/394/5/052024

[7] Mung Chiang and Tao Zhang. 2016. Fog and IoT: An Overview of Research Opportunities. *IEEE Internet of Things Journal* 3, 6 (2016), 854–864. https://doi.org/10.1109/JIOT.2016.2584538

[8] Jenny Cifuentes, Geovanny Marulanda, Antonio Bello, and Javier Reneses. 2020. Air Temperature Forecasting Using Machine Learning Techniques: A Review. *Energies* 13, 16 (2020). https://doi.org/10.3390/en13164215

[9] Fernando Costa, João Nuno de Oliveira e Silva, Luís Veiga, and Paulo Ferreira. 2012. Large-scale volunteer computing over the Internet. *J. Internet Serv. Appl.* 3, 3 (2012), 329–346. https://doi.org/10.1007/S13174-012-0072-0

[10] Fernando Costa, Luís Veiga, and Paulo Ferreira. 2013. Internet-scale support for map-reduce processing. *J. Internet Serv. Appl.* 4, 1 (2013), 18:1–18:17. https://doi.org/10.1186/1869-0238-4-18

[11] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: simplified data processing on large clusters. *Commun. ACM* 51, 1 (Jan. 2008), 107–113. https://doi.org/10.1145/1327452.1327492

[12] Catarina Gonçalves, José Simão, and Luís Veiga. 2026. A function-as-a-service middleware for decentralized collaborative edge computing. *Future Gener. Comput. Syst.* 175 (2026), 108069. https://doi.org/10.1016/J.FUTURE.2025.108069

[13] Ibrahim Hashem, Nor Anuar, Abdullah Gani, Ibrar Yaqoob, Feng Xia, and Samee Khan. 2016. MapReduce: Review and open challenges. *Scientometrics* 109 (04 2016). https://doi.org/10.1007/s11192-016-1945-y

[14] Manjula Ka and Karthikeyan .P. 2010. DISTRIBUTED COMPUTING APPROACHES FOR SCALABILITY AND HIGH PERFORMANCE. *International Journal of Engineering Science and Technology* 2 (07 2010).

[15] Pradeeban Kathiravelu, Ashish Sharma, Helena Galhardas, Peter Van Roy, and Luís Veiga. 2019. On-demand big data integration - A hybrid ETL approach for reproducible scientific research. *Distributed Parallel Databases* 37, 2 (2019), 273–295. https://doi.org/10.1007/S10619-018-7248-Y

[16] Latif U. Khan, Ibrar Yaqoob, Nguyen Tran, S.M. Kazmi, Tri Nguyen Dang, and Choong Seon Hong. 2019. Edge-Computing-Enabled Smart Cities: A Comprehensive Survey. https://doi.org/10.48550/arXiv.1909.08747

[17] Guorui Li and Ying Wang. 2013. Automatic ARIMA modeling-based data aggregation scheme in wireless sensor networks. *EURASIP Journal on Wireless Communications and Networking* 2013, 1 (25 Mar 2013), 85. https://doi.org/10.1186/1687-1499-2013-85

[18] Saraju Mohanty. 2016. Everything You Wanted to Know About Smart Cities. *IEEE Consumer Electronics Magazine* 5, 60–70. https://doi.org/10.1109/MCE.2016.2556879

[19] Taewoo Nam and Theresa A. Pardo. 2011. Conceptualizing smart city with dimensions of technology, people, and institutions. In *Proceedings of the 12th Annual International Digital Government Research Conference: Digital Government Innovation in Challenging Times* (College Park, Maryland, USA) *(dg.o '11)*. Association for Computing Machinery, New York, NY, USA, 282–291. https://doi.org/10.1145/2037556.2037602

[20] Román Osorio, Mario Peña-Cabrera, I. Lopez-Juarez, Gaston Lefranc, and R. Tovar-Medina. 2017. Smart semaphore using image processing. 1–7. https://doi.org/10.1109/CHILECON.2017.8229679

[21] Piero Paialunga. 2021. Weather forecasting with Machine Learning, using Python. https://towardsdatascience.com/weather-forecasting-with-machine-learning-using-python-55e90c346647

[22] André Pires, José Simão, and Luís Veiga. 2021. Distributed and Decentralized Orchestration of Containers on Edge Clouds. *J. Grid Comput.* 19, 3 (2021), 36. https://doi.org/10.1007/S10723-021-09575-X

[23] Remo Scolati, Ilenia Fronza, Nabil El Ioini, Areeg Samir, and Claus Pahl. 2019. A Containerized Big Data Streaming Architecture for Edge Cloud Computing on Clustered Single-Board Devices. https://doi.org/10.5220/0007695000680080

[24] Eman Shaikh, Iman Mohiuddin, Yasmeen Alufaisan, and Irum Nahvi. 2019. Apache Spark: A Big Data Processing Engine. 1–6. https://doi.org/10.1109/MENACOMM46666.2019.8988541

[25] Simar Preet Singh, Anand Nayyar, Rajesh Kumar, and Anju Sharma. 2019. Fog computing: from architecture to edge computing and big data processing. In *The Journal of Supercomputing*, Vol. 75. 2070–2105. https://doi.org/10.1007/s11227-018-2701-2

[26] Shanhe Yi, Zijiang Hao, Zhengrui Qin, and Qun Li. 2015. Fog Computing: Platform and Applications. https://doi.org/10.1109/HotWeb.2015.22

[27] Xuyun Zhang, Lianyong Qi, and Yuan Yuan. 2022. Editorial: Convergency of AI and Cloud/Edge Computing for Big Data Applications. *Mobile Networks and Applications* 27 (06 2022). https://doi.org/10.1007/s11036-022-01911-z

Received 13 October 2025