Low-Latency Anomaly Detection on the Edge-Cloud Continuum for Industry 4.0 Applications: the SEAWALL Case Study

Lorenzo Bacchiani*, Giuseppe De Palma*, Luca Sciullo*,
Mario Bravetti*, Marco Di Felice*, Maurizio Gabbrielli*, Gianluigi Zavattaro*
Roberto Della Penna†
*Department of Computer Science and Engineering, University of Bologna, Italy
†Bonfiglioli S.P.A., Italy

***Abstract*—Several emerging Industry 4.0 applications related to the monitoring and fault diagnostic of critical equipment introduce strict bounds on the latency of the data processing. Edge computing has emerged as a viable approach to mitigate the latency by offloading tasks to nodes nearby the data sources; at the same time, few industrial case studies have been reported so far. In this paper, we describe the design, implementation and evaluation of the SEAWALL platform for the heterogeneous data acquisition and low-latency processing in Industry 4.0 scenarios. The framework has been developed within the homonymous project founded by the Italian BIREX industrial consortium and involving both academic and industrial partners. The proposed framework supports data collection from heterogeneous production line machines mapped to different IoT protocols. In addition, it enables the seamless orchestration of workloads in the edge-cloud continuum so that the latency of the alerting service is minimized, while taking into account the constrained resources of the edge servers. We evaluate the SEAWALL framework in a small-case industrial testbed and quantify the performance gain provided by the dynamic workload allocation on the continuum.**

***Index Terms*—Industry 4.0, Industrial Internet of Things (IIoT), Edge computing, Web of Things (WoT), Workload orchestration**

I. INTRODUCTION

In Industry 4.0 scenarios, the Internet of Things (IoT) and the cloud computing are considered the key technological enablers of industrial process monitoring and automation [1]. Among the enabled applications, early fault detection of assembly line machines plays a crucial role to minimize operation downtime and to maximize production efficiency. Most of the reported deployments of industrial fault detection systems are based on a cloud-centric approach, i.e. the sensory data collected from the assembly line machines are transferred to remote cloud infrastructures where they are stored and analyzed e.g. by using Machine Learning (ML) tools. While this approach can ensure the service scalability and the possibility to cope with dynamic and massive data flows, it does not fit the requirements of emerging, time-critical industrial applications that introduce strict Quality of Service (QoS) requirements in terms of reliability and latency of the operations. This is the case, for instance, of industrial robots or unmanned/guided vehicles used for transportation of tools and products [2]. How to support low-latency operations in time-critical industry 4.0 scenarios is becoming a major research challenge and pushing the adoption of novel solutions in two complementary domains. On the communication side, standards like Time Sensitive Networking (TSN) and the 5G [3] are envisioned to minimize the data acquisition latency while ensuring the deterministic delivery of messages. On the processing side, edge computing solutions [4] have been largely investigated as a viable solution to reduce the processing latency and to save bandwidth. Generally speaking, edge computing is a relatively new paradigm that attempts to offload computational tasks to devices nearby the IoT data sources: example of edge devices may include micro-controllers (in this case, they are also referred as the extreme edge), microcomputers, Base Stations (BS) or servers, all characterized to be geographically close to sources of data. To do that, edge computing solutions involve multiple components [5]: (*i*) edge *caching*, i.e. techniques allowing to store portions of data directly on the edge devices; (*ii*) edge *intelligence*, i.e. techniques aimed at extracting knowledge from the cached data, often adapting existing ML techniques to be executed on hardware constrained edge devices; and (*iii*) edge *offload*, i.e. the possibility of assign tasks to other devices in case a single edge node does not have enough resource to execute them. Regarding the latter, several recent works enlarge such concept through the emerging paradigm of the edge-cloud continuum, which refers to possibility to seamlessly allocate resources and workloads to edge/cloud or intermediate nodes based on resource utilization or QoS metrics [6]. At the same time, while several components of the continuum, such as the task allocation policies, have been investigated, few works report real-world deployments in industrial scenarios.

In this paper, we attempt to fill such gap by describing the design and implementation of a software architecture for Industry 4.0 scenarios enabling edge-cloud continuum operations of machine monitoring and fault detection. The architecture has been deployed within the SEAmless loW lAtency cLoud pLatforms (SEAWALL) project founded by the BIREX[1] consortium, a competence center for the Industry 4.0 recently established in Bologna, Italy. The project involved

[1]https://bi-rex.it/en/

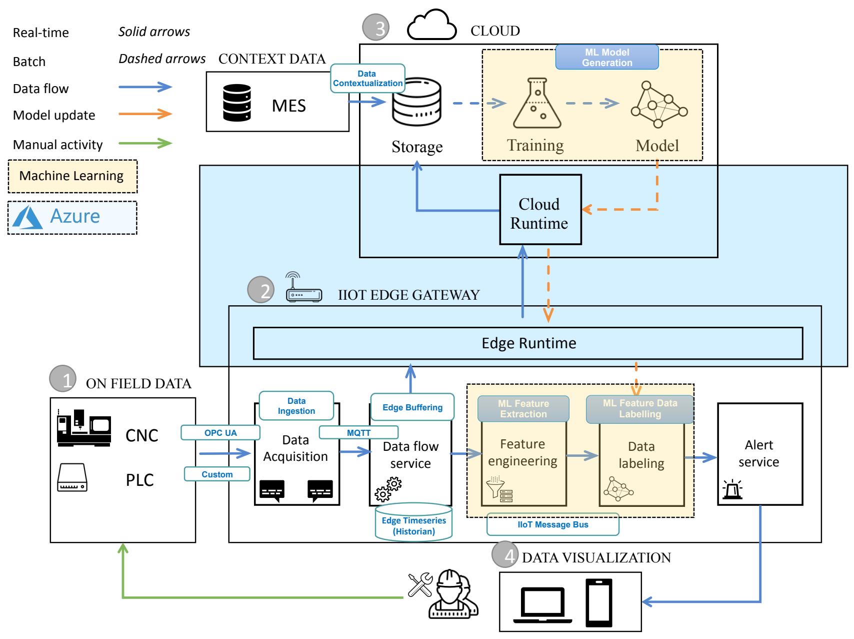


Fig. 1: Previous architecture of the case study.

academic researchers and companies, the latter being technical services providers or end-users (i.e., manufacturing companies interested in the digitalization of industrial processes). More in detail, the proposed architecture allows to address two main requirements of Industry 4.0 scenarios, respectively the data heterogeneity support and the time-aware data processing support. Regarding the first, the SEAWALL framework employs the emerging Web of Things (WoT) standard promoted by the World Wide Web Consortium (W3C) to design effective data acquisition strategies from heterogeneous production lines, by abstracting from the underlying IoT protocols. In this way, different machines can be easily integrated into our framework, decoupling the data processing from the acquisition. Regarding time-aware data processing, the SEAWALL framework enables the dynamic orchestration of workloads among the nodes of the edge-cloud continuum. The data processing services can be dynamically activated/deactivated on different edge/cloud nodes so that the overall latency is minimized while the edge servers are not overloaded. To this aim, we validate our platform in a use-case proposed by an industrial automation company (Bonfiglioli S.P.A.). The use case focuses on the seamless allocation of a Data Alerting (DA) service - in charge of running anomaly detection algorithms over the data of a production line - between edge and cloud nodes. We evaluate different policies for the DA allocation by varying the amount of data generated by the production line and measuring the corresponding latency to generate the alarm. In [7], a preliminary description of the SEAWALL architecture has been presented. Here, we extend such work with implementation details and experimental results. .
The rest of the paper is structured as follows. In Section II we review existing edge-cloud workload allocation schemes for the IoT. The industrial use case within the BIREX project is described in Section III. Section IV introduces the SEAWALL architecture. The implementation of the SEAWALL modules is detailed in Section V. Experimental results for a small-case industrial testbed are reported in Section VI. Conclusions and future works are in Section VII.

II. Related Works

IoT applications are composed of multiple, interacting components enabling the storage, processing, and valorization of sensory data as well as the actuation on the target environment. For this reason, the micro-service patterns have emerged as a viable approach to decompose an IoT application into a set of loosely coupled services [6]; the latter can be containerized via virtualization techniques such as Docker[2] and deployed at different nodes of the compute continuum thanks to orchestration services such as Kubernetes[3]. The question of how the continuum can help meet the QoS requests is a novel yet investigated topic for generic micro-service-based applications; at the same time, few studies refer to the IoT or to Industry 4.0 scenarios. Efficient workload allocation is the major concern in most of these papers. In [8] the authors assume that an IoT application can be modeled as a set of micro-services (called Processing Elements) forming a Directed Acyclic Graph (DAG); hence, they formulate the Processing Elements (PE) scheduling problem where the goal is to minimize the latency of the whole PE workflow. Similarly, the framework in [9] attempts to allocate Web Things (WTs) to nodes of the continuum by taking into account the interdependencies among the WTs so that the in-network overhead

[2]https://www.docker.com
[3]https://kubernetes.io

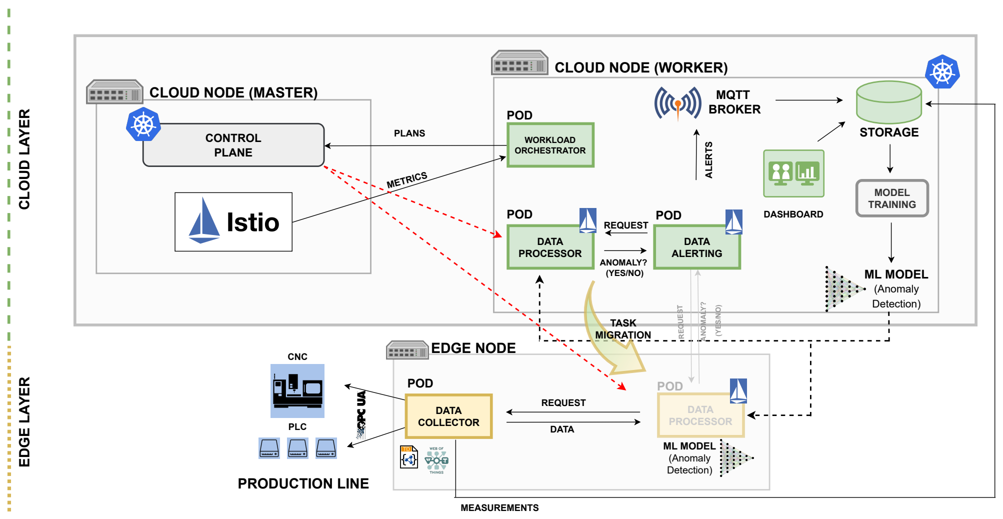


Fig. 2: The proposed cloud/edge architecture and the enabling technologies.

is minimized. The placement of components must find the optimal trade-off between conflicting requirements in terms of QoS requests from users/applications and unpaired resource and cost availability on edge and cloud nodes. For these reasons, multi-objective workload allocation strategies have been proposed like in [10] where the placement framework attempts to jointly minimize the service completion time, the energy computation and communication consumption, and the storage and communication cost. Other recent studies raise the concern about the dynamic variation of metrics used for allocation policies, such as execution time or CPU utilization, which can vary significantly over time or on different nodes of the continuum, and, for this reason, they employ ML-based solutions to learn workload patterns [11]. In the computing continuum environment proposed in [12], each edge cluster contains a scheduler node in charge of receiving requests from clients and of taking decisions whether to execute the task locally, on the cloud or to reject it; the decision is based on a Reinforcement Learning (RL) engine which receives a positive reward for every task completed within a deadline. All the previous studies assume a global IoT workload balance. Vice versa, [13] proposes a decentralized load-balancing system for IoT service placement which aims at reducing the cost of service execution by enabling each edge/cloud node to generate a predefined number of possible service placement plans. Finally, a further issue to consider is how to implement the workload allocation. In allocation-only schemes, containers are switched on/off at different nodes but no software mobility occurs. Vice versa, migration-based strategies enable container transfer among nodes of the continuum, in a stateless or stateful way as discussed in [14].

III. Industry 4.0 Use Case

he SEAWALL platform was designed as a solution to a case study proposed by Bonfiglioli S.P.A, a world leader manufacturer in industrial automation, mobile machinery, and wind energy. It concerns, see Figure 1, an edge-cloud continuum system to control and perform anomaly detection over CNC (Computer Numerical Control) or PLC (Programming Logic Controller) machines. Such machines produce various kinds of data, transmitted following specific protocols like OPC UA, describing their current working state. The data must be stored to perform data analysis, including the production of anomaly detection models, by exploiting ML techniques. Due to the heavy amount of storage and of computational resources needed to store the data and to train the ML models, this part of the system is expected to be deployed in the cloud. The ML models are periodically re-trained by considering more recent data, and are then used within a control-loop that, due to low-latency constraints, is expected to be deployed on the edge. The control loop is triggered by an alert service that periodically uses the ML model to classify the most recent data acquired from the production line machines and the outcomes of the predictions are communicated to the machine operators. The use-case cannot be considered a strict real-time IoT system like the robotics/unmanned vehicles applications mentioned in the Introduction, due to the usage of ML techniques (which are generally delay-tolerant) and of best-effort networking solutions, and of the nature of the system output, which is mainly informative and does not involve automatic actions. However, there is still the need of minimizing the latency of the data alerting process for the human operators. The realization of the system has been developed by adopting edge-cloud solutions offered and managed by a public cloud provider.

This system has been already successfully experimented on one machine. Nevertheless, the company found out that the large-scale adoption of the current architecture is not possible for several limitations. We mention below some of such limitations:

- a unique ML classifier deployed in the edge cannot manage an intensive flow of data, possibly produced in

parallel by several machines connected to the same edge gateway;
- the implementation of several independent control loops (one for each machine, or one for each flow of data possibly generated by different sensors in the same machine) cannot be managed due to the limited edge storage and parallel computing resources;
- the current system does not support dynamic automatic scaling of the computing resources at the edge level;
- the adoption of an edge-cloud solution managed by public cloud providers imposes further limitations; in particular, it is not possible to dynamically migrate components from edge to cloud, and vice versa, depending on customer-defined load-balancing rules.

In the next Sections, we will present the design, implementation, and experimental validation of an alternative architecture that attempts to overcome the serious limitations of the existing deployment.

IV. Architecture

The architecture proposed in Figure 2 includes two main layers, i.e., the *cloud* and *edge* layers and it is micro-service oriented, with each macro-functionality that has been mapped to a single component. More in detail, at the edge layer we find all services in charge of producing and collecting data. Data is generated by a *production line*, that can be considered as the extreme edge layer of the architecture, and that consists of a set of industrial machineries equipped with heterogeneous sensors monitoring the working conditions of the machineries themselves. We mapped the entire production line to a W3C Web Thing to take benefit of the new W3C Web of Things (WoT) standard [15] in terms of interoperability. As a result, we abstract in this paper from the specific machinery in use and from the sensor types, since new kind of machineries can be easily added by providing the WoT mapping interface. An edge node, close to the data source, hosts two services, a static one -the *Data Collector* (*DC*)- for gathering data through the OPC UA protocol, and a dynamic one -the *Data Processor* (*DP*)- that instead is moved between the cloud and edge layers depending on the specific needs. The *DP* can be invoked to retrieve the latest measurements from the *DC* and to execute anomaly detection for discovering a possible misbehavior of the production line. The detector is constituted by a pre-trained Machine Learning (ML) model. In this paper, we do not propose any new ML technique for anomaly detection, since the focus is on where to allocate the workload of the processing task more than on the algorithm itself. As a result, the SEAWALL platform is general enough to support multiple anomaly detection algorithms, assuming that the code for training and inference is provided and properly connected to the DP interface. In the cloud layer we find the *Data Alerting* (*DA*) service, which is in charge of periodically triggering the *DP* in order to get in output the response of a possible anomaly detection. Furthermore, the cloud nodes host: (*i*) a storage service that acts as data lake of alerts and raw sensory data of the production line; (*ii*) an IoT dashboard showing the current machineries conditions and triggered alarms and (*iii*) and the workload orchestrator service, whose task is to monitor whether the *DP* has to be moved between cloud and edge, for instance when specific latency conditions have been exceeded. The dotted lines in the Figure highlight the possibility of re-training the ML model when new data available are available on the cloud, and of transferring back the updated parameters to the *DP* service. We note that under normal operations, the anomaly detector is executed on the cloud given the higher availability of computational resources than the edge node. However, different policies can be easily implemented and deployed in the workload orchestrator in order to adapt the entire system to the specific use case.

V. Implementation

In this Section we briefly present the main technologies used for the implementation of the architecture described in Section IV. The *production line* has been mapped to a Thing Description (TD) in order to be exposed as a Web Thing by the *DC* service; the latter communicates with the physical machineries through OPC UA although it may acquire their data through any WoT-compliant protocol. For this purpose, we used *Node-wot*[4], the official NodeJS WoT framework that is able to consume and produce a Web Thing adhering to the W3C WoT Scripting API indications[5]. Both the *DP* and the *DA* services are written in Typescript and run over the NodeJS runtime[6]. In particular, they expose some REST API for the data interaction. The MQTT broker is the open-source Eclipse *Mosquitto*[7] tool, while the storage system is constituted by an *InfluxDB*[8] instance. Finally, the dashboard module is an Angular web application. The whole architecture has been implemented through containerization, i.e., each service is a Docker container managed through a Kubernetes controller running on the Master node. We continuously monitor the performance of the *DA* service through the Istio framework[9], in terms of latency of the alert generation: the latter includes both the latency for data retrieval (from the *DP* module) and the latency of the ML model execution. These metrics are then used by our custom *Workload Orchestrator* service to decide where to allocate the *DP*, i.e., whether on the cloud or edge layer. The current policy offloads the *DP* service to the edge node when the latency of the alert generation becomes higher than a threshold tunable by the system administrator through a simple dashboard. The output of the *Workload Orchestrator* is implemented by the Kubernetes Control Plane by activating/deactivating PODs at different nodes, without any active code migration.

VI. Performance Evaluation

We test the effectiveness of our edge-cloud continuum approach using the settings described in Table I: we deploy

[4]https://github.com/eclipse/thingweb.node-wot
[5]https://www.w3.org/TR/wot-scripting-api/
[6]https://nestjs.com/
[7]https://mosquitto.org/
[8]https://www.influxdata.com/
[9]https://istio.io/

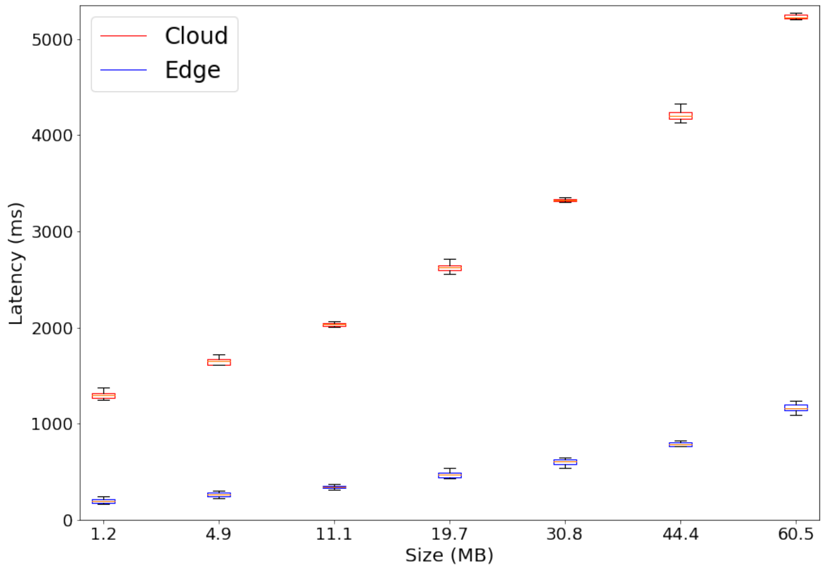


Fig. 3: Latency-Size relation for *DP* in cloud or in edge.

a Kubernetes cluster composed of 3 nodes, provisioned by Digital Ocean, each one with $2vcpu$ and $4gb$. More precisely, we have 2 nodes situated in London, one of which is the master node with the Kubernetes control plane and the other is a worker node embodying the cloud layer. The third node is situated in San Francisco as the edge layer. To evaluate the performance of our architecture, we place the *DA* in the cloud layer, the *DC* in the edge layer, and allow the placement of the *DP* either on the edge or in the cloud.

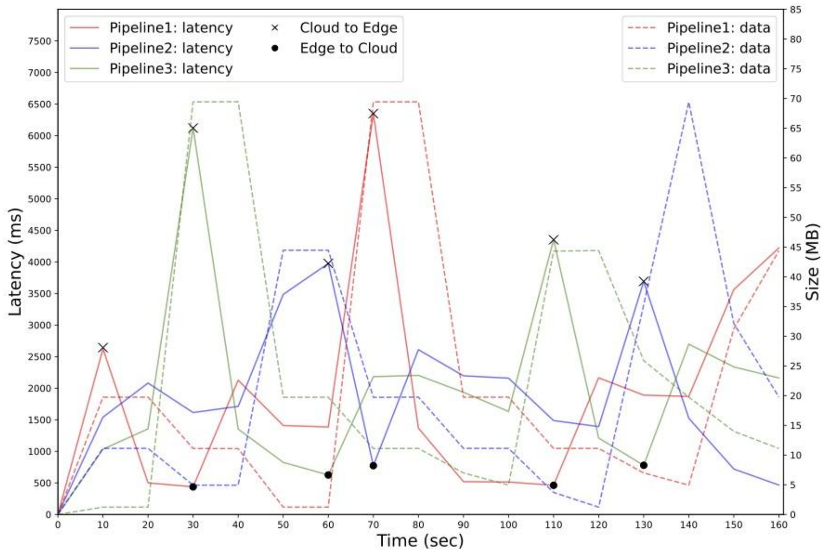
Nodes	3
Node locations	London, San Francisco
Node resources	$2vcpu$ with $4gb$ RAM
Request payload	3500 B
Workload size	x*x*3500 with $x \in$ [20 140] B
Request interval	3s
Monitoring interval	10s

TABLE I: Experimental settings.

The *DA* has a request interval of 3 seconds, while the workload orchestrator has a monitoring interval of 10 seconds. The workload used to test our architecture is composed of synthetic data reproducing the same pattern of real data used in Bonfiglioli S.P.A.

The first experiment aims at evaluating the performances of the *DP* on edge, and in the cloud, under different amounts of data sent to the *DP* by the *DC*. As possible amounts of data, we pick a value x from the interval $[20, 140]$ and then compute the actual size as $x*x*3500$ bytes (3500 is the original size of the payload of a single request). We consider seven possible values for x, starting from 20 and increasing it by 20 for six times. For each value of x we execute ten runs in which we perform requests to the *DC*, and we depict in Figure 3 the measured latencies with a box plot, indicating the minimal, maximal, and median latency, and latency variance. The experimental data show that, despite the workload size grows, the increment in latency with the *DP* in the edge layer is very limited, while for the *DP* service fixed in the cloud layer the performance rapidly degrades, as the workload size increases. These two behaviors follow from the different network latencies for the communications between the *DP* and the *DC*: communication is obviously faster when they are both placed on the edge (hence physically in the same data center in San Francisco) while it is slower when they are placed in the two distinct edge and cloud layers (hence 1 in San Francisco and 1 in London). This confirms the expected advantages of the data locality principle, i.e. moving services instead of data.

In the light of this initial observation, we decided to consider a realistic pipeline in which the size of data sent by the *DC* varies over time, and the *DP* is moved from cloud to edge, and vice versa, depending on the actual latency/size of the data. The variation of the size of data has been generated by randomly picking a value $x \in [20, 140]$, and then computing the actual size as described above. The workload orchestrator keeps the *DP* service in the cloud until the latency is under a 2 seconds threshold. When it increases, the service is moved to the edge layer and kept there as long as the size is above a threshold resulting from a combination of latency and size: the *DP* service can return in the cloud only if *latency* $\leq 1s$ and *request size* $<$ 11MB. We extract the request size value from Figure 3: a size of 11MB is related to a latency of 2 seconds for the *DP* in the cloud layer. We executed 35 runs of this single pipeline example, which confirm there is a low variance of performance. In Figure 5, we present 4 runs that we consider particularly significant as they follow the 2 patterns observed in our experiments. These patterns execute the same mobility operations except for $t = 140s$: *Run* $1 - 2$ don't swap (they swap at $t = 150s$), while the others swap. Figure 5 also shows that our mobility has no overhead on performance: it captures



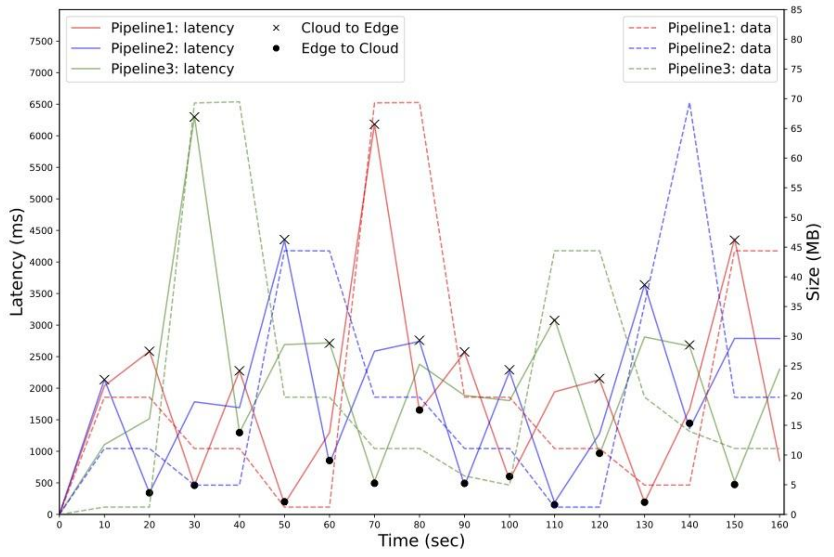


Fig. 4: Multiple pipeline mobility (latency-based policy above, size-based one below).

the best from edge computing (low latency despite big request sizes) and cloud computing (no waste of edge resources).

The final experimented scenario considers multiple independent pipelines. When multiple services are deployed to the edge layer, there is the risk to overload the edge nodes, meanwhile the cloud nodes can be more easily added to the cluster thanks to cloud elasticity. For this reason, in such a scenario it is interesting to evaluate workload orchestrator policies that deploy in edge only a few *DP* services, while keeping in the cloud the remaining ones. To evaluate the flexibility of our architecture to deal with a multiple pipeline scenario, we have considered 3 independent pipelines, following three distinct randomly generated profiles for the workload size, under the constraint that at most 1 *DP* service can run in the edge. We start with the 3 *DP* services in the cloud layer and every 10 seconds the latency and workload size are queried. We compare two possible workload mobility policies: the first one, in each step, moves the service with the highest latency to the edge layer, meanwhile the second policy moves the service with the highest workload size instead. An analysis of both policies is presented in Figure 4. The first policy, as shown on the left graph, incurs the problem of swapping *DP* services at each step. In step t_n the DP_i service with the highest latency is chosen and it is moved to the edge, and in the next step t_{n+1}, DP_i will have the lowest latency, benefiting from the data locality principle, therefore another DP_j with $i \neq j$ (currently in the cloud layer) will be chosen and swapped with DP_i. For example, at $t = 40s$ the orchestrator moves DP_3 to the cloud despite the fact that it has the highest request size (and consequently the highest latency if kept in the cloud). The second policy is shown in the right graph and, being based on the workload sizes, is more stable and incurs fewer swaps.

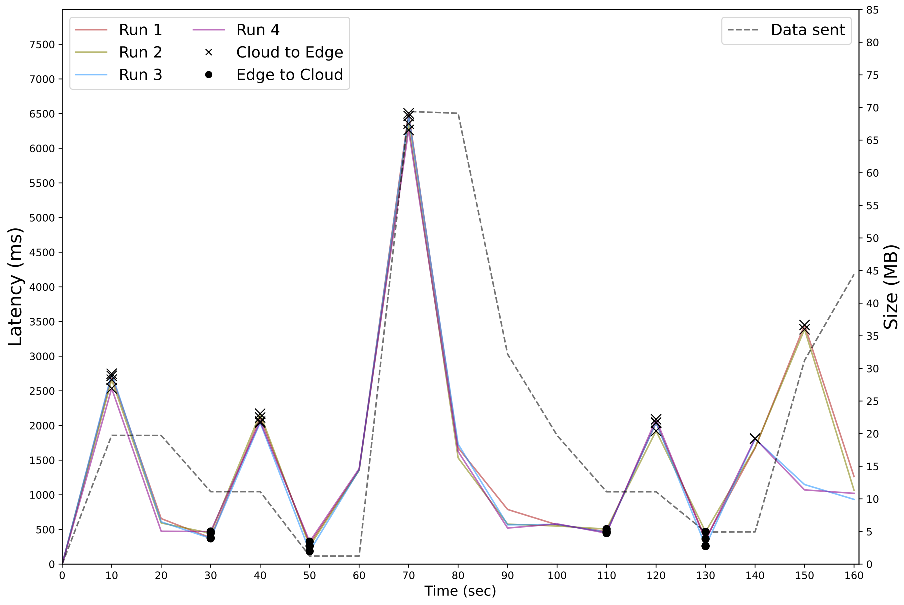


Fig. 5: Single pipeline mobility

More precisely, the service with the highest workload size which is greater than 11MB is placed in the edge node. The policy can avoid swaps when no service satisfies the condition. Comparing the two policies, as can be seen in Table II, the workload size-based one effectively halves the number of swaps, reacting only when the workload significantly increases for a pipeline. Both policies react to the latency peaks and try to improve the performance of the system resulting in a similar latency flow.

	Latency-based	Sized-based
Pipeline 1: swaps	11	4
Pipeline 1: avg latency	1.8s	1.8s
Pipeline 2: swaps	10	3
Pipeline 2: avg latency	1.8s	1.8s
Pipeline 3: swaps	8	4
Pipeline 3: avg latency	2s	1.9s

TABLE II: Trade-offs between the two policies.

A graph comparing the average performance of these two policies is shown in Figure 6. Both policies generally incur the same average latency trends with the latency-based one performing slightly worse.

VII. Conclusion and Future Works

We have presented a micro-service-based architecture, to be deployed over the edge-cloud continuum, that was designed on purpose to address the limitations of an Industry 4.0 application, provided us by one of the industrial partners of the SEAWALL project. Our novel architecture, implemented by adopting open standards and open technologies, solves the most critical limitations of the previous implementation, e.g., it can support heterogeneous sources of data by adopting the W3C Web of Things (WoT) standard, instead of considering a unique classifier deployed in the edge it can support multiple classifiers (hence also several independent control-loops) to be placed in the edge or in the cloud layers, it allows for the easy customization of policies for the dynamic mobility of workloads between the edge and cloud layers, it allows for the sharing of resources among multiple independent pipelines for the analysis of data, and it avoids the typical problems deriving from the adoption of IoT edge-cloud solutions offered and managed by public cloud vendors, like limitations in the customization of the system or vendor lock-in.

We are currently planning the transfer to the production line of our architecture, which successfully passed our tests on synthetic data. As future work, we will follow at least two lines of extension. On one hand, we will enrich our tests by considering different sources of synthetic data, possibly located on distinct nodes/regions. In such a setting, we will have to implement a topology-aware workload orchestrator, able to appropriately move the services closer to their sources of data. On the other hand, we will enrich the workload mobility policies supported by our system, by considering the possibility to use machine learning techniques to anticipate increments/decrements of the amount of generated data, thus being able to anticipate also the mobility of the involved services towards the data sources.

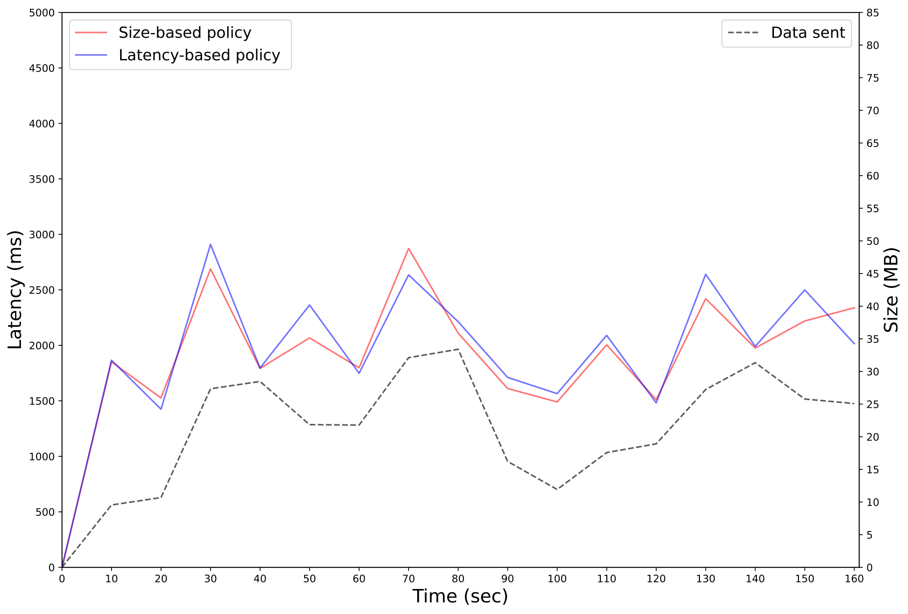


Fig. 6: Performance comparison between latency-based and workload size-based policies.

ACKNOWLEDGEMENTS

This work has been funded by the SEAWALL (SEAmless loW lAtency cLoud pLatforms) project, founded by the BIREX consortium (https://bi-rex.it/en/).

REFERENCES

[1] H. Xu, W. Yu, D. Griffith, and N. Golmie, "A survey on industrial internet of things: A cyber-physical systems perspective," *IEEE Access*, vol. 6, pp. 78 238–78 259, 2018.

[2] A. Fellan, C. Schellenberger, M. Zimmermann, and H. D. Schotten, "Enabling communication technologies for automated unmanned vehicles in industry 4.0," in *2018 International Conference on Information and Communication Technology Convergence (ICTC)*, 2018, pp. 171–176.

[3] M. K. Atiq, R. Muzaffar, Seijo, I. Val, and H.-P. Bernhard, "When ieee 802.11 and 5g meet time-sensitive networking," *IEEE Open Journal of the Industrial Electronics Society*, vol. 3, pp. 14–36, 2022.

[4] T. Qiu, J. Chi, X. Zhou, Z. Ning, M. Atiquzzaman, and D. O. Wu, "Edge computing in industrial internet of things: Architecture, advances and challenges," *IEEE Communications Surveys Tutorials*, vol. 22, no. 4, pp. 2462–2488, 2020.

[5] D. Xu, T. Li, Y. Li, X. Su, S. Tarkoma, T. Jiang, J. Crowcroft, and P. Hui, "Edge intelligence: Architectures, challenges, and applications," *arXiv: Networking and Internet Architecture*, 2020.

[6] J. Santos, T. Wauters, B. Volckaert, and F. De Turck, "Towards low-latency service delivery in a continuum of virtual resources: State-of-the-art and research directions," *IEEE Communications Surveys Tutorials*, vol. 23, no. 4, pp. 2557–2589, 2021.

[7] L. Bacchiani, G. De Palma, L. Sciullo, M. Bravetti, M. De Felice, M. Gabbrielli, G. Zavattaro, R. Della Penna, C. Iorizzo, A. Livaldi, L. Magnotta, and M. Orsini, "Seawall: Seamless low latency cloud platforms for the industry 4.0," in *2022 5th Conference on Cloud and Internet of Things (CIoT)*, 2022, pp. 90–91.

[8] A. Karamoozian, A. Hafid, and E. M. Aboulhamid, "On the fog-cloud cooperation: How fog computing can address latency concerns of iot applications," in *2019 Fourth International Conference on Fog and Mobile Edge Computing (FMEC)*, 2019, pp. 166–172.

[9] C. Aguzzi, L. Gigli, L. Sciullo, A. Trotta, and M. Di Felice, "From cloud to edge: Seamless software migration at the era of the web of things," *IEEE Access*, vol. 8, pp. 228 118–228 135, 2020.

[10] D. Kimovski, N. Mehran, C. E. Kerth, and R. Prodan, "Mobility-aware iot applications placement in the cloud edge continuum," *IEEE Transactions on Services Computing*, pp. 1–1, 2021.

[11] S. Tuli, S. Ilager, K. Ramamohanarao, and R. Buyya, "Dynamic scheduling for stochastic edge-cloud computing environments using a3c learning and residual recurrent neural networks," *IEEE Transactions on Mobile Computing*, vol. 21, pp. 940–954, 2022.

[12] G. P. Mattia and R. Beraldi, "Leveraging reinforcement learning for online scheduling of real-time tasks in the edge/fog-to-cloud computing continuum," in *2021 IEEE 20th International Symposium on Network Computing and Applications (NCA)*, 2021, pp. 1–9.

[13] Z. Nezami, K. Zamanifar, K. Djemame, and E. Pournaras, "Decentralized edge-to-cloud load balancing: Service placement for the internet of things," *IEEE Access*, vol. 9, pp. 64 983–65 000, 2021.

[14] C. Puliafito, A. Virdis, and E. Mingozzi, "Migration of multi-container services in the fog to support things mobility," in *2020 IEEE International Conference on Smart Computing (SMARTCOMP)*, 2020, pp. 259–261.

[15] M. Kovatsch, R. Matsukura, M. Lagally, T. Kawaguchi, K. Toumura, and K. Kajimoto, "Web of Things (WoT) Architecture," W3C Recommendation, Apr. 2020, https://www.w3.org/TR/wot-architecture/.

Lorenzo Bacchiani got his M.Sc. degree in Computer Science at the University of Bologna in 2020. He is now a Ph.D. student in Computer Science and Engineering in the same university. He is currently working on behavioral based approaches and languages for component interaction, adaptation and deployment..

Giuseppe De Palma Giuseppe De Palma received his M.Sc. degree in Computer Science at the University of Bologna in 2020. He is now a Ph.D. student in Computer Science and Engineering in the same university and he is active in the Cloud Computing and DevOps fields, with a focus on Serverless Computing.

Luca Sciullo is a postdoctoral researcher with the University of Bologna, Italy, where he also obtained his Ph.D. in Computer Science and Engineering in the 2021 and his Master Degree (summa cum Laude) in Computer Science in 2017. He was a Visiting Researcher at the Huawei European Research Center of Munich, Germany. He is part of the IoT Prism Lab directed by Prof.Marco Di Felice and Prof. Luciano Bononi. His research interests include wireless systems and protocols for emergency scenarios, wireless sensor networks, IoT systems and Web of Things.

Mario Bravetti is a Full Professor at the Computer Science and Engineering Department of University of Bologna. He is also permanent member of the FOCUS (FOundations of Component-based Ubiquitous Systems) team which is part of the INRIA Sophia Antipolis - Méditerranée French research center. He is PhD in computer science and winner of the award for the two best Italian PhD theses in theoretical computer science in the year 2002, assigned by the Italian Chapter of the European Association for Theoretical Computer Science. His research activity spans from formal description and verification of distributed systems based on concurrency and probability theory to more applicative topics such as service oriented and cloud computing.

Marco Di Felice is a Full Professor of Computer Science with the University of Bologna, where he is the co-director of the IoT PRISM laboratory. He received the Laurea (summa cum laude) and Ph.D. degrees in computer science from the University of Bologna, in 2004 and 2008, respectively. He was a Visiting Researcher with the Georgia Institute of Technology, Atlanta, GA, USA and with Northeastern University, Boston, MA, USA. He authored more than 120 papers on wireless and mobile systems, achieving three best paper awards for his scientific production. He is Associate Editor of the IEEE Internet of Things journal. His research interests include self-organizing wireless networks, unmanned aerial systems, IoT, WoT, and context-aware computing.

Maurizio Gabbrielli is professor of Computer Science since 2001 at the Department of Computer Science and Engineering of the University of Bologna, University of Bologna. He is also Associate dean for AI and digital soul and director of the Master in Management of Digital Technology at Bologna Business School, member of the INRIA project team FOCUS. He received his Ph.d. in Computer Science in 1992 from the Univeristy of Pisa. He has been employed at CWI (Amsterdam), at the University of Pisa and at the University of Udine. He is author of more than 100 publications in the fields of programming languages and artificial intelligence and he has been Director of the European EIT Digital Doctoral School from J2015 to 2017. He has also been: president of the Italian association for Logic Programming (GULP); member of the advisory board of TPLP; member of the Executive Committee of the Association for Logic Programming; chair of the Steering Committee of the ACM conference PPDP; member of the board of the European association for Programming Languages and Systems.

Gianluigi Zavattaro is a full professor in computer science at the University of Bologna, where he obtained his PhD in computer science in 2000. His main research interests are on concurrency theory and verification of concurrent and distributed programs. He is co-author of more than 150 publications, and he was invited speaker at several international conferences like CONCUR'15, ESOCC'13 and FORTE'10. He is currently the chair of the steering committee of the International Federated Conference on Distributed Computing Techniques, and he served as chair of the Program Committee of international conferences like Microservices'20, ECOWS'11, CONCUR'09, and COORDINATION'08.

Roberto Della Penna was born in Vasto, Italy, in 1995. He received the bachelor's degree in computer engineering in 2017 and the master's degree in computer engineering in 2020, both from the Unversity of Bologna. Since 2020 he works in Bonfiglioli S.p.A. as Machine Learning Engineer. He is responsible for developing discrete process models of anomaly detection for predictive maintenance of machine tools and building pipelines for automating training and deployment.