# IoT Workload Distribution Impact between Edge and Cloud Computing in a Smart Grid Application

Otávio Carvalho[1], Manuel Garcia[1], Eduardo Roloff[1],
Emmanuell Diaz Carreño[1], Phillipe O. A. Navaux[1]

Informatics Institute – Federal University of Rio Grande do Sul (UFRGS)
Porto Alegre – RS – Brazil
{omcarvalho,magnapa,eroloff,edcarreno,navaux}@inf.ufrgs.br

**Abstract.** The advent of Internet of Things is now part of our reality. Increasing amounts of data are being continuously generated and monitored through widespread sensing technologies such as personal smartphones, large scale smart cities sensor deployments and smart electrical grids.

However, the ability to aggregate and act upon such data gathered by sensors is still a significant research and industrial challenge. Devices that are able to collect and act on data at network edges are bounded by the amount of data that can be sent over networks.

In this paper, we analyze the impact of workload distribution in a smart grid application, evaluating how we can increase processing rates by leveraging each time more powerful edge node processors.

Our results show that our test bed application, leveraging cloud nodes processing and processing windows, is able to achieve processing rates of approximately 800k measurements per second using four edge node processors and a single cloud node.

## 1 Introduction

The Internet of Things (IoT) is now a reality, and we are connecting each time more devices – such as personal consumer electronic devices, home appliances, cameras, medical devices, and all types of sensors – to the Internet environment. This ubiquity unlocks the potential to innovations that can use the data generated by those devices to enable smart cities, smart infrastructures and smart services that can improve quality of life.

By 2025, researchers estimate that the IoT will have a potential economic impact of 11 trillion per year – which would be equivalent to about 11% of the world economy. They also expect that one trillion IoT devices will be deployed by 2025. In majority of the IoT domains such as infrastructure management and healthcare, the major role of IoT is the delivery of highly complex knowledge-based and action-oriented applications in real-time [9].

Technologies and applications being created for mobile computing and the Internet of Things (IoT) are driving computing toward dispersion. Edge computing is a new paradigm in which substantial computing and storage resources – variously referred to as cloudlets, micro datacenters, or fog nodes – are placed at the Internet's edge in close proximity to mobile devices or sensors [30].

Smart grids will allow consumers to receive near real-time feedback about their energy consumption and price, enabling them to make their own informed decisions about consumption and spending. On the producer point-of-view, we can leverage home consumption data to produce energy forecasts, enabling near real-time reaction and a better scheduling of energy generation and distribution [7]. In this way, smart grids will save billions of dollars on both sides in the long run, for consumers and the generators, according to recent forecasts [26].

Since millions of end-users will be taking part into processes and information flows of smart grids, high scalability of these methods turns into an important issue. To solve these issues, cloud computing services present themselves as a viable solution, by providing reliable, distributed and redundant capabilities at global scale [10]. However, there is a large set of applications which cannot accept the delay caused by transferring data to the cloud and back, being bounded by latency. Also, it is not efficient to send a large number of small packages of data to the cloud for processing, as it would saturate network bandwidth and decrease scalability of the applications [13].

In this paper, we explore what is achievable in a realistic application for Short-Term Load Forecast (STLF), in terms of latency and bandwidth, by moving varying portions of computation from cloud to edge nodes. By building a distributed application that handles communication and processing through edge processors and cloud computing machines, we are able to distribute load between cloud and edges nodes and analyze what is possible to obtain in terms of processing speedup in comparison with a pure cloud-based application.

## 2 Related work and Discussion

The state-of-the-art works on edge computing includes platforms and frameworks that aim to provide scalable processing closer to the network border, in order not only to provide low latency results, but also to better utilize resources available on the network. The approaches to solve these problems include predominantly cloud-like deployments at the edge (often described as cloudlets). The most prominent approaches and how they relate to our work are briefly described below. In the end of Section 2 we explain how these works relate to the proposed solution on the Section 3.

Femtoclouds [19], REPLISOM [1], CloudAware [24] and ParaDrop [22] are examples of applications that explore computational offloading to nearby devices. The most predominant usages either rely on offloading to edges that are underutilized or offload processing to nearby network centralizers (modified wireless network access points or specialized mobile network base station hardware). The

only work that explores computation to considerably more performant nodes is EdgeIoT, which provisions virtual machines in a nearby mobile base station.

Most of the related works differ from ours by relying on hardware changes or significant modifications on the underlining communication protocols. ParaDrop applies changes to nearby wireless access points in order to provide its functionality. HomeCloud [25] relies on a Software Defined Networking (SDN) implementation on the networks in order to schedule its Network Function Virtualization (NFV) capabilities. EdgeIoT [31] relies on changes on the nearby mobile base stations in order to deploy Virtual Machines (VMs) which are used for computation offloading.

Several of the state-of-the-art works go ahead of the scope this work on the regarding of work scheduling capabilities. Cumulus [15] provide a complete framework that controls task distribution under a heterogeneous set of devices on its cloudlet. FemtoClouds and CloudAware monitor device usage on its cloudlets in order to improve device usage as part of its scheduling algorithms. CloudAware also provides a specific Application Programming Interface (API) to improve the experience of implementing applications using their framework.

Although the related works present multiple initiatives towards Edge Computing that improve computational offloading to nearby nodes, none of them explore the potential of combining the low latency edge nodes processing with scalable and more performant sets of commodity machines on public clouds.

In the next sections, we describe and evaluate our approach to this problem, applying edge processing capabilities as a complement to long running jobs on private clouds, in order to improve throughput, decrease network traffic and minimize latency.

## 3 Architecture and Implementation

The architectural infrastructure of our test bed application deployment can be described as a composition of three layers, as it is represented on Figure 3: (1) Cloud layer, where we execute long running scalable jobs that can provide more performant processing at the cost of latency; (2) Edge layer, composed by edge nodes that are used to pre-process and aggregating data before sending to the Cloud; and the (3) Sensor layer, which is composed by the sensors that communicate directly with Edge layer nodes to receive actuation requests and provide measurements to the network.

### 3.1 Cloud layer

In this layer, we create virtual machines to execute our application in order to aggregate data to be received from the Edge layer nodes. The Cloud layer should be composed by elements that can be able to process data as it arrives. It can be instantiated by implementing the same application logic from the layer below, but instead it should be configured to receive data from multiple edge nodes.

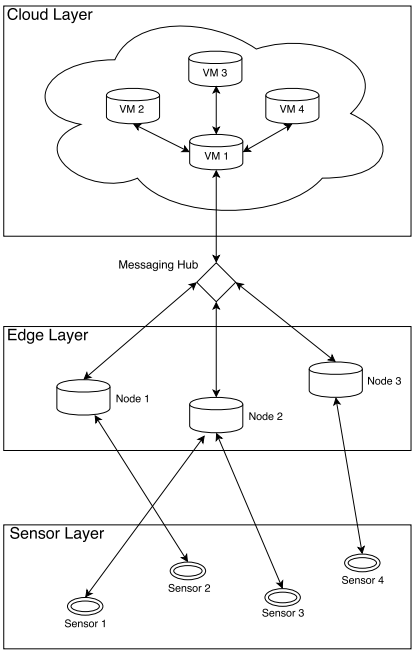


**Fig. 1.** The architecture is composed by 3 layers: Cloud, Edge and Sensor

This layer receives data from queues and exchanges through a message hub, so that the inputs can be parallelized through multiple consumers. It can also be configured to support clusters of machines to execute transformations as distributed stream processing jobs over these queues and exchanges.

**Table 1.** Cloud layer configuration: Virtual machine type and toolset description

| **Parameter** | **Description** |
|---|---|
| Instance Type | Basic_A3 (4 cores, 7 GB of RAM) |
| Operating System | Ubuntu 16.04 LTS |
| Location | Brazil South |
| Golang version | 1.8 |
| GRPC version | 1.3.0-dev |
| Protocol Buffers version | 3.2.0 |
| Network Interconnection | 5Mbps HFC (between Edge and Cloud) |

In our evaluation work, it is implemented as an application running in a single node inside a Linux VM at Microsoft Azure, which was chosen due to its availability at our research lab due to its cost benefits over other cloud platforms [28]. This application was written in Go programming language and receives process-

ing request from the layer below through GRPC communication framework. The VM instance utilized was configured as it is described in Table 1.

### 3.2 Edge layer

The Edge layer is composed by a set of nodes that receive sensor measurements one-at-time and executes its operators. Operators can be either transformations over results, combinations with sets of measurements received or mappings to machines on the Cloud layer above.

On this layer, the application code will be expressed to define which computation will be done inside of edge nodes and which computation will be managed by VMs on the Cloud Computing environment. The degree of control provided by this level makes possible to decrease the number of messages sent to the Cloud. In this way, it is possible to decrease the amount of data that is sent to the Cloud. Also, by processing certain amounts of data directly on the edge nodes, the latency experienced by actuator sensors is in the order of tenths of milliseconds instead of a couple of seconds of cloud processing latencies.

Actuator sensor logic can also be implemented on this layer, in such a way that when a given condition is matched by an edge node, it can trigger actuators on the Sensor layer in order to act on external applications. For example, a consumer can configure its smart grid energy meter to maintain the energy consumption below a certain level during peak cost energy hours. In this way, the smart grid meter can turn off certain machines when the average consumption reaches a certain threshold.

In our evaluation test bed, this layer was composed by a set of Raspberry Pi edge nodes connected to the internet through wireless connection. Each edge node is a complete Linux machine running our application in an ARM architecture. They were configured to communicate with the underlining sensors directly, as well as the Linux VM on the Azure Cloud service. The configuration of our edge nodes is described in details on Table 2.

**Table 2.** Edge layer configuration: Architecture and software description

| **Parameter** | **Description** |
|---|---|
| Number of Edge Nodes | 4 |
| Hardware | Raspberry Pi Zero W |
| Hardware CPU | 1 GHz Single Core CPU |
| Hardware RAM | 512 MB |
| Operating System | Raspbian Jessie Lite 4.4 (Debian Jessie based) |
| Golang version | 1.8 |
| GRPC version | 1.3.0-dev |
| Protocol Buffers version | 3.2.0 |
| Wireless Router | HUMAX HG100R (802.11a/b/g/n) |

### 3.3 Sensor layer

The Sensor layer is represented by a given set of sensors that communicate with the Edge nodes. Ideally, sensors should communicate with Edge Nodes through their available input/output hardware interconnections or lightweight wireless connection such as Bluetooth or LTE networks. However, in order to limit the analysis scope of this work, we only rely on a sensor network dataset that was previously loaded into edge nodes prior to execution of tests.

Smart grid environments rely on specific meters and plugs on households to collect data, which are provided by the energy grid provider or standardized to support only a set of accepted and verified plugs and meters types. The data types generated by these environments also need to respect a certain schema to be shared, aggregated and analyzed by the energy provider companies.

The dataset used to the evaluation of this work is based on the dataset provided by by the 8th ACM International Conference on Distributed Event-Based Systems (DEBS). This conference provides competitions with problems which are relevant for the industry. In the year of 2014, the conference challenge focus was on the ability of Complex Event Processing (CEP) systems to apply on real-time predictions over a large amount of sensor data. For this purpose, household energy consumption measurements where generated, based on simulations driven by real-world energy consumption profiles, originating from smart plugs deployed in households [35]. For the purpose of this challenge, a large number of smart plugs has been deployed in households with data being collected roughly every second for each sensor in each smart plug.

### 3.4 Communication Protocol

Although multiple protocols for communication in IoT systems have been proposed in the recent years, the protocols in use today are still being evaluated and are subject of discussion and standardization initiatives, mainly due to advancements of internet protocols to support mobile and IoT applications. The most widely adopted protocols in use today are, respectively, MQTT [4] and CoAP [6].

One of the most prominent proposals on this area is the HTTP/2 protocol. The standard was finished in 2005 and provides several improvements over previous protocols, mainly due to the capability of multiplexing data, avoiding handshake overhead and their data compression capabilities [5] [29].

As an alternative to broker centric communications protocols and synchronization costly protocols such as REST [27], Google Inc. has adopted a RPC protocol and service discovery framework Stubby/Chubby [8]. The open source version of its tool is called GRPC [18], which relies on HTTP/2 in order to avoid handshake overhead, and Protocol Buffers [16] to communicate using a binary method, which provides better data compaction by reducing the message size.

Due to the performance benefits reported from the usage of HTTP/2 protocols over standard HTTP, and their ease of usage for flexible prototyping of

distributed applications, GRPC was used to build a reliable and fast communication channel for all of the communication layers implemented on this work.

GRPC as a communication platform presents several advantages over TCP only connections and communication protocols such as REST. It does not only hides complexity but also provides connections to keep alive for long periods, avoiding unnecessary handshake communication but also multiplexing several requests inside of the same channel. However, their usage is still subject of evaluation, mainly on networks with high package loss percentages, which are a limiting factor not only for HTTP/2 but also for AMQP based applications [17] [21] [12] [32].

### 3.5 Measurement Algorithm

Smart grids promise to provide better control and balance of energy supply and demand through near real-time, continuous visibility into detailed energy generation and consumption patterns. Methods to extract knowledge from near real-time and accumulated observations are hence critical to the extraction of value from the infrastructure investment.

In this context, Short-Term Load Forecasting (STLF) refers to the prediction of power consumption levels in the next hour, next day, or up to a week ahead. Methods for STLF consider variables such as date (e.g., day of week and hour of the day), temperature (including weather forecasts), humidity, temperature-humidity index, wind-chill index and most importantly, historical load. Residential versus commercial or industrial uses are rarely specified.

Time series modeling for STLF has been widely used over the last 30 years and a myriad of approaches have been developed. These methods [20] can be summarized as follows:

- Regression models that represent electricity load as a linear combination of variables related to weather factors, day type, and customer class.
- Linear time series-based methods including the Autoregressive Integrated Moving Average (ARIMA) model, auto regressive moving average with external inputs model, generalized auto-regressive conditional heteroscedastic model and State-Space Models (SSMs).
- SSMs typically relying on a filtering-based (e.g., Kalman) technique and a characterization of dynamical systems.
- Nonlinear time series modeling through machine learning methods such as nonlinear regression.

Shawkat Ali [2] argues that the three most accurate models for load prediction are, respectively, Multilayer Perceptron (MLP), Support Vector Machine and Least Mean Squares. Due to the model fit in relation to the distributed architecture, we decide to pursue an approach similar to the suggested by the DEBS 2014 conference committee [35], that is schematically described in Equation (1). This approach could be interpreted as a mixed approach between MLP and ARIMA. It brings together characteristics from both Linear time series-based methods and SSMs [11].

More specifically, the set of queries provide a forecast of the load for: (1) each house, i.e., house-based and (2) for each individual plug, i.e., plug-based. The forecast for each house and plug is made based on the current load of the connected plugs and a plug specific prediction model. The aim of these queries is not at the over the better prediction model, but at stressing the interplay between modules for model learning that operate on long-term (historic) data with components that apply the model on top of live, high velocity data.

$$L(s_{i+2}) = \frac{avgL(s_i) + median(avgL(s_j))}{2} \tag{1}$$

In the Equation (1), $avgL(s_i)$ represents the current average load for the slice $s_i$. The value of $avgL(s_i)$, in case of plug-based prediction, is calculated as the average of all load values reported by the given plug with timestamps $\in s_i$. In case of a house-based prediction the $avgL(s_i)$ is calculated as a sum of average values for each plug within the house. $avgL(s_j)$ is a set of average load value for all slices $s_j$ such that:

$$s_j = s_{i+2-n*k} \tag{2}$$

In the Equation (2), $k$ is the number of slices in a 24 hour period and $n$ is a natural number with values between 1 and $floor(\frac{i+2}{k})$. The value of $avgL(s_j)$ is calculated analogously to $avgL(s_i)$ in case of plug-based and house-based (sum of averages) variants.

## 4 Evaluation

The evaluation of our test bed application is made in two phases: Communication evaluation and application evaluation. In each step, we evaluate aspects of the application that build on top of each other to improve our overall processing throughput, such as: Latency, message sizes, concurrency degree and message windows.

### 4.1 Communication evaluation

In order to successfully evaluate our test bed middleware implementation, we first evaluate the underlining network connection. The network evaluation started by measuring the maximum amount of data that could be sent from our edge nodes to the cloud provider, using the specified network router and the given network connection described on Section 3.

Our measurements started by analyzing throughput through the Iperf tool [33]. Those experiments have shown us that the average throughput was slightly below the network bandwidth described by the provider, which is common on internet providers of cable connections and was expected on our measurements [14]. The results of this first measurement step are described on Table 3.

**Table 3.** Network measurements with Iperf

| Connection Type | TCP window size | Interval | Transfer | Bandwidth |
|---|---|---|---|---|
| Edge Node to Edge Node | 43.8 KByte | 60 seconds | 388 MBytes | 54.1 Mbits/sec |
| Edge Node to Cloud | 43.8 KByte | 60 seconds | 7.76 MBytes | 1.03 Mbits/sec |

After have a glimpse about the real throughput of our underlining network, we designed a simple application in order to evaluate the perfomance or the GRPC middleware and the HTTP/2 protocol with varying message sizes.

The simple application designed for this task is called PingPong, which executes the following steps: (1) sends a message from the edge node to the cloud node; (2) the cloud node receives the message and sends it back to the edge node, completing a round-trip.

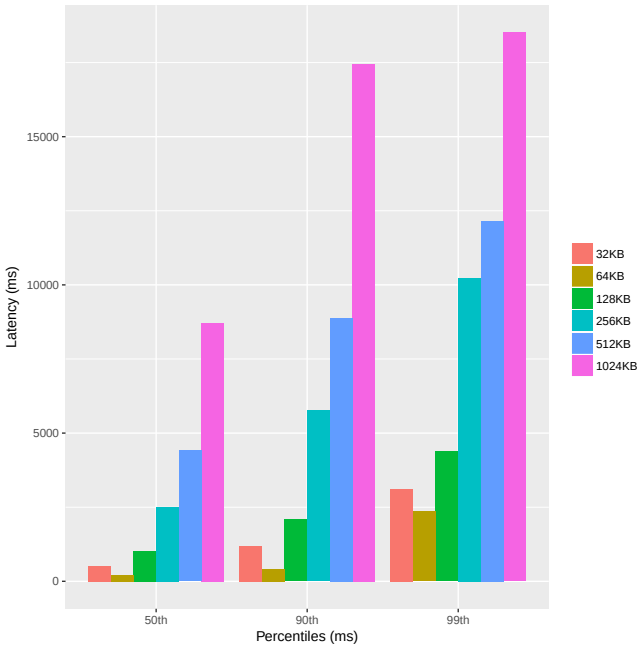


**Fig. 2.** PingPong: Latency Percentiles by Message Sizes (32KB to 1MB)

It is a known fact that distributed applications suffer from latency tails that can be several times greater than the expected average latency. On applications with multiple users that send thousands of messages per second, these fat tail latencies might be experienced by several users of the systems [23] [3].

In the Figure 2 we analyze the impact of message sizes, from 32KB up to 1MB, in the latency of the messages being sent over the network. As we can see, the 50th percentile (the median), is as low as a couple of milliseconds for small messages sizes, but it increases highly when we analyze the tail latencies. The impact of messages that are delayed by Garbage Collection (GC) pauses, package losses or other network failures is a limiting factor depending on the application. These measurements also serve to us a guideline to build message windows, given that we should expect, for example, messages latencies up to 2.5 seconds for 64KB messages at 99th percentile.

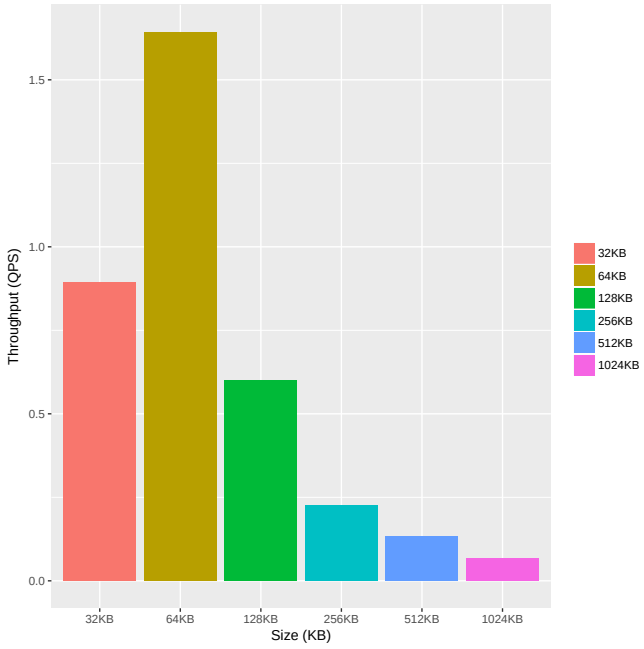


**Fig. 3.** PingPong: Maximum Throughput by Message Size (32KB to 1MB)

We have also used the PingPong method to evaluate the maximum achievable network throughput under our communication middleware, which can be seen on Figure 3. This evaluation was important to understand that the usage of 64KB messages increases the throughput, probably due to a better fit on TCP windows used by the GRPC communication framework.

### 4.2 Application evaluation

The application evaluation was done by distributing the aggregation step of the application processing between edge and cloud nodes. Our main objectives for this evaluation were to understand how latency affects processing throughput,

as well as to analyze the performance gains obtained by aggregating data on edge nodes before sending data to the cloud.

In our evaluations, we preload our edge nodes with necessary sample sizes before running the algorithm. The schema is quite similar to the original dataset and is composed by a timestamp, the value of the energy measurement (in Watts) and an identifier id of the house/plug that is been measured. As it is compressed by the Protocol Buffers binary procotol, the final message payload size is 32 bytes.

Our evaluation can be better described in three phases: (1) Evaluation of the impact of the concurrency degree on the throughput; (2) Scalability evaluation to understand how the number of edge nodes impacts into the cloud node throughput; (3) Windowing and strategies to aggregate data on the edge node before sending data to the cloud node.

**Concurrency evaluation** Given that our edge nodes execute multiple requests per second to its respective cloud nodes, it is important to explore concurrency strategies to obtain performance gains by executing multiple concurrent requests to the remote services.

In the Go programming language, the concurrency execution is done not directly through the creation of threads directly, but through Go's green threads model which are called Goroutines [34].

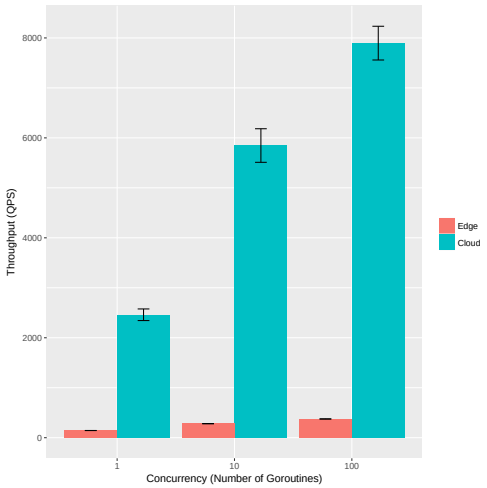


**Fig. 4.** Concurrency Analysis: Impact of Goroutines usage on throughput (Edge and Cloud nodes)

This experiment explored the impact of the Goroutines usage into the throughput the application. The outcome of this experiment suggests that both

edge and cloud nodes are able to benefit from concurrency. Our experiments show that, in comparison with the sequential approach, it is possible to achieve a 4 times speedup on our test bed application by using 100 Goroutines.

**Scalability evaluation** Another important aspect for our application is the ability of our cloud node to being able of aggregate messages from multiple edge nodes. In order to evaluate the scalability of our application, we have maintained a single cloud node and increased the number of edge nodes from one to four. As it is shown on Figure 5, a single cloud node is able to scale linearly up to four edge nodes, each one of them maintaining an average of approximately 500 requests per second.

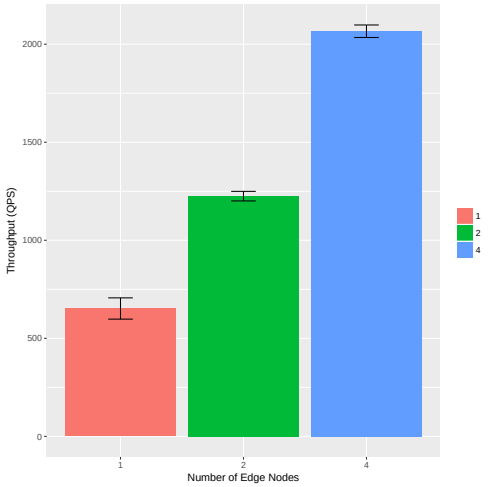


**Fig. 5.** Scalability Analysis: Throughput with multiple consumers (1 to 4 edge nodes)

**Impact of message windowing** Finally, in order to explore our limitations of communication bandwidth and latency, we decided to explore possibilities of aggregating multiple energy measurements into edge nodes before sending data to cloud nodes. Prior to that, our experiments relied on the idea of receiving measurements on edge nodes, sending data to be processed on the cloud and finally receiving the updated forecast for each new measurement.

In our experiments, as it is presented on Figure 6, we have analyzed the behavior of the test bed application when processing locally grouped sets of messages before sending to the cloud. In this way, we could evaluate our assumption that by processing more messages at the edge, and reducing the number of requests to the cloud node, we would be able to improve our overall through-

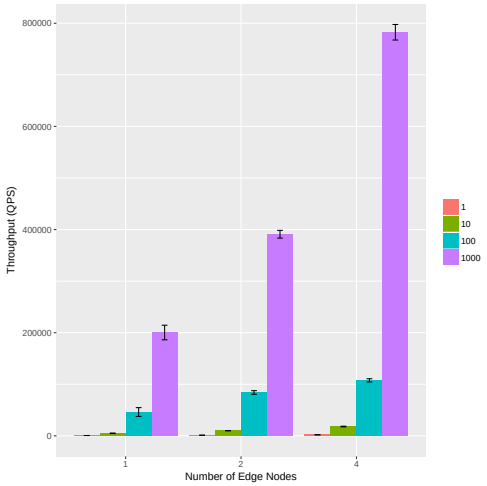


**Fig. 6.** Windowing Analysis: Windowing impact on throughput (1 to 1000 messages per request)

put and increase scalability (by decreasing the number of measurements being processed on the cloud node).

Our results show that the overall application (the combination of cloud and edge nodes) was able to process almost 800k messages per second by using 4 edge nodes and window sizes of 1000 combined messages.

The approach we have used to build windows of processing consists in aggregating the local measurements into a unified representation of the set, so that we could send the grouped view with the same payload size a single message of the window.

## 5 Conclusion and Future Works

In this work, we have studied the implementation of a smart grid application over a mixed cloud and edge processing middleware. This application was able to achieve a higher throughput by leveraging processing on edge nodes and data aggregation to reduce communication with the cloud environment.

In our future works, we would like to explore how the middleware used could be evolved into a generic framework for applications that span processing over edge and cloud. Also, it would be important to study how other communication protocols behave in this given setup.

Furthermore, we would like to explore the possibility of providing a generic method to schedule distributed tasks on this system, as well as to provide operators and potentially a query language to specify those generic data aggregations.

## Acknowledgments

This research received partial funding from CYTED for the RICAP Project.

It has also received partial funding from the EU H2020 Programme and from MCTI/RNPBrazil under the HPC4E project, grant agreement no. 689772.

Additional funding was provided by FAPERGS in the context of the GreenCloud Project.

## References

1. Abdelwahab, S., Hamdaoui, B., Guizani, M., Znati, T.: Replisom: Disciplined tiny memory replication for massive iot devices in lte edge cloud. IEEE Internet of Things Journal 3(3), 327–338 (2016)
2. Ali, A.B.M.S.: Smart Grids: Opportunities, Developments, and Trends. Springer Science & Business Media (2013)
3. Bailis, P., Kingsbury, K.: The network is reliable. Queue 12(7), 20 (2014)
4. Banks, A., Gupta, R.: MQTT Version 3.1. 1. OASIS standard (2014)
5. Belshe, M., Thomson, M., Peon, R.: Hypertext transfer protocol version 2 (HTTP/2). Internet Engineering Task Force (IETF) - RFC-7540 (2015)
6. Bormann, C., Castellani, A.P., Shelby, Z.: CoAP: An application protocol for billions of tiny internet nodes. IEEE Internet Computing 16(2), 62–67 (2012)
7. Brown, R.E.: Impact of Smart Grid on Distribution System Design. In: Power and Energy Society General Meeting-Conversion and Delivery of Electrical Energy in the 21st Century, 2008 IEEE. pp. 1–4. IEEE (2008)
8. Burrows, M.: The Chubby lock service for loosely-coupled distributed systems. In: Proceedings of the 7th symposium on Operating systems design and implementation. pp. 335–350. USENIX Association (2006)
9. Buyya, R., Dastjerdi, A.V.: Internet of Things: Principles and paradigms. Elsevier (2016)
10. Buyya, R., Yeo, C.S., Venugopal, S., Broberg, J., Brandic, I.: Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility. Future Generation computer systems 25(6), 599–616 (2009)
11. Bylander, T., Rosen, B.: A Perceptron-like Online Algorithm for Tracking the Median. In: Neural Networks, 1997., International Conference on. vol. 4, pp. 2219–2224. IEEE (1997)
12. Chowdhury, S.A., Sapra, V., Hindle, A.: Is HTTP/2 more energy efficient than HTTP/1.1 for mobile users? PeerJ PrePrints 3, e1280v1 (2015)
13. Dastjerdi, A.V., Buyya, R.: Fog Computing: Helping the Internet of Things Realize Its Potential. Computer 49(8), 112–116 (2016)
14. Dischinger, M., Haeberlen, A., Gummadi, K.P., Saroiu, S.: Characterizing residential broadband networks. In: Internet Measurement Conference. pp. 43–56 (2007)
15. Gedawy, H., Tariq, S., Mtibaa, A., Harras, K.: Cumulus: A distributed and flexible computing testbed for edge cloud computational offloading. In: Cloudification of the Internet of Things (CIoT). pp. 1–6. IEEE (2016)
16. Gligorić, N., Dejanović, I., Krčo, S.: Performance evaluation of compact binary XML representation for constrained devices. In: Distributed Computing in Sensor Systems and Workshops (DCOSS), 2011 International Conference on. pp. 1–5. IEEE (2011)

17. Goel, U., Steiner, M., Wittie, M.P., Flack, M., Ludin, S.: HTTP/2 Performance in Cellular Networks. In: ACM MobiCom (2016)
18. Google: gRPC Motivation and Design Principles (2015), http://www.grpc.io/blog/principles
19. Habak, K., Ammar, M., Harras, K.A., Zegura, E.: Femto clouds: Leveraging mobile devices to provide cloud service at the edge. In: Cloud Computing (CLOUD), 2015 IEEE 8th International Conference on. pp. 9–16. IEEE (2015)
20. Kyriakides, E., Polycarpou, M.: Short Term Electric Load Forecasting: A Tutorial. In: Trends in Neural Computation, pp. 391–418. Springer (2007)
21. Lee, S., Kim, H., Hong, D.k., Ju, H.: Correlation analysis of MQTT loss and delay according to QoS level. In: Information Networking (ICOIN), 2013 International Conference on. pp. 714–717. IEEE (2013)
22. Liu, P., Willis, D., Banerjee, S.: ParaDrop: Enabling Lightweight Multi-tenancy at the Network's Extreme Edge. In: Edge Computing (SEC), IEEE/ACM Symposium on. pp. 1–13. IEEE (2016)
23. Maas, M., Harris, T., Asanovic, K., Kubiatowicz, J.: Trash day: Coordinating garbage collection in distributed systems. In: HotOS (2015)
24. Orsini, G., Bade, D., Lamersdorf, W.: CloudAware: A Context-Adaptive Middleware for Mobile Edge and Cloud Computing Applications. In: Foundations and Applications of Self* Systems, IEEE International Workshops on. pp. 216–221. IEEE (2016)
25. Pan, J., Ma, L., Ravindran, R., TalebiFard, P.: HomeCloud: An edge cloud framework and testbed for new application delivery. In: Telecommunications (ICT), 2016 23rd International Conference on. pp. 1–6. IEEE (2016)
26. Reuters: U.S. Smart Grid to Cost Billions, Save Trillions (2011), http://www.reuters.com/article/2011/05/24/us-utilities-smartgrid-epri-idUSTRE74N70420110524
27. Richardson, L., Ruby, S.: RESTful web services. " O'Reilly Media, Inc." (2008)
28. Roloff, E.: Viability and Performance of High-Performance Computing in the Cloud. Ph.D. thesis, Universidade Federal do Rio Grande do Sul (2013)
29. Ruellan, H., Peon, R.: HPACK: Header Compression for HTTP/2. Internet Engineering Task Force (IETF) - RFC-7541 (2015)
30. Satyanarayanan, M.: The Emergence of Edge Computing. Computer 50(1), 30–39 (2017)
31. Sun, X., Ansari, N.: EdgeIoT: Mobile Edge Computing for the Internet of Things. IEEE Communications Magazine 54(12), 22–29 (2016)
32. Thangavel, D., Ma, X., Valera, A., Tan, H.X., Tan, C.K.Y.: Performance evaluation of MQTT and CoAP via a common middleware. In: Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 2014 IEEE Ninth International Conference on. pp. 1–6. IEEE (2014)
33. Tirumala, A., Qin, F., Dugan, J., Ferguson, J., Gibbs, K.: Iperf: The TCP/UDP bandwidth measurement tool. http://iperf.fr (2005)
34. Togashi, N., Klyuev, V.: Concurrency in Go and Java: performance analysis. In: Information Science and Technology (ICIST), 2014 4th IEEE International Conference on. pp. 213–216. IEEE (2014)
35. Ziekow, H., Jerzak, Z.: The DEBS 2014 Grand Challenge. In: Proceedings of the 8th ACM International Conference on Distributed Event-based Systems, DEBS. vol. 14 (2014)