ProFog: A Proactive Elasticity Model for Fog Computing-based IoT Applications

Guilherme Gabriel Barth, Rodrigo da Rosa Righi[a], Cristiano André da Costa[b] and Vinicius Facco Rodrigues[c]

Applied Computing Graduate Program, Universidade do Vale do Rio dos Sinos, São Leopoldo, Brazil

Keywords: Fog Computing, IoT, Resource Management, Elasticity, Prediction.

Abstract: Today, streaming, Artificial Intelligence, and the Internet of Things (IoT) are being some of the main drivers to accelerate process automation in various companies. These technologies are often connected to critical tasks, requiring reliable and scalable environments. Although Fog Computing has been on the rise as an alternative to address those challenges, we perceive a gap in the literature related to adaptability on the number of resources on both cloud and fog layers. Multiple studies suggest different Cloud-Fog architectures for IoT implementations, but not thoroughly addressing elasticity control mechanisms. In this context, this article presents ProFog as a proactive elasticity model for IoT-based Cloud-Fog architectures. ProFog uses the ARIMA prediction model to anticipate load behaviors, so triggering scaling actions as close to when they are required as possible. This strategy allows the delivery of new resources before reaching an overloaded or underloaded state, benefiting performance, and energy saving. We developed a ProFog prototype that showed an improvement of 11.21% in energy consumption in favor of ProFog.

1 INTRODUCTION

The advance of the Internet-of-Things (IoT) industry follows many segments of the economy needs, such as the automation in sectors as part of Industry 4.0 (I4.0) (Masood and Sonntag, 2020). Today, it is common to see IoT spreading on smart cities, transportation systems, and healthcare scenarios, enabling not only process monitoring and newer notification systems but also challenges related to scalability and performance(Mohamed, 2017). Over the past few years, more industries started to adhere to process automation using IoT solutions, and the tendency is the usage growth as the results of such deployments begin to be seen and generate value, minimizing risks and costs(Fischer et al., 2020). Relying only on a local central processing server to address all IoT implementations would result in high costs on infrastructure and maintenance. This type of deployment requires a robust server and network to avoid overloading the server or flooding the network with in-transit data. IoT systems provide data to critical applications that require high availability from the server in most industrial scenarios. Failure of the system may result in significant financial losses for the company or even damage to machinery.

Sending requests to a Cloud server adds a constant latency overhead to the communication that can be unacceptable in some areas. For instance, in healthcare scenarios, response time is sometimes critical to decide between life or death of patients (Shah and Aziz, 2020). Vaquero explained that the existence of billions of devices always producing data on the edge of the network may cause the network to become a clear bottleneck (Vaquero and Rodero-Merino, 2014). Using Cloud Computing as the primary paradigm for IoT scenarios will only further contribute to this network congestion (Yin et al., 2018) as the Internet is not scalable and efficient enough to handle IoT big data as explained by Xiang (Sun and Ansari, 2016). Also, submitting massive loads of data to a Cloud Computing server may result in high costs for the service as providers may charge based on the amount of data going into the Cloud. That reveals the necessity of designing new architectures and solutions that enable higher scalability while reducing requests on the network and maintaining the required Quality of Service (QoS). Fog Computing has been gain-

[a] https://orcid.org/0000-0001-5080-7660
[b] https://orcid.org/0000-0003-3859-6199
[c] https://orcid.org/0000-0001-6129-0548

Barth, G., Righi, R., André da Costa, C. and Rodrigues, V.
ProFog: A Proactive Elasticity Model for Fog Computing-based IoT Applications.
DOI: 10.5220/0010707000003058
In *Proceedings of the 17th International Conference on Web Information Systems and Technologies (WEBIST 2021)*, pages 380-387
ISBN: 978-989-758-536-4; ISSN: 2184-3252

ing a position as an architecture for IoT scenarios due to its concept of keeping the processing near its data sources, thus allowing reduced latency, stable QoS (Suganuma, 2018) and addressing another of the critical concerns of IoT, which is the data security.

In this context, in this article, we introduce ProFog, a proactive elasticity model for resource management on Cloud-Fog environments for IoT deployments. The model proves an architecture and a middleware to run elastic-driven IoT applications on both Cloud and Fog. ProFog targets applications where response time to react against received IoT data is critical. Scalability and financial costs are vital concerns for ProFog, which addresses elasticity using ARIMA (Auto-Regressive Integrated Moving Average) (da Rosa Righi et al., 2020) to predict the system's behavior to never achieving an under or overloaded situation. When perceiving it, ProFog scales up or down resources in the fog layer, adding or removing resources as close to the situation requirements as possible. The next sections describes ProFog and its evaluation.

2 ProFog MODEL

This section describes ProFog, a model that uses proactive elasticity to improve resource allocation for IoT applications that execute in the Fog.

2.1 Design Decisions

Considering the varying requirements of different IoT scenarios, a Cloud-Fog architecture is currently the most promising solution for IoT implementation in the long-term. It allows for a wider variety of scenarios to be implemented while also providing additional security and control over the data, reducing network overload and facilitating QoS adherence. To meet QoS requirements and create a stable and reliable environment for the execution of IoT services, the system needs to use elasticity control techniques and scale itself on demand. Proactive elasticity models apply the collected load data on predictive algorithms to estimate future load. This implies triggering scaling operations before an undesired load situation occurs since the resources become available before the server reaches, in fact, either an underloaded/overloaded state. Given the importance of stability and QoS for IoT applications, we have decided to use proactive elasticity on ProFog. Unlike Cloud servers, Fog nodes are limited on processing capacity, limiting the use of vertical elasticity (i.e., resizing of resources like CPU, memory, or disk). For that reason, we also have chosen to apply horizontal elasticity by adding and removing processing instances (nodes and containers, for example).

Figure 1 illustrates a high-level overview of a Cloud-Fog architecture for IoT implementation. We modeled ProFog in three different physical layers: Internet-of-Things, Fog Computing, and Cloud Computing. The IoT layer comprises network edge devices that collect data to be analyzed, interact with its surroundings, or require services made available on the Fog Computing layer. IoT devices may vary from simple sensors to smart devices, like those in Smart Cars. The Fog layer comprises multiple Fog nodes that perform initial data treatment and provide time-sensitive services for the IoT layer and other service consumers. Here, single-board computers like Raspberry Pi and Arduino can appear as possible solutions. The Cloud layer is in charge of monitoring and scaling the system. In the Cloud, we have non-time-sensitive services such as data analytic and storage, for example.

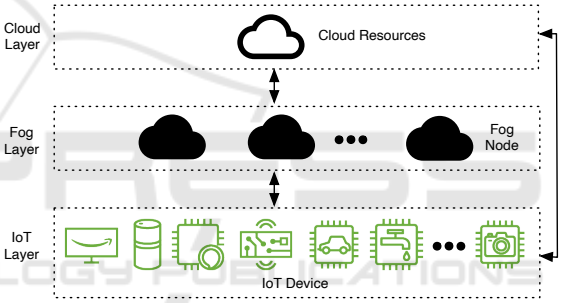

Figure 1: High-level view of a Cloud-Fog architecture.

2.2 Architecture

ProFog focuses on elasticity for the execution of time-critical applications on Fog Computing. For that purpose, we apply load prediction algorithms on previously collected time-series data to determine resource resizing needs and trigger proactive scaling. ProFog acts as middleware by managing the deployment and execution of different services and applications seamlessly from a user viewpoint. Figure 2 provides an overview of the core components of the model, highlighting in red the ones that are part of ProFog. The model is composed of three physical layers - Cloud Computing, Fog Computing, and IoT. However, ProFog is distributed only on the Cloud Computing and Fog Computing layers.

ProFog is composed of three different modules: (*i*) Cloud Manager; (*ii*) Fog Manager; and (*iii*) Elasticity Manager. The Cloud Manager is responsible for triggering services to the Fog layer based on the IoT

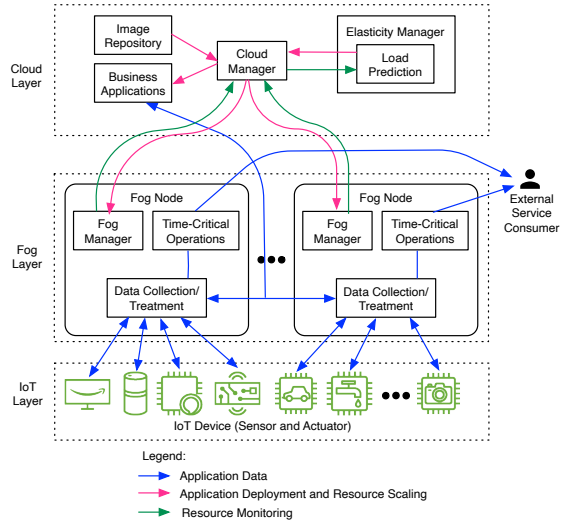

Figure 2: ProFog model applied to Cloud-Fog environment. Elements highlighted with a red dot represent the main modules of ProFog.

devices' requests, load balancing, and communication with Fog nodes about load and triggering scaling actions on the Fog and Cloud layers. The Fog Manager downloads application images from the Cloud and instantiates them as containers on the Cloud Manager's request while also constantly monitoring their load and updating the Cloud Manager about the same. The Elasticity Manager receives data about the load of the Fog nodes and Cloud applications from the Cloud Manager, applies the collected data to load prediction algorithms, and informs the Cloud Manager about its scaling decisions.

ProFog requires a Fog Manager's deployment to all Fog nodes of the architecture and a Cloud Manager and an Elasticity Manager to the selected Cloud server. The number of Fog nodes on the system depends on the complexity of the implemented scenario. Applications and services deployed on the Fog layer must also follow specific formats for compatibility with ProFog. In runtime, IoT devices initially request services from the Cloud Manager that redirects them to the Fog node, which provides the service, as depicted in Figure 3. From this point, the IoT device communicates directly with the service provider on the Fog. These services may perform data treatment and forward the data to the Cloud for analytics (or other processing intensive task) or provide time-critical functionalities to the IoT layer or external service consumers. This strategy allows a reduction in network congestion, favoring QoS adherence for time-critical services benefitting from Fog's reduced latency.

ProFog concentrates all management activities on

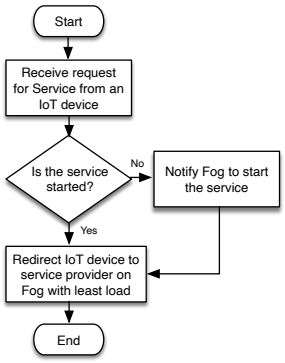

Figure 3: Service request routing process performed by the Cloud Manager.

a single point: the Cloud. This strategy facilitates system management and reduces complexity. The Cloud Manager also monitors the load of local applications and collects information about Fog applications' load from each Fog Manager. It then forwards the data to the Elasticity Manager, who applies load prediction algorithms based on the collected data. This manager also provides proactive elasticity to both Cloud and Fog layers with no human intervention.

2.3 Application Model for Fog Deployment

Applications are exposed for deployment as images, like Docker container images of Virtual Machine (VM) templates, including all the necessary services for its operations. This strategy is similar to what was proposed by Nguyen, Phan, Park, Kim S., and Kim T. (Nguyen et al., 2020). ProFog creates an instance of the container image to start the application, requiring the image to be built and configured to run the application upon initialization. Any necessary parameters for the application start-up can be provided during the container's instantiation as long as the image was previously built with an ENTRYPOINT or CMD statement to support command-line argument consumption through Docker. Deploying applications as isolated containers simplifies the deployment process while also providing more flexibility to the application development. It becomes possible to install any necessary resources for the application directly on its container.

As applications consist of containers or VM templates, they are isolated from the Fog Manager, requiring them to internally implement any functionalities necessary for consuming data or providing services. For instance, an application that receives data from edge devices performs operations on such data and then forwards it to the Cloud must implement

an HTTP (Hypertext Transfer Protocol) server to receive data from edge devices through HTTP requests. The Fog Manager provides the application with the port number to run its HTTP server upon initialization. The application must implement a shutdown service and reroute any connected clients back to the Cloud prior to the shutdown. The Fog Manager calls the shutdown service to notify the application of any eventual shutdown due to down-scaling, providing it the time left for the shutdown and the Cloud Manager's address, which may be used to reroute clients to another active application instance. The application is given the duration of the time left for the shutdown to handle active clients, whatever way it suits it. Once the time is over, the Fog Manager performs the shutdown, finishing all of its running processes. All monitoring and elasticity control-related tasks are managed by the Fog Manager and Cloud Manager, relieving application developers of any concerns of elasticity control at the application level.

2.4 Elasticity Algorithm

ProFog uses proactive elasticity to provide stable QoS and prevent the system from overloading, which may compromise its operations. More specifically, the Elasticity Manager evaluates the load of a given metric (CPU, for instance) against thresholds to decide whether to reorganize the number of Fog nodes and application instances in the Cloud. Proactive elasticity is not only capable of achieving more consistent and reliable scaling decisions with better rates of false-positive results in case of sudden peaks of the load when compared to reactive elasticity, but it also allows to include relevant metrics such as deployment and start-up time into consideration for scaling decisions. Proactive elasticity makes it possible to have the needed resources ready before required, preventing the application from going into an overloaded state.

First, proactive elasticity depends on constant monitoring of the system's state. Second, we have the prediction of future load based on the collected data through time-series forecasting algorithms. We selected the mathematical model named ARIMA for this purpose. It is widely used for time series-based prediction, and it can provide predictions in fewer boot cycles than the machine learning models (da Rosa Righi et al., 2020). Auto Regression (AR) means that it is a regression of the variable against itself. Thus the variable of interest is forecasted using a linear combination of its past values. Moving Average models use past forecast errors rather than past values of the variable. ARIMA takes both the previous values of the variable and the previous forecast errors into consideration for its predictions.

ARIMA has several configurations that are dependent on three parameters: d which is the minimum number of differences that are required to make the time series stationary, p which is the order of auto-regressive terms (AR), and q which is the order of the moving average (MA) term. We have used Auto Arima, a function from the time series analysis library, to determine these parameters' most appropriate values. As ARIMA's predictions depend on the analysis of time-series data, it is necessary to collect a given number of load observations to start the load predictions. Righi, Correa, Gomes, and Costa (da Rosa Righi et al., 2020) suggest the use of at least six cycles - or six monitoring observations - for the calculation of the predictions. We have chosen to use the data of ten cycles for our predictions, meaning ARIMA will take $10 \times monitoring\_observation\_period$ to initialize and provide its first prediction. For example, if there are 10 seconds between each observation of the system's load, it will take 100 seconds to collect enough values to make the first prediction.

We have also defined an *ahead* parameter based on Equation 1. Righi, Correa, Gomes, and Costa (da Rosa Righi et al., 2020) proposed this Equation to determine how many cycles ahead into time ARIMA should make its prediction to allow the deployment of new resources in time. A proper definition of the *ahead* parameter is crucial to avoid two possible pitfalls:

1. too high of a value may cause the prediction to missing short-term changes on the application behavior incurring in false-negative or false-positive elasticity;
2. if the value is too low, the elasticity action may deliver the resources after they start to be necessary, causing the application to run on an overloaded state for some time, affecting its operations (da Rosa Righi et al., 2020).

$$ahead = \frac{Abs(Max(scaling\_out\_time))}{monitoring\_observation\_period} \quad (1)$$

Figure 4 depicts the flow of actions performed by ProFog for elasticity management. Like reactive models, proactive elasticity models also require the definition of upper (UT) and lower (LT) threshold settings. After performing load prediction, the predicted values are compared to the threshold values to determine if scaling actions are required. Figure 5 illustrates the behavior of proactive elasticity models during execution.

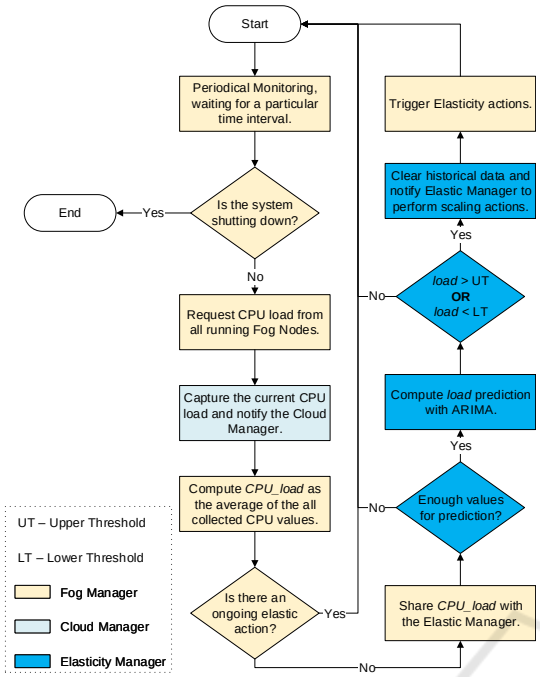

Figure 4: ProFog elasticity management flowchart.

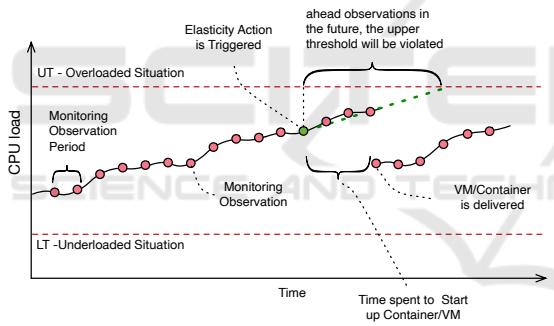

Figure 5: Server elasticity managed by a proactive elasticity model.

3 EVALUATION METHODOLOGY

This section describes the methodology we employed to evaluate the model.

3.1 Application

After analyzing a series of possible IoT scenarios documented by the OpenFog Consortium, an association for the advance of Fog Computing as a connected and interoperable architecture, we decided to use a video streaming use case for the prototype's test. Cases such as healthcare and Industry 4.0 require precise knowledge of the industry to recreate scenarios properly. The variables, data, and the analysis of such data are particular to their environment. However, streaming scenarios are far more versatile as they are more connected to the information technology area and do not involve as many unfamiliar end devices and sensors.

Currently, we could not identify any applications or benchmarks designed to evaluate such scenarios' performance, so we have modeled an application to evaluate the prototype. The official version of the streaming scenario proposed by the Industrial Internet Consortium comprises multiple different Fog levels, each responsible for a specific task to achieve the highest level of performance for live-event streaming. As this project's solution is not specific for streaming, but for a general Cloud-Fog architecture, we simplified the streaming scenario to a single Fog level that provides Video-On-Demand (VOD) to clients monitoring its state and performing scaling operations proactively.

3.2 Infrastructure

In order to evaluate the results of ProFog, we have deployed our solution to a Cloud-Fog environment. We have used an Azure Container instance for the Cloud layer, configured with 2 CPUs and 4GB of RAM, to host the Cloud Manager and Elasticity Manager. The Fog layer was composed of three Raspberry PI 4 Model B microcomputers as Fog nodes, hosting application services.

On the Industrial Internet Consortium's proposition of the video broadcasting scenario, which we have based our case on, the IoT layer comprises HD video cameras that collect data for transmission. Our goal is to validate the elastic capacity of ProFog and not the streaming scenario. We have removed the HD video cameras in exchange for pre-recorded video files (VOD), allowing us to simplify the application and test infrastructure while maintaining the elastic behavior. By doing that, we have removed the IoT layer from the test infrastructure. In our test scenario, Video-On-Demand clients consume the Fog layer's services. JMeter generates these clients on a separate machine, creating service load for the Fog and scaling needs. The machine we used to emulate the clients has an Intel I7 7th Generation processor and 16GB of RAM.

3.3 Prototype

We built both Cloud Manager and the Fog layers' services on top of Node.js[1]. The Fog nodes provide the

[1]https://nodejs.org/

streaming service through an HTTP server running on Node.js, which streams video content to the connected clients using the HTTP Live Streaming (HLS) protocol. In turn, we developed the Elasticity Manager in Python, employing the pmdarima[2] library for time series analysis. Finally, we have used Docker[3] to create and manage the container instances for the prototype while Dockerhub was used as the container image repository.

For our prototype, we have used FFMpeg to encode a nine-minute long video file and stored the generated index file and video stream chunks locally for consumption for the preparation of the video files. Streaming clients initially connect to the Cloud Manager, which then redirects them to an active Fog node that provides the requested service seamlessly to the client. Upon receiving a service request, the Cloud Manager performs load balancing between different active Fog nodes, always directing clients to the node with the least load. The prototype does not contain any application on the Cloud side for the evaluation of Cloud elasticity.

3.4 Workload and Execution Scenarios

We have used JMeter[4], a load testing tool maintained by the Apache Foundation, to simulate streaming clients as it is a stable load testing tool with an active community that can handle massive amounts of threads. We have used different JMeter elements to emulate the required streaming logic, allowing the redirect of streaming clients to different servers during the streaming process and a few custom Groovy[5] scripts that perform validations of the request responses emulate video buffer control.

During the test execution, each client generated by JMeter makes HTTP requests to a service provider on the Fog layer for transport stream files - video chunks. We have configured JMeter to emulate clients at two different points in time. The first client emulation created 400 clients over three minutes (180 seconds). The second client emulation created 200 more clients over one minute and 40 seconds (100 seconds). The first load was triggered at the test start and the second load at four minutes and 10 seconds (250 seconds).

It is important to note that we also developed ProFog to support reactive elasticity for comparison purposes. In order to evaluate the prototype, we have configured ProFog to accept two different execution modes, which allow us to observe two different scenarios in the same environment:

- Scenario 1: Streaming service with ProFog running on reactive elasticity mode;
- Scenario 2: Streaming service with ProFog running on proactive elasticity mode.

3.5 Evaluation Metrics

We observed the system's average load to determine the efficiency of ProFog on handling resizing upon demand. Average load is a metric that considers CPU load, network connections, and I/O operations, being a more reliable metric for specific applications that do not rely exclusively on CPU-intensive operations. Additionally, we observe energy consumption as a metric to evaluate this behavior. We can estimate the energy consumption by analyzing the deployment times of containers and for how long they were active (da Rosa Righi et al., 2020), as shown in Equation 2. In the equation, n is the maximum number of containers, and $T(i)$ is the time spent running with i containers. For example, suppose the following scenario: 1 container for 40 seconds, 2 containers for 25 seconds, 3 containers for 50 seconds; this would result in $Energy = 1 \times 40 + 2 \times 25 + 3 \times 50 = 240$.

$$Energy = \sum_{i=1}^{n} (i \times T(i)) \quad (2)$$

3.6 Evaluation Parameters

We have configured ProFog to collect data from each of the Fog nodes every 5 seconds. Calculating the prediction on the Elasticity Manager takes up to 4 seconds. With these metrics, the total interval data for each collection is around 9 seconds. The download of the container image from Dockerhub takes up to 20 seconds, depending on network congestion. The initialization of the application using the container image is less than 2 seconds, adding up to 22 seconds. Based on Equation 1, predictions should be performed for 2.44 measures ahead, resulting from 22 divided by 9. We have decided to round this value up to 3 to compensate for unaccounted communication delays.

The upper and lower CPU threshold values for elasticity control were set to 90% and 50%, respectively, based on observing the system's behavior over multiple test runs.

[2]https://pypi.org/project/pmdarima/
[3]https://www.docker.com/
[4]https://jmeter.apache.org/
[5]https://groovy-lang.org/

4 RESULTS

This section presents the results from running the prototype in the scenarios previously described.

4.1 Load and Resource Allocation

Figure 6 illustrates the system behavior using the reactive elasticity approach throughout a twelve-minute load test. Analyzing the graph, we can see that a load peak at 1:45min has triggered the second Fog node's start-up. After that, the same situation repeats itself at 2:35min. At 2:40min, three Fog nodes were already active and remained active until the load dropped below the lower threshold at 10:25min.

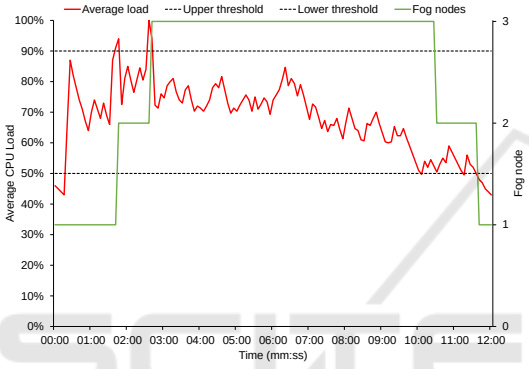

Figure 6: Scenario 1: Streaming service with ProFog running on reactive elasticity mode.

Figure 7 illustrates the system behavior with ProFog running in proactive mode. We can see that the second Fog node's start-up suffered a delay as the predicted values were lower than the observed load value, which was a peak. In this scenario, the second Fog node's start-up took place at 2:05min and the third node at 4:25min. We can also see that the predicted values are very close to the observed load values. This behavior may be related to ARIMA's parameterization or that the predictions only occur three measures ahead, which is relatively low. Increasing the *ahead* parameter could improve the predictions, but it could also incur a more prolonged resource usage than necessary. The drops to zero in the predicted values mean that the Elastic Manager has requested an elastic action to be taken and cleared its load values buffer. This action could be followed or ignored by the Cloud Manager depending on resource availability at the moment.

We have opted not to calculate a standard deviation or mean at this point as the application used for the test is dynamic and subject to multiple variables of the emulation environment, which would affect the results. Furthermore, the most crucial point for us to monitor is the system's elastic behavior and its impact on resource allocation. As the current prototype does not contain any applications that run on the Cloud, we could not verify the Cloud side's elastic behavior either.

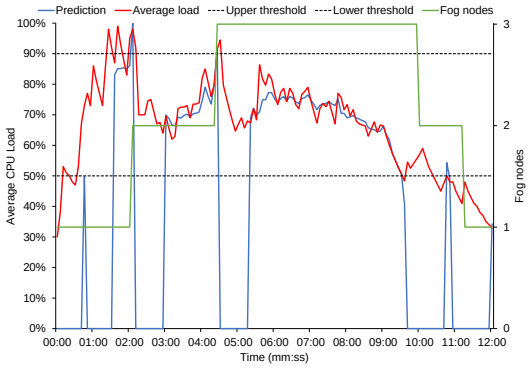

Figure 7: Scenario 2: Streaming service with ProFog running on proactive elasticity mode.

4.2 Energy Consumption

We have mapped the Fog node initialization times for each scenario to calculate an estimate of energy consumption. Table 1 presents the results for Scenario 1 and table 2 the results for Scenario 2. By comparing the tables' total values, we see that scenario two has presented lower energy consumption. Even though the predictions were not far enough ahead to trigger scaling actions in advance on the tested scenario, we can see that time series analysis has helped smooth the data and minimize unnecessary scaling on sudden peaks. This behavior has improved energy consumption by 11.21%, resulting from (1785/1605 - 1).

Table 1: Energy consumption estimate based on equation 2 for the scenario 1.

Number of Fog Nodes	Period of time (s)	Energy consumption
1	125	125
2	125	250
3	470	1410
Total		1785

4.3 Discussion

The proximity between the actual load values and the predicted load values shows us that the use of proactive elasticity for fast deployment environments, such as those based on containers, may be challenging. The lower the time required for deployment is, the

Table 2: Energy consumption estimate based on equation 2 for the scenario 2.

Number of Fog Nodes	Period of time (s)	Energy consumption
1	170	170
2	215	430
3	335	1005
Total		1605

more difficult it is to make a prediction capable of scaling the system in time. By modifying the *ahead* parameter used for the predictions, we could further anticipate scaling needs. However, at the same time, we may deliver resources before they are required, which may incur unnecessary energy consumption and additional costs.

Even though the predictions performed by ProFog were too close to the actual load values for us to see preemptive scaling actions take place, we can see that the use of time series analysis has brought other benefits to the system. By analyzing the system load over time for Scenario 2, we see that the load predictions have smoothed load peaks, helping avoid unnecessary deployment of new Fog nodes. By preventing unnecessary deployment of Fog nodes caused by load peaks, ProFog has reduced the number of active machines for some time, consequently reducing energy consumption and operating costs for the system.

5 CONCLUSION

As IoT grows across multiple industries, it becomes necessary to review how solutions design IoT systems. Cloud data centers are often used to implement IoT scenarios as they offer scalability and reliability. However, such configuration poses many challenges - from security to QoS and network congestion - that may prevent certain use cases or the adoption in specific industries. These challenges have fostered the advance of another system architecture named Fog computing. This architecture brings the processing of data closer to the resource, allowing better response times, lower latency, and increased security.

In this context, this article addressed proactive elasticity for resource allocation on Fog computing for IoT implementations by presenting a model named ProFog. This model manages resource allocation and provides proactive elasticity to applications without any user intervention or elasticity control logic from the application end. To validate the model, we have built a prototype using Microsoft Azure - like the Cloud - and three Raspberry Pi 4 microcomputers - which operated as Fog nodes. We evaluated our prototype using a Video-On-Demand streaming scenario. However, ProFog covers various scenarios, such as manufacturing, healthcare, and Smart Cities.

ACKNOWLEDGMENT

The authors would like to thank the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - CAPES (Finance Code 001) and Conselho Nacional de Desenvolvimento Científico e Tecnológico - CNPq (Grant Number 303640 / 2017-0).

REFERENCES

da Rosa Righi, R., Correa, E., Gomes, M. M., and da Costa, C. A. (2020). Enhancing performance of iot applications with load prediction and cloud elasticity. *Future Generation Computer Systems*, 109:689 – 701.

Fischer, G. S., da Rosa Righi, R., de Oliveira Ramos, G., da Costa, C. A., and Rodrigues, J. J. (2020). Elhealth: Using internet of things and data prediction for elastic management of human resources in smart hospitals. *Engineering Applications of Artificial Intelligence*, 87:103285.

Masood, T. and Sonntag, P. (2020). Industry 4.0: Adoption challenges and benefits for smes. *Computers in Industry*, 121:103261.

Mohamed, N. (2017). SmartCityWare: A Service-Oriented Middleware for Cloud and Fog Enabled Smart City Services. *NEW ERA OF SMART CITIES: SENSORS COMMUNICATION TECHNOLOGIES AND APPLICATIONS*.

Nguyen, N. D., Phan, L. A., Park, D. H., Kim, S., and Kim, T. (2020). Elasticfog: Elastic resource provisioning in container-based fog computing. *IEEE Access*, 8:183879–183890.

Shah, S. A. B. and Aziz, S. M. (2020). Response time determinism in healthcare data analytics using machine learning. In Yang, H., Pasupa, K., Leung, A. C.-S., Kwok, J. T., Chan, J. H., and King, I., editors, *Neural Information Processing*, pages 203–210, Cham. Springer International Publishing.

Suganuma, T. (2018). Multiagent-Based Flexible Edge Computing Architecture for IoT. *IEEE Network*.

Sun, X. and Ansari, N. (2016). Edgeiot: Mobile edge computing for the internet of things. *IEEE Communications Magazine*, 54(12):22–29.

Vaquero, L. M. and Rodero-Merino, L. (2014). Finding your way in the fog: Towards a comprehensive definition of fog computing. *SIGCOMM Comput. Commun. Rev.*, 44(5):27–32.

Yin, L., Luo, J., and Luo, H. (2018). Tasks scheduling and resource allocation in fog computing based on containers for smart manufacturing. *IEEE Transactions on Industrial Informatics*, 14(10):4712–4721.