Data Processing in Cyber-Physical-Social Systems through Edge Computing

Rustem Dautov*, Salvatore Distefano*†, Dario Bruneo†, Francesco Longo†, Giovani Merlino†, Antonio Puliafito†
* Kazan Federal University
Kazan, Russian Federation
{rdautov, s_distefano}@it.kfu.ru
†Università di Messina
Messina, Italy
{sdistefano, dbruneo, flongo, gmerlino, apuliafito}@unime.it

***Abstract*—Cloud and Fog computing have established a convenient and widely adopted approach for computation offloading, where raw data generated by edge devices in the Internet of Things (IoT) context is collected and processed remotely. This vertical offloading pattern, however, typically does not take into account increasingly pressing time constraints of the emerging IoT scenarios, in which numerous data sources, including human agents (i.e. Social IoT), continuously generate large amounts of data to be processed in a timely manner. Big Data solutions could be applied in this respect, provided that networking issues and limitations related to connectivity of edge devices, are properly addressed. Although edge devices are traditionally considered to be resource-constrained, main limitations refer to energy, networking and memory capacities, whereas their ever-growing processing capabilities are already sufficient to be effectively involved in actual (Big Data) processing. In this context, the role of human agents is no longer limited to passive data generation, but can also include their voluntary involvement in relatively complex computations. This way, users can share their personal computational resources (i.e. mobile phones) to support collaborative data processing, thereby turning the existing IoT into a global cyber-physical-social system (CPSS). To this extent, this paper proposes a novel IoT/CPSS data processing pattern based on the Stream Processing technology, aiming to distribute the workload among a cluster of edge devices, involving mobile nodes shared by contributors on a voluntary basis, and paving the way for cluster computing at the Edge. Experiments on an Intelligent Surveillance System deployed on an edge device cluster demonstrate the feasibility of the proposed approach, illustrating how its distributed in-memory data processing architecture can be effective.**

***Index Terms*—Internet of Things; Internet of People; Cyber-Physical-Social System; Edge Computing; Big Data; Stream Processing; Horizontal and Vertical Offloading; Apache NiFi.**

I. Introduction

The increasing demand for efficient network communications and data transferring, as well as the ubiquitous penetration of 'smart' devices in almost every aspect of people's everyday life have been supported by the rapid progress in information and communication technologies. These advancements have boosted the development and wide adoption of the Internet of Things (IoT), and introduced emerging research challenges to be addressed by both industrial practitioners and academic researchers. Among the range of pressing concerns, such as security and interoperability [1], a particularly demanding topic for investigation is computational speed and reaction time of IoT systems in performing complex computational tasks that require extra resources, given the increasing amount of data and time constraints.

This is particularly critical in the light of the increasing number of human agents in the digital world and the active involvement of people in a wide range of cyber-physical processes, leading to the emergence of cyber-physical-social systems (CPSSs). As a result, the amounts of generated data are already growing exponentially, as the number of personal and mobile devices has exceeded 7 billions according to recent statistics.[1] On the other hand, however, the active involvement of people in the IoT opens previously unseen opportunities for leveraging this global collection of personal devices to support various computational activities in participatory and/or opportunistic way, as envisioned by the Mobile Crowdsensing [2] approach.

Involving mobile devices in data processing tasks partially overlaps with the Fog Computing paradigm that has emerged to complement the remote Cloud-based hardware resources with much lower network latency of computational nodes located in close proximity to the actual source of data. This way, Fog/Cloud resources are usually provisioned as elastic on-demand services, typically implementing a 'vertical' offloading pattern. This established way of task offloading, data transferring and processing typically includes three main levels starting from sensing devices at the bottom 'edge' of such topologies, which generate data then transferred through network devices (e.g. gateways, switches, routers) to the server side (e.g. a 'cloudlet' or a public Cloud), to be permanently stored and processed by a dedicated software analytics system. As a result of this analysis, a feedback command may be propagated down the network to enable edge device actuators. In this light, balanced coordination across Fog and Cloud to support the IoT data processing has been widely explored. As a result, existing works aim at enabling resource allocation and orchestration, which would transparently provision containerised resources, finding a right balance between low network latency of the Fog [3], [4], [5], [6], [7] and increased computational capacities of the Cloud [8], [9], [10], [11].

[1]http://www.independent.co.uk/life-style/gadgets-and-tech/news/there-are-officially-more-mobile-devices-than-people-in-the-world-9780518.html

This way, IoT systems can benefit from seemingly infinite computational and storage capabilities of a nearby Fog node or a remote Cloud platform. On the other hand, however, there is a considerable delay between the moment when raw data is first collected and the moment when it is processed, correctly interpreted, and corresponding reactive actions are taken. There are more and more scenarios, indeed, where the established vertical model fails to meet pressing requirements in terms of reaction time and network latency, especially in the presence of considerably large datasets, typically referred to as Big Data. These scenarios may demand for near real-time data processing and reaction, and thus cannot rely on (potentially outdated) results obtained by sending data over the network to a remote location. Admittedly, there are emerging situations, when time delays and network latency cannot be tolerated, and require more timely decision-taking procedures. This becomes particularly challenging in the context of CPSSs involving personal mobile devices, which are typically bandwidth-constrained, calling for novel solutions to address the emerging Big Data issues.

To this end, the presented paper aims to facilitate computationally-intensive processing of large CPSS datasets, using clustered computing techniques on top of wireless communication facilities and exploiting mobile devices contributed by their owners. More specifically, the paper proposes a distributed Stream Processing architecture to enable support for data processing by clustering edge devices and utilising their shared pool of computational resources. By pushing intelligence to the very edge of the network topology – that is, as close to the data source as possible – the proposed architecture aims at minimising the amount of data sent to the server, and thus achieve faster execution results. This way, the proposed solution is able to benefit from the human participation in CPSSs by aggregating personal portable devices and involving them in collaborative data processing activities.

In this light, the main contribution of the paper is five-fold: *i)* a solution for Big Data stream processing at the network edge, implementing the Edge Computing paradigm; *ii)* new collaborative, horizontal offloading patterns for distributed data processing on clustered edge devices; *iii)* a Stream Processing architecture extending Apache NiFi with new services to discover and select devices able to perform an offloaded task according to specific (hardware and software) requirements, as well as to orchestrate the resulting edge cluster; *iv)* a framework providing mechanisms for managing social involvements and contributions in the context of CPSSs; and *v)* a comparison of vertical (i.e. Cloud) and horizontal (i.e. Edge) offloading patterns through an experimental case study.

The rest of the paper is organised as follows. Section II introduces the existing limitations and challenges through a running surveillance system example. Section III explains the main aspects of the proposed approach, whereas Section IV looks into details of the node involvement, covering subscription, authentication and networking mechanisms used to enable *ad-hoc* edge clusters. Section V provides an in-depth description of the proposed clusterisation process of edge devices. Section VI describes the design and implementation of the proof-of-concept prototype, compares the two (i.e. vertical and horizontal) offloading models via benchmarking experiments, and discusses obtained results. Section VII summarises the paper.

II. Motivating Example

The IoT can be seen as an ecosystem of considerably 'smart', network-connected objects interacting to provide services and applications. From this perspective, people are typically seen only as passive users of the IoT services, neglecting the potential opportunity to involve them in the cyber-physical loop. The situation is changing with the increasingly popular social trends on the sharing economy and technological approaches based on the principles of volunteer computing and crowdsourcing. According to these approaches, people are expected to actively play an important role in cyber-physical processes, thus giving rise to the concepts of the Internet of People (IoP) [12] and the Social IoT (SIoT) [13]. The two concepts extend the established IoT with social aspects, highlighting human behaviour, social relationships, and interactions between people and their cyber-counterparts, i.e. personal assets, such as mobile and portable devices. By including people in cyber-physical processes, the traditional digital ecosystem is converted into a cyber-physical-social system [14], [15], defined as "... a kind of common complex system that is constituted by a physical system, its social system including human beings, and the cyber system that connects both of them" [16].

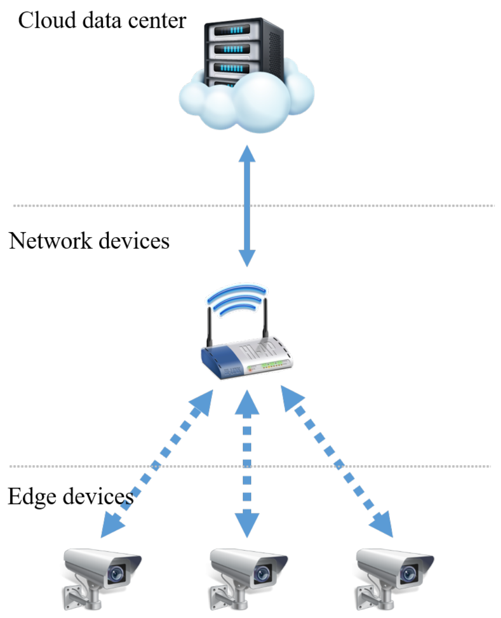


Fig. 1: The 'vertical' offloading pattern in CPSS/ISS.

In the context of the SIoT and IoP, CPSSs are characterised by a constantly growing number of nodes, whose interactions usually generate extreme amounts of data, coming from devices equipped with sensing capabilities. Such IoT data streams have introduced previously-unseen workload on the network communication infrastructures, calling for proper management, collection, storage and processing solutions to address challenges related to (Big) data volume, variety and velocity. Indeed, the ever-increasing amounts of raw data generated in these contexts render the established 'vertical' offloading pattern, depicted in Fig. 1, not scalable enough to support timely data processing. In the first place, this affects the very bottom link of the network topology, which is typically implemented using one of the existing wireless technologies, potentially limited in their network throughput and not necessarily designed to handle large amounts of dynamically generated raw data. Nevertheless, the link between edge devices and IoT gateways is seen as a primary system component to face the Big Data challenges to ensure that large amounts of raw, unprocessed data are dynamically transferred from edge devices to network gateways, located at distances ranging from several meters to several kilometers. As a result, these links often become system bottlenecks with a negative impact on the overall performance of IoT systems. As a result, the delayed data analysis and generation of feedback commands often cannot be tolerated by some mission-critical systems, which rely on timely (i.e. near real-time) operation.

As far as the SIoT is concernd, the involvement of people in the IoT ecosystem further complicates this scenario, but, on the other hand, can introduce a potentially disruptive positive effect on the processing capabilities and performance. In this respect, volunteer-based and crowdsourcing approaches can be exploited to support data processing in a relevant application scenario. This calls for mechanisms and tools to support such approaches, allowing to enrol contributors as well as to manage their random and unpredictable joining and leaving (i.e. churning), in a trusted way.

An example of such emerging CPSS domains, where the increased amount of data, originated by different, wireless connected sources, has to be processed in a timely manner, is Intelligent Surveillance Systems (ISSs) [17], [18]. ISSs are surveillance systems, where the involvement of human operators has been minimised to avoid such shortcomings as, for example, high labour cost or limited capability for multiple screens. ISSs rely on existing technological achievements in computer vision, pattern recognition, and artificial intelligence, used to identify certain patterns (e.g. abnormal behaviour, suspicious objects, missing people, etc.) in video streams. More specifically, ISSs are widely adopted to enable timely detection of crime suspects in crowded public spaces. Usually, such an ISS consists of a number of Internet-connected cameras, installed in a public location (e.g. building, shop, airport, stadium, concert hall, etc.), constantly streaming video to one central node equipped with more powerful computational and storage capabilities. Video streams are then processed using existing image/object recognition techniques to detect suspects and prevent potential crimes/terrorist attacks by alerting police officers nearby.

Object detection and recognition are considered a computationally-intensive task, which cannot always be performed by an edge device (i.e. a CCTV camera) on its own, and therefore usually requires to transfer raw data to an external computational service for analysis. This is illustrated by Fig. 1, in which raw images from a CCTV camera are transferred through a wireless network to a Fog/Cloud server for processing. That is, CCTV cameras are mainly responsible for video capturing and occasionally for simple detection and recognition operations, whereas more complex operations (e.g. object detection/recognition in crowded areas) are usually undertaken on the server side.

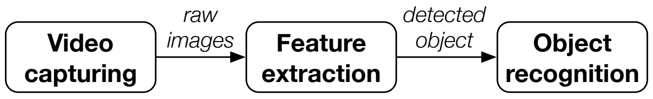


Fig. 2: A typical ISS workflow.

In this respect, a typical ISS workflow can be conceptually split into the following three main steps, as depicted in Fig. 2:

1) *Video capturing* is undertaken by edge CCTV cameras, which continuously capture raw video and transfer it for processing as a continuous video stream or as a sequence of sampled static images. Given the increased adoption of wireless CCTV cameras and the ever-growing image sensor resolution and quality, this may result in extremely large amounts of data being transferred over a wireless network to a remote processing location.
2) *Feature extraction* is usually performed by the server, which applies sophisticated image processing techniques to detect specific elements in the input video. Images, containing detected objects are then transferred for object recognition.
3) *Object recognition* is also usually performed by the server, which recognises detected elements, typically with respect to an input training set.

Arguably, the resulting three-step workflow might take quite long, and a corresponding reactive action (e.g. the police is alerted) might be generated and executed too late. As it follows from this motivating scenario, the performance of an ISS is strongly affected by the quality of its connection, worsening with the number of hops – a limitation hardly addressable within the context of the 'vertical' offloading model due to the inevitable requirement to send data to a remote Fog/Cloud processing location. By the time the Cloud-based face recognition software detects a suspect criminal and sends back a corresponding signal, this person may have already escaped the initial CCTV-covered area. In these circumstances, minimisation (or complete elimination) of the amount of data transferred over the network comes as a natural fit.

III. Proposed Approach

As illustrated by the presented ISS scenario, the CPSS data processing challenges should be addressed by an overarching approach attacking the problem from different perspectives, i.e. Edge/Fog Computing – on the one hand, and Big Data – on the other, converging into a lightweight solution for data processing through computation offloading to collocated edge devices. Given the increasingly important role of human agents in the SIoT, these may include personal and mobile devices, contributed by their owners to support dynamic IoT scenarios, such as the ISS one, by processing computational tasks offloaded by some other closely-located edge device (e.g. a CCTV camera surveilling the local area).

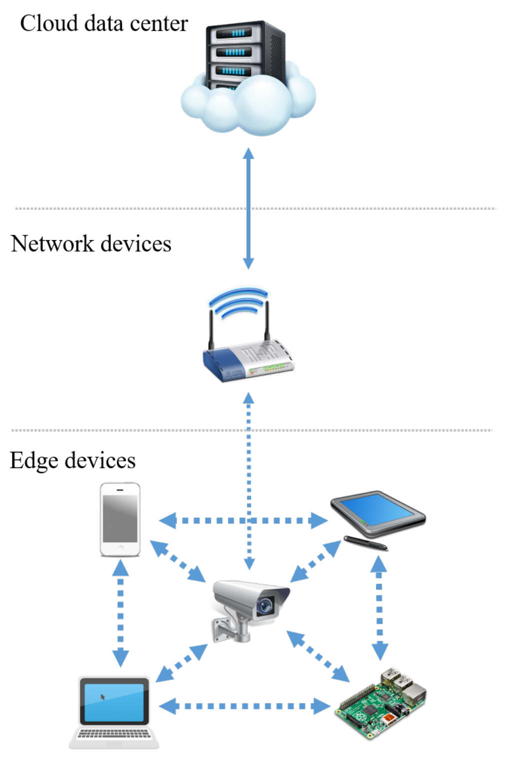


Fig. 3: The 'horizontal' offloading pattern in CPSS/ISS.

Taken together, these considerations propose a solution that aims at implementing the above discussed idea of combining Big Data and Edge Computing approaches into a framework enabling in-memory processing of computational tasks offloaded to a cluster of edge device using Stream Processing techniques, thereby paving the way for the novel approach of Clustered Edge Computing. Three main factors underpin this idea of 'horizontal' offloading using a Stream Processing architecture.

1) Edge (and especially mobile) devices are getting more and more powerful in terms of their hardware specifications (i.e. CPU and RAM). They have advanced beyond the simplistic notion of collecting and transferring raw data, and nowadays act as fully-functional processing units in their own right. They are widely recognised and used as effective computing systems, and are more and more often taken into account for a laptop/desktop PC replacement.
2) The world is experiencing a continuously growing use of embedded and mobile devices in all aspects of people's daily activities. In the IoT era, the world is flooded with all kinds of 'smart' devices, which can be seen as potential contributors to the shared pool of local cluster resources.
3) The last but not the least, despite the increasing processing capabilities of edge devices, they are still relatively limited in their storage functionality and are not yet equipped with full-featured hard disks to store large data sets. In this light, it naturally follows that edge devices are more suited for in-memory data processing – i.e. data processing, which does not write data to a local mass storage, but rather keeps all the computation in memory, thus potentially achieving better performance.

Starting from these three technological trends, the rationale behind the proposed approach is to maximise the amount of computation to be performed on edge devices – i.e. as close to the original source of data as possible – such that minimum amount of data is sent over the network to the Fog/Cloud, and results can be achieved almost immediately on the spot. To this purpose, the traditional 'vertical' data processing pattern and the corresponding IoT reference model [19] should be revised. As opposed to Fog Computing that actively involves networking-level processing units (e.g. gateways, routers, 'cloudlets', etc.), the suggested approach assumes pushing intelligence to the very edge of the network – that is, to end devices, exploiting Fog Computing mechanisms only to coordinate their activities. To implement this, the proposed approach creates an architecture for distributed clusters of edge devices to share computational tasks immediately on the spot on top of Stream Processing middleware.

This is seen as a next step from the current state-of-the-art baseline (i.e. individual edge devices are able to perform data processing only within their computational and storage capabilities, otherwise the data are transferred to the Cloud and/or the Fog) towards an architecture, where the Cloud is not seen as a central component anymore, but only as a secondary processing/storage location. In these circumstances, the primary location for data analytics remains the local distributed cluster of edge devices (i.e. the Edge environment), which are able to spread the incoming workload among themselves, avoiding time delays associated with network latency, and thus achieving better performance. Fig. 3 is intended to demonstrate through the ISS example that the majority of data exchange and computation takes place at the very edge of the network topology. Smart edge devices (e.g. mobile phones and smart CCTV cameras) form a local cluster and are able to spread the workload among themselves. This way, the CCTV camera can distribute video/image processing tasks among worker nodes constituting the local edge cluster.

Clustering, churn management, AAA (authentication, authorisation, and accounting) and security, job distribution and scheduling, data serialisation, and synchronisation are all challenging tasks in their own right, and require an advanced middleware platform, supporting all these activities. As stated above, given the relatively constrained nature of edge devices, such a platform is expected to be lightweight and support in-memory data processing of continuously streaming raw data. Taken together, all these factors have paved the way for Apache NiFi[2] to be employed in the context of the presented research effort as the underlying Stream Processing cluster middleware.

Apache NiFi is an open-source Stream Processing framework, based on the notion of 'Flow-Based Programming'. A data flow conceptually represents a multi-step processing sequence, through which data is streamed. These processing steps, known as 'processors', range from simple mathematical operations to more advanced ones, such as translation or data format conversion. To date, there are more than 100 built-

[2]https://nifi.apache.org/

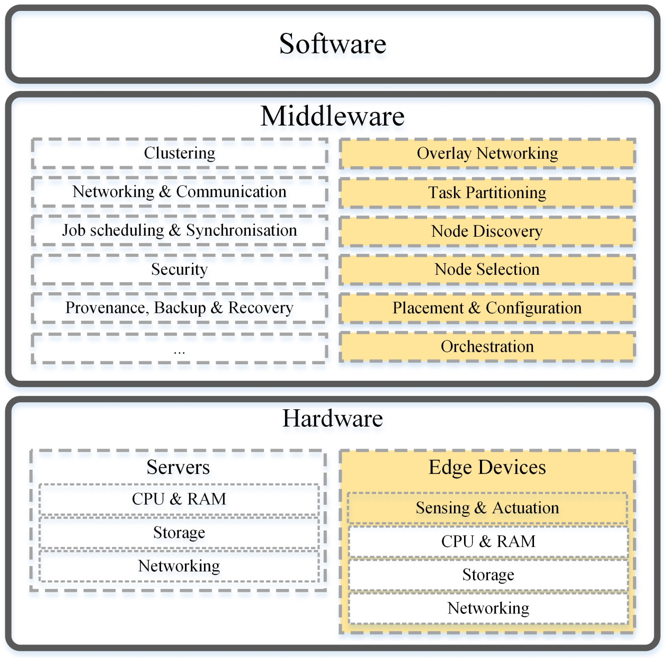


Fig. 4: Three-tier architecture on top of Apache NiFi.

in processors in NiFi, with a possibility of developing and plugging user-customised processors. Processors are equipped with input/output ports, which serve to connect them, and thus create complex data flow topologies.

Based on these considerations, a three-tier architecture on top of NiFi, depicted in Fig. 4, is proposed to deal with the described CPSS scenarios that involve Big Data Stream Processing at the very edge. The proposed architecture includes the hardware, middleware, and software tiers, adapting and extending them towards edge devices and associated requirements. The diagram also highlights novel aspects, which have not been part of the NiFi default stack. First, the hardware level is extended beyond the traditional concept of full-blown servers, and now also includes edge devices together with physical sensors/actuators attached to them. In the context of ISSs, these might include smart CCTV cameras, as well as any other smart devices, equipped with processing and networking capabilities. Second, the middleware level implements five additional functions – namely, *Task Partitioning*, *Node Discovery*, *Node Selection*, *Placement and Configuration*, and *Orchestration*, as well as employs *Overlay Networking* facilities with support for ad-hoc network topologies, as required by the dynamic nature of mobile devices and (wireless) networks. As far as the ISS scenario is concerned, these activities serve to establish a cluster of edge devices by discovering and selecting appropriate network devices that are then configured to run the face detection/recognition routine. These novel features are specifically required to handle dynamic edge clusters, and their roles in the proposed approach are discussed below in more details.

IV. Edge Node Configuration and Management

To support the proposed solution and modules, corresponding enabling technologies and mechanisms should be provided. Constituted by multiple mobile and portable smart devices that can move across different geophysical and network locations, the CPSS ecosystem is very dynamic in its nature, mainly due to its social component. Accordingly, the related challenges can be conceptually split into issues dealing with user subscription, AAA, and reputation – on the one hand, and *ad-hoc* networking – on the other.

A. Subscription and AAA

Asa fundamental underpinning og the proposed approach, it is necessary to provide basic subscription and AAA mechanisms to enable contributors and edge nodes to join the CPSS ecosystem. In particular, each contributor/node has to define a policy regulating the contribution, by, for example, specifying an upper bound on resource (CPU, memory, storage) utilisation, or a lower bound on the battery level, or even more complex compound metrics. Moreover, a reputation management system associated with proper incentive mechanisms has to be provided to improve node discovery and selection, rewarding contributing nodes with a positive record of completed tasks.

All such features call for an overarching solution that could be provided by the concept of *community*, widely adopted in the IoT, IoP and SIoT contexts such as *smart communities* [20]. A community-based approach could be a way of addressing the issues raised above, relying on a community management framework providing all the required services, similarly to the concept of virtual organisation in Grid Computing.

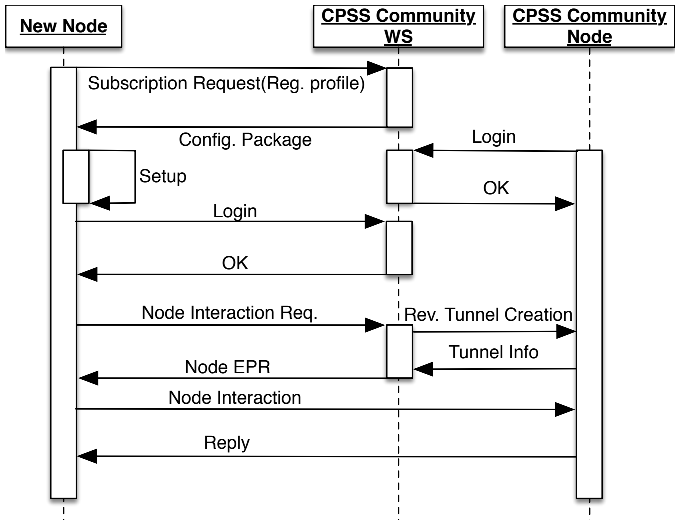


Fig. 5: Sequence diagram of subscription and authentication phases.

To establish and manage a CPSS community, a specific server hosting a Web service on a (physical or virtual) machine is required (i.e. `CPSS Community WS`, as shown in Fig. 5). It is in charge of managing new node subscription, also providing AAA facilities and advanced (overlay) networking mechanisms (which are discussed in the next section). Subscriptions can be implemented through Web forms to be filled with personal data (full name, login, password) and a contribution profile (bounds on resource utilisation, battery depletion, etc.). The full registration request is then submitted to the `CPSS Community WS` which elaborates it and sends back a configuration package including the contribution client and related settings (such as the contribution profile). This is then installed by the new node that, upon logging in the CPSS Community, can trustfully interact with other nodes (thereby becoming a `CPSS Community Node`).

It is important to remark that the community-based mechanism is not strictly necessary for the proposed approach, but if computational task offloading has to be performed with some (even minimal) security/trustworthiness requirements, it becomes mandatory. In this regard, the `CPSS Community WS` also provides other facilities that can prove to be useful during cluster node discovery and selection (in particular, for mobile devices), such as reputation management and related incentive mechanisms. Furthermore, as stated above, it also provides advanced networking functionalities in order to overcome networking issues through overlays, as discussed in the following section.

B. Networking

To establish an edge cluster it is expected that involved edge nodes are able to communicate via a network. Given the increasingly important role of mobile devices, relying on an assumption that edge clusters are based on a fixed network topology limits the application scope of the proposed approach to rather static, well-defined scenarios, where cluster nodes and their network locations are known well in advance and, therefore, are not necessarily required to be discovered and selected. On contrary, CPSS network topologies are typically not fixed, but rather continuously change with respect to mobile nodes joining and leaving the network at unpredictable rates.

As a more generic, scalable and flexible alternative, in the presented approach, dynamic run-time clusterisation is aided by overlay networking facilities. To this end, we extend the existing NiFi's built-in support for static network and cluster configurations and introduce support for *ad-hoc* topologies, in which worker nodes can be discovered, selected, added and removed dynamically at run-time in a seamless and transparent manner. Since the dynamic nature of such topologies is underpinned by wireless connectivity coupled with mobility patterns, possibly inducing the traversal of different network domains, it is important to take into account some issues that may arise as a result of these conditions. These issues may include (sudden) introduction of address/port translators or security-oriented appliances (e.g. firewalls) between any two nodes, which may immediately block or significantly modify inter-node communications, hindering the process of wireless node discovery and clusterisation.

We rely on existing work [21] for that, to provide support for (transparent) network communications among edge devices traveling across heterogeneously-administrated subnets (e.g. in a Metropolitan Area Network, or even smaller scope, such as a university campus), with the help of an (overlay) networking coordinator (i.e. `CPSS Community WS`), which gets contacted by all nodes at the system start-up to establish an always-on command-and-control stream of messages, compliant to WebSocket-based WAMP (Web Application Messaging Protocol). Available commands to be sent by the coordinator include requests for nodes to establish (reverse) tunnels to the CPSS Community server, as shown in the bottom of Fig. 5.

In this solution, WebSockets are leveraged to actually pierce 'middle boxes' and implement overlay networks among CPSS edge nodes by transporting (node-initiated) tunnels, as described in [22]. In particular, transparent Layer-3 (L3) networking is enabled by the overlay coordinator instantiating, managing and routing coordinator-terminated tunnels to all cluster nodes. Network barriers are overcome through WebSocket-based (reverse) tunneling, setting up the functional equivalent of private, isolated, secure VPN environments. This way, clustered edge devices, such as smart CCTV cameras, are enabled to discover and interact with peer nodes, as if they all were on the same physical network.

V. Clusterisation Process

The clusterisation process covers several steps to be taken by a node either to *i)* establish an edge cluster from scratch, or *ii)* join an existing cluster. In the former case, the node acts as the cluster initiator and governs the whole clusterisation process, eventually becoming the cluster coordinator. In the latter case, it interacts with the coordinator of an already existing cluster in order to join the cluster. From a behavioural perspective, this dynamics of a NiFi edge cluster adopting the proposed solution is shown in Fig. 6. It highlights the workflow of a stream processing job in an edge cluster – i.e. partitioning into tasks to be (horizontally) offloaded to worker nodes.

As discussed above, involved nodes are resource-constrained edge devices, which could be battery-powered and/or could not have enough resources to support the requested computation on their own. In the CPSS context, contribution and mobility of personal devices rise up the complexity of the problem at hand. In the described ISS scenario, the activity is initiated by a specific CCTV camera, referred to as the *Initiator* (which will later become the *Coordinator*), which starts interacting with the other nodes by sending a broadcast *discovery* offloading request through the *Node Discovery* service provided by the framework. These requests specify the main functional requirements (both hardware and software) each node has to provide to become part of the clustered computation.

Edge nodes meeting such requirements could either accept or refuse the offloading request. In the former case, available nodes could be further selected depending on other (non-functional) parameters (e.g. distance, battery life-time, potential security issues) at the *Selection* step. Next, the worflow topology is placed and configured on the selected nodes at the *Selection and Configuration* step. Once the cluster configuration is finalised, the *Processing* phase will run on the selected nodes in parallel to *Orchestration/Lifecycle Management* on the Coordinator. The latter is in charge of the *Orchestration* service, which interacts with the *Job Scheduling & Synchronisation* module of the customised NiFi framework. Further details on these stages are reported below.

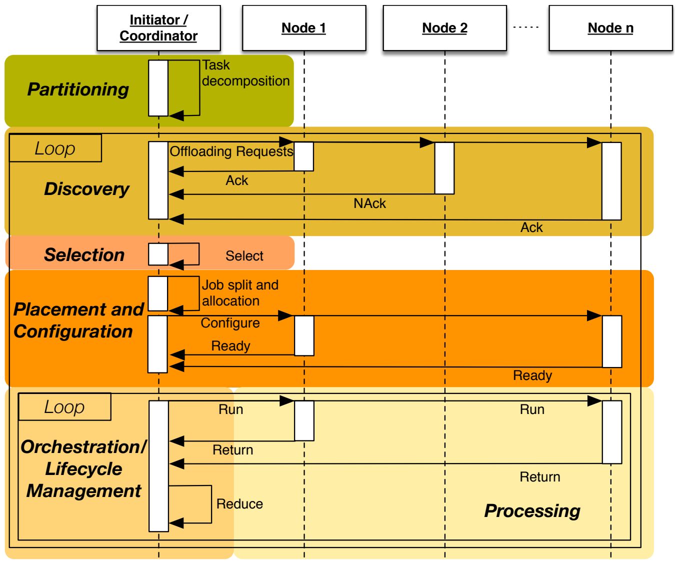


Fig. 6: Sequence diagram of the clusterisation process.

A. Task Partitioning

Usually tasks to be processed are computationally-intensive to an extent making individual edge devices not capable of accomplishing them on their own. This limitation requires edge devices to partially offload computation to peer network nodes. In the Stream Processing paradigm, such task offloading can be seen as a multi-step data processing workflow, in which each individual step is performed by a dedicated software component deployed on one of the nodes, constituting a local-area cluster. From this perspective, a computational task is seen as a pipelined sequence of atomic data processing operations. In this light, task partitioning – i.e. identifying individual steps of a more complex task, and their interconnections – becomes an important challenge. Moreover, task partitioning also serves to outline functional requirements for future cluster nodes – that is, by identifying specific operations within the workflow and understanding what resources are required to perform them, it is possible to discover and select corresponding devices with matching capabilities.

Task partitioning is acknowledged to be the most challenging functionality to be implemented in an automated manner. Closely related to the notion of software composability, it requires exhaustive descriptions of all the individual elements of a complex task, including intermediate processing steps and input/output ports. Such descriptions are expected to model both the semantic (e.g. what information is being transferred or processed, or what functionality an individual processing step implements) and the syntactic (i.e. the structure and the format of the data) aspects. Ideally, taking these self-describing building blocks, the system is able to chain complex workflows, validate information flows, as well as input data and output results in a completely automated manner. Admittedly, task partitioning is a challenging research topic in its own right, going beyond the scope of the presented research.

Therefore, in the ISS example, it is assumed to be performed manually – i.e. the system administrator is aware of the individual atomic steps constituting the use case scenario, and thus is able to design the workflow. This way, also reflecting Fig. 2, the following three tasks have been identified for the ISS workflow:

1) *CaptureVideo* (CV) continuously captures video from a camera, splitting the stream into separate frames (i.e. static images) and sending them to an output port.
2) *DetectFaces* (DF) implements feature extraction by detecting and cropping human faces in each of the received frames. Once faces have been detected, corresponding objects are serialised and transferred to the next task/processor.
3) *RecogniseFaces* (RF) is responsible for the actual recognition of faces detected at the previous stage. This component is first trained against a predefined set of human faces. Once trained, the processor is ready to perform the face recognition routine: it takes as input an image (containing a face), processes it with respect to its training set, and outputs a prediction value for each of the faces in the training set. Simply put, it decides to which extent the detected face resembles each of the faces in the training set.

B. Run-Time Node Discovery

To tap into the idle potential of ubiquitous edge devices, it is important to discover them first, thus facilitating their integration into a local-area cluster. This process should happen dynamically at run-time, since many of the edge devices are expected to be mobile (e.g. mobile phones, tablets, and other hand-held portable devices) – i.e. joining and leaving the wireless network at unpredictable rates. Furthermore, it becomes particularly challenging, as far as edge devices with sensing/actuating capabilities are concerned – i.e. as opposed to more traditional Stream Processing servers, these need to be (semantically) described to be discoverable.

Accordingly, node discovery is seen as a first step of the two-phase procedure, in which suitable cluster nodes are first discovered and then selected. In the presence of a wide range of ‘heterogeneous’ edge devices, it is not guaranteed that all of these nodes will necessarily be capable of processing the current workload for a number of reasons (e.g. missing hardware/software components, low computational capabilities, high network latency, etc.). In these circumstances, it is important to check first whether a particular node is indeed suitable for processing a given task – that is, to discover nodes and check their functional compliance for the aforementioned task. To enable such kind of analysis, it is expected that the cluster initiator, after partitioning the task and identifying corresponding requirements, will send a broadcast request to potential cluster nodes (which could be conveyed though the `CPSS Community WS` to enlarge the scope). This offloading request may include both functional and non-functional requirements, specifying, for example, available hardware resources and software components, the type of power supply (power line vs battery), the type of network connection (wired vs wireless), the distance from/to the cluster initiator, the type of device (static vs mobile), security and privacy mechanisms available on-board, etc. Next, these nodes are expected to perform basic compliance check by analysing whether they meet the incoming task requirements or not, by matching them against their self-descriptions. For example, in the considered ISS scenario, there may be a set of ten computational nodes, out of which only five are actually equipped with face detection/recognition software, and, therefore, only these five will reply to the cluster initiator, acknowledging they are functionally suitable to participate in the given ISS scenario.

Implementation-wise, in the ISS proof of concept, discovery can be implemented by means of the TCP port scanning facilities integrated into NiFi's initialisation code. As a result, the cluster initiator (i.e. the smart CCTV camera) first scans for other nodes with a specific network port open – this way, it becomes aware of other nodes running the NiFi middleware, and, therefore, potentially ready to join the cluster. To avoid situations when some other software occupies the given port, nodes discovered via the port scanner are also expected to report their unique ID, as part of the heartbeat payload. If no node ID is reported, the network device is assumed not to be running a NiFi instance, and therefore is no longer considered for clustered processing. It is important here to remark that TCP scanning, albeit a simple and standardised solution for node discovery, is limited in its effectiveness by the requirement that nodes first have to be routable and not subject to address/port translation (or any kind of filtering). Thus, a prerequisite for node discovery lies in leveraging the aforementioned overlay networking capabilities to establish a virtual network for unhindered communication among nodes, as discussed in Section IV-B.

C. Node Selection

There are many situations when functional compliance check of task requirements performed by potential cluster initiators is not enough to identify suitable nodes and establish the cluster. There is a crucial distinction to be taken into consideration, in this respect – it is important to differentiate between *suitability for a task* and *suitability for being part of the cluster*. The motivation behind this difference is that nodes have a limited view on the arrangement of a cluster – that is, they are only able to evaluate their individual capabilities to address the task requirements, but not their suitability to be engaged in the cluster. For example, a device might be equipped with sufficient hardware resources, as well as face recognition software (i.e. thus meeting the ISS task requirements). However, it might turn out that, due to its network location and configuration, network latency between the cluster initiator and this node is unacceptably high, which might become a cluster bottleneck in the future. Admittedly, the node itself is not expected to be aware of this 'external' context-related information, which becomes known only to the cluster initiator once it receives acknowledgements from nodes. In these circumstances, the cluster initiator has to govern these internal cluster dependencies, aiming to achieve an efficient and robust topology.

Accordingly, the selection of edge nodes – i.e. the second step after the node discovery – becomes an important duty of the cluster initiator that receives replies from all nodes, and, therefore, has a global view on the system, including context-related information. The cluster initiator has to further analyse and evaluate available nodes that acknowledged its offloading requests, with respect to their suitability to become part of the cluster for the given task. The selection process is supposed to be underpinned by a set of corresponding policies, which manage the selection process and provide the initiator with selection decision rules.

Referring to the target ISS implementation, after discovery, the initiating CCTV camera is aware of other network nodes, which have 'advertised' themselves (e.g. unique ID, node type, available hardware resources, network location, available software functionality, etc.) by exchanging heartbeats. The heartbeat payload can be a JSON message, which includes all relevant fields. Next, the CCTV camera, based on node selection policies and task requirements on the one hand, as well as on available nodes and resources on the other, is able to configure the cluster as required. More specifically, in the context of the presented ISS scenario, based on internal selection rules, the smart camera decides to involve mobile phones only if they will have sufficiently high reputation and reliability. Furthermore, a corresponding policy for personal and mobile devices contributing to the ISS can force their owners to inform the system before leaving and complete ongoing task processing. This way, more stable behaviour of the overall system and guaranteed job processing are expected since there would be no dropped tasks.

D. Placement and Configuration

Once all nodes and their capabilities are identified, it is time to actually deploy and configure the workflow topology on the resulting cluster, taking into account available resources. More specifically, it is important to align the software requirements with the number of cluster nodes and their computational resources. For example, having identified that there are mobile nodes present in the cluster, it makes sense to assign these nodes with less intensive tasks – i.e. taking less time to be accomplished, given that these nodes may disconnect from the cluster at any point.

From this perspective, such behaviour can be described as 'software-defined' – that is, high-level software requirements determine and modify the underlying infrastructure and network topology. To implement such kind of software-defined functionality, the proposed approach utilises NiFi's API and extends its core functionality in following three ways.

1) Creating custom prioritisers that define the order, in which jobs are delivered to processors, is the simplest, yet limited way of implementing 'software-defined' behaviour in NiFi. In addition to default prioritisers (e.g. 'First In – First Out', 'Last In – First Out', etc.), an attribute-based custom prioritiser can be defined. Based on flowfile attributes, such a custom processor is able to define which flowfile in a queue has the highest priority, and, therefore, has to be processed first. Prioritisers, however, are not expected to modify the underlying cluster configuration or workflow topology, but are rather used to manage the order of job processing.

2) The Stream Processing paradigm typically assumes that individual processors within a workflow are loosely coupled and there is no direct communication between them – that is, when a NiFi topology execution is triggered, a flowfile is transferred forward from one processing step to another via flowfile queues. This way, processors are isolated from each other, and are not 'aware' of upcoming processors (and their capabilities) that are yet to appear down the workflow pipeline. This means that dynamic run-time flowfile routing based on characteristics of upcoming processors does not seem to be feasible (or at least is not so straightforward). In these circumstances, flowfile attribute-based compliance check might be a solution. For example, to ensure that a flowfile containing a video frame will be processed by processors running on a node with sufficient software/hardware resources, it is required to put corresponding requirements as attributes on the flowfile. Thus, once received, flowfile attributes will be parsed to decide whether the current node is an appropriate candidate for processing. If not, the processor will roll back – i.e. the flow file will be first placed back on the input queue, and eventually will be transferred to a different node.[3]
3) NiFi can be accessed and managed via its RESTful interface.[4] Among other things, the API provides entry points for querying and updating the current cluster configuration by, for example, connecting/disconnecting nodes. More specifically, by checking compliance of the nodes, currently constituting the cluster, it is possible to isolate (i.e. temporarily disconnect) non-compliant ones from the cluster, and spread the workload only among the rest of the nodes, which acknowledged themselves as computation-ready, thus ensuring that only suitable cluster nodes participate in the current computation. This kind of node discovery can be implemented as a programming script or a separate custom processor – in both cases they need to be invoked before executing the main topology.[5]

Once the CCTV camera (now – the *Coordinator*) knows all nodes within the cluster, it is time to deploy the workflow topology specified by partitioning, as described in Section V-A. This functionality can be implemented by means of the previously described NiFi's RESTful API. Among other things, the API provides entry points for querying and updating the current cluster configuration by, for example, connecting/disconnecting nodes or specifying *standalone* processes (i.e. to be executed on a single node).

[3]To avoid situations, where a flowfile is being infinitely queued due to the absence of relevant processing nodes, it is also possible to implement a custom processor, which would query the list of available processing nodes in the system, and, if there are no suitable ones, remove the flowfile from the queue and persist it in a repository for later processing.

[4]NiFi's Web interface is essentially a visual user-friendly 'wrapper' for invoking RESTful commands.

[5]Please note that the RESTful API, including cluster and topology re-configuration commands, can be invoked at any point during run-time. These calls are, however, not recommended after the processing starts, as they may result in inconsistent and unstable behaviour of the overall system.

E. Orchestration/Lifecycle Management and Processing

Once the cluster has been initialised, the selected cluster nodes are ready to proceed with the processing step. The processing tasks run in parallel on worker nodes, returning results to the elected cluster coordinator (i.e. the CCTV camera), which periodically reduces and aggregates them, according to the application/business logic. In parallel to this, the coordinator keeps on periodically scanning the network for new edge nodes appearing on the network in the meanwhile, supported by the `CPSS Community WS`. Whenever a new node appears, it should expect to receive a task offloading request from the coordinator, perform the initial compliance check and reply back to the coordinator. If eventually selected by the coordinator, the node will be integrated into the running cluster and will start receiving jobs for processing. The clustered processing will continuously iterate on new tasks till completion.

This functionality is also supported by NiFi's built-in 'zoo keeping' functionality that handles node churning and synchronises topology changes across all cluster nodes – that is, whenever a new node is added (or an existing node disconnects), these changes are propagated across the whole cluster.

VI. Case Study

As discussed in Section II, the presented proof-of-concept focuses on an ISS use case scenario, where slow or congested (mobile) network connectivity prevents sending video streams for analysis to a remote processing location. This could apply, for example, to a city park or suburban area, where a CCTV smart camera is tasked with monitoring a given area to detect (and recognise faces of) potential suspects. When the region of interest is clear or just few people are within the camera field of view, the full processing workflow (face detection and recognition) can be performed on the camera itself. If the region of interest becomes crowded (i.e. exceeding 3-4 people), the camera is no longer able to process the images, and has to request support from third parties, i.e. static and mobile devices currently located nearby. Fig. 7 schematically depicts this case study.

To implement this case study, NiFi supports custom processors to be defined and added to the set of existing built-in processors. The NiFi code base was therefore extended accordingly to provide the enhanced functionality and support data processing on an edge cluster, applying the reference architecture in Fig. 4 to the face detection/recognition problem for the described ISS scenario. As a result, the schematic workflow topology depicted in Fig. 8 is placed on the cluster for execution.

To implement the processors, an established open-source face detection/recognition library JavaCV[6] has been used. Essentially, it is a collection of Java wrappers for the C++ library OpenCV[7], which contains a wide range of utility methods for handling various face detection/recognition tasks.

[6]https://github.com/bytedeco/javacv/

[7]http://opencv.org/

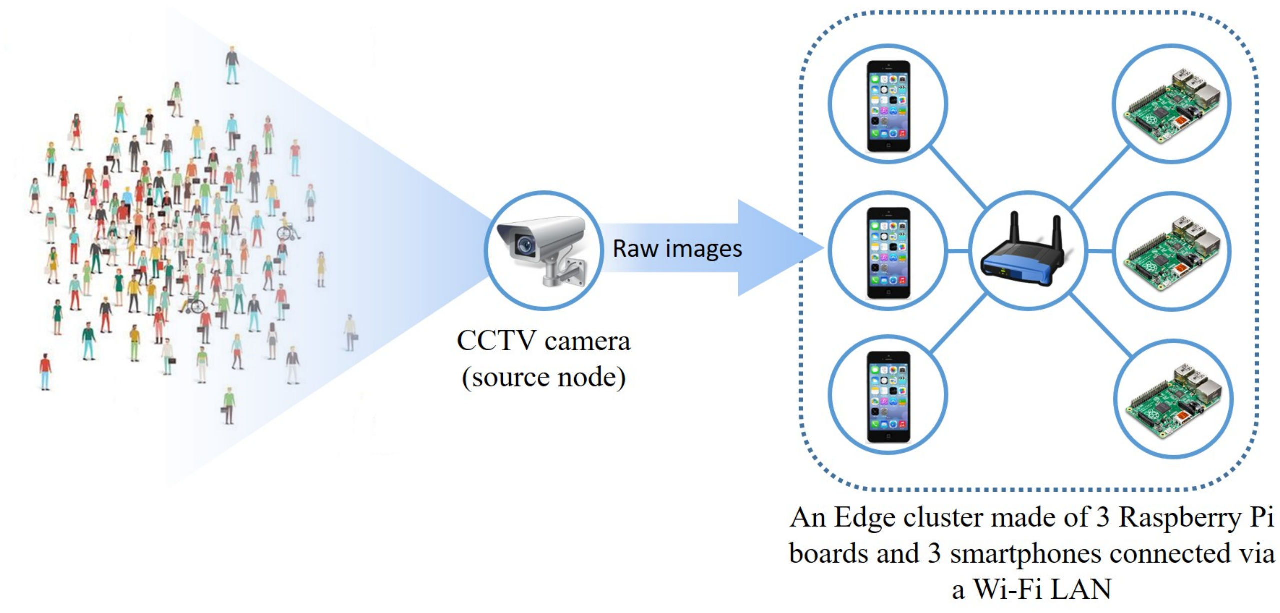

Raw images
CCTV camera
(source node)
An Edge cluster made of 3 Raspberry Pi boards and 3 smartphones connected via a Wi-Fi LAN

Fig. 7: The ISS scenario of face detection/recognition on a park or suburban area.

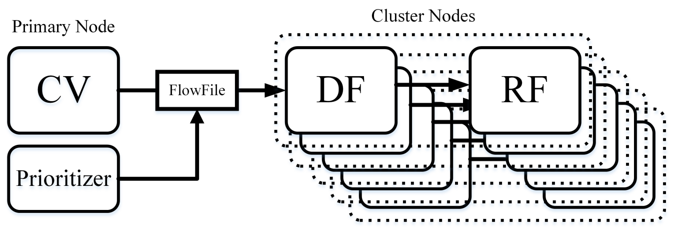


Fig. 8: The streaming workflow in the context of the ISS scenario.

A. Testbed Setup

The goal of the experiments is two-fold: *i)* to demonstrate how the performance of an edge cluster changes, as volunteering mobile devices join and leave the cluster, and *ii)* to evaluate the performance of the cluster of edge (static and mobile) devices with respect to a similar Cloud setup. To achieve the former goal, the experiments were conducted over a relatively long timeline, where mobile nodes randomly join and leave the cluster of static nodes. Performance corresponding to each time segment, as well as the average value, were measured accordingly. To address the latter goal, in addition to the setup with an edge cluster of static and mobile nodes, two Cloud setups were implemented, providing similar face recognition functionality.

1) Apache NiFi edge cluster setup: To establish the described ISS scenario using the proposed approach, the following equipment was used:

- an 8-Megapixel camera acted as the CCTV source;
- three Raspberry Pi 3 boards (ARM Cortex A53 CPU 1.2 GHZ, 1GB RAM) were the default static processing nodes;
- three Android smartphones (Google Nexus 4 – Qualcomm Snapdragon S4 Pro CPU 2.3GHz, 2GB RAM) were the selected mobile processing nodes, qualified for the given scenario.

All devices, connected through a local wireless network, run a Linux OS with the installed enhanced NiFi middleware. To emulate the Linux environment on top of Android OS on smartphones, Linux Deploy[8] was used.[9] The experiments were conducted in a public park, as discussed above.

2) Cloud setup: The same ISS scenario has been implemented using the traditional 'vertical' pattern, in which a similar NiFi architecture was deployed on a single cloud instance.[10] Such an architecture includes a Cloud-based NiFi deployment, running the face detection/recognition processors, which receive images from a CCTV camera, serialised, and transferred via a messaging queue. NiFi then extracts images from the queue, detects and recognises faces in the frames. The detection/recognition workflow is identical to the one in the edge cluster setup – i.e. the system is first trained on a set of images, and then executes the recognition routine over the incoming frames. With a few modifications, the camera

[8]https://github.com/meefik/linuxdeploy

[9]Admittedly, the performance of such a setup is somewhat lowered by the virtualised architecture. Also, for demonstration purposes we were running 'clean' versions of Android OS with minimum number of user apps and processes running at the background. In practice, however, it is expected that mobile smartphones are primarily busy with their personal jobs, and can contribute their resources to the cluster only partially. We expect this kind of aspects to be also specified in the contribution profile submitted when subscribing to the system, as described in Section IV-A.

[10]It is worth noting that we also implemented a similar clustered setup on the cloud, which composed of five interconnected NiFi-enabled cloud instances. This, however, turned out to be a redundant and inefficient solution due to the increased network latency. As it will be demonstrated below by the experiments, primary time delays in the Cloud setups were due to the network communication, rather than insufficient processing capabilities. That is, even in the presence of multiple cloud instances, task processing is undertaken only by one of them, while the rest always stay idle. For this reason, the cloud-based NiFi setups are deployed on a single instance, which is sufficient as far as processing capabilities are concerned.

performs a similar role in this setup – it captures a video stream, samples it into frames, and sends the resulting images to the Cloud-based queue service using a public broadband Internet connection.

To conduct the described experiments, Heroku[11] (VM instance Standard-1X – Intel Xeon CPU 2.5GHz, 512MB RAM) and Amazon EC2[12] (VM instance T2 Medium – Intel Xeon CPU 2.4GHz, 4GB RAM) – two well-established Cloud platforms – were chosen. As queueing facilities, Amazon Simple Queue Service[13] (SQS) and Heroku CloudAMQP[14] services were used respectively.

3) Network Configuration and Benchmarking Metrics: The network configuration of the three testbeds is summarised in Table I.

TABLE I: Testbed network configuration.

	Uplink, Mbit/s	**Downlink, Mbit/s**	**Round Trip Time, ms**
Edge Cluster	21.45	22.41	50
Heroku	2.53	14.67	88
EC2	2.73	5.18	144

It is worth noting that unlike the local NiFi setup, both Cloud setups are assumed to be dependent on the quality of the external Internet connection. The latter factor puts the following constraints on the scenario under discussion: *i)* the CCTV camera in charge of image capturing is expected to be connected to the Internet to be able to transfer captured frames; *ii)* the network itself might range from dedicated broadband connections to mobile (i.e. 3G/4G) networks; *iii)* the bandwidth of the network might also be limited in some way (e.g. either traffic-shaped by the network provider, or throttled by the client system software due to metered subscription fees).

To be able to compare different setups, it is important to agree on common benchmarks, against which each of the them has to be evaluated. Accordingly, the main benchmarking metric was *time delay* – i.e. the time difference between the moment when an image is first captured by the camera and the moment when the system accomplishes the face recognition task and returns the results to the Coordinator. This time difference includes all possible delays associated with network latency, data serialisation, and queueing. To achieve more stable and deterministic results, the experiments were conducted over several days with more than 1000 iterations in each setup. In all experiments the size of images sent over the network for face recognition is 3264×2448 pixels, which results in 1200 KB aggregate payload transferred on average. Considering the sampling frequency of 5 frames per second, this requires a bandwidth of 5.86 MB/s.

B. Experiments and Benchmarking

Figure 9a illustrates the dynamic behaviour of the edge cluster, where constantly present three static nodes are supported by randomly joining mobile devices. Each time interval on the graph corresponds to 10 minutes, and the blue time-wise graph depicts how many nodes were present in the cluster at different time intervals. The horizontal red line indicates 4.6 – an average number of nodes constituting the cluster over the overall timeline of 200 minutes.

Please note that for demonstration purposes the graph in Fig. 9a does not include the transient intermediate phases – i.e. the time periods, when nodes join and leave the cluster – which, nevertheless, need to be benchmarked for a fair overview of the viability of the presented solution. As it was explained, the current implementation of node discovery and selection is based on broadcast network scanning, which makes this process relatively fast (i.e. up to 3 seconds to scan up to 256 LAN addresses, collect acknowledgements, and reconfigure device settings accordingly). The performance drops, however, with restarting the devices – that is, after each node has overwritten its cluster settings, it is required to reboot in order for the new configuration to be applied. This process might take up to 1 minute (depending on the number of cluster nodes and deployed NiFi processors). Same applies to a situation, when a node joins an already running cluster – i.e. having received cluster configuration, it needs to update its settings, which takes up to 1 minute. This lack of support for 'hot plug' is seen as a limitation of the current version of Apache NiFi, albeit this feature is already announced to be included in one of the future releases. In any case, the clusterisation process is a one-off process that is not expected to affect the system performance in the long run.

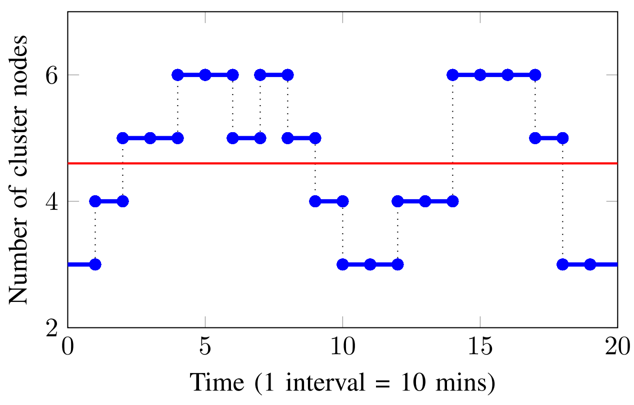
Figure 9b illustrates how the performance of the cluster is affected, as more nodes are added to the cluster. More specifically, the default cluster of three static nodes is able to process incoming images at the rate of 7.274 seconds per frame, whereas by including the other three mobile nodes, this number gets as low as 1.813 seconds per image. Accordingly, taking the considered time frame of 200 minutes and 4.6 as the average number of nodes present in the cluster, it can be assumed that the average performance of the cluster approximately equalled to 2.967 seconds per frame.

The results obtained from running the second set of experiments are summarised in Fig. 10, which depicts average values for time delay for a single job (face detection and recognition) processing on (edge cluster vs Cloud) nodes, including the 95% confidence interval, which is negligible (± 1.18 for the cluster, ± 4.26 for Heroku, and ± 4.14 for EC2) due to the high number of experiments (>1000). By looking at the histogram chart, it can be highlighted that the proposed Stream Processing architecture on top of the wireless Apache NiFi edge cluster consisting of 6 nodes performs up to 5-6 times faster than the Cloud deployments on top of either EC2 or Heroku, respectively.

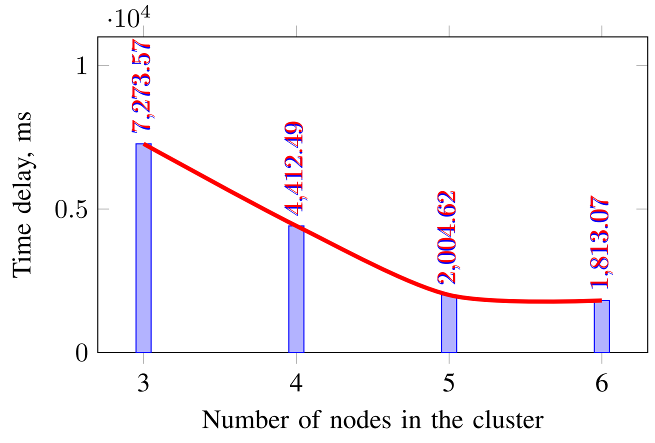
VII. Conclusions

The IoT is still primarily acting as just a source of data, only capable of pushing this data upwards to the Fog and/or the Cloud over a network in a vertical manner. However, with the new generation of users increasingly dependent on their

[11]https://www.heroku.com/
[12]https://aws.amazon.com/ec2/
[13]https://aws.amazon.com/sqs/
[14]https://elements.heroku.com/addons/cloudamqp



(a) Mobile nodes randomly join and leave the static cluster.



(b) Time delay benchmarking results.

Fig. 9: Benchmarking results for the edge cluster.

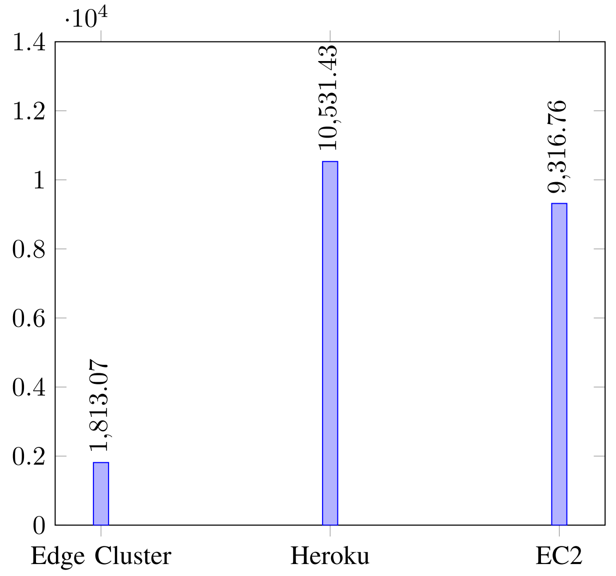


Fig. 10: Time delay in three setups, ms.

mobile and portable devices, the IoT has transformed into a complex cyber-physical-social ecosystem, in which humans (and their personal devices) are seen not only as passive data generators and IoT service consumers, but rather as active participants and contributors. More specifically, the emerging Social IoT and IoP reveal a great potential of increasingly powerful portable devices to be leveraged in the context of various data processing tasks in close proximity to actual data sources, thereby addressing network latency issues.

As a potential way of fulfilling this vision based on the principles of volunteer computing and mobile crowdsensing, this paper presented a novel approach to perform collaborative data processing at the very edge of an IoT network topology, utilising idle resources of mobile devices. As opposed to the established practice to offload computational tasks to the Cloud (through the Fog) in a 'vertical' manner, the proposed approach relies on enabling local clusters of edge devices on top of the NiFi stream processing middleware. In these circumstances, wireless edge devices, belonging to the cluster, are able to spread workload among themselves – that is, implement a 'horizontal' offloading pattern – and minimise the amount of data sent over the potentially congested link to the wide-area network. As demonstrated by the ISS proof-of-concept implementation and a number of benchmarking experiments, the proposed approach has the potential to outperform Cloud-centric setups by *i)* keeping the computation locally, close to the data source, and *ii)* involving volunteering mobile devices in clustered computation at the Edge.

References

[1] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of things (IoT): A vision, architectural elements, and future directions," *Future generation computer systems*, vol. 29, no. 7, pp. 1645–1660, 2013.

[2] G. Merlino, S. Arkoulis, S. Distefano, C. Papagianni, A. Puliafito, and S. Papavassiliou, "Mobile crowdsensing as a service: a platform for applications on top of sensing clouds," *Future Generation Computer Systems*, vol. 56, pp. 623–639, 2016.

[3] X. Masip-Bruin, E. Marín-Tordera, G. Tashakor, A. Jukan, and G.-J. Ren, "Foggy clouds and cloudy fogs: a real need for coordinated management of fog-to-cloud computing systems," *IEEE Wireless Communications*, vol. 23, no. 5, pp. 120–128, 2016.

[4] N. Choi, D. Kim, S.-J. Lee, and Y. Yi, "A fog operating system for user-oriented iot services: Challenges and research directions," *IEEE Communications Magazine*, vol. 55, no. 8, pp. 44–51, 2017.

[5] C. C. Byers, "Architectural imperatives for fog computing: Use cases, requirements, and architectural techniques for fog-enabled iot networks," *IEEE Communications Magazine*, vol. 55, no. 8, pp. 14–20, 2017.

[6] S. Yang, "IoT Stream Processing and Analytics in the Fog," *IEEE Communications Magazine*, vol. 55, no. 8, pp. 21–27, 2017.

[7] Z. Wen, R. Yang, P. Garraghan, T. Lin, J. Xu, and M. Rovatsos, "Fog orchestration for internet of things services," *IEEE Internet Computing*, vol. 21, no. 2, pp. 16–24, 2017.

[8] R. Vilalta, A. Mayoral, D. Pubill, R. Casellas, R. Martínez, J. Serra, C. Verikoukis, and R. Muñoz, "End-to-end sdn orchestration of iot services using an sdn/nfv-enabled edge node," in *Optical Fiber Communications Conference and Exhibition (OFC), 2016*. IEEE, 2016, pp. 1–3.

[9] A. Botta, W. De Donato, V. Persico, and A. Pescapé, "On the integration of cloud computing and internet of things," in *Future Internet of Things and Cloud (FiCloud), 2014 International Conference on*. IEEE, 2014, pp. 23–30.

[10] O. Skarlat, S. Schulte, M. Borkowski, and P. Leitner, "Resource provisioning for iot services in the fog," in *Service-Oriented Computing and Applications (SOCA), 2016 IEEE 9th International Conference on*. IEEE, 2016, pp. 32–39.

[11] B. I. Ismail, E. M. Goortani, M. B. Ab Karim, W. M. Tat, S. Setapa, J. Y. Luke, and O. H. Hoe, "Evaluation of docker as edge computing platform," in *Open Systems (ICOS), 2015 IEEE Confernece on*. IEEE, 2015, pp. 130–135.

[12] J. Miranda, N. Mäkitalo, J. Garcia-Alonso, J. Berrocal, T. Mikkonen, C. Canal, and J. M. Murillo, "From the internet of things to the internet of people," *IEEE Internet Computing*, vol. 19, no. 2, pp. 40–47, 2015.

[13] L. Atzori, A. Iera, G. Morabito, and M. Nitti, "The social internet of things (siot)–when social networks meet the internet of things: Concept, architecture and network characterization," *Computer networks*, vol. 56, no. 16, pp. 3594–3608, 2012.

[14] F. Y. Wang, "The emergence of intelligent enterprises: From cps to cpss," *IEEE Intelligent Systems*, vol. 25, no. 4, pp. 85–88, July 2010.

[15] J. Zeng, L. T. Yang, M. Lin, H. Ning, and J. Ma, "A survey: Cyber-physical-social systems and their system-level design methodology," *Future Generation Computer Systems*, 2016.

[16] G. Xiong, F. Zhu, X. Liu, X. Dong, W. Huang, S. Chen, and K. Zhao, "Cyber-physical-social system in intelligent transportation," *IEEE/CAA Journal of Automatica Sinica*, vol. 2, no. 3, pp. 320–333, 2015.

[17] H. Qian, X. Wu, and Y. Xu, *Intelligent surveillance systems*. Springer Science & Business Media, 2011, vol. 51.

[18] M. Valera and S. A. Velastin, "Intelligent distributed surveillance systems: a review," *IEE Proceedings-Vision, Image and Signal Processing*, vol. 152, no. 2, pp. 192–204, 2005.

[19] "The Internet of Things: Capturing the Accelerated Opportunity." [Online]. Available: http://blogs.cisco.com/digital/the-internet-of-things-capturing-the-accelerated-opportunity

[20] X. Li, R. Lu, X. Liang, X. Shen, J. Chen, and X. Lin, "Smart community: an internet of things application," *IEEE Communications Magazine*, vol. 49, no. 11, pp. 68–75, November 2011.

[21] F. Longo, D. Bruneo, S. Distefano, G. Merlino, and A. Puliafito, "Stack4Things: a sensing-and-actuation-as-a-service framework for IoT and cloud integration," *Annals of Telecommunications*, pp. 1–18, 2016.

[22] G. Merlino, D. Bruneo, F. Longo, S. Distefano, and A. Puliafito, "Cloud-Based Network Virtualization: An IoT Use Case," in *International Conference on Ad Hoc Networks*. Springer, 2015, pp. 199–210.