Sánchez Gallegos, D., Carrizales Espinoza, D. Reyes Anastacio, H., González Compean, J.L., Carretero Pérez, J., Morales Sandoval, M., Galaviz Mosqueda, A. (2020). From the edge to the cloud: A continuous delivery and preparation model for processing big IoT data. *Simulation Modelling Practice and Theory*, 105, 102136

DOI: 10.1016/j.simpat.2020.102136

From the edge to the cloud: A continuous delivery and preparation model for processing big IoT data

Dante D. Sánchez-Gallegos[a], Diana Carrizales-Espinoza[a], Hugo G. Reyes-Anastacio[a], J. L. Gonzalez-Compean[a], Jesus Carretero[b], Miguel Morales-Sandoval[a], Alejandro Galaviz-Mosqueda[c]

[a] *Cinvestav Tamaulipas, Cd, Victoria, Mexico*
[b] *Universidad Carlos III de Madrid, Madrid, Spain*
[c] *CICESE, Ensenada, Mexico*

Abstract

This paper presents a processing model for big IoT data. The model includes a continuous delivery scheme based on building blocks for constructing software pipelines from the edge to the cloud. It also includes a data preparation scheme based on parallel patterns for establishing, in an efficient manner, controls over the production and consumption of IoT data. This scheme adds data properties such as cost-efficiency storage, security, and reliability, which are useful to avoid alterations in data and repudiation situations as well as to mitigate risks still arisen in the cloud such as confidentiality violations and service outages. An overlay structure, including planes such as Pub/Sub, control, and preservation, integrates the proposed schemes into software pipelines. The proposed model was developed in both prototype and simulator of software pipelines. Case studies were conducted based on pipeline services deployed from the edge, passing from the fog to the cloud for processing and managing real climate data repositories, which were produced by three different data sensor sources, such as ground stations deployed on Mexico and Spain, as well as small distributed IoT devices. Information sharing patterns for end-users to retrieve raw and/or processed IoT data were also studied. The experimental evaluation revealed the feasibility of using continuous delivery scheme to create dataflows from the edge to the cloud, the efficacy of the overlay structure to create information sharing patterns, as well as the efficiency of data preparation schemes and parallel patterns to improve the end-user service experience in comparison with traditional state-of-the-art solutions.

Keywords: Cloud computing, big IoT data, fog computing, edge computing, data preparation.

Email addresses: dante.sanchez@cinvestav.mx (Dante D. Sánchez-Gallegos), diana.carrizales@cinvestav.mx (Diana Carrizales-Espinoza), hugo.reyes@cinvestav.mx (Hugo G. Reyes-Anastacio), joseluis.gonzalez@cinvestav.mx (J. L. Gonzalez-Compean), jcarrete@inf.uc3m.es (Jesus Carretero), miguel.morales@cinvestav.mx (Miguel Morales-Sandoval), agalaviz@cicese.edu.mx (Alejandro Galaviz-Mosqueda)

1. Introduction

The production of IoT devices has observed a dramatic increment over the few years according to reports [1]. The industry even is projecting that around 28 billion IoT devices will be connected/installed by 2021 [2]. Thus, the volume of data produced by IoT devices that is managed by organizations has incremented dramatically over the past years, which is producing a data accumulation effect [3]. In real scenarios, this trend results in a *big IoT data* processing environment where large repositories of data continuously produced by IoT devices (volume) are processed by multiple procedures (variety) to obtain useful information used as input (value and veracity) in critical decision-making processes (velocity) [4]. Traditionally, cloud computing [5] has been a support for big data scenarios [6] and the most popular solution to store and process data from IoT devices [7]. However, as the systems scale-out and the data amount increases in big data scenarios, a centralized data collection and processing is unfeasible.

To alleviate this problem edge, fog, and cloud computing paradigms have been deployed as a hierarchical approach for the IoT data management, which avoids the saturation of cloud nodes and improves response time in IoT scenarios. The edge computing paradigm considers a collection of technologies where general-purpose computing is usually interconnected directly with sensors devices [8]. Following this approach, IoT devices, that are widely distributed, are connected to edge nodes which provide storage and computation power to reduce the amount of data sent to the upper level in the fog [9, 4]. Fog computing [10] and fog storage [11] are extensions of the cloud computing proposed by Cisco in 2014, in which computing capabilities are placed close to the edge network. Fog computing is based on a decentralized computing architecture where storage and processing applications are distributed between the edge and the cloud [10]. The use of fog and cloud for data management commonly leads to a lack of controls over stored data [12]. This could result in critical incidents such as outages, violations of confidentiality, data loss, and unauthorized data access [13].

Existing end-to-end and in-house solutions have been proposed for adding to contents properties such as security [14, 15], reliability [16, 17], and integrity [18], which could mitigate and, in some cases, fully address the effects produced by an incident that could arise in the cloud. Nevertheless, applying those solutions in real scenarios represents a challenge for the organizations to face up the expensive costs of the processing and exchange of data produced in different environments (any of edge, fog, and cloud). In such a scenario, the number of operations sent to outsourcing services as well as the data volume is costly and time-consuming tasks increasing the aforementioned issues originated by the accumulation of data.

In this context, it is also required to face problems associated with the existence of duplicated data produced during the management of documents versions and to efficiently manage the accumulated backups performed by the end-users. Data preparation is a technique mainly used in big data scenarios to preprocess and transform data for improving its quality. We previously presented an efficient data preparation scheme for cloud storage based on containerized parallel patterns [19], as an approach for organizations to prevent

suffering side-effects from the lack of control over their outsourced data. This approach also reduced unnecessary processing time associated with replicated data managed by cloud operations.

However, implementing these schemes over different environments (any combination of edge, fog, or cloud) and making the data available at each of these environments for end-users is not a trivial issue to solve in an immediate, portable and cost-efficient manner. In an IoT solution, portability results in reducing downtime for troubleshooting and fixing installation errors, whereas efficiency and quick availability end up improving the response times in decision-making procedures. There is thus a need for frameworks enabling organizations to build portable, secure, and reliable solutions for processing big IoT data on different environments by using multiple services in a cost-efficiency manner.

This paper presents a processing model to build software pipelines from the edge to the cloud for big IoT data scenarios. This model is based on a continuous delivery scheme and portable infrastructure-agnostic black boxes and building blocks. Continuous delivery is a popular technique in software engineering for delivering software updates from the developers through the testing stages to the end-users in an interrupted manner [20]. The basic idea is to apply this very concept from software engineering to the big IoT data processing pipelines to produce instrumented processing and data delivery from the edge to the fog to the cloud.

The black boxes are management structures for building pipelines over any combination of edge, fog, or cloud, whereas the building blocks are used to encapsulate the applications, defined by organizations, into independent pieces of software that perform, in an implicit manner.

These structures perform the following management tasks in implicit manner: *i)* The automatic delivery of data through the pipelines with a focus on the reduction of I/O operations by using deduplication and compression techniques. *ii)* The security and reliability of data arriving/departing to/from the black boxes. Continuous controls were established over the production and consumption of data by adding to data properties such as confidentiality, access controls, integrity verification, and fault tolerance. *iii)* The efficient execution of applications by using both schemes based on in-memory storage and I/O interfaces as well as recursive parallelism patterns for improving the performance of building blocks to compensate for the costs of implicit management. In this model, the deployment and coupling of portable black boxes and building blocks in the form of software pipelines is performed in automatic and transparent manners. An overlay structure has also been proposed to establish supervision over planes such as Pub/Sub, control, and preservation. This overlay integrates the preparation and cost-efficiency schemes into software pipelines for producing a quick availability of IoT data at different stages of the lifecycle. This overlay also enables organizations to share IoT data and achieved information with end-users and/or partners at each stage of a pipeline.

This model has been implemented in a prototype that was developed to show the efficacy and feasibility of the model presented in this paper. As testing in real-world environments can be slow and cumbersome, a simulator of the system has also been developed for organizations to fine-tune the parameters of the prototype by measuring each software pipeline design. The number of stages, types of stages, and pattern features are examples of param-

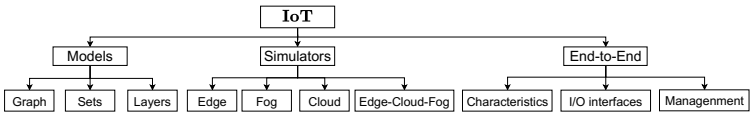


Figure 1: IoT solutions taxonomy.

eters than can be tuned in this simulator, whereas different benchmarks can be defined by using the probabilistic models described in this paper. This simulator is available for edge, fog, and end-users' devices by now (cloud latency and processing are aspects not considered in this simulator).

The experimental evaluation was conducted in the form of two case studies based on the management and processing of real climate data repositories from Mexico and Spain as well as small distributed IoT devices. The experimental evaluation revealed the feasibility of using a continuous delivery approach to create dataflows from the edge to the cloud. It also revealed the efficacy of the overlay to create an information-sharing pattern as well as the efficiency of data preparation schemes to mitigate risks in the cloud when using parallel patterns, which also improved the end-user service experience.

As a summary, the main contributions of this work are:

- *A continuous delivery scheme* based on building blocks for constructing software pipelines from the edge to the cloud.
- *Data preparation schemes based on parallel patterns and IoT data sharing based on overlay structure.* These schemes enable organizations to establish, in an efficient manner, control over the production and consumption of IoT data by adding to data properties such as cost-efficiency storage, security, and reliability.
- *A parallel pattern and software pipeline simulator based on virtual containerized building blocks.* A new simulator based on virtual containers enables organizations to estimate service and response times of preparation schemes for processing IoT datasets.

The rest of the paper is organized as follows: Section 2 describes the related work. Section 3 describes a continuous delivery model for big IoT data based on building blocks. Section 4 describes the design principles of an overlay structure for integrating data preparation schemes into software pipelines. Section 5 presents the design and implementation of a simulator based on both, the proposed model and the overlay structure. Section 6 presents the implementation details of a prototype. Section 7 shows the results of the experimentation using the provided simulator and prototype. Section 8 presents an analysis of strengths, weaknesses, opportunities, and threats. Finally, Section 9 gives conclusion remarks and future work.

2. Related work

The complex challenge of providing solutions for the processing of big IoT data has been addressed from different perspectives and scopes [6, 4]. Figure 1 shows a taxonomy of such

Table 1: IoT simulators summary.

Work	Environments			Task/element simulated									
							Processing						
							Patterns			Purpose			
	edge	fog	cloud	Sensors	Networking	Data gathering	MR	D&C	M/W	NFR Validation	Analytics	Task placement	Resource provisioning
MRPerf (2009) [33]	-	-	✓	-	-	-	✓	-	-	-	✓	-	-
Mumak (2009) [26]	-	-	✓	-	-	-	✓	-	-	-	✓	-	-
MRSim (2010) [27]	-	-	✓	-	-	-	✓	-	-	-	✓	-	-
SimMR (2011) [34]	-	-	✓	-	-	-	✓	-	-	-	✓	-	-
CloudSim (2011) [30]	-	-	✓	-	-	-	-	-	-	-	-	✓	✓
MR-CloudSim (2012) [35]	-	-	✓	-	-	-	✓	-	-	-	✓	-	-
Brambilla et al. (2014) [36]	✓	✓	-	-	-	✓	-	-	-	-	-	-	-
SimIoT (2014) [24]	✓	✓	-	-	-	✓	-	-	-	-	-	-	-
DPWSim (2014) [37]	✓	✓	-	-	-	✓	-	-	-	-	-	-	-
DISSECT-CF (2015) [38]	✓	✓	-	-	-	✓	-	-	-	-	-	-	-
Edge-Fog-Cloud (2016) [9]	✓	✓	✓	-	-	-	-	-	-	-	-	✓	-
Narrowband-IoT/OPNET (2017) [21]	✓	-	-	-	✓	-	-	-	-	-	-	-	-
iFogSim (2017) [25]	✓	✓	-	-	-	✓	-	-	-	-	-	-	-
Yaozhong Song et al. (2017) [29]	✓	✓	✓	-	-	-	-	-	-	-	-	✓	-
IoTSim (2017) [31]	-	-	✓	-	-	-	✓	-	-	-	✓	-	-
MultiRECloudSim (2017) [32]	-	-	✓	-	-	-	-	-	-	-	-	✓	✓
System Vue (2018) [23]	✓	-	-	✓	✓	-	-	-	-	-	-	-	-
Minh-Quang Tran et al. (2019) [28]	✓	✓	✓	-	-	-	-	-	-	-	-	✓	-
Mercury (2019) [39]	✓	✓	-	✓	✓	✓	-	-	-	-	-	-	-
Overlay simulator	✓	✓	✓	-	-	-	-	✓	✓	✓	-	-	-

solutions including: i) simulators for the edge, the fog, and the cloud; ii) models based on graphs, sets, or layers; and iii) end-to-end solutions for the processing and management of data.

2.1. Simulation tools

Simulation of edge, fog, or cloud environments can be classified into six categories depending on the task or elements being simulated: sensors, networking, data gathering, data processing, task placement, and resource provisioning.

Representative sensor simulators are OPNET [21] and ATEMU [22]. The first one models the protocols and technologies to analyze communication networks whereas the second one includes network integration. Vue [23] is a simulation of networks such as 5G-NR, LTE, LTE-Advance, HSPA+, DC-HSPDA, MIMO, and multistandard radio signals. By the side of data gathering, simulators SimIot [24], DPWSim, and iFogSim [25] have been used in healthcare scenarios, to evaluate data gathering process performance, and for measuring the impact of resource management techniques on latency, network congestion, and energy consumption in IoT models. For data processing, the simulator Mumak [26] allows studying different scheduling algorithms for MapReduce (MR) tasks while MRSimn [27] simulates the configuration of a Hadoop cluster, the time for completing a job in the cluster and hardware utilization. In the case of task placement, Mohan and Kangasharjy [9] presented an edge-fog-cloud simulator aiming at minimizing the processing and network costs. Minh-Quang Tran et al. [28] presented an extension of SIMIC simulation by adding an IoT layer to perform the allocation of virtual machines. Yaozhong Song et al. [29] presented another approach for the distribution and management of tasks through the edge satisfying quality-of-service requirements. Finally, CloudSim [30], IoTsim [31], and MultiRECloudSim [32] are representative simulator for modeling, simulation, and evaluation of resource provisioning algorithms.

Table 1 shows a comparison of previously reported simulators against the one proposed in this paper, analyzing two large subcategories. The first one is the environment covered (edge, fog, and cloud) and the tasks/element simulated. The second subcategory it is checked if the simulator is for sensors [23], networking [21], data gathering [24, 25, 37, 36, 38]; data

Table 2: Qualitative comparison of IoT models from the state of the art.

Work	Environment				Representation			Element modeled					
									Apps				
	Sensors	edge	fog	cloud	Graphs	Sets	Layers	Networking	FR	NFR	Sensing Systems	Streaming	Analytics
Yinghui and Guanyu (2010) [40]	✓	-	-	-	✓	✓	-	✓	-	-	✓	-	-
NoFLo (2015) [44]	-	✓	-	-	✓	-	-	-	∼	-	-	-	-
Calvin (2015) [45]	-	✓	-	✓	✓	-	✓	-	∼	•	-	-	-
Yuankun Xue et al. (2017) [41]	-	✓	✓		✓	-	-	✓	∼	•⋄	✓	-	-
RADF (2017) [42]	-	-	✓	✓	✓	-	-	✓	∼	≈	-	✓	-
CCM (2018) [43]	-	-	-	✓	-	-	✓	-	∼	•	✓	-	✓
Overlay model	-	✓	✓	✓	✓	-	-	-	∼	•≈⋄	-	-	-

FR: functional requirements, NFR: non-functional requirements, efficacy: ∼, security: ⋄, reliability: •, efficiency: ≈.

processing [26, 27, 35, 33, 34, 31], placement of tasks in the infrastructure available [9, 28, 29]; and resource provisioning in the cloud [30].

2.2. Models

The different abstract representations to model the management of components of IoT environments at the edge, fog and the cloud can be classified into six groups based on the element being modeled: IoT network, IoT apps, IoT sensing systems, data streaming, and data analytic.

Examples of networking models are the ones proposed by Yinghui and Guanyu [40], Yuankun Xue et al. [41], and Sabine and Andreas [42]. These models focused on the representation of communication protocols, the tokens to expose the network, the lossless channels, and the bandwidth of the IoT network. Application models [42, 43, 44, 45] allows studying the interaction among software components, management for mapping software components to hardware, and modeling the behavior of software processes. Representative models for sensing systems can be found in [41, 43, 40, 46]. RADF [42] is an adaptive data flow model for network-of-system specifications supporting event-driven executions. CCM [43] is a concentric computing model that considers the complexity, heterogeneity, data volume, and execution of applications (e.g., data ingestion, cleaning, conformation, transformation, and shaping).

Table 2 shows a comparison of existing models against the one proposed in this paper. Three large subcategories were analyzed in that table: the environment covered (IoT sensors, edge, fog, and cloud), the abstract representation of the model (graphs [44, 42, 41, 40], sets [40] and layers [45, 43]), and the elements abstracted in the model. In the third subcategory, it is checked the models of networking IoT devices and infrastructure [40, 41, 42], the functionality of apps and the accomplishment of non-functional requirements (security, reliability, and efficiency) [42, 43, 44, 45], as well as the IoT sensing systems [41, 43, 40, 46], the data streaming [42], and data analytic [43].

2.3. End-to-end solutions

Non-functional requirements such as confidentiality, integrity, privacy, and reliability are crucial in sensitive scenarios when managing IoT data in critical decision-making procedures. Further, those requirements should be met for the accomplishment of data management norms and laws imposed by government and organizations [47]. Existing end-to-end and workflows solutions can be classified according to the non-functional requirements that they

Table 3: Summary of end-to-end solutions and workflows engines.

Work	Scope	Efficiency					Reliability		Security		
		Processing			Storage						
		Threads	Patterns	In-memory	Compressing	Deduplication	Data	Processing	Access control	Confidentiality	Integrity
Wiseman et al. (2005) [53]	End-to-end	-	-	-	✓	-	-	-	-	-	-
CloudSeal (2011) [18]	End-to-end	-	-	-	-	-	-	-	✓	✓	✓
Jenkins (2011) [49]	CD/CI pipelines	✓	-	-	-	-	-	✓	✓	-	-
Makeflow (2012) [48]	Workflow engine	✓	-	-	-	-	-	✓	-	-	-
Zhang et al. (2015) [15]	End-to-end	-	-	-	-	-	-	-	✓	✓	-
Mao et al. (2015) [17]	End-to-end	-	-	-	-	-	✓	-	-	-	-
StorReduce (2018) [55]	Dedup system	-	-	-	-	✓	-	-	-	-	✓
AES4SeC (2018) [14]	End-to-end	-	-	-	-	-	-	-	✓	✓	✓
Sacbe (2018) [16]	End-to-end	-	-	✓	-	-	✓	✓	-	-	-
Dedupv1 (2010) [56]	Dedup system	-	-	-	-	✓	-	-	-	-	✓
Parsl (2018) [51]	Workflow engine	✓	✓	-	-	-	-	✓	-	-	-
DagOn* (2018) [50]	Workflow engine for IoT	✓	-	-	-	-	-	✓	✓	-	-
Overlay	Workflows and software pipelines	-	✓	✓	✓	✓	✓	✓	✓	✓	✓

managed: efficiency, data integrity, data reliability based on data coding and processing reliability (fault-tolerance), and security based on access control and confidentiality.

Solutions such as Makeflow [48], Jenkins [49], DagOn* [50], and Parsl [51] provide implicit parallelism based on threads for the execution of tasks in a workflow/pipeline. In-memory schemes for pipelines and even in a non-distributed workflow [16, 52] have been proposed for the exchange of data between the stages in a pipeline/workflow. Storage cost-efficiency schemes based on techniques such as deduplication and data compression are available to reduce the volume of data to transport and store in the cloud [53]. Deduplication techniques to find replicated data [54] such as StorReduce [55] and Dedupv1 [56] are few examples of this type of solution. The integrity requirement has been achieved by computing checksums (e.g. SHA3) over data before transmission [57] and to identify data alterations during data movement [58]. Solutions for studying processing reliability have focused on relaunching a complete pipeline and starting from the beginning of data processing, for example, Parsl [51] and DagOn* [50], or on recovering the stage with failure and restarting the pipeline from the last safe state before of the failure, as in Sacbe [16]. Confidentiality, defined as the capacity of a solution to preserve data privacy between processing stages and nodes [59]; and access control, defined as the capacity of the solution to give access to data only to those authorized users have been proposed as services [53, 15, 14]. Data fault tolerance has been achieved by using information dispersal and data codification techniques [16, 17].

Table 3 shows a comparison of end-to-end existing solutions against the prototype presented in this paper. The first subcategories show the scope of the solution (end-to-end, CD/CI pipelines, workflow engine, or dedup system). In the second subcategory, it is shown the non-functional requirements of efficiency for data processing (threads, patterns, and in-memory) and data storage (compressing and deduplication), reliability for fault-tolerance of data storage locations and processing applications, and security by applying users access control, data confidentiality, and data integrity techniques.

2.4. Discussion

Figure 2 presents a visual summary of the related work previously described based on quadrants that represent edge, fog, cloud, and end-users. In each quadrant, the horizontal axis describes the studied properties (efficacy, efficiency, security, and reliability), whereas the vertical axis represents the scope of the solutions (models, simulators, emulators, and prototypes). References of related works previously discussed are placed in the diagram

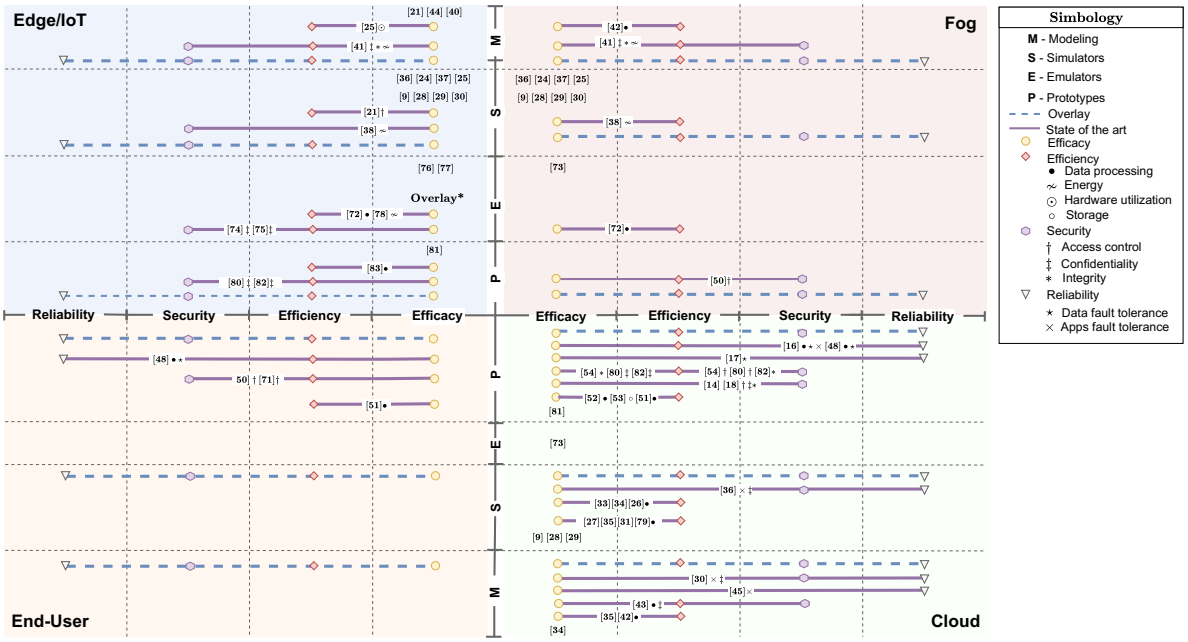


Figure 2: State of the art summary based on edge, fog, end-user and cloud quadrants.

showing the requirements they meet, their technological nature, and their area of application (any of edge, fog, cloud, or end-user). It can be observed from Figure 2 that the existing works in the literature have been focused mainly on functional requirements. However, the flexibility of the overlay model presented in this paper allows considering both functional and non-functional requirements for different environments (edge, fog, cloud, and end-users' devices) as well as for different scopes (model, simulator and prototype).

The model proposed in this paper not only provides a representation tool but also provides data preparation schemes for security and data transformation, standardized connections between components, and efficiency patterns through parallel deployment. The proposed overlay uses integrated parallel patterns (*manager/worker* and *divide&conquer*) and in-memory data flows to improve the efficiency of processing tasks. Implicit parallelism is provided by cloning apps encapsulated into virtual containers, which not only improves the continuous delivery of data but also the portability of black boxes and building blocks. Fault-tolerant data processing is provided by relaunching those virtual containers in case of failures. Data integrity is achieved using checksums embedded in the input/output connectors, and by using the information dispersal algorithm (IDA) [60] to provide data redundancy. Access control is provided by using certificates.

From the model, the simulator proposed in this paper allows organizations to quickly estimate the execution time of a data processing workflow in IoT-edge-fog-cloud, without putting a lot of specific details on any component or restricting the processing to MapReduce operations as most simulators for data processing currently do. Moreover, the proposed model in this paper is focused on the abstraction of non-functional requirements of security, reliability, and efficiency in the management of data, not on the abstraction of the network or

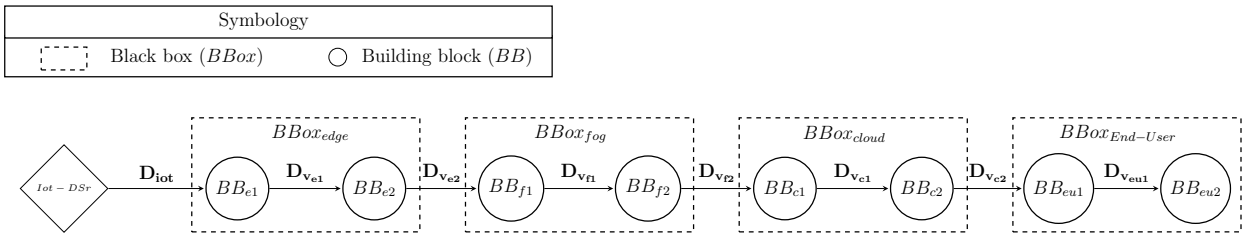


Figure 3: A DAG describing a structure for processing IoT data based on black boxes and building blocks.

the sensing systems. Non-functional requirements are required by the end-users for avoiding data delivery interruption, confidentiality violations, and alterations of IoT data before, during, and after them are processed from the edge to the cloud, which is crucial in critical decision-making procedures and to observe norms and regulations when processing sensitive IoT data.

3. A continuous delivery model for big IoT data

This section presents a continuity processing model for big IoT data based on abstractions called *black boxes* ($BBoxes$), which can be used by developers to construct software structures for processing IoT data from the edge to the cloud by following the principle of continuous delivery. $BBoxes$ thus are associated to an IoT environment: $BBox_{Egde}$, $BBox_{fog}$, $BBox_{cloud}$ and $BBox_{End-User}$ (see Figure 3). Thus, a processing IoT solution ($IoTS$) can include any combinations of available $BBoxes$ ($IoTS = \{BBox_{edge}, BBox_{fog}, \ldots, BBox_{cloud}\}$).

The $BBoxes$ consider portable infrastructure-agnostic construction units called *building blocks* (BBs) to deploy applications on a given environment. This means a set of BBs available in a pool of services can be associated to a $BBox$ (e.g., $\{BB_1, BB_2, \ldots, BB_2\} \in BBox_{fog}$).

These processing solutions for big IoT data are created by following the next steps: *i)* to encapsulate a set of applications with their libraries, dependencies, and control structures into BBs; *ii)* to associate BBs structures with $BBoxes$ (see $BBoxes$ in Figure 3), which also includes control structures to be deployed on different environments (see a DAG of $BBoxes$ mapped with an environment in Figure 3); *iii)* to apply the continuous delivery concept from software engineering for coupling $BBoxes$ to create big IoT data processing solutions [1]; and *iv)* to produce uninterrupted processing and data delivery through the $BBoxes$ and BBs. To illustrate this continuity effect, see in Figure 3 how D_{IoT}, extracted from an IoT data source, is transformed from the edge to the end-users producing a different version of D_{IoT}.

The model proposed in this paper is based on trees of *directed acyclic graphs* (DAG) to manage, create, and deploy the IoT processing solutions. Nodes of the DAG represent $BBoxes$, which also include another DAG where nodes represent BBs. Edges represent the

[1]Continuous delivery is a technique used in software engineering for uninterruptedly delivering software updates from the developers through the testing stages to the end-users. [61, 20].

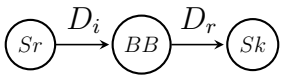


Figure 4: A DAG describing the ETL process of a BB.

I/O interfaces used for interconnecting $BBoxes$ at a top-level and BBs at a bottom level. These I/O interfaces are managed by control structures embedded within the $BBoxes$ and BBs, and they can be configured to use any of the memory, network, or the file system. To interconnect the nodes of these DAGs, structures are defined as an Extract, Transform, and Load (ETL) process. As a result, a $BBox$ can be defined by the following expression:

$$BBox = (DSr, (BB_{i=1}^{n}, DAG), DSk), \tag{1}$$

where DSr is a data source connected to the input of the $BBox$ (extraction stage). $BB_{i=1}^{n}$ is a set of BBs managed by the $BBox$ (transformation stage), DAG represents the topology of the BBs connections, and DSk is a data sink connected to the output of the $BBox$ (load stage). A BB is modeled by the very processing model used in $BBoxes$ as follows:

$$BB = (in, task, out), \tag{2}$$

As it can be seen, a BB includes I/O interfaces and an application for processing data (any of program, application, or binary). Where in is the input interface, $task$ is the processing unit, and out is the output interface. When a $BBox$ and a BB are modeled by an ETL, it is created a pipeline as the DAG showed in Figure 4 [62], where a data D_i is extracted from a data source (Sr) by a transforming entity (BB), which delivers the transformed data (D_r) to a data sink (Sk).

Thus, the goal of the model is to produce a dataflow through the edges (I/O interfaces) for nodes ($BBoxes$ and BBs) to uninterruptedly process the incoming data. This is achieved by using the continuous delivery concept to create a continuous deployment of $BBoxes$ and BBs on edge-fog-cloud environments. This concept also is used to continuously deliver data and to establish controls over the execution of the nodes and edges defined in the tree of DAGs.

Abstractions such as $BBoxes$ and BBs, as well as their combinations to build IoT data processing structures are managed as services, which enable end-users to reuse the $BBoxes$ and BBs for building multiple solutions, as well as enable apps to consume data from the output of each $BBox$. A service mesh manages these services, which will be described in the implementation of Section 6.

3.1. Building IoT data processing solutions based on patterns: pipelines and parallel patterns

This section describes different patterns to create IoT processing solutions such as pipelines, parallel patterns, and workflows which can be built by using the ETL processing model previously described.

3.1.1. Pipelines

In this model, a pipeline is modeled as a set of $BBoxes$ and BBs (filters) connected in an adjacent manner through I/O paths (pipes), denoted by: $PBBoxes = (\{BBoxes_1, BBoxes_2, \ldots, BBoxes_n\})$ and $PBBs = (\{BB_1, BB_2, \ldots, BB_n\})$. The input data received and processed by a $BB/BBox$ is denoted as $BB[in]$, the data resulting after $BB/BBox$ processing is denoted as $BB[Out]$, and the processing task(s) into the BB is denoted as $BB[task]$. This is an ETL *pattern*, named as $Patt_{etl}$, defined as:

$$Patt_{etl} = \{Sr_i \rightarrow BB_i[in], BB_i[task], BB_i[Out] \rightarrow Sk_i\}. \tag{3}$$

where Sr represents a data source as input, and Sk represents a data sink to deliver results produced by this BB. The notation of an ETL pattern generalizes the definition of chained BBs. For a given BB_j, in a pattern $Patt_{etl,1}$, its source Sr_j can be the sink Sk_i of BB_i. The output of BB_j can be associated to the source of BB_k given $Patt_{etl,2}$. This is denoted as:

$$\{Patt_{etl,1}, BB_j\} = \{Sk_i, BB_j, Sk_j\}, \tag{4}$$

$$\{BB_j, Patt_{etl,2}\} = \{Sr_j, BB_j, Sr_k\}. \tag{5}$$

Eq. 4 defines the chaining of BB_i with BB_j and Eq. 5 defines the chaining of BB_j with BB_k. A topology is created by an implicit software embedded within a $BBox$ by using the previous notation, which is used for the deployment, coupling, and execution of the BBs considered in a pattern. Figure 3 shows the DAG of an implicit software pipeline, which creates, within a $BBox$, a sequence of BBs (nodes) adjacently connected through I/O interfaces (edges). The notation of the $BBox$ for this *pipeline* is:

$$Pipeline = (DSr, (BB_{i=1}^{n}, Patt_{etl,x}), DSk), \tag{6}$$

where DSr is connected to $BB_1[in]$, DSk is connected to $BB_n[out]$, n is the number of BBs in the pipeline, $n \geq 1$, and $Patt_{etl,x}$ is defined as:

$$Patt_{etl,x} = \{BB_i[Out] \rightarrow BB_{i+1}[In]\}_{i=1}^{n-1}. \tag{7}$$

This pattern creates a dataflow from a data source (DSr) to a data sink (DSk), where each BB_i will transform the input data. For instance, in Figure 3, $\mathbf{D_{iot}}$ input in $BBox_{edge}$ is transformed into $\mathbf{D_{ve2}}$ and forwarded to $BBox_{fog}$ through a *pipe* represented by an edge.

3.1.2. Implicit parallel patterns at black box level

In this model, IoT solution designers can build implicit parallel patterns to transparently improve the efficiency of $BBoxes$ by including reserved BBs as control structures such as *divider* (D) and *worker* (w), and in some cases *consolidator* (C).

Figure 5 depicts an example of a traditional *manager/worker* pattern [63]. In this pattern, D extracts data from a source, and distributes these data as tasks ($P = \{c_1, c_2, \ldots, c_n\}$, where n is the size of the contents set c_i) to w workers by generating w subsets of contents ($p_w = \{c_1, c_2, \ldots, c_m\}$). Each worker processes data subsets to produce a content (c_i'), which

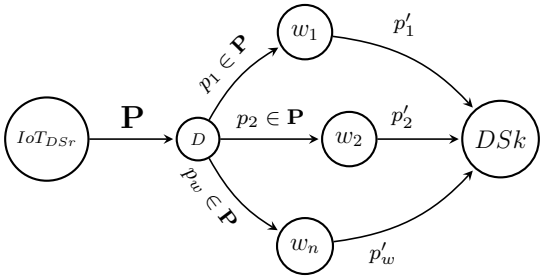


Figure 5: Divide&Containerize pattern represented as DAG.

are stored in a given data sink (DSk) that can be placed at another $BBox$ (e.g., any of $BBox_{fog}$, $BBox_{cloud}$ in a cloud storage location, or $BBox_{End-User}$). Workers can deliver the results to any of a $BBox$, sink, another divide (in a recursive fashion) or a *consolidator* entity.

A *consolidator* BB is required when each incoming content is split into segments and these cannot be individually used by a $BBox$. These segments are processed by the workers, and the processed segments are sent to the consolidator, which organizes and merges the received segments into a consolidated result. This method produces either a traditional *divide&conquer* or *fork/join* pattern [63] depending on the control structures used by both workers and the consolidator. The map of a *divide&conquer* pattern is represented by the following DAG notation:

$$BB_{Patt} = \{DSr, (D, (w_{i=1}^{n}, Patt), CE), DSk\}, \tag{8}$$

where $n > 1$, $Patt = \{D\&C \xrightarrow{s_i} w_i \xrightarrow{r_i} Cq\}_{i=1}^{n}$, and CE is a consolidator entity (any of Cq for *divide&conquer* or Jn for *fork/join* patterns. As it can be seen, different types of combinations of patterns ($Patt$) can be built in this model for creating a BB, in the same way, that a software pipeline of BBs can be built to create a $BBox$.

Notice that parallel patterns are also managed as a service (formally a microservice), thus additional control structures are added to the patterns for enforcing the implicit management of data processing. The control structures considered for these tasks are a load balancing BB for allocation workload as well as messages, and data I/O manager for enforcing the ETL in the patterns, which includes in-memory data resource library to reduce the operations sent to slow interfaces (e.g., file system and network).

4. An overlay structure for sharing IoT data in workflows and software pipelines

In this section, we describe an overlay structure developed to organize the patterns of BBs of each $BBox$ in the form of pipelines, that not only process IoT data but also enables the decision-makers and end-users to automatically get access at each stage ($BBox$) of the IoT lifecycle. Figure 6 depicts the main components of the overlay structure, that includes planes such as *Pub/Sub*, *Control*, and *Preservation*, which establish controls over metadata, structured IoT data, and files respectively. The idea is to decouple the data processing from their preservation and consumption.

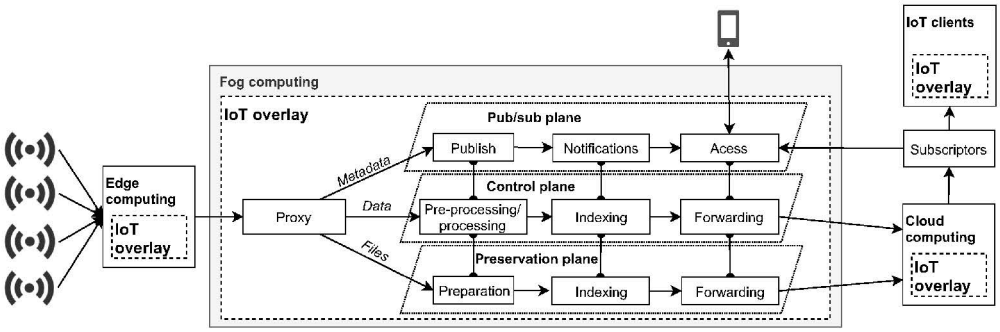


Figure 6: Architecture overlay.

The overlay considers a reserved BB called *proxy* (or a set of proxies depending on the designers' needs), which collects data from the input interface of a $BBox$ where the overlay is deployed on and launches the managers of the three planes. Each action in the control plane triggers an action at the preservation and pub/sub planes. The overlay planes establish control points over the consumption/production of information, making it available at the pub/sub plane.

The *pub/sub* plane as its name suggest is based on a publication/subscription model and manages the publishing of the data produced by each BB, the control of the metadata, the access to the data and results, and to notified users subscribed to a catalog when a new result is produced. Three roles are established in this plane: *i)* the *publishers* are $BBoxes$ acquiring or producing data that will make available for other $BBoxes$; *ii)* the *consumers* are end-users or $BBoxes$ consuming data by subscribing published catalogs; and *iii)* the *managers* are users that establish rules in the pub/sub patterns, which are created by the subscriptions performed to publications of catalogs.

The data acquired by the proxy, the data produced at the control plane, and the files prepared at the preservation plane can be published and consumed on-demand through the Pub/Sub plane. This means the information can be accessed implicitly and transparently by any of end-users' applications, BBs, and $BBoxes$ by only subscribing a given stage at the processing IoT data structure.

In the *control* plane, the data are converted into information in the preprocessing/ processing phase by executing pipelines of BBs. For instance, preprocessing pipelines with BB to eliminate/detect outliers, and to enrich the data, or processing pipelines with BBs to transform, interpolate, and analyze the data to produce information that helps in the decision-making process. These BBs are selected by the *manager*, depending on how he/she wants to manipulate their data. Preprocessing/processing pipelines retrieve the data from the proxy and deliver the processed data to the preservation plane. The information produced by this layer is sent to the pub/sub plane, which generates automatic notifications to the consumers who are subscribed to the sensor.

In the *preservation* plane, stages such as preparation, indexing, and forwarding are considered. The preparation stage invokes pipelines to add properties to the data such as security and reliability for establishing controls over the data lifecycle. These properties

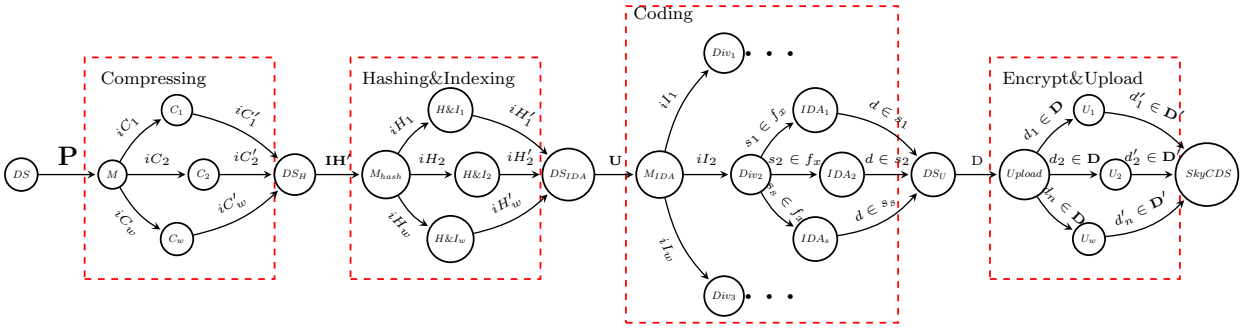


Figure 7: DAG representing the data preparation process.

are applied by using techniques such as data compression (to improve the cost-efficiency of storage), data integrity (to preserve data integrity), encryption (to establish control access), and information dispersal (to establish fault tolerance). These techniques are managed as BBs, modeled as parallel patterns, and chained as a pipeline. In the indexing stage, the status of the prepared files is informed to the control plane. In the forwarding stage, the control and the data are passed to another $BBox$.

Notice that the BBs included in a given plane can be implemented as a parallel pattern.

4.1. Applying data preparation and retrieval schemes to the overlay

As established in the study of the related work, non-functional requirements are crucial for IoT processing scenarios. In the preservation plane of the overlay, the goal is to meet security and reliability requirements for managing any of ensuring sensitive data, withstanding data unavailability, avoiding confidentiality violations, detecting alterations in the data during their lifecycle, or presenting solutions for non-repudiation. Figure 7 shows the DAG of a preparation scheme, which includes four BBs. The first BB improves the cost-efficiency of capacity management by applying a compression application (LZ4 [64]) implemented as a *manager/worker* pattern, where a set of paths P is retrieved from the data source DS. P is divided into w sub-lists in a load-balanced manner [65], to w number of BBs.

The second BB performs an integrity and duplicity verification (*hashing&indexing*) for detecting alterations and replicated data. This BB obtains the hash of the files using MD5 and SHA3-256 algorithms to achieve two goals: *i)* to identify files previously prepared, and *ii)* to verify the end-to-end integrity of the files, identifying data alterations during their transportation. The output of this BB is a set of lists (L) containing the hash of the processed files. These lists are sent to the metadata management service, which indexes the files and returns a list with files that were not previously indexed. At this point, a reduction in the number of files to be processed depends on the deduplication BB efficacy and on the saving achieved by compressing BB.

The third BB (*coding*) encodes data using the information dispersal algorithm (IDA) [60], which adds redundancy to data. This process splits the data d into n redundant portions, where m is sufficient to recover d whenever $n > m$. This produces fault-tolerance and reduces storage utilization (in comparison with replication techniques). This BB implements

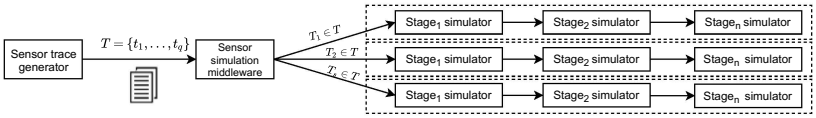


Figure 8: Conceptual representation of the simulator developed.

a combination of a *manager/worker* pattern with a *divide&conquer* pattern. This means that each worker invokes a *divide&conquer* were the incoming data are split into *div* number of BBs, which execute IDA (a double segmentation is produced in this BB).

The data arriving at the fourth stage (*Encrypt&Upload*), are sent to an encrypt and forwarding demon of a content delivery network called SkyCDS [66]. This demon executes ciphering techniques based on AES [14] and CP-ABE [67] to encrypt the incoming data and to create a ciphered object including access controls. The deployment of BBs in the form of parallel patterns reduces or even eliminates the overhead of applying integrity, security and reliability to data in these schemes.

5. Parallel patterns and software pipeline simulator based on containerized BBs

This section describes a simulator of BB patterns and pipelines used to estimate execution times and costs of the preservation plane operations in the overlay structure. Figure 8 depicts the conceptual representation of the simulator based on a modular approach. A *trace generator* of IoT sensors produces traces including the size of data and interarrival time of records produced by sensors. The data size can be configured by using a static parameter (in the case of the IoT devices) and from a normal distribution when selecting a sensor of the ground station. It is assumed that the traffic profile of each sensor can be managed by the wireless technology (e.g., IEEE 802.15.4). Sensors are arranged in a star network topology, where the sink is connected to a BB. Each sensor includes a timestamp and a sequence number to every transmitted packet for packet loss control purposes.

The generator is connected with a *data workload dispatcher*, which receives the traces and acts as middleware between the generator and the BB pipelines. The traces are distributed in a load-balanced manner between s number of pipelines of simulated BB. These pipelines are encapsulated into virtual containers (VCs), and each BB simulates a stage in the data preparation/recuperation schemes (see stages in Figure 8). The simulator includes a *service time estimation module* per pipeline, which produces the interpolation of the service time from the input data size. This module delivers the calculated service time and the interarrival time for each request in a trace to a queue simulator module based on an exponential distribution [68]. This module returns the service time, which now includes the queue contention and accumulates this time to the service time of the previous BB if any.

5.1. Interpolating the service time of BBs

The service time spent by a stage to process IoT data is computed through interpolation of values in tables reflecting the behavior of the system, as follows:

$$f(x) = ((x - x_1)/(x_0 - x_1)) * (y_0 - y_1) + y_1, \tag{9}$$

where x is the input data size to process, x_0 and x_1 represent the range between this size is founded, in such a way that $x_0 \leq x \leq x_1$, as well as y_0 and y_1 represent the service time of the size files to x_0 and x_1, respectively. Notice that y_0 and y_1 depend on the stage of the data preparation/retrieval schemes, as well as on the type of system where the test is executed in. For real hardware platforms, the tables for interpolation have been created through performance evaluation of each stage in the system. However, the tables have to be redefined in platforms with new features or even variants of actual systems (e.g., changing the network or the amount of memory) by testing the preparation/retrieval schemes.

5.2. Simulating awaiting queue in a BB

This section describes the design of the queue simulator algorithm (see Algorithm 1) developed to estimate the average time of simulation, average server utilization, number operations in the queue, and average delay in the queue of a BB. An exponential variable generation function (see Algorithm 2) produces a $\mathcal{U}(0,1)$ random number and an exponential random number. At the end of the simulation a report is generated for each BB including the response time spent by a pipeline to process each trace.

Algorithm 1 QueueSimulator(m_i, m_s, n_d)

Require: mean inter-arrival (m_i), mean service (m_s), number of delays required (n_d), seed one ($s_1 = 99275.0$), seed two ($s_2 = 48612.0$)

$yA = s_1$;
$y2 = s_2$;
while $num\_progs\_delayed \leq n_d$ **do**
  {Determine the next event and update time-average statistical accumulators.}
  switch (next_event)
  case 1**:**
    time_next_event[1] = $time + expon(m_i, yA)$; {Schedule next arrival.}
    if $server\_status == BUSY$ **then**
      $num\_in\_q += 1$;
    else
      $num\_progs\_delayed += 1$; {Server is idle, so arriving customer has a delay of zero.}
      $server\_status = BUSY$;
      $time\_next\_event[2] = time + expon(m_s, y2)$; {Schedule a departure (service completion).}
    end if
  case 2**:**
    if $num\_in\_q == 0$ **then**
      $server\_status = IDLE$; { Check to see whether the queue is empty.}
      $time\_next\_event[2] = 1.0e + 30$;
    else
      $delay = time - time_arrival[1]$; {The queue is nonempty, so decrement the number of customers in queue. }
      $total\_of\_delays += delay$;
      $num\_progs\_delayed += 1$;
      $time\_next\_event[2] = time + expon(m_s, y2)$; {Move each customer in queue (if any) up one place.}
    end if
  end switch
end while

5.3. Simulating a software pipeline

The preservation plane of the overlay is the one that produces the most significant costs in the response time of the pipelines considered in this paper. A data preparation/retrieval scheme includes a set of BBs (PBBs = $\{pBB_1, \ldots, pBB_m\}$) which are chained to create

Algorithm 2 expon($mean, ygen$)

Require: mean ($mean$), seed ($ygen$)
1: {Generate a $U(0,1)$ random variate using $ygen$ variable.}
2: $u = \mathcal{U}((0,1), ygen)$
3: **return** $-mean * log(u)$;

patterns: mainly pipelines, *manager/worker* and *divide&conquer* patterns. Three different times are included in the service time produced by a BB, which we recall is based on ETL model including the input data read time (IT_{pBBi}), data processing time (ST_{pBBi}), and the output data writing time (OT_{pBBi}). The service time (STp) of a pipeline thus is calculated by the sum of the service times of the BBs:

$$STp = \sum_{i=1}^{|\mathrm{PBB}|} ST_{pBBi}. \tag{10}$$

The total input (IT) and output (OT) times of a pipeline are calculated by the sum of the input (IT_{pBBi}) and output (OT_{pBBi}) times of each pBB_i. Therefore, the time observed by the end-user when a pipeline is executed is the response time (RT), which is calculated by the sum of ST, IT and OT, as follows: $RT = IT + ST + OT$.

5.3.1. Simulating parallel patterns

This section describes the simulation scheme created by using the parallel patterns (*manager/worker* and *divide&conquer*).

Manager/Worker simulation. This processing parallel pattern includes two main roles: the manager (M) and the workers ($\mathrm{W} = \{w1, \ldots, w_n\}$), where n is the number of BBs ($n > 1$). Given an input list of contents ($\mathrm{L} = \{co_1, \ldots, co_o\}$) with o elements and $o > 0$, again, three main times are considered in the service time of a manager: *i)* the time spent by the manager to read this content list ($RSource$); *ii)* the time spent to generate, in a load-balanced manner by using the *two choices algorithm* [65], a sub-list ($\mathrm{L_{wj}} = \{c_1, \ldots, c_p\}$) for each worker in the pattern (LB), where p is the size of the sub-list ($\mathrm{L_{wj}}$); and *iii)* the time spent by the manager to deliver n sub-lists to the n workers (LDW). Therefore, the service time of a manager is the sum of these times: $MT = RSource + LB + LDW$.

The service time of a worker (ST_{wj}) is calculated by the sum of the input ($IT_{wj,c}$), processing ($ST_{wj,c}$), and output ($OT_{wj,c}$) times for the content in the list ($\mathrm{L_{wj}}$), as follows:

$$ST_{wj} = \sum_{c \in \mathrm{L_{wj}}} [IT_{wj,c} + ST_{wj,c} + OT_{wj,c}]. \tag{11}$$

Therefore, the total service time of the workers is produced by the slower worker, as follows:

$$STW = \max_{j=0}^{|\mathrm{W}|}(ST_{wj}). \tag{12}$$

Thus, the service time of the M/W pattern is the sum of the times MT and STW: $ST_{MW} = MT + STW$.

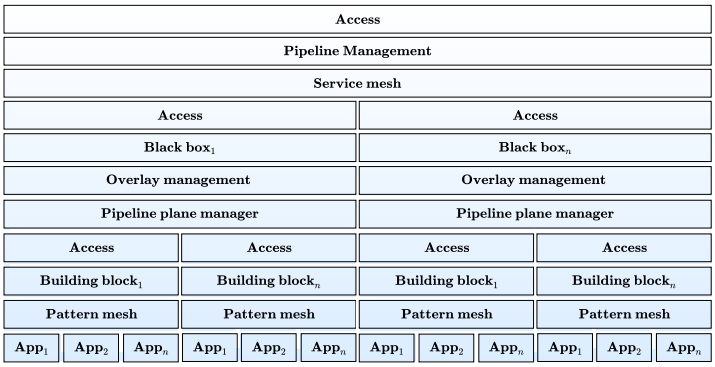


Figure 9: Microservice architecture of the prototype based on the overlay.

Divide&Conquer simulation. Given a set of contents (S = $\{s1, \ldots, s_{ns}\}$), this parallel pattern includes three main service times: *i)* the time ($DivTime$) spent to split a content c into ns segments; *ii)* the time spent by each worker to process a segment (ST_{DCk}), where k is the index of the worker in the pattern; and *iii)* the time spent by the pattern to integrate the processed segments ($ConqTime$) into a new content (c'). The $DivTime$ time includes the time spent by the pattern to read the content c, whereas the $ConqTime$ includes the time spent by the pattern to write the new content c' in either memory or the file-system. Thus the total service time of the pattern is:

$$ST_{DC} = DivTime + \sum_{k=0}^{|S|} ST_{DCk} + ConqTime, \tag{13}$$

5.3.2. Simulator validation

The simulator was validated by contrasting the service produced by each BB of the simulator with those metrics produced by the prototype (see Section 6). The service times of the BBs such as compression (LZ4), integrity (MD5), and reliability (IDA) were compared to the implementation of these software pieces deployed on real virtual containers. The BBs included in this simulator were fine-tuned by using the results produced by a prototype. Notice that the transportation of data to the cloud is not considered in this paper as the prototype is using SkyCDS for this very task and the simulation model of this content delivery has already been defined (see SkyCDS simulation model [66]).

6. Implementation of a prototype based on the overlay model

A prototype was developed and implemented by following the design principles of the continuous delivery model based on pipelines of $BBoxes$, the overlay of BBs, and parallel patterns. The prototype follows a microservice architectural model[2]. It is expected that the

[2]A microservice is a modular abstraction used to break down large monolithic applications into smaller and more specialized applications that are independent, language-neutral, and are bounded context [69].

microservices will be interconnected through a common interface (an HTTP API) [69], which results useful as heterogeneous applications are used to create solutions in IoT scenarios. To materialize the model into implementation, the first step is to encapsulate the applications into virtual containers to create portable pieces of software. The second one is to manage the resultant virtual containers by using the coupling model based on DAGs as well as control structures based on the overlay to create IoT data processing pipelines. In this context, the *BBoxes* and the coupling schemes are managed by a microservice architecture enabling the inter-operation of *BBoxes* in the form of pipelines.

The management of the components of the overlay in form of virtual containers ensures the portability and deployment of a IoT solution on different environments having installed a virtual container platform (e.g., Docker and Linux containers). This technical aspect and the implicit management of the overlay converts an IoT processing solution into an infrastructure-agnostic software. When implementing the management of IoT data by using this overlay model are: i) the preservation and the processing of IoT data are independent and transparent procedures; ii) the pipelines for preparation/retrieval at the preservation plane add quality properties to IoT data. iii) The implicit parallelism in the preprocessing and processing pipelines at the control plane results in independent processing structures; and iv) The processing of structured data (at data control plane) and raw IoT data (at preservation plane) are managed in secure, reliable, and efficient manners.

Figure 9 shows the microservice architecture used for the deployment and management of software pipelines as services. The architecture includes a service mesh to manage the *BBoxes* of a pipeline as microservices. This design decision was focused on making feasible the management of *BBoxes* by following the model presented in this paper. In a recursive manner, the *BBoxes* include an overlay builder (*overlay pipeline manager*). This component includes a pipeline builder per each plane enabled in each solution, which can choose *BBs* from a pool of microservices. The *BBs*, in a recursive manner, can execute either one or a set of apps from the apps pool by using the pattern mesh. As it can be seen, the service mesh only requires the notation of the model to make compositions of different types of pipeline solutions for different environments (*BBoxes*), including different control, preparation, and sharing of IoT data (*BBs*), to execute different types of big IoT data applications.

6.1. Black boxes implementation

The microservice architecture components were developed in C programming language. The microservices were managed by a REST API interface implemented in Python language. This API is added into the virtual containers of each *BB*, which enables other *BBs* and *BBoxes* to invoke the functions of that *BB*. The service mesh of the *BBoxes* was developed in C. Each *BBox* is configured by using a text plain file, where each line is written in the form `key:value`. The *value* can be the path of the source data of the *BB*, the number of workers in the pattern, and the access tokens to the content delivery network (CDN). Each *BBox* produces an YML file, which is deployed with Docker Compose.

The applications encapsulated into *BBs* can be managed in the form of source code or binaries. The control structures manage the execution of the application(s) that were included in the pattern mesh. The recollection of the output data produced by the execution

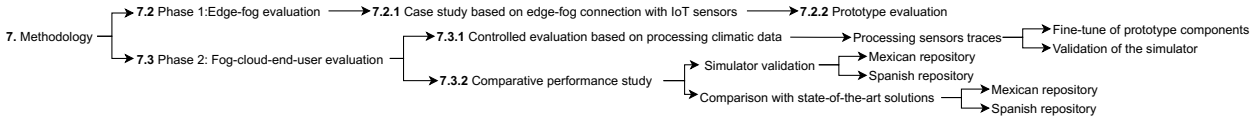


Figure 10: Experimental evaluation methodology.

of the application is managed by pattern mesh using the IPC memory, local file system calls, and put/get CURL functions.

6.2. Deploying of pipelines of black boxes and building blocks

A launcher builds and deploys the BBs of the overlay on the corresponding infrastructure. In deployment time, an *Interpreter Service* included in the launcher receives, as input parameter, a configuration file that is transformed into a DAG. The launcher follows the DAG to create the virtual containers images by adding the control entities (e.g., a manager, a segmenter, or a load-balancer), and the applications for each BB. The *service mesh* uses this information to perform the deployment of each BB considered in each $BBox$. In execution time, the first $BBox$ in the pipeline performs the verification of BBs coupling by using the I/O interfaces defined in the DAG. When the responses from all the $BBoxes$ are received, the pipeline is ready to process the data source and to receive requests. A record is created for each event occurred during the execution of an application encapsulated into a BB (e.g. successful/failed executions and the execution times).

7. Experimental evaluation methodology and results

This section presents a detailed description of the applied methodology to conduct the experimental evaluation of the overlay model proposed in this paper. This methodology considers two phases (see Figure 10) to study the processing of climatic data produced by sensors of IoT devices of two dimensions (small and large). Controlled evaluation in laboratory and case studies were conducted in this evaluation to fine-tune the prototype, to validate the simulator, and to evaluate the flexibility and feasibility of the implementation of this model in real scenarios in a direct compassion with state-of-the-art solutions.

The first phase of this methodology (based mainly on small sensors) was designed to evaluate the prototype components when processing IoT data at the edge-fog environments (see phase 1 branch in Figure 10).

The second phase considers: *i)* a controlled evaluation to fine-tune the prototype components and to validate the simulator of parallel patterns for preparation schemes of non-functional requirements (security and reliability). And, *ii)* to show the feasibility and flexibility of implementing the overlay model in real scenarios including fog-cloud-EndUser environments by conducting a performance direct comparative study between this model and traditional state-of-the-art solutions for data transportation in the cloud (SCP, rsync), continuous delivery (Jenkins[49]) as well as workflow engines (Makeflow[48] and DagOn*[50]), which were also implemented in a prototype (see phase 2 branch 2 in Figure 10) when processing repositories of data produced by sensors of ground stations of two countries. In this phase the simulator also was evaluated with the studied climatic repositories.

Table 4: Characteristics and response time of the hub to attend eight sensors transmitting packets at different rates.

Configuration	Sensor	Number of sensors/Packets-per-second (pps)	Response time (ms)
1		8/14	1.155
2	CC1350 SensorTag	8/128	1.676
3		4/14 and 4/128	1.675

The methodology details, descriptions of studied solutions and configuration as well as evaluation results are described in the following subsections.

7.1. Metrics

In the case studies conducted in the two phases of this experimental evaluation were considered the response time, service time, and the throughput as metrics to measure the performance of each configuration and solution implemented in the prototypes and/or the simulator.

The *service time* represents the ETL time of each BB. In the case of the model presented in this paper, the service time of the $BBoxes$ also includes the preparation/retrieval of the data. The *response time* represents the time observed by the end-users when executing the configurations and solutions evaluated in this paper. The *Throughput* of a system can be inferred by analyzing response/service times with the amount of IoT data processed by that system, which can be directly compared with the metrics produced by similar systems to determine, for instance, the costs of the management overhead of each system. In this experimental evaluation, these metrics are measured and evaluated to determine the overhead of the preparation schemes of security, integrity, accesses controls and reliability of the model proposed in this paper.

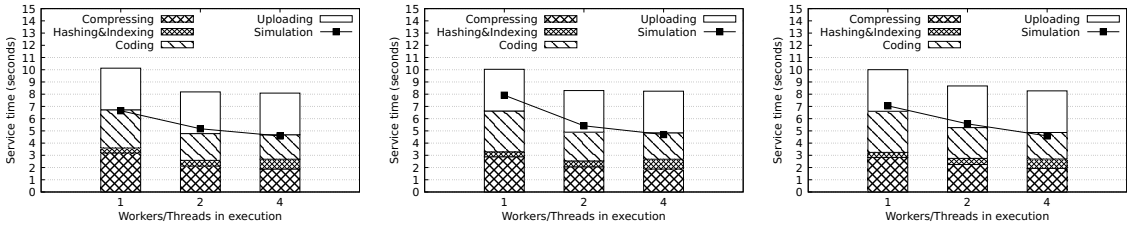
7.2. Phase 1: IoT data analytics in edge-fog environments

In this phase is conducted a case study based on processing environmental data records produced by sensors of IoT devices (see characteristics of these sensors in Table 4). These sensors obtained environmental data such as temperature, humidity, atmospheric pressure, solar radiation, etc.

7.2.1. From IoT sensors to the edge-fog study: prototype details

The prototype was used to evaluate the overlay at the edge. In the prototype, an edge-fog IoT solution was created in the form of a pipeline of two $BBoxes$, and one was deployed on the edge (WSN) and the other one on the fog.

The first $BBox$ included a pipeline of two applications encapsulated into three BBs: the first BB acquired the data produced by eight sensors with the following settings: *i)* low packet rate with all the eight sensors sending 14 packets per second (pps), with 55 samples each. *ii)* high packet rate with all the eight sensors sending 128 pps, with 16 samples each. *iii)* a mixed configuration with four sensors sending 14 pps (55 samples) and four sensors sending 128 pps (16 samples) The second BB processed the acquired data to identify and



(a) 14 pps. (b) 128 pps. (c) Combined.

Figure 11: Response time, at the edge, for preparing traces obtained from eight sensors transmitting samples at different rates.

remove outliers, whereas the third one prepared the data (compressed, encoded to withstand failures, encrypted to ensure confidentiality and producing a hash for future integrity verification) before to send them to the fog.

The last *BBox*, deployed on the fog, received the data and indexed the contents to be used in the future for analytic processes.

7.2.2. Experiments, configurations, and performance evaluation results

Three different configurations were created for this IoT solution. The characteristics of the configurations and the corresponding performance are showed Table 4. The experiments were performed as follows: each sensor sends 40,000 samples in total regardless of the packet rate. Note in Table 4 shows that the lower response time is for the low packet rate configuration. The 14 pps is the lower contention at the MAC layer in comparison with the other two configurations. In the WSN was implemented the contention-based access media control (MAC) mechanism CSMA/CA, which means that the number of active nodes (transmitting) is closely related to the contention at the MAC layer, thus the network performance worsens as the number of active nodes increases.

Table 4 shows that for configurations two and three with a high packet rate (128 pps) nodes, the response time increases compared with configuration one (all low packet rate nodes). This behavior can be explained because sensors are transmitting at a higher packet rate (128 pps), which increases the contention at the MAC layer and thus increases the interarrival time of packets at the server. Therefore, the response time of the server increases, as it is measured at the end of the experiment.

For each sensor transmitting packages, the hub creates one new file that is prepared and transmitted to the fog. Figure 11 shows three plots with the results obtained to prepare the files generated after test each configuration of the eight sensors. Each plot in the vertical axis shows the service time observed to prepare and upload to the fog the eight files by using a different number of workers in the preparation schemes (horizontal axis). The bars in the plot depict the service time of the prototype whereas the lines depict the service time of the produced by the simulator. The average size of each file generated with the $40,000$ packages was 1.95 MB, which was prepared for sending forward to the end-users consuming this data (cost-efficiency by compression, integrity by hashing, and fault tolerance by dispersal information coding).

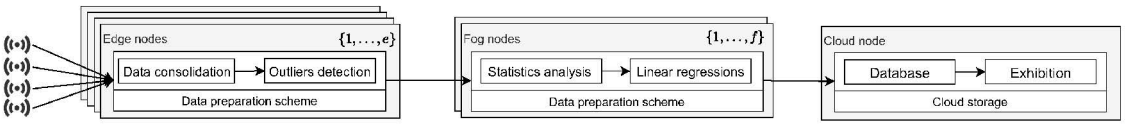


Figure 12: Representation of an IoT processing solution for the analysis of meteorological weather records from the sensors to the cloud.

Figures 11a, 11b, and 11c depict the results when processing files generated by the sensors in 8/14, 8/128, and $4/14, 4/128$ configurations respectively. The service time observed to process each set of files by using four workers in the parallel patterns were of 8.08 seconds, 8.24 seconds, and 8.27 seconds respectively for each configuration. Comparing the results with one and four workers in the pattern, we can observe a reduction in the service time of 20.14%, 17.80%, and 17.35% for each configuration in the sensors respectively.

Results in this section show that the proposed overlay can be used to enable continuous processing of IoT data at the edge, sending the records collected to the fog (or the cloud) with the accomplishment of non-functional requirements (efficiency, reliability, and security).

7.3. Phase 2: Fog-Cloud-EndUser solutions for processing big IoT data

This phase of the evaluation considers two studies: a controlled evaluation in laboratory for the fine-tuning of prototype and simulators as well as a case study based on a comparative performance evaluation between the overlay model with state-of-the-art solutions.

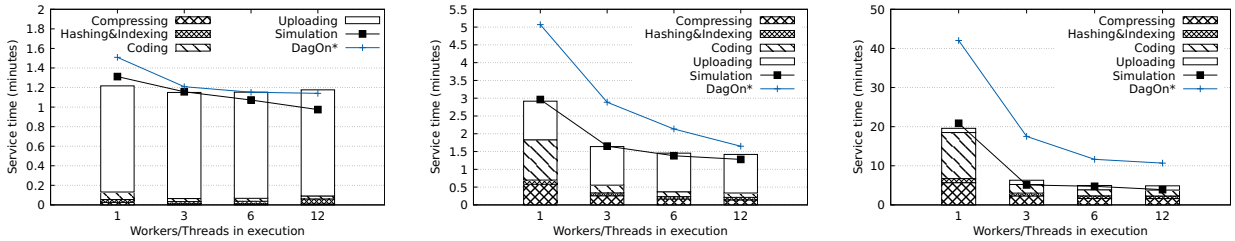
7.3.1. Controlled evaluation based on processing climatic data

This study is based on an IoT solution that was implemented to conduct a fine-tuning of the prototype components, which was based on a service of analysis of meteorological weather records from the sensors to the cloud (see a conceptual representation of this IoT solution in Figure 14). This laboratory study also considers the validation of the simulator of the preparation pipelines and parallel patterns. The results of this simulator are compared with the results produced by both the IoT solution implemented in the prototype as well as a solution from the state-of-the-art.

IoT solution for processing climatic data: prototype details. In this study, for evaluation purposes, a trace generator produced a set of s traces of IoT data based on the data acquired by the sensors described and evaluated in phase 1 of the evaluation. In this context, s represents the number of sensors considered in the simulation.

In this way, different experiments could be performed in laboratory varying the number of traces (data produced by a sensor) that were processed by the IoT solution created for this case study. These traces were used as data source for an IoT solution that considered four $BBoxes$ (one per edge, fog, Cloud and End-users)

During the experiments, a BB executed in the BBox deployed on the edge retrieved the traces starting the processing of the IoT data. In that BBox, a BB processed the traces extracted from different sensors and consolidated them in a unique format. A preprocessing BB received these consolidated IoT data and executed an outlier detection process.



(a) 10 traces. (b) 100 traces. (c) 1000 traces.

Figure 13: Service time spent to process 10, 100, and 1000 traces produced with the sensor simulator.

The preprocessed data were forwarded to the BBox deployed on the fog, where f BBs were also deployed on. These processing BBs calculate statistical measures such as the mean, standard deviation, median, and the number of values over a predefined threshold as well as linear regression for each variable in the structured IoT data preprocessed at the edge nodes.

The results produced by the BBs deployed on the fog were sent to the $BBox$ deployed on the cloud, where a BB stored this information and indexed in a database, which is consumed by an exhibition by end-users by using a visualization BB.

Experiment details, studied solutions, simulator configurations and results. The following experiments were performed with this IoT solution: 10, 100, and 1000 traces each including 1000 records with a packet size average of 20.9 KBs, that were processed and the metrics were captured for each experiment to calculate the effects and costs of the preparation schemes on the IoT solution performance.

These experiments were defined to respond to the following questions:

1. What are the costs of each BB of the preparation scheme enforced by the overlay model?

2. How accurate is the simulator of patterns used in the preparation schemes?

3. Is the performance of the IoT solutions developed in this study competitive in comparison with solutions from the state-of-the-art?

Three configurations were evaluated in this study: *i) Overlay*: this configuration represents the IoT processing solution defined in this study implemented in the prototype. *ii) Simulator*: this configuration represents the IoT processing solution defined in this study but implemented in the simulator. *iii) DagOn** [50]: this configuration represents a workflow engine focused on IoT environments. The results of configurations *Overlay*, *Simulator* and $DagOn*$ will be used to answer the first, second and third stabled questions respectively.

Figure 13 shows the results obtained for the simulation (for *Simulation* configuration) and execution (for *overlay* and $DagOn*$) of the IoT solution processing the three subsets of 10 (Figure 13a), 100 (Figure 13b) and 1000 (Figure 13c) files produced by the sensor

Table 5: Percentual error obtained in each BB.

Number of files	Compressing (%)	Hashing (%)	Indexing (%)	Coding (%)
1	2.711	15.466	2.155	9.624
100	0.593	10.352	5.639	15.247
1000	-6.601	6.022	-1.786	-11.966

simulator. As can be seen, the more the workload (traces) and the more workers included the patterns, the fewer service times for the three studied configurations.

For the first stabled question about the costs of the preparation schemes (cost-efficiency, integrity and fault tolerance), the answer is: the more large the workload is, the less significant are the costs of the transportation among the environments ($edge \rightarrow fog \rightarrow cloud$). In this context, when the IoT solution is large (e.g., 1000 traces of 1000 records) the costs of processing are more significant than the transportation data, which makes suitable this solution for big IoT data scenarios.

For the second question about the accuracy of the simulator, the answer is the simulator follows the trend showed by the prototype with real-world data sets. Interestingly, the system scales well with the number of files, as the execution time grows proportionally less than the increase in the number of files. Another interesting result of the simulator is that the optimum number of workers can be easily chosen. As may be seen, this optimum is 12 workers for the platform simulated and those datasets.

Table 5 shows the error (in %) between real dataset processed and its simulation for each BB. Negative errors mean that the simulator is faster than the prototype. As may be seen, the simulator error varies depending on the functionality of the BB, as each service is sensible to the number of files processed. In general compression and indexing have the lower average errors (around 3.3% in absolute value) for all cases, while hashing and coding errors are higher (around 10% and 11% respectively). Compression and hashing are sensible to the number of files. Hashing and coding errors are mostly due to the differences in the estimation of file open and read operations from the operating system. Adjustments will be made in future versions of the simulator to capture more accurately the behavior of the queues produced by the data exchange per each pair of BBs as well as the queues of the tasks assigned to the virtual cores.

Nevertheless, the goal of the simulator is to deliver an estimation of the impact of parallel patterns on the service times of the BBs for organizations could simulate large scale scenarios. In this aspect, it is interesting to see that for the whole overlay preparation process, the error decreases when the number of files increases (7.5% for 10 files and 3.5% for 1000 files), which is coherent with the results of the prototype on real datasets. Moreover, the simulator error rate got seems acceptable to provide a fast estimation of results.

For the third question about the performance comparison between DagOn* and the Overlay model, the answer can be obtained from Figure 13c, where it can be observed that the proposed Overlay processed the 1000 traces produced by 1000 simulated sensors in 5.80 minutes by using 12 workers in the pattern, whereas DagOn* using 12 threads processed the same data in 10.68 minutes, this means that the overlay has an improvement in the service

Table 6: Datasets used for experimentation.

Region	Number of files	Avg. file (MB)	Size (GB)	Years	Sampling interval
Spain	44	51.77	2.2	2011	10 minutes
Mexico	4386	0.45	2.1	1985-2018	Diary

time 54.30%. Moreover, the overlay not only executes the data analytic of data produced by IoT devices but also, adds properties to the data, which is not offered by DagOn*. In addition, it was observed that the more the large workload produced by IoT devices, the more the efficiency of the parallel patterns, which compensates the time spent by the overlay to prepare IoT data. Notice that the patterns are not only processing data in a concurrent manner but also are balancing the load per worker and reducing the number of expensive I/O requests (file systems and network) by using in-memory schemes, whereas the DagOn* only produce implicit management of threads.

7.3.2. Comparative performance evaluation between the overlay model with state-of-the-art solutions

To offer a better answer to the third question formulated in the previous section, additional compression is performed with other solutions from the state-of-the-art in the next case study. This case study is based on the implementation of a fog-cloud-EndUser IoT processing solution for performing a classification analytic procedure. This IoT solution was evaluated with two environmental data repositories produced by sensors of ground stations deployed in Mexico and Spain. This study is conducted in two directions: the first one performs a validation of the simulator in this type of scenario and the second one is focused on a comparison with state-of-the-art solutions (both transport tools and frameworks to build processing structures).

Repositories experimental scenarios, and prototyping details. The datasets contain records captured by sensors of ground stations distributed through the territory of two countries: Mexico and Spain (see Table 6). The Mexican dataset includes records captured by the Mexican weather station network (EMAS)[3], which covers the 32 states of the Mexican territory. Each station of the EMAS system has registered weather metrics, such as maximum and minimum temperatures (measured in Celsius degrees), as well as the precipitation (measured in millimeters), for 33 years (from 1985 to 2018). The Spanish dataset includes records from 56 stations that cover the whole Spanish territory. Each ground station has registered weather metrics, such as a timestamp of the record, precipitations, wind speed, temperature, and relative humidity. This dataset only considers records of the 2011 year, which were captured by sensors in a constant period of 10 minutes in a 24/7 manner.

Figure 14 shows the *BBoxes* included in the IoT processing solution, which was developed both in the prototype (described in Section 6) and the simulator (described in Section 5) to process the climate repositories from the fog to deliver useful information to

[3]`https://smn.cna.gob.mx/es/observando-el-tiempo/estaciones-meteorologicas-automaticas-ema-s`

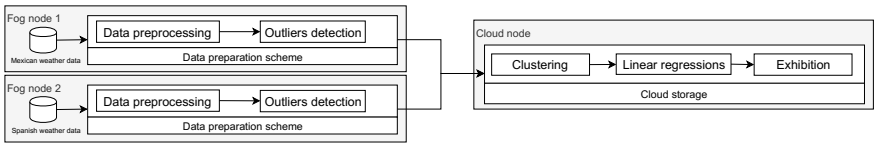


Figure 14: Experiment 1: simulation scenario. Processing environmental data from the fog to the cloud.

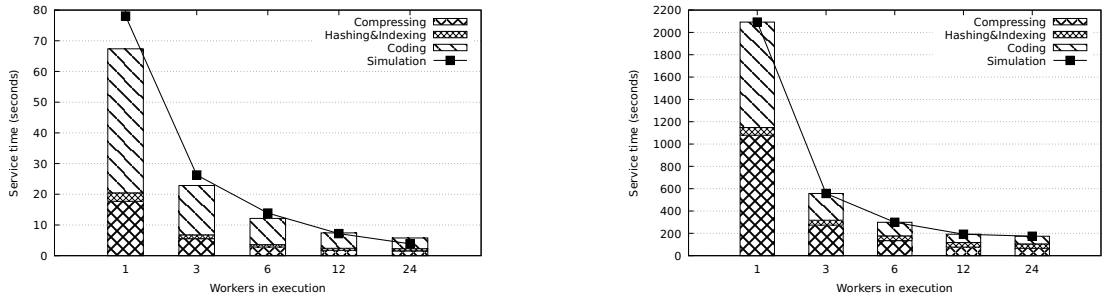
the cloud. Figure 14 shows that this IoT processing solution starts with a *BBox* deployed on the fog, where the records are placed in the form of text files. These files are processed in two different fog nodes: one for processing Mexican records and another one for Spanish records. In the fog *BBox*, at the preservation plane, the files are sent by the proxy to the processing and preservation planes where both raw and analyzed/classified IoT data are prepared for sending them to the cloud in secure, efficient and reliable manners by invoking a preparation pipeline including *BBs*.

In parallel, at the processing *BBs* performs tasks such as removing outliers of temperatures over the thresholds established for each region analyzed and the results are sent, as structured data, to the preservation plane to make them available for future processing operations and forwarding control to the *BBox* deployed on the cloud. All the processed files from the fog nodes are sent to the cloud by using SkyCDS [66], which is connected to the pub/sub of the overlay. Therefore, each data forwarded by the fog *BBox* is available for other *BBoxes* by using subscriptions of the catalog used by SkyCDS when sending data to the cloud.

In the cloud *BBox*, the control plane executes an overlay for the proxy to send the data received from the fog to the control plane where the preprocessing/processing stage invokes a set of *BBs* that executes a distributed clustering algorithm based on K-Means. The clustering algorithm was implemented as a service based on a microservice architecture [70]. This pipeline also includes *BBs* including a linear regression service in charge of obtaining trend graphics for each group of stations and an exhibition service for creating graphs based both on the clustering and the linear regression. Of course, the resultant data are also prepared to be sent to the cloud storage associated with the *BBox* and these information assets are available by connecting the catalogs of SkyCDS with the pub/sub plane.

Experiment details, studied solutions and results. The experiments in this study were performed to answer the question about the efficiency of the overlay model to create continuous processing IoT data from the edge to the cloud in a secure and reliable manner in comparison with solution from the state-of-the-art. The IoT data processing solution described in the previous section was thus implemented in the following solutions:

- *Rsync*: is a traditional tool for syncing local and cloud locations by compressing and encrypting data during data transportation. The preprocessing/processing of data, the establishment attribute-based access controls, the verification of integrity, and the establishment of fault-tolerance are not considered by this solution.
- *SCP*: is a traditional solution used for sending data to the cloud in a secure manner as this tool is based on a private/public key model. Thus, it includes data preparation not



(a) Spanish dataset.

(b) Spanish dataset.

Figure 15: Service time spent by the preparation scheme to process sensor datasets with a varying number of workers.

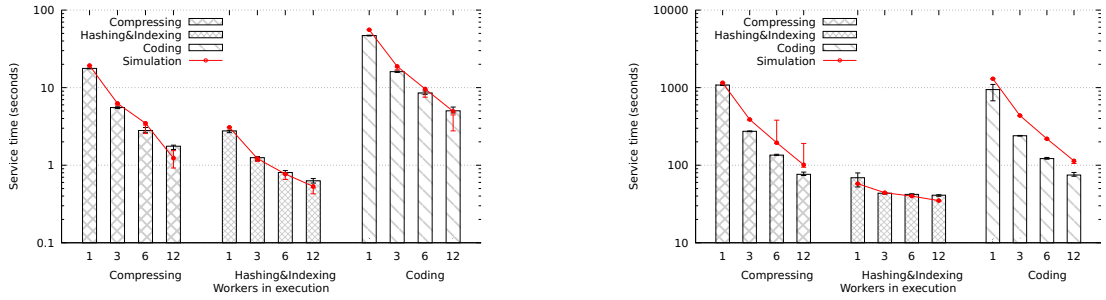
only during transportation but also end-to-end and produces reduced access control based on a primary key concept.

- *DagOn** [50]: this platform creates workflows commonly used to process climate data and IoT in the cloud.
- *Jenkins* [71]: it is a platform that enables to create CI/CD software pipelines automatizing the process of building, testing, and deployment of software solutions.
- *Makeflow* [48]: it is a workflow engine that allows the automatization in the execution of workflows in clusters, clouds or containerized environments.
- *Overlay*: this solution represents the IoT processing solution created by using the model proposed in this paper and implemented in the previously described prototype.

The experiments were performed in two phases. In the first one, the repositories were processed by using the overlay prototype and the simulator to measure the accuracy of the simulator when evaluating parallel patterns. In the second phase, the performance of the workflow of $BBoxes$ implementing overlays in the prototype is compared with the solutions of the state-of-the-art previously described in this section.

Simulator validation. In this phase of the case study, both the speed-up of the parallel patterns and the behavior of the prototype and the simulator were evaluated when processing the datasets defined in this case.

Figure 15a shows the service time when processing all the records of the Spanish dataset (vertical axis) for a different number of workers (horizontal axis) used in the patterns executed in each BB in the preparation pipeline executed at the preservation plane. As expected, the implicit parallel patterns used by each BB reduces the service time of these BBs. Coding and compressing represent the bottleneck in the pipeline. As it can be observed, the simulator behavior closely follows the performance produced by the prototype, with accuracy enhancing when parallel patterns are executed (when more than 1 worker is used in the pattern). Figure 15b, shows the service times of the BBs when processing repositories of the sensors of Mexican ground stations varying the number of workers. In



(a) Spanish dataset.

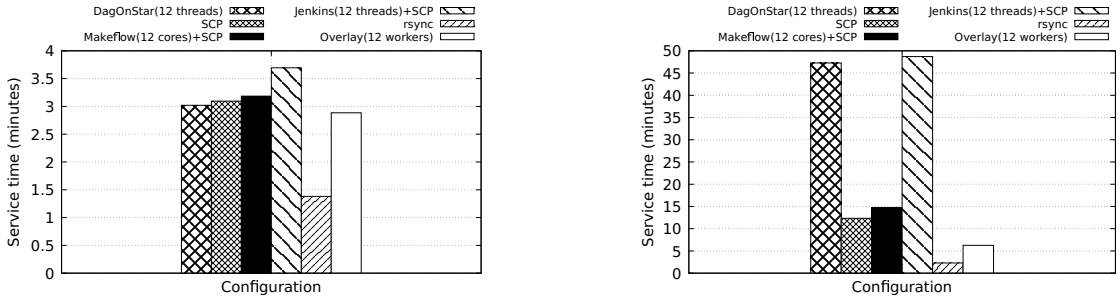
(b) Mexican dataset.

Figure 16: Service time spent by each BB in the pipeline to process the datasets.

this repository, more tasks are performed as the measurements were recorded each day of the 33 years considered in the records of this repository. As may be seen, the behavior of the simulator follows well that of the prototype. It is interesting to see how the number of files affects each BB. Hashing and indexing are scaling well. However, coding and compression time increases even if the files are smaller. This behavior is due to the file operations in the operating system and the initialization times. The conclusion to tune the proxy is that, in general, it is better to have fewer files ranging from several megabytes as input.

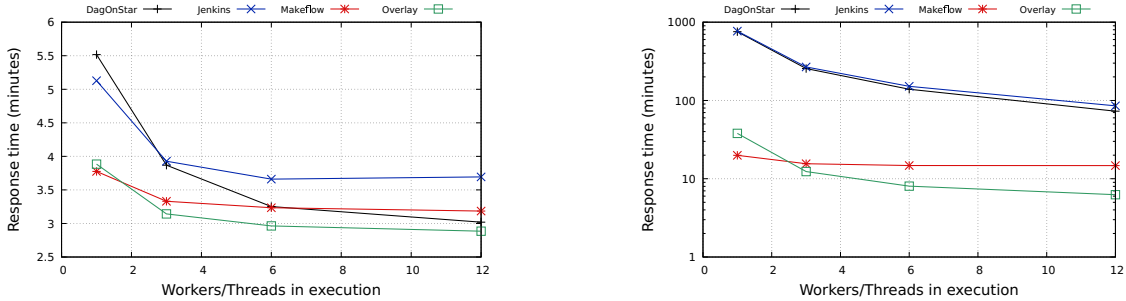
Figure 16a shows results similar to those shown in Figure 15a, but showing the effects of the parallel patterns on each BB when varying the number of workers. As expected, not only the pipeline improves the performance when increasing the number of workers as shown in Figure 15a, but also all the BBs improves its performance in a similar way. Similar trends can be observed in Figure 16b, where more tasks are performed. For compressing and coding, a speedup of ten is mostly achieved, showing very good scalability for the prototype. As can be seen, the simulator mimics the behavior of the prototype for both datasets. Nevertheless, expensive tasks (BBs for compressing and coding) have a higher deviation, which indicates that simulator should consider, not only the parallel pattern characterization but improving the characterization of the queues of the interaction of the BBs in the pipeline.

Results of performance comparative study. In this phase of the case study is performed a direct comparison of the $Overlay$ configuration with $DagOn*$ [50], SCP, $rsync$, $Jenkins$ [49] and $Makeflow$ [48]. In these engines, the data upload stage was performed by using SCP as indicated in the documentation of these solutions. Figure 17a and Figure 17b show, in the vertical axis, the service time spent by the solutions to process the Spanish dataset and the Mexican dataset respectively. Figure 17 shows that $Overlay$ configuration processes and transfers IoT data in less time than DagOn*, Jenkins, Makeflow, and even SCP. Figure 17a shows that the Overlay process and transfer the data in 2.88 minutes, whereas DagOn*, Makeflow, and Jenkins perform the same operation in 3.01 minutes, 3.18 minutes, and 3.69 minutes respectively, which means a percentage of gain of 4.45%, 9.40%, and 21.89% in comparison with DagOn* Makeflow and Jenkins respectively. Comparing the Overlay with SCP, it can be observed a gain in the service time of 6.75%, given that SCP transfers the data to the cloud in 3.09 minutes. Rsync is the solution that yields the best service time of the four



(a) Spanish dataset. (b) Mexican dataset.

Figure 17: Direct comparison with solutions in the state of the art to process and transfer data to the cloud.



(a) Spanish dataset. (b) Mexican dataset.

Figure 18: Direct comparison with workflows engines by using different parallelism configurations in the number of concurrent tasks.

solutions tested as it only uploads the contents to the cloud in 1.38 minutes. Nevertheless, this time does not consider the execution of the IoT processing solution applications as only the processed data at the cloud are delivered to the end-users. This means this time should consider the service times of the IoT solution.

Figure 17b shows the service time spent by each solution to process the Mexican dataset. The difference in the percentage of performance gain between *Overlay* and the other solutions implementing the IoT solution (*DagOn∗*, *Jenkins* and *Makeflow*) is increased in comparison with the results obtained when processing the Spanish dataset. This effect is caused by the fact of the Mexican dataset includes more records than the Spanish one. Overlay processes this dataset in 6.25 minutes, DagOn* in 47.27 minutes, Makeflow in 14.71 minutes, Jenkins in 85.32 for a performance gain percentage of Overlay of 86.76%, 57.46%, 92.66% respectively. Again, rsync and SCP only upload the processed data in 2.30 and 12.32 minutes respectively. Nevertheless, the time produced by rsync only represents the transportation time as this solution only sends data to the cloud by syncing local and cloud locations. These solutions are only presented to show the overhead of the IoT processing solutions (without data transportation).

Once shown the overhead of the IoT solutions implemented in frameworks based on continuous schemes such (*DagOn∗*, *Jenkins*, *Makeflow*, and *Overlay*), a direct comparison

is now showed in Figure 18. Figure 18 shows, in vertical axis, the response time produced by the studied solutions when varying the number of workers/threads (horizontal axis) when processing Spanish (see 18a) and Mexican (see Figure 18b) datasets. As it can be seen, the Overlay configuration produced a better response time than Jenkins (12 threads), Makeflow (12 threads) and DagOn* (12 threads) by 4.31%, 21.95%, 9.43% in comparison with DagOn*, Jenkins, and Makeflow respectively When processing the Spanish dataset. In turn, the Overlay configurations produced better response time than Jenkins (12 threads), Makeflow (12 threads), and DagOn* (12 threads) by 92.67%, 57.51% and 91.43% respectively when conducting the case study based on Mexican dataset.

Notice that in the model proposed in this paper, the parallelism operations are managed by a service mesh (pattern mesh) including a non-deterministic load balancing algorithm based on the utilization of virtual containers as well as an in-memory data management. In turn, traditional solutions only launch threads per task in a virtual container.

8. Discussion and analysis of strengths, weaknesses, opportunities, and threats

In a direct comparison of the model with state-of-the-art tools, the evaluation revealed that the parallel patterns not only reduced the overhead of preparation schemes but also produced a better performance than Jenkins, Makeflow, and DagOn*. Note that the pub/sub makes available not only the resultant information produced by processing IoT data but also all the versions produced in the IoT data lifecycle. Moreover, the properties added to the IoT data not only produce reliability, portability, and security but also improve the efficiency of the IoT processing, which is significant in big IoT data. Besides these findings, in this section is presented an analysis of strengths, weaknesses, and opportunities to provide a comprehensive view of the model proposed:

A continuous delivery scheme based on BBs for constructing software pipelines from the edge to the cloud. **Strengths**: It is presented as a processing model for big IoT data, which is based on reusable abstract structures ($BBoxes$ and BBs). Big IoT data solutions based on pipelines and implicit parallelism (patterns) can be deployed on the edge, cloud, fog, or end-user. **Weaknesses**: Developers require a previous background to understand how to define a DAG and the functionality of abstractions ($BBoxes$, BBs, and overlays). **Opportunities**: An intuitive and graphical tool should be developed to hide from the developers the details of creating DAGs. Moreover, other patterns besides the patterns implemented in the prototype can be explored (e.g., *master/slave* or fabric).

A data preparation scheme based on parallel patterns and an overlay structure. **Strengths**: The proposed overlay allows developers to create separated layers to different behaviors, which can be updated by adding or removing stages following the proposed model. Also, stages share data in a secure manner. **Weaknesses**: The overlay only includes three planes and how these planes can be modified (reduced or increased) is not considered in this paper. **Opportunities**: The model requires another layer for managing an overlay plane mesh in a similar way to the pipeline service mesh and the pattern mesh.

A parallel pattern and software pipeline simulator based on containerized BBs. **Strengths**: This component enables to measure the costs of creating preparation schemes for big IoT data solutions in terms of capacity redundancy and performance, to fine-tune the parameters of the patterns of these BBs. **Weaknesses**: This simulator only was validated for the composition of pipelines for parallel patterns used in the preparation schemes. As shown in the related work, the big IoT data processing is a complex multidimensional issue. This simulator does not cover all the aspects of this complex issue. **Opportunities**: Other BBs (e.g., for clustering and classification analytic) should be simulated, which can be used as a DevOps mechanism by the pipelines in decision-making procedures.

The implementation of the prototype and its evaluation. **Strengths**: The experimental evaluation revealed the efficacy of the prototype to deploy IoT solutions on different environments, as well as the efficiency to cost-efficiently preserve the security, reliability, and integrity in comparison with tools from the stage-of-the-art (rsync, SCP, Jenkins, and DagOn*). **Weaknesses**: The prototype is not ready for delivering into production. The acquisition of IoT data only considers a type of sensors. In terms of evaluation, the prototype could be evaluated by using case studies based on other IoT industry scenarios. It is also required a more detailed evaluation for the $BBox$ deployed at the edge based on industry benchmarks. **Opportunities**: More sensor APIs should be added to the $BBox$ at the edge. In terms of evaluation, it would result quite interesting to study digital twins in continuous IoT processing.

Threats are managed in a general manner as the model is supported by a prototype and a simulator. Also, equal or more work than performed on their implementation should be made to convert them into production software. Threats arise in the form of attacks related to the dependability and resilience of these components, which is not addressed in this paper.

9. Conclusions

This paper presented a model implemented as a framework that enables organizations to build flexible big IoT data solutions for processing data, in an uninterrupted manner, from the edge to the fog to the cloud to the end-users' devices, and making feasible to provide these solutions with security, efficiency, and reliability schemes in a cost-efficient manner. This model creates dataflows through end-to-end workflows, which enforces the preparation of data in an ordered sequence of stages executed in implicit, transparent, and automatic manners. An adaptive compression stage reduces the data sent to the fog and the cloud to also reduce the costs of the outsourced preservation data management tasks, a ciphering stage keeps data privacy, a digital signature stage enables end-users to discover alterations of downloaded files, and a segmentation stage adds data redundancy for withstanding unavailability of data or storage locations. Finally, a stage delivers prepared data to different environments (any of edge, fog or cloud).

The model is based on $BBoxes$ and BBs including I/O connectors created by using an ETL model and has shown to be flexible to build data IoT solutions modeled as DAGs. These

structures enable developers to encapsulate any of programs, applications or functions into the BBs, which can be coupled to other $BBoxes$ or BBs by using the principle of continuous delivery through the I/O interfaces that can be configured to use any of the memory, network or the file system.

The evaluation based on use cases revealed the feasibility of applying these data preparation schemes to IoT processing solutions by using implicit parallel patterns and service mesh. The overlay structure enables organizations to establish, in an efficient manner, controls over the production and consumption of IoT data by creating IoT data and information sharing patterns at each environment considered in an IoT processing solution (any of edge, fog, or cloud). The data preparation scheme has been proved to be flexible and efficient allowing it to establish controls over the production and consumption of IoT data by adding data properties such as cost-efficiency storage, security, and reliability. Moreover, the usage of the overlay structure has demonstrated a high potential to integrate the schemes to create IoT data and information sharing patterns.

A simulator of parallel patterns and software pipelines based on virtual containerized BBs was also proposed for organizations to estimate service and response times of preparation schemes for processing IoT datasets. This simulator was designed to make fast estimations of time and cost for the continuous data preparation approach for big IoT data. The simulator has a modular architecture, which allows it to easily include new BBs. Behavior can be modeled using statistic functions or through functions reflecting services. In this case, values are obtained by interpolating values in performance tables. This simulator allows solution designers to explore the performance and the scalability of the solution and to choose the optimum number of workers to dimension a further production deployment.

The experimental evaluation in two real-world uses cases for meteorological sensors data revealed the feasibility of using a data preparation approach for IoT to mitigate risks that still could arise in the cloud. The evaluation showed that heavy usage of parallelism, by providing virtual containers implementing parallel patterns, is beneficial in all steps as it was observed that the larger the workload produced by IoT devices, the more the efficiency of the parallel patterns, which compensates the time spent by the overlay to prepare IoT data. As shown in the evaluation, compared with popular tools such as DagOn*, Makeflow, and Jenkins, a speedup of 11x, 13x, and 2x were achieved respectively. The simulation of the same scenarios showed that the error obtained is low, thus showing the validity of the simulator proposed.

As future and ongoing work, the model is under evaluation with other case studies, such as medical imagery and organizational environments, to explore its generality. The process of the acquisition and delivery of data by performing the write and read operations in parallel is already under evaluation for providing secure and cost-efficiency sync for multi-cloud environments, as data movement is costly and doing it sequentially is a bottleneck in current solutions. The accuracy of the simulation is also being adjusted to new $BBoxes$ and BBs, not only at parallel pattern level where a high accuracy has been observed, but also trying to capture the characteristics of the queuing of requests in the data exchange performed in each pair of BBs in a pipeline as well as when the patterns use virtual cores, which is not sufficiently captured by the version of the simulator presented in this paper.

We are also working on including different communication standards for IoT devices such as LoRa and Bluetooth.

Acknowledgments

This research was partially supported by "Fondo Sectorial de Investigación para la Educación", SEP-CONACyT Mexico, through projects 281565 and 285276, and by Madrid regional Government (Spain) under the grant "Convergencia Big data-Hpc: de los sensores a las Aplicaciones. (CABAHLA-CM)". Ref: S2018/TCS-4423.

References

[1] M. Malik, Internet of things (iot) healthcare market by component (implantable sensor devices, wearable sensor devices, system and software), application (patient monitoring, clinical operation and workflow optimization, clinical imaging, fitness and wellness measurement): Global opportunity analysis and industry forecast, 2014–2021, Allied Market Research (2016) 124.

[2] Miller, World Internet of Things (IoT) in Healthcare Market (Feb 2016).
URL `https://www.alliedmarketresearch.com/iot-healthcare-market`

[3] Reinsel, Gantz, Rydning, The digitization of the world: from edge to core, Framingham: International Data Corporation.

[4] Anawar, Wang, A. Zia, Jadoon, Akram, Raza, Fog computing: An overview of big iot data analytics, Wireless Communications and Mobile Computing 2018.

[5] B. Rimal, E. Choi, I. Lumb, A taxonomy and survey of cloud computing systems, in: 2009 Fifth International Joint Conference on INC, IMS and IDC, Ieee, 2009, pp. 44–51.

[6] V. Malik, S. Singh, Cloud, big data & iot: Risk management, in: 2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon), IEEE, 2019, pp. 258–262.

[7] A. Botta, W. De Donato, V. Persico, A. Pescapé, Integration of cloud computing and internet of things: a survey, Future generation computer systems 56 (2016) 684–700.

[8] Shi, Cao, Zhang, Li, Xu, Edge computing: Vision and challenges, IEEE internet of things journal 3 (5) (2016) 637–646.

[9] Mohan, Kangasharju, Edge-fog cloud: A distributed cloud for internet of things computations, in: 2016 Cloudification of the Internet of Things (CIoT), IEEE, 2016, pp. 1–6.

[10] S. Yi, C. Li, Q. Li, A survey of fog computing: concepts, applications and issues, in: Proceedings of the 2015 workshop on mobile big data, 2015, pp. 37–42.

[11] Naas, Parvedy, Boukhobza, Lemarchand, ifogstor: an iot data placement strategy for fog infrastructure, in: 2017 ICFEC, IEEE, 2017, pp. 97–104.

[12] Singh, Chatterjee, Cloud security issues and challenges: A survey, Journal of Network and Computer Applications 79 (2017) 88–115.

[13] Chow, Golle, Jakobsson, Shi, Staddon, Masuoka, Molina, Controlling data in the cloud: outsourcing computation without outsourcing control, in: CCSW'2009, ACM, 2009, pp. 85–90.

[14] Morales, Gonzalez, Diaz, Sosa, A pairing-based cryptographic approach for data security in the cloud, IJISP 17 (4) (2018) 441–461.

[15] Zhang, Zhang, Secure and efficient data-sharing in clouds, CCPE 27 (8) (2015) 2125–2143.

[16] Gonzalez, Sosa, Diaz, Carretero, Yanez, Sacbe: A building block approach for constructing efficient and flexible end-to-end cloud storage, Journal of Systems and Software 135 (2018) 143–156.

[17] Mao, Wu, Jiang, Improving storage availability in cloud-of-clouds with hybrid redundant data distribution, in: IPDPS 2015m, IEEE, 2015, pp. 633–642.

[18] Xiong, Zhang, Zhu, Yao, Cloudseal: End-to-end content protection in cloud-based storage and delivery services, in: SecureComm, Springer, 2011, pp. 491–500.

[19] Carrizales, Sánchez, Reyes, Gonzalez, Morales, Carretero, Galaviz, A data preparation approach for cloud storage based on containerized parallel patterns, in: 2019 IDCS, Springer, 2019, pp. 478–490.
[20] L. Chen, Continuous delivery: Huge benefits, but challenges too, IEEE Software 32 (2) (2015) 50–54.
[21] Y. Miao, W. Li, D. Tian, M. Hossain, M. Alhamid, Narrowband internet of things: simulation and modeling, IEEE Internet of Things Journal 5 (4) (2017) 2304–2314.
[22] Polley, Blazakis, McGee, Rusk, Baras, Atemu: a fine-grained sensor network simulator, in: SECON 2004, IEEE, 2004, pp. 145–152.
[23] Acharya, Ranjan, Mandal, A novel approach for deployment and throughput analysis of lte-advanced self-organizing network by using system vue, International Journal of Computational Intelligence & IoT 1 (1).
[24] S. Sotiriadis, N. Bessis, E. Asimakopoulou, N. Mustafee, Towards simulating the internet of things, in: 2014 AINA, IEEE, 2014, pp. 444–448.
[25] H. Gupta, A. Vahid Dastjerdi, S. Ghosh, R. Buyya, ifogsim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments, Software: Practice and Experience 47 (9) (2017) 1275–1296.
[26] A. Murthy, Mumak: Map-reduce simulator, MAPREDUCE-728, Apache JIRA.
[27] S. Hammoud, M. Li, Y. Liu, N. Alham, Z. Liu, Mrsim: A discrete event based mapreduce simulator, in: 2010 FSKD, Vol. 6, IEEE, 2010, pp. 2993–2997.
[28] M. Tran, D. Nguyen, V. Le, D. Nguyen, T. Pham, Task placement on fog computing made efficient for iot application provision, Wireless Communications and Mobile Computing 2019.
[29] Y. Song, S. Yau, R. Yu, X. Zhang, G. Xue, An approach to qos-based task distribution in edge computing networks for iot applications, in: 2017 IEEE international conference on edge computing (EDGE), IEEE, 2017, pp. 32–39.
[30] R. Calheiros, R. Ranjan, A. Beloglazov, C. De Rose, R. Buyya, Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms, Software: Practice and experience 41 (1) (2011) 23–50.
[31] Zeng, Garg, Strazdins, Jayaraman, Georgakopoulos, Ranjan, Iotsim: A simulator for analysing iot applications, Journal of Systems Architecture 72 (2017) 93–107.
[32] Lin, Xu, He, Li, Multi-resource scheduling and power simulation for cloud computing, Information Sciences 397 (2017) 168–186.
[33] G. Wang, A. Butt, P. Pandey, K. Gupta, A simulation approach to evaluating design decisions in mapreduce setups, in: 2009 MASCOTS, IEEE, 2009, pp. 1–11.
[34] A. Verma, L. Cherkasova, R. Campbell, Play it again, simmr!, in: 2011 IEEE International Conference on Cluster Computing, IEEE, 2011, pp. 253–261.
[35] J. Jung, H. Kim, Mr-cloudsim: Designing and implementing mapreduce computing model on cloudsim, in: 2012 International Conference on ICT Convergence (ICTC), IEEE, 2012, pp. 504–509.
[36] G. Brambilla, M. Picone, S. Cirani, M. Amoretti, F. Zanichelli, A simulation platform for large-scale internet of things scenarios in urban environments, in: 1st EAI Urb-IoT 2020, 2014, pp. 50–55.
[37] Han, Lee, Crespi, Heo, V. Luong, Brut, Gatellier, Dpwsim: A simulation toolkit for iot applications using devices profile for web services, in: 2014 WF-IoT, IEEE, 2014, pp. 544–547.
[38] G. Kecskemeti, Dissect-cf: a simulator to foster energy-aware scheduling in infrastructure clouds, Simulation Modelling Practice and Theory 58 (2015) 188–218.
[39] Cárdenas, Arroba, Blanco, Malagón, Risco-Martín, Moya, Mercury: A modeling, simulation, and optimization framework for data stream-oriented iot applications, SIMPAT.
[40] Huang, Li, Descriptive models for internet of things, in: 2010 International Conference on Intelligent Control and Information Processing, IEEE, 2010, pp. 483–486.
[41] Xue, Li, Nazarian, Bogdan, Fundamental challenges toward making the iot a reachable reality: A model-centric investigation, ACM TODAES 22 (3) (2017) 1–25.
[42] Francis, Gerstlauer, A reactive and adaptive data flow model for network-of-system specification, IEEE Embedded Systems Letters 9 (4) (2017) 121–124.
[43] ur Rehman, Ahmed, Yaqoob, Hashem, Imran, Ahmad, Big data analytics in industrial iot using a

concentric computing model, IEEE Communications Magazine 56 (2) (2018) 37–43.
[44] H. Bergius, Noflo–flow-based programming for javascript, URL: http://noflojs. org.
[45] Persson, Angelsmark, Calvin-merging cloud and iot., in: ANT/SEIT, 2015, pp. 210–217.
[46] Jabbar, Ullah, Khalid, Khan, Han, Semantic interoperability in heterogeneous iot infrastructure for healthcare, Wireless Communications and Mobile Computing 2017.
[47] Phillips, International data-sharing norms: from the oecd to the general data protection regulation (gdpr), Human genetics 137 (8) (2018) 575–582.
[48] Albrecht, Donnelly, Bui, Thain, Makeflow: A portable abstraction for data intensive computing on clusters, clouds, and grids, in: 2012 SWEET, 2012, pp. 1–13.
[49] J. Smart, Jenkins: The Definitive Guide: Continuous Integration for the Masses, " O'Reilly Media, Inc.", 2011.
[50] R. Montella, D. Di Luccio, S. Kosta, Dagon*: Executing direct acyclic graphs as parallel jobs on anything, in: 2018 WORKS, IEEE, 2018, pp. 64–73.
[51] Babuji, Chard, Foster, Katz, Wilde, Woodard, Wozniak, Parsl: Scalable parallel scripting in python., in: IWSG, 2018, pp. 1–6.
[52] Santiago-Duran, Gonzalez-Compean, Brinkmann, Reyes-Anastacio, Carretero, Montella, Pulido, A gearbox model for processing large volumes of data by using pipeline systems encapsulated into virtual containers, Future Generation Computer Systems.
[53] Wiseman, Schwan, Widener, Efficient end to end data exchange using configurable compression, ACM SIGOPS Operating Systems Review 39 (3) (2005) 4–23.
[54] Meister, Brinkmann, Multi-level comparison of data deduplication in a backup scenario, in: Proceedings of SYSTOR 2009., ACM, 2009, p. 8.
[55] K. Miller, Cloud deduplication, on-demand: Storreduce, an apn technology partner: Amazon web services (Mar 2018).
[56] Meister, Brinkmann, dedupv1: Improving deduplication throughput using solid state drives (ssd), in: MSST 2010, IEEE, 2010, pp. 1–6.
[57] Chen, Lee, Enabling data integrity protection in regenerating-coding-based cloud storage: Theory and implementation, IEEE transactions on parallel and distributed systems 25 (2) (2013) 407–416.
[58] Liu, Yang, Zhang, Chen, External integrity verification for outsourced big data in cloud and iot: A big picture, Future generation computer systems 49 (2015) 58–67.
[59] Puttaswamy, Kruegel, Zhao, Silverline: toward data confidentiality in storage-intensive cloud applications, in: Proceedings of the 2nd ACM Symposium on Cloud Computing, 2011, pp. 1–13.
[60] M. O. Rabin, Efficient dispersal of information for security, load balancing, and fault tolerance, JACM 36 (2) (1989) 335–348.
[61] Humble, Farley, Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation (Adobe Reader), Pearson Education, 2010.
[62] Buschmann, Henney, Schmidt, Pattern-oriented software architecture, on patterns and pattern languages, Vol. 5, John wiley & sons, 2007.
[63] Ortega-Arjona, Patterns for Parallel Software Design, 1st Edition, Wiley Publishing, 2010.
[64] Bartík, Ubik, Kubalik, Lz4 compression algorithm on fpga, in: 2015 IEEE International Conference on Electronics, Circuits, and Systems (ICECS), IEEE, 2015, pp. 179–182.
[65] Morales-Ferreira, Santiago-Duran, Gaytan-Diaz, Gonzalez, Sosa, Lopez, A data distribution service for cloud and containerized storage based on information dispersal, in: SOSE, IEEE, 2018, pp. 86–95.
[66] Gonzalez, Perez, Sosa-Sosa, Sanchez, Bergua, Skycds: A resilient content delivery service based on diversified cloud storage, SIMPAT 54 (2015) 64–85.
[67] Odelu, Rao, Kumari, Khan, Choo, Pairing-based cp-abe with constant-size ciphertexts and secret keys for cloud environment, Computer Standards & Interfaces 54 (2017) 3–9.
[68] Gkoutioudi, Karatza, Task cluster scheduling in a grid system, Simulation Modelling Practice and Theory 18 (9) (2010) 1242–1252.
[69] J. Ghofrani, D. Lübke, Challenges of microservices architecture: A survey on the state of the practice., in: ZEUS, 2018, pp. 1–8.

[70] Sánchez, Gonzalez, Alvarado, Sosa, Tuxpan, Carretero, A containerized service for clustering and categorization of weather records in the cloud, in: CSIT, IEEE, 2018, pp. 26–31.
[71] N. Pathania, Learning continuous integration with Jenkins: a beginner's guide to implementing continuous integration and continuous delivery using Jenkins 2, Packt Publishing Ltd, 2017.
[72] Mayer, Graser, Gupta, Saurez, Ramachandran, Emufog: Extensible and scalable emulation of large-scale fog computing infrastructures, in: 2017 FWC, IEEE, 2017, pp. 1–6.
[73] Coutinho, Greve, Prazeres, Cardoso, Fogbed: A rapid-prototyping emulation environment for fog computing, in: 2018 ICC, IEEE, 2018, pp. 1–7.
[74] Mohamed, Lorandel, Romain, Regnery, Baheux, A versatile emulator of mitm for the identification of vulnerabilities of iot devices, a case of study: smartphones, in: 3rd ICFNDS, 2019, pp. 1–6.
[75] Chang, Tso, Tsai, Iot sandbox: to analysis iot malware zollard, in: ICC '17, 2017, pp. 1–8.
[76] Le-Trung, Towards an iot network testbed emulated over openstack cloud infrastructure, in: 2017 SigTelCom, IEEE, 2017, pp. 246–251.
[77] Brady, Hava, Perry, Murphy, Magoni, Portillo-Dominguez, Towards an emulated iot test environment for anomaly detection using nemu, in: 2017 GIoTS, IEEE, 2017, pp. 1–6.
[78] Eriksson, Österlind, Finne, Tsiftes, Dunkels, Voigt, Sauter, Marrón, Cooja/mspsim: interoperability testing for wireless sensor networks, in: 2nd SIMUTools, 2009, pp. 1–7.
[79] Nikdel, Gao, Neville, Dockersim: Full-stack simulation of container-based software-as-a-service (saas) cloud deployments and environments, in: 2017 PACRIM, IEEE, 2017, pp. 1–6.
[80] Lin, Xia, Wang, Tian, Song, System design for big data application in emotion-aware healthcare, IEEE Access 4 (2016) 6901–6909.
[81] Cao, Wachowicz, Renso, Carlini, Analytics everywhere: generating insights from the internet of things, IEEE Access 7 (2019) 71749–71769.
[82] Patel, Ali, Sheth, On using the intelligent edge for iot analytics, IEEE Intell. Syst. 32 (5) (2017) 64–69.
[83] Hong, Tsai, Cheng, Uddin, Venkatasubramanian, Hsu, Supporting internet-of-things analytics in a fog computing platform, in: 2017 CloudCom, IEEE, 2017, pp. 138–145.